NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

INTERDISCIPLINARY POSTGRADUATE PROGRAM "INFORMATION TECHNOLOGIES IN MEDICINE AND BIOLOGY"

MSc THESIS

Web-Interface for querying and visualizing Alcoholic Liver Disease Patients' data from database using GraphQL

Nikolaos I. Begetis

Supervisors:

Dr. Ema Anastasiadou, Researcher-Lecturer Level, Biomedical Research Foundation of the Academy of Athens (B.R.F.A.A.)

Dr. Styliani Georgiou, Postdoctoral Level, Biomedical Research Foundation of the Academy of Athens (B.R.F.A.A.)

Pavlos Kafouris, PhD Candidate, Department of Informatics and Telecommunication – National and Kapodistrian University of Athens (N.K.U.A.)

ATHENS

JULY 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
"ΤΕΧΝΟΛΟΓΙΕΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΣΤΗΝ ΙΑΤΡΙΚΗ ΚΑΙ ΤΗ ΒΙΟΛΟΓΙΑ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διαδικτυακή διεπαφή για επερωτήματα και οπτικοποιήσεις σε δεδομένα ασθενών αλκοολικής ηπατικής νόσου με χρήση βάσης δεδομένων και βοηθητικού λογισμικού GraphQL

Νικόλαος Ι. Μπεγέτης

Επιβλέποντες: **Δρ. Έμα Αναστασιάδου,** Ερευνήτρια Δ', Ίδρυμα Ιατροβιολογικών Ερευνών Ακαδημίας Αθηνών (Ι.Ι.Β.Ε.Α.Α.)

Δρ. Στυλιανή Γεωργίου, Μεταδιδακτορική Ερευνήτρια, Ίδρυμα Ιατροβιολογικών Ερευνών Ακαδημίας Αθηνών (Ι.Ι.Β.Ε.Α.Α.)

Παύλος Καφούρης, Υποψήφιος Διδάκτορας, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθήνας (Ε.Κ.Π.Α.)

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2019

MSc THESIS

Web-Interface for querying and visualizing Alcoholic Liver Disease Patients' data from database using GraphQL

Nikolaos I. Begetis

R.N.: ΠΙΒ0111

SUPERVISORS:

Dr. Ema Anastasiadou, Researcher-Lecturer Level, Biomedical Research Foundation of the Academy of Athens (B.R.F.A.A.)

Dr. Styliani Georgiou, Postdoctoral Level, Biomedical Research Foundation of the Academy of Athens (B.R.F.A.A.)

Pavlos Kafouris, PhD Candidate, Department of Informatics and Telecommunication – National and Kapodistrian University of Athens (N.K.U.A.)

EXAMINATION COMMITTEE:

Dr. Ema Anastasiadou, Researcher-Lecturer Level, Biomedical Research Foundation of the Academy of Athens (B.R.F.A.A.)

Dr. Styliani Georgiou, Postdoctoral Level, Biomedical Research Foundation of the Academy of Athens (B.R.F.A.A.)

Dr. Eleftherios Ouzounoglou, Senior Researcher, I.C.C.S.-N.T.U.A.

July 2019

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διαδικτυακή διεπαφή για επερωτήματα και οπτικοποιήσεις σε δεδομένα ασθενών αλκοολικής ηπατικής νόσου με χρήση βάσης δεδομένων και βοηθητικού λογισμικού GraphQL

Νικόλαος Ι. Μπεγέτης

Α.Μ.: ΠΙΒ0111

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Δρ. Έμα Αναστασιάδου, Ερευνήτρια Δ', Ίδρυμα Ιατροβιολογικών Ερευνών Ακαδημίας Αθηνών (ΙΙΒΕΑΑ)

Δρ. Στυλιανή Γεωργίου, Μεταδιδακτορική Ερευνήτρια, Ίδρυμα Ιατροβιολογικών Ερευνών Ακαδημίας Αθηνών (Ι.Ι.Β.Ε.Α.Α.)

Παύλος Καφούρης, Υποψήφιος Διδάκτορας, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθήνας (Ε.Κ.Π.Α.)

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Δρ. Έμα Αναστασιάδου, Ερευνήτρια Δ', Ίδρυμα Ιατροβιολογικών Ερευνών Ακαδημίας Αθηνών (Ι.Ι.Β.Ε.Α.Α.)

Δρ. Στυλιανή Γεωργίου, Μεταδιδακτορική Ερευνήτρια, Ίδρυμα Ιατροβιολογικών Ερευνών Ακαδημίας Αθηνών (Ι.Ι.Β.Ε.Α.Α.)

Δρ. Ελευθέριος Ουζούνογλου, Έμπειρος Ερευνητής, Ε.Π.Ι.Σ.Ε.Υ.-Ε.Μ.Π.

Ιούλιος 2019

ABSTRACT

Alcoholism is one of the most serious and most common problems faced by modern societies. Approximately, 5%-10% of the population in European countries do alcohol abuse, with prolonged alcohol consumption causing liver fibrosis and cirrhosis (alcoholic liver disease, ALD). Alcoholic disease is the development of fatty liver, alcoholic hepatitis, and finally cirrhosis of the liver. The early stages of fibrosis and alcoholic hepatitis are symptomless, and when the disease is finally manifested, the clinical picture is acute. In clinical practice, the diagnosis of ALD is based on the historical alcohol ingestion, patient symptomatology and laboratory tests (e.g. liver enzymes, blood pressure, blood glucose, etc.). The dissertation aims to create a database for the collection and classification of all laboratorial, clinical, etc. examinations of patients.

Data search and graph plots and charts are created in real-time with the use of GraphQL queries and middleware query caching. The design process of the interface takes into account data changes as well as reusability of this tool in different kind of data from other tests or experiments and can be used in all types computing systems as it is containerized and responsive. This bioinformatic tool will help physicians and researchers to simplify the process of data selection, analysis and visualization by using graphs and diagrams of all data. As a result, the tool facilitates the day-to-day physicians and researchers schedule and as has the effect of letting them focus more on the essence of research, i.e. to draw conclusions about the main categories of information that lead patients to alcoholic liver disease, and less on processes.

SUBJECT AREA: Bioinformatics, Database, Web Applications, Biological Networks

KEYWORDS: web-interface, bioinformatics tool, database, alcoholic liver disease, liver steatosis

ΠΕΡΙΛΗΨΗ

Ο αλκοολισμός αποτελεί ένα από τα σοβαρότερα και συχνότερα προβλήματα που αντιμετωπίζουν οι σύγχρονες κοινωνίες. 5%-10% του πληθυσμού στις ευρωπαϊκές χώρες κάνει κατάχρηση αλκοόλ, με την παρατεταμένη κατανάλωση αλκοόλ να επιφέρει ίνωση και κίρρωση του ήπατος (αλκοολική νόσος, Alcohol Liver Disease, ALD). Η αλκοολική νόσος συνίσταται στην ανάπτυξη του λιπώδους ήπατος, στην αλκοολική ηπατίτιδα, και τελικά στην κίρρωση του ήπατος. Τα πρώτα στάδια της ίνωσης και της αλκοολικής ηπατίτιδας είναι ασυμπωματικά ενώ όταν τελικά εκδηλωθεί η νόσος, η κλινική εικόνα είναι οξεία. Στην κλινική πράξη η διάγνωση της ALD βασίζεται στο ιστορικό χρήσης αλκοόλ, στην συμπτωματολογία του ασθενούς, και σε εργαστηριακές εξετάσεις (π.χ. ηπατικά ένζυμα, αρτηριακή πίεση, γλυκόζη αίματος, κ.α.). Η διπλωματική εργασία αποσκοπεί στη δημιουργία μιας βάσης δεδομένων για την συλλογή και ταξινόμηση όλων των εργαστηριακών, κλινικών, κ.α. εξετάσεων των ασθενών.

Η αναζήτηση δεδομένων και δημιουργία γραφημάτων γίνεται σε πραγματικό χρόνο μέσω της χρήσης GraphQL επερωτήσεων. Η σχεδίαση της διεπαφής λαμβάνει υπόψη την αλλαγή των δεδομένων καθώς επίσης και την επαναχρησιμοποίηση σε διαφορετικού είδους δεδομένα από άλλα πειράματα και τη χρήση από άλλα υπολογιστικά συστήματα. Με αυτό το βιοπληροφορικό εργαλείο θα απλοποιηθεί η διαδικασία επιλογής δεδομένων, ανάλυσης και προβολής με χρήση γραφημάτων και διαγραμμάτων όλων των δεδομένων από ιατρούς και ερευνητές. Αυτό έχει ως αποτέλεσμα το εργαλείο να διευκολύνει την καθημερινότητα των ιατρών και ερευνητών ώστε να επικεντρώνονται περισσότερο στην ουσία της έρευνας, δηλαδή στην εξαγωγή συμπερασμάτων για τις βασικότερες κατηγορίες των δεδομένων που οδηγούν τους ασθενείς στην πάθηση της αλκοολικής ηπατικής νόσου, και λιγότερο στις διαδικασίες.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Βιοπληροφορική, Βάσεις Δεδομένων, Διαδικτυακές Εφαρμογές, Βιολογικά Δίκτυα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: διαδικτυακή διεπαφή, εργαλείο βιοπληροφορικής, βάση δεδομένων, αλκοολική ηπατική νόσος, ηπατική στεάτωση

To my parents and sisters

Στους γονείς μου και τις αδερφές μου

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere thanks to my supervisor, Dr. Ema Anastasiadou for assigning me this thesis which really excited me, as she understood exactly my interests and my strengths and she gave me this opportunity to contribute in a European research project. I really appreciate her advice from start to finish and her continuous participation in this thesis.

I would also like to express my sincere thanks to Dr. Styliani Georgiou, for her support and guidance, for being always present to answer my questions and for embracing and contributing in this thesis both with her theoretical and her technical knowledge. Great thanks to some of my best professors and instructors for inspiring me and disseminating me with an enthusiasm and a desire for perfection in my scientific field. These are: Dr. Ema Anastasiadou, Prof. Koutsoupias Elias, Prof. Ioannis Emiris, Prof. Dimitris Gunopulos, Prof. Sergios Theodoridis, Prof. Dimitris Achlioptas, Prof. Ioannis Ioannidis, Prof. Alex Delis, and Prof. Giannis Smaragdakis, Dr. George Spyrou, Dr. Evangelia Chrysina, Dr. George Tsangaris, Dr. Xanthou Georgina and Dr. Perantonis Stavros. I would also like to give many thanks to all my colleagues for their continuous support and help, and some special thanks to PhD Candidate Vicky Filippa and Vassilis Pierros.

Furthermore, I will always be thankful to my family for their support through all these years and for having educated me properly. Special thanks to my mother, Irene, and my father, Yannis, who are both teachers of computer science in secondary education, and whom I consider as my mentors in computer science and even more in my whole life. Also, to my younger sisters, Sophia, Eleni and Demetra, from whom I always receive endless love and encouragement and for who I would like to engrave the path to go. My heartfelt thanks to Margarita, for her unconditional support, love and encouragement. And last but not least, special thanks go to my school friends, who have taken part to my behavior shaping in all these years. The support that everyone above has given me, in all my pursuits has been tremendous. And for me, I hope to worthy and to continue my work in research, publish this thesis work to a scientific magazine and possibly, in near future, pursue a PhD.

ΕΥΧΑΡΙΣΤΙΕΣ

Πρώτα απ' όλα, θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες στην επιβλέπουσα της διπλωματικής εργασίας μου, Δρ. Έμα Αναστασιάδου, για την ανάθεση αυτής της διπλωματικής εργασίας που πραγματικά με ενθουσίασε, καθώς κατάλαβε με ακρίβεια τα ενδιαφέροντά μου και τις δυνάμεις μου και μου έδωσε αυτή την ευκαιρία να συνεισφέρω σε ένα ευρωπαϊκό ερευνητικό πρόγραμμα. Εκτιμώ πραγματικά τις συμβουλές της από την αρχή μέχρι το τέλος και τη συνεχή συμμετοχή της σε αυτή τη διπλωματική.

Θα ήθελα επίσης να εκφράσω τις ειλικρινείς ευχαριστίες μου προς την Δρ. Στυλιανή Γεωργίου για την υποστήριξη και την καθοδήγησή της, που είναι πάντα παρούσα για να απαντήσει στις απορίες μου και που εναγκάλισε και συνέβαλε σε αυτή τη διπλωματική τόσο με τις θεωρητικές όσο και με τις τεχνικές της γνώσεις. Ευχαριστώ μερικούς από τους αγαπημένους μου καθηγητές και εκπαιδευτές οι οποίοι με εμπνέουν και με γεμίζουν ενθουσιασμό και επιθυμία για να εξελίσσομαι στην επιστήμη μου. Αναφορικά αυτοί είναι οι: Δρ. Έμα Αναστασιάδου, Καθ. Ιωάννης Εμίρης, Καθ. Δημήτρης Γουνόπουλος, Καθ. Σέργιος Θεοδωρίδης, Καθ. Δημήτρης Αχλιόπτας, Καθ. Ηλίας Κουτσουπίας, Καθ. Ιωάννης Ιωαννίδης, Καθ. Αλέξης Δελής, Καθ. Γιάννης Σμαραγδάκης, Δρ. Γιώργος Σπύρου, Δρ. Ευαγγελία Χρύσινα, Δρ. Γιώργος Τσάγγαρης, Δρ. Γεωργιάνα Ξάνθου και Δρ. Σταύρος Περαντώνης. Θα ήθελα επίσης να ευχαριστήσω όλους τους συναδέλφους μου για τη συνεχή υποστήριξή τους και βοήθεια και πιο συγκεκριμένα να δώσω ειδικές ευχαριστίες προς την υποψήφια διδάκτορα Βίκυ Φίλιππα και τον Βασίλη Πιέρρο.

Επιπλέον, θα είμαι πάντα ευγνώμων στην οικογένειά μου για όλη την υποστήριξή τους όλα αυτά τα χρόνια και για τη σωστή εκπαίδευση και διαπαιδαγώγησή μου. Ευχαριστώ ιδιαιτέρως τη μητέρα μου, την Ειρήνη και τον πατέρα μου, τον Γιάννη, οι οποίοι και οι δύο είναι δάσκαλοι της επιστήμης των υπολογιστών στη δευτεροβάθμια εκπαίδευση και τους οποίους θεωρώ ως μέντορές μου στην επιστήμη των υπολογιστών και ακόμη περισσότερο σε όλη μου τη ζωή. Ευχαριστώ επίσης τις μικρότερες αδελφές μου, τη Σοφία, την Ελένη και τη Δήμητρα, από τις οποίες λαμβάνω καθημερινά την ατελείωτη αγάπη και ενθάρρυνσή τους και για τις οποίες θέλω να χαράξω το μονοπάτι για να ακολουθήσουν. Ευχαριστώ θερμά τη Μαργαρίτα για την άνευ όρων υποστήριξη, αγάπη και ενθάρρυνση. Και, τέλος, θέλω να δώσω ιδιαίτερες ευχαριστίες στους σχολικούς μου φίλους, οι οποίοι συμμετείχαν στη διαμόρφωση της συμπεριφοράς μου σε όλα αυτά τα χρόνια. Η υποστήριξη που μου έδωσαν όλοι οι παραπάνω, σε όλες τις επιδιώξεις μου, ήταν τεράστια. Και για μένα, ελπίζω να συνεχίσω το έργο μου στην έρευνα, να δημοσιεύσω αυτό το έργο της διπλωματικής εργασίας σε κάποιο επιστημονικό περιοδικό και πιθανόν, στο εγγύς μέλλον, να ακολουθήσω την καριέρα μου με ένα διδακτορικό.

CONTENTS

LIST OF PICTURES

LIST OF TABLES

LIST OF CODES

1. INTRODUCTION

Alcoholism is one of the most serious and most common problems faced by modern societies. Alcoholic disease is the development of fatty liver, alcoholic hepatitis, and finally cirrhosis of the liver. The early stages of fibrosis and alcoholic hepatitis are symptomless, and when the disease is finally manifested, the clinical picture is acute. In clinical practice, the diagnosis of ALD is based on the historical alcohol ingestion, patient symptomatology and laboratory tests. The dissertation aims to create a database for the collection and classification of all laboratorial, clinical, etc. examinations of patients. Data search and graph plots and charts are created in real-time with the use of GraphQL queries and middleware query caching. This bioinformatic tool will help physicians and researchers to simplify the process of data selection, analysis and visualization by using graphs and diagrams of all data.

In the first chapter, we will take an insight look in the liver in order to better understand the factors that lead the liver to pathological conditions. General information regarding its' importance for the human organism an the variety of reactions that it regulates. An overview of the diseases that occur in the liver is presented and a connection between alcohol consumption and liver disease. Then, alcoholic liver disease (ALD) will be presented focusing on the epidemiology, pathogenesis and lastly diagnosis. Lastly, we devote a section on the data that this thesis uses. We start with the patients that consent in sharing their personal information, and then categorise these data information and describe each one of them.

In the second chapter, we present the architectural schema of the web application we created and analyze it from a business aspect. We start by analyzing the software development lifecycle and software release lifecycle, which consist of development phases, release stages and the software versioning. This way we then refer to working models and how they are connected through software phases. Also, we analyze two main working models, Waterfall and Agile framework, emphasizing on the advantages, disadvantages and their differences. Finally, we explain the reason why we chose our working model methodology, Agile Kanban, and how we used it.

In the third chapter, we present the architectural schema of the web application we created, and this time we analyze its technological aspect. We refer to separation of concerns for each entity, and we de-structure the architectural schema in pieces and explain each one of them until we glue them back together again. Then, we present the web application's technologies stack, which we used and how we blended them together. Next, we analyze the coding design patterns that we used and how we assure that the web application is designed with main concerns the scalability, maintainability, reusability and performance. Finally, we explain the importance of visualizations and how we managed to select which visualization charts and graphs fit better to our use cases.

In the fourth chapter, we present an overview of the different functionality offered from the web application we created. In the beginning we refer to every screen of the application and the value that each one tries to offer for the application's users. Then, we continue with some example use cases, that handle main user concerns like data sharing among users, data discovery using search, data projection and predictability using visualizations, data cleansing, data filtering, data import from csv formatted file, data and filtered data extraction to csv format, data history saving, data exploration with navigation, data and filtered data visualization for comparisons, composition, understanding, distribution, trend and relations analysis purposes based on criteria, data additions, editing and deletions Finally, we make a comparison with some of the best tools nowadays for data-selection and visualization.

In the final chapter, we present the conclusions of this thesis and what can be done to expand or improve this software. We start by making an overview of all chapters and then we focus on the value that this tool gives to our users. Then, we continue with some statements about the process Kanban model that we uses by setting 2 statistics that derive from the Kanban board, and pose some possible future extensions and improvements.

2. LIVER

2.1 Introduction

In this chapter we will take an insight look in the liver in order to better understand the factors that lead the liver to pathological conditions. General information regarding its importance for the human organism and in more detail for the anatomy, macroscopically and microscopically will be presented in subsection 2.2. The three main surfaces that the liver possess; superior, inferior and posterior and what are the main factors of the liver lobe. Also, as the liver regulates a wide variety of reactions, some of them will be analyzed in this subsection. Liver glucose metabolism, liver fatty acid metabolism, metabolism of amino acids and protein synthesis are some of the functions of liver which are disturbed when alcohol is consumed for a long period of time.

The position of the liver is a site that is vulnerable to the passing of infection. In the next subsection 2.3 an overview of the diseases that occur in the liver is presented, a connection between the reasons of alcohol consumption and liver disease. Moreover, liver inflammation will be analysed as being the cause of chronic liver disease that can lead to death.

At the last subsection 2.4, alcoholic liver disease (ALD) will be presented focusing on the epidemiology, pathogenesis and lastly diagnosis.

2.2 Importance of liver

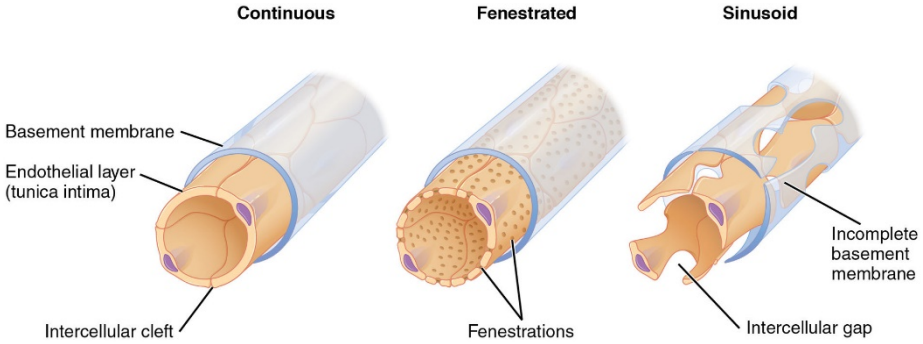
First of all, the terminology related to the liver often starts in *hepat-* from ἡπατο-, which is the Greek word for liver. The liver is an organ only found in vertebrates with the main functions of detoxifying various metabolites, synthesizing vitamins and produces biochemicals which are necessary for digestion. Basically, it filters blood coming from the digestive track, before passing it to the rest of the body. It receives 30% of the blood circulating in our system every minute and performs chemical reactions to remove harmful toxins and distribute and store essential nutrients. Also, it is the largest gland of the human body and has both external and internal secretions which are formed in the hepatic cells. Hepatocytes are the liver's highly specialized cells which regulate a wide variety of reactions. The liver counts approximately 500 functions.

The liver is a reddish-brown, wedge-shaped organ with four lobes of unequal size and shape. In humans it is located in the upper and right part of the abdominal cavity, below the diaphragm. In the male it weights from 1400 to 1600g and in the female 1200 to 1400g. It is relatively larger in the fetus than in the adult, consisting about one-eighteenth in the fetus and one thirty-sixth of the entire body weight of the adult. Its other roles in metabolism include the regulation of glycogen storage, decomposition of red blood cells and the production of hormones.

Normally, you cannot feel the liver because it is protected by the rib-cage. In order for the liver to stay in place there are several attachments that contribute. Attaching with the diaphragm by the coronary and triangular ligaments and the intervening connective tissue of the uncovered area, together with the connection of the inferior vena cava by the connective tissue and hepatic veins would hold up the posterior part of the liver. Several pressure support such as that of the abdominal visceral whose muscular walls are always in tonic contraction. Moreover, the atmospheric pressure holds the superior surface of the liver perfectly fitted under the diaphragm, holding against it. Also, the superior surface is held up by the negative pressure in the thorax.

The liver is connected to two large blood vessels: the hepatic artery and the portal vein and common hepatic duct. The hepatic artery carries oxygen-rich blood from the aorta

via the celiac plexus, whereas the portal vein carries blood rich in digested nutrients from the entire gastrointestinal tract and also from the spleen and pancreas. These blood vessels subdivide into small capillaries, which then lead to lobules. The lobules are held together by a fine, dense, irregular, fibro elastic connective tissue layer which extends from the fibrous capsule covering the entire liver known as Glisson's capsule.



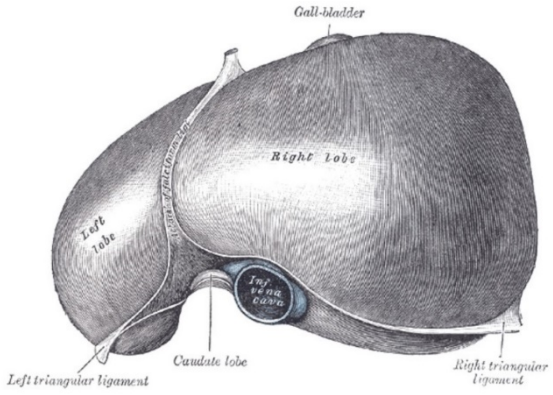
Picture 1: Capillaries lead to lobules [1]

As said before the liver is one of the most vital organs, not only it executes multiple reactions for the production of vital components, but also because its' biochemistry is so complex that no one understands all the functions it serves, or the details of how it works, making it difficult to mimic all the functions completely. Artificial livers are yet to be developed to promote long-term replacement. As a short term replacement is the howl and mid liver transplantation which are the only options for liver failure.[1]

2.2.1 Anatomy

First of all, the liver has upper and a lower surface analogous to its' position. Its upper surface is entirely diaphragmatic and its lower surface is directed to the other abdominal organs and abuts the abdominal esophagus, the stomach, the duodenum and the kidneys. For presenting the anatomy of the liver we will move from the exterior to the interior and the construction units developing the structure of the liver.

The liver is composed of three surfaces; the superior, the inferior and the posterior.



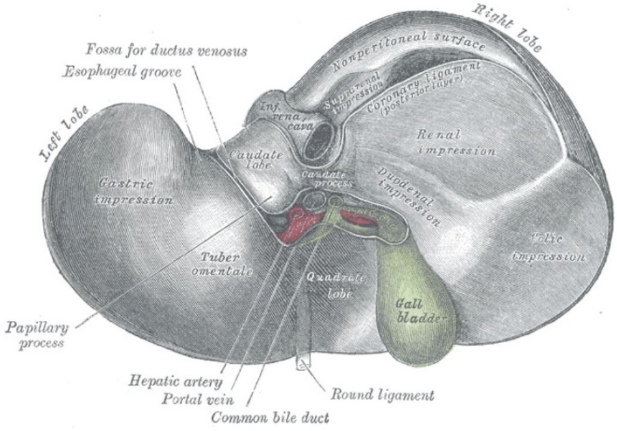
Picture 2: Superior Surface [2]

The superior surface is attached to the diaphragm and anterior abdominal wall by a triangular or falciform fold of peritoneum, the falciform ligament. The line of attachment of the falciform ligament divides the liver into two parts, termed the right and left lobes, the

[1] https://en.wikipedia.org/wiki/Liver

right being much larger. A line can be imagined running from the left of the vena cava and all the way forward to divide the liver and gallbladder into two halves. This line is called "Cantlie's line".

The **superior surface** comprises a part of both lobes, and, as a whole, is convex, and fits under the vault of the diaphragm. Its middle part lies behind the xiphoid process, and, in the angle between the diverging rib cartilage of opposite sides, is in contact with the abdominal wall. Behind this the diaphragm separates the liver from the lower part of the lungs and pleura, the heart and pericardium and the right costal arches from the seventh to the eleventh inclusive. On the diaphragmatic surface, apart from a triangular bare area where it connects to the diaphragm, the liver is covered by a thin, double-layered membrane, the peritoneum. This surface covers the convex shape of the two lobes where it accommodates the shape of the diaphragm. The peritoneum folds back on itself to form the falciform ligament and the right and left triangular ligaments. [2]



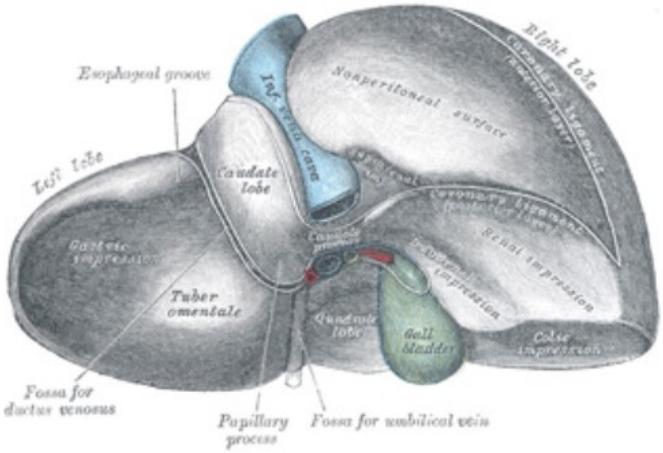
Picture 3: Inferior Surface [2]

The **inferior and posterior** surfaces are divided into four lobes by five fossa. The left limb it is known as the left sagittal fossa, and consists of two parts, viz., the fossa for the umbilical vein in front and the fossa for the ductus venosus behind. The right limb is formed in front by the fossa for the gall-bladder, and behind by the fossa for the inferior vena cava; these two fossa are separated from one another by a band of liver substance, termed the caudate process. The bar connecting the two limbs is the porta in front of it is the quadrate lobe, behind it the caudate lobe. [2]

The **inferior surface** is uneven, concave, directed downward, backward, and to the left. The surface is almost completely invested by peritoneum; the only parts devoid of this covering are where the gall-bladder is attached to the liver, and at the porta hepatis where the two layers of the lesser omen tum are separated from each other by the blood vessels and ducts of the liver. The inferior surface of the left lobe presents behind and to the left the gastric impression and to the right of this a rounded eminence. The under surface of the right lobe is divided into two unequal portions by the fossa for the gall-bladder; the portion to the left, the smaller of the two, is the quadrate lobe, and is in relation with the pyloric end of the stomach, the superior portion of the duodenum, and the transverse colon. [2]

The portion of the under surface of the right lobe to the right of the fossa for the gall-bladder presents two impressions, one situated behind the other, and separated by a ridge. The anterior of these two impressions, the colic impression, is shallow and is produced by the right colic flexure; the posterior. Medial to the renal impression is a third and slightly marked impression, lying between it and the neck of the gall-bladder. This is caused by the descending portion of the duodenum, and is known as the duodenal impression. Just in front of the inferior vena cava is a narrow strip of liver tissue,

the caudate process, which connects the right inferior angle of the caudate lobe to the under surface of the right lobe. [2]

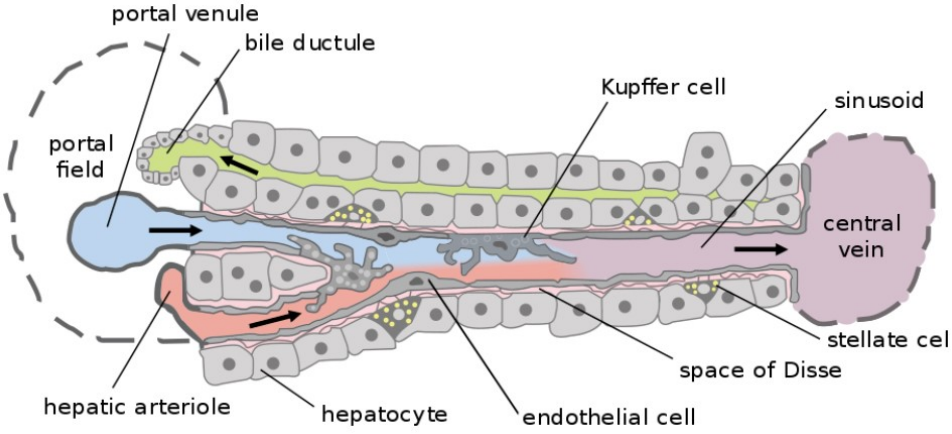


Picture 4: Posterior Surface [2]

The **posterior surface** is rounded and broad behind the right lobe, but narrow on the left. Over a large part of its extent it is not covered by peritoneum. It is marked off from the upper surface by the line of reflection of the upper layer of the coronary ligament, and from the under surface by the line of reflection of the lower layer of the coronary ligament. The central part of the posterior surface presents a deep concavity which is moulded on the vertebral column and crura of the diaphragm. To the right of this the inferior vena cava is lodged in its fossa between the uncovered area and the caudate lobe. To the left of the inferior vena cava is the caudate lobe. Its lower end projects and forms part of the posterior boundary of the porta; on the right, it is connected with the under surface of the right lobe of the liver by the caudate process, and on the left it presents the papillary process. Its posterior surface rests upon the diaphragm, being separated from it merely by the upper part of the omental bursa. To the left of the fossa for the ductus venosus is a groove in which lies the antrum cardiacum of the esophagus. [2]

2.2.2 Microscopic anatomy

The substance of the liver is composed of lobules, held together by an extremely fine areolar tissue, in which ramify the portal vein, hepatic ducts, hepatic artery, hepatic veins, lymphatics, and nerves; the whole being invested by a serous and a fibrous coat.



Picture 5: Microscopic anatomy of liver [1]

Microscopically, each liver lobe is seen to be made up of hepatic lobules. The lobules are roughly hexagonal, and consist of plates of hepatocytes radiating from a central vein. The central vein joins to the hepatic vein to carry blood out from the liver. A distinctive component of a lobule is the portal triad, which can be found running along each of the lobule's corners. Between the hepatocyte plates are liver sinusoids, which are enlarged capillaries through which blood from the hepatic portal vein and hepatic artery enters via the portal triads, then drains to the central vein.

Each lobule consists of a mass of cells, hepatic cells, arranged in irregular radiating columns between which are the blood channels (*sinusoids*). These convey the blood from the circumference to the center of the lobule, and end in the intralobular vein, which runs through its center, to open at its base into one of the sublobular veins. Between the cells are also the minute bile capillaries. Therefore, in the lobule there are all the essentials of a secreting gland; that is to say: cells, by which the secretion is formed; blood vessels, in close relation with the cells, containing the blood from which the secretion is derived; and ducts, by which the secretion, when formed, is carried away.

The liver is generally comprised of hepatocytes, which make up approximately two thirds of its total cell population (60%-70%) and non-parenchymal cells (30%-40%). The population of non-parenchymal cells includes liver sinusoidal endothelial cells (LSEC) (approximately 50%), Kupffer cells (approximately 20%), lymphocytes (approximately 25%), biliary cells (approximately 5%), hepatic stellate cells (HSCs). About 70–85% of the liver volume is occupied by parenchymal hepatocytes. Non-parenchymal cells constitute 40% of the total number of liver cells but only 6.5% of its volume. The liver sinusoids also contain sinusoidal endothelial cells, phagocytic Kupffer cells, adipocytes, "pit" cells and epithelial cells which cover the choleaggia. The endothelial cells are responsible for the wall of the colposids. The Kupffer cells are macrophage cells associated with the endothelium in the blood vessels and come to contact with almost all of the blood from the gastrointestinal (which means the possible existence of endotoxin bacteria). They make up the 1/3 of reticulocyte cells and are involved in the catabolism of red blood cells. Numerous macrophages are located in the lumen of the colposids with the task of microbial phagocytosis, T-cell stimulation and the production of substances such as prostaglandins, growth factors and peptidases. [2] [3]

[4] [5]

2.2.3 Functions

The liver is the largest reticuloendothelial metabolic organ and gland of the human body. It governs body energy metabolism and secretes valuable hormones. It works as a sensor and a controller of glycerol and glycogen as well as for many hormones like insulin, bilirubin and angiotensinogen in order to maintain the stability of our organism. Moreover, it is the exclusive organ for the biosynthesis of urea by converting ammonia, the metabolism of glycogen to glucose and the synthesis of heparin. It basically acts as a hub to metabolically connect to various tissues, including skeletal muscle and adipose tissue. Food is digested in the gastrointestinal (GI) tract, and glucose, fatty acids, and amino acids are absorbed into the bloodstream and transported to the liver through the portal vein circulation system. Liver is the place where carbohydrate, protein, amino acid and lipid metabolism happens.

Liver produces bile which is transferred by the biliary duct into the duodenum or it is either stored in the gal-bladder via the cystic duct. Also, it plays a major role in the detoxification through modifying toxic substances e.g. through methylation. It acts as a storing tank for multiple substances including glycogen, vitamin A, vitamin D, vitamin B12, vitamin K, iron and copper. The phagocyte system that exists due to the Kupffer cells is responsible for

immunological effects, as these cells act as 'sieve' for antigens. The liver is the primary site of synthesis of nearly all coagulation factors. [6]

The functions of a normal liver are presented in the following table:

Table 1: Functions of a normal liver [7] [8]

No.	Function
1	Liver glucose metabolism
2	Liver fatty acid metabolism
3	Amino acid metabolism
4	Detoxification
5	Protein metabolism and Synthesis
6	Metabolism and storing of Vitamin A
7	Metabolism of Vitamin D
8	Bile acid production and metabolism
9	Blood coagulation
10	Excretion of enzymes and bicarbonate ions
11	Drug Metabolism and Excretion-Detoxification
12	Immunological function
13	Absorbing and metabolizing bilirubin
14	Synthesis of angiotensinogen
15	Protective functions and clearance functions
16	Production of urea

2.2.3.1 Liver glucose metabolism

The liver is a pioneer in regulating carbohydrate metabolism and in particular in maintaining stable blood glucose levels, which is controlled by hormone levels such as insulin, glucagon, and under stress conditions (e.g., trauma, septicemia) of growth hormone and catecholamines). Takes about 50% of the glucose from the circulation, absorbed by the intestine and converts it to glycogen by 5% and to triglycerides by 35%.

Blood glucose enters hepatocytes via a plasma membrane glucose transporter (GLUT2). Hepatocyte-specific deletion of GLUT2, blocks hepatocyte glucose uptake. Glucose is phosphorylated by glucokinase in hepatocytes to generate glucose 6-phosphate (G6P), leading to a reduction in intracellular glucose concentrations which further increases glucose uptake. Moreover, G6P is unable to be transported by glucose transporters, so it

is retained within hepatocytes. In the fed state, G6P acts as a precursor for glycogen synthesis (glycogenogenesis). It is also metabolized to generate pyruvate through glycolysis. Pyruvate is channelled into the mitochondria and completely oxidized to generate ATP through the tricarboxylic acid (TCA) cycle and oxidative phosphorylation. Alternatively, pyruvate is used to synthesize fatty acids through lipogenesis. G6P is also metabolized via the pentose phosphate pathway to generate NADPH. NADPH is required for lipogenesis and biosynthesis of other bioactive molecules. In the fasted state, G6P is transported into the endoplasmic reticulum (ER) and dephosphorylated by glucose-6-phosphatase (G6Pase) to release glucose (glycogenolysis).

The above procedures appear to be hormone dependent. In particular, insulin is stimulated by elevated glucose levels during the meal thus promoting glycogen synthesis while glucagon is excited when plasma glucose levels tend to decrease and thus promote cleavage glycogen. The relationship between insulin and glucagon is important for hunger. Consequently, blood glucose concentration remains stable despite steep and broad changes in the rate of uptake and its consumption rate.

In individuals with liver disease there are large fluctuations in blood glucose levels. There is insulin resistance or decreased catabolism or increased insulin secretion and thus hyperglycemia is accompanied by elevated levels of insulin during fasting or post-meal times. [6] [9] [10]

2.2.3.2 Liver fatty acid metabolism

When carbohydrates are abundant, the liver not only utilizes glucose as the main metabolic fuel but also converts glucose into fatty acids. Hepatocytes also obtain fatty acids from the bloodstream, which are released from adipose tissue or absorbed from food digestion in the GI. Fatty acids are esterified with glycerol 3-phosphate to generate TAG or with cholesterol to produce cholesterol esters. TAG and cholesterol esters are either stored in lipid droplets within hepatocytes or secreted into the circulation as VLDL particles. Fatty acids are also incorporated into phospholipids, which are an essential component of cell membranes, and the surface layer of lipid droplets, VLDL, and bile particles. In the liver, the synthesis of two enzymes involved in metabolism is made of lipids, lecithin-cholesterol-acyl-transferase (LCAT), responsible for plasma esterification of cholesterol and hepatic lipase disrupts triglycerides on the surface of hepatic colposids. Also, cholesterol is synthesized by 80% of acetyl synergy A via a metabolic pathway that binds carbohydrate metabolism to fat metabolism. Cholesterol is used in many metabolic pathways, including bile acid production in the liver itself. It is primarily composed of hepatocytes because it is needed for the secretion of triglyceride-rich lipoproteins.

The triglycerides and cholesterol synthesized in the liver are taken up by the VLDL and secreted into the plasma. The liver contributes significantly to the removal of cholesterol from the plasma as the hepatocytes are the richest in LDL receptors. In addition, dietary fiber is processed into lipoproteins, which enter the blood for peripheral metabolism. During the meal, ingestion of dietary carbohydrates promotes lipid synthesis in the liver to convert carbohydrates to TG for long term storage. In the fasted state, fatty acids are oxidized mainly in the mitochondria to generate energy supply (ADP) necessary for gluconeogenesis, as well as ketone bodies.

Also, the liver synthesizes free fatty acids (FFA) when there is an increased supply of carbohydrates, but it also receives free fatty acids that have been released by triggering fatty tissue triglycerides from the effects of certain hormones such as adrenaline and glucocorticoids. [6] [11] [12] [13]

2.2.3.3 Metabolism of amino acids and proteins

The proteins are hydrolyzed to amino acids, dipeptides and are absorbed. The amino acid metabolic processes are performed in the liver. During amino acid metabolism,

ammonia is produced and due to its' toxicity it is converted into urea, which is not toxic. Through the urea cycle the liver makes sure that the excess nitrogen derived from amino acids is converted. The liver is a major protein metabolism site, with urea reproduction. Only in severe cases of liver damage the synthesis of urea decreases, resulting in an increase in ammonia. Some of the points and the symptoms of chronic liver disease are due to inadequate synthesis of these vital proteins.

After lunch, the concentration of amino acids in the portal circulation is up to 55% approximately. When the concentration limits are exceeded, the surplus is metabolised. The branched-chain amino acids, on the other hand, are transferred to muscles and fat where they are metabolised. In the fasted state, the glucose-alanine cycle is a way of removing nitrogen from amino acid catabolism in muscles during periods of increased proteolysis and is transferred to the liver where it is converted to glucose through neoglycogenesis. [6] [14] [15]

2.2.3.4 Protein synthesis

The liver is the main organ for the synthesis of plasma proteins such as albumin. Albumin contributes to plasma cholelithiasis by reducing the synthesis rate when fasting and the opposite when pathological loss of albumen is noted. The same happens due to the synthesis or transport of various substances, organic anions and cations, hormones, tryptophan, bilirubin, trace elements, drugs, fatty acids, lipoproteins, angiotensinogen, insulin-like growth factor and clotting factors such as fibrinogen, prothrombin and agents V, VII, VIII, IX, X, XI, XII, XIII, ferrous and copper-associated proteins and acute phase proteins. Other proteins synthesized in the liver are transferrin, seruloplasmin, α-1-antitrypsin, α and β globins, cyanoplasmin, ferritin, haptoglobin, lipoproteins and hemocutin.

Through the production of these proteins, liver does not only play an important role in plasma cholelithiasis but in the blood coagulation as well, in the regulation of blood pressure, in the physical development and metabolism. In liver damage, plasma proteins such as fibrinogen, haptoglobin, C-reactive protein, α-1-antitrypsin and other α and β-globins, are increased. [16] [17]

2.2.3.5 Coagulation

The liver produces many of the coagulation factors including prothrombin (II) and fibrinogen (I) as well as factors V, VI, VIII, IX, X, XI, XII, XIII. The synthesis of factors II, VII, IX, X is essential for the gastrointestinal absorption of vitamin K which is needed to produce the anticoagulant agents. In conditions where there is a major hepatocellular disaster, the plasma levels of factors II, V, VII, IX and X are reduced, and it is very rare to find low levels of fibrinogen (I) unless diffuse intravascular coagulation coexists. Insufficient production of factors coagulation induces disturbances in the mechanism of both the intrinsic and exogenous coagulation pathway. This disorder is detected in vitro by prolongation of prothrombin time and partial thromboplasty time. When this is due to a disorder of vitamin K absorption then the administration parenteral vitamin K normalizes the disorder. [18]

2.2.3.6 Detoxification

Many substances produced in the body are present in the liver conversions that often result in their inactivation, including steroid hormones (corticosteroids, estrogen, progesterone), which are linked to glucuronide, are converted into water-soluble derivatives that are easily eliminated by them kidneys, the proteins of protein metabolites as well as a variety of foreign particles. It also detoxifies many metabolites, especially nitrate elements, hormones and drugs. It is a useful protective measure, since toxic substances are absorbed from the intestine must pass through the liver before they reach the rest of the body.

Significant is the liver's contribution to inactivation and elimination of ammonia which in high concentrations is particularly toxic in contrast to urea produced which is not toxic and is easily eliminated from the kidneys. Very important is the role of the liver and metabolism of exogenous substances, in particular the cytochrome P-450 and the reduced glutathione. P-450 plays a key-role in metabolism and metabolism excretion of many drugs. Substances that stimulate P-450 are barbiturates, sedatives, antihistamines, analgesics, insecticides, polycyclic hydrocarbons and others.

Glutathione is found at high concentrations in hepatic cells primarily in its reduced form (GSH), and its biological role lies in the inactivation of oxidizing substances such as peroxide hydrogen or free radicals. Glutathione serves as substrate of many phase II coupling reactions of detoxification medicines. Most of the enzymes involved in the detoxification and excretion of drugs and other substances are found in the smooth endoplasmic reticulum of hepatocytes. These metabolic pathways are involved not only in the metabolism of drugs but also in the metabolism of endogenous substances (e.g., bilirubin and cholesterol) that are difficult to dispose otherwise from the cells. In most cases the detoxifying function of the liver involves the conversion of lipophilic substances to more hydrophilic compounds. The above procedures are known as biotransformations, allowing certain substances to be excreted directly into the urine or to be excreted via stool bile. The liver also participates indirectly in other functions such as maintaining the balance of water and electrolytes. [19]

2.2.3.7 Transport and storage of chemical compounds

The liver is an important storehouse for a number of substances including iron, copper, as well as the intake, transport, storage and activation of vitamins (mainly fatty acids A, D, E, K and vitamin B12) and trace elements (zinc, iron, magnesia), which can be increased too much in case of disease. For example, the amount of vitamin B12 stored in the liver is sufficient for about one year if no vitamin is taken up with food.

The liver is being neutralized with drugs and toxic substances through phase I and II biotransformation reactions and then biliary excretion. The conversion by bile, fat and fat-soluble vitamins into food water-soluble compounds, for the purpose of their uptake by enterocytes. At the same time, the liver has the composition and secretion of various protein carriers such as transferrin, the steroidal carrier globulin hormones, thyroid hormone carrier globulin, seruloplasmin and metallothionein. In some cases the liver can, by interaction with proteins, accumulate and store a high concentration of certain substances in a non-toxic form. An example is iron which is a key nutrient. Free iron is toxic to cells both directly and indirectly. The liver produces proteins essential for binding and for iron metabolism. Thus, the liver synthesizes and secretes into the blood the transferrin, an iron carrier protein. The transferrin is then released from iron (due to pH reduction) and is linked to the cytoplasm of the hepatocyte with ferritin, a cytoplasmic iron storage protein. Liver, also performs the synthesis and secretion of VLDL and pre-HDL lipoproteins and their removal from circulation HDL, LDL and chylomicron residues. [16]

2.2.3.8 Purification operations and protective functions

Many of the liver functions can be considered such as the detoxification of drugs and the elimination excess cholesterol in the bile after conversion to a water-soluble form. The liver participates in the removal of bacteria and antigens that pass the bowel barrier and enter the portal circulation. It contributes to the purification of blood from endogenously produced cellular comparisons on the surface of Kupffer cells where there are specific receptors that bind glycoproteins, immunoglobulin-coated molecules or complement, thereby identifying altered plasma proteins, activated factors coagulation, immune complexes, aged erythrocytes, etc. In liver:

- The ammonia is cleared through the urea cycle.

- Drugs are detoxified by microsomal oxidases and coupling systems.
- Glutathione is synthesized and excreted.
- Portal circulation is cleared from dead cells and proteins, hormones, drugs and activated clotting factors and clearance of portal circulation from bacteria and antigens. Bilirubin is a product of degradation of hemoglobin and is excreted from the liver in the bile. [20]

2.2.3.9 Regeneration

Many of the liver functions can be considered such as the detoxification of drugs and its The liver is the only visceral organ that can regenerate. It can regenerate completely, as long as a minimum of 25 percent of the tissue remains. The liver can regrow to its previous size and ability without any loss of function during the growth process. In humans, the process takes slightly longer, but regeneration can still occur in 8 to 15 days - an incredible achievement, given the size and complexity of the organ. Over the following few weeks, the new liver tissue becomes indistinguishable from the original tissue. This regeneration is organized by a number of compounds, including growth factors and cytokines.

The liver primarily processes nutrients from food, synthesizes bile, removes toxins from the body and produces proteins. Inflammation of the liver, or hepatitis, interferes with these important functions and can lead disease onset. Fortunately, the liver is extremely resilient and most cases of liver inflammation don't even come to medical attention, but in cases of the severe liver disease, there can be a serious interruption of these essential liver functions. When the liver is severely damaged, such as in liver failure, it can't continue to process nutrients from the blood. Without aggressive medical care, the absence of these essential liver functions can result in signs of serious illness like brain damage and coma. [21]

2.3 Diseases

The liver is a vital organ and supports almost every other organ in the body. Because of its strategic location and multidimensional functions, the liver is also prone to many diseases. The bare area of the liver is a site that is vulnerable to the passing of infection from the abdominal cavity to the thoracic cavity.

2.3.1 Alcohol consumption and Liver Diseases

Evidence is overwhelming about the link between excessive use of alcohol and a wide range of harmful outcomes including mortality. The conditions that lead to excessive alcohol consumption in some individuals and not in others are very complex. Alcoholism is a multigenic disorder involving interactions between genetic, psychosocial, environmental, and neurobiological factors. The pharmacological effects of ethanol that support alcohol reward and alcohol seeking behaviour involve actions of multiple receptors and neurochemical systems occurring throughout the body. Neuropharmacology provided evidence for specific neurochemical mechanisms in the brain that can lead to chronic alcohol consumption. There are many neurotransmitter systems that become deregulated during the development of alcohol dependence, including gamma (γ)- aminobutyric acid (GABA), opioid peptides, glutamate, serotonin and dopamine systems.

Further, alcohol use often exacerbates liver injury, as it coexists with other factors (e.g. viral hepatitis). According to the World Health Organization (WHO), about 2 billion people consume alcohol worldwide and upwards of 75 million are diagnosed with AUDs (Alcohol Used Disorders). Worldwide annual consumption in 2010 was 6.2 liters of alcohol per person aged 15 years or older. In Belarus, Moldova and Lithuania, annual per capita alcohol consumption was above 15 liters. Age-standardized heavy drinking was highest in European countries. In addition, the highest percentage of 15–19 years old who drink heavily was seen in Germany, the Netherlands and France. In 2012, about 3.3 million deaths (5.9% of all global deaths) were attributed to alcohol consumption. In 2012, 139 million DALYs (Disability-Adjusted Life Years), or 5.1% of the global burden of disease and injury, were attributed to alcohol consumption. Alcohol is the leading global risk factor for death and DALYs among those less than 20 years old. Globally, over 50% of mortality related to cirrhosis is attributed to alcohol misuse. [22] [23]

Table 2: Global morbidity related to chronic liver disease, 2015 [23]

	DALYs (000s)	Rank	% DALYs	DALYs per 100,000 population	Rank	YLLs (000s)	% YLLs	YLLs per 100,000 population
Global	41,486	16	1.6	565	12	40,986	2.1	558
WHO African Region	7,242	18	1.2	732	17	7,195	1.3	727
WHO Region of the Americas	4,890	17	1.8	496	10	4,826	2.7	489
WHO South-East Asia Region	15,581	13	2.2	808	10	15,450	3.0	801
WHO European Region	3,608	>20	<1.3%	n/a	12	3,502	1.7	385
WHO Eastern Mediterranean Region	3,409	17	1.4	530	11	3,371	1.8	524
WHO Western Pacific Region	6,518	19	1.3	351	11	6,407	2.0	345

*Data available from Global Health Estimates 2015 : Disease burden by Cause, Age, Sex, by Country and by Region.

Liver transplantation is the second most common solid organ transplantation, yet less than 10% of global transplantation needs are met at current rates. Though these numbers are sobering, they highlight an important opportunity to improve public health given that most causes of liver diseases are preventable

2.3.2 Liver inflammation is the key of chronic liver disease (CLD) progression

The progression of CLDs is known to depend upon the combination of different causes (i.e. Hepatitis B virus (HBV) - Hepatitis C virus (HCV) co-infection, excess alcohol consumption or NASH (non-alcoholic steato-hepatitis) associated with CHB. CLDs are not generally recognized by physicians because of the absence of symptoms and limited biochemical abnormalities. The extent of the global public health burden of CLDs, whatever the cause, is not well known and is certainly underestimated.

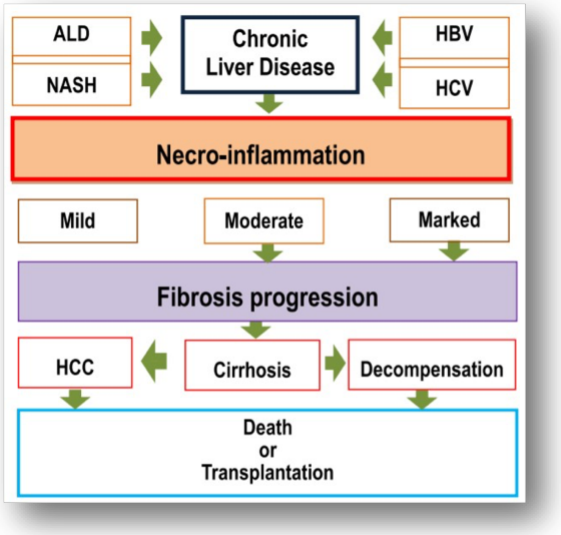
Liver inflammation can cause the necrosis of hepatocyte cells, leading to fibrosis and further deterioration to advanced-stage cirrhosis. Even after cirrhosis has developed, continued fibrogenesis causes even more deterioration to advanced-stage cirrhosis and then hepatocellular carcinoma (HCC) , causing morbidity and mortality.

However based on the following diagram fibrosis seems to be a consequence, not the cause, due to moderate and marked Necro-inflammation. Necro-inflammation is known to be independent of viral load in CHB and CHC and is not correlated with the height of alcohol consumption.

Histological scores included the grade of inflammation in addition to the stage of fibrosis, to determine the prognosis of CLD and the indication for therapy. Also, non-progressive ALD versus progressive ALD is defined by the presence of inflammation (alcoholic hepatitis). Non-progressive NAFLD is differentiated from progressive NAFLD (NASH) by the presence of inflammation. The grade of Necro-inflammation is known to be correlated with the stage of fibrosis and its prognosis in CLD. Indeed, in large histology studies performed in CHC patients, the stage of fibrosis is correlated with the grade of Necro-inflammation. The progression of CLD is driven by Necro-inflammation. This is demonstrated by the histological effect of antiviral treatments in CHB and CHC, which are

associated with the disappearance of Necro-inflammation and the regression of fibrosis, even in patients with cirrhosis. Finally, stopping inflammation stops the fibro genesis process and allows natural fibrolysis to occur. Thus, inflammation should be the target of therapy and better knowledge of the mechanisms responsible for the excessive immune response in different CLDs is needed to develop more effective therapies. [23]

Alcohol is the main risk factor of cirrhosis in Europe, where 1.8% of all deaths are attributable to liver disease. Although alcohol per se is the most important risk factor for alcoholic cirrhosis, only about 35% of heavy drinkers develop the disease and there is not a clear dose-response pattern. Moreover, even light drinkers, who consume one to two drinks a day, are at increased risk of alcoholic cirrhosis compared to abstainers. Alcohol use is therefore a bad predictor for the development of liver cirrhosis. Genetic and environmental risk factors do also not explain the substantial inter individual differences in susceptibility to ALF. [23] [24] [25]



Picture 6: Liver diseases [23]

2.4 Alcoholic liver disease (ALD)

In this chapter section we will refer generally to the alcoholic liver disease emphasizing qualitative and quantitative characteristics associated with epidemiology, pathogenesis and diagnosis.

2.4.1 Definition and Epidemiology

Alcoholic liver disease is a result of overconsuming alcohol that damages the liver, leading to a build-up of adiocytes, inflammation, and scarring. Chronic alcohol overuse causes alcoholic liver fibrosis (ALF). Currently, the pathophysiology of ALF is not completely understood due to its heterogeneous nature. Continuous exposure to alcohol in susceptible individuals induces a damage repair process that results in imbalanced tissue turnover with accumulation of extracellular matrix proteins and liver fibrosis. Alcoholic cirrhosis is considered irreversible, but its precursor, liver fibrosis, is reversible if detected early.[26] [27]

The end stage of fibrotic liver disease, cirrhosis, is the most frequent single disease entity attributed to alcohol overuse. The spectrum of alcoholic liver disease includes steatosis, steatohepatitis, alcoholic fibrosis and cirrhosis. Alcoholic hepatitis is the acute manifestation whereas ALF is the long term consequence ultimately leading to cirrhosis.

Age difference

Alcohol abuse can be found in all age groups. Drinking often starts at a relatively young age. A 2002 study found that 28.6% of high-school seniors admitted to having five or more drinks in the previous 2 weeks, an increase from 27.5% in 1993. In fact, underage drinkers consumed 19.7% of the total alcohol consumed in 2002. At the other end of the age spectrum, a study of elderly patients presenting to an emergency room found that 24% of those over 65 met criteria for alcohol abuse or problem drinking. However, most of those affected by alcohol are between the two age ranges. The age group at highest risk for hospitalizations due to alcohol related liver diseases remains those between 45 and 64, with a prevalence of 94.8 per 10,000. The prevalence rate in 25 to 44 year olds is approximately 60. In elderly hospitalized patients, the prevalence of alcohol-related diagnoses ranges from 48.2 - 54.7 per 10,000 for men and 14.8 per 10,000 for women to 65.1 per 10,000. The overall prevalence decreases with increasing age, but the prevalence always remains higher for men than for women. There is also a considerable geographic variation in alcohol-related. [28]

Males and Females

Alcohol abuse and alcoholic liver disease are found predominately in men. In an Italian study by Bellentani et al, there was a 9:1 ratio of men to women with cirrhosis in the general population. However, 13 to 33% of Americans who are either abusing alcohol or depend on it are women. In addition, alcohol consumption by women is increasing in the United States, as well as in Europe and Asia. Women throughout the world, on average, consume less alcohol than men do (43.8 g/d in men versus 15.7 g/d in women) and are less likely to be heavy users when compared with men, but the duration of drinking is similar in both sexes. Women have an increased susceptibility to the detrimental effects of alcohol.

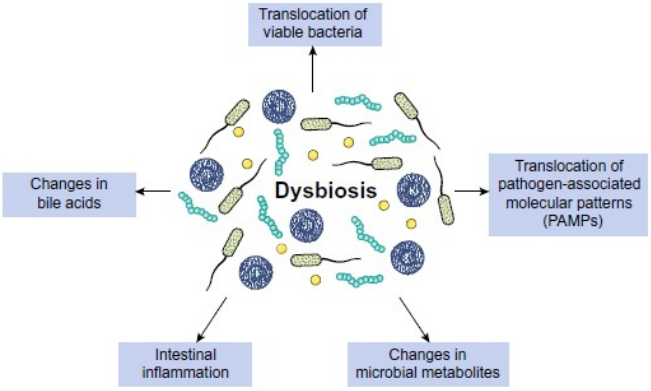
A 12-year study of 13,000 men and women found that women have a higher risk of developing cirrhosis than men do for any given level of alcohol intake. In men, the risk of cirrhosis increases with daily alcohol intake greater than 40 to 80 g/d. Women have a lower threshold and are considered to be at increased risk with daily alcohol intake greater than 20 to 60 g/d. For example, in those who have 28 to 41 drinks per week, men were found to have one third the risk of developing cirrhosis compared with women, and the risk of cirrhosis in women consuming this amount of alcohol was 16 times higher than in abstinent women.

In addition, there is a more rapid development of liver disease in women who abuse alcohol as compared with men, and the progression of liver disease in women with alcoholic hepatitis who either continue drinking or stop drinking is greater. Not all studies found an increased risk of alcoholic liver disease in women. In the population-based Dionysus study, there was no gender difference in the susceptibility to alcoholic liver disease. However, there was a non–statistically significant increase in the risk of developing alcoholic liver disease in women with alcohol intake in the 30 to 80 g/d range. Multiple mechanisms for this increased susceptibility to alcohol-related liver injury in women have been postulated. These mechanisms include differences in endotoxin levels and gut permeability to endotoxin, the effect of estrogen and androgens on endotoxin and alcohol-mediated liver injury, and differences in alcohol elimination rates in women either because of variations in first-pass metabolism or enzymatic activity or because of differences in volumes of distribution and peak blood alcohol levels. [28]

2.4.2 Pathogenesis

Anyone interested in the pathogenesis and treatment of ALD realizes that abstinence from drinking is the ultimate answer to this huge problem that includes both societal and

health issues. Research devoted to the pathogenesis of hepatotoxicity has shifted from the central role of malnutrition to direct alcohol hepatotoxicity to the final realization that no one central mechanism is operative. The place of other factors as equally important in the pathogenesis of ALD is emphasized by the clinical observation that only 15 to 20% of alcoholics end up with end-stage liver disease. It is believed that multidisciplinary efforts incorporating cell and molecular biology, immunology, and genetics will ultimately lead to a deeper understanding of the pathogenesis of ALD as well as to improved treatment modalities. [29] [30]



Picture 7: Liver Dysbiosis [89]

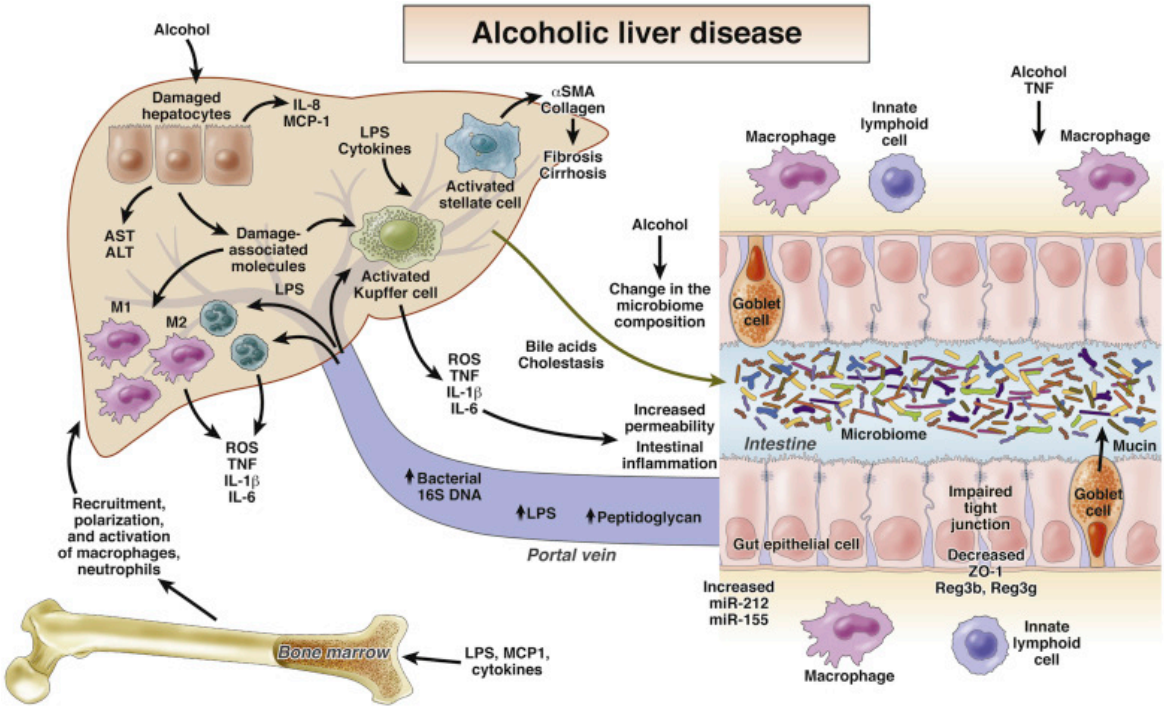
The pathogenesis of ALD can be conceptually divided into:

i. Ethanol mediated liver injury
ii. Inflammatory Immune response to injury
iii. Intestinal permeability and microbiome changes

Ethanol mediated liver injury, is related to mouse models like NIAAA where ad libitum ethanol liquid is used for producing liver pathology limited to steatosis or steatohepatitis. Also, the metabolism of ethanol is often correlated to the acetaldehyde metabolism from aldehyde dehydrogenase (ALDH) in the mitochondria. In depletion of the isozyme, accumulation of acetaldehyde, following a minor pathway which involves microsomal enzyme oxidation system and results in lipid peroxidation, mitochondrial glutathione depletion and S-adenosylmethionine depletion. The cytochrome that catalyzes the reaction is induced in chronic alcoholism, contributing to liver injury. Moreover, alcohol metabolism leads to oxidative stress and hepatocyte death. The damaged hepatocytes can release endogenous DAMPs, which activate cellular pattern recognition receptors, leading to activation of inflammation. Alcohol catbolic byproducts can regulate the transcription factors of lipid turnover resulting in accumulation triglycerides, phospholipids and cholesterol esters in hepatocyte, which characterizes steatosis. That is possible as alcohol can regulate the transcription factors of lipid metabolism. SREBP-1c is upregulated and SREBP-1c is downregulated, including AMPK, Sirtuin 1, adiponectin and STAT3. Also, alcohol inhibits fatty acid oxidation by inhibiting the transcriptional activity and downregulating PPAR-a. All these contribute to the development of steatosis.

Inflammatory Immune response to injury, is related to innate immune signalling in the early stage of ALD with simple steatosis even before the onset of inflammation. After activating interferon regulatory factor 3 (IRF3) with phosphorylases (after exposure to alcohol), inflammation starts to develop. The IRF3 is necessary for the mitochondrial apoptosis pathway and at the Kupffer cells, if in deficiency it can cause only marginal damage. Moreover, certain receptors on Kupffer cells lead to the production of pro-inflammatory cytokines (TNF-a), palmitic acid, the downregulation of proteasome functions and can cause hepatocytes and HSCs to produce chemokines for neutrophil

recruitment. Overwhelming bacterial infection can lead to multiorgan failure. T-cells increased after lipid peroxidation are heightened with inflammation. PD-1, immunoglobulin and mucin protein 3 (TIM-3) are inhibitory receptors on T lymphocytes expressed during chronic inflammation, can lead to immune exhaustion. [31] [32]



Picture 8: Liver Mechanisms after inflammation that lead to ALD [32]

Intestinal permeability and microbiome changes happen after alcoholic liver disease. Bacterial overgrowth and a lower proportion of Bacteroidacceae and probiotic bacterial such as Lactobacillus caused by intestinal dysmobility and alterations in bile acid pool. What is known as "leaky gut" is a phenomenon found in patients with chronic alcohol abuse and caused by higher level of plasma endotoxin.

Zinc deficiency is common in ALD. In animals zinc deficiency attenuates alcohol-induced liver injury and impairs the intestinal barrier, leading to endotoxin-induced cytokine production. Also, it can lead to downregulation of antioxidant enzymes like superoxide dismutase 1.

Epigenetics and microRNAs (miRNAs) control the expression of genes involved in cell growth, differentiation, and apoptosis, and are believed to be involved in the pathogenesis of liver disease, particularly cancer. Short-term alcohol exposure upregulates miRNA-212 (intestinal permeability), miRNA-217 (lipid synthesis and reduction of fatty acid oxidation). Chronic alcohol also decreases the expression of miRNA 196a and c, which are involved in early regeneration. The expression of liver miRNAs has also been shown to be significantly altered in alcohol-fed mice, but the functions of these miRNAs in the pathogenesis of ALD are not clear. [29] [33]

2.4.3 Diagnosis

In clinical practice the separation of ALD from non-alcoholic fatty liver disease is difficult as their histological alterations observed from patients are similar and the absence of liver biopsy data is based on the self-reported history alcohol consumption of the patient. In most cases the maximum level of alcohol consumption for the diagnosis of ALD is two glasses of alcohol per day for men (140g ethanol / week) and one day for women (70g of ethanol / week).

Alcoholic hepatitis is diagnosed predominantly on clinical history, physical examination, and laboratory testing, although liver biopsy is often necessary.

Experimental evidence suggests that cytokine pathways signalling cell death are critical in initiating and/or perpetuating alcohol-induced liver injury through apoptosis and necrosis. In particular, apoptosis appears to be a prominent event in both clinical and experimental alcoholic liver disease. The initial event may be mediated by the effects of alcohol on the gut. [34]

In its early stages, ALD is a silent disease and can only be detected by laboratory tests or imaging techniques. There are few programs aimed at early detection of ALD at its asymptomatic stages. Some patients with early ALD can show stigmata of alcohol abuse such as bilateral parotid gland hypertrophy, muscle wasting, malnutrition, Dupuytren's sign, and signs of peripheral neuropathy. However, some signs such as gynecomastia and extensive spider angiomas may be more frequently seen in those with alcohol as the main cause of liver disease. The diagnosis of ALD is frequently suspected upon documentation of excessive alcohol consumption (>40–50 g/day) and the presence of clinical and/or biological abnormalities suggestive of liver injury.

Laboratory blood tests such as mean corpuscular volume, gamma glutamyl transpeptidase (GGT) and aspartate amino transferase (AST) can indicate early ALD whereas advanced ALD is suspected if there is decreased albumin, increased INR, elevated bilirubin level or low platelet count. There are several laboratory markers that estimate persistent alcohol intake. Among them, carbohydrate deficient transferrin and GGT are the most frequently used markers to detect previous alcohol consumption. In patients with ALD, the AST/ALT ratio typically is greater than 1. This ratio is typically greater than 2 in AH and can also be found in patients with advanced cirrhosis regardless of the etiology.

Liver biopsy is not clearly indicated in patients with early stages of ALD or when established cirrhosis is revealed by clinical, analytical and imaging data. The liver biopsy can be done percutaneously in most patients but requires a transjugular approach in patients with a low platelet count and/or a prolonged prothrombin time. The precise indications of liver biopsy are not well established in routine practice. However, it is suggested in patients with aggressive forms of ALD such as AH requiring specific therapies (e.g., corticosteroids and/or pentoxifylline) and in patients with other cofactors suspected of contributing to liver disease. In the setting of clinical trials, the assessment of liver histology by performing a liver biopsy is recommended. The typical findings in patients with ALD include steatosis, hepatocellular damage (ballooning and/or Mallory-Denk bodies), an inflammatory infiltrate basically composed of PMN cells that predominates in the lobules, and a variable degree of fibrosis and lobular distortion that may progress to cirrhosis. For the assessment of liver fibrosis in patients with ALD, there are non-invasive methods including serum markers and liver stiffness measurements.

Most non-invasive tests have been largely validated in patients with hepatitis C, while few studies have included patients with ALD. Thus, AST to platelet ratio index (APRI), FibroTest, Fibro meter, Hepascore, and Fibro sure can be useful in patients with ALD. They are useful to distinguish between mild and severe fibrosis, but have limited utility in intermediate degrees of fibrosis. In terms of prognostic value, FibroTest (AUROC for survival = 0.79 ± 0.04), Fibro meter (0.80 ± 0.04) and Hepascore (0.78 ± 0.04) had a prognostic value equivalent to liver biopsy (0.77 ± 0.04) [54]. Transient elastography (Fibro Scan) is commonly used to assess fibrosis in patients with chronic liver disease. Fibro Scan calculates an estimate, expressed in kPa (kilopascals), for the stiffness from the measurement of wave velocity. The diagnostic threshold with the optimal diagnostic value for the detection of cirrhosis varies.

For society to be able to respond to this important challenge there is an urgent need to better understand alcoholic liver disease mechanisms and develop new tools for early diagnosis and monitoring of treatment effect, and better treatment options. [34]

2.5 Alcoholic liver disease (ALD) Data

Data are information and knowledge that can be represented as a set of values that any subject can have. These values can either be qualitative or quantitative. Sometimes, though, we use the word information or knowledge for the data that have been already analyzed or processed, or we refer to data that have not been processed, as raw data. Data is what consists any measurement, collection, report or analysis and many times it is combined with visualizations using charts or images because it is easier for the human mind to visualize, when trying to grasp what the data represents.

Raw data, is a collection of numbers or characters before it has been "cleaned" and corrected by researchers. Raw data needs to be corrected to remove outliers or obvious instrument or data entry errors. Data processing commonly occurs by stages, and the "processed data" from one stage may be considered the "raw data" of the next stage. Finally, there exist two main categories of raw data, the field data and the experimental data, and our thesis is based on experimental data. Field data is raw data that is collected in an uncontrolled "in situ" environment, while Experimental data is raw data that is generated within the context of a scientific investigation by observation and recording, using measurement tools for collecting them.

In this section we describe the data that our web-application's database is filled with, in order to visualize them efficiently for our users to extract meaningful conclusions. For collecting the data we used the survey data collection of patients with ALD who agreed to give specimens of their liver biopsies for experimentation. These surveys and liver biopsies were received at the Odense University Hospital in Denmark. Then, with respect to the user data privacy rules and having the patients given their consents, these raw data were edited in order not-to be related directly to patients according to General Data Protection Regulation(GDPR). The consents that the patients agreed in were to give their specimens of liver biopsies for lab experimentation to help in finding possible reasons or patterns that may result in suffering from ALD. [35]

The above consents allow access in patients' data only from the researchers that are participating in this research, and forbid any kind of sharing this data with legal penalties.

Picture 9: Odense University Hospital

Odense University Hospital has an immediate collaboration with the University of Southern Denmark (SDU)[2], and there has been created a research collaborative team between SDU and BRFAA[3] Dr. Anastasiadou[4] Ema's Lab[5] with the main purpose to analyze these patients data.

2.5.1 Data Categories

The data that was given for lab experimentation consisted of two types. The patients' survey data collection, and the specimens of liver biopsies that were extracted at Odense University Hospital.

This thesis purpose is to use the patient's survey data collection to find patterns in patients alcohol history, clinical, demographic, medication and comorbidity background. This data is going to fill in the database of our web-tool and provide our users possible informative visualizations. In later use, this web-tool will also be filled with experiment results of liver biopsies and possibly provide enlightening information for relations among miRNA or mRNA findings based on the liver biopsy specimens.

As mentioned before, the survey data collections consist of five main categories: demographic data, clinical data, alcohol history data, comorbidity data, and medication data. And all of these categories together consist of 479 distinct characteristics, that were all extracted and given to our lab in a MS Excel file.

Below we give a short description about each category and refer to some of the most important characteristics of each one.

Demographic Data

[2] https://www.sdu.dk/en

[3] http://www.bioacademy.gr/?lang=en

[4] http://www.bioacademy.gr/faculty-details/H8o/ema?lang=en

[5] http://www.bioacademy.gr/lab/anastasiadou?lang=en

Demographic data[6] is the data that is statistically socio-economic in nature such as population, race, income, education and employment, which represent specific geographic locations and are often associated with time. For example, when referring to population demographic data, we have characteristics such as area population, population growth or birthrate, ethnicity, density and distribution. With regard to employment, we have employment and unemployment rates, which can be related further to gender and ethnicity. In our case scenario the demographic data that were collected consist of 18 characteristics: id, gender, age in years, age with score 0-5 for calculating AP-index, smoking qualitative variables for smokers, ex-smokers and non-smokers, smoking years, smoking volume, pack years, height (cm), weight (kg), body mass index (kg/m^2), waist circumference (cm), hip circumference (cm), waist-hip ratio, living arrangements, employment status, longest education, parenthood, and income before taxes (DKK/year).

All these characteristics are very useful for our data analysis use-cases, therefore bibliography mostly looks for relations that have to do with age and gender.

Clinical Data

Clinical data[7] is a staple resource for most health and medical research. Clinical data is either collected during the course of ongoing patient care or as part of a formal clinical trial program. There exist six major types of clinical data which are: Electronic health records, administrative data, claims data, patient / disease registries, health surveys, and clinical trials data. In our case scenario the clinical data that were collected consist of 296 characteristics, that will be attached with this thesis submission. All these characteristics are very useful for our data analysis use-cases, therefore the ones that are more interesting for being used in data visualizations are the following 18: histological characterization, NAS fibrosis subscore, stage of fibrosis, NAS steatosis subscore, histological inflammation (lobular and portal), steatohepatitis appearance, alcoholic inflammation, Child-Pugh (A, B, C), Model of end-stage liver disease, FibroTest score, ActiTest score, FIB-4 index, Age-Platelet index (0-10), Forns index, Transient Elastography (kPa), 2D-SWE (kPa), and ascites appearance.

Historical Data

Historical data[8], in a broad context, is collected data about past events and circumstances pertaining to a particular subject, that are usually useful for predictive future data relations. By definition, historical data includes most data generated either manually or automatically within an enterprise, organization, government, or even individual persons. In particular, in this thesis given data, the provided characteristics that were can be classified as alcohol historical data are the following 14: patient's abstinence at inclusion, patient's ongoing alcohol overuse (>2 units/day for women, >3 units/day for men), years of excess drinking, years of alcohol abuse, number of current drinks/day, number of drinks in the week up to inclusion, average drink units/week for the last 3 months, height of alcohol consumption (drinks/day), years of abstinence, weeks since last drink (for patients who quit drinking within the last 12 months), wine consumer (yes/no), beer consumer (yes/no), spirits consumer (yes/no), fortified wine consumer (yes/no).

All these characteristics are very useful for our data analysis use-cases, and in many cases there overlap. Though, for our visualization purposes the following five

---

[6] https://www.techopedia.com/definition/30326/demographic-data

[7] http://guides.lib.uw.edu/hsl/data/findclin

[8] https://whatis.techtarget.com/definition/historical-data

characteristics cover our users' needs: years of abstinence, ongoing alcohol overuse (>2 units/day for women, >3 units/day for men), years of excess drinking, height of alcohol consumption (drinks/day) and years of abstinence.

Comorbidity Data

In medicine, comorbidity[9] is the presence of one or more additional conditions co-occurring with a primary condition. Comorbidity describes the effect of all other conditions an individual patient might have other than the primary condition of interest. The term can indicate either a condition existing simultaneously, but independently with another condition or a related medical condition. The latter sense of the term may cause some overlap with the medicine's concept of complications. In our case scenario the comorbidity relevant data that were collected consist of 22 characteristics, as follow: No comorbidity, arterial hypertension, COPD, other lung disease, hypercholesterol_mi, atrial fibrillation, heart failure / insufficiency, pacemaker and/or ICD unit, other heart disease, myotonic dystrophy(DM), diabetes type 2 without complications, diabetes type 2 with complications, chronic nephropathy, chronic pancreatitis, previous malignant disease (diverted to planocellulart carcinoma), chronic skin disease, chronic rheumatological disorder, depression, anxiety, other mental illness, chronic pain, and other chronic disorder.

All these characteristics are very useful for our data analysis use-cases and relations exploration, therefore bibliography mostly looks for relations that have to do with no comorbidity, arterial hypertension, other heart disease, myotonic dystrophy(DM), and chronic rheumatological disorder.

Medication Data

A medication[10] is a drug used to diagnose, cure, treat, or prevent disease. Drug therapy (pharmacotherapy) is an important part of the medical field and relies on the science of pharmacology for continual advancement and on pharmacy for appropriate management. In our case scenario the clinical data that were collected consist of 49 characteristics, that will be attached with this thesis submission. All these characteristics are very useful for our data analysis use-cases, therefore the one that we mostly are interested for data-visualization and diagnosis purposes is whether there were used antibiotics 3 months prior to inclusion, as such a case may affect the quality of the visualized data.

To sum up, this thesis web-application tool hopefully is going to provide our research team an ease at handling and visualizing the above data relations. Being able to easily manipulate datasets and create charts and other visualizations offers users a tremendous time-efficiency gain, which can result for researchers to truly research interesting data relations and don't have a great loss in their daily time for just trying to manipulate data themselves and as a result missing possibly a significant use case experimentation scenario due to lack of time.

[9] https://en.wikipedia.org/wiki/Comorbidity

[10] https://en.wikipedia.org/wiki/Medication

3. SOFTWARE DEVELOPMENT - BUSINESS CONCERNS

3.1 Introduction

In this chapter we analyze software development from a business point of view. We start with the software development lifecycle and software release lifecycle, which consist of development phases, release stages and the software versioning. This way we then refer to working models and how they connect with software phases. We analyze two main working models, Waterfall and Agile framework, their advantages, disadvantages and their differences and then we write about our working model methodology, Agile Kanban, and how we accomplished to use this. [35]

3.2 Software Development Life Cycle

The systems development life cycle (SDLC), also referred to as the application development life-cycle, is a term used in systems engineering, information systems and software engineering to describe a process for planning, creating, testing, and deploying an information system. The systems development lifecycle concept applies to a range of hardware and software configurations, as a system can be composed of hardware only, software only, or a combination of both.

SDLC works by lowering the cost of software development while simultaneously improving quality and shortening production time. SDLC achieves these apparently divergent goals by following a plan that removes the typical pitfalls to software development projects. That plan starts by evaluating existing systems for deficiencies. Next, it defines the requirements of the new system. It then creates the software through the stages of design, development, testing, and deployment. By anticipating costly mistakes like failing to ask the end user for suggestions, SLDC can eliminate redundant rework and after-the-fact fixes. [36] [37]

SDLC done right can allow the highest level of management control and documentation. Several pitfalls can turn an SDLC implementation into more of a roadblock to development than a tool that helps us. Failure to take into account the needs of customers and all users and stakeholders can result in a poor understanding of the system requirements at the outset. The benefits of SDLC only exist if the plan is followed faithfully.

The product life cycle describes the process for building information systems in a very deliberate, structured and methodical way, reiterating each stage of the product's life. The systems development life cycle, according to Elliott & Strachan & Radford (2004), "originated in the 1960s, to develop large scale functional business systems in an age of large scale business conglomerates. Information systems activities revolved around heavy data processing and number crunching routines". [11]

Several systems development frameworks have been partly based on SDLC, such as the structured systems analysis and design method (SSADM) produced for the UK government Office of Government Commerce in the 1980s. Ever since, according to Elliott (2004), "the traditional life cycle approaches to systems development have been increasingly replaced with alternative approaches and frameworks, which attempted to overcome some of the inherent deficiencies of the traditional SDLC".

The process of software development is a never-ending cycle. The first release of a software application is rarely completed. There are almost always additional features and

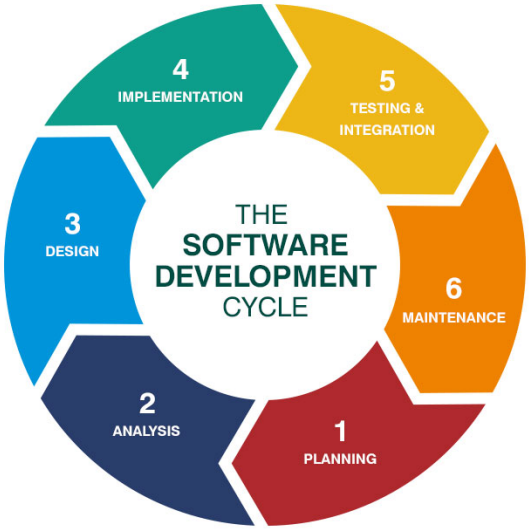
[11] https://en.wikipedia.org/wiki/Systems_development_life_cycle

bug fixes waiting to be designed, developed and deployed. Reports from error monitoring software about usability and bugs feed back into the process of software development, and become new feature requests and improvements to existing features. This is why the Software Development *Life Cycle* is the most general term for software development methods. The steps of the process and their order vary by method. Regardless of method, they typically run in cycles, starting over with each iteration.

The SDLC is not a methodology, but rather a description of the phases in the life cycle of a software application. All software development methodologies follow the SDLC phases but the method of doing that varies vastly between methodologies. These methodologies are obviously quite different approaches, yet they both contain the SDLC phases in which a requirement is born, then travels through the life cycle phases ending in the final phase of maintenance and support, after-which the whole life cycle starts again for a subsequent version of the software application.

In the following diagram the steps of the SDLC are presented. These steps are roughly the same from one methodology to another. They tend to occur in this order, though they can also be mixed together, such that several steps occur in parallel. [38]

[12]



Picture 10: The Software Development Cycle [38]

3.2.1 Planning

The most important parts of software development, requirement gathering or requirement analysis are usually done by the most skilled and experienced software engineers in the organization. After the requirements are gathered from the client, a scope document is created in which the scope of the project is determined and documented.

Planning, always begins with a preliminary analysis, proposing alternative solutions, describing costs and benefits, and submitting a preliminary plan with recommendations. Discover the organization's objectives and the nature and scope of the problem under study. Even if a problem refers only to a small segment of the organization itself, find out what the objectives of the organization itself are. Then see how the problem being studied fits in with them.

First the IT system proposal is investigated. During this step, consider all current priorities that would be affected and how they should be handled. Before any system planning is done, a feasibility study should be conducted to determine if creating a new or improved

[12] https://www.drasecurity.com/secure-design-review.html

system is a viable solution. This will help to determine the costs, benefits, resource requirements, and specific user needs required for completion. The development process can only continue once management approves of the recommendations from the feasibility study. [38]

The following represent different components of the feasibility study:

- Operational feasibility
- Economic feasibility
- Technical feasibility
- Human factors feasibility
- Legal/Political feasibility

3.2.2 Analysis

During the analysis step project goals are defined and expressed into functions and operations of the intended application. This involves the process of gathering and interpreting facts, diagnosing problems, and recommending improvements to the system. Project goals will be further aided by analysis of end-user information needs and the removal of any inconsistencies and incompleteness in these requirements.

A series of steps followed by the developer include:

1. Obtain end user requirements through documentation, client interviews, observation, and questionnaires.
2. Identify pros and cons of the current system in-place, so as to carry forward the pros and avoid the cons in the new system.
3. Find solutions to the shortcomings described in step two and prepare the specifications using any specific user proposals.

The goal of analysis is to determine where the problem is, in an attempt to fix the system. This step involves breaking down the system in different pieces to analyze the situation, analyzing project goals, breaking down what needs to be created, and attempting to engage users so that definite requirements can be defined. [36]

3.2.3 Design

At this step desired features and operations are described in detail, including screen layouts, business rules, process diagrams, pseudocode, and other documentation. screen layouts, business rules, process diagrams, and other documentation. The output of this stage will describe the new system as a collection of modules or subsystems.

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts.

Design elements describe the desired system features in detail, and they generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo-code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the system in sufficient detail, such that skilled developers and engineers may develop and deliver the system with minimal additional input design. [36] [37] [38]

3.2.4 Development

This phase produces the software under development. Depending on the methodology, this phase may be conducted in time-boxed "sprint" or may proceed as a single block of effort. Development teams should produce working software as quickly as possible. Detailed designs that brings initial design work into a completed with form of

specifications. Also, it includes the specification of interfaces between the system and its intended environment and a comprehensive evaluation of the systems logistical, maintenance and support requirements. The detail design and development is responsible for producing the product, process and material specifications and may result in substantial changes to the development specification.

Key steps within the development stage include:

- Development of engineering and prototype models
- Revision of development specification
- Product, process and material specification
- Critical design review

Business stakeholders should be engaged regularly to ensure that their expectations are being met. The output of this phase is testable, functional software. [38]

3.2.5 Integration and Testing

This is the process of finding defects or bugs in the created software. All the pieces are brought together into a special testing environment, then checked for errors, bugs, and interoperability. The code is tested at various levels in software testing. Unit, system, and user acceptance testings are often performed.

3.2.6 Documentation

Every step in the project is documented for future reference and for the improvement of the software in the development process. The design documentation may include writing the application programming interface (API).

3.2.7 Training

Once a system has been stabilized through adequate testing, the SDLC ensures that proper training on the system is performed before transitioning the system to its support staff and end users. Training usually covers operational training for those people who will be responsible for supporting the system as well as training for those end users who will be using the system after its delivery to a production operating environment.

After training has been successfully completed, systems engineers and developers transition the system to its final production environment, where it is intended to be used by its end users and supported by its support and operations staff.

3.2.8 Deployment

The software is deployed after it has been approved for release. This stage is ideally a high automated phase. In high-maturity enterprise, the software is deployed the instant it is ready. Although in smaller enterprises, the process involves manual approvals. Application Release Automation (ARA) tools are used in medium and large-size enterprises to automate the deployment of applications to Production environments. The output of this phase is the release to Production of working software. Moreover, the deployment of the system includes changes and enhancements before release. [38]

3.2.8.1 Cloud Deployment

Cloud deployment refers to the enablement of SaaS (Software as a Service), PaaS (Platform as a Service) or IaaS (Infrastructure as a Service) solutions that can be accessed on demand by end users. A cloud deployment model refers to the type of cloud computing architecture a cloud solution will be implemented on. Also, cloud deployment includes all of the required installation and configuration steps that must be implemented before used provisioning can occur. SaaS deployment is a type of cloud deployment that is typically initiated using a public cloud or a private cloud deployment model, however it may also be initiated using a hybrid cloud deployment model when hybrid cloud resources are owned and/or managed by the same entity. What is more, virtual private clouds can

be used for SaaS deployment as well. Virtual private clouds are technically public clouds that function the same as private clouds, since only trusted entities may gain access to the virtual private cloud resources.
After cloud deployment has been completed for a SaaS, PaaS or IaaS solution, user provisioning can occur based on user permissions, where access is provided for cloud resources based on the consumer's classification as either a trusted or untrusted entity. Trusted entities may receive access permission to managed cloud, private cloud or hybrid cloud resources. Untrusted entities may receive access permission to public cloud, managed cloud or hybrid cloud resources. [39] [40]

Docker Cloud

Docker in cloud computing is a technology, which has undoubtedly taken an evolutionary step towards the management of deployment platforms. It is an extremely new way of working in premises where work management becomes easier for industries. No extra finance for the storage server and cloud infrastructure maintenance is required in Docker cloud. The Docker is an open-source environment of product containers. These containers help applications to work while it is being shifted from one platform to another like – migration from developer's laptop to staging to the production. This is a new era technology, which enables enterprises to ship, build, and run any product from any geolocation.

It permits applications to be bundled and copied where all apps are dependent on each other. Cloud users find this concept useful when it comes to working with a scalable infrastructure. When docker gets integrated with cloud, it is named as Docker Cloud.

Docker Cloud is an official online service to deliver Docker products. Several online services like Azure, AWS, Google cloud platform, etc., are present for enterprises in today's date. Although these services provide flexibility in work, they require configurations of everything. On the other hand, Docker Cloud is found as an advance managed cloud system, where it could render orchestration and develop different options for its clients. This new concept prevent customers from wasting their time in several kinds of configuration processes and enables them to work more on their business growth. [41] [42] [43]

3.2.9 Support and Maintenance

During the maintenance stage of the SDLC, the system is assessed/evaluated to ensure it does not become obsolete. This is also where changes are made to initial software. Software maintenance is done for future reference. The operations and maintenance phase is the "end of the beginning". The Software Development Life Cycle does not end here. Software must be monitored constantly to ensure proper operation. Bugs and defects discovered in Production must be reported and responded. [38]

Generally, eeffectiveness and efficiency of the system must be continuously evaluated to determine when the product has met its maximum effective lifecycle. Considerations include: Continued existence of operational need, matching between operational requirements and system performance, feasibility of system phase-out versus maintenance, and availability of alternative systems.

3.3 Software Release Life Cycle

3.3.1 Stages [13] [14]

Picture 11: Stages of Software Development Life Cycle [1]

3.3.1.1 Alpha

This is the first phase to begin software testing. In this phase, developers generally test the software using white-box techniques and perform additional validation using black-box or gray-box techniques, by another testing team. Moving to black-box testing inside the organization is known as *alpha release*.

Alpha software can be unstable and could cause crashes or data loss. In general, external availability of alpha software is uncommon in proprietary software, while open source software often has publicly available alpha versions. The alpha phase usually ends with a feature freeze, indicating that no more features will be added to the software.

3.3.1.2 Beta

This is the software development phase following alpha. Software in the beta stage is also known as *betaware*. Beta phase generally begins when the software is feature complete but likely to contain a number of known or unknown bugs. Software in the beta phase will generally have many more bugs in it than completed software, speed or performance issues, and may still cause crashes or data loss. The focus of beta testing is reducing impacts to users, often incorporating usability testing.

The process of delivering a beta version to the users is called *beta release* and this is typically the first time that the software is available outside of the organization that developed it. Software beta releases can either be public or private, depending on whether they are openly available or only available to a limited audience. Beta version software is often useful for demonstrations and previews within an organization and to prospective customers.

Beta testers are people who actively report issues of beta software. They are usually customers or representatives of prospective customers of the organization that develops the software

[13] https://wpsitesync.com/release-candidate-vs-general-release/

[14] https://en.wikipedia.org/wiki/Software_release_life_cycle

3.3.1.3 Release candidate

It is also known as "going silver", is a beta version with potential to be a final product, which is ready to release unless significant bugs emerge. In this stage of product stabilization, all product features have been designed, coded and tested through one or more beta cycles with no known showstopper-class bugs.

A release is called code complete when the development team agrees that no entirely new source code will be added to this release. There could still be source code changes to fix defects, changes to documentation and data files, and peripheral code for test cases or utilities.

3.3.1.4 Software upgrade versioning[15] [16]

Software upgrade versioning is the process of assigning either unique *version names* or unique *version numbers* to unique states of computer software. Within a given version number category (major, minor), these numbers are generally assigned in increasing order and correspond to new developments in the software. At a fine-grained level, revision control is often used for keeping track of incrementally different versions of information, whether or not this information is computer software.

Modern computer software is often tracked using two different software versioning schemes—internal version number that may be incremented many times in a single day, such as a revision control number, and a *released version* that typically changes far less often, such as *semantic versioning* or a project code name.

Picture 12: Software upgrade versioning [1]

3.4 Software development models

The software development models are the various processes or methodologies that are being selected for the development of the project, depending on the project's aims and goals. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out.

There are various Software development models or methodologies.

1. Waterfall model
2. V model
3. Incremental model

---

[15] https://en.wikipedia.org/wiki/Software_versioning

[16] https://blog.soenneker.com/a-new-increment-for-software-versioning-9fc2a9068c23

4. RAD model
5. Agile model
6. Iterative model
7. Spiral model
8. Prototype model

Subsequently, the Waterfall and Agile software development models will be explained.

3.4.1 Waterfall Model

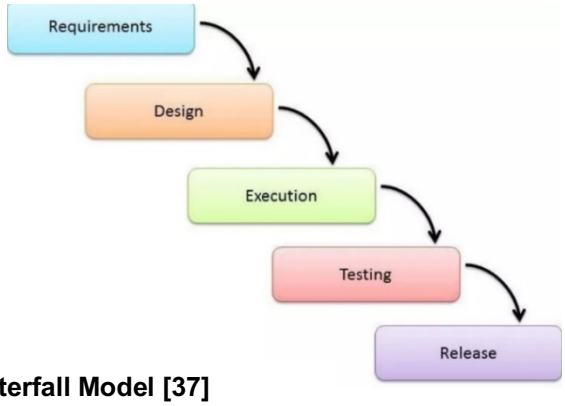
The Waterfall Model is the oldest model created. The first formal description of the method is often cited as an article published by Winston W. Royce, in 1970. Royce presented this model as an example of a flawed, non-working model.

The waterfall model was an early attempt to provide structure, metrics and control to the development of large and complex application systems, usually mainframe systems.[17]

In Royce's original waterfall model, the following phases are followed in order:

1. System and software requirements: captured in a product requirements document
2. Analysis: resulting in models, schema, and business rules
3. Design: resulting in the software architecture
4. Coding: the development, proving, and integration of software
5. Testing: the systematic discovery and debugging of defects
6. Operations: the installation, migration, support, and maintenance of complete systems

Thus the waterfall model maintains that one should move to a phase only when its preceding phase is reviewed and verified.[18] [37]



Picture 13: Waterfall Model [37]

Various modified waterfall models (including Royce's final model), however, can include slight or major variations on this process. In the olden days, Waterfall model was used to develop enterprise applications like Customer Relationship Management (CRM) systems, Human Resource Management Systems (HRMS), Supply Chain Management Systems, Inventory Management Systems, Point of Sales (POS) systems for Retail chains etc. Waterfall model was used significantly in the development of software till the year 2000.

This "inflexibility" in a pure waterfall model has been a source of criticism by supporters of other more "flexible" models. It has been widely blamed for several large-scale government projects running over budget, over time and sometimes failing to deliver on

[17] https://en.wikipedia.org/wiki/Waterfall_model

[18] https://melsatar.blog/2018/02/16/the-waterfall-model-a-different-perspective/

requirements due to the Big Design Up Front approach. Except when contractually required, the waterfall model has been largely superseded by more flexible and versatile methodologies developed specifically for software development. [37]

Table 3: Advantages and disadvantages of the Waterfall model

Advantages	Disadvantages
Easy to explain to the users	Assumes that the requirements of a system can be frozen
Structures approach	Very difficult to go back to any stage after it finished
Stages and activities are well defined	A little flexibility and adjusting scope is difficult and expensive
Helps to plan and schedule the project	Costly and required more time, in addition to the detailed plan
Verification at each stage ensures early detection of errors/misunderstanding	
Each phase has specific deliverables	

3.4.2 Agile Model

In 2001, a group of software developers met to discuss some lightweight development methods that emerged from the late 1970s: Kent Beck, Ward Cunningham, Dave Thomas, Jeff Sutherland, Ken Schwaber, Jim Highsmith, Alistair Cockburn, Robert C. Martin, Mike Beedle, Arie van Bennekum, Martin Fowler, James Grenning, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, and Steve Mellor. Together they published the *Manifesto for Agile Software Development*.

Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s). It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change.[19]

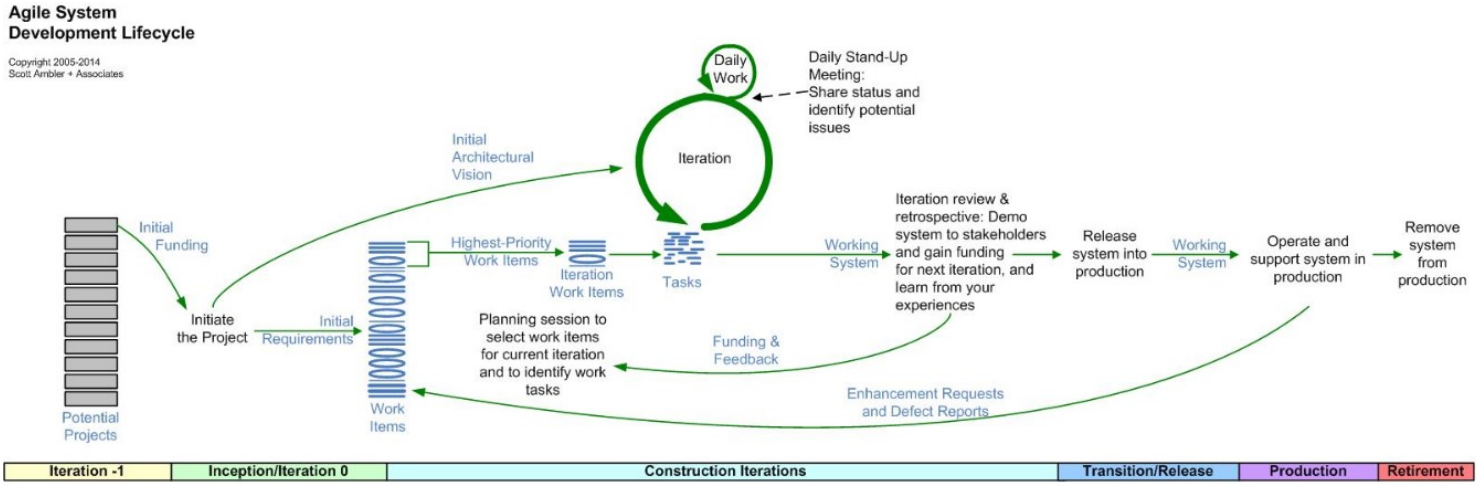
Advantages of Agile model:

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

[19] https://en.wikipedia.org/wiki/Agile_software_development

Disadvantages of Agile model:

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want. Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources. [44]



Picture 14: Agile Development Life Cycle [46]

3.4.2.1 Categories

1. Scrum
2. Kanban
3. Extreme Programming
4. Lean Software Development
5. Rapid Application Development

3.4.2.1.1 Scrum

Scrum is an agile framework for managing knowledge work, with an emphasis on software development, although it has wide application in other fields and is slowly starting to be explored by traditional project teams more generally. It is designed for teams of three to nine members, who break their work into actions that can be completed within time boxed iterations, called *sprints*, no longer than one month and most commonly two weeks, then track progress and re-plan in 15-minute time-boxed stand-up meetings, called *daily scrums.*

Hirotaka Takeuchi and Ikujiro Nonaka introduced the term *scrum* in the context of product development in their 1986 *Harvard Business Review* article, "The New Product Development Game". The authors described a new approach to commercial product development that would increase speed and flexibility, based on case studies from manufacturing firms in the automotive, photocopier and printer industries. They called this the holistic or rugby approach, as the whole process is performed by one cross-functional team across multiple overlapping phases, in which the team "tries to go the distance as a unit, passing the ball back and forth".

In the early 1990s, Ken Schwaber used what would become Scrum at his company, Advanced Development Methods; while Jeff Sutherland, John Scumniotales and Jeff

McKenna developed a similar approach at Easel Corporation, referring to it using the single word Scrum.

In 2001, Schwaber worked with Mike Beedle to describe the method in the book, *Agile Software Development with Scrum*. Scrum's approach to planning and managing product development involves bringing decision-making authority to the level of operation properties and certainties.

In 2002, Schwaber with others founded the Scrum Alliance and set up the *Certified Scrum* accreditation series. Schwaber left the Scrum Alliance in late 2009 and founded Scrum.org which oversees the parallel *Professional Scrum* accreditation series. Since 2009, a public document called *The Scrum Guide* has officially defined Scrum. In 2018, Schwaber and the Scrum.org community, along with leaders of the Kanban community, published *The Kanban Guide for Scrum Teams*. [44] [45]

A Scrum process is distinguished from other agile processes by specific concepts and practices, divided into the three categories of Roles, Artifacts, and Time Boxes. These and other terms used in Scrum are defined below. Scrum is most often used to manage complex software and product development, using iterative and incremental practices. Scrum significantly increases productivity and reduces time to benefits relative to classic "waterfall" processes. Scrum processes enable organizations to adjust smoothly to rapidly-changing requirements, and produce a product that meets evolving business goals. An agile Scrum process benefits the organization by helping it to:

- Increase the quality of the deliverables
- Cope better with change (and expect the changes)
- Provide better estimates while spending less time creating them
- Be more in control of the project schedule and state

Scrum is facilitated by a scrum master, who is accountable for removing impediments to the ability of the team to deliver the product goals and deliverables. The scrum master is not a traditional team lead or project manager but acts as a buffer between the team and any distracting influences. The scrum master ensures that the scrum framework is followed. The scrum master helps to ensure the team follows the agreed processes in the scrum framework, often facilitates key sessions, and encourages the team to improve. The role has also been referred to as a team facilitator or servant-leader to reinforce these dual perspectives. [20] [37] [46]

3.4.2.1.2 Kanban

Kanban is a lean method to manage and improve work across human systems. This approach aims to manage work by balancing demands with available capacity, and by improving the handling of system-level bottlenecks. Work items are visualized to give participants a view of progress and process, from start to finish - usually via a Kanban board. Work is pulled as capacity permits, rather than work being pushed into the process when requested.

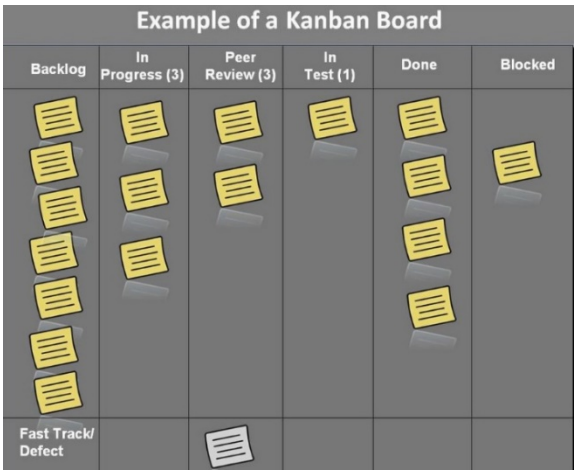
In knowledge work and in software development, the aim is to provide a visual process-management system which aids decision-making about what, when and how much to produce. The underlying Kanban method originated in lean manufacturing (inspired by the Toyota Production System and Taiichi Ohno) it is now used in software development

[20] https://en.wikipedia.org/wiki/Scrum_(software_development)

and technology-related work and has been combined with other methods or frameworks such as Scrum.

David Anderson's 2010 book, *Kanban*, describes the method's evolution from a 2004 project at Microsoft to a 2006-2007 project at Corbis in which the kanban method was identified. [21]

A Kanban board looks like the following:



Picture 15: Kanban Board [48]

Although Kanban does not require that the team or organization use a Kanban board, they can be used to visualize the flow of work. Typically, a Kanban board shows how work moves from left to right, each column represents a stage within the value stream. The image above is a typical view of a simplified Kanban board, where work items move from left to right. In some cases, each column has a work in progress limit. This means that each column can only receive a fixed amount of work items with the aim to encourage focus, and make system constraints evident.

Kanban is becoming a popular way to visualize and limit work-in-progress in software development and information technology work. Teams around the world are adding Kanban around their existing processes to catalyze cultural change and deliver better business agility. [47]

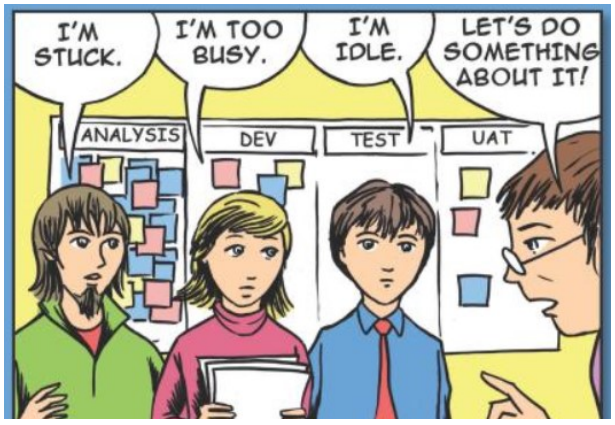
Organizations use the Kanban method to manage the project's creation while emphasizing on the continued delivery and not overburdening the development team. Like Scrum, Kanban processes are designed to help teams work more efficiently together.

There are three principles:

1. Visualize what you do: see all the items within context of each other – more informative
2. Limit the amount of work in progress (WIP): balance the flow-based approach so teams are not committed to doing too much work at once
3. Enhance the flow: as soon as one task is finished, start on the next highest job from the backlog

[21] https://en.wikipedia.org/wiki/Kanban

The Kanban method promotes continued collaboration by the client and team. It encourages ongoing learning and improvements to provide the best possible workflow for the team.



Picture 16: Kanban Board through a cartoon

3.4.2.1.3 Extreme Programming (XP)

Extreme Programming (XP) was originally described by Kent Beck. It is one of the most popular and controversial agile methodologies. XP is a highly disciplined method of continually delivering high-quality software faster. The customer is actively involved with the close-knit team to perform continued planning, testing and rapid feedback to deliver working software frequently. The software should be delivered in intervals everyone to three weeks. [48]

The original XP method is based on four simple values:

1. Simplicity
2. Communication
3. Feedback
4. Courage

It has 12 supporting practices:

1. Planning game
2. Small releases
3. Customer acceptance tests
4. Simple design
5. Pair programming
6. Test-driven development
7. Refactoring
8. Continuous integration
9. Collective code ownership
10. Coding standards
11. Metaphor
12. Sustainable pace

3.4.2.1.4 Lean software development

Lean software development is an iteration methodology originally developed by Mary and Tom Poppendieck. Many of the principles and practices in Lean Software Development came from the lean enterprise movement and were first used by big companies like Toyota. This value based method focuses on giving the customer an efficient "Value Stream" mechanism that delivers the value to the project. [48]

The main principles of this methodology are:

- Eliminate waste
- Amplify learning
- Make decisions as late as possible
- Deliver results as quickly as possible
- Empower the team
- Build integrity
- Envision the whole project

By choosing only the features that have real value for the system, prioritizing and delivering them in small batches eliminates waste. Instead, the lean methodology emphasizes on speed and efficiency; relying on rapid, reliable feedback between the customers and programmers. It focuses on the idea that customer requests "pull" the product. The focus is more about the faster and more efficient decision-making abilities of individuals or small teams instead of a hierarchy controlled method. This methodology concentrates on the efficiencies of its team's resources, ensuring everyone is as productive as possible always. [48]

3.4.2.1.5 Rapid-application development (RAD)

Rapid-application development is both a general term, used to refer to adaptive software development approaches, as well as the name for James Martin's approach to rapid development. In general, RAD approaches to software development put less emphasis on planning and more emphasis on an adaptive process. [48]

RAD is especially well suited for developing software that is driven by user interface requirements. Graphical user interface builders are often called rapid application development tools. Other approaches to rapid development include the adaptive, agile, spiral, and unified models.

3.4.3 Agile vs. waterfall

One of the differences between agile software development methods and waterfall is the approach to quality and testing. In the waterfall model, there is always a separate *testing phase* after a *build phase*; however, in agile software development testing is completed in the same iteration as programming.

Another difference is that traditional "waterfall" software development moves a project through various Software Development Lifecycle (SDLC) phases. One phase is completed in its entirety before moving on to the next phase.

Because testing is done in every iteration—which develops a small piece of the software—users can frequently use those new pieces of software and validate the value. After the users know the real value of the updated piece of software, they can make better decisions about the software's future. Having a value retrospective and software re-planning session in each iteration—Scrum typically has iterations of just two weeks—

helps the team continuously adapt its plans so as to maximize the value it delivers. This follows a pattern similar to the PDCA cycle, as the work is *planned*, *done*, *checked* (in the review and retrospective), and any changes agreed are *acted* upon.

This iterative approach supports a *product* rather than a *project* mindset. This provides greater flexibility throughout the development process; whereas on projects the requirements are defined and locked down from the very beginning, making it difficult to change them later. Iterative product development allows the software to evolve in response to changes in business environment or market requirements.[38]

Because of the short iteration style of agile software development, it also has strong connections with the lean startup concept. [49]

Table 4: Differences between Agile and Waterfall [49]

	Agile	Waterfall
1	It separates the project development lifecycle into sprints.	Software development process is divided into distinct phases.
2	It follows an incremental approach	sequential design process
3	flexibility	rigid
4	Can be considered as a collection of many different projects.	Software development will be completed as one single project.
5	Changes can be made in the project development requirements even if the initial planning has been completed.	No change in the requirements once the project development starts.
6	Iterative development approach - phases may appear more than once.	All the project development phases are completed once
7	Test plan is reviewed after each sprint	The test plan is rarely discussed during the test phase.
8	Requirements are expected to change and evolve.	Definite requirements and changes not at all expected.
9	Testing is performed concurrently with software development.	Testing phase comes after the Build phase
10	Focuses on the needs of its end customers	Focus completely on accomplishing the project.

11	Time & Materials or non-fixed funding. It may increase stress in fixed-price scenarios.	Reduces risk in the firm fixed price contracts by getting risk agreement at the beginning of the process.
12	Prefers small but dedicated teams with a high degree of coordination and synchronization.	Team coordination/synchronization is very limited.
13	Products owner with team prepares requirements just about every day during a project.	Business analysis prepares requirements before the beginning of the project.
14	Test team can take part in the requirements change without problems.	It is difficult for the test to initiate any change in requirements.

3.4.4 This thesis working model

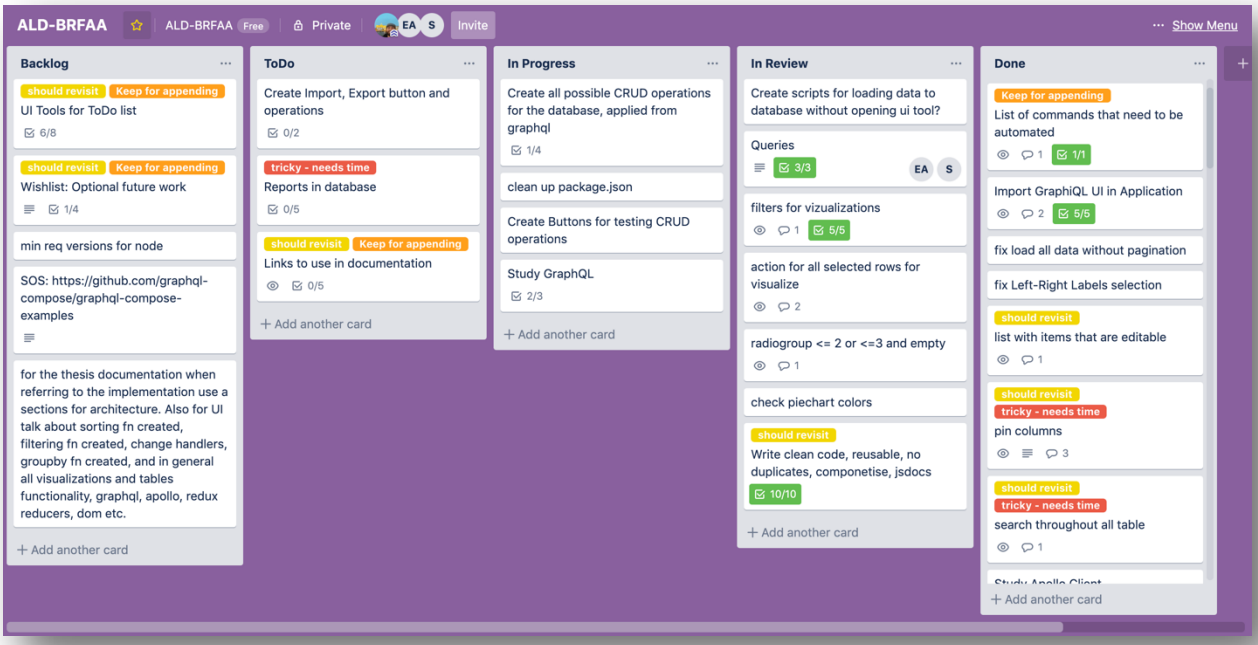
Our thesis working model is based on Kanban lean methodology, as from the start of this thesis we gathered all the responsible people for this thesis, me and my supervisors (in role of stakeholders) and discussed about the working model that we should work with, as I was also working on a full-time job and had only evenings and weekends to work on this thesis in a daily base.

Having considered all available options, both waterfall and agile models, we ended up using Kanban. The reason is that Kanban approach aims to manage work by balancing demands with available capacity, and by improving the handling of system-level bottlenecks. This approach, fitted exactly our case, as the capacity was almost exclusively dependent on my occupation with this thesis in a daily base. What is more, we tried to avoid bottlenecks by analyzing in depth the backlog task that would proceed to the next "To Do" progress phase.

Another reason that resulted in our decision to use Kanban methodology was that in Kanban work items are visualized to give participants a view of progress and process, from start to finish - usually via a Kanban board, which in our case we developed it and visited it daily by using Trello[22]. This way, we shared among us a Trello Kanban-like board (see image below) which the stakeholders (my supervisors) could visit at any time to analyze the backlog or to respond in any comment in a Kanban task that was for me an

[22] https://trello.com/

impediment for software development, while also to have a visual process-management over the progress of my thesis.



Picture 17: This thesis's Kanban board

Using Trello, my supervisors based on their needs and other users' needs could reorder tasks in a Kanban progress column as we decided that the top task in each column had the maximum priority.

Finally, in our Kanban methodology we borrowed two processes, let us refer to them as "Kanban Sprint Review" and "Kanban Sprint Grooming", based on another methodology's functions, the Scrum Sprint Review and the Scrum Sprint Grooming. Scrum is a very useful methodology to work with and is the most applied agile methodology globally. Nevertheless, Scrum requires a team of at least 3 people, and a daily Scrum meeting which in our case, neither of the two could apply, and for this reason we rejected working with this framework. So, based on Scrum, in our Kanban methodology we decided that once in a two-week "Kanban-Sprint" we should meet to review the work done ("Kanban-Review"), and to discuss about the next "Kanban-sprint" backlog tasks that I was going to implement ("Kanban-Grooming").

This methodology of ours worked greatly and appeared very interesting and appealing to stakeholders that did not know about it.

4. SOFTWARE DEVELOPMENT - TECHNOLOGICAL CONCERNS

4.1 Introduction

In this chapter we present the architectural schema of the web application created, and we analyze its technological aspect. We refer to separation of concerns for each entity, and we de-structure the architectural schema in pieces and explain each one of them until we glue them back together again. Then, we present the web application's technologies stack, which we used and how we blended them together. Next, we analyze the coding design patterns that we used and how we assure that the web application is designed with main concerns the scalability, maintainability, reusability and performance, while providing samples of code. Finally, we explain the importance of visualizations and how we selected the visualization charts and graphs that fit better to our use cases.

4.2 Architecture Schema – Technological Overview

In this section we present the architectural schema of the web application we created. In computer science, separation of concerns (SoC)[23] is a design principle for separating a computer program into distinct sections, so that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program. A concern can be as general as the details of the hardware the code is being optimized for, or as specific as the name of a class to instantiate. A program that embodies SoC well is called a modular program. Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a well-defined interface. Encapsulation is a means of information hiding. Layered designs in information systems, as we are going to find in this chapter section, are another embodiment of separation of concerns.

Separation of concerns results in higher degrees of freedom for some aspect of the program's design, deployment, or usage. What is common amongst these is the higher degrees of freedom for reaching simplification and maintenance of code. When concerns are well-separated, there are higher degrees of freedom for module reuse as well as independent development and upgrade. Because modules hide the details of their concerns behind interfaces, increased freedom results to later improve or modify a single concern's section of code without having to know the details of other sections, and without having to make corresponding changes to those sections. Modules can also expose different versions of an interface, which increases the freedom to upgrade a complex system in piecemeal fashion without interim loss of functionality.

4.2.1 Data pre-processing

In this thesis, as in every other thesis, research analysis or business analysis aims in creating new tools based in a case scenario, which will help research to evolve. In order for the developer to create the most efficient tools or models, he needs to have the appropriate data for the specific application. Generally, data plays a crucial role in the journey of making a research or a business model. There is no place nor time for poor data representations or unexpected results, due to incomplete, noisy and inconsistent data. Raw data has to be preprocessed prior to being imported in an application. [50]

Data preprocessing is a proven method for resolving such issues and it is executed in 5 or more simple steps:

[23] https://en.wikipedia.org/wiki/Separation_of_concerns

1. Gather the data
2. Import the datasets
3. Divide the dataset into Dependent and Independent variables
4. Check for missing values and for Categorical values
5. Normalize

Web application's Dataset

Data is raw information, it's the representation of both human and machine observation of the world. Dataset entirely depends on what type of problem you want to solve, and in most cases each problem has its own unique nuance and approach. And sometimes it can be quite hard to find the specific dataset for the variety of problems. Fortunately, there is dataset for every problem and more specifically the dataset for this thesis was given, so as to only have to deal with the application development. [51]

Nonetheless, we still needed to apply some of the above preprocessing rules, mostly for preventing application error cases but also for research purposes as sometimes we needed to calculate the non-answered fields.

So, this is a snapshot of how our dataset looks like:

at	at_interpret	a2m	hapto	apoa1	proc3	PROC3	proc3x	PROC3X
.0664901717225525	A0	2.59	1.809	1.59	9.148	8.6	21.11	
.1835862185148243	A0-A1	2.7	617	2.19	13.776	13.1	9.1	4.6
.0534336601847531	A0	1.706	2.154	1.45	12.278	12.9	4.124	5.4
.6271869358168174	A3	2.328	196	1.5	93.33199999999999	83.7	19.332	14.4
.1492504052560204	A0	2.278	1.58	1.37	38.336	48.8	6.948	
.1198088048584981	A0	1.639	355	1.93	34.986	38.8	6.734	8.5
.2043743439269604	A0-A1	1.759	1.901	1.69	9.824	13.2	7.616	
.1549268746461555	A0	1.961	1.039	1.31	93.322	94.2	14.482	15.3
.3210355212844576	A1	1.363	1.61	1.51	9.956	9.5	10.688	4.5
.0688588881016985	A0	1.229	1.601	1.77	5.558	5.6	2.7	3.4
.5170977410378513	A1-A2	3.275	338	1.78	29.312	34.7	17.604	

Picture 18: Initial Data set

The dataset was given in an MS Excel format from the researchers, so our job was to carefully convert it in CSV format and then import it in our database. Nevertheless, in the following paragraphs the data preprocessing done by the researchers will be presented.

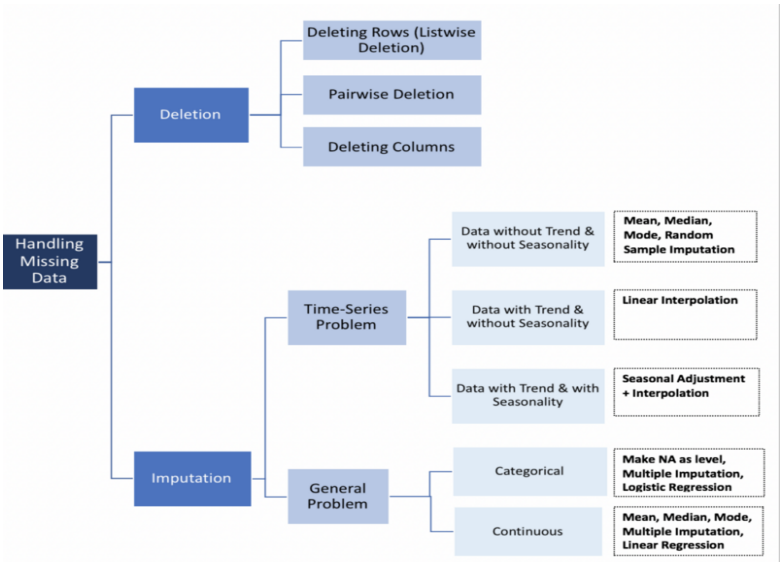
The researchers had to deal with missing variables and decided how they want to fill these fields. In order to anticipate possible miss outs and to overcome crashing, we incorporated the possibility of editing the data cells, in the application. What is more, in some cases the researchers wanted to count also the missing data as an answer, so we also managed to keep track of missing cell data too, using imputation with a constant "not-answered", which means that we replaced missing values with empty strings "" or the null special character (the zero-valued ASCII character) if it was applied in a column of numbers. [52]

```
const groupArrays = groupUniqueCategories.map(
    (category) =>
      category !== ''
        ? {
            key: category,
            value: data.filter(
              (e) => (!!groupByVar1 ? e[groupByVar1] === category : e[xAxis] === category)
            ),
          }
        : {
            key: 'not-answered',
            value: data.filter(
```

```
            (e) => (!!groupByVar1 ? e[groupByVar1] === category : e[xAxis] === category)
          ),
        }
  );
```

Code 1: Show how we represented the empty cells of excel

Other kinds of imputation could be to use the most frequent value, or the mean value, or the median value. In our case though we wanted to keep track of the unassigned cells and don't use any logic for filling them. [53]



Picture 19: All ways of handling missing data based on problem type and data type [97]

In the above figure, our problem classifies as a "Categorical General Problem" and this is also how we decided to handle the missing data, by making NA as a different answer in our filtering dropdown selection checkboxes.

Finally, in our case the dataset as we described previously consists only of patients data, and our end users are going to be researchers and doctors. For this reason, and because the data are not analyzed by a machine learning algorithm, but on the contrary they are going to be analyzed directly by the end user, it is not recommended to scale them and normalize them as such case will confuse the end users. For that reason we skipped this pre-processing step. In future work, though, a machine learning algorithm can be applied and if so then normalization would fit well for algorithm to handle data processing in a better way.

4.2.2 Software Systems Layered Design

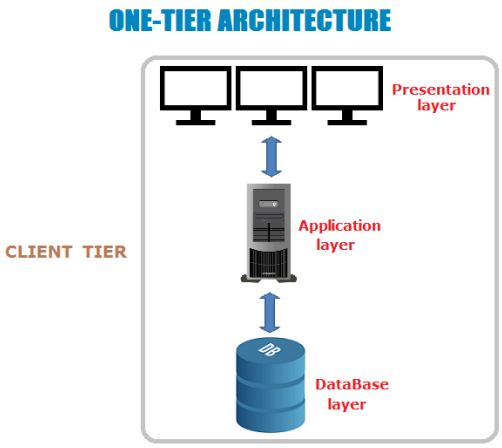
In this thesis one of our main goals is to design a web application interface that can connect to a database and present the database's data with some meaningful visualizations. That said, what we are dealing with, in other words can be described as a Database Management System (DBMS). DBMS is almost used in any application nowadays. DBMSs focus on the design, development, implementation and maintenance of computer programs that store and organize information for businesses, agencies and institutions. DBMSs use programming languages to design a particular type of software for businesses or organizations. The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. The tiers are classified as follows: [54]

1. 1-tier architecture
2. 2-tier architecture

3. 3-tier architecture
4. n-tier architecture.

1. 1-tier architecture

One-tier architecture involves putting all of the required components for a software application or technology on a single server or platform.

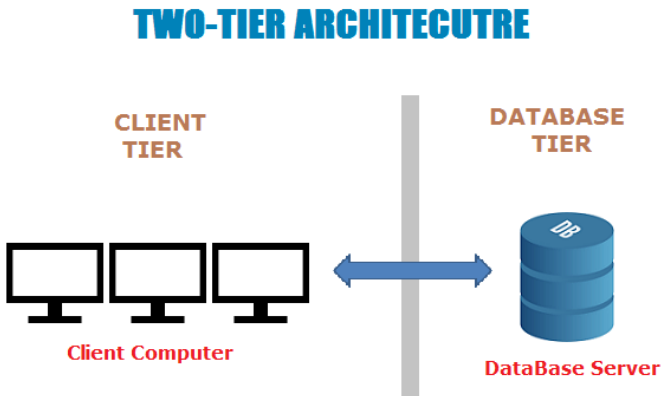


Picture 20: 1-tier architecture [54]

Basically, an one-tier architecture keeps all of the elements of an application, including the interface, Middleware and back-end data, in one place. Developers see these types of systems as the simplest and most direct way.

2. 2-tier architecture

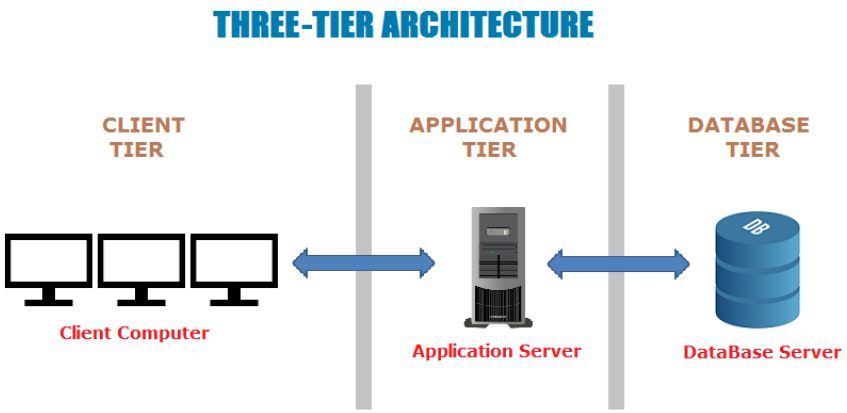
The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server.



Picture 21: 2-tier architecture [54]

3. 3-tier architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



Picture 22: 3-tier architecture [54]

This architecture has different usages with different applications. It can be used in web applications and distributed applications. The strength in particular is when using this architecture over distributed systems.

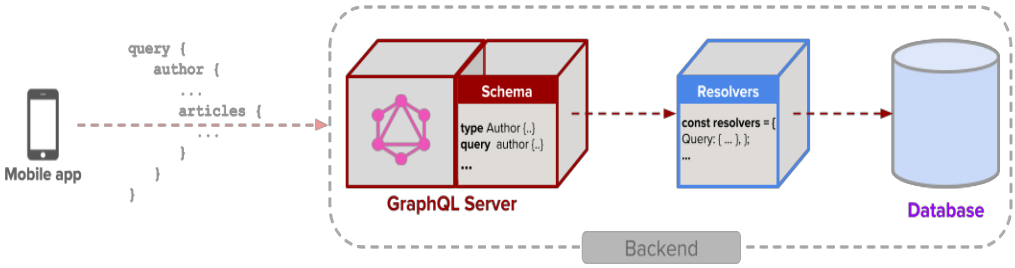
1. Database (Data) Tier − At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level. It is also called the persistence layer.
2. Application (Middle) Tier − At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database. It is also called the data access and business logic layer.
3. User (Presentation) Tier − End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier. It is also called the presentation layer.

4. n-tier architecture

N-tier architecture would involve dividing an application into three different tiers. These would be the following:

1. data access and business logic tier,
2. the presentation tier, and
3. the persistence tier

And then, separate one or more tiers into simpler layers. N-tier is the physical separation of the different parts of the application as opposed to the usually conceptual or logical separation of the elements in the model-view-controller (MVC) framework. Another difference from the MVC framework is that n-tier layers are connected linearly, meaning all communication must go through the middle layer, which is the logic tier. In MVC, there is no actual middle layer because the interaction is triangular, the control layer has access to both the view and model layers and the model also accesses the view, the controller also creates a model based on the requirements and pushes this to the view. However, they are not mutually exclusive, as the MVC framework can be used in conjunction with the n-tier architecture, with the n-tier being the overall architecture used and MVC used as the framework for the presentation tier. [54] [55]



Picture 23: n-tier architecture [55]

In our case, we use a 4-tier architecture where we place a fourth tier between the web-interface (presentation layer) and the server (data access layer). This fourth tier is the business logic tier, which we separated it from being totally connected with the data-access layer. In the above figure we see that we use GraphQL query language to handle all the business logic layer. This tier lets us filter or even alter database schema and create a new schema representation for our data that is more user-friendly. This way user doesn't need to have access to database specific values and data. What is more, this tier gives a lot bigger flexibility to users to search in a database schema and caches users choices for future reuse.

4.2.3 Persistence Layer - Database

When it comes to choosing a database, one of the biggest decisions is picking a relational (SQL) or non-relational (NoSQL) data structure. While both are viable options, there are certain key differences, between the two that users must keep in mind, when making a decision. SQL databases use structured query language (SQL) for defining and manipulating data. On the one hand, this is extremely powerful because SQL is one of the most versatile and widely-used options available, making it a safe choice and especially great for complex queries. On the other hand, it can be restrictive. SQL requires that you use predefined schemas to determine the structure of your data before you work with it. In addition, all of your data must follow the same structure. This can require significant up-front preparation, which if omitted or not handled carefully can result in painful situations where a simple change in the structure, such as a change in a table's field name, would come to be both difficult and disruptive to the whole system.

NoSQL, has also been interpreted as the abbreviation of "Not Only SQL" or "no SQL at all" are types of databases often used for storing of the big data in non-relational and distributed manner, and its concurrency model is weaker than the ACID transactions (standing for Atomicity, Consistency, Isolation, Durability, all being the set of core SQL properties) in relational SQL-like database systems. [56]

In this thesis we use a NoSQL database. First of all, it better matches our data. Also, the database schema may often change and more columns may be added in the future, as well as the schema is created dynamically, based on key-value pairs. Furthermore, we may have a lot of empty cells that do not need storing and as a result we can maintain the size of the database smaller. Also, due to separation of concerns and our n-tier architectural structure model we want to add operations and filters on data where SQL-like databases perform better, so for this reason we transferred the business logic to the GraphQL layer in order to achieve similar performance velocity as if we used SQL-like databases.

As we mentioned above, NoSQL systems are ACID-non-compliant by design, and the complexity for enforcing ACID properties does not exist for most of the properties. For

example some of the ACID compliant NOSQL databases are: Redis, Aerospike and Voldemort as key-value stores; where as Neo4jDB and Sparksee are as graph-based data stores. In contrast to this MongoDB is not ACID compliant document-oriented database, and better matches our data changing scenario, so this is what we decided to use. Nevertheless, column-oriented could also be a good case to use, and graph-based data stores could be a better match when in future it can be used for machine-learning algorithms. [56]

Key-value stores

These are systems that store values against the index keys, as key-value pairs. The keys are unique to identify and request the data values from the collections. Such databases has emerged recently and are influenced heavily by Amazon's Dynamo key-value store database, where data is distributed and replicated across multiple servers. The values in such databases are schema flexible and can be simple text strings or more complex structures like arrays. The simplicity of its data model makes the retrieval of information very fast, therefore supports the big data real-time processing along the scalability, reliability and highly available characteristics.

Column-oriented databases

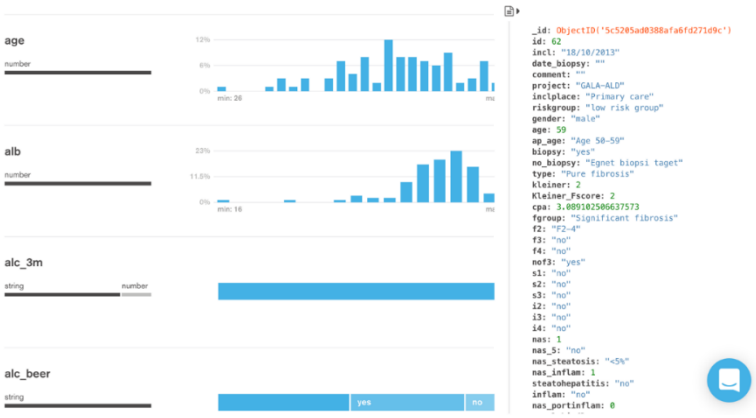
Relational databases have their focus on rows in which they store the instances or the records and return rows or instances of an entity against a data retrieval query. Such rows possess unique keys against each instance for locating the information. Whereas column-oriented databases store their data as columns instead of the rows and use index based unique keys over the columns for the data retrieval. This supports attribute level access rather than the tuple-level access pattern.

Graph databases

Graph databases, as a category of NoSQL technologies, represent data as a network of nodes connected with edges and are having properties of key-value pairs. Working on relationships, detecting patterns and finding paths are the best applications to be solved by representing them as graphs.

Document databases

These are the most general models, which use JSON (JavaScript Object Notation) or BSON (Binary JSON) format to represent and store the data structures as documents for the data management. Document stores provide schema flexibility by allowing arbitrarily complex documents, i.e. sub-documents within document or sub-documents; and documents as lists. A database comprises one or more collections, where each collection is a named group of documents. A document can be a simple or complex value, a set of attribute-value pairs, which can comprise simple values, lists, and even nested sub documents. Documents are schema-flexible, as one can alter the schema at the run time hence providing flexibility to the programmers to save an object instances in different formats, thus supporting polymorphism at the database level.



Picture 24: Mongodb representation of data [56]

The above figure illustrates a patient's document using MongoDB, our selection for a document database. It is evident that a collection can have different formats of documents in JSON format and they have hierarchies among themselves. [56]

4.2.4 Data-access Layer – Back-end Server

A back-end system is any system that supports back-office applications. These systems are used as part of corporate management and they work by obtaining user input and gathering input from other systems to provide responsive output. The separation of front-end and back-end computer systems simplifies the computing process when dealing with multilayered development and maintenance. Back-end systems deal with databases and data processing components, so the purpose of the back-end system is to launch the operating system's programs in response to front-end system requests and operations. In other words, the back-end system implements responses to what the front end has initiated. [57]

The main reason for using a backend server is to abstract the actual database engine or other data store, such that the application we use can switch a database model to another without the other layers(like the middleware and the presentation layer being affected. More specifically, as mentioned above, there may be a possible future change from using MongoDB to using Neo4j graph database, as in case we apply machine learning, Neo4j is a better applying database to our dataset. So, now that we use a data-access layer, this change can happen pretty easily. [57]

```
// Project ALD-BRFAA-API
// Filepath: src/server.js

connect('mongodb://localhost/ALD-BRFAA');
connection.once('open', () => {
  console.log('Database connected!');
});
server.listen().then(({ url }) => {
  console.log(`🚀 Server ready at ${url}`);
});
```

Code 2: Connect server to mongodb database

Another case for using a data-access layer is to abstract the logical data model such that the business-logic layer is decoupled from this knowledge and is agnostic of it. This gives to the system structure the ability to have its logical data model modified without impacting the business layer. In our application's case, we also did abstract the logical data model[24].

[24] https://medium.com/@ahsan.ayaz/how-to-find-schema-of-a-collection-in-mongodb-d9a91839d992

For the thesis purposes, we just used the same labels and naming in order to make an abstraction, using the mongo schema of patients collection extracted with an open-source tool[25] that currently doesn't have any affect but in the future it will be easier to use this abstraction for business purposes. [57]

```
// Project ALD-BRFAA-API
// Filepath: src/models/patients
// STEP 1: DEFINE MONGOOSE SCHEMA AND MODEL
const patientSchema = new Schema(schema.patients);
const Patient = mongoose.model('Patient', patientSchema);
```

Code 3: Abstract logical data using mongoose configuration options

```
// Project ALD-BRFAA-API
// Filepath: schema.json
// extracted mongo schema using extraction tool[4]
{
  "patients": {
    "_id": {
      "type": "Object",
      "structure": {
        "_bsontype": {
          "type": "string",
          "required": false
        },
        "id": {
          "type": "Uint8Array",
          "required": false
        },
        "get_inc": {
          "type": "function",
          "required": false
        },
...
    },
    "age": {
      "type": "number",
      "required": false
    },
    "ap_age": {
      "type": "string",
      "required": false
    },
    "biopsy": {
      "type": "string",
      "required": false
    },
    "no_biopsy": {
      "type": "string",
      "required": false
    },
    "kleiner": {
      "type": "number",
      "required": false
    },
...
    "smoke": {
      "type": "string",
      "required": false
    },
    "smokeyears": {
      "type": "number",
      "required": false
    },
    "smokevolume": {
      "type": "number",
      "required": false
    },
```

[25] https://github.com/perak/extract-mongo-schema

```
...
}
```

Code 4: Mongo extracted schema for use by GraphQL

4.2.5 Business-Logic Layer – Back-end Middleware (using GraphQL)

The business logic layer is the business components that provide services to return data or start business processes. The presentation layer uses these services to display data, or to invoke a business process. The business logic provides data required by the presentation layer. The business logic layer exists because in most cases more than just fetching and updating data is required by an application; there is also additional business logic independent of the presentation layer. [60]

This business logic is better practice to be used in the backend in order to be handled once and don't depend on the client-side systems. Software developers tend to represent this layer as a software in the middle of the presentation layer and the database layer (middleware). Middleware is computer software that provides services to software applications beyond those available from the operating system. It can be described as "software glue". Middleware is multipurpose software that provides services to applications outside of what's offered by the operating system. Any software between the kernel and user apps can be middleware. Middleware doesn't offer the functions of a traditional app, it connects software to other software. For instance, GraphQL is open source middleware that offers real-time data querying capabilities for your applications. There exists a variety of Middleware software, but in general we can categorize it to:

Application Programming Interface (API)

APIs are sets of tools, definitions, and protocols for building application software, which lets your product or service communicate with other products and services without having to know how they're implemented. In our case of the web interface we used the GraphQL API middleware as a surface between the GUI and the server that connents to our database.

Application Server

Platform for app development. An application server is a framework that provides the functionality to create apps and a server on which to run them.

Application Integration

Application integration is the practice of combining data from several apps through an integration framework. The framework can limit the number of point-to-point connections across your organization that can lead to complex dependencies and potential points of failure.

Data Integration

Data integration is the practice of combining data from heterogeneous sources into a unified view for users to access and manipulate.[26]

4.2.6 Presentation Layer – Front-end User Interface

A front-end system is part of an information system that is directly accessed and interacted with by the user to receive or utilize back-end capabilities of the host system.

[26] https://www.redhat.com/en/topics/middleware/what-is-middleware

It enables users to access and request the features and services of the underlying information system. The front-end system can be a software application or the combination or hardware, software and network resources. A front-end system is primarily used to send queries and requests, and receive data from the back-end system or the host information system. It serves or provides users with the ability to interact and use an information system. Typically, front-end systems have very limited computational or business logic processing capabilities and rely on the data and functions from the host system. However, nowadays this has been changed and companies tend to have some more advanced level front-end systems do maintain copies of data, cache them in local client's memory for performance reasons and make quick simple computations that are faster to do in client-side than unnecessarily connect and stream to backend to do so, for example store in local memory a query answer that previously was queried in server, in order to fetch the data from cache than from database or backend middleware.

A front-end system may include or consist of textual or graphical user interface (GUI) , visualizations, and/or a front-end client application that is connected by the back-end system. In this thesis case, our front-end is written mostly in JavaScript using React's JSX and we use all of the aforementioned processes, like caching, visualizations, etc. that we will also analyze in further in the next section.[27]

4.3 Web Application's Technologies Stack

In this section we present the stack of the technologies used to build this web interface for querying data patients data and easily manipulate and visualize them with a handy user interface with the use of UX principles. In short, everything we used is open-source and free to use, and is applicable to work in any system's platform architecture and operation system (i.e. Windows, Unix based systems, etc.).

The main pillar for this web interface's software development were:

1. Git tracking system for code tracking and changelog
2. MongoDB as the NoSQL application's database
3. Node.js for data access to MongoDB and serving to client's Web-UI and for manipulating the middleware GraphQL using Apollo engine
4. ReactJS, for creating the Web-UI, with main support of packages Material-UI that follows the material design principles, Ant Design – which is a powerful UI components' library and react-vis, which we used for creating the visualizations.

4.3.1 Git

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its' goals include speed, data integrity, and support for distributed, non-linear workflows. As with most other distributed version-control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server.

[27] https://www.techopedia.com/definition/3799/front-end-system

Some of Git's features that make it stand out through its other competitor version-control systems are:

1. Branching and Merging
2. Size and performance
3. It can be distributed
4. Data assurance
5. It has a third area of "working space" except for the local and remote - the "stage" area
6. It is free and open-source

Branching and merging is the most important aspect that lets Git standalone from nearly every other SCM out there. Git allows and encourages you to have multiple local branches that can be entirely independent from each other. The creation, merging, and deletion of those lines of development takes seconds. [58]

Picture 25: Git branching and merging [58]

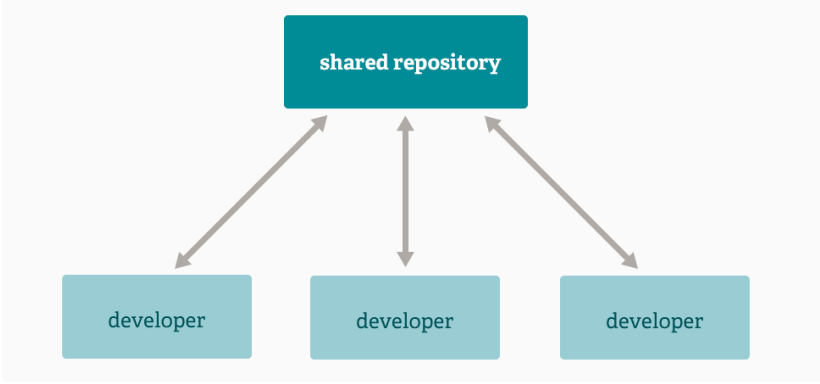
```
nmpegetis (wipdeploy) ALD-BRFAA-api
$ git lg2
* eb34edb - Wed, 19 Jun 2019 18:56:53 +0300 (3 days ago) (HEAD -> wipdeploy, origin/wipdeploy)
|          wip prepare for deploy - Nikolas Begetis
| *   e108b91 - Wed, 19 Jun 2019 18:56:21 +0300 (3 days ago) (refs/stash)
| |\           On wipdeploy: prepareForDeploy20190619 - Nikolas Begetis
|/ /
| * b9b4c01 - Wed, 19 Jun 2019 18:56:21 +0300 (3 days ago)
|/           index on wipdeploy: bd6133c build: update packages and add mongo and server distinct script
and start - Nikolas Begetis
* bd6133c - Mon, 10 Jun 2019 00:49:00 +0300 (13 days ago) (master)
|          build: update packages and add mongo and server distinct script and start - Nikolas Begetis
* f65883c - Thu, 31 Jan 2019 23:44:34 +0200 (5 months ago)
|          feat: load mongodb data to graphql - Nikolas Begetis
| * f01927e - Thu, 31 Jan 2019 23:44:34 +0200 (5 months ago) (origin/master, origin/HEAD)
|/           feat: load mongodb data to graphql - Nikolas Begetis
* 55376b2 - Wed, 30 Jan 2019 23:30:21 +0200 (5 months ago)
|          feat: add package for turning a mongodb schema to json - Nikolas Begetis
* 1eee298 - Tue, 29 Jan 2019 01:31:18 +0200 (5 months ago)
|          feat: connect apollo-server to mongodb example by creating a patient collection. - Nikolas Beg
etis
```

Code 5: Git branching and merging in our use-case in local repository

Git size and performance also make it stand out as nearly all operations are performed locally, giving it a huge speed advantage on centralized systems that constantly have to communicate with a server somewhere. Initially, Git was built to work on the Linux kernel, meaning that it has had to effectively handle large repositories from day one. Git is written in C, reducing the overhead of runtimes associated with higher-level languages. Some benchmarks from the official git page show that common operations stack up against Subversion, a common centralized version control system that is similar to CVS or Perforce. *Smaller is faster.*

Picture 26: Git vs. SVN Benchmarks [58]

Git is distributed which means that even if you're using a centralized workflow, every user essentially has a full backup of the main server. Each of these copies could be pushed up to replace the main server in the event of a crash or corruption. In effect, there is no single point of failure with Git unless there is only a single copy of the repository.



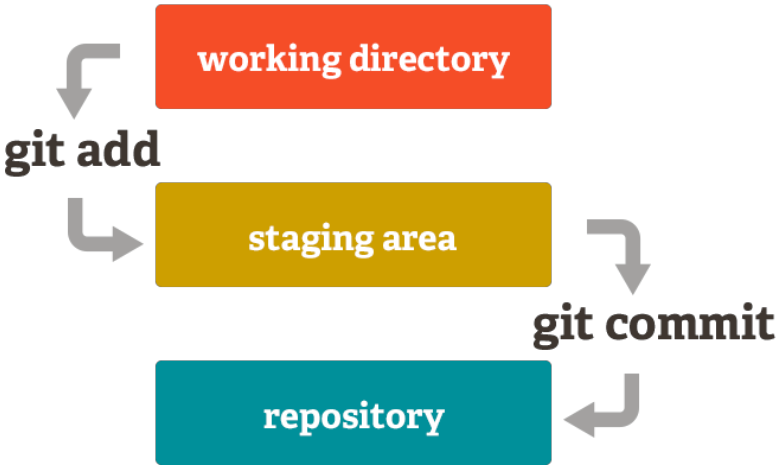
Picture 27: Git is distributed. Each developer has his own local copy of a shared repository [58]

Git is safe and provides users with data assurance. This happens because the data model that Git uses, ensures the cryptographic integrity of every bit of your project. Every file and commit is checksummed and retrieved by its checksum when checked back out. It's impossible to get anything out of Git other than the exact bits you put in.

As referenced earlier, Git, unlike the other systems, has something a third "area", called the "staging area" or "index". This is an intermediate area where commits can be formatted and reviewed before completing the commit. This is also one of the reasons that sets Git apart from other tools, as it's possible to quickly stage some of your files and commit them without committing all of the other modified files in your working directory or having to list them on the command line during the commit.[28] [29]

[28] https://medium.com/@nmpegetis/git-how-to-start-code-changes-commit-and-push-changes-when-working-in-a-team-dbc6da3cd34c

[29] https://medium.com/@nmpegetis/git-commit-message-conventions-841d6998fc4f



Picture 28: Git intermediate staging area [58]

```
nmpegetis (master *+) ALD-BRFAA-api
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 2 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   src/models/patient.js

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   src/server.js

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        api.zip
        schema.html
        schema.json
```

Picture 29: Git use case of intermediate staging area, working directory and local-remote differences [58]

Finally, Git is released under the GNU General Public License version 2.0[30], which is an open source license[31]. The Git project chose to use GPLv2 to guarantee freedom to share and change free software, which ensures that the software is free for all its users.[32]

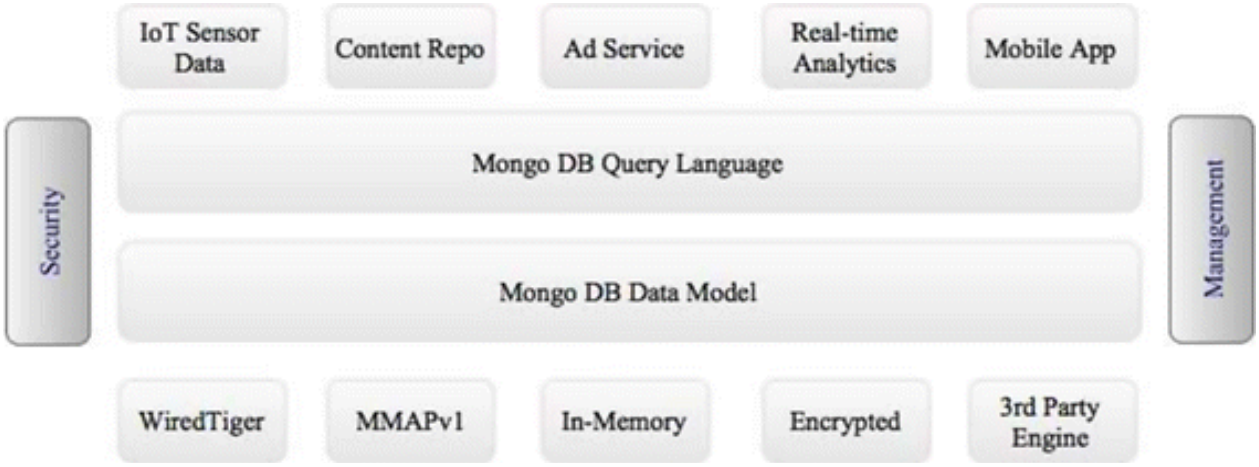
[30] GNU General Public License version 2.0

[31] open source license

[32] https://medium.com/@nmpegetis/git-how-to-start-code-changes-commit-and-push-changes-when-working-in-a-team-dbc6da3cd34c

4.3.2 MongoDB

MongoDB, created by 10gen software company in 2007, is a document oriented database for today's applications which are not possible to develop using the traditional relational databases. It is an IoT database which instead of tables (as in RDBMS) provides one or more "collection(s)" as main storage components consisted upon similar or different JSON or BSON based documents or sub documents. Documents that tend to share some of the similar structure are organized as collections, which can be created at any time, without predefinitions. A document can simply be considered as a row or instance of an entity in RDBMS (Relational DBMS), but the difference is that, in MongoDB we can have instances within instances or documents with in documents, even lists or arrays of documents. The types for the attributes of a document can be of any basic data type, such as numbers, strings, dates, arrays or even a sub-document.



Picture 30: Flexible storage architecture, optimizing MongoDB for unique application demands [56]

MongoDB provides unique multiple storage engines within a single deployment and automatically manages the movement of data between storage engine technologies using native replication. MongoDB 3.2 consists of four efficient storage engines as shown in the above figure, all of which can coexist within a single MongoDB replica set. The default Wired Tiger storage engine provides concurrency control and native compression with best storage and performance efficiency. MongoDB allows both the combinations of in-memory engine for ultra-low-latency operations with a disk-based engine for persistence altogether.

It allows to build large-scale, highly available, robust systems and enables different sensors and applications to store their data in a schema flexible manner. There is no database blockage, such as we encounter during "alter table" commands in RDBMS during schema migrations. However in rare cases, such as during the write-intensive scenarios in master-slave nature of MongoDB there may be blockage at the document level or bottleneck to the system if sharding is not used, but these cases are avoidable. MongoDB enables horizontal scalability because table joins are not as important as they are in the traditional RDBMS. MongoDB provides auto-sharding in which more replica server nodes can easily be added to a system. It is a very fast database and provides indexes not only on the primary attributes rather also on the secondary attributes within the sub-documents even. For the cross comparison analysis between different collections we have different technologies, such as aggregation framework, Map Reduce Hadoop etc. [56] [59] [33]

[33] https://www.mongodb.com/

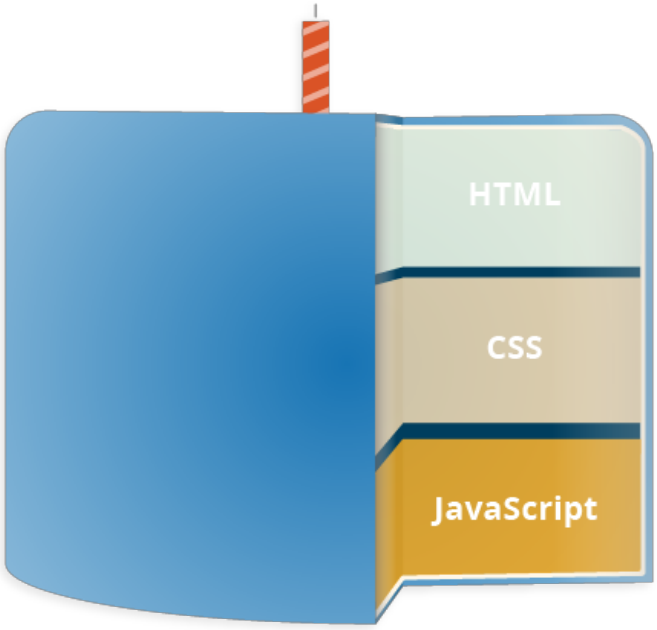
In this thesis our dataset is large, but not so that latencies would be incorporated or sharding would be needed. So for the purposes of the present thesis, the goals achieved by using MongoDB were multiple.

1. Schema-flexibility, for adding more columns or even a new dataset.
2. Polymorphism, for saving object instances in different formats at database level.
3. Fast installation and changes that require schema change.

All of these purposes are met by using MongoDB as our researchers are familiar with the JSON-like structure, and so they can easily understand a MongoDB schema and edit it if needed. Importing a CSV in mongo is handy and can be achieved with the *mongoimport* command which provides the option to convert the csv directly to a mongo collection by creating its mongo schema in the background. In this thesis these commands are running in a docker container which works in all types of operating systems that in future may be installed. Now, docker is installed in a lab server that we use for deployment purposes.

4.3.3 JavaScript

JavaScript is a scripting or programming language that allows the implementation of complex things on web pages, like displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. It is the third layer of the layer cake of standard web technologies, alongside HTML and CSS.



Picture 31: Standard web technologies [60]

In this graph, HTML is the markup language that we use to structure and give meaning to our web content, for example defining paragraphs, headings, and data tables, or embedding images and videos in the page. CSS is a language of style rules that we use to apply styling to our HTML content, for example setting background colors and fonts, and laying out our content in multiple columns. Also, JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and almost everything else one can find in a web page.

Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases, and in non-web programs such as word processors and PDF software, and in runtime environments that make JavaScript available for writing mobile and desktop applications, including desktop widgets.

The terms Vanilla JavaScript and Vanilla JS refer to JavaScript not extended by any frameworks or additional libraries. Scripts written in Vanilla JS are plain JavaScript code. Although there are similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design. JavaScript was influenced by programming languages such as Self and Scheme.

The JSON serialization format, used to store data structures in files or transmit them across networks, is based on JavaScript. This format is also used in many NoSQL databases, as we already saw before, and more specifically for our web application, this format is used both in the MongoDB database and in the client-side labels handling that we use for all columns to have more user-friendly names. [60]

Some of JavaScript's common features using an eagle's eye perspective list as follows:

- **Universal support**

It has a Universal support, which means that all popular modern Web browsers support JavaScript with built-in interpreters.

- **Imperative and structured**

It is imperative and structured, meaning that it supports much of the structured programming syntax from C (e.g., if statements, while loops, switch statements, do while loops, etc.). One partial exception is scoping: JavaScript originally had only function scoping with var. ECMAScript 2015 added keywords let and const for block scoping, meaning JavaScript now has both function and block scoping. Like C, JavaScript makes a distinction between expressions and statements. One syntactic difference from C is automatic semicolon insertion, which allows the semicolons that would normally terminate statements to be omitted.

- **Dynamic typing and runtime evaluation**

It is Dynamic in typing and in runtime evaluation. Dynamically typed like most other scripting languages, means that a type is associated with a value rather than an expression. For example, a variable initially bound to a number may be reassigned to a string. JavaScript supports various ways to test the type of objects, including duck typing. Run-time evaluation means that it includes an eval[function that can execute statements provided as strings at run-time.

- **Prototype-based**

It is Prototype-based (object-oriented). In JavaScript, an object is an associative array, augmented with a prototype (see below); each string key provides the name for an object property, and there are two syntactical ways to specify such a name: dot notation (obj.x = 10) and bracket notation (obj['x'] = 10). A property may be added, rebound, or deleted at run-time. Most properties of an object (and any property that belongs to an object's prototype inheritance chain) can be enumerated using a for...in loop. JavaScript has a small number of built-in objects, including Function and Date. Prototypes: JavaScript uses prototypes where many other object-oriented languages use classes for inheritance. It is possible to simulate many class-based features with prototypes in JavaScript. Functions as object constructors: Functions double as object constructors, along with their typical role. Prefixing a function call with new will create an instance of a prototype, inheriting properties and methods from the constructor (including properties from the Object

prototype). ECMAScript 5 offers the Object.create method, allowing explicit creation of an instance without automatically inheriting from the Object prototype (older environments can assign the prototype to null).

The constructor's prototype property determines the object used for the new object's internal prototype. New methods can be added by modifying the prototype of the function used as a constructor. JavaScript's built-in constructors, such as Array or Object, also have prototypes that can be modified. While it is possible to modify the Object prototype, it is generally considered bad practice because most objects in JavaScript will inherit methods and properties from the Object prototype, and they may not expect the prototype to be modified. Functions as methods: Unlike many object-oriented languages, there is no distinction between a function definition and a method definition. Rather, the distinction occurs during function calling; when a function is called as a method of an object, the function's local this keyword is bound to that object for that invocation.

- **Functional**

It is Functional. A function is first-class; a function is considered to be an object. As such, a function may have properties and methods, such as .call() and .bind(). A nested function is a function defined within another function. It is created each time the outer function is invoked. In addition, each nested function forms a lexical closure: The lexical scope of the outer function (including any constant, local variable, or argument value) becomes part of the internal state of each inner function object, even after execution of the outer function concludes. JavaScript also supports anonymous functions.

- **Delegative**

It is delegative, as it supports implicit and explicit delegation. More specifically JS provides Functions as roles (Traits and Mixins) and Object composition and inheritance. Functions as roles (Traits and Mixins), means that JavaScript natively supports various function-based implementations of Role patterns like Traits and Mixins. Such a function defines additional behavior by at least one method bound to the this keyword within its function body. A Role then has to be delegated explicitly via call or apply to objects that need to feature additional behavior that is not shared via the prototype chain. What is more, Object composition and inheritance means that whereas explicit function-based delegation does cover composition in JavaScript, implicit delegation already happens every time the prototype chain is walked in order to, e.g., find a method that might be related to but is not directly owned by an object. Once the method is found it gets called within this object's context. Thus inheritance in JavaScript is covered by a delegation automatism that is bound to the prototype property of constructor functions[34]. [61]

4.3.3.1 Node.js

Node.js is an open-source, cross-platform JavaScript run-time environment that runs JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting, meaning to run scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server- and client-side scripts.

Though .js is the standard filename extension for JavaScript code, the name "Node.js" does not refer to a particular file in this context and is merely the name of the product. Node.js has an event-driven architecture capable of asynchronous I/O. These design

[34] https://github.com/getify/You-Dont-Know-JS

choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications.

The Node.js distributed development project, governed by the Node.js Foundation, is facilitated by the Linux Foundation's Collaborative Projects program. Node.js allows the creation of Web servers and networking tools using JavaScript and a collection of "modules" that handle various core functionality.

Modules are provided for file system I/O, networking (DNS, HTTP, TCP, TLS/SSL, or UDP), binary data (buffers), cryptography functions, data streams, and other core functions. Node.js's modules use an API designed to reduce the complexity of writing server applications. Though initially the module system was based on commonjs module pattern, the recent introduction of modules in the ECMAScript specification has shifted the direction of using ECMAScript Modules in Node.js by default instead. Node.js is primarily used to build network programs such as Web servers. [62] [63]

The most significant difference between Node.js and PHP is that most functions in PHP block until completion (commands only execute after previous commands finish), while Node.js functions are non-blocking (commands execute concurrently or even in parallel, and use callbacks to signal completion or failure). So, some of Node.js's features using an eagle's eye perspective list as follows:

Platform architecture

Node.js brings event-driven programming to web servers, enabling development of fast web servers in JavaScript. Developers can create scalable servers without using threading, by using a simplified model of event-driven programming that uses callbacks to signal the completion of a task. Node.js connects the ease of a scripting language (JavaScript) with the power of Unix network programming.

Node.js was built on the Google V8 JavaScript engine since it was open-sourced under the BSD license. It is proficient with internet fundamentals such as HTTP, DNS, TCP. JavaScript was also a well-known language, making Node.js accessible to the web development community. Node.js is a JavaScript runtime environment that processes incoming requests in a loop, called the event loop.

Threading

Node.js operates on a single thread event loop, using non-blocking I/O calls, allowing it to support tens of thousands of concurrent connections without incurring the cost of thread context switching. The design of sharing a single thread among all the requests that use the observer pattern is intended for building highly concurrent applications, where any function performing I/O must use a callback. To accommodate the single-threaded event loop, Node.js uses the libuv library - which, in turn, uses a fixed-sized thread pool that handles some of the non-blocking asynchronous I/O operations.

A thread pool handles execution of parallel tasks in Node.js. The main thread function call posts tasks to the shared task queue, which threads in the thread pool pull and execute. Inherently non-blocking system functions such as networking translate to kernel-side non-blocking sockets, while inherently blocking system functions such as file I/O run in a blocking way on their own threads. When a thread in the thread pool completes a task, it informs the main thread of this, which in turn, wakes up and executes the registered callback.

A downside of this single-threaded approach is that Node.js doesn't allow vertical scaling by increasing the number of CPU cores of the machine it is running on without using an additional module, such as cluster, StrongLoop Process Manager, or pm2. However, developers can increase the default number of threads in the libuv thread pool. The server operating system (OS) is likely to distribute these threads across multiple cores. Another

problem is that long lasting computations and other CPU-bound tasks freeze the entire event-loop until completion.

V8 JavaScript Engine

V8 is the JavaScript execution engine which was initially built for Google Chrome. It was then open-sourced by Google in 2008. Written in C++, V8 compiles JavaScript source code to native machine code during runtime instead of interpreting it in ahead of time (AOT). Node.js uses libuv to handle asynchronous events. Libuv is an abstraction layer for network and file system functionality on both Windows and POSIX-based systems such as Linux, macOS, OSS on NonStop, and Unix. The core functionality of Node.js resides in a JavaScript library. The Node.js bindings, written in C++, connect these technologies to each other and to the operating system.

Package management

npm is the pre-installed package manager for the Node.js server platform. It installs Node.js programs from the npm registry, organizing the installation and management of third-party Node.js programs. Packages in the npm registry can range from simple helper libraries such as Lodash to task runners such as Grunt.

Unified API

Node.js can be combined with a browser, a database that supports JSON data (such as Postgres, MongoDB – what we used in this thesis application, or CouchDB) and JSON for a unified JavaScript development stack. With the adaptation of what were essentially server-side development patterns such as MVC, MVP, MVVM, etc., Node.js allows the reuse of the same model and service interface between client-side and server-side.

Event loop

Node.js registers with the operating system so the OS notifies it of connections and issues a callback. Within the Node.js runtime, each connection is a small heap allocation. Traditionally, relatively heavyweight OS processes or threads handled each connection. Node.js uses an event loop for scalability, instead of processes or threads. In contrast to other event-driven servers, Node.js's event loop does not need to be called explicitly. Instead callbacks are defined, and the server automatically enters the event loop at the end of the callback definition. Node.js exits the event loop when there are no further callbacks to be performed.

Project governance

In 2015, various branches of the greater Node.js community began working under the vendor-neutral Node.js Foundation. The stated purpose of the organization "is to enable widespread adoption and help accelerate development of Node.js and other related modules through an open governance model that encourages participation, technical contribution, and a framework for long-term stewardship by an ecosystem invested in Node.js' success."

The Node.js Foundation Technical Steering Committee (TSC) is the technical governing body of the Node.js Foundation. The TSC is responsible for the core Node.js repo as well as dependent and adjacent projects. Generally the TSC delegates administration of these projects to working groups or committees. The LTS group that manages long term supported releases is one such group. Other current groups include: Website, Streams, Build, Diagnostics, i18n, Evangelism, Docker, Addon API, Benchmarking, Post-mortem, Intl, Documentation, and Testing. [64]

Usage in our web application

In this thesis we implemented the whole application in two different code projects, one for the frontend and one for the backend. As we described in previous sections the data persistence layer, the data access layer and the business logic layer are commonly all of them parts of the so called "backend". What is more while describing Node.js we referenced in the **Unified API** that it can be combined with a browser, a database that supports JSON data (in our case MongoDB) and JSON (we used it for data-columns user-friendly labeling) for a unified JavaScript development stack. So, as a matter of fact this is how we organized our application development.

We adapted what used to be essentially server-side development patterns like MVC, MVP, MVVM, etc., and reused the same model and service interface between client-side and server-side. To do so, we had Node.js running in backend and handle anything that had to do with connection to MongoDB, data providing to GraphQL interface using Apollo, assigning Apollo GraphQL response to API services and finally serving its API services to presentation layer.

```
// Project ALD-BRFAA-API
// Filepath: src/server.js

import { ApolloServer, gql } from 'apollo-server';
import { connect, connection } from 'mongoose';
import graphqlSchema from './models/patient';

const server = new ApolloServer({ schema: graphqlSchema });

connect('mongodb://localhost/ALD-BRFAA');
connection.once('open', () => {
  console.log('Database connected!');
});
server.listen().then(({ url }) => {
  console.log(`🚀 Server ready at ${url}`);
});
```

Code 6: Create server, connect mongo, run API services and listen for connection

Packages

Using packages already tested and published by their owners is very common in application development, as all software developers want to finish their work the fastest they can and don't want to reinvent the wheel. But, as the dependencies grow in number and as the time passed and new package versions are released this may result in an obstacle. So to avoid such impediments or the so called Dependency Hell[35], we always have to be careful and research the packages that we are using that they have for instance a long time endurance, many backers and contributors, or are supported from big organizations.

With that said, in this application we decided to use the less possible needed shared packages, in order to avoid on the one hand the possible future application breaking due to dependency hell impediments, and on the other hand to use only the required really helpful and back-supported dependencies that will make the development faster.
The backend packages in this application's backed list as follow:

1. **Apollo-server[36]**

   Apollo Server is the best way to quickly build a production-ready, self-documenting API for GraphQL clients, using data from any source. It's open-source and works

[35] https://en.wikipedia.org/wiki/Dependency_hell

[36] https://www.apollographql.com/docs/apollo-server/

great as a stand-alone server, an addon to an existing Node.js HTTP server, or in "serverless" environments.

2. **GraphQL compose Group of packages needed**[37]

   GraphQL compose is a toolkit for generating complex GraphQL schemas in Node.js. We used it to create GraphQL types and more specifically derive automatically the GraphQLType from our mongoose model. What is more GraphQL compose provided us with some convenient CRUD resolvers, including relay connection and pagination. So finally we had easily found a way to serve database data to the presentation layer.

3. **Mongoose**[38]

   Mongoose is an elegant mongodb object modeling helper tool, built for node.js. It provides a straight-forward, schema-based solution to model our application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

4. **Nodemon**[39]

   Nodemon is a utility that will monitor for any changes in your source and automatically restart your server. Some of its features that helped our development is:
   - Automatic restarting of application.
   - Detects default file extension to monitor.
   - Default support for node & coffeescript, but easy to run any executable (such as python, make, etc).
   - Ignoring specific files or directories.
   - Watch specific directories.

5. **Cross-env**[40]

   Run scripts that set and use environment variables across platforms. Useful for building the application in all environment and operating systems.

6. **Babel Group of packages needed**[41]

   Babel is a JavaScript transpiler, or in other words a compiler for writing next generation JavaScript. By using Babel, our ES6+ code can be compiled down to a supported version.

7. **Extract-mongo-schema**[42]

   Extract mongo schema is a small package script developed for extracting the mongo schema to a JSON file with ease. We used it only once in the beginning of code writing as we needed to convert it later to a graphql-schema.

---

[37] https://graphql-compose.github.io/

[38] https://mongoosejs.com/

[39] https://nodemon.io/

[40] https://www.npmjs.com/package/cross-env

[41] https://babeljs.io/

[42] https://github.com/perak/extract-mongo-schema

```
// Project ALD-BRFAA-API
// Filepath: package.json

{
  "name": "adl-brfaa-api",
  "version": "0.1.0",
  "scripts": {
    "server": "nodemon --exec babel-watch src/server.js",
    "start": "cross-env NODE_ENV=development yarn server"
  },
  "keywords": [
    "apollo",
    "apollo-server",
    "api",
    "graphql"
  ],
  "author": "nmpegetis@gmail.com",
  "dependencies": {
    "apollo-server": "^2.3.2",
    "graphql": "^14.1.1",
    "graphql-compose": "^7.1.0",
    "graphql-compose-connection": "^6.0.3",
    "graphql-compose-mongoose": "^7.0.3",
    "graphql-compose-pagination": "^6.0.3",
    "mongoose": "^5.4.8",
    "nodemon": "^1.18.9"
  },
  "devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-preset-env": "^1.7.0",
    "babel-watch": "^2.0.8",
    "cross-env": "^5.1.3",
    "extract-mongo-schema": "^0.2.7"
  }
}
```

Code 7: The package.json showing the backend dependencies

4.3.3.2 GraphQL middleware (with Apollo)

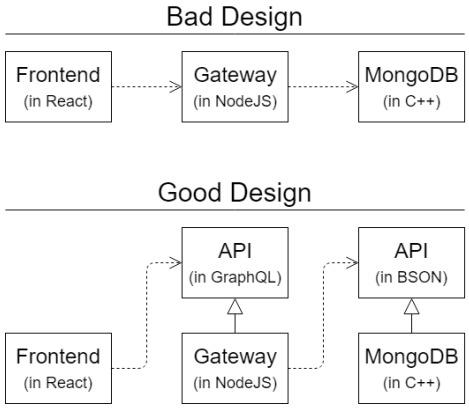
In the previous sections we already mentioned that we used Apollo and GraphQL and that by using Apollo Server were able to quickly build a production-ready, self-documenting API for GraphQL clients, using data from our MongoDB database. So in this section we are going to provide some more knowledge resources about GraphQL and Apollo Server.

But, before describing GraphQL, we should mention that we decided to use GraphQL because as mentioned earlier that between n-tier layers there we should have some well-designed layers that are abstract and there are no ambiguous areas, and more specifically for our use case an ambiguous area is the area where we can't distinguish if some procedures belong to the data-access or they belong the business-logic layer. The Dependency Inversion Principle[43] indicates that:

> A. High-level modules should not depend on low-level modules. Both should depend on abstractions.
> B. Abstractions should not depend on details. Details should depend on abstractions.

Having that in mind the following figure now makes sense.

[43] https://en.wikipedia.org/wiki/Dependency_inversion_principle



Picture 32: Our application's design using best practices (GraphQL for the Business-Logic) [65]

In this Figure we see that we separate the concerns of the backend layer in Node.js using the data access layer and the business logic layer using only mongoose tool by leaving only the data accessing layer to be done mongoose tool and then we convert the data schema returned by mongoose data access to a GraphQL schema so that for now on GraphQL should be responsible for the business logic layer.

GraphQL is an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data. GraphQL was developed internally by Facebook in 2012 before being publicly released in 2015. On 7 November 2018, the GraphQL project was moved from Facebook to the newly-established GraphQL Foundation, hosted by the non-profit Linux Foundation. Since 2012, GraphQL's rise has followed the adoption timeline as set out by Lee Byron, GraphQL's creator, with surprising accuracy. Byron's goal is to make GraphQL omnipresent across web platforms.

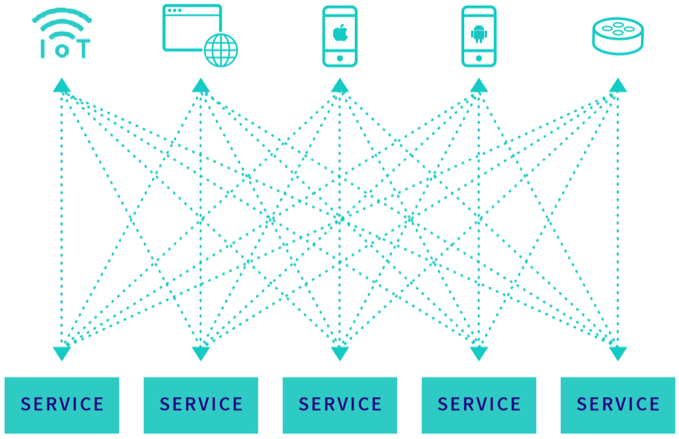
It provides an efficient, powerful and flexible approach to developing web APIs, and has been compared and contrasted with REST and other web service architectures. It allows clients to define the structure of the data required, and exactly the same structure of the data is returned from the server, therefore preventing excessively large amounts of data from being returned, but this has implications for how effective web caching of query results can be. The flexibility and richness of the query language also adds complexity that may not be worthwhile for simple APIs. It consists of a type system, query language and execution semantics, static validation, and type introspection. GraphQL supports reading, writing (mutating) and subscribing to changes to data (realtime updates). [65] [66]

GraphQL isn't just a developer megatrend. It's the smart architectural choice for any team, large or small, that needs to quickly build high-quality apps in the modern environment. And today, users expect high-quality personalized experiences that are available on all of their devices.

Picture 33: Showcase of different platforms and architectures that need to receive data [66]

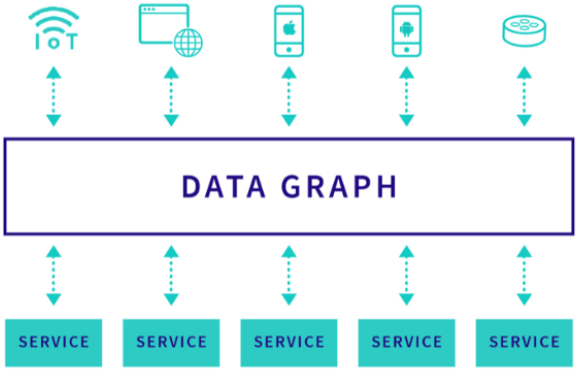
Nowadays, developers need to build these apps on top of an ever-increasing number of data services. How can they manage this development complexity? This is what GraphQL is trying to achieve. The unification of all different policies, protocols and API technologies. [65]

Nowadays, REST services are used to be the state of the art. The point-to-point nature of REST, a procedural API technology, forces the authors of services and clients to coordinate each use case ahead of time. When frontend teams must constantly ask backend teams for new endpoints, often with each new screen in an app, development is dramatically slowed down. Both teams need to move fast independently.



Picture 34: Today situation using REST services [66]

GraphQL is a comprehensive solution to the problem of connecting modern apps to services in the cloud. As such, it forms the basis for a new and important layer in the modern application development stack: the data graph. This new layer brings all of a company's app data and services together in one place, with one consistent, secure, and easy-to-use interface, so that anyone can draw upon it with minimal friction. [65]



Picture 35: Todays' situation using GraphQL [65]

GraphQL decouples apps from services by introducing a flexible query language. Instead of a custom API for each screen, app developers describe the data they need, service developers describe what they can supply, and GraphQL automatically matches the two together. Teams ship faster across more platforms, with new levels of visibility and control over how their data is used. [65]

A GraphQL service is created by defining types and fields on those types, then providing functions for each field on each type. For example, a GraphQL service that tells us who the logged in user is (me) as well as that user's name might look something like this:

```
type Query {
  me: User
}

type User {
  id: ID
```

```
  name: String
}
```

Code 8: GraphQL type definition

Along with functions for each field on each type:

```
function Query_me(request) {
  return request.auth.user;
}

function User_name(user) {
  return user.getName();
}
```

Code 9: GraphQL functions

Once a GraphQL service is running (typically at a URL on a web service), it can be sent GraphQL queries to validate and execute. A received query is first checked to ensure it only refers to the types and fields defined, then runs the provided functions to produce a result.

For example the query:

```
{
  me {
    name
  }
}
```

Code 10: GraphQL query

Could produce the JSON result:

```
{
  "me": {
    "name": "Luke Skywalker"
  }
}
```

Code 11: GraphQL query response

So, some of GraphQL's features using an eagle's eye perspective list as follows:

- **Shipping Faster**

Don't write a ton of code or rely on new rigid API endpoints when there is need to fetch data for a new screen in an app. This is easily shown in our application's presentation layer, as you can find out that we connect our presentation layer to only one endpoint serving the GraphQL data, where each time needed, and in any screen needed we just send to the only one endpoint a different query.

- **Better Apps**

Build features with the best data and services possible, not the API available that day. GraphQL helps you put personalization into every app. This is also easily shown as we dynamically change in our presentation layer the "getPatients" query, each time requesting for more or less data to be provided to us.

- **Parity Across Platforms**

Makes applications consistent across all channels. This happens by moving data-related functionality that is common between platforms into the shared GraphQL layer.

- **Powerful Partner APIs**

Get new partners onto your API without custom changes, while ensuring a high-quality experience for your mutual customer.

- **Visibility Into Your Data**

Gives to development teams real-time visibility into exactly what services are available for them to build on top of. We also provide this in our web interface as a custom page build inside the web-application, but also we have allowed production read visibility to the data using GraphQL Playground[44].

- **Positive Control**

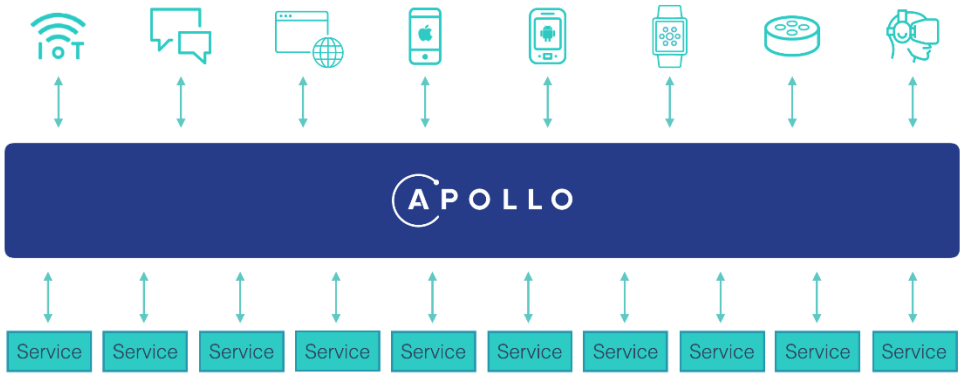
Get a single point of control to secure and analyze all access to the applications' data and see how it's used.

Usage in our web application

In this thesis we implemented GraphQL by using the Apollo Server. As was presented in a previous section, we used the apollo-server package dependency and also created and run an Apollo Server instance with the following line of code:

```
const server = new ApolloServer({ schema: graphqlSchema });
```

Apollo Server is the best way to quickly build a production-ready, self-documenting API for GraphQL clients, using data from MongoDB or any other source, and that is because it implements a spec-compliant GraphQL server which can be queried from any GraphQL client, as we will also refer-to later when describing the presentation layer.



Picture 36: Our web application's situation using GraphQL [67]

Apollo server is open-source and works great as a stand-alone server, an addon to an existing Node.js HTTP server, or in "serverless" environments. So by combining a server with GraphQL specs preinstalled and the graphql-compose set of packages described in the previous section we were ready to easily and in a few lines of code provide a ton of services to the client. [67] [68]

So, our main concern left, in order to finish with the backend coding, is to find a way to translate the MongoDB schema, that we got from the data-access layer, to a GraphQL schema. GraphQL Compose toolkit package, described in the previous section, helped us on doing so, as it offers a graphql-compose-mongoose package which exports a

[44] https://github.com/prisma/graphql-playground

composeWithMongoose function that converts the mongo schema into a graphql schema. [69]

The GraphQL Schema Definition Language (SDL)

A schema definition is the most concise way to define a GraphQL query. To define the schema of a GraphQL API, you have to use the type system provided by GraphQL, and this is where composeWithMongoose helps us, as it is automatically converting the MongoDB schema type and relations definitions to GraphQL type and relation definitions. The syntax for writing schemas is called Schema Definition Language. The main components of a schema definition are the types and their fields. A GraphQL schema describes the types, shapes and relationships in your data and can be written in any programming language that implements the type system.

Types are used to describe the set of possible data you can query from a service. Here's an example of how to define a type in GraphQL:

```
type User {
  name: String!
  username: String!
}
```

Code 12: Example User graphql type definition

The *User* type we just described is a GraphQL Object Type which means that it is a type that contains some fields. This "*User"* type has two fields: "*name"* and "*username"*, and they're both of type *String*. The *!* following the types indicates that the field is required (non-nullable). In this application's use case, one part of the MongoDB schema given in an earlier section describing MongoDB, would convert using graqhl-compose-mongoose from:

```
"patient": {
...
   "age": {
      "type": "number",
      "required": false
    },
    "biopsy": {
      "type": "string",
      "required": true
    },
...
}
```

Code 13: mongodb code before graphql-compose convertion

To:

```
Type Patient {
...
    age: Number,
    biopsy: String!,
...
}
```

Code 14: graphql code after graphql-compose convertion

In GraphQL, you can express **relationships** between types, too. This means that the Type Patient that we created in the above code can be used as a type definition of another type's field, for example Hospital.

```
Type Hospital {
...
   patients: [Patient!]!,
...
```

Code 15: graphql type definition using complex types

In our application's case we didn't need such a case at the moment, as the collection we have in the database is only one, but in future possibly if we have more data about the patients we could easily find such scenarios. So finally, after using the graphql-compose-mongoose method we managed to translate the MongoDB schema to a GraphQL schema with its GraphQL Object Type definitions[45]. Having the GraphQL schema ready for use, the next step is to append it by creating the Queries and Mutations that will be needed from the presentation layer.

Queries are used by the client to request the data it needs from the server. Unlike REST APIs where there's a clearly defined structure of information returned from each endpoint, GraphQL always exposes only one endpoint, allowing the client to decide what data it really needs from a predefined pattern. Here's an example based on the previous "User" type definition:

```
{
  Users {
    name
  }
}
```

Code 16: Example graphql query definition

The "Users" field in our query above is called the root field of the query. Anything that comes after the root field is known as the payload. Since we only added name in the payload, the query will return a list of all the users like this:

```
{
  "Users": [
    {"name": "Nikolas"},
    {"name": "Ema"},
    {"name": "Styliani"},
    {"name": "Pavlos"},
  ]
}
```

Code 17: Example graphql query response after execution

You'll notice that the query only returned the names of the users; this is because in the query payload, we specified that we need only the name of each user. This capability is described by GraphQL as ask for what you need and you'll get exactly that. Now assuming we need more information about the users, we can decide to add username to the payload:

[45] https://medium.com/software-insight/graphql-types-and-relationships-cbb046a541c4

```
{
  Users {
    name
    username
  }
}
```

Code 18: Example graphql query definition with more fields

Now the server will include the username of each user in its response:

```
{
  "Users": [
    {"name": "Nikolas", "username": "nmpegetis"},
    {"name": "Ema", "username": "eanastasiadou"},
    {"name": "Styliani", "username": "sgeorgiou"},
    {"name": "Pavlos", "username": "pkafouris"},
  ]
}
```

Code 19: Example graphql query response after execution with more fields

This way we have more flexibility over the data we receive from the server, which is immensely valuable for many reasons. For example, assuming we want to request the server to send information of only the last two users who were added, we can use arguments to specify it like this:

```
{
  Users(last: 2) {
      name
      username
    }
  }
}
```

Code 20: Example graphql query definition with filter

Note: Zero or more arguments (like last: 2) can be provided for each field in GraphQL. Each argument passed to any field must be defined in the schema.

Thus the server will only return:

```
{
  "Users": [
    {"name": "Styliani", "username": "sgeorgiou"},
    {"name": "Pavlos", "username": "pkafouris"},
  ]
}
```

Code 21: Example graphql query response after execution with filter

Also assuming we have defined an argument in our schema for getting a single User from its username, we can pass in the username as an argument, like this:

```
{
  Users(username: “nmpegetis”) {
      name
      username
    }
```

```
  }
}
```

Code 22: Example graphql query definition with find

As expected, this query returns the user whose username matches our argument:

```
{
  "Users": [
    {"name": "Nikolas", "username": "nmpegetis"},
  ]
}
```

Code 23: Example graphql query response after execution with find

Passing arguments the way we just did is helpful, but in most instances, the arguments needs to be dynamic. GraphQL provides a way for us to pass in dynamic values as arguments—by treating them as variables

Variables are used to factor dynamic values out of the query and pass them as a separate dictionary. For example, if we have a search field where a user types in the name of patient he or she wants to view. To allow that, we'd use variables like this: [70]

```
query Patients($namevar: String) {
  patient(name: $namevar) {
    name
    username
  }
}
```

Code 24: Example graphql query definition with variables

In the example above "($name: String)" is the variable definition. "namevar" is the variable name and it is prefixed by "$", followed by the type in this case "String". That's it. This is all, we used about queries in our web application's code.

Finally, to sum up and present our implementation code that handles all the above scenarios we just have to mention Mutations**.**

So, so far we have seen the way to retrieve data from the server using queries, but how do we create, update and delete data in GraphQL? The answer is with Mutations. Mutations in GraphQL are used to CUD:

- Create new data
- Update existing data
- Delete existing data

The syntax for mutations look almost the same as queries, but they must start with the "mutation" keyword. Here's an example that'll allow us create a new user:

```
mutation {
  createUser(name: "Lefteris", username: "louzounoglou") {
    name
    username
  }
}
```

Code 25: Example graphql create mutation

Notice that the mutation has a root field and payload, just like a query. There are also two arguments passed to the mutation: "name" and "username". To be able to manage the

data of our users effectively, we need to assign a unique identity to each user. GraphQL has a type ID that can be attached to a field. When that is done, the server will automatically generate a unique id for each new object (user).

Recall from the previous article that we created a GraphQL type called User, and now we can update it to include an id field as follows:

```
type User {
  ID: ID!
  name: String!
  username: String!
}
```

Code 26: Example User graphql type definition with ID

Now we can update the mutation by passing only the id to the payload. That way the server will return the id of the user we just created. [71]

```
mutation {
  createUser(name: "Lefteris", username: "louzounoglou") {
    id
  }
}
```

Code 27: Example graphql create mutation optimization

Subscriptions, is an improvement for future work. Subscriptions, are a way to create and maintain real time connection to the server. This enables the client to get immediate information about related events. Basically, a client subscribes to an event in the server, and whenever that event is called, the server will send the corresponding data to the client. Subscriptions are event based[46].

Let's say a hypothetical user profile app allows users to change their username on their profile and the displayed username needs to be updated in real time. This means:

- For the initial page load, we need to fetch the displayed username using GraphQL queries.
- Next, we'll listen for the new username and get it through a subscription.
- We update the UI with a notification that the username change was complete.

We can also create a subscription that notifies users for deletions, etc. in their profile. In the previous example, we will want to get updates from the server whenever a new user is created that way we can update the client automatically without having to make a call for it. Here is an example subscription:

```
subscription {
  newUser {
    name
    username
  }
}
```

Code 28: Example graphql subscription

---

46 https://medium.com/fbdevclagos/understanding-graphql-queries-mutations-and-subscriptions-a80a8b5c877c

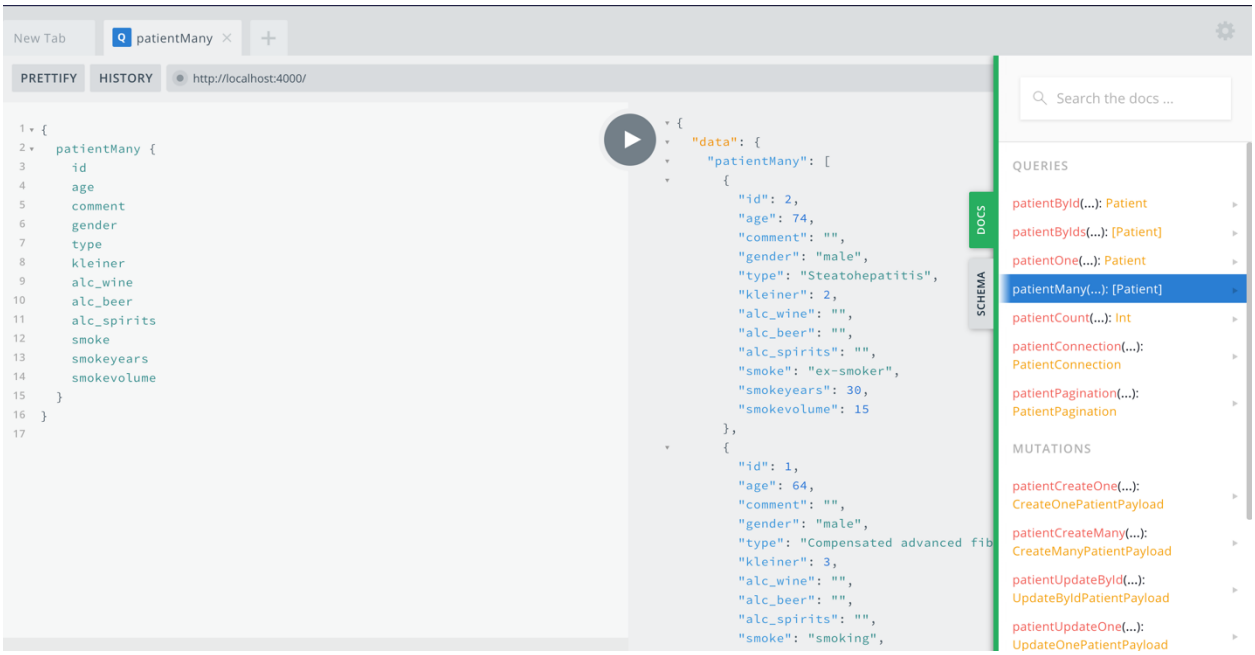
That's all folks about coding GraphQL. Now let's finally see how we created our queries and mutations for our web application.

```
// Project ALD-BRFAA-API
// Filepath: src/models/patients

import schema from '../../schema.json';
import mongoose from 'mongoose';
import { composeWithMongoose } from 'graphql-compose-mongoose';
import { schemaComposer } from 'graphql-compose';
const Schema = mongoose.Schema;

// STEP 1: DEFINE MONGOOSE SCHEMA AND MODEL
const patientSchema = new Schema(schema.patients);
const Patient = mongoose.model('Patient', patientSchema);

// STEP 2: CONVERT MONGOOSE MODEL TO GraphQL PIECES
const customizationOptions = {}; // left it empty for simplicity, described below
const PatientTC = composeWithMongoose(Patient, customizationOptions);

// STEP 3: Add needed CRUD Patient operations to the GraphQL Schema
// via graphql-compose it will be much much easier, with less typing
schemaComposer.Query.addFields({
  patientById: PatientTC.getResolver('findById'),
  patientByIds: PatientTC.getResolver('findByIds'),
  patientOne: PatientTC.getResolver('findOne'),
  patientMany: PatientTC.getResolver('findMany'),
  patientCount: PatientTC.getResolver('count'),
  patientConnection: PatientTC.getResolver('connection'),
  patientPagination: PatientTC.getResolver('pagination'),
});
schemaComposer.Mutation.addFields({
  patientCreateOne: PatientTC.getResolver('createOne'),
  patientCreateMany: PatientTC.getResolver('createMany'),
  patientUpdateById: PatientTC.getResolver('updateById'),
  patientUpdateOne: PatientTC.getResolver('updateOne'),
  patientUpdateMany: PatientTC.getResolver('updateMany'),
  patientRemoveById: PatientTC.getResolver('removeById'),
  patientRemoveOne: PatientTC.getResolver('removeOne'),
  patientRemoveMany: PatientTC.getResolver('removeMany'),
});

const graphqlSchema = schemaComposer.buildSchema();
export default graphqlSchema;
```

Code 29: Create graphql queries and mutations using graphql-compose

The above figure shows how simply we managed to convert our mongoose schema to a graphql schema and let GraphQL manage the business logic of our application. To do so, we used the "getResolver" graphql-compose's method on a GraphQL Object Type Composer to create the relative Queries and Mutation such as those of MongoDB, and finally we built all this GraphQL schema using the method buildSchema() of the graphql-compose schemaComposer[47].

The final result of all this backend configuration for connecting with the MongoDB database, then converting the data to GraphQL and finally serving a playground for user access to data, where the user can handle with ease queries and mutations can be found in the following Picture.



Picture 37: The GraphQ playground for queries and mutations

4.3.3.3 React[48]

React (also known as React.js or ReactJS) is an open-source JavaScript library for building user interfaces specifically for single page applications. It is used for handling view layer for web and mobile apps, while also allows us to create reusable UI components. React was first created by Jordan Walke, a software engineer working for Facebook, and it was first deployed on Facebook's newsfeed in 2011 and on Instagram.com in 2012. It is now maintained by Facebook and a community of individual developers and companies.

React can be used to create large web / mobile / AR etc. applications, as it is optimal for fetching rapidly changing data that needs to be rendered, without reloading the page. So, it works only on user interfaces in application, which corresponds to view in the MVC template, and its main purpose is to be fast, scalable, and simple. However, fetching data is only the beginning of what happens on a web page, which is why complex React applications usually require the use of additional libraries for state management, routing, and interaction with an API, and for this reason it is usually used with a combination of

[47] https://github.com/graphql-compose/graphql-compose-mongoose

[48] https://reactjs.org/

other JavaScript libraries or frameworks. We refer below some of ReactJS commonly used features.

JSX

In React, instead of using regular JavaScript for templating, it uses JSX. JSX is simple JavaScript which allows HTML quoting and uses these HTML tag syntax to render subcomponents, then the HTML syntax is processed into JavaScript calls of the React library. Nevertheless, this feature in not obligatory and one can also use HTML for creating React components in Vanilla JavaScript.

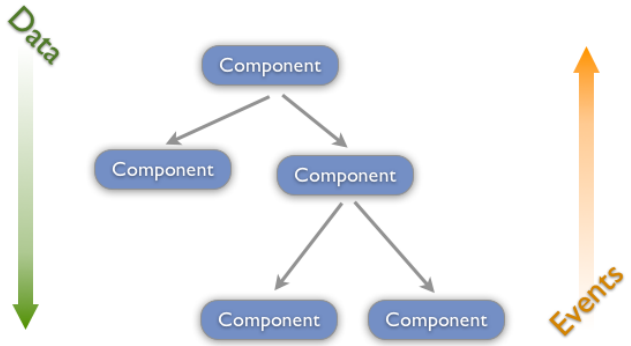
```
const Example = () => (
  <Parent>
    <Child/>
  </Parent>
)
```

```
React.createElement(
  Parent,
  null,
  React.createElement(Child, null)
);
```

Picture 38: Create React Element with JSX or with Vanilla JS [74]

Single-Way data flow

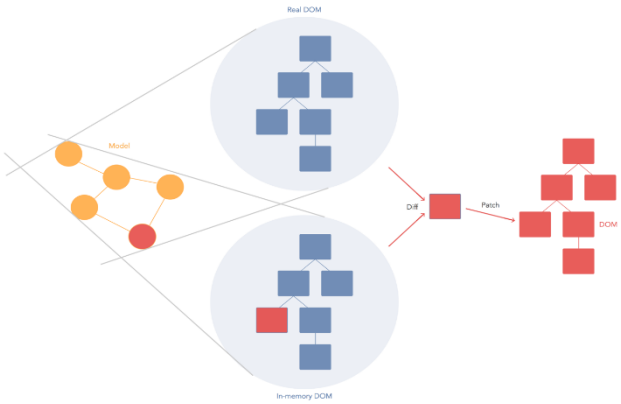
In React, a set of immutable values are passed to the components renderer as properties in its HTML tags. Component cannot directly modify any properties but can pass a callback function with help of which we can do modifications. This complete process is known as "properties flow down; actions flow up".



Picture 39: React props and action event-handlers [74]

Virtual Document Object Model

React creates an in-memory data structure cache, the Virtual DOM, which is a copy of the browser's DOM. In every event change React computes the changes made on the Virtual-DOM and then updates the only the part of DOM that needs to be changed without changing all the DOM. This way with a little more space complexity React manages to handle much better the time of rendering complexity.



Picture 40: react Virtual DOM usage [74]

And now, having referred some of most interesting React features that are commonly used, the main question that arises is why one should use ReactJS. There are so many open-source platforms for making the front-end web application development easier, like Angular, or VueJS, and more. Some of the benefits of React over other competitive technologies or frameworks, are presented next:

Simplicity

ReactJS is just simpler to grasp right away. The component-based approach, well-defined lifecycle, and use of just plain JavaScript make React very simple to learn, build a professional web (and mobile applications), and support it. Especially by using JSX, React allows the developer to easily mix HTML with JavaScript

Easy to learn

Anyone with a basic previous knowledge in web-programming can easily understand React while other libraries or frameworks like Angular and Ember are referred to as 'Domain specific Language', which implies that they are more difficult to learn, and that to dive into them one needs to study the framework first, and can't start using it with no prior knowledge.

Native Approach

React can be used to create mobile applications (React Native). And React is a diehard fan of reusability, meaning extensive code reusability is supported. So at the same time one can make iOS, Android and with small amount of changes a web-application.

Data Binding

React uses one-way data binding and an application architecture called Flux controls the flow of data to components through one control point – the dispatcher. One-way binding in comparison to two-way binding makes debugging easier at self-contained components of large ReactJS apps, but on the other hand need more coding.

Un-opinionated

React does not offer any concept of a built-in container for dependency. One can use whichever module wants to accompany React's building process, transpilation, etc.

Testability

ReactJS applications are very easy to test. React views can be treated as functions of the state, and because state manipulation is easily handled one can fast get the output by triggering actions, events, functions, etc. [74] [75] [76]

Usage in our web application

In this thesis as mentioned also in the Node.js section, we implemented the whole web-application in two different code projects, one for the frontend and one for the backend. As we described in previous sections the data persistence layer, the data access layer and the business logic layer are commonly all of them parts of the so called "backend". So what is left for the "frontend" is the presentation layer. In general, some years before, handling data in the frontend was not recommended and also sometimes compelling as at that time using JavaScript was said: "it is like adding a time bomb on one's application". Nowadays and after Angular, React, etc. arrival, while also because of the mobile era and IoT and in general the different devices usage for the presentation layer this has changed. Devices and their browsers are much more powerful and resourceful and so a lot of things that previously were supposed as a backend-concern, now they are a frontend concern. As a matter of fact, in our web-application project we left on the backend only the data access and the things that were truly having to do with the business-logic layer. All the rest, and the biggest part of our development was transferred to the frontend.

So, the tasks we had to implement in the front-end using React and JavaScript were (we also provide some small implementation snippets):

1. Manipulate the data from GraphQL Queries and Mutations

In the following code snippet we show an example of query creation for adding to Query ApolloClient React component that helps in connecting the presentation layer with the business logic layer. We use this Query component inside another parent React component named Layout

```
// Project ALD-BRFAA-UI

// Filepath: src/layouts/layout.js

...

class Layout extends React.Component {
  constructor(props) {
    super(props);
    const labelKeys = Object.keys(props.labels).slice(0, 15);

    const query = `
    {
      patientMany {
        ${labelKeys.join().replace(/,/g, `\n`)}
      }
    }
  `;

...

  render() {
    const { classes, history } = this.props;
    const { query } = this.state;
    const graphQLQuery = gql`${query}`;

    return (
      <div className={classes.root}>
        <CssBaseline />
        <Header open={this.state.open} onDrawerOpen={this.handleDrawerOpen} history={history} />
        <Drawer open={this.state.open} onDrawerClose={this.handleDrawerClose} />
        <main className={classes.content}>
          <div className={classes.appBarSpacer} />
          <Query query={graphQLQuery}>
            {({ loading, error, data, subscribeToMore }) => {
              if (error) return <div> Error {error.message} </div>;
              return <Switch>{this.renderRoutes(data, loading)}</Switch>;
            }}
          </Query>,
        </main>
      </div>
    );
  }
}

...
```

Code 30: Create a graphql query from React Application

2. Create different SPA (Single Page Application) routes in respect to the functionality we want to provide to the users

We decided to distinguish routes based on every different business functionality, meaning that we created pages based on different business use cases, i.e. a page for visible application data manipulation, a page to visualize data, a page to store data, etc.. In the following code snippet we show an example of a React menu navigation creation, using React Icons and a List made of React ListItems that link to different routes.

```
// Project ALD-BRFAA-UI

// Filepath: src/layouts/menu.js

...
```

```
export const mainMenuEntries = {
  Dashboard: { link: '/', renderIcon: () => <DashboardIcon /> },
  Query: { link: '/query', renderIcon: () => <QueryIcon /> },
  Analysis: { link: '/analysis', renderIcon: () => <LayersIcon /> },
  Reports: { link: reportsLink, renderIcon: () => <BarChartIcon /> },
};

export const mainListItems = Object.keys(mainMenuEntries).map((entry) => (
  <Link to={mainMenuEntries[entry].link} style={{ textDecoration: 'none' }}>
    <ListItem button>
      <ListItemIcon>{mainMenuEntries[entry].renderIcon()}</ListItemIcon>
      <ListItemText primary={entry} />
    </ListItem>
  </Link>
));


...
```

Code 31: Create a menu entries for different applicstion routes

3. Create manageable widgets for use (Buttons, Tables, Fields, etc.)

In the following code snippet we show an example of a reusable React Table widget creation that wraps a React Table component from the antd package library.

```
// Project ALD-BRFAA-UI
// Filepath: src/components/Table/TableView.js

...

export class TableContainer extends React.Component {
  render() {
    const {
      data,
      classes,
      onDataChange,
      labelKeys,
      labels,
      loading,
      onTitleChange,
      title,
      isTitleEditable,
    } = this.props;
    return (
      <TableView
        data={data}
        classes={classes}
        onDataChange={onDataChange}
        labelKeys={labelKeys}
        labels={labels}
        loading={loading}
        onTitleChange={onTitleChange}
        title={title}
        isTitleEditable={isTitleEditable || false}
      />
    );
  }
}

...
```

Code 32: React reusable Table component

4. Transform data from GraphQL to charts and graphs x-y coordinates

In the following code snippet we show an example of some methods that handle transformations needed for applying the GraphQL data in graphs, diagrams and chart visualizations.

```
// Project ALD-BRFAA-UI
// Filepath: src/routes/DashBoardView.js

...
```

```
transformDataToXY = (data) => {
  const { xAxis, yAxis } = this.state;

  const vizdata = data.map((e, i) => ({
    x: e[xAxis],
    y: e[yAxis],
    xAxisName: xAxis,
    yAxisName: yAxis,
  }));

  return vizdata;
};

f = (a, b) => [].concat(...a.map(d => b.map(e => [].concat(d, e))));
cartesian = (a, b, ...c) => (b ? this.cartesian(this.f(a, b), ...c) : a);

getGroupArrays = (data) => {
  const { groupByVar1, xAxis, groupByVar2, groupByVar3} = this.state;
  const groupUniqueCategories = [ ...new Set(data.map((e, i) => (
        !!groupByVar1 ?
        e[groupByVar1] :
        e[xAxis]
      ))) ];

  const output = !!groupByVar3 ? this.cartesian(groupUniqueCategories,
      groupUniqueCategories2, groupUniqueCategories3) : !!groupByVar2 ?
          this.cartesian(groupUniqueCategories, groupUniqueCategories2) :
          groupUniqueCategories;

...

  return groupArrays;
};

transformDataToAngles = (data) => {
  const groupArrays = this.getGroupArrays(data);

  const dataCategories = groupArrays.map((e, i) => ({
    label: groupArrays[i].key,
    angle: Math.floor(groupArrays[i].value.length * 100 / data.length),
  }));

  return dataCategories;
};

computeStandardDeviation = (array) => {
  const len = array.length;
  const mean = array.reduce((a, b) => a + b) / len;
  const sd = Math.sqrt(array.map((x) => Math.pow(x - mean, 2)).reduce((a, b) => a + b) / len);
  return sd;
};

calculateDataProps = (data) => {
  const groupData = this.transformDataToXY(data);
  const arrayOfx = groupData.map((obj) => obj.x).filter((num) => !isNaN(num));
  const arrayOfy = groupData.map((obj) => obj.y).filter((num) => !isNaN(num));
  const minx = Math.min(...arrayOfx);
  const miny = Math.min(...arrayOfy);
  const maxx = Math.max(...arrayOfx);
  const maxy = Math.max(...arrayOfy);
  const meanx = (maxx + minx) / 2;
  const meany = (maxy + miny) / 2;
  const sdx = arrayOfx.length > 0 && this.computeStandardDeviation(arrayOfx);
  const sdy = arrayOfy.length > 0 && this.computeStandardDeviation(arrayOfy);
  const xVariance = maxx - minx;
  const yVariance = maxy - miny;
  const size = groupData.length;

  return groupData.map((o) => ({
    ...o,
    xVariance: xVariance + 0.1,
    yVariance: yVariance + 0.1,
    minx,
    maxx,
    miny,
```

```
    maxy,
    meanx,
    meany,
    sdx,
    sdy,
    size,
  }));
};

transformGroupData = (data) => this.getGroupArrays(data).map((array) => this.calculateDataProps(array.value));

...
```

Code 33: Math methods used for data transformation to apply in visualizations

5. Use Visualizations

In the following code snippet we create a LineGraph visualization using React package react-vis's LineSeries.

```
// Project ALD-BRFAA-UI

// Filepath: src/components/Charts/LineGraph/LineGraphView.js



...

const LineGraphView = ({
  data,
  groupedData,
  groupedCategories,
  ...
}) => {

...

  let name = 'lineGraph';

  const categories = (groupedCategories && groupedCategories.map((category) => category.label)) || [];
  return (
    <React.Fragment>
...
      <div id={name}>
        <FlexibleXYPlot
          xType={showXQualitative ? 'ordinal' : undefined}
          yType={showYQualitative ? 'ordinal' : undefined}
          animation
          height={!isFullScreen ? 300 : height * ratio}
        >
          {!showGrouped ? (
            <LineSeries data={data} />
          ) : (
            (groupedData || []).map((groupData) => <LineSeries data={groupData} />)
          )}
          <VerticalGridLines />
          <HorizontalGridLines />
          <XAxis title={data[0] && data[0].xAxisName} tickLabelAngle={-45} />
          <YAxis title={data[0] && data[0].yAxisName} />
        </FlexibleXYPlot>
      </div>
    </React.Fragment>
  );
}
);
...
```

Code 34: Wrap a react-vis LineSeries element and create a new with more capabilities

6. Provide access to backend and store in cache previously queried queries

In the following code snippet we create an Analysis route where we plug the GraphQL playground as a React component with all its functionality, one of which is also storing the already applied queries.

```
// Project ALD-BRFAA-UI

// Filepath: src/routes/Analysis/AnalysisView.js
```

```
...

const AnalysisView = (props) => (
   <React.Fragment>
     <Provider store={store}>
       <Playground
         subscriptionEndpoint="http://localhost:4000/query"
         createApolloLink={(session, subscriptionEndpoint) => {
           return { link: link };
         }}
         shareEnabled={true}
         canSaveConfig={true}
       />
     </Provider>
   </React.Fragment>
 );
...
```

Code 35: add Graphql playground as a React component in Analysis route

Packages

In this thesis as mentioned also in the Node.js section, using packages already tested and published by their owners is very common in application development, as all software developers want to finish their work the fastest they can and don't want to reinvent the wheel. But, we have to keep in mind always the impediments of Dependency Hell[49]. And so we also did for the packages we used for this application's frontend. The packages list as follow:

1. Material-UI Group of packages[50]

Material-UI is an open-source project that features React components that implement Google's Material Design. It kick-started in 2014, not long after React came out to the public, and has grown in popularity ever since. With over 48,000 stars on GitHub, Material-UI, now in v.4, is one of the top user interface libraries for React. One of Material-UI features that makes it stand out, and also reason for us to select it for this web-application is its responsive design for all types of screens, like Google Material Design demands. In specific, we used Material-UI in all visual components of our Web-UI, i.e. all Buttons, Menus, Lists, Headers, Fields, DropDownLists, Icons, and ButtonIcons of the user interface are created with Material-UI's components, except for the Table component.

2. Ant Design[51]

Ant Design is also an open-source project that features React components but specially created for internal desktop applications. We selected to add this package alongside Material-UI because it has a very good API and it's very easy to use. More specifically, the Table component that it provides the developer with many handy functionalities, much more than the relevant provided by Material-UI. What is more, It has an equal reputation to Material-UI with over 47,000 starts on GitHub.

3. Apollo Group of packages[52]

Apollo Client is the best way to use GraphQL to build client applications. The client is designed to help the developer quickly build a UI that fetches data with GraphQL, and can be used with any JavaScript front-end. More specifically, we used the Query React

---

[49] https://en.wikipedia.org/wiki/Dependency_hell

[50] https://material-ui.com/

[51] https://ant.design

[52] https://www.apollographql.com/docs/react/

component that provides us with a way to link the presentation layer to the GraphQL business-logic layer, while also all the other packages that provide us, linking protocols, memory caching, etc.

4. **Axios**[53]

Axios is a JavaScript library used to make HTTP requests from node.js or XMLHttpRequests from the browser that also supports the ES6 Promise API. In this thesis, we used axios to help this web application get more unaware for the provided data. One of the main concerns that we had from the beginning of the project setup is to build with project and make it context unaware, meaning that the web-tool can work with any kind of data, providing only two independent variables. First, the dataset can change in MongoDB and so with only a script execution we can import the new dataset to MongoDB and then extract the new schema and provide it from the persistence layer to the data access layer, to the business logic layer, to the presentation layer, automatically. Second, in the presentation layer we have got a user-friendly display of all columns that if willing the user can create for helping himself and upload it to the user-interface for use. For example there is a column "kleiner" which in the web application labels was labeled as "NAS fibrosis subscore" which is very helpful, especially for users that don't have to do with the dataset creation. This is where axios is needed for. Axios is used in the HTTP "fetch labels" request to upload labels in the UI. Of course in cases of no labels existence the user interface works with datasets column keys.

5. **React-router**[54]

React Router is a collection of navigational components that compose declaratively with the application. As we already have indicated this thesis's application is a Single Page Application (SPA) which means that it really exists in only one route URL. But as we need for this application to have more than one routes, and have options like having bookmarkable URLs for the web app, this is where React Router navigational components help us solve this impediment.

6. **GraphQL playground React**[55]

GraphQL Playground uses components of GraphiQL (provided by GraphQL) under the hood but is meant as a more powerful GraphQL IDE enabling better development workflows. Compared to GraphiQL, the GraphQL Playground ships with the additional features: Interactive, multi-column schema documentation, Automatic schema reloading, Support for GraphQL Subscriptions, Query history, Configuration of HTTP headers and Tabs. In our application such a case loses the developer's hands as by importing this React Component, we solve the aware of saving the queries already requested, the multi-query user's tasking, and queries caching issues. Nevertheless, in future we can store these queries also to a previous tier layer, like in business-logic, in order that caching is also done in the backend and provided to all connected frontends.

7. **Html2Canvas**[56]

[53] https://github.com/axios/axios

[54] https://reacttraining.com/react-router/web/guides/quick-start

[55] https://github.com/prisma/graphql-playground

[56] https://html2canvas.hertzen.com/

Html2Canvas is a helper package that provided to our application the ability to easily handle the image convertion from .svg format to .png format and download.

```
// Project ALD-BRFAA-UI
// Filepath: package.json

{
  "name": "adl-brfaa-ui",
  "version": "0.1.0",
  "keywords": [
    "reactjs",
    "apollo",
    "graphql",
    "react-vjs",
    "visualization"
  ],
  "author": "nmpegetis@gmail.com",
  "dependencies": {
    "@material-ui/core": "^4.0.0-rc.0",
    "@material-ui/icons": "^3.0.2",
    "antd": "^3.16.1",
    "apollo-cache-inmemory": "^1.4.2",
    "apollo-client": "^2.4.12",
    "apollo-link": "^1.2.11",
    "apollo-link-http": "^1.5.14",
    "apollo-utilities": "^1.3.0",
    "autosuggest-highlight": "^3.1.1",
    "axios": "^0.19.0",
    "connected-react-router": "^6.2.2",
    "graphql": "^14.1.1",
    "graphql-playground-react": "^1.7.20",
    "graphql-tag": "^2.10.1",
    "html2canvas": "^1.0.0-rc.3",
    "prop-types": "^15.6.2",
    "react": "^16.8.6",
    "react-apollo": "^2.4.0",
    "react-autosuggest": "^9.4.3",
    "react-dom": "^16.8.6",
    "react-highlight-words": "^0.16.0",
    "react-input-autosize": "^2.2.1",
    "react-redux": "^5.0.6",
    "react-router-dom": "^4.3.1",
    "react-scripts": "2.1.3",
    "react-vis": "^1.11.6",
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
  },
  "devDependencies": {
    "css-loader": "^2.1.0"
  }
}
```

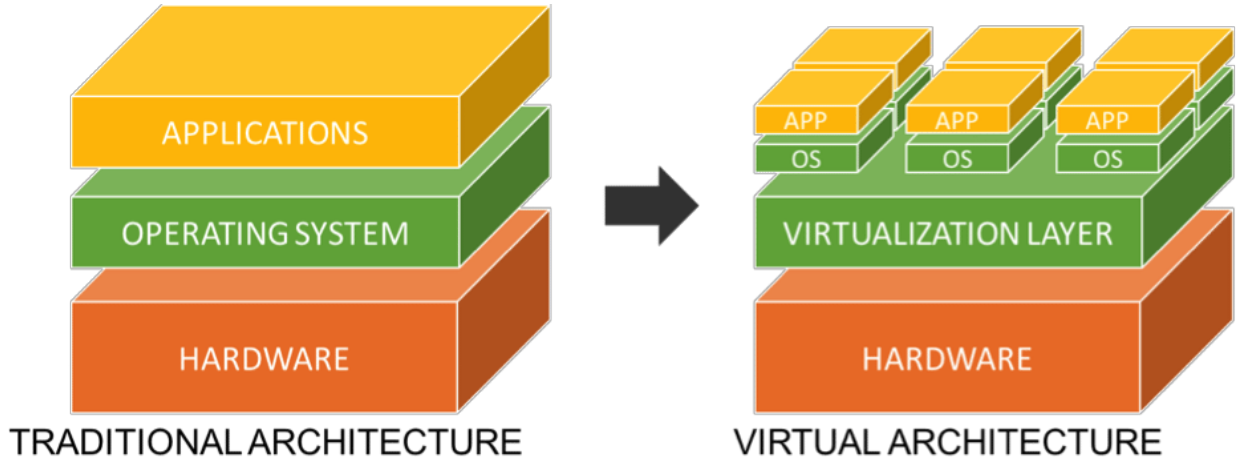
Code 36: The package.json showing the fronted dependencies

4.3.4 Virtualization

In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources. Virtualization began in the 1960s, as a method of logically dividing the system resources provided by mainframe computers between different applications. Since then, the meaning of the term has broadened. One type of virtualization is desktop virtualization, which is the concept of separating the logical desktop from the physical machine. And one form of desktop virtualization, is virtual desktop infrastructure (VDI), which can be thought of as a more advanced form of hardware virtualization, in which the user rather than interacting with a host computer directly via a keyboard, mouse, and monitor, interacts with the host computer and uses another desktop computer or a mobile device by means of a network connection, such

as a LAN, Wireless LAN or even the Internet. In addition to that, the host computer in this scenario becomes a server computer capable of hosting multiple virtual machines at the same time for multiple users.

The last decades, organizations continue to virtualize and converge their data center environment. That effects in the continuation of evolution of client architectures in order to take advantage of the predictability, continuity, and quality of service delivered by their converged infrastructure. For example, companies like HP and IBM provide a hybrid VDI model with a range of virtualization software and delivery models to improve upon the limitations of distributed client computing. Selected client environments move workloads from PCs and other devices to data center servers, creating well-managed virtual clients, with applications and client operating environments hosted on servers and storage in the data center. For users, this means they can access their desktop from any location, without being tied to a single client device. Since the resources are centralized, users moving between work locations can still access the same client environment with their applications and data. For IT administrators, this means a more centralized, efficient client environment that is easier to maintain and able to more quickly respond to the changing needs of the user and business.



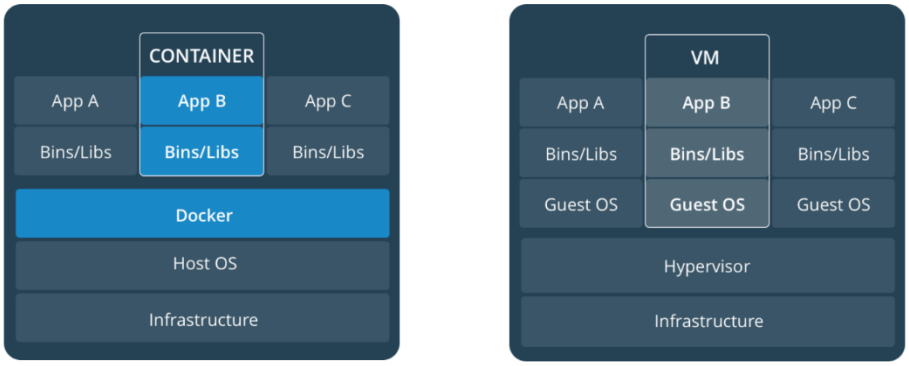
Picture 41: Virtualization image, Virtual Desktop Infrastructure (VDI) [77]

Nowadays, moving virtualized desktops into the cloud creates hosted virtual desktops (HVDs), in which the desktop images are centrally managed and maintained by a specialist hosting firm. Benefits include scalability and the reduction of capital expenditure, which is replaced by a monthly operational cost. Operating-system-level virtualization, also known as containerization, refers to an operating system feature in which the kernel allows the existence of multiple isolated user-space instances. Such instances, called containers, partitions, virtual environments (VEs) or jails, may look like real computers from the point of view of programs running in them. A computer program running on an ordinary operating system can see all resources (connected devices, files and folders, network shares, CPU power, quantifiable hardware capabilities) of that computer. However, programs running inside a container can only see the container's contents and devices assigned to the container. Containerization started gaining prominence in 2014, with the introduction of Docker. [77]

Docker is a set of coupled software-as-a-service and platform-as-a-service products that use operating-system-level virtualization to develop and deliver software in packages called containers. The software that hosts the containers is called **Docker Engine**. It was first started in 2013 and is developed by Docker, Inc. The service has both free and premium tiers.

Containers are isolated from each other and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines. Containers are created from "images" that specify their precise contents. Images are often created by combining and modifying standard images downloaded from public repositories.

Docker is developed primarily for Linux, where it uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable file system such as OverlayFS and others to allow independent containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines (VMs). The Linux kernel's support for namespaces mostly isolates an application's view of the operating environment, including process trees, network, user IDs and mounted file systems, while the kernel's cgroups provide resource limiting for memory and CPU. Since version 0.9, Docker includes the "libcontainer" library as its own way to directly use virtualization facilities provided by the Linux kernel, in addition to using abstracted virtualization interfaces via libvirt, LXC and systemd-nspawn.
Building on top of facilities provided by the Linux kernel (primarily cgroups and namespaces), a Docker container, unlike a virtual machine, does not require or include a separate operating system. Instead, it relies on the kernel's functionality and uses resource isolation for CPU and memory, and separate namespaces to isolate the application's view of the operating system. Docker accesses the Linux kernel's virtualization features either directly using the libcontainer library, which is available as of Docker 0.9, or indirectly via libvirt, LXC (Linux Containes) or systemd-nspawn. [78]



Picture 42: Containerized architecture using Docker vs common VM architecture [78]

4.4 Design Patterns

In this section we present the coding design patterns and the coding principles that we use to follow when developing software applications. The contents of this sections are described as follow. We start by describing the software directory file structure, and then we continues with the design patterns and principles as applied to software development in React, by focusing on the scalability, maintenance, reusability and performance and providing for each principle some relative React code snippets.

4.4.1 Software Directory File structure

Software architects always mind except for the software details that need to be handled, they consider also of business organization structures, users point of view about the application, possible future trends, developments tools used, better data modeling, optimization and performance, better problem to be solved definition etc. These qualities and more are what makes them stand out among other senior developers.

In this section we want to define and describe what is a good software directory structure and why the proposed structure that we are used in this thesis application is a good structure.
All companies, even the best ones struggle with this exact problem. The problem that a lot of business projects where their projects' directory structure philosophies were not thought out carefully enough kept being changed through the project lifecycle again and again, resulting in huge causes of unnecessary problems. Such changes inadvertently generate huge volumes of work, and of course this happens because a lot of times projects are in the rush and under deadlines pressure, resulting in considering the project structure as a low priority.

But, there is no global solution to this project structure definition. Software project directory structure should neither be too complex nor be too simple and this has to do with the size of the project, the developers working in the project, the users of the project and the scope of value that this software project gives to the business.[57]

File Directory Organization

There is a very fine line in balancing the understanding of project members about the structure of the file system with the complexity of the file system you deploy. If too complex, they won't understand it or use it properly. If too simple, each person will begin inventing their own more complex version.

```
ALD-BRFAA-UI/
│
├── public/
│   ├── favicon.ico
│   ├── fonts.css
│   ├── index.html
│   ├── labels.json
│   └── manifest.json
│
├── src/
│   ├── layouts/
│   │   │
│   │   ├── Drawer.js
│   │   ├── Header.js
│   │   ├── index.js
│   │   ├── Layout.js
│   │   ├── menu.js
│   │   └── styles.js
│   │
│   ├── routes/
│   │   │
│   │   ├── Analysis/
│   │   │   ├── AnalysisView.js
│   │   │   ├── styles.js
│   │   │   └── index.js
│   │   │
│   │   ├── DashBoard/
│   │   │   ├── DashBoardView.js
│   │   │   ├── styles.js
│   │   │   └── index.js
│   │   │
│   │   └── Query/
```

[57] https://www.cmcrossroads.com/article/importance-directory-structure-development-process

```
│   │   ├── QueryView.js
│   │   ├── styles.js
│   │   └── index.js
│   │
│   ├── components/
│   │   │
│   │   ├── Autocomplete/
│   │   │   ├── AutocompleteView.js
│   │   │   ├── styles.js
│   │   │   └── index.js
│   │   │
│   │   ├── Charts/
│   │   │   ├── BarPlot
│   │   │   │   ├── BarPlotView.js
│   │   │   │   └── index.js
│   │   │   ├── BubbleChart
│   │   │   │   ├── BubbleChartView.js
│   │   │   │   └── index.js
│   │   │   ├── CandleStick
│   │   │   │   ├── CandleStickView.js
│   │   │   │   └── index.js
│   │   │   ├── Histogram
│   │   │   │   ├── HistogramView.js
│   │   │   │   └── index.js
│   │   │   ├── LineGraph
│   │   │   │   ├── LineGraphView.js
│   │   │   │   └── index.js
│   │   │   ├── PieChart
│   │   │   │   ├── PieChartView.js
│   │   │   │   └── index.js
│   │   │   ├── ScatterPlot
│   │   │   │   ├── ScatterPlotView.js
│   │   │   │   └── index.js
│   │   │   ├── Whisker
│   │   │   │   ├── WhiskerView.js
│   │   │   │   └── index.js
│   │   │   ├── index.js
│   │   │   └── styles.js
│   │   │
│   │   └── Table/
│   │       ├── EditableRow.js
│   │       ├── TableView.js
│   │       ├── styles.js
│   │       └── index.js
│   │
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   └── serviceWorker.js
│
├── node_modules/
│
├── .gitignore
├── package.json
├── README.md
└── yarn.lock
```

Code 37: Web-application's directory structure

Good guidelines and training early with fairly disciplined rules on directory creation at certain levels should help with this problem. The above directory diagram attempts to explain in plain view how the project is structured, which consist of conceptual levels of

decomposition and intuition starting from the root level and componentizing business assets in software files. As one can easily, in our opinion, find out, even without any software knowledge, the main source code can be found under the src/ sub-directory, whereas in public/ directory are placed common files possibly without any code that interests only the developers but all of projects stakeholders. Truthly, patients.json file with the labeling of columns is also placed in this directory.

As a matter of intuition, the three pages that are observable to the user when navigating from a browser to this web-application are with their relevant names under the "routes/" sub-directory.

Similarly, the common parts of the web-application what are re-used in more than one pages are reusable components, and as a matter of fact they are placed under the components sub-directory. This sub-directory consists of an Autocomplete React component, a Table React components and some Charts React components under the Charts sub-directory. This derives the conclusion that possibly as it seems only these components are commonly used among more than one pages, and this is why we componentized these React components, in order to reuse them with the given functionality we implemented for the purposes of this thesis. Finally, we observe the "layouts/" sub-directory of "src/", which leads us to the summation that this includes big components that capture the largest screen size. And this derives from the naming "layouts" as well as because this directory was not placed under components but as a root-development directory, meaning that it possible applies also to all routes. And that is correct! Layouts sub-directory consists of components that are rendered in all routes, and such components are the Header, the Drawer Menu and the Page Components itself.

To sum up, to conclude to a directory file structure is neither easy to achieve, nor it should be passed by – even when deadlines are applied. Directory structure should be definitely thought and designed from the beginning of a software application and should be definitely considered a time estimation in future for re-designing the structure as a lot of requirements always change as time passes and **backlog** changes. Having a good file directory structure helps everyone in a project, both the creator, the users and the possible future maintainers. [58]

4.4.2 Design Patterns and Principles: Scalability

Scalability is an essential design principle in enterprise software development. Prioritizing it from the start leads to lower maintenance costs, better user experience, and higher agility. Software design is a balancing act where developers work to create the best product within a client's time and budget constraints. There's no avoiding the necessity of compromise. Tradeoffs must be made in order to meet a project's requirements, whether those are technical or financial. Too often, though, companies prioritize cost over scalability or even dismiss its importance entirely. This is unfortunately common in big data initiatives, where scalability issues can sink a promising project. Scalability isn't a "bonus feature." It's the quality that determines the lifetime value of software, and building with scalability in mind saves both time and money in the long run.

A system is considered scalable when it doesn't need to be redesigned to maintain effective performance during or after a steep increase in workload. "Workload" could refer to simultaneous users, storage capacity, the maximum number of transactions handled, or anything else that pushes the system past its original capacity. Scalability isn't a basic

[58] https://medium.com/swift2go/code-structure-and-readability-part-4-project-structure-99f9a6671ce3

requirement of a program in that unscalable software can run well with limited capacity. However, it does reflect the ability of the software to grow or change with the user's demands. Any software that may expand past its base functions- especially if the business model depends on its growth- should be configured for scalability.

Principles of Scalability

Several factors affect the overall scalability of software:

A. Usage

Usage measures the number of simultaneous users or connections possible. There shouldn't be any artificial limits on usage. Increasing it should be as simple as making more resources available to the software. In our thesis application this is handled from the Node.js backed. As we described in Node.js section, Node.js works using and event-stack and never locks the users, even if they work simultaneously.

B. Maximum stored data

This is especially relevant for sites featuring a lot of unstructured data: user uploaded content, site reports, and some types of marketing data. Data science projects fall under this category as well. The amount of data stored by these kinds of content could rise dramatically and unexpectedly. Whether the maximum stored data can scale quickly depends heavily on database style (SQL vs NoSQL servers), but it's also critical to pay attention to proper indexing. In our thesis application this was one of the many reasons we selected a NoSQL database and more specifically a MongoDB, as also described in MongoDB section.

C. Code

Inexperienced developers tend to overlook code considerations when planning for scalability. Code should be written so that it can be added to or modified without refactoring the old code. Good developers aim to avoid duplication of effort, reducing the overall size and complexity of the codebase. Applications do grow in size as they evolve, but keeping code clean will minimize the effect and prevent the formation of "spaghetti code". In our thesis application this is one of the reasons of using React, as it allows us to create reusable components, as we also found out earlier when describing React and the directory structure.

D. Scaling Out Vs Scaling Up

Scaling up (or "vertical scaling") involves growing by using more advanced or stronger hardware. Disk space or a faster central processing unit (CPU) is used to handle the increased workload. Scaling up offers better performance than scaling out. Everything is contained in one place, allowing for faster returns and less vulnerability. The problem with scaling up is that there's only so much room to grow. Hardware gets more expensive as it becomes more advanced. At a certain point, businesses run up against the law of diminishing returns on buying advanced systems. It also takes time to implement the new hardware. Because of these limitations, vertical scaling isn't the best solutions for software that needs to grow quickly and with little notice. Scaling out (or "horizontal scaling") is much more widely used for enterprise purposes. When scaling out, software grows by using more- not more advanced- hardware and spreading the increased workload across the new infrastructure. Costs are lower because the extra servers or CPUs can be the same type currently used (or any compatible kind). Scaling happens faster, too, since nothing has to be imported or rebuilt. There is a slight tradeoff in speed, however. Horizontally-scaled software is limited by the speed with which the servers can communicate. The difference isn't large enough to be noticed by most users, though, and there are tools to help developers minimize the effect. As a result, scaling out is considered a better solution when building scalable applications. In our thesis application

this is one of the reasons we consider the deployment in docker containers, as virtualization in clouds helps in scaling out.

Guidelines for Building Highly Scalable Systems

It's both cheaper and easier to consider scalability during the planning phase. Here are some best practices for incorporating scalability from the start:[59]

1. Use load balancing software

Load balancing software is critical for systems with distributed infrastructure (like horizontally scaled applications). This software uses an algorithm to spread the workload across servers to ensure no single server gets overwhelmed. It's an absolute necessity to avoid performance issues. In our thesis application we don't use a load balancer as it wasn't needed. Nevertheless, if this web application, should in future be used by multiple users then surely a load balancer would help.

2. Location matters

Scalable software does as much near the client (in the app layer) as possible. Reducing the number of times apps must navigate the heavier traffic near core resources leads to faster speeds and less stress on the servers. Edge computing is something else to consider. With more applications requiring resource-intensive applications, keeping as much work as possible on the device lowers the impact of low signal areas and network delays. In our thesis application this is why we decided to do the calculations of data on the frontend and leave to the backend only the pure business logic. As described earlier, devices nowadays are much more powerful than supercomputers of 1-2 decades ago. This means, that except for big data processing, and a few more occasions, the rest of computation can be done on the frontend. Moreover, in addition to what is described above, a react service worker implementation would fit for cases with offline usage and cached queries in future.

3. Cache where possible

Be conscious of security concerns, but caching is a good way to keep from having to perform the same task over and over. In our thesis application we cache queries in the frontend but in future we could cache also queries on the backend for faster responses.

4. Lead with API

Users connect through a variety of clients, so leading with API that don't assume a specific client type can serve all of them. In our thesis application this is why we decided to let GraphQL manipulate the business logic as described in the relative section.

5. Asynchronous processing

It refers to processes that are separated into discrete steps which don't need to wait for the previous one to be completed before processing. For example, a user can be shown a "sent!" notification while the email is still technically processing. Asynchronous processing removes some of the bottlenecks that affect performance for large-scale software. In our thesis application using React and JS callback we manage to handle asynchronous processing.

6. Limit concurrent access to limited resources

[59] https://conceptainc.com/blog/importance-of-scalability-in-software-design/

Don't duplicate efforts. If more than one request asks for the same calculation from the same resource, let the first finish and just use that result. This adds speed while reducing strain on the system. In our thesis application this is a built in process that Apollo with GraphQL does. Another way to handle such purposes would be with Relay.js or with React Redux-Saga libraries.

7. Use a scalable database

NoSQL databases tend to be more scalable than SQL. SQL does scale read operations well enough, but when it comes to write operations it conflicts with restrictions meant to enforce ACID principles. Scaling NoSQL requires less stringent adherence to those principles, so if ACID compliance isn't a concern a NoSQL database may be the right choice.[60]
Some snippets showing scalability of our project list as follow:

We can add easily more charts in Charts directory where they all share the same styling, without needing to add it in each one of them

```
...
|   ├── components/
|   |   |
|   |   ├── Charts/
|   |   |   ├── Chart1
|   |   |   |   ├── Chart1View.js
|   |   |   |   └── index.js
|   |   |   ├── Chart2
|   |   |   |   ├── Chart2View.js
|   |   |   |   └── index.js
|   |   |   ├── index.js
|   |   |   └── styles.js
...
```

Code 38: Scalable directory structure

We can easily add a Chart in a DashBoard and apply to it all the responsive styles and feed it with all the data needed to render by only adding it in the array returned from the chartComponents:

```
// Project ALD-BRFAA-UI
// Filepath: src/routes/DashBoardView.js
...
chartComponents = (filter = null) => (
    vizData,
    groupDataCategories,
    vizGroupData,
    commonShowGrouped,
    commonShowXQualitative,
    commonShowYQualitative
  ) =>
    !filter
      ? [
          <Chart1
            data={vizData}
            groupedData={vizGroupData}
            groupedCategories={groupDataCategories}
```

60 https://medium.com/salesloft-engineering/react-atl-building-scalable-applications-using-react-8f2eade49347

```
          commonShowGrouped={commonShowGrouped}
          commonShowXQualitative={commonShowXQualitative}
          commonShowYQualitative={commonShowYQualitative}
          enterFullScreen={this.enterFullScreen}
          isFullScreen={this.state.isFullScreen}
          saveImage={this.saveImage}
        />,
        <Chart2
          data={groupDataCategories}
          groupedData={vizGroupData}
          groupedCategories={groupDataCategories}
          commonShowGrouped={commonShowGrouped}
          enterFullScreen={this.enterFullScreen}
          isFullScreen={this.state.isFullScreen}
          saveImage={this.saveImage}
        />,
...

    <Grid container style={{ padding: 30 }}>
     {this.chartComponents(fullScreenVizType)(
       vizData,
       vizDataCategories,
       vizGroupData,
       commonShowGrouped,
       commonShowXQualitative,
       commonShowYQualitative
     ).map((component) => (
       <Grid item xs={12} sm={!isFullScreen && 4} md={!isFullScreen && 3} style={{ padding: 20 }}>
         {component}
       </Grid>
     ))}
    </Grid>,
...
```

Code 39: Scalable visualization addition

We can easily add as many ListItems and their links to the Drawer menu as we like by simply adding an entry in mainMenuEntries:

```
// Project ALD-BRFAA-UI

// Filepath: src/layouts/menu.js


...

export const mainMenuEntries = {
  Dashboard: { link: '/', renderIcon: () => <DashboardIcon /> },
  Query: { link: '/query', renderIcon: () => <QueryIcon /> },
  Analysis: { link: '/analysis', renderIcon: () => <LayersIcon /> },
  Reports: { link: reportsLink, renderIcon: () => <BarChartIcon /> },
  NextItem: { link: '/nextitem', renderIcon: () => <AnotherIcon /> },
};

export const mainListItems = Object.keys(mainMenuEntries).map((entry) => (
  <Link to={mainMenuEntries[entry].link} style={{ textDecoration: 'none' }}>
    <ListItem button>
      <ListItemIcon>{mainMenuEntries[entry].renderIcon()}</ListItemIcon>
      <ListItemText primary={entry} />
    </ListItem>
  </Link>
));

...
```

Code 40: Scalable menu creation

4.4.3 Design Patterns and Principles: Maintainability

Maintainability is another essential design principle in enterprise software development. Maintainability means fixing, updating, servicing and to modify the system or update the software for performance improvements or for the correction of faults.

Maintainability also includes the addition of new functionality or the adaptation of software to meet new requirements for the customer needs. Software maintainability is the degree of an application to repaired or enhanced it. During the system development life cycle

(SDLC) this phase requires more development effort than any other phase. It has also been calculated that approximately 75 percent of the software development cost is related to software maintenance.

What is more, maintainability increases the reliability, efficiency or safety of the software. It is also used to make future maintenance easier. It is used to increase the lifetime of the software. Maintainability repair or replace the faulty components and make the software even better as compared to the previous condition of the software.

Software maintenance is required when the customer demands new features and new functions in the software. Sometimes maintenance is required when the hardware of the system is changed then the modification of software is needed. Market conditions and organization changes are also the reasons for software modification. It also includes that when the issue is detected, immediately fix it before it becomes a big problem. Sometimes viruses and malware are detected in the software which causes problems for the user than software maintenance is required to fix it or improve the performance.

Types of Maintainability

Software maintainability consists of four types: [72]

1. Corrective maintenance

Corrective maintenance is defined as maintenance of bugs or errors. It means when the error is detected in the software then the corrective maintenance is required to fix it. These bugs or errors are responsible for the faults which may appear in the code, design or logic of the software. Sometimes the user asks for the enhancements of the software and not about fixing the bugs. Corrective maintenance requires the correction of existing faults in the software. Sometimes a change in hardware also cause bugs or errors. In this thesis application for example in several occasions we had to use the corrective maintenance as using the KanBan framework of the Agile methodology has to do with adding in Backlog new features and implementing them. So as described earlier, some bugs or not were occasionally added in backlog to be developed or fixed.

2. Adaptive maintenance

Adaptive maintenance includes the environmental changes where your software is living. Changes to the hardware, operating system, software dependencies, and organizational business rules and policies are handled in adaptive maintenance. By these modifications to the environment, changes can occur in the other parts of the software. In changing circumstances adaptive maintenance is required to keep your software fresh or to increase the lifetime of the software. In this thesis we used adaptive maintenance when we had to deal with the multi-system build and serve. The application was then developer so that it can adapt to setups in any platform both by using docker and by just be set up to a Windows or Linux server.

3. Perfective maintenance

Perfective maintenance refers to the changes in features and requirements in your existing system. After sometime when user suggests for new features and new functionality of the software than adaptive maintenance is used. In adaptive maintenance, some features are removed from the software which features are not effective for the software. Adaptive maintenance involves 50-55% of the maintenance work. In this thesis as you will also find in the following Image every once in a while between fixes in code or new feature development we also use a refactor commit, which means that we rewritten the same code but in different style that is more suitable, adaptive, reusable and scalable.

4. Preventive maintenance

Preventive maintenance maximizes the maintainability or understanding of the software system. Documentation updating or code optimizing are involved in preventive maintenance. Preventive maintenance helps the software to become more scalable, stable, understandable, maintainable. This maintenance acts as medicine to prevent the problems. Restructuring the data and code of the software are implemented in preventive maintenance. As also mentioned in perfective maintenance, in this web application development we used both these types of maintenance with main purposes the scalability, performance and reusability of the application components.

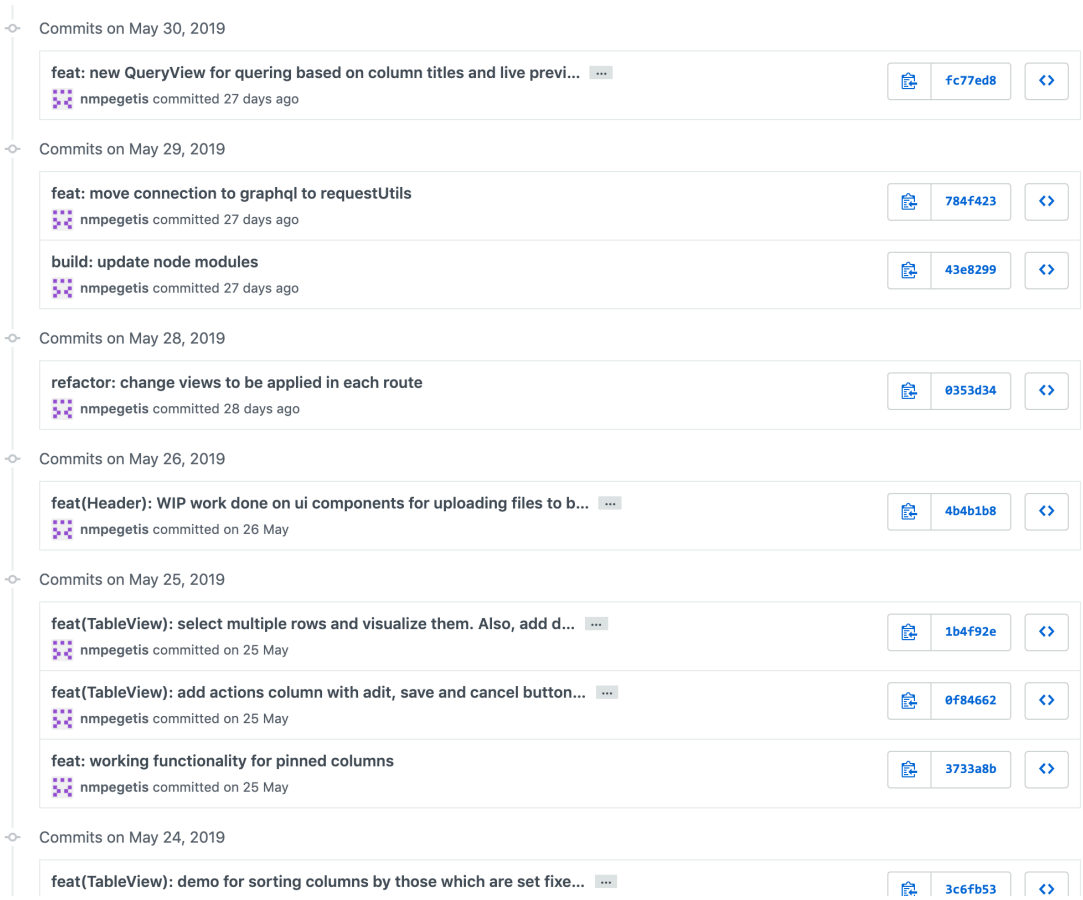
Some examples showing maintainability of our project list as follow:
In our project we used the commit conventions that are referenced in this Medium article[61] written by the author of this thesis. These commit conventions can be assigned to these four types of maintainability as follows.

Commit message types relations to maintenance types:

- feat (a new feature) - *Adaptive*, *Corrective, Perfective*
- fix (bug fix) - *Corrective*
- perf (a code change that improves performance) - *Perfective*
- ci (changes to CI configuration files and scripts (e.g. kubernetes, swarm, jenkins, openshift, heroku)) - *Adaptive*
- build (changes that affect the build system or external dependencies (e.g. gulp, npm, yarn, .env variables)) - *Adaptive*
- docs (documentation only) - *Preventive*
- style (formatting, missing semi colons) - *Adaptive, Corrective*
- refactor (a code change that neither fixes a bug nor adds a feature) - *Perfective, Preventive*
- test (code changes in tests) - *Adaptive, Preventive*
- chore (maintain (e.g. remove console.log, unnecessary comments)) - *Corrective, Perfective, Preventive*

The Image below as referenced earlier, shows a screen capture of the commits applying the commit conventions with messages that justify the Maintenance type they belong to.

[61] https://medium.com/@nmpegetis/git-commit-message-conventions-841d6998fc4f



Picture 43: Git commit type and messages justify the maintenance types [72]

Also, additionally to the above here you can also find two code snippets that justifies the Preventive maintenance type using Documanation with @JSDocs[62].

```
// Project ALD-BRFAA-UI
// Filepath: src/layouts/Drawer.js

...

/**
 * @param {object} classes
 * @param {boolean} open
 * @param {function} onDrawerClose
 * @returns {Component} {Drawer}
 */
const SideDrawer = withStyles(styles)(({ classes, open, onDrawerClose }) => (
  <Drawer
    variant="permanent"
    classes={{
      paper: classNames(classes.drawerPaper, !open && classes.drawerPaperClose),
    }}
    open={open}
  >
    <div className={classes.toolbarIcon}>
      <IconButton onClick={onDrawerClose}>
        <ChevronLeftIcon />
      </IconButton>
    </div>
    <Divider />
    <List>{mainListItems}</List>
    <Divider />
```

[62] https://devhints.io/jsdoc

```
    <List>{secondaryListItems}</List>
  </Drawer>
));

...
```

Code 41: Documentation usage in Drawer menu React component

```
// Project ALD-BRFAA-UI
// Filepath: src/layouts/Header.js

...

/**
 * @param {object} classes
 * @param {boolean} open
 * @param {function} onDrawerOpen
 * @param {object} history
 * @returns {Component} {Header}
 */
const Header = withStyles(styles)(({ classes, open, onDrawerOpen, history }) => {
  const [ loaded, setLoaded ] = useState(0);
  const [ selectedFile, setSelectedFile ] = useState(0);

  const handleselectedFile = (event) => {
    setSelectedFile(event.target.files[0]);
    setLoaded(0);
  };

...
```

Code 42: Documentation usage in Header React component with hooks

4.4.4 Design Patterns and Principles: Reusability

While developing a complex software, programmers or software developers require immense knowledge and brain-storming. Absolutely, their natural tendency to write long lines of code which can be intricate, hard to manage and can't be reused. However, programmers must store the code components and knowledge that can be reused, which in return can save time for developing a software requiring similar code capabilities.

Software reusability is a process of developing a software from existing software components, instead of developing an entire software from scratch. In other words, software reusability is developing a brand new software from an existing one. Software reusability results into increased productivity and quality by minimizing risk of newly developed projects. Software reusability not only depends on code. On the contrary, it entails all entities of software development life cycle like software components, test suites, documentations and designs.

Reusability can be achieved by opting for software metrics technique. Software metrics are considered to be vital in software development and management. Within an organization, various types of metrics can be applied which includes reuse metrics and software and quality metrics. These kind of metrics prove helpful in shaping appropriate practices, while developing a software and its entire life cycle. However, component based programs is a common method of reusable programs where they are highly dependent on software reuse repositories. Component based reuse programs are becoming extremely popular owing to their cost-effective approach for software development. These programs are highly inclined towards design and development of software systems using reusable components. This is also a development pattern that can be found in all recent large UI libraries and frameworks, like React, Angular, Vue, etc.

Components developed by developers within an organization are not only meant to reuse within a particular organization instead, they are distributed in form of an object code and reused by other environments. Hence, developers looking for reused components are unable to attain exact source codes of components as only object codes are available. Sharing object code online is one of the finest way to make components available for

reusability. Though, there can be few things which can hamper the reusability of component.

Some aspects that strengthen Reusability list as follow:

i. The software component must be easily available for use whenever required.
ii. An appropriate and simple documentation on how to use component can make it more understandable and easy to implement.
iii. The components must not be too complex and should be bug-free.
iv. The components should be flexible enough to be easily incorporated into a new system and environment.
v. Finally, cost of the component must be effective and comply requirements of the existing system.

Software reusability is changing the way programmer's code and organizations operate. Being an evolving concept, lot of organizations are moving forward to incorporate software reusability within their environment. Software reusability can encourage innovation in traditional development methods and also it is a cost effective option. The software reusability research started during 1960 and it has a long way to go.

In this thesis as we silently implied we use software reusability in some, if not all, of the more core and significant parts of the web-application. Npm (node package manager) that we used to download some packages does this exact thing. Npm stores in its registry reusable software that is backed from industries, organizations and simple developers that like to contribute to community open-source software. As mentioned in previous chapters, Apollo, graphql-compose, mongoose, Material-UI, Ant Design, react-vis and even React itself are packages that have arisen to the most successful libraries because they use software reusability, and one can find that the above 5 Reusability aspects, they all apply to these package libraries. What is more, except for the velocity we gain by using such software, it is much more certain that our web application will not fail or act erroneously, because this software as indicated before is not backed by just one developer, but on the contrary by hundreds of the community's developers. [80]

What is more, though, reusability doesn't just have to do with libraries and packages that are publicly available. Reusability is a pattern that all good developers adopt in order to both write less code and adaptive code. React in specific, uses a variety of patterns for code reuse. Some of which are:

I. Inheritance Pattern

Used sparingly to share common code across React class components. In this thesis we didn't use this pattern as in JavaScript it is not that much frequent to use it. Nevertheless, all class created components use this pattern by default as they should inherit the reusable Component class of React.

```
// Project ALD-BRFAA-UI
// Filepath: src/App.js

class AppContainer extends Component {
  render() {
    return (
      <BrowserRouter>
        <Provider store={store}>
          <ApolloProvider client={client}>
            <Layout labels={this.props.appLabels} />
          </ApolloProvider>
        </Provider>
      </BrowserRouter>
    );
  }
}
```

```
export default AppContainer;
```

Code 43: Reusability example using class inheritance pattern

II. Composition Pattern

The core pattern for separating concerns while creating complex UIs with React. Composition is a code reuse technique where a larger object is created by combining multiple smaller objects. In other words it is also the "componentization". We build smaller reusable components to build up larger ones and so on and so forth.

In our thesis web-application a very good showcase for this pattern can be found the the following two code snippets. In the first one we use composition to compose a larger React Fragment Component made of main menu ListItems and Secondary menu ListItems. These two larger React Fragments are exported for a later use from the Drawer menu component in another file that also is composes these two React Fragments among with Dividers and the menu open Button

```
// Project ALD-BRFAA-UI
// Filepath: src/layouts/menu.js

...

export const mainListItems = Object.keys(mainMenuEntries).map((entry) => (
  <Link to={mainMenuEntries[entry].link} style={{ textDecoration: 'none' }}>
    <ListItem button>
      <ListItemIcon>{mainMenuEntries[entry].renderIcon()}</ListItemIcon>
      <ListItemText primary={entry} />
    </ListItem>
  </Link>
));

export const secondaryListItems = (
  <React.Fragment>
    <ListSubheader inset>Saved reports</ListSubheader>
    {secondaryListEntries.map((entry) => (
      <Link to={`${reportsLink}/${entry}`} style={{ textDecoration: 'none' }}>
        <ListItem button>
          <ListItemIcon>
            <AssignmentIcon />
          </ListItemIcon>
          <ListItemText primary={entry.split('T')[0]} />
        </ListItem>
      </Link>
    ))}
  </React.Fragment>
);

...
```

Code 44: Reusability example using composition pattern part 1

```
// Project ALD-BRFAA-UI
// Filepath: src/layouts/Drawer.js

...

/**
 * @param {object} classes
 * @param {boolean} open
 * @param {function} onDrawerClose
 * @returns {Component} {Drawer}
 */
const SideDrawer = withStyles(styles)(({ classes, open, onDrawerClose }) => (
  <Drawer
    variant="permanent"
    classes={{
      paper: classNames(classes.drawerPaper, !open && classes.drawerPaperClose),
    }}
    open={open}
  >
    <div className={classes.toolbarIcon}>
      <IconButton onClick={onDrawerClose}>
        <ChevronLeftIcon />
      </IconButton>
```

```
    </div>
    <Divider />
    <List>{mainListItems}</List>
    <Divider />
    <List>{secondaryListItems}</List>
  </Drawer>
));

...
```

Code 45: Reusability example using composition pattern part 2

III. Decorator Pattern

Used to provide a nice interface for separating out logic shared by multiple components and centralizing it. Decorators are a pattern for editing an instance of a class to give it additional behaviors than it had previously. In contrast to inheritance, decorators are not part of the class definition, but are modifications to the class at run time to allow a subset of objects to have additional data or behavior. The React community generally refers to this pattern as Higher Order Components(HOC).

In this thesis we mostly use HOCs for styling purposes are Material-UI provides us with a withStyles HOC that wraps almost all of our React components and applies styling to them styling in runtime.

```
// Project ALD-BRFAA-UI
// Filepath: src/components/Autocomplete/AutocompleteView.js

...

class IntegrationAutosuggest extends React.Component {

...

  render() {

...

    return (
      <div className={classes.root}>
        <Typography>{this.props.label}</Typography>
        <Autosuggest
          {...autosuggestProps}
          inputProps={{
            classes,
            placeholder: 'Search an available label',
            value: this.state.single || this.props.initialValue,
            onChange: this.handleChange('single'),
          }}
          theme={{
            container: classes.container,
            suggestionsContainerOpen: classes.suggestionsContainerOpen,
            suggestionsList: classes.suggestionsList,
            suggestion: classes.suggestion,
          }}
...
        />
      </div>
    );
  }
}

IntegrationAutosuggest.propTypes = {
  classes: PropTypes.object.isRequired,
};

export default withStyles(styles)(IntegrationAutosuggest);
```

Code 46: Reusability example

IV. Mixin Pattern

Mixins are useful when you have classes with different purposes that shouldn't share an inheritance tree but do have some shared behavior. In practice though they have

historically been tricky to use because it is easy to tightly bind mixin code to the implementation of the code that it is mixed into. More specifically to our thesis use case, we also used React hooks. React hooks is the newest pattern that React has added to its bag of code reuse tricks. So even though "Mixins" are gone now in newer versions of JavaScript, the "Mixin pattern" is back with React Hooks. Hooks use a variation on the Mixin pattern to allow sharing related behavior and data between unrelated function components easily. And with Hooks, we can create a much cleaner version of our Higher Order Component use cases in the decorator pattern section. [73]

In this project for all of our charts creation, we used both mixin and decorator pattern with React hooks and with HOCs. [63]

```
// Project ALD-BRFAA-UI
// Filepath: src/components/Charts/BarPlot/BarPlotView.js

const BarPlotView = ({
    classes,
    data,
    groupedData,
    groupedCategories,
    commonShowGrouped,
    commonShowXQualitative,
    commonShowYQualitative,
    enterFullScreen,
    isFullScreen,
    saveImage,
  }) => {
    const [ showGrouped, setShowGrouped ] = useState(false);
    let [ prevCommonShowGrouped, prevSetCommonShowGrouped ] = useState(null);

    const [ showXQualitative, setShowXQualitative ] = useState(false);
    let [ prevCommonShowXQualitative, prevSetCommonShowXQualitative ] = useState(null);
    const [ showYQualitative, setShowYQualitative ] = useState(false);
    let [ prevCommonShowYQualitative, prevSetCommonShowYQualitative ] = useState(null);

    const [ height, setHeight ] = useState(0);

    useEffect(() => {
      // Similar to componentDidMount
      window.addEventListener('resize', () => setHeight(window.innerHeight));

      // Similar to componentDidUpdate
      setHeight(window.innerHeight);

      // Similar to componentWillUnmount
      return () => window.removeEventListener('resize', () => setHeight(window.innerHeight));
    });

    if (commonShowGrouped !== prevCommonShowGrouped) {
      setShowGrouped(commonShowGrouped);
      prevSetCommonShowGrouped(commonShowGrouped);
    }

    if (commonShowXQualitative !== prevCommonShowXQualitative) {
      setShowXQualitative(commonShowXQualitative);
      prevSetCommonShowXQualitative(commonShowXQualitative);
    }

    if (commonShowYQualitative !== prevCommonShowYQualitative) {
      setShowYQualitative(commonShowYQualitative);
      prevSetCommonShowYQualitative(commonShowYQualitative);
    }

    let name = 'barPlot';
```

[63] https://medium.com/the-andela-way/reactjs-the-art-of-reusable-components-5b67359e8ebb

```
  const categories = (groupedCategories && groupedCategories.map((category) => category.label)) || [];
  return (
    <React.Fragment>
      <Typography variant="h4" gutterBottom component="h2">
        Bar Plot
        <IconButton onClick={enterFullScreen(isFullScreen ? null : 'barplot')}>
          {!isFullScreen ? <FullScreen /> : <FullScreenExit />}
        </IconButton>
        <IconButton onClick={saveImage(name)}>
          <SaveIcon />
        </IconButton>{' '}
      </Typography>
      <Typography variant="caption" gutterBottom component="h2">
        <Switch checked={!!showGrouped} onChange={() => setShowGrouped(!showGrouped)} />
        Show Grouped Data
      </Typography>{' '}
      <Typography variant="caption" gutterBottom component="h2">
        <Switch checked={!!showXQualitative} onChange={() => setShowXQualitative(!showXQualitative)} />
        Show Qualitative Data in X Axis
        <Switch checked={!!showYQualitative} onChange={() => setShowYQualitative(!showYQualitative)} />
        Show Qualitative Data in Y Axis
      </Typography>
      <Typography component="div" className={classes.chartContainer}>
        {showGrouped && <DiscreteColorLegend items={categories} orientation="horizontal" height="70px" />}
        {!data.length && (
          <Typography className={classes.flexCenter}>
            <CircularProgress className={classes.progress} />
          </Typography>
        )}
        <div id={name}>
          <FlexibleXYPlot
            xType={showXQualitative ? 'ordinal' : undefined}
            yType={showYQualitative ? 'ordinal' : undefined}
            animation
            height={!isFullScreen ? 300 : height * ratio}
          >
            {!showGrouped ? (
              <VerticalBarSeries cluster="stack 1" data={data} />
            ) : (
              (groupedData || [])
                .map((groupData) => <VerticalBarSeries cluster="stack 1" data={groupData} />)
            )}
            <VerticalGridLines />
            <HorizontalGridLines />
            <XAxis title={data[0] && data[0].xAxisName} tickLabelAngle={-45} />
            <YAxis title={data[0] && data[0].yAxisName} />
          </FlexibleXYPlot>
        </div>
      </Typography>
    </React.Fragment>
  );
 }
);

export default withStyles(styles)(BarPlotView);
```

Code 47: BarPlot creation using both mixin and decorator reusability pattern

4.4.5 Design Patterns and Principles: Performance

Software performance testing is the practice of determining whether a given application has the capacity to perform in terms of scalability and responsiveness under a specified workload. Responsiveness refers to the ability of a given application to meet pre-determined objectives for throughput, while scalability is the number of activities processed within a given time. Performing this type of testing is a key factor when ascertaining the quality of a given application.

Software performance testing is done to serve three main purposes. First, testing is done to determine whether the application meets the specified performance criteria. For example, a performance criteria may specify that an application must be able to handle 500 concurrent users. Secondly, it compares two or more applications with the objective of determining which one can perform better. For instance, say your application needs a

tool to export reports. You can compare tools to see which tool's performance can best handle the export requirements. Lastly, performance testing is done to measure the parts or configuration of a given application that are responsible for the poor performance of the application. A common example of this would be insufficient memory leading to performance bottlenecks.

Goals of Performance Testing

The entire process of software performance testing is done to accomplish a set of four goals: [74]

i. To determine the throughput or the rate of transaction.
ii. To determine the server response time, which is the time taken by a given application node to give a response to a request made by another node.
iii. To determine the response time of the render, which requires the inclusion of functional test scripts in the test scenario.
iv. To determine the performance specifications and document them in the test plan.

As for React performance can be easily measured by making a performance audit and see how the applications React components are spending their time. This feature is an extension to all big browser. In this thesis for measuring Performance we used the Chrome Dev Tools Audit section, where one can fine tune the app and find expensive components and methods.

When developing in React one of the most frequent performance issues that are discussed is that a lot of times, especially in React Lists creation an performance issue appear that has to do with the "key" prop. This also appears in the console of the browser. This key prop is used from React in similar sibling React components to distinguish them. If it is applied, React instantly finds the DOM element that corresponds to each component, elsewise React has to loop through all elements to decide what is the DOM element that it may be looking for. [64]

Some more frequent performance issues list as follow:

i. Manage carefully the shouldComponentUpdate lifecycle with a logical statement
ii. Related to the use case prefer to use PureComponent instead of Component, if applicable
iii. Use immutable data
iv. Use stateless components
v. Analyze the webpack bundle.js and clear packages not used

In this application before deploying to the server we made some audits and fixes issues that had to do with performance. More specifically we fixed an issue like the one described above with the key omitting in Lists. Another performance boost that we gave to the application was that in the last two months of the implementation we refactored our code and by switched from using Class Components to using stateless components where applicable and React Hooks. Finally, as for the webpack bundle, we didn't remove anything from the package.json as all packages, where used in the application, while also this was a concern we had from the beginning of the implementation – to use the less possible needed package libraries.

[64] https://medium.com/@joekarlsson/building-high-applications-react-applications3d02145a81e6

4.5 Visualization

In this chapter's section we are going to define what is a visualization. The word "visualization" is in many cases problematic, and there have been a lot of definitions, some unsuccessful, that try to define this field. But before moving on to define a visualization, let's consider this: What is not a visualization? It is easy to argue that anything visual is a visualization in some way – but does that mean anything? Here is a definition of visualization and a few examples of different chart types, that are created with react-vis (this thesis visualization library) to illustrate the different criteria according to which it is recommended to select a chart type for use.

4.5.1 Definition

The following are three minimal criteria that any visualization has to fulfill to be considered a pragmatic visualization. A good visualization certainly has to do more, but these criteria are useful to draw the line between a lot of things that are often called visualization and what we consider visualization in this field. [75]

Based on (non-visual) data

A visualization's purpose is the communication of data. That means that the data must come from something that is abstract or at least not immediately visible (like the inside of the human body). This rules out photography and image processing. Visualization transforms from the invisible to the visible.

Produce an image

It may seem obvious that a visualization has to produce an image, but that is not always so clear. Also, the visual must be the primary means of communication, other modalities can only provide additional information. If the image is only a small part of the process, it is not visualization.

The result must be readable and recognizable

The most important criteria is that the visualization must provide a way to learn something about the data. Any transformation of non-trivial data into an image will leave out information, but there must be at least some relevant aspects of the data that can be read. The visualization must also be recognizable as one and not pretend to be something else (see the discussion of Informative Art). This definition was published in a paper on Visualization Criticism.

Data visualization refers to techniques used to communicate insights from data through visual representation. Its main goal is to distill large datasets into visual graphics to allow for easy understanding of complex relationships within the data. It is often used interchangeably with terms such as information graphics, statistical graphics, and information visualization.

It is one of the steps of the data science process developed by Joe Blitzstein, which is a framework for approaching data science tasks. After data is collected, processed, and modeled, the relationships need to be visualized so a conclusion can be made. It's also a component of the broader discipline of data presentation architecture (DPA), which seeks to identify, locate, manipulate, format, and present data in the most efficient way. [76]

4.5.2 Visualizations Necessity

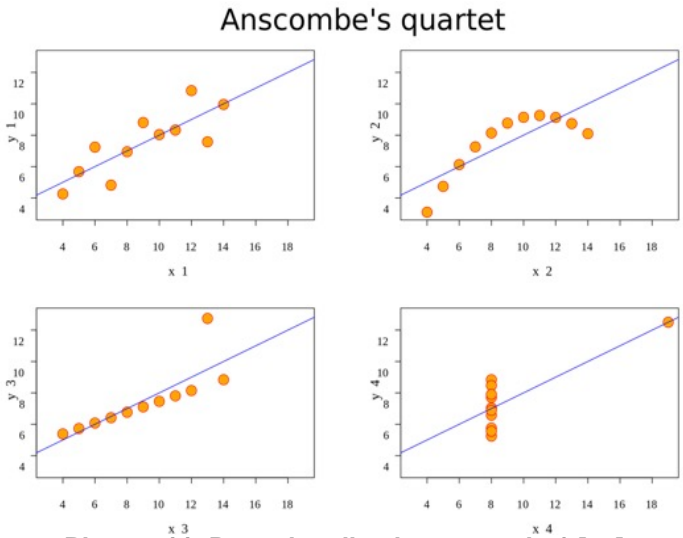
According to the World Economic Forum, the world produces 2.5 quintillion bytes of data every day, and 90% of all data has been created in the last two years. With so much data, it's become increasingly difficult to manage and make sense of it all. It would be

impossible for any single person to wade through data line-by-line and see distinct patterns and make observations. Data proliferation can be managed as part of the data science process, which includes data visualization. [77]

Some reasons that verify the visualizations necessity lists as follows:

Improved Insight

Data visualization can provide insight that traditional descriptive statistics cannot. A perfect example of this is Anscombe's Quartet, created by Francis Anscombe in 1973. The illustration includes four different datasets with almost identical variance, mean, correlation between X and Y coordinates, and linear regression lines. However, the patterns are clearly different when plotted on a graph. Below, you can see a linear regression model would apply to graphs one and three, but a polynomial regression model would be ideal for graph two. This illustration highlights why it's important to visualize data and not just rely on descriptive statistics.



Picture 44: Data visualization example 1 [77]

Faster Decision Making

Companies who can gather and quickly act on their data will be more competitive in the marketplace because they can make informed decisions sooner than the competition. Speed is key, and data visualization aides in the understanding of vast quantities of data by applying visual representations to the data. This visualization layer typically sits on top of a data warehouse or data lake and allows users to discover and explore data in a self-service manner. Not only does this spur creativity, but it reduces the need for IT to allocate resources to continually build new models.

For example, say a marketing analyst who works across 20 different ad platforms and internal systems needs to quickly understand the effectiveness of marketing campaigns. A manual way to do this would be to go to each system, pull a report, combine the data, and then analyze in Excel. The analyst will then need to look at a swarm of metrics and attributes and will have difficulty drawing conclusions. However, modern business intelligence (BI) platforms will automatically connect the data sources and layer on data visualizations so the analyst can slice and dice the data with ease and quickly come to conclusions about marketing performance.

Basic Example

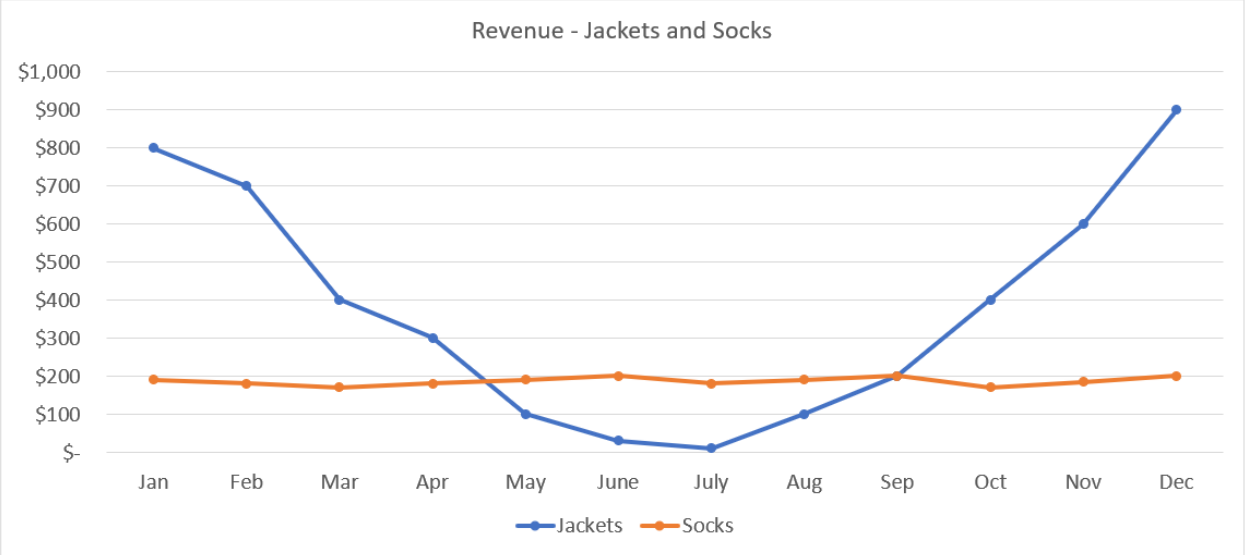
Let's say you're a retailer and you want to compare sales of jackets to sales of socks over the course of the previous year. There's more than one way to present the data, and tables are one of the most common. Here's what this would look like:

Revenue - Jackets and Socks (Thousands of U.S. $)

	Jan	Feb	Mar	Apr	May	June	July	Aug	Sep	Oct	Nov	Dec
Jackets	$ 800	$ 700	$ 400	$ 300	$ 100	$ 30	$ 10	$ 100	$ 200	$ 400	$ 600	$ 900
Socks	$ 190	$ 180	$ 170	$ 180	$ 190	$ 200	$ 180	$ 190	$ 200	$ 170	$ 185	$ 200

The table above does an excellent job showing precise if this information is needed. However, it's difficult to instantaneously see trends and the story the data tells.

Now here's the data in a line graph visualization:



Picture 45: Data visualization example 2 [77]

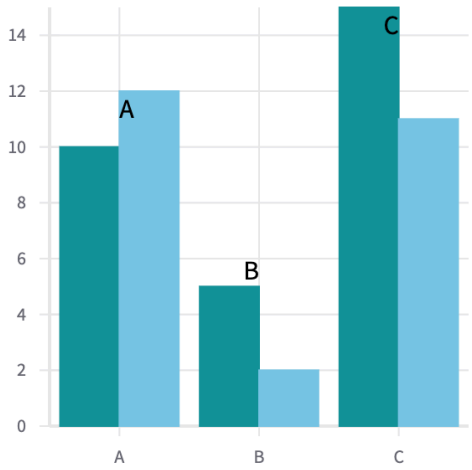
From the visualization, it becomes immediately obvious that sales of socks remain constant, with small spikes in December and June. On the other hand, sales of jackets are more seasonal, and reach their low point in July. They then rise and peak in December before decreasing monthly until right before fall. You could get this same story from looking at the chart, but it would take you much longer. Imagine trying to make sense of a table with thousands of data points. [78]

4.5.3 Chart Types

In this section we illustrate all different types of charts that we used in this thesis. Here is an overview of each type of chart used in this thesis web-application: [79] [76]

Column Chart

A column chart is used to show a comparison among different items, or it can show a comparison of items over time. You could use this format to see the revenue per landing page or customers by close date.



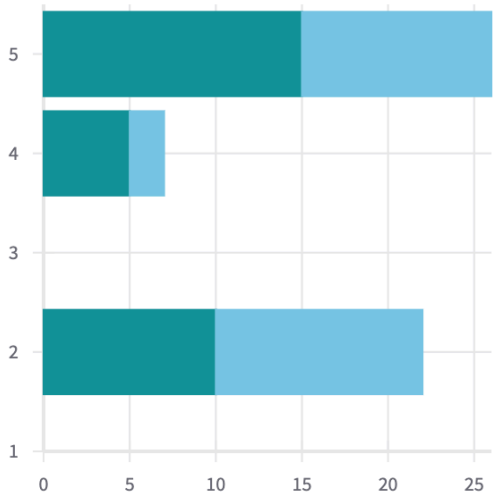
Picture 46: Column chart using react-vis [79]

Design Best Practices for Column Charts:

- Use consistent colors throughout the chart, selecting accent colors to highlight meaningful data points or changes over time.
- Use horizontal labels to improve readability.
- Start the y-axis at 0 to appropriately reflect the values in your graph.

Bar Chart

A bar chart, basically a horizontal column chart, should be used to avoid clutter when one data label is long or if you have more than 10 items to compare. This type of visualization can also be used to display negative numbers.



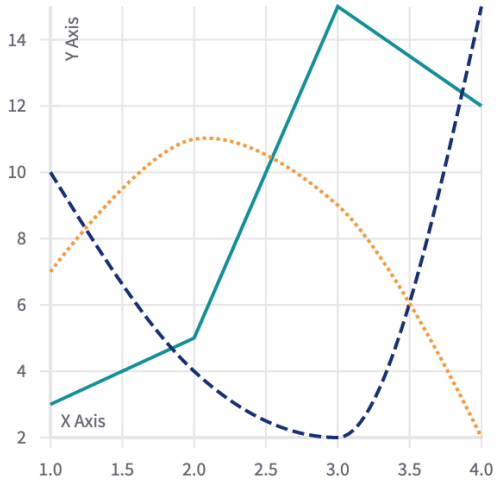
Picture 47: Bar chart using react-vis [76]

Design Best Practices for Bar Charts:

- Use consistent colors throughout the chart, selecting accent colors to highlight meaningful data points or changes over time.
- Use horizontal labels to improve readability.
- Start the y-axis at 0 to appropriately reflect the values in your graph.

Line Chart

A line chart reveals trends or progress over time and can be used to show many different categories of data. One should use it with a continuous data set.



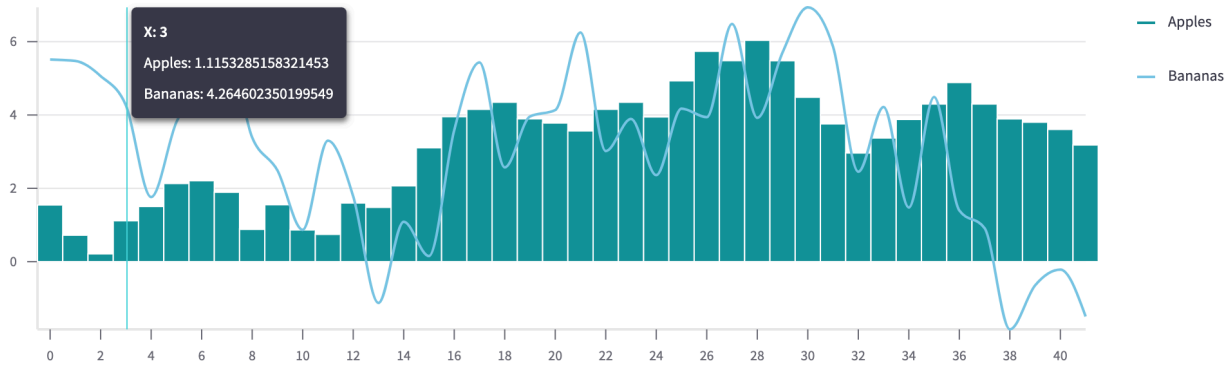
Picture 48: Line chart using react-vis [76]

Design Best Practices for Line Charts:

- Use solid lines only.
- Don't plot more than four lines to avoid visual distractions.
- Use the right height so the lines take up roughly 2/3 of the y-axis' height.

Dual Axis Chart

A dual axis chart allows you to plot data using two y-axes and a shared x-axis. It's used with three data sets, one of which is based on a continuous set of data and another which is better suited to being grouped by category. This should be used to visualize a correlation or the lack thereof between these three data sets.



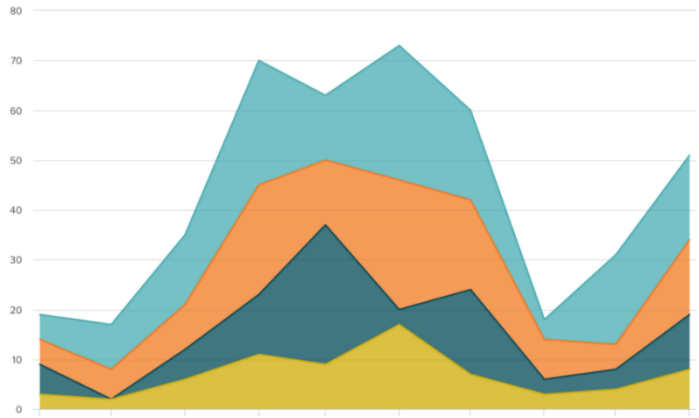
Picture 49: Dual Axis chart using react-vis [76]

Design Best Practices for Dual Axis Charts:

- Use the y-axis on the left side for the primary variable because brains are naturally inclined to look left first.
- Use different graphing styles to illustrate the two data sets, as illustrated above.
- Choose contrasting colors for the two data sets.

Area Chart

An area chart is basically a line chart, but the space between the x-axis and the line is filled with a color or pattern. It is useful for showing part-to-whole relations, such as showing individual sales reps' contribution to total sales for a year. It helps you analyze both overall and individual trend information.



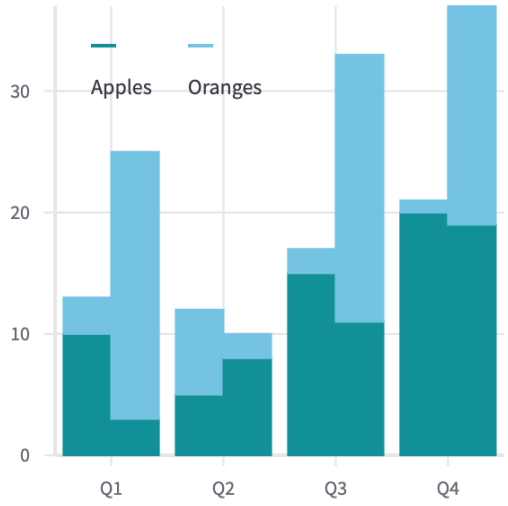
Picture 50: Column chart using react-vis [76]

Design Best Practices for Area Charts:

- Use transparent colors so information isn't obscured in the background.
- Don't display more than four categories to avoid clutter.
- Organize highly variable data at the top of the chart to make it easy to read.

Stacked Bar Chart

This should be used to compare many different items and show the composition of each item being compared.



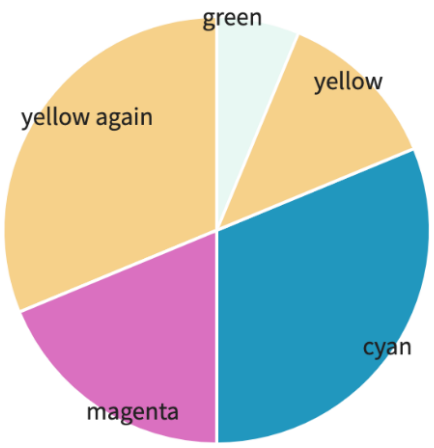
Picture 51: Stacked Bar chart using react-vis [76]

Design Best Practices for Stacked Bar Graphs:

- Best used to illustrate part-to-whole relationships.
- Use contrasting colors for greater clarity.
- Make chart scale large enough to view group sizes in relation to one another.

Pie Chart

A pie chart shows a static number and how categories represent part of a whole -- the composition of something. A pie chart represents numbers in percentages, and the total sum of all segments needs to equal 100%.



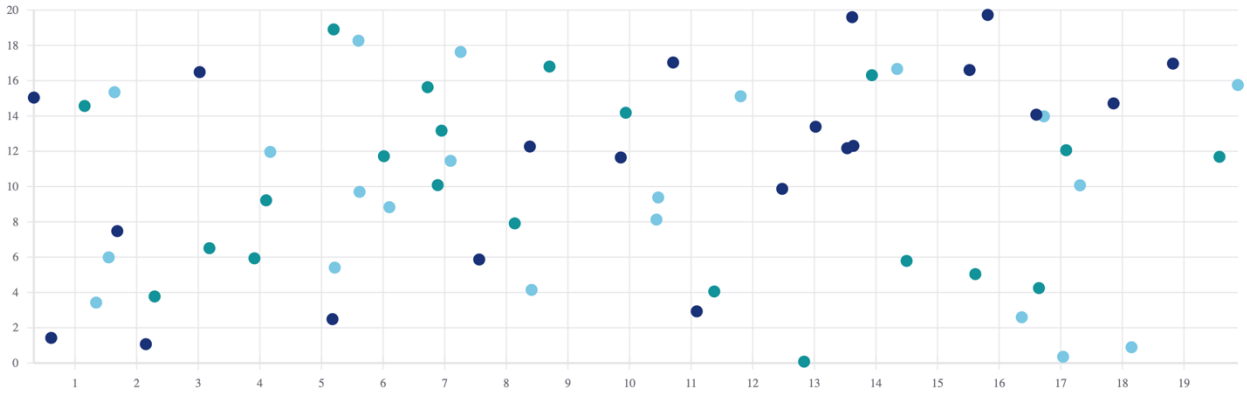
Picture 52: Pie chart using react-vis [76]

Design Best Practices for Pie Charts:

- Don't illustrate too many categories to ensure differentiation between slices.
- Ensure that the slice values add up to 100%.
- Order slices according to their size.

Scatter Plot Chart

A scatter plot or scattergram chart will show the relationship between two different variables or it can reveal the distribution trends. It should be used when there are many different data points, and you want to highlight similarities in the data set. This is useful when looking for outliers or for understanding the distribution of your data.



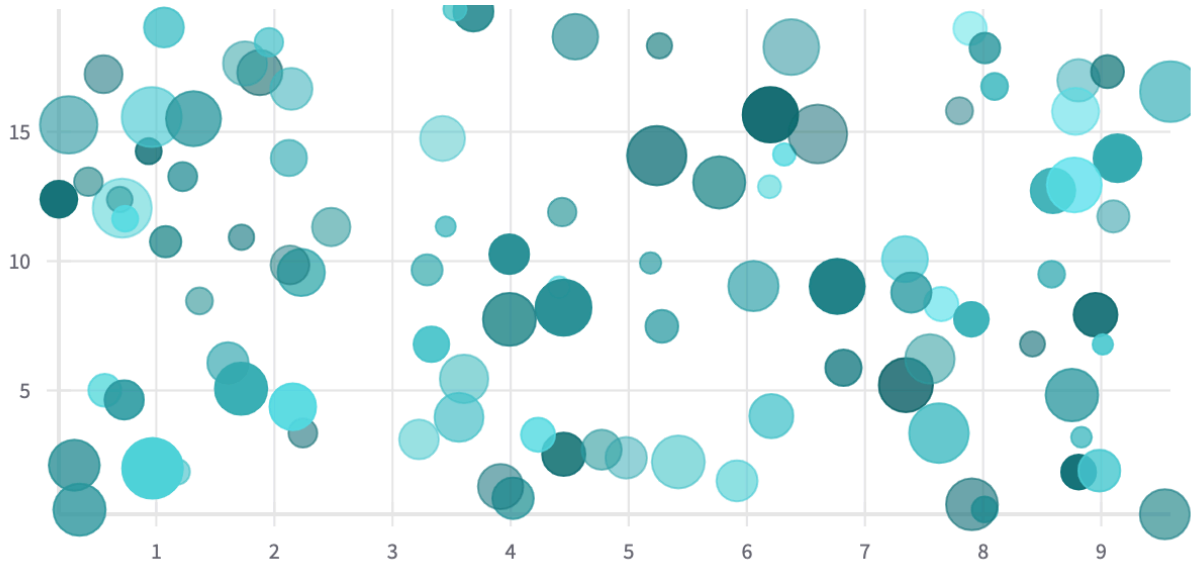
Picture 53: Scatter Plot chart using react-vis [76]

Design Best Practices for Scatter Plots:

- Include more variables, such as different sizes, to incorporate more data.
- Start y-axis at 0 to represent data accurately.
- If you use trend lines, only use a maximum of two to make your plot easy to understand.

Bubble Chart

A bubble chart is similar to a scatter plot in that it can show distribution or relationship. There is a third data set, which is indicated by the size of the bubble or circle.



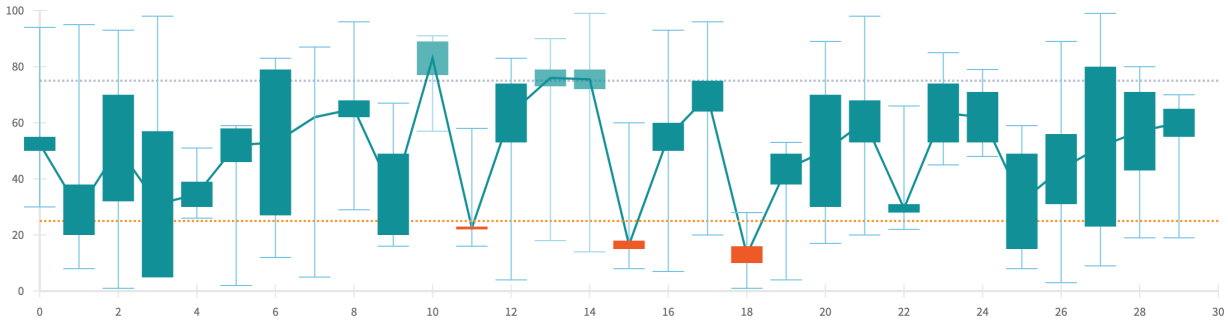
Picture 54: Bubble chart using react-vis [76]

Design Best Practices for Bubble Charts:

- Scale bubbles according to area, not diameter.
- Make sure labels are clear and visible.
- Use circular shapes only.

Whisker Chart

Whisker charts, also known as boxplots, are very useful when large numbers of observations are involved and when two or more data sets are being compared. The Box & Whisker chart displays the spread and skewness in a batch of data through its five-number summary: minimum, maximum, median, upper and lower quartiles. [80]



Picture 55: Whisker chart using react-vis [80]

Design Best Practices for Whiskers and Boxplots:

- For a quick understanding of the distribution of a dataset
- To know whether a distribution is skewed or not
- To find out unusual observations/errors in the data set

4.5.4 Chart Selection

Finally in this section we are going to give some recommendations for chart usage regarding the purposes of using a visualization. Given as purposes the following six distinct reasons: data comparing, data composing, understanding of data distribution, analyzing data trends, and data set relations understanding we recommend for each reason to use the following Charts as they are listed below: [76] [79]

Comparison

Charts are perfect for comparing one or many value sets, and they can easily show the low and high values in the data sets. To create a comparison chart, we recommend using

almost all described types of charts: Column, Bar, Pie, Line, Scatter Plot, Bubble, and Whiskers.

Composition

We recommend using this type of chart to show how individual parts make up the whole of something. To show composition, the proposed charts are: Pie, Stacked Bar, Stacked Column, and Area.

Data distribution

Distribution charts help in understanding outliers, the normal tendency, and the range of information in the dataset's values. Use these charts to show distribution: Scatter Plot, BubbleCharts, Line, Whiskers.

Data Trend Analysis

If you want to know more information about how a data set performed based on a specific third value like a time period, there are specific chart types that do extremely well, such as Line Charts, Dual-Axis Line Charts, Column Charts, and Area Chart.

Dataset relationships

Relationship charts are suited to showing how one variable relates to one or numerous different variables. You could use this to show how something positively effects, has no effect, or negatively effects another variable. When trying to establish the relationship between things, use these charts: a Scatter Plot, a Bubble Chart, or a Line Chart.

5. WEB APPLICATION OVERVIEW

5.1 Introduction

In this chapter we present an overview of the different functionality offered from the web application we created. In the beginning we refer to every screen of the application and the value that each one tries to offer for the application's users. Then, we continue with some example use cases, that handle main user concerns like data sharing among users, data discovery using search, data projection and predictability using visualizations, data cleansing, data filtering, data import from csv formatted file, data and filtered data extraction to csv format, data history saving, data exploration with navigation, data and filtered data visualization for comparisons, composition, understanding, distribution, trend and relations analysis purposes based on criteria, data additions, editing and deletions. In these use cases we note the ease with which a user (researcher or doctor in this thesis case) can handle all the above data-related concerns based on the pre-application's workload and post-application's workload. Finally, we make a comparison with some of the best tools nowadays for data-selection and visualization.

5.2 Web Application Screens

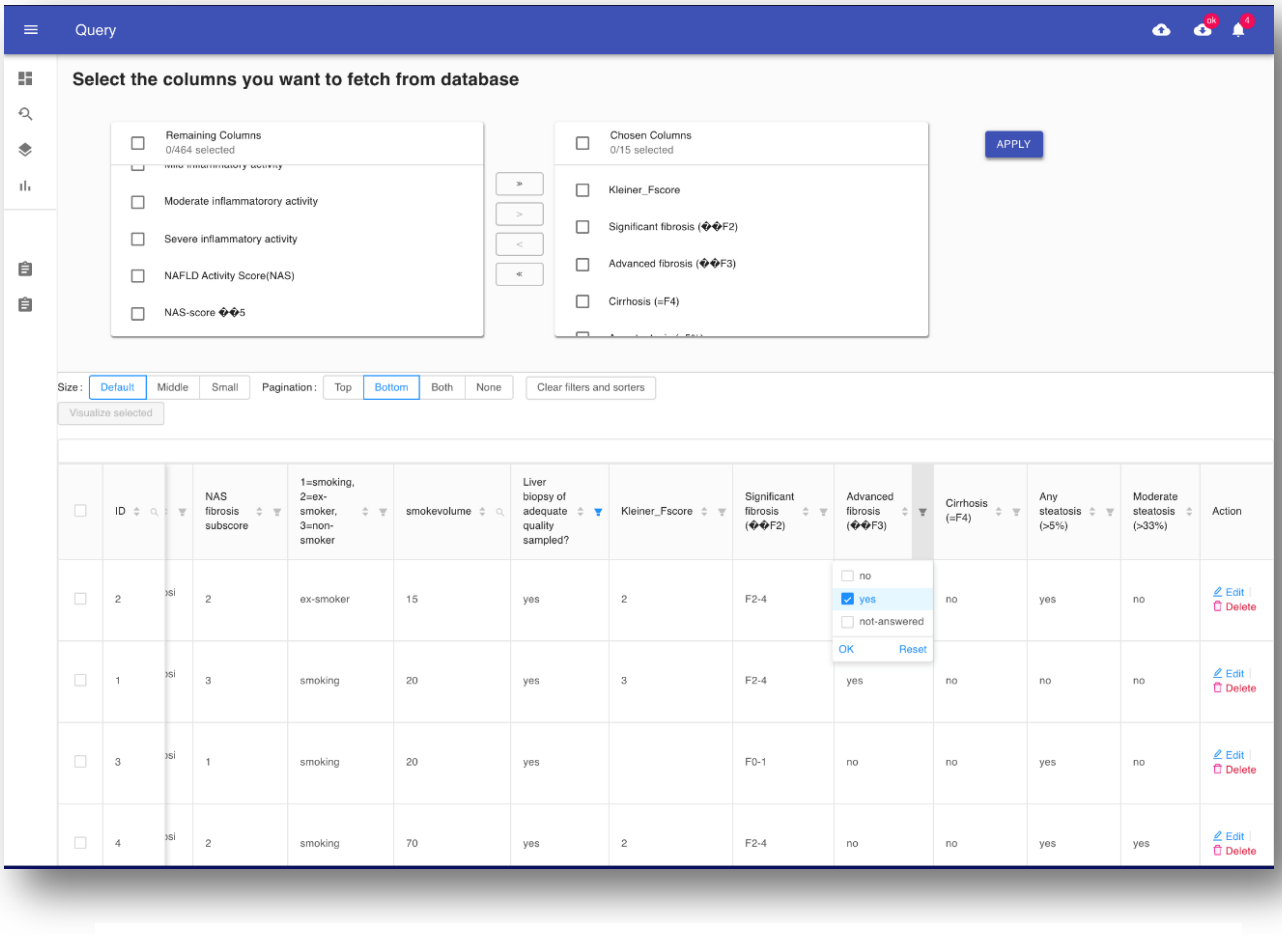
In this section we present the three main screens of the Application that function by sharing the same data and use this data for their unique function, based on user's needs. These screens are Query Screen, Dashboard Screen and Analysis Screen.

5.2.1 Query Screen

Query is the screen of the presentation layer that users use to do one or more of the following processes:

1. Import data to the MongoDB database from .csv files.
2. Import data columns labeling (not required).
3. Define the application's dataset for use, by querying and caching using GraphQL, the imported dataset to filter and fetch only needed columns.
4. Filter the predefined dataset's data.
5. Sort the predefined dataset's data.
6. Search for column-data values in predefined dataset's data.
7. Arrange the data columns by preference.
8. Edit and update the data in the predefined application's dataset.
9. Delete data records from the database.
10. Export the final predefined and updated dataset in .csv format.

The above processes are distinct and don't block one another.



Picture 56: Web application's query screen

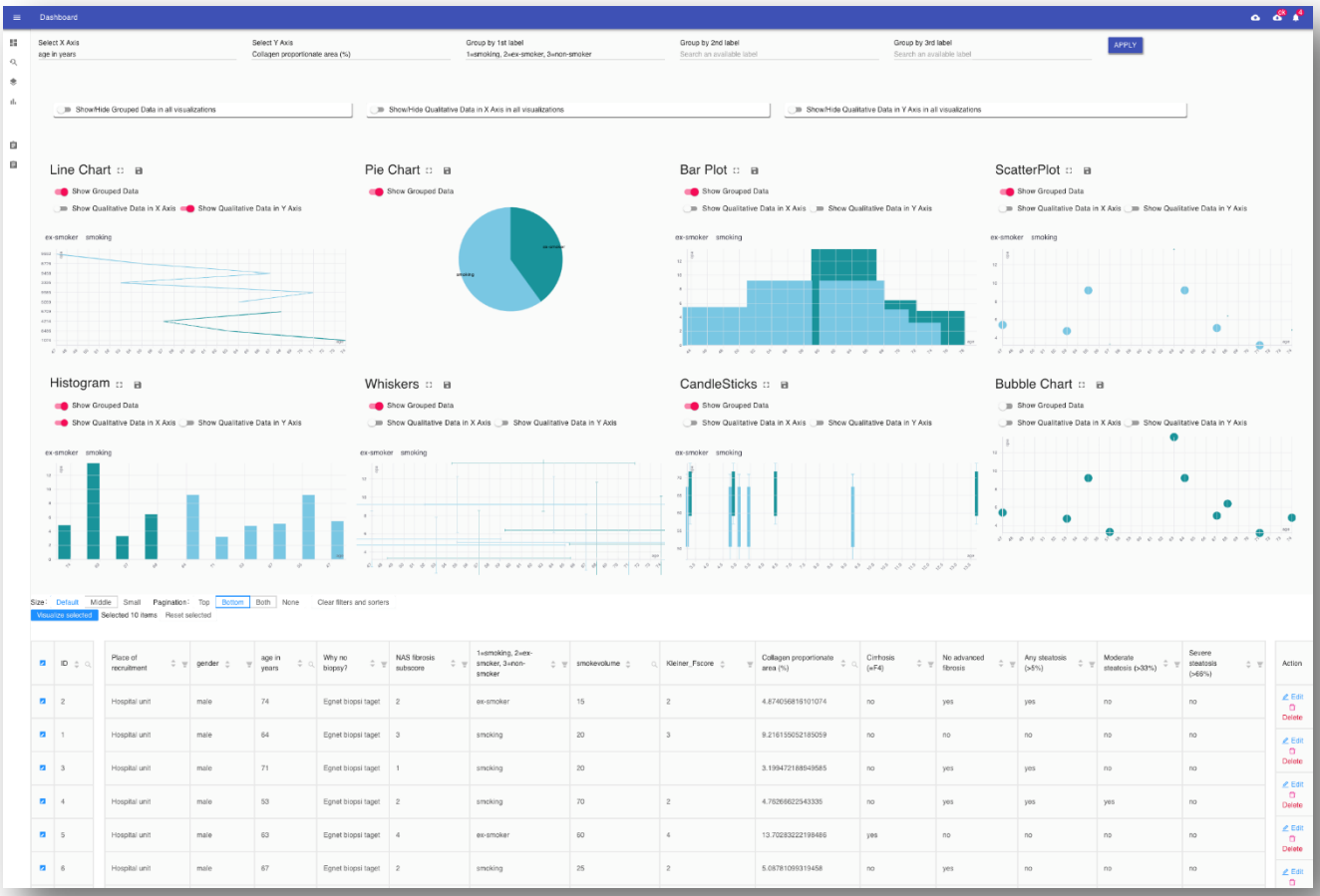
5.2.2 Dashboard Screen

Dashboard Screen which is also the default screen of the application, represents the presentation layer that the users use to do one or more of the following processes:

1. Filter the predefined dataset's data.
2. Sort the predefined dataset's data.
3. Search for column-data values in predefined dataset's data.
4. Arrange the data columns by preference.
5. Edit and update the data in the predefined application's dataset.
6. Delete data records from the database.
7. Export the final predefined and updated dataset in .csv format.
8. Automatically visualize in real-time every table change based on filtering.
9. Custom select certain data-table rows to visualize.
10. Have all different types of visualization charts together in one screen, getting all different facets of the data.
11. Select as x and y axis for all charts whichever column (or labeled column) of the predefined dataset needed.
12. Group by one, two or three different columns the data visualized on x axis giving that way a third, fourth or fifth dimension to every visualization chart.
13. Decide for each chart or all charts together whether the visualizations should show the customly predefined data-groups based on labels or not.
14. Decide for each chart or all charts together whether the visualizations should show the customly predefined X axis data or grouped-data with a qualitative or a quantitative way.

15. Decide for each chart separately or all charts together whether the visualizations should show the customly predefined Y axis data or grouped-data with a qualitative or a quantitative way.
16. Select a visualization chart for full screen usage to extract more visual information, while also have filtering, searching, grouping and the ability to change the axis and labels.
17. Download each chart in .png format either in a full-screen mode with more information included or in a normal mode with the only the basic information.

The above processes are distinct and don't block one another.



Picture 57: Web application's DashBoard screen

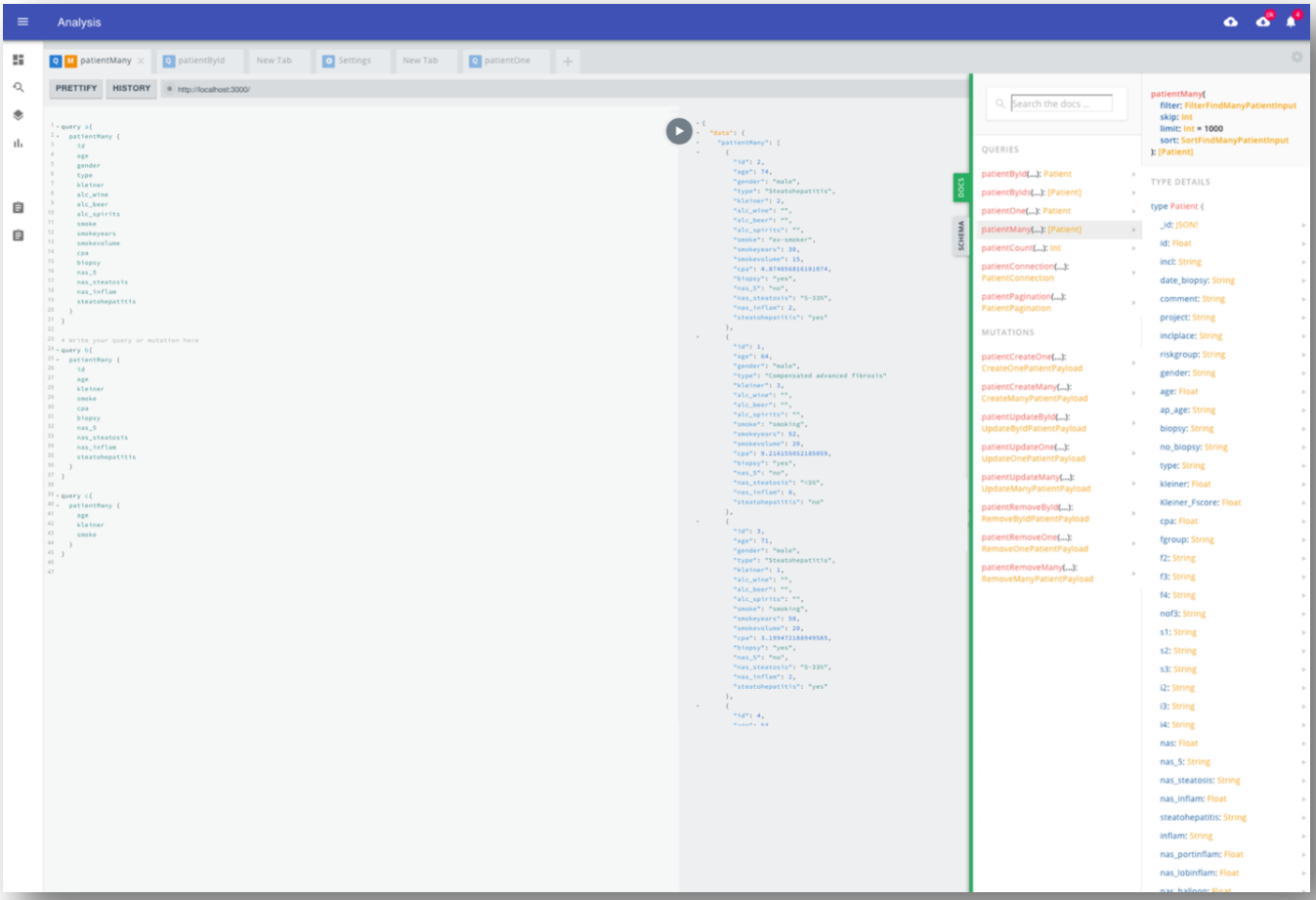
5.2.3 Analysis Screen

Analysis Screen is the screen of the presentation layer that the users use to do one or more or the following processes:

1. Test more complex queries on database's datasets directly without the use of Query screen for faster access and filter applying (suitable for Big Data, multiple collections).
2. Apply multiple GraphQL queries or mutations at once in a queue.
3. Use different queries for fetching data in multiple tabs and compare them.
4. Cache queries for faster fetching when requested from the visualizations.
5. Save data exploration history for later use.
6. Explore the GraphQL data schema.
7. Explore the documentation for all enabled queries or mutations that are allowed for use.

8. Share queries and mutations, history and multiple tabs with collaborators.

The above processes are distinct and don't block one another.



Picture 58: Web application's Analysis screen

5.3 User Cases

In this section we present some use cases for the applications' users, that handle their main concerns. These are; data sharing among users, data discovery using search, data projection and predictability using visualizations, data cleansing, data filtering, data import from csv formatted file, data and filtered data export to csv format, data history saving, data exploration with navigation, data and filtered data visualization for comparisons, composition, understanding, distribution, trend and relations analysis purposes based on criteria, data additions, editing and deletions.

In these use cases we note the ease with which a user (researcher or doctor in this thesis case) can handle all the above data-related concerns based on the pre-application's workload and post-application's workload. For each use case, we present the applied work (and the estimated time to accomplish the task) that has to be done on this web tool and a hypothesis on what could have happened without its existence.

5.3.1 Scenario 1: share data with other users

Before web-application's existence:

Users send different excel versions among one another and this might result in a confusion in regards to the version they are working on.

Using the web-application:
All data exists in the database and it is common for all users, but still the application cannot support simultaneous editing. In a future work, the application will evolve and data should be edited by all users at the same time in order to ensure better efficiency.

5.3.2 Scenario 2: discover data using search

Before web-application's existence:

The application has many labels and for each one we create an index in the beginning (label-1-1-title). There is a capability of visiting indexes and searching by the name of labels in order to find the corresponding column title. By the time we find all the column titles for all the labels, we search for them in the excel file. We can do this one column at a time or create a macro for all search fields.

Using the web-application:

The application has many labels and for each one we create an index in the beginning (label-1-1-title) which can be uploaded in a web-application start. The web-application automatically translates each column title to the corresponding label and uses it in the application. There is a capability of scrolling in the web-application's table and use a search field which allows the selection of rows. After having only the needed rows, you can apply a second search on this rows etc.

5.3.3 Scenario 3: *predict and/or project data using visualizations*

Before web-application's existence:

In order to predict or project data you first filter and manipulate the data in the excel sheet in order to have the final dataset that you want to visualize and then you apply the dataset to the desired chart. If you cannot find a chart that suits the case you want then you ask of someone (developer) to create the visualization chart you want. If the visualization at the end is still not perfect then you start again from the beginning by altering or using a smaller dataset.

Using the web-application:

In the application created for this thesis, you can select data from the web-application's table and every selection is automatically visualized in real-time for all visualization charts that are provided.

5.3.4 Scenario 4: *import data from a csv file*

Before web-application's existence:

There doesn't exist any database for common use among doctors and researchers.

Using the web-application:

Import data in the web application by running an import local script followed by a csv file provided with the web-application.

5.3.5 Scenario 5: *add, edit, or delete data*

Before web-application's existence:

Create a copy of the excel sheet for backup, add/edit/delete rows or columns to the excel sheet. Save file and distribute it to all other collaborators.

Using the web-application:

For all rows, you can download the csv data of the table for backup. Add/edit/delete rows in the web application's table and then click save. After you save the changes all collaborators to the web-application can have access to the new data.

For the columns of the web-application's table in this thesis they cannot be added, edited or deleted. In a future work this could happen, though. But there exist some side-ways that can be done to handle such cases. An easy one is to download the csv data of the table and duplicate the file. Keep the one for backup and edit the other adding the new column with its data. Then, go back to the web-application and upload the data by importing the new edited csv file.

5.3.6 Scenario 6: *filter data*

Before web-application's existence:

Similarly with Scenario 2, the application has many labels and for each one we create an index in the beginning (label-1-1-title). There is a capability of visiting indexes and searching by the name of labels in order to find the corresponding column title. By the time you find all the column titles for all the labels, you can use the excels' filter for the columns that you want. We can do this one column at a time or create a macro for all search fields.

Using the web-application:

The application has many labels and for each one we create an index in the beginning (label-1-1-title) which can be uploaded in a web-application start. The web-application automatically translates each column title to the corresponding label and uses it in the application. There is a capability of scrolling in the web-application's table and use a filter field which filters the table and keeps only the rows that you want. After having only the needed rows, you can apply a second search on this rows etc.

5.3.7 Scenario 7: *clean data*

Before web-application's existence:

You can either write a program that cleans data as described in a previous chapter or select one column at a time and manually check for errors, miswritten values, empty values, etc. Edit them and then save the file. You can send the copy of the saved file to all other collaborators so they can use this version.

Using the web-application:

In the application created for this thesis, you can click in each column filter and search for empty value, or find in filtered lists the "not-answered" annotation. You can filter the table based on this annotation and edit it until no "not-answered" annotations exist. In the same way, if filters in a column have a miswritten value, you can select this filter to keep only the rows with this miswritten text or number and edit it. Eventually, you save the changes and then all the other users have access to this changes due to the common database.

5.3.8 Scenario 8: *export data or filtered-data to a csv file*

Before web-application's existence:

A user can duplicate a file and keep it for backup, then he can make changes to excel data or filter the data and finally save the file in csv format.

Using the web-application:

By clicking the "Download in .c csv format" button the data that are present in the Table are downloaded in .csv format.

5.3.9 Scenario 9: *explore data using navigation*

Before web-application's existence:

Data in excel are plain text and cannot be used for exploration easily. What is more, in this thesis case there are more than 460 different columns and many of them consists of rows filled with numbers. It is very easy for users to confuse and to miss something, while also header title keys in excel is very uncomfortable to follow and understand, and so, as we described also in Scenario 2 and Scenario 6 users what to go back and forth to check their column-title to labels index in order to understand the data.

Using the web-application:

This thesis web-application's interface is based on User Experience(UX) principles and was evaluated with all the lab's researchers at this time period. Users can navigate with ease to all different screens and manipulate the data available in the interface each time by their preference, while also help themselves with the real-time visualizations in their data exploration. Furthermore, users can now go in the Analysis Screen and select only the columns that they want to visualize (which also helps in speeding up the page data-loading). In this way, they don't have to scroll amongst 460+ columns but only the needed ones, while also they can rearrange the columns in their preference possibly to observe some visual patterns in data. What is more, to even strengthen more this powerful web-tool as described in Scenario 2 and Scenario 6, now all column-titles are replaced everywhere within the interface with more user-friendly labels that every user can upload and use. All the above, result in a simple and easy to use web interface.

5.3.10 Scenario 10: *compare, compose, understand, distribute, analyze relations and trends on data or filtered data visualizations based on criteria*

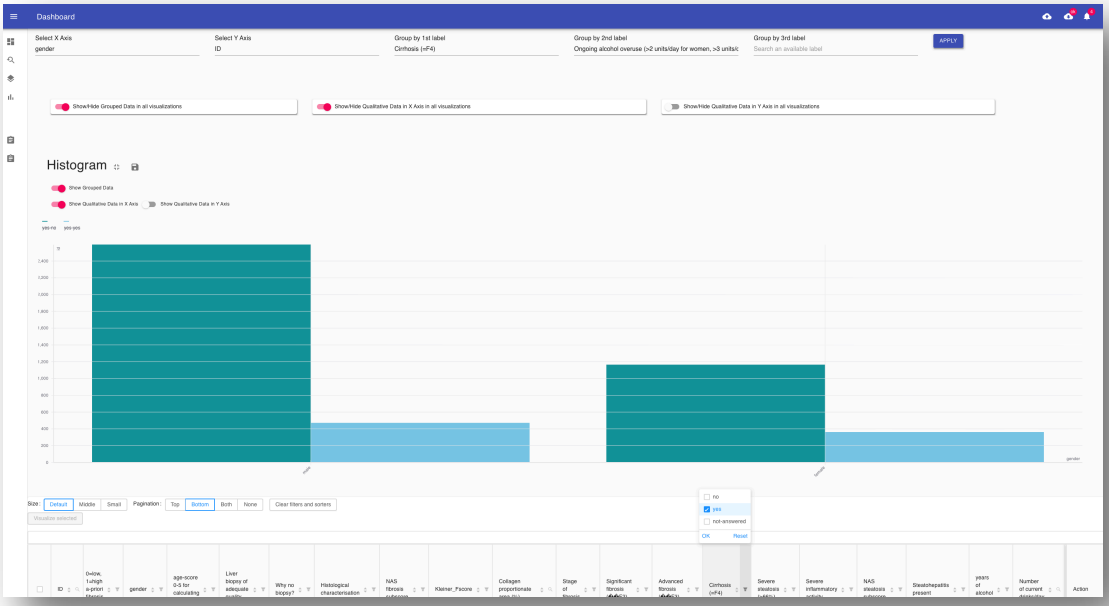
Before web-application's existence:

Data in excel are plain text and cannot be used for research purposes using visualizations at once without first preprocessing them, until their data-shape is applicable as an entry to visualization charts. So we decided to describe it with an example. For instance, in Chapter 3.5.2, it is argued that *"Alcohol is the main risk factor of cirrhosis in Europe, where 1.8% of all deaths are attributable to liver disease. Although alcohol per se is the most important risk factor for alcoholic cirrhosis, only about 35% of heavy drinkers develop the disease and there is not a clear dose response pattern. Moreover, even light drinkers, who consume one to two drinks a day, are at increased risk of alcoholic cirrhosis compared to abstainers. Alcohol use is therefore a bad predictor for the development of liver cirrhosis. Genetic and environmental risk factors do also not explain the substantial inter individual differences in susceptibility to ALF..". Also, a few sections later, in Chapter 3.6.1., it is argued that: " In an Italian study by Bellentani et al. there was a 9:1 ratio of men with cirrhosis in the general population. However, 13 to 33% of Americans who are either abusing alcohol or depend on it are women.."*

So, to evaluate these statements the researchers and doctors have to filter in excel the data by gender and start with the males, then they have to filter again by alcoholic cirrhosis and compare the total males that have alcoholic cirrhosis to a part of them that are also heavy drinkers. Then, having done so, they need to apply the same filters for females, in the initial excel data, in order to combine the two above statements. At follow, they gather all the 4 custom dataset (male with cirrhosis – male with cirrhosis and heavy drinkers - female with cirrhosis – female with cirrhosis and heavy drinkers) and before visualizing the data using Excel charts for this use case, they would expect that the total cirrhosis data sample would consist of more men than women, but the percentage of women that have both cirrhosis and are also heavy drinkers would be bigger than that of men. Finally, they visualize the data to find out if such a case applies also to these data. Unfortunately, in order for users to experiment, verify and explore these types of simple cases this job is time consuming and many times results to unexpected and misunderstood solutions.

Using the web-application:

This thesis web-application's interface on the contrary is created by the will to present and visualize data in real-time. Considering this, the above use-case experiment can very simply be elaborated, as the users only need to select the X and Y chart-axis which they want to visualize, and group the visualized data by Cirrhosis and by Alcohol Overuse (>2 units/daily for women and >3 units/daily for men). To filter the data even more, they also use the data for those that are Positive to Cirrhosis.



Picture 59: Use-case experimentation for verifying statements in Cited Publications

Truthfully, we can see that both males with Cirrhosis are more than females, and also cirrhosis in females may be more relevant to alcohol abuse than cirrhosis in males.

5.3.11 Scenario 11: *cache data for reuse and save history*

Before web-application's existence:

Users didn't have a database for caching or saving data, all history is kept by using file duplication. Users create backup copies for keeping track of the changes.

Using the web-application:

Using the database, users are capable to keep track of the database changes but at this time they cannot access these changes, only the administrator can access them. The reason is that there is not any screen for users implemented to access these data. And this will be one of the first features to be added in future. Nevertheless, caching of similar data responses to queries is implemented, while also in Analysis screen a user can save queries for personal use and access them again in future using the history tab.

5.4 Competition

In this section we present several applications that exist at the moment in the market of "data-visualization-software". These applications are proposed by Capterra[65] which is a "Applications Software" Search Engine with a good reputation. Capterra's main function

[65] https://www.capterra.com/

is to provide users the ability to easily find software that is related to their needs, simply by querying a keyword or browsing into its categories. What is more Capterra provides users with main descriptions for each application found and user reviews for rating purposes.

For this thesis purposes we searched for the keywords "data" and "visualization" and we found the category "Data Visualizations Software" which is where our web-application also classifies. Then, for purposes of a fair product comparison we selected from the filter side-menu the filters "Analytics", "Dashboard creation", "Visual Discovery", and "Web-Based". Finally after sorting based on highest rated software we came up with the following list of top-5 products: CanvasJS Charts[66], Informer[67], DataPlay[68], CleverAnalytics[69], and Worksheet Systems[70]. Below, we present a comparison amongst our web-application and these five products. [82]

Table 5: 2019 comparison with other data visualization software of competition

	CanvasJS Charts	Informer	DataPlay	CleverAnalytics	Worksheet Systems	Thesis web-tool
Use common database	✅	✅	✅	✅	✅	✅
Database location (for privacy concerns)	Remote	Remote	Remote	Remote	Remote	Remotely and/or in-house
Code editing allowed	✅	❌	❌	❌	❌	✅
Responsive web application	✅	✅	❌	✅	❌	✅
Price/Free	$399.00/year/user, Free trial 30 days	-	$140.00/month/user, Free trial 30 days	$499.00/month/10-users, Free trial 15 days	£100.00/month/user, Free trial 15 days	Free

[66] https://canvasjs.com/

[67] http://www.entrinsik.com/

[68] https://www.margasoft.com/

[69] https://www.cleveranalytics.com

[70] https://www.worksheet.systems/

Platforms	Cloud-SaaS-Web, Mac, Windows, Android, iOS	Cloud-SaaS-Web, Mac, Windows	Cloud-SaaS-Web, Windows	Cloud-SaaS-Web	Cloud-SaaS-Web	Cloud-SaaS-Web
Storage	Limited	Limited	Limited	Limited, up to 2GB	Lmited, up to 100,000 records and 50 data tables	Unlimited
Real-time charts on data change	✅	✅	✅	✅	✅	✅
Dashboard Content management	❌	❌	✅	❌	❌	✅
Filtered database views	❌	✅	❌	❌	❌	✅
Visual discovery using Dashboard	✅	✅	❌	❌	✅	✅
Data file types supported import	>2	>2	>2	>2	2	1
Data filetypes supported export	>2	>2	>2	>2	2	2
Number of visualizations offered	30	>10	>10	5-10	5-10	5-10
Data filtering	✅	✅	✅	✅	✅	✅
Support	Online-Business Hours	Online-Business Hours	24/7	Online-Business Hours	Online-Business Hours	24/7

Creator	Fenopix Technologies	Entrinsik	Margasoft	CleverAnalytics	FalconSoft	
Open API	✅	❌	❌	❌	❌	✅
Import Table Labels representation	❌	❌	❌	❌	❌	✅
Database History Changes View	❌	❌	❌	❌	❌	❌

From the above table we can infer that there is no-perfect tool in competition and that there is a lot of work needed to be done in this field as the users' needs always increase, and there should be tools created to fill in these needs. We can also infer that for our use case the web-application implementation was an one-way to follow, as most of the tools that do what our researchers and doctors want are paid, and the ones that are free (not enclosed in the table above) have many disadvantages or do not have that much features as our users want So they preferred to use Excel. Furthermore, Data Privacy Concerns, are ensured as the database and all the computers' network are all running inside the BRFAA intranet. In case we decide to use the docker in a cloud based platform then we should revisit such issues.

In addition to the above, an advantage of our web-application creation is that we can decide about what changes we want to implement and we can prioritize them first and develop them in-house, i.e. we can develop whatever custom charts based on our users' needs, we can add A.I. and use deep learning inference algorithms, we can scale out the application based on our storage, memory and processors needs of use, all costs that occur are development costs without any middlemen, or because we have ownership permissions on the application, if it grows larger it could even spin off the lab.

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The main purpose of this thesis was to create a data visualization web-tool that would be connected to a database with survey, clinical, and laboratorial data taken from patients of Alcoholic Liver Disease. To do so, we decided to experiment and use the state-of-the-art web technologies that are also used in the biggest companies worldwide and are supported by the community and by international companies such as Amazon, Google and Facebook, and in order to succeed in our goal we followed by the book the software development lifecycle and versioning, using the Kanban model.

In our tech stack we started from a technical background software architecture theory that suggest to use separation of concerns with n-tiers (data persistence, data access, business-logic and presentation layer) and we implemented this theory in practice step by step using the most recent and emerging web-technologies stack and the best design patterns for scalability, reusability, maintainability and performance applied in these technologies.

Our technological stack consisted of MongoDB, Node.js, GraphQL and ReactJS, each one of them used for each one of the 4-tier architecture of our web-application. Of course, in this web application would mean nothing if it didn't fulfill its purpose, which is to visualize the database's data in user-friendly and handy ways on a nice user interface that uses good UX principles. So, three of the most important UI libraries that correspond to the largest part of the interface were Material-UI that follows the Google Material Design pattern for UI and UX purposes, Ant Design that provided as with a great React component for Table representation and react-vis made by Uber, that provided us with all the different types of data visualization charts.

Then, having glue together all these different technologies and libraries and having handled with a centralized manner all of the data processing and filtering for visualization purposes, our web-application could start get tested by our users. We discussed with supervisors 11 interesting use cases and tried to compare for each use case the steps a user should follow to do these 11 casual scenarios before and after this web-tool creation. The results were very satisfying, as it seems that the web-tool may save more than an hour in a daily basis to each user that is using it, because it provides feature such as: labeling columns with user-friendly titles, easy column and multiple column filtering with also multiple options criteria, real-time visualizations in several charts, common database with all other users - no share needed, export filtered data in csv format and visualizations in png format, responsive web-application that loads in every device type, and more.

Finally, we challenged ourselves and made a comparison with some of the best tools nowadays for data-selection and visualization and found out that there is no-perfect tool in competition and that there is a lot of work needed to be done in this field as the users' needs continuously increase, and there should be tools created to fill in these needs. We can found out that the good software for data visualizations is mostly paid and this is why many researchers are still using MS Excel.

As a result, we concluding that the creation of this a web-application tool was an one-way to go, as this way we don't need to entangle with Data Privacy Concerns as the database and all the computers network that connect to its data are all running inside the BRFAA intranet and because we want to provide the researched more effective time to work on cases that they are really concerned.

To sum up, this bioinformatic tool will help physicians and researchers to simplify the process daily of data selection, analysis and exploration by using visualization charts and

data filtering. As a result, the tool facilitates the day-to-day physicians and researchers schedule to focus more on the essence of research, i.e. to draw conclusions about the main categories of information that lead patients to alcoholic liver disease, and less on processes.

6.2 Future Work

This web-application tool as it was also described in the conclusions will always need more features as the users' demands increase continuously, and the competition is also growing. In this thesis we implemented almost the 95% of our initial plan, as one can find out in our Kanban board (the 5% that we didn't implement was because of requirements changes). Software planning can never be set in advance as software is everyday changing. This is also another metric, as we finally, so far added almost a 20% bigger load in the backlog than the initial calculated and expected. This means that while the thesis had started, in grooming processes every 2-weeks we thought about and finally added some more feature in the backlog that either we hadn't thought from the beginning, or were software impediments that we could continue without implementing them, or were nice to have and easy to implement and so we added them. Nonetheless, this thesis came to an end, or better a checkpoint. This web-application is fully functional and supports what was aforementioned in this thesis document. Therefore, while reading this thesis there are notes scattered in text that refer to some future work that could be done to optimize or extend some processes. These future work proposals are listed below, in an unordered list:

- apply machine learning to find important relations among data characteristics
- add more visualization types, distributions, ROC curves, etc.
- use docker virtualization
- collaborative editing. This can happen by adding GraphQL subscriptions which use websockets. This way concurrent users can edit in database and data are updated in all other clients
- move data caching and history saving to backend
- add user roles (user profiling and edit tracking)
- use a service worker for offline data visualization and cache data in browser
- group by more than three labels with a multiple selection field
- add/edit/delete column data
- keep track of history changes in database

7. ABBREVIATIONS - ARCTICS - ACRONYMS

ALD	Alcoholic Liver Disease
CLD	Chronic Liver Disease
HBV	Hepatitis B Virus
HCV	Hepatitis C Virus
NASH	Non-alcoholic steato-hepatitis
HCC	Hepatocellular carcinoma
NAFLD	Non-alcoholic fatty liver disease
ALF	Alcoholic liver fibrosis
GDPR	General data protection regulation
BRFAA	
SDU	University of Southern Denmark
SDLC	Software development life-cycle
SSADM	structured systems analysis and design method
API	Application Programming Interface
ARA	Application release automation
SaaS	Software as a service
PaaS	Platform as a service
IaaS	Infrastructure as a service
CRM	Customer Relationship Management
HRMS	Human Resource Management System
XP	Extreme Programming
RAD	Rapid-application development
PDCA	Plan-do-check-act
SoC	Separation of Concerns
DBMS	Database management system
SQL	Structured query language
ACID	Atomicity, Consistency, Isolation, Durability
JSON	JavaScript Object Notation
BSON	Binary JSON
RDBMS	Relational DBMS
TSC	Technical Steering Committee
SDL	Schema Definition Language
SPA	Single Page Application

8. BIBLIOGRAPHY

[1] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Liver.

[2] [Online]. Available: https://www.bartleby.com/107/250.html.

[3] H. G. e. al., Gray's Anatomy: The Anatomical Basis of Medicine and Surgery, 1995.

[4] G. Fornari, The Body Athlas Steve Parker, 1993.

[5] J. E. H. Arthur C. Guyton, Textbook of Medical Physiology, 1995.

[6] V. G. Filippa, "Bioinformatic Analysis of Clinical and Molecular Patient Data with Non-Alcoholic Fatty Liver Disease and Implementation of Statistical Methods in Co-Expression Networks," Athens, 2018.

[7] [Online]. Available: https://www.webmd.com/digestive-disorders/picture-of-the-liver#1.

[8] [Online]. Available: https://www.stanfordchildrens.org/en/topic/default?id=anatomy-and-function-of-the-liver-90-P03069.

[9] N. P.-F. C. F.-F. C. D.-G. C. P.-G. María M. Adeva-Andany, "Liver glucose metabolism in humans," *Biosciences Reports,* vol. 36, no. 6, 2016.

[10] B. E. Roden M, "Hepatic glucose metabolism in humans-its role in health and disease.," *PubMed,* vol. 3, pp. 365-383, 2003.

[11] A. P. Y.-J. H. Frayn KN, "Fatty acid metabolism in adipose tissue, muscle and liver in health and disease.," *Essays Biochem,* vol. 42, pp. 89-103, 2006.

[12] D. B. Jump, "Fatty acid regulation of hepatic lipid metabolism," *PMC ,* 2012 .

[13] "VIVO Pathophysiology," [Online]. Available: http://www.vivo.colostate.edu/hbooks/pathphys/digestion/liver/metabolic.html.

[14] C. MR, "Protein metabolism and liver disease.," *Baillieres Clin Endocrinol Metab,* pp. 617-635, 1996.

[15] R. University, "BC Open Textbooks," [Online]. Available: https://opentextbc.ca/anatomyandphysiology/chapter/24-4-protein-metabolism/.

[16] "Hepatitis C Trust," [Online]. Available: http://www.hepctrust.org.uk/information/liver/protein-synthesis.

[17] L. P. De Feo P, "Liver protein synthesis in physiology and in disease states," *Curr Opin Clin Nutr Metab Care,* pp. 47-50, 2002.

[18] M. EF, "Coagulation abnormalities in liver disease," *Hematol Oncol Clin North Am,* pp. 1247-1257, 1992.

[19] T. A. Woreta, "John Hopkins Medicine," [Online]. Available: https://www.hopkinsmedicine.org/health/wellness-and-prevention/detoxing-your-liver-fact-versus-fiction.

[20] "Medicine Plus," [Online]. Available: https://medlineplus.gov/lab-tests/bilirubin-in-urine/.

[21] G. K. Michalopoulos, "Liver Regeneration," *J Cell Physiol ,* 2009.

[22] B. K. K. Patrick Marcellin, "Liver diseases: A major, neglected global public health problem," *Liver International,* 2017.

[23] B. K. K. Patrick Marcellin, "Liver diseases: A major, neglected global public health problem requiring urgent actions and large-scale screening," *Liver International,* 2018.

[24] R. G. R. S. M. Massimo Pinzani, "Progression of fibrosis in chronic liver diseases: time to tally the score," *Journal of Hepatology,* 2001.

[25] K.-K. Q.-X. L. C. H. N. L. J.-M. S.-X. B. Q. C. M.-Q. &. M.-Q. Sheng-Sen Chen, "Factors associated with significant liver necroinflammation in chronic hepatitis B patients with cirrhosis," *Nature,* 2016.

[26] S. D. A. J. M. e. a. Robert S. O'Shea, "Alcoholic liver disease," *Hepatology,* 2009.

[27] M. M. M. J. M. M. a. T. R. M. M. Sarathy Mandayam, "Epidemiology of Alcoholic Liver Disease," 2004.

[28] J. T. D. L. G. Askgaard, "A measure of alcohol consumption in late adolescence associated with liver disease after 39 years of follow-up is insufficient to guide alcohol safe limits," *Journal of Hepatology* .

[29] P. T. M. D. J. P. a. K. K. K. P. Natalia A. Osna, "Alcoholic Liver Disease: Pathogenesis and Current Management," *Alcohol Research,* pp. 147-161, 2017.

[30] M. K. V. NARAYANAN MENON, M. GREGORY J. GORES and M. AND VIJAY H. SHAH, "Pathogenesis, Diagnosis, and Treatment of Alcoholic Liver Disease," *Mayo Clin Proc,* 2001.

[31] R. B. BIN GAO, "Alcoholic Liver Disease: Pathogenesis and New Therapeutic," *Gastroenterology.,* 2011.

[32] G. Szabo, "Gut–Liver Axis in Alcoholic Liver Disease," *Gastroenterology,* 2015.

[33] V. P. a. E. A. S. Georgiou, "The Role of microRNAs in the Gut-Liver Axis," in *The Human Gut-Liver-Axis in Health and Disease,*, 2019, pp. 207-234.

[34] P. M. a. R. Bataller, "Trends in the management and burden of alcoholic liver disease," *J Hepatol.,* pp. 38-46, 2015.

[35] "European Union," [Online]. Available: https://europa.eu/youreurope/citizens/consumers/internet-telecoms/data-protection-online-privacy/index_en.htm.

[36] "Learn.org," [Online]. Available: https://learn.org/articles/What_is_Database_Architecture.html.

[37] "Stackify," 2017. [Online]. Available: https://stackify.com/what-is-sdlc/.

[38] "technopedia," [Online]. Available: https://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc.

[39] "raygun," [Online]. Available: https://raygun.com/blog/software-development-life-cycle/.

[40] "Apprenda," [Online]. Available: https://apprenda.com/library/cloud/deployment-to-the-cloud/.

[41] "deploy partners," [Online]. Available: https://deploypartners.com/security-and-maintenance-considerations-for-the-cloud/.

[42] "Cloud Codes," 2018. [Online]. Available: https://www.cloudcodes.com/blog/cloud-and-docker-cloud.html.

[43] "Open Source," [Online]. Available: https://opensource.com/business/14/7/why-docker-new-craze-virtualization-and-cloud-computing.

[44] "open source," [Online]. Available: https://opensource.com/resources/what-docker.

[45] "tryqa," [Online]. Available: http://tryqa.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/.

[46] "cprime," [Online]. Available: https://www.cprime.com/resources/what-is-agile-what-is-scrum/.

[47] "ambysoft," [Online]. Available: http://www.ambysoft.com/essays/agileLifecycle.html.

[48] "luis-goncalves," [Online]. Available: https://luis-goncalves.com/what-is-agile-methodology/.

[49] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Liver .

[50] "guru99," [Online]. Available: https://www.guru99.com/waterfall-vs-agile.html.

[51] H. D. O. M. A. G. Y. I. Orfeas Aidonopoulos, "DCV: From cleaning to analytics through WEB based interactive".

[52] M. K. Y. I. Anna Gogolou, "Data exploration: a roll call of all user-data interaction functionality," in *the Third International Workshop*, 2016.

[53] A. Gogolou, "Exploration and Data Cleansing in Biomedical Bases," Athens, 2016.

[54] "DCV curation tools and services to automatically and manually acquire high-quality curated data," Athens, 2015.

[55] "Software Testing Material," [Online]. Available: https://www.softwaretestingmaterial.com/software-architecture/.

[56] M. S. J. H. Joseph M. Hellerstein, "Architecture of a database," *The essence of knowledge,* vol. 1, no. 2, pp. 141-259, 2007.

[57] "Medium," [Online]. Available: https://medium.com/oceanize-geeks/concepts-of-database-architecture-dfdc558a93e4.

[58] R. C. a. L. M. Nadeem Qaisar Mehmood, "Modeling temporal aspects of sensor data for MongoDB NoSQL database," *Journal of Big Data,* vol. 4, no. 8, 2017.

[59] "technopedia," [Online]. Available: https://www.techopedia.com/definition/1405/back-end-system.

[60] "IBM," [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdbusinesslogicbase.htm.

[61] S. C. a. B. Straub, Pro Git, 2014.

[62] "MongoDB," [Online]. Available: https://www.mongodb.com/.

[63] "MDN web docs mozilla," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.

[64] "MDN web docs mozilla," [Online]. Available: https://developer.mozilla.org/el/docs/Web/JavaScript.

[65] "The Linux Foundation Projects," [Online]. Available: https://www.linuxfoundation.org/projects/.

[66] "Foundation nodejs," [Online]. Available: https://foundation.nodejs.org/.

[67] "readwrite," [Online]. Available: https://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/.

[68] "Facebook Code," [Online]. Available: https://code.fb.com/core-data/graphql-a-data-query-language/.

[69] "Techcrunch," [Online]. Available: https://techcrunch.com/2018/11/06/facebooks-graphql-gets-its-own-open-source-foundation/?renderMode=ie11.

[70] "Apollo QraphQL," [Online]. Available: https://www.apollographql.com/.

[71] "Apollo GraphQL," [Online]. Available: https://www.apollographql.com/docs/apollo-server/.

[72] "ITNEXT," [Online]. Available: https://itnext.io/graphql-mongoose-a-design-first-approach-d97b7f0c869.

[73] "GraphQL," [Online]. Available: https://graphql.org/learn/queries/#variables.

[74] "Medium," [Online]. Available: https://medium.com/fbdevclagos/understanding-graphql-queries-mutations-and-subscriptions-a80a8b5c877c.

[75] "c-sharp corner," [Online]. Available: https://www.c-sharpcorner.com/article/what-and-why-reactjs/.

[76] "itnext.io," [Online]. Available: https://itnext.io/using-advanced-design-patterns-to-create-flexible-and-reusable-react-components-part-3-render-d7517dfe72bc.

[77] "Github Page," [Online]. Available: https://nmpegetis.github.io/eurobank-training/.

[78] M. R. E. B. I. E. Alex Fishman, "HVX: Virtualizing the Cloud," *Ravello Systems.*

[79] "Docker," [Online]. Available: https://docs.docker.com/engine/docker-overview/.

[80] "t4tutorials," [Online]. Available: https://t4tutorials.com/software-maintainability-in-software-engineering/.

[81] "e-zest," [Online]. Available: https://blog.e-zest.com/why-reusability-of-software-components-is-essential.

[82] "benmccormick," [Online]. Available: https://benmccormick.org/2019/02/11/reusable-react.

[83] "sequetech," [Online]. Available: https://www.seguetech.com/what-is-software-performance-testing/.

[84] "eagereyes," [Online]. Available: https://eagereyes.org/criticism/definition-of-visualization.

[85] "microstrategy," [Online]. Available: https://www.microstrategy.com/us/resources/introductory-guides/data-visualization-what-it-is-and-why-we-use-it.

[86] "weforum," [Online]. Available: https://www.weforum.org/agenda/2017/09/the-value-of-data/.

[87] "microstrategy," [Online]. Available: https://www.microstrategy.com/us/resources/introductory-guides/data-visualization-what-it-is-and-why-we-use-it.

[88] "Hubspot," [Online]. Available: https://blog.hubspot.com/marketing/types-of-graphs-for-data-visualization.

[89] "Fusion Charts," [Online]. Available: https://www.fusioncharts.com/resources/chart-primers/box-and-whisker-chart.

[90] "Capterra," [Online]. Available: https://www.capterra.com/sem-compare/data-visualization-software?gclid=CjwKCAjwr8zoBRA0EiwANmvpYO3QKOxYoSXnHeFfvAbuGxoQeoBhRHRMcLvsJ8qwqBNYFB5DbI3fnhoCsbAQAvD_BwE.

[94] "Medium," [Online]. Available: https://medium.com/@nmpegetis/git-how-to-start-code-changes-commit-and-push-changes-when-working-in-a-team-dbc6da3cd34c.

[95] "Github," [Online]. Available: https://github.com/getify/You-Dont-Know-JS.

[96] "Medium," [Online]. Available: https://medium.com/software-insight/graphql-types-and-relationships-cbb046a541c4.

[97] "Github," [Online]. Available: https://github.com/graphql-compose/graphql-compose-mongoose.

[98] H. D. J. E. P. S. K. Sumeet K. Asrani, "Burden of liver diseases in the world," *Journal of Hepatology,* 2018.

[99] A. P. B. S. Shiv K. Sarin, "Microbiome as a therapeutic target in alcohol-related liver disease," *Journal of Hepatology,* 2018.

[100] M. M. M. J. M. M. Sarathy Mandayam, "Epidemiology of Alcoholic Liver Disease," 200.

[101] B. K. K. Patrick Marcellin, "Liver diseases: A major, neglected global public health problem," *Liver International,* 2017.

[102] "Software Testing Material," [Online]. Available: https://www.softwaretestingmaterial.com/software-architecture/.

[103] "Software testing material," [Online]. Available: https://www.softwaretestingmaterial.com/software-architecture/.

[104] "Medium," [Online]. Available: https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4.