UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

telecos
BCN

Foodie environment MEAN web application

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Alicia García Sanz

In partial fulfilment

of the requirements for the degree in

TELECOMMUNICATION ENGINEERING

Advisor: Marcel Fernandez Muñoz

Barcelona, June 2021

Abstract

Currently, human beings are constantly in a rush. Nobody has the time to be conscious of their eating habits. This thesis aims to unify the cooking responsibilities on a website. Hence, the user is in control of the food intake while saving time. A MEAN project is developed to accomplish such a purpose, a web application based on an Angular client-side. Then, the resources come from a MongoDB database connected through an Express server-side. Everything is in a NodeJS runtime environment.

Thus the result is a Single Page Application. The user can browse through any recipe in the database. To write personal recipes is an option too. Planning a weekly menu, writing the grocery list or having a food expenses imprint are the different features available. Therefore, it unifies the cooking duties in a highly responsive, multiplatform application.

Resum

Actualment, els humans viuen en tensió constant. Ningú té temps de ser conscient dels seus hàbits alimentaris. Aquesta tesi pretén unificar les responsabilitats lligades a la cuina en una aplicació web. D'aquesta manera, l'usuari controla la seva ingesta alimentària i estalvia temps. Un projecte MEAN és desenvolupat per aconseguir aquest objectiu, una web amb un client formulat en Angular. Seguidament, els recursos són obtinguts d'una base de dades de MongoDB connectada a través d'un servidor implementat en Express. L'entorn d'execució és NodeJS.

Per tant, el resultat és una aplicació de pàgina única. L'usuari pot consultar qualsevol recepta de la base de dades o escriure una de nova. Les diferents funcionalitats són planificar un menú setmanal, fer la llista de la compra o tenir un seguiment de les despeses en l'entorn alimentari. Aconseguint així unificar les responsabilitats lligades a la cuina en una aplicació multiplataforma, altament responsiva.

Resumen

Actualmente, los humanos viven en tensión constante. Nadie tiene tiempo para ser consciente de sus hábitos alimenticios. Esta tesis pretende unificar las responsabilidades en el entorno de la cocina. De esta forma, el usuario tiene el control sobre su ingesta alimentaria mientras ahorra tiempo. Un proyecto MEAN es desarrollado para cumplir este propósito, una aplicación web basada en un cliente formulado en Angular. Seguidamente, los recursos son extraídos de una base de datos implementada en MongoDB conectada a través de un servidor definido con Express. El entorno de ejecución utilizado es NodeJS.

Por lo tanto, el resultado es una aplicación de página única. El usuario puede consultar todas las recetas de la base de datos, o aportar nuevas personales. Las diferentes funcionalidades también son planear menús semanales, escribir la lista de la compra o llevar un seguimiento de los gastos en comida. De esta forma, se unifica todo el entorno alimenticio en una aplicación multiplataforma, altamente receptiva.

Revision history and approval record

Revision	Date	Purpose
0	19/03/2021	Document creation
1	04/04/2021	Document revision
2	27/04/2021	Document revision
3	11/05/2021	Document revision
4	01/06/2021	Document revision
5	16/06/2021	Document revision
6	21/06/2021	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Alicia García Sanz	alicia.garcia.sanz@estudiantat.upc.edu
Marcel Fernandez Muñoz	marcel.fernandez@upc.edu

Written by:		Reviewed and approved by:	
Date	19/03/2021	Date	21/06/2021
Name	Alicia García Sanz	Name	Marcel Fernandez Muñoz
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

Glossary

REST REpresentational State Transfer

API Application Program Interface

SPA Single Page Application

CRUD Create Read Update Delete

MEAN MongoDB Express Angular NodeJS

CLI Command-line interface

HTML HyperText Markup Language

CSS Cascading Style Sheet

JSON JavaScript Object Notation

JWT JSON Web Token

NPM Node Package Manager

IDE Integrated Development Environment

VSC Visual Studio Code

HTTP Hypertext Transfer Protocol

URL/URI Uniform Resource Locator / Uniform Resource Identifier

SQL Structured Query Language

MVC Model View Controller

KDF Key Derivation Function

DOM Document Object Model

1. Introduction and Statement of purpose

Nowadays, people are not conscious of their eating habits. They regularly skip meals and fall for delivery or fast-food services. Eating is a survival habit because any living being necessitates food intake to stay alive. Most importantly, we are what we consume. The purpose of the thesis is to conjoin all the work related to cooking.

A website has been developed to help people organize their weekly menus. The website presents a unified workspace food environment, where recipes are shared automatically. Hence, the website is creating an inspiring space for each user. Those recipes can be saved individually in the users' recipes book.

Once those recipes are stored or inscribed, it facilitates the user to determine the several meals of the week. There is an area to plan various weekly menus identifying the recipes and the food intake. Moreover, the user can integrate batch cooking. He or she can reserve time to prepare multiple meals for the whole week. Thus, saving time, energy and money.

Additionally, it is plausible to automatically create a grocery list with ingredients selected from the recipe's ingredients, editable on demand. Finally, there is an economic imprint for the user to control the food expenses if wished.

Hence, the application creates a little community of people sharing recipes, allowing others to discover new ideas or even new flavors. Additionally, it helps organize and save time while cooking and grocery shopping, which also saves money.

The full-stack project requires to fulfil the services introduced:

- To have a RESTful API with a scalable database and a highly responsive interface. The application web speed must be less or equal to five seconds because increasing that time loses the user's attention.
- The website must be organized, easy to use and understandable. It must have a clean and user-friendly interface.
- At least the CRUD (Create Read Update and Delete) operations must be contained in the different data models.
- Lastly, understand and master the technologies used in the project.

The implementation to achieve the requirements presented is shortly described. The database models that will be detailed in section 5 have been designed, taking into account the project scalability. The user can add, delete and edit recipes, a planner, a groceries list and the expenses tracker. A clean interface with persistent icon design and a light color theme is proposed. Finally, to secure the user's data, the password has been hashed before saving it in the database.

2. Time plan

This section has two different parts. Firstly, section 2.1. divides the project development tasks by their content. It regroups the assignments into significant packages. In section 2.2. there are the milestones and the time plan combined to present the project's development. They present a deliverable guide with start and end dates.

2.1. Work Packages

The work packages are a description of the different parts of the project development. In this project's case, three work packages can be distinguished. The first one is choosing and learning the different technologies, the second one englobes the front-end, and the third is the application's back-end. The complete version can be found in the appendix A.1. of this project.

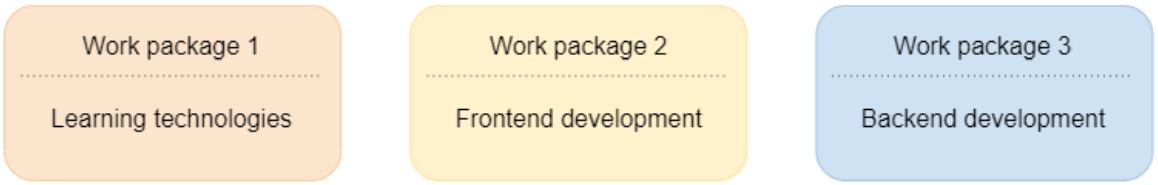


Figure 1: Work packages division, source: own compilation

2.2. Time plan and milestones

The time plan is represented in a table since the project's development is individual and linear. The milestones are different targets to keep track of the project's progress. The following table combines both of them to analyze the project's development.

Task	Start Date	End Date
Learning technologies:	22/02/2021	29/03/2021
Typescript		
Agular		
NodeJS and MongoDB		
Design and requirements	29/03/2021	19/04/2021
Frontend	19/04/2021	14/06/2021
Backend	10/05/2021	07/06/2021
Documentation	19/03/2021	21/06/2021

Table 1: Time plan diagram, source: own compilation

3. Overview

The completion of the project is entirely done from scratch with no prior knowledge of the technologies used.

This thesis consists of a MEAN stack project. It stands for MongoDB database, Express a server-side framework, Angular a client-side framework and NodeJS a JavaScript runtime. Thus the technology used is TypeScript, a superset of JavaScript. A Single Page Application is developed with Angular, which petitions to MongoDB through a RESTful API built with Express framework on a NodeJS environment. The reason for choosing those technologies is detailed in section 4.2.

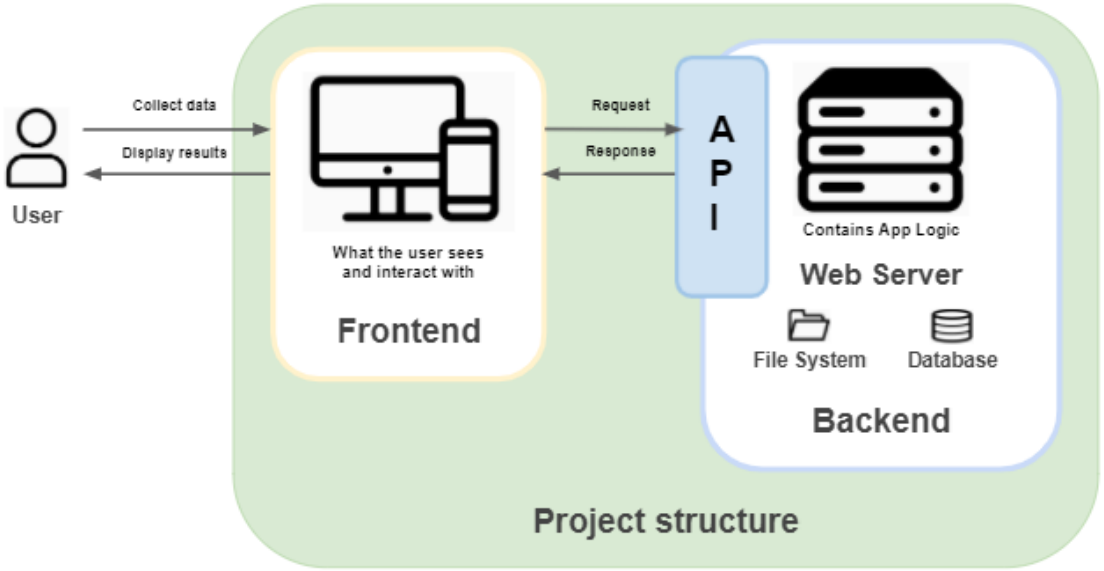


Figure 2: Project Structure, source: own compilation

For programming the SPA, Angular has a command-line interface tool used to initialize, develop, serve and build the application, Angular CLI. It is beneficial while scripting. The `ng serve` command generates a development server. It automatically rebuilds the application and reloads the page when a change is made in any source files, preventing wasting time while compiling. Subsequently, the display and the styling is mainly defined with HTML and CSS. Some libraries such as Bootstrap and Angular Material are also used.

Moreover, the backend of the application compounds the database and the RESTful API. The database stores the resources in different collections with documents in JSON, which relies on key-value pairs. JSON format provides fast access to data, and it eases its readability. Then the API, the Application Programming Interface, creates the connection between the SPA and the database. The API governs the endpoints, the web address that gives the client access to the different resources.

Tokens are introduced to preserve the user session. More precisely, JWT is kept on the client-side and sent to the server through the header's request, allowing authentication. Finally, the package manager used is NPM, the one for NodeJS, and the IDE where the project has been created in Visual Studio Code.

4. State of the art

This part is divided into two subsections. First, the description of different existing solutions to the problem presented. A contrast is made between the multiple solutions and the thesis proposal. Second, a report of the technologies used in the project and the reasons for the decisions presented.

4.1. Related commercial works

There are different solutions in the market. Nevertheless, they either resolve time planning, help to keep a list of the groceries or offer meal plans. Sometimes those meal plans come with delivery services. Shortly an exhibition of those solutions is going to be made relating them to the project scope.

To begin with, the most prominent platforms for time planning are Google Calendar [24] and Notion [25]. As its name stands out, the first one is a calendar. The user can arrange his or her meal plans and add a task description. The recipes for such meal can be detailed and even scribbled. However, this can be a tedious task. No suggestions are made, and the information is not classified.

Another solution is Notion, an all-in workspace organizer. The user has to design every feature to implement something similar to the product presented. They are consuming much valuable time building a space to achieve the same features as this thesis provides. Furthermore, the free version only allows sharing data with five other guests, while the website presented leads to create a community where everybody has access to the recipes.

Other applications that help keep track of a grocery list or write down a recipe are the Notes application of a phone device or Google Keep [23]. These applications allow scribbling fast an ingredient or a recipe, but none of them presents a space to plan the meals or share those recipes. Contrary to the project, the scope has a fast access notepad, not creating an environment to plan the cooking-related errands.

Next, there are web pages like ‘Directo al Paladar’ [27] or ‘Recetas Gratis’ [28] where a user can find an extensive range of recipes and, in the first one, even menus. Both of them can be very helpful. However, unfortunately, there are no characteristics to write down the users’ recipes. Neither to keep track of the ingredients and the expenses. Unlike the application presented. Another similar service is HelloFresh [26], a company that presents defined meals. It has a delivery service that supplies the ingredients required to cook the recipe. The same vacancy for holding the users’ recipes or expenses is present.

Finally, the Realfooding application [29] presents a free version with similar features. The user has access to its recipes or other users’ ones. In order to plan ahead of the meals or have a groceries list, there is a premium plan. Depending on the subscription selected, they have three different fees: a month, six months or a year. The subscription includes a weekly meal plan personalized depending on the user’s goals where no ultra-processed food is taken into account. Although our system does not have this already planned weekly meals functionality, presenting a restriction on the users’ food intake is not our product’s approach.

4.2. Technologies applied

4.2.1. TypeScript

The project is built entirely in TypeScript [2], besides the web display written in HTML [7] and the styles defined in CSS [8]. This language is an open-source superset of JavaScript [3]. It purports that all the features of the given language are included in the superset and have been expanded.

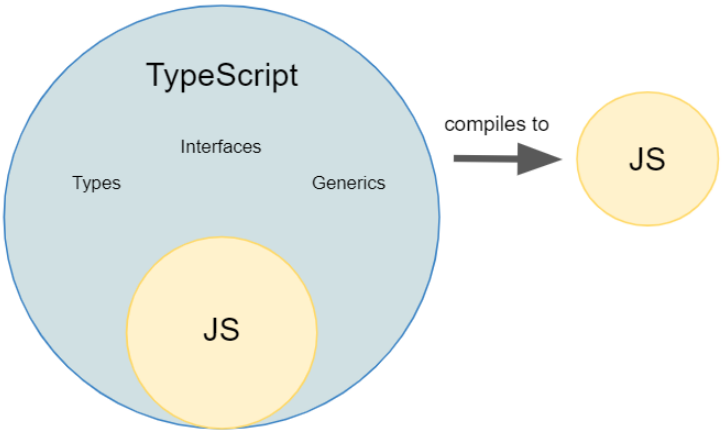


Figure 3: TypeScript superset, source: own compilation

For instance, it implements static type definitions, classes and interfaces, which brings more robust software. It provides a way to define the shape of an object and provides better documentation. Static typing is optional since TypeScript implements inheritance, detecting data types, giving the same strength without writing additional code.

On the contrary to JavaScript, TypeScript is a compiled language allowing compile-time error controls. It makes debugging more straightforward and therefore enhances code handling. It endorses creating more extensive and complex projects, which is the case. Additionally, it saves valuable time to developers.

However, the browsers can not run TypeScript since, as mentioned, it has to be compiled. So once the code is written, it compiles to JavaScript files directly run on the browser. For all those reasons, the project is programmed in Typescript.

4.2.2. Angular

Angular, or Angular 2 [4], is a complete development platform and Google's framework for building scalable SPAs. It implements an extensive collection of well-integrated libraries that include a wide range of features such as client-server communication, forms management, routing, data injection and more. Diversely, there is ReactJS [5], also known as React, a JavaScript open-source library for building user interfaces. Both are component-based.

Even though React dominates the market due to its maturity and long-time presence, the project is implemented in Angular. The reasons are presented shortly. Angular, as mentioned, is a complete Framework with the firm structure presented later on in the document. It makes learning and coding in Angular more complex, but it makes it more robust since coding structures are more defined, permitting faster and easier scalability.

Furthermore, it follows a template approach for HTML instead of having JS scripts, which can be confusing. The code is shorter and more readable. With injectors, data is passed through parent and child components bidirectionally. Finally, Angular CLI includes built-in tools to compile, serve, generate and test the application.

4.2.3. HTML and CSS

As mentioned above, Angular structures the Web page with HTML, HyperText Markup Language. It consists of a series of elements and attributes that state how the browser displays the content. Those elements are defined with tags, a starting and ending point. The content is placed in between those tags.

Moreover, CSS, Cascading Style Sheet, describes how the HTML elements are displayed on the screen. It defines the layout, font size, and font color compressing a web page's formatting in a single file. Those files can be used by multiple pages simultaneously.

For styling, the SPA uses other libraries such as Bootstrap and Angular Material have been implemented. They increase the speed of front-end development and endure a responsive page.

4.2.4. NodeJS and Express

NodeJS [9] is an asynchronous event-driven runtime for building scalable network applications. Many connections can be handled concurrently. No threads are employed, and since there are no locks, the process can not be dead-locking. If no service is required, the system is in an idle state.

This runtime presents an event loop hidden from the user, which does not need a start event. After executing the input script, it enters the loop, and when there are no more callbacks to perform, the loop is exited.

The Express framework [10] is implemented to provide a foundation for NodeJS. It is a minimal and flexible framework and provides a robust set of features for the web application. It defines a routing table used to perform the actions based on HTTP methods and URLs. Further, to install and publish the packages, npm [11] is used, the package manager of NodeJS.

4.2.5. MongoDB

There are two main types of databases, relational and non-relational databases. Also known as SQL databases, where SQL stands for Structured Query Language. And NoSQL, which means not only SQL. The principal distinction between the two is the structure.

The first one stores data in columns, rows and tables. Each column accommodates a data point, a category, and the row represents the value for that category. Then the table stores data only for one object. The relationship between tables and fields is called schema and must be clearly defined. Therefore, this kind of database emphasizes the structure.

The second one is document-oriented. It can store information under different categories, which all depend on different commands. In this scenario, the database uses columns and rows to enter data types and values and identify the object with keys. A

specific table is not required for a particular object. The database will automatically structure the information based on the key of the object.

The advantages of using a non-relational database over a relational database are the agility of updating the documents and its readability. For example, opening an individual document instead of having to shift between multiple tables eases the lecture. Further, it is simpler to handle them since fewer dependencies provide scalability and flexibility. In addition, those databases can store any data in enormous amounts with little structure, even unstructured data.

For those reasons, the project includes a non-relational database. There are four types of non-relational databases. We are just going to focus on document-based databases. A comparison between Mongo Database [12] and Firebase [30], two JSON-like document data models, is presented in the following paragraphs. Both databases are built to mitigate application development.

First, MongoDB is a high-performance document-based database, whereas Firebase is ideal for storing and synchronizing data in real-time. MongoDB offers scalability and flexibility with the querying and indexing of the developer needs. The prime importance of this database is on the data storage factor. It lacks a complete ecosystem. On the other hand, Firebase mentioned above has a complete ecosystem for generating mobile and web applications. It is a real-time engine with background connectivity. It has many more services, like hosting, storage, cloud function, and machine learning, compared to MongoDB.

The comparison between the two databases is presented below.

Comparison	MongoDB	Firebase
Performance	High performance with a high traffic application	Does not support high performance
Supported Languages	Python, Java, JavaScript PHP, NodeJS, C, C# ...	Java, PHP, NodeJS, JavaScript, Objective-C, Swift, C++ ...
Application	Best suitable for large-scale applications	Best suitable for small-scale applications
Scalability	Powerful sharding and scaling capabilities	Instant data updates without refreshing
Pricing	Free version when you configure on-premise, with paid version the developer gets a serverless set up	Pay-as-you-go plan model with flexible rates
Management	Has confusing ‘middleman’ hosting arrangements, complex queries are complicated to work with	Dealing with relations and data migration is quite complicated
Security	Considered highly secured because no SQL injection can be made	Allows straightforward hosting in Google’s Cloud Platform

Table 2: MongoDB and Firebase comparison, source: own compilation

After presenting both databases, we can say that both are great for application development, but for a complete backend as a service, MongoDB is better due to its performance. For that reason, and since it is a more flexible technology, it is used in this project.

5. Backend

5.1. Introduction

The project's back-end is composed of a Mongo database and a RESTful API [14], and web services.

Figure 4: Backend components, source: own compilation

As presented previously in the technologies section, this project has a non-relational database. It allows modelling the collection with schemas, which are, in this case, typescript files.

The RESTful API is the endpoint's set. The addresses where requests are sent from the client allow retrieving resources from the database.

5.2. Database

5.2.1. Design

The database has been designed to fulfil the needs of the application. For that reason, it contains six collections, User, Recipe, Ingredient, Expense, Menu and Day. Based on the functionalities presented previously in the document, the designed attributes of the collections are presented in the appendices A.2.

All the schemas are defined with a timestamp. It automatically adds a createdAt and updatedAt field when a document is created. The unique Id of a document is also generated when saving the document.

The design of the schemas has been defined following MongoDB best practices. On the one hand, it favored embedding documents, considering that a document size limit is 16MB. For this reason, all the meals that compound a day are in the same document.

On the other hand, it is using references to avoid duplication of data and have smaller documents. It allows better scalability and is the reason for having a separate collection for expenses, menus and days. If less than 20% of the document is not used, it is considered useless.

Next, storing references. When there are Many to Many relations, the data should be stored in both documents. The data can be reached from any of them, so it has to be accessible. Nevertheless, the decision not to save the references in the ingredients documents for user or recipe and ingredient relation has been made. Knowing for example, how many users need to buy an ingredient is irrelevant to the application. For that reason and to preserve space, those fields are omitted on the ingredient side.

Moreover, in the one to many relations, the id or field of one endpoint is stored in the documents of the many endpoints. It allows filtering all the documents with the id specified. For example, a user can have many menus. Therefore, the user's email is saved in the menu document to get all the menus that contain the email address required.

Hence, the relation between the collections is the following ones.

User schema:

The recipes attribute an array of recipes ObjectId, allowing the relationship between the user and the recipes. Those are the recipes that either has been created by the user or have been saved.

The groceries attribute is an array of ingredient names that allows making the relation between the user and the ingredients. Those are the ingredients missing in the user's kitchen and need to be bought for future meals.

Recipe schema:

The ingredients attribute is an array of ingredients that allow making the relation between the recipe and the ingredients. Those are the ingredients required in order to follow the recipe.

The creator attribute is the email of the user, which allows making the relation between the recipe and the user. It is the person that has added the recipe.

The saved attribute is an array of users emails which allows making the relation between the recipes and the users. Those are the users that have saved the recipe.

Menu schema:

The _user attribute is the email of the user, which allows making the relation between the menu and the user. Those are the menus owned by a user.

Day schema:

The _menu attribute is the id of the menu, which allows making the relation between the menu and the day. Those are the days that form the menu.

Expenses schema:

The _user attribute is the email of the user, which allows making the relation between the expense and the user. Those are the expenses done by a user.

5.2.2. Management

In order to manage the database, the library used in the project is mongoose [18]. This library implements the queries to manage the database, such as saving, searching, updating or deleting data.

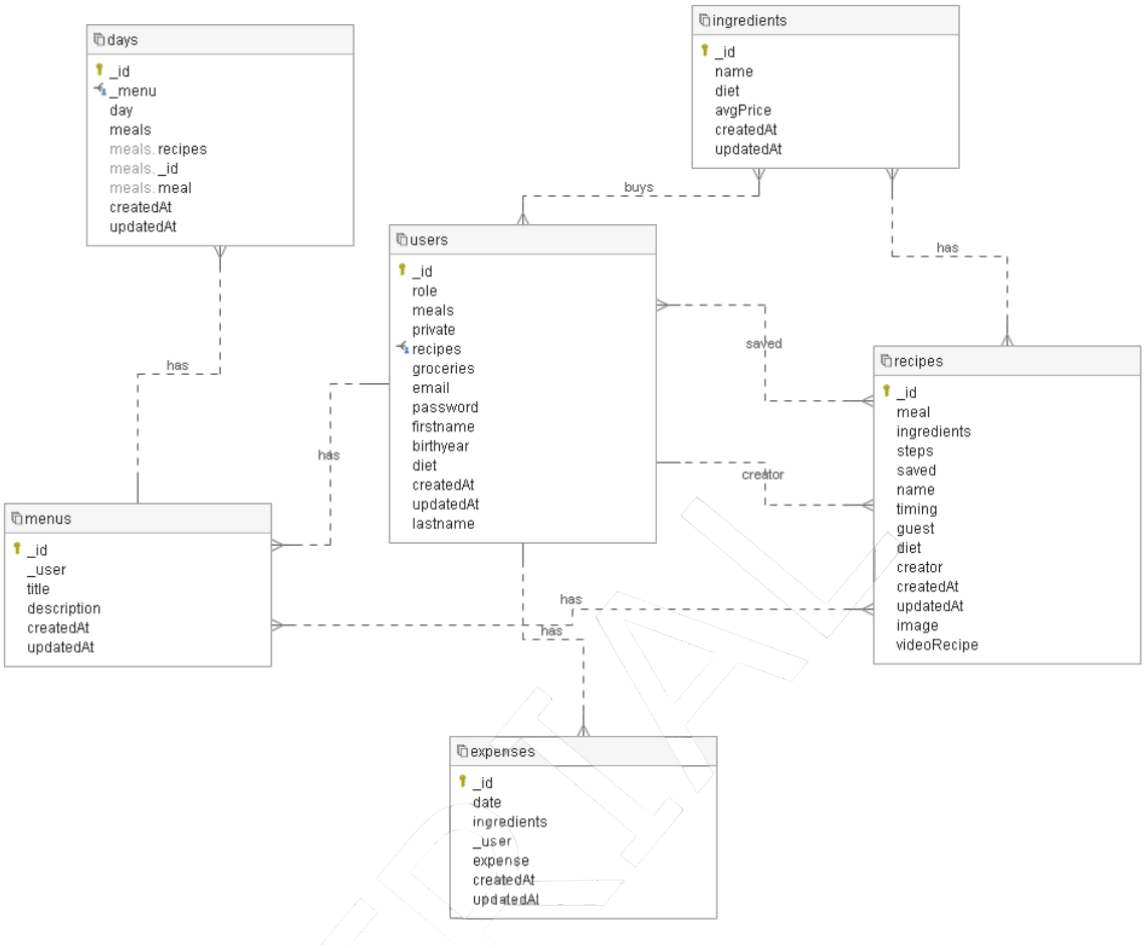
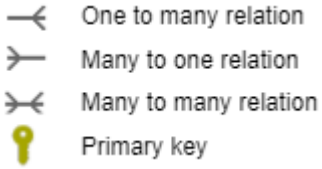


Figure 5: Database conceptual map, source: Dataedo Entity Relationship Diagram

Legend:



5.3. REST API

REST is an acronym for Representational State Transfer, an architectural style for an application program interface (API). It allows communication between the user and the database. The separation of the user interface involvements and the data storage charge improve the portability of the interface across multiple platforms and the scalability of the server by simplifying its components.

The REST API is stateless. It means that all the information required to understand a request is on the client-side. Therefore the session state is kept by the client. Furthermore, it uses HTTP requests to access and use data. The transferred data is in JSON format in the body of the request. A token is given within the header detailed in the authentication section to maintain the state.

In this project, the primary operations implemented are CRUD [15], Create, Read, Update and Delete. Those are accessed with the respective POST, GET, PUT and DELETE requests. The first one adds or retrieves data given some information. The second one retrieves information from the database, the third one updates the data, and the delete request removes the specified data.

The REST technology has been chosen since it uses less bandwidth, enduring efficient internet usage.

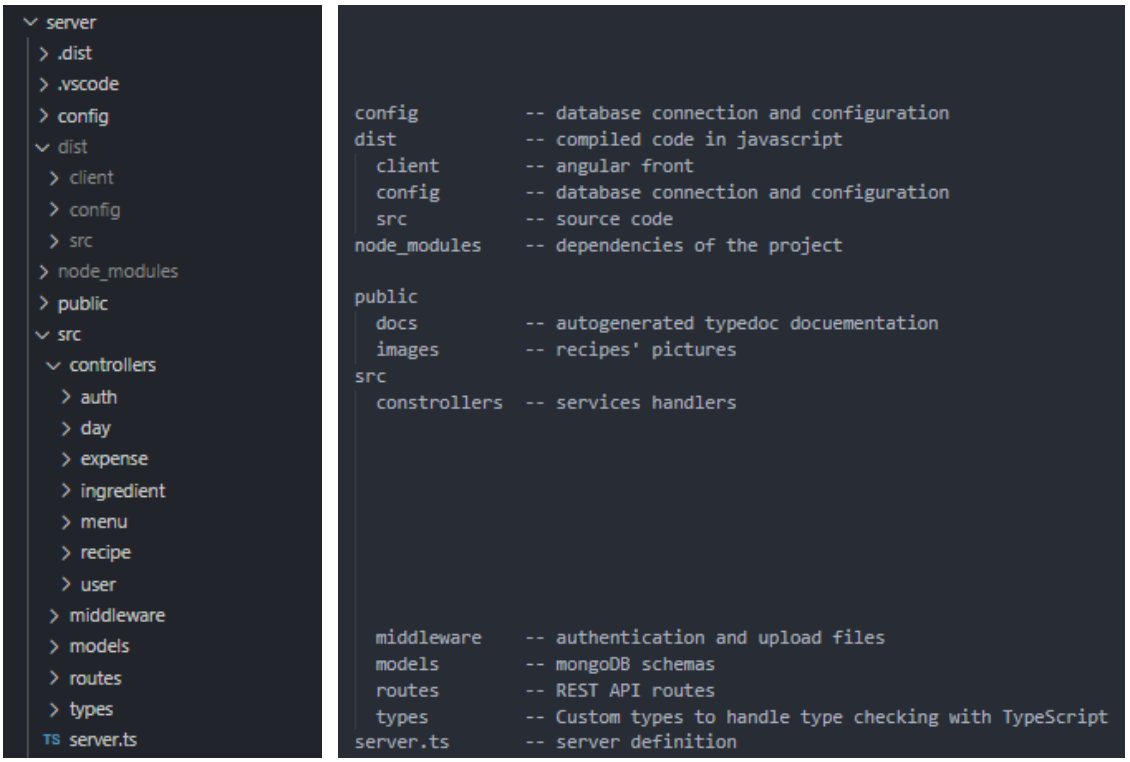
5.4. Architecture

The server has an MVC architecture [13], which stands for Model, View, Controller. The goal of this organizing method is that each section of the code has a specific purpose.

First, we have the Model, which holds the raw data. It is the schema modelling of each collection of the database. It allows defining the components of the application. It englobes the data-related logic for the user to interact.

Next, the View permits the interaction between the backend and the frontend, which in other words is the UI logic (User Interaction). It is the definition of all the routes that enable making requests from the client to the server.

Then, the Controller, which acts as an interface between the Model and the View. Given the information, the Controller states the data usage or how to process it to pass it to the View, rendering the final output.



Figures 6, 7: RESTful API tree folder architecture and details, source: own compilation

5.5. Authentication

Users have to log in to access most of the application's features. They first have to sign up and set the user's email and password to accomplish it. In addition, security is introduced on the server-side. Passwords are hashed before introducing them in the database, preventing possible attacks.

This encryption is done with BCrypt [11] [17], a library to hash passwords. When a user signs up, the system checks if any user in the database has the same email. Then if it does not exist, a salt is generated to hash the password obtained from the client-side, and then all the user's data and the hashed password is saved in a new document.

BCryptis a Key Derivation Function (KDF) whose purpose is to slowly convert input data to a fixed-size, deterministic and unpredictable output. It means that, indifferently, the input size of the output is always the same. Being deterministic implies that a hash is unique for every data input. When a user logs in, the BCrypt compare function hashes the input data and compares it with the hash saved in the database. If it matches, the password is correctly introduced by the user. The output is unpredictable since KDF has additional properties to a hash function. Those are key stretching, whitening, separation and strengthening. It is based on the Blowfish block cypher cryptomatcd algorithm. An algorithm that allows adaptive hash function means that the cost of hashing can be adjusted.

The output format is the following one.

```
$[algorithm]$[cost]$[salt][hash]
```

Where two chars hash algorithm identifier prefix. "`$2a$`" or "`$2b$`" indicates BCrypt. Cost-factor (n) represents the exponent used to determine how many iterations 2^n, 16-byte (128-bit) salt, base64 encoded to 22 characters and 24-byte (192-bit) hash base64 encoded to 31 characters.

```
$2b$10$nOUIs5kJ7naTuTFkBy1veuK0kSxUFXfuaOKdOKf9xYT0KKIGSJwFa
 |  |  |                     |
 |  |  |                     hash-value = K0kSxUFXfuaOKdOKf9xYT0KKIGSJwFa
 |  |  |
 |  |  salt = nOUIs5kJ7naTuTFkBy1veu
 |  |
 |  cost-factor => 10 = 2^10 rounds
 |
 hash-algorithm identifier => 2b = BCrypt
```

Figure 8: Bcrypt Output Format example, source: https://www.npmjs.com/package/bcrypt

The salt, random data used as additional input is generated by the genSalt() function. The argument passed to is the cost of the hashing, which in the application case is ten hashes/second.

Since RESTful API is stateless and some API endpoints need authentication, a token is required. Whether the user signs up for the first time or he or she is logging in, a JWT, a JSON web token is generated with the jsonwebtoken library and saved on the client-side. How it is stored is explained in the following section. JWT [16] is an open standard that defines a compact and self-contained way for securely transmitting data. It is digitally signed with the HS256 algorithm.

Furthermore, the library allows the token's verification ensuring that the user has the authorization to access routes. The validity of the token is set to 1h. Afterwards, the token is no longer valid. The structure is header, payload and signature separated by dots. An example is provided in the following figures.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2MGE1MzhhNGVmZGNiODQxMDRiZDE5YmIiLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE2MjM3Nzc4NTMsImV4cCI6MTYyMzc4NTA1M30.0C_eiljfcCNxqZvP0UMK6_eQMAP6lBt4yAbI-KTkNWE

Figure 9: JWT example, source: own compilation

HEADER:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD:

```
{
  "userId": "60a538a4efdcb84104bd19bb",
  "role": "admin",
  "iat": 1623777853,
  "exp": 1623785053
}
```

VERIFY SIGNATURE:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  0C_eiljfcCNxqZvP0UMK6_eQMAP6lBt4yAbI-KTkNWE
)
```

Table 3: JWT decode example, source: https://jwt.io/

5.6. Uploading files

The recipe has an image field. Thus images files can be uploaded to the website. The multer library [11] is used to manage files. Given a form data, it allows storing them in the server. A variable is defined to configure the name and the destination of the files uploaded. Its size is limited to 25MB.

The files are stored in the public server's folder, meanwhile the path is stored in the database. In a future version, those files will be in a separate database of just images.

6. Frontend

6.1. Introduction

As introduced earlier, the frontend is the user interface. In this project, a single page application built on Angular. It is a website that interacts with the webserver without reloading a page, working within the browser. Therefore if it is correctly configured, the user experience improves compared to a multi-page application. Such an application loads a new page every time a link is clicked because transitions are dynamic.

From a corporate perspective, increasing speed is a significant advantage. Several studies state that one second of additional delay in a page load costs 1% of sales [22], which could represent millions of dollars in the case of vast enterprises. For those reasons, a SPA loads many resources when the application is launched, such as HTML, CSS or Scripts files, making the application highly responsive. Overall there is a lower impact on the server since only the changing data is demanded.

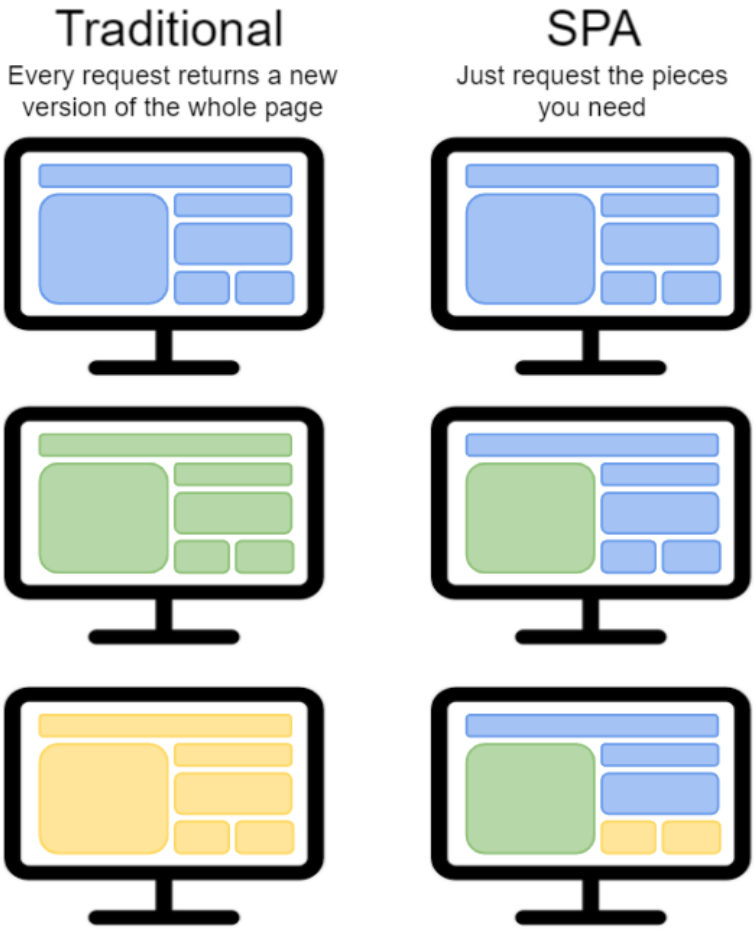


Figure 10: Loading data comparison, source: own compilation

Subsequent, from a developer perspective, the making of the application is streamlined and optimized. As revealed earlier, Angular has an entire environment that mimics a server and reloads the SPA each time a source file changes. It allows visualizing faster the development while building the application. Meanwhile, the browser monitors the network operations, thus, facilitating the debugging.

6.2. Functionalities

This thesis purpose is to unify the cooking-related duties. Therefore, the features presented before are implemented with the subsequent composition. First, a home page is required, where all the recipes are shared for any user. Then, different filters can be applied, such as writing the recipe's name, defining the meal, the diet or looking by the ingredients that constitute such a recipe to narrow down various recipes.

Once logged in, a user can save any recipe to his or her personal recipe's book or add a new one. The creator of a recipe can additionally edit it. Another showcase is removing a recipe. A user can permanently remove a recipe from his or her recipes book. There are two plausible cases, people having the recipe preserved or being the only person to own the recipe. In the first case, either the user is removed from the saved attribute of the recipe, or randomly a user in that array is chosen as creator, gaining editing permissions. Otherwise, the recipe is removed from the database, as explained in figure 11. Additionally, filtering through recipes is also available in that view.

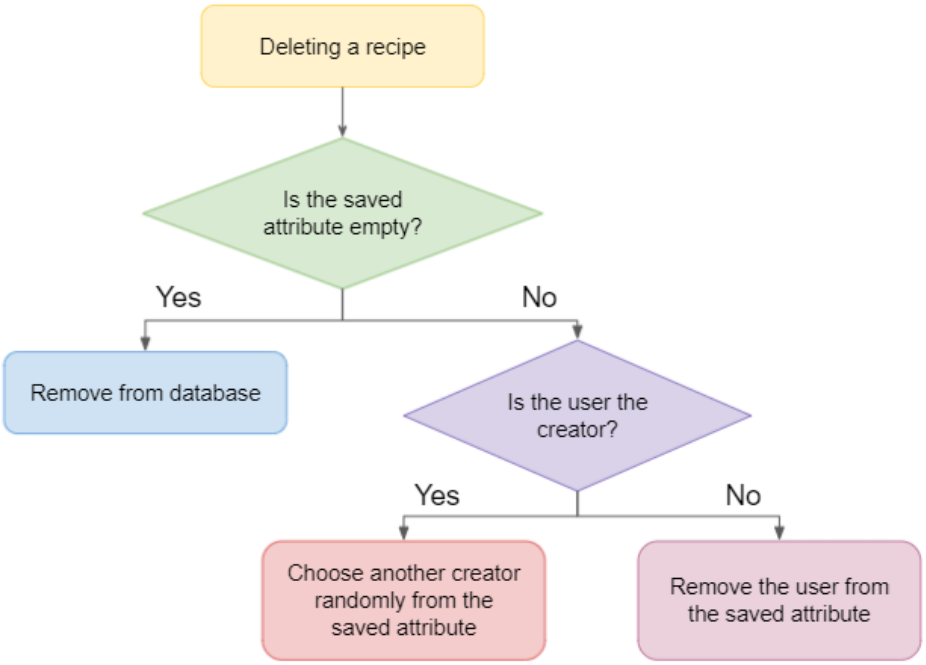


Figure 11: Deleting a recipe, source: own compilation

The first time the user accesses its planner, it is empty. When creating a menu, he or she can specify which meals usually eat, allowing having different menus with different meals, which can also be deleted. A title is required, and a description can be added. Then the menu is generated automatically. Each slot represents a meal of a weekday where recipes can be added or deleted. The user can navigate to the description of any recipe present in the menu.

The user keeps track of the ingredients missing in the fridge for preparing the different meals of the week in a groceries list. Any element can be added or deleted. Also, ingredients can be added from the recipes' ingredients list to simplify matters and benefit the user. If the ingredient is already on the list, a message will warn the user. Next, the ingredients can be checked while buying, and once the shopping is over, there is a checkout button, which removes the bought ingredients and adds the record as a new purchase in the expense tracker with the actual date.

As mentioned, to help users keep track of their expenses, every time they go to the supermarket following the steps mentioned above, it automatically makes a log of the ingredients bought. It can be edited to add the expense for such a purchase. This way, the user knows when was the last time he or she went to the grocery store, what he or she bought, and how much he or she spent.

Finally, any user can check their data and edit the profile.

6.3. Architecture

The client side has Angular's architecture [4], which is represented in the next diagram. There are eight main building blocks: Components, Data binding, Dependency injections, Directives, Metadata, Modules, Services and Templates.

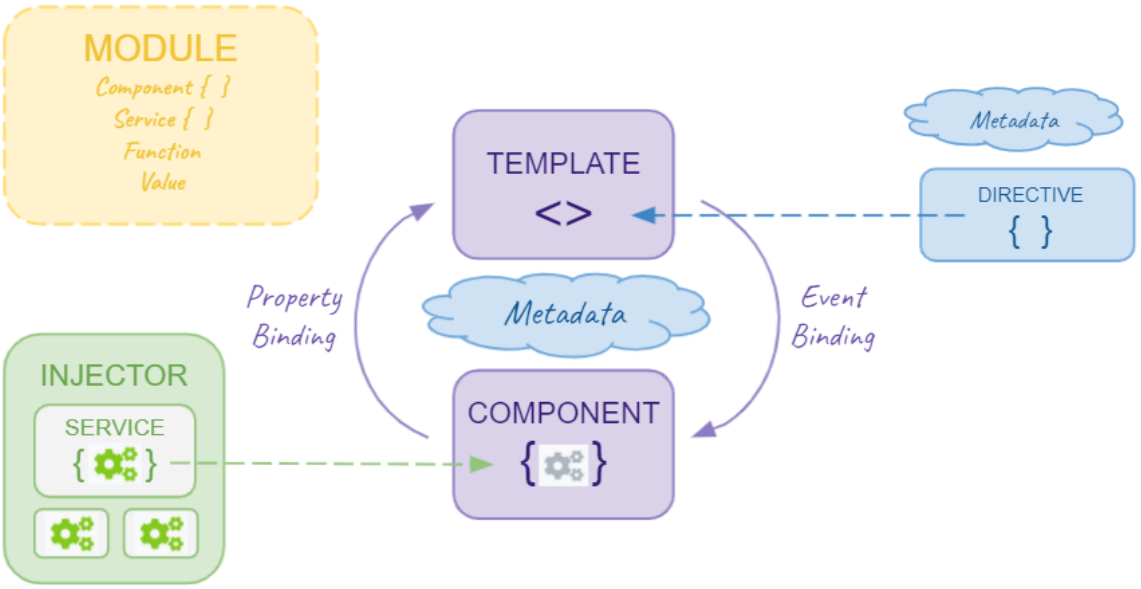


Figure 12: Angular architecture, source: own compilation

Beforehand, an Angular application is modular, based on its modularity system called NgModules. At least, an angular application has one module, the root module, named AppModule. As an application grows in complexity, different feature modules are added. A decorator, @NgModule, attaches the metadata that defines the declarations, imports, exports, and providers' properties to know if a class is a module. More details will be added in the next section.

Succeeding, there are components, which control a patch of the screen, a view. They are classes that support the application logic, for example acquiring data from service through the API to bind it on the user interface. Another role is binding user events with a click on a button.

Each component has a template, an HTML file that defines how to render the view. It can also have a CSS file that states the style of the view. The display is not just plain HTML. Angular allows having custom elements, child components that have their template. Furthermore, there is a template syntax that increases the functionalities of the HTML file.

Metadata attach data to a component through the @Component decorator, for example, the selector, which presents the tab of such element in HTML. It allows Angular to insert an instance of a component in any view throughout the template definition.

Another block is directives, which a decorator also defines. According to those instructions, when the directives are rendered, it transforms the DOM, the document object model. For example, the component decorator is a directive decorator extended with template-oriented features, implying how the view is rendered. Other directives are the structural ones. They present conditional operations, *ngFor and *ngIf, in the template file. *ngFor loops through an array defined in the component and the *ngIf, the element is present in the view when the condition is valid. There are a few more, the attribute directives, others that modify the layout structure or the aspects of the DOM.

This framework also supports data binding, which avoids developers being responsible for pushing data into the HTML or retrieving user actions. There are four systems of data binding syntax.

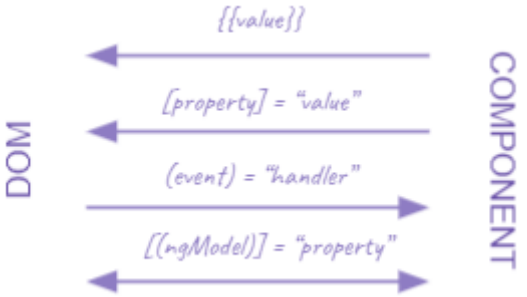


Figure 13: Data binding syntax, source: own compilation

The first one, interpolation, displays a component's property value within an HTML element. Next, property binding, which passes a property value from a parent component to a child component. Consecutively, the event binding is either a user event or passing a child component property value to a parent component. Last, the two-way data binding, where the property value flows to the input box from the component as property binding, allows users to reset the value to the latest value. Therefore, it is essential in the template with component communications and parent and child components communications.

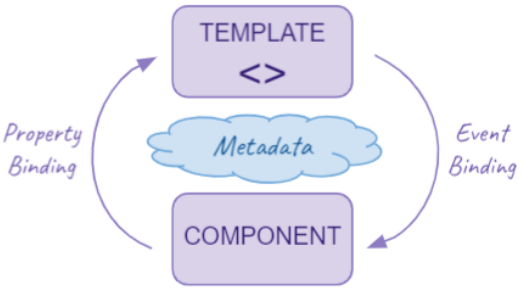


Figure 14: template - component data binding

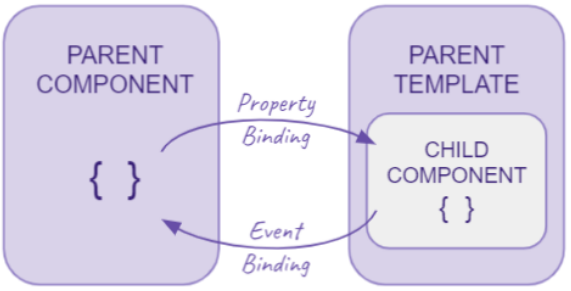


Figure 15: parent - child component data binding

Lastly, we are going to present the services block. It is a class with a well-defined purpose, there is no base class for such a block, but it is an essential part of the architecture. The services manage the connection with the backend API, defining the requests. A service can be anything from a tax calculator to a logging service.

6.4. Design

After introducing how Angular is structured and how the SPA interface is built, the different elements will be presented in more depth, explaining how the application works, with actual examples.

6.4.1. Views

As presented, Angular views are composed of three files, a typescript file where the component is defined, an HTML file which is the display template, and lastly, a CSS file that englobes the view's style.

The components are reusable. Some of them are shared through all the screens of the web page and in the navigation bar. Others have a specified purpose, meaning that components can be either nested or combined, giving complete flexibility to developers. It is a tremendous advantage since, due to data binding, data transmission between components is faster than reloading the page and sending a request to the API.

This flexibility also generates a highly responsive environment. Views adapt themselves to a phone, tablet or computer screen, making the website multiplatform. In addition, the CSS stylesheet and flexbox display allow stating how items render on the view.

6.4.2. Navigation

Although it may seem ironic, a SPA requires navigation. Therefore, routes are defined to navigate through the screens, the different components.

In pursuance of changing the views, sidebar navigation has been implemented. Depending on the screen's width, the sidebar is static or dynamic, meaning that it is permanently present on the left side of the screen or hidden until the user pushes the menu button.

Angular's navigation also permits redirecting a user to a specific page. In this project case, if a user is not logged in, every page except for the home page redirects him or her to the login screen. The redirection is also introduced when a form is submitted or when accessing the description of a recipe.

Angular has an @angular/router library to implement navigation in an application, where the Routes, Router and RouterModule classes are found. The first class is used to define the routes array, specifying the path and the component to be displayed for each one of them. The order in which the routes are defined is important because the Router uses the first-match strategy when suiting routes. Therefore more specific routes are placed above the less specific ones. Further, the wildcard route comes last, which is the not found route since it matches every URL. Thus, the Router selects it when no other route is matched.

The routes are defined in a routing module, which is a module that imports Routes and RouterModule. Hence, the module's properties have to export the RouterModule and import the RouterModule, calling either the forRoot() or forChild() method. In both cases, the argument passed is the routes' variables. The first case is present in the app.routing.ts file, the root routing module implementing the Router service, and the second is present in all the other components routing modules, not including the Router service. Consequently, the routing module has to be in the imports property of the component module or the app module to use the routes.

Lastly, the routes have to be included in the application. They can be added in the component template with the routerLink attribute where the path is defined. The tab <router-outlet> needs to be present in the template, too, so Angular is informed and can update the view with the selected route. It is also possible to indicate navigation actions within the components. For example, importing the Router class in the component, the method navigateTo(), given a route as an argument, switches the view.

6.4.3. Structure

Modules are required to organize related things collectively. Every module is a class with an @NgModule decorator, which attaches metadata. It describes how to compile the component, its directives, pipes, and extend its capabilities through external libraries.

The four main properties attached are imports, exports, declarations and providers. Imports include other modules in the current one, with the components and directives within those modules. It is possible if the elements are public, meaning that they are present in the exports attribute. The declarations state which components, directives, and pipes belong to the module. Lastly, it provides services that other components can use in the application.

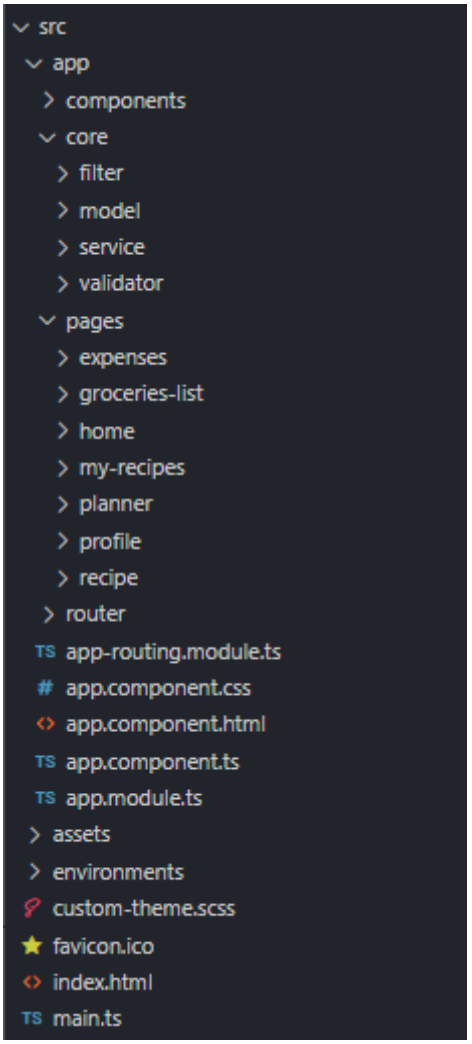


Figure 16: SPA tree folder architecture, source: own compilation

6.4.4. API connection

A service is required to establish the connection between the application and the API. More precisely, a service with an HTTP client since the requests are done through HTTP protocol.

The service is a single class with the four operations possible and an errorHandler to regroup all the possible requests. There are particular methods for the get, post, put and delete requests. All of them have as first argument the string representing the endpoint of the API. Next, this string is concatenated with the URI base, the IP direction that allows locating the server on the Internet, obtaining the full URI to make the request.

The second argument of the get and delete method is the httpOptions, which is also the third argument of the post and put requests. This argument has the headers of the petition. Some requests require authentication, as explained before, is fulfilled with a token. This token is passed from the user to the API through the request headers, so the backend middleware can get the information and do the procedures to identify the user.

For the post and the put methods, the second argument is the body. It enables passing the data from the user to the backend in JSON format in the project's case.

Afterwards, the HTTPClient method is called, and the different arguments mentioned are passed. Finally, it returns an observable, which is then handled in the component, except if an error raises.

Angular has a class to handle errors, the HttpErrorResponse. It detects whether the error is raised on the client or the server side. It allows the developer to define what happens in each case. If the HTTP error response is an error event, the error is passed to the component as an observable too, with the error message, to detect what happened. Otherwise, it is a server error. The error state gives the error definition and the error message defined in the backend to describe the errors.

Further, either the API returns data or an error, the component needs to be aware. For that reason, in both cases, an observable is returned when a component calls one of the service methods. The observable delivers the values that are consumed further by the component with the subscribe function. This function defines how to obtain the data, what to do with it, and the messages to be published. The developer adapts it for the different cases.

6.4.5. Keeping state

In the last section, notice that the connection between the frontend and the backend is stateless, meaning that the server does not keep information from the client. For this reason, when authentication is required, a JWT token is passed through the request header. This token is kept on the client-side until the user decides to log out.

It is accomplished with another service, the local storage service. First, the application ensures that local storage is available on the client window. If it supported three methods, manage the data, a get, set and remove method.

The data is saved as a key-value pair. When a user logs in, the token retrieved from the server and the email are stored with the set method. Hence, it makes them available in the application while the session is kept alive, with the get method. Finally, when the user logs out, or the session is expired, both elements are deleted from the storage with the remove method.

7. Results

This section is divided into four parts: the front-end results, the database results, the server results, and a global perspective.

The different functionalities presented during the document have been accomplished. The website presents a homepage where all the recipes are presented in a card format. The card contains the diet, the meals, the name and the picture of the recipe. Then from the recipe's name, the description is deployed. Four tabs are present to filter the recipes by name, diet, meals or ingredients contained. A visualization of this description can be found in figure 17.

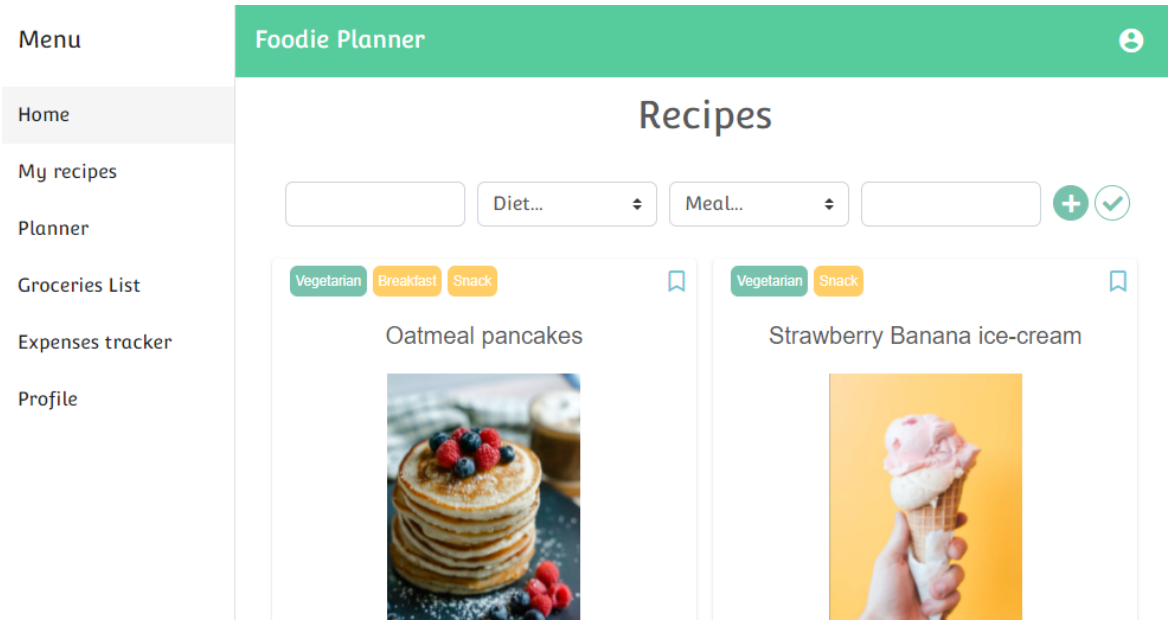


Figure 17: Home screen, source: own compilation

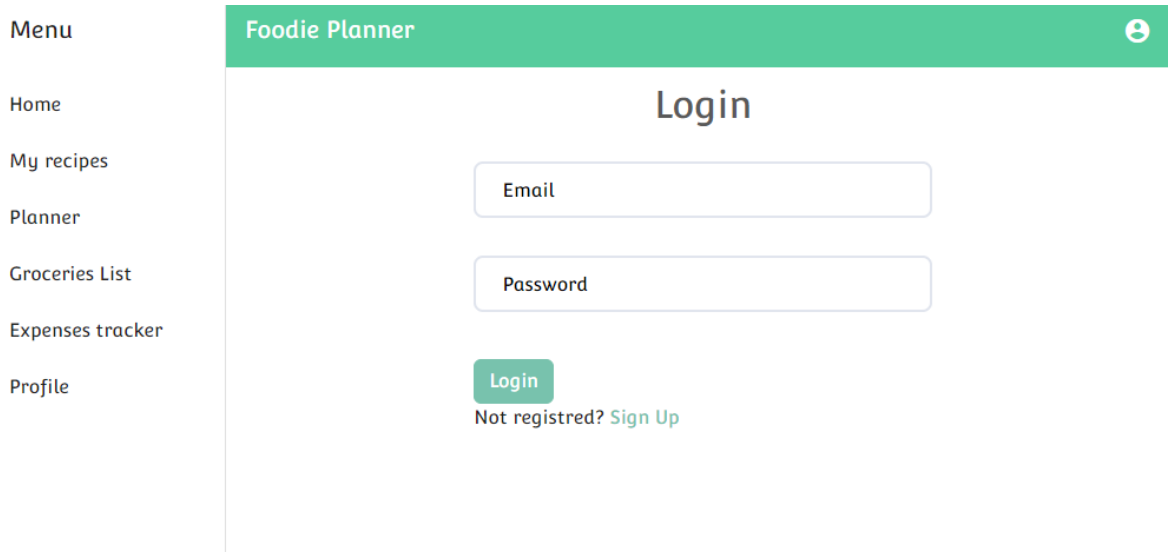


Figure 18: Log in screen, source: own compilation

In order to log in, the user email and password is required. If the user has not signed up, three different forms are sequentially shown to set all its data. First, the email is checked since it must be unique. Next, the password must be at least eight characters

long with mayors, minus and special characters, and numbers. The second form asks for user data, and the third one his or her eating habits.

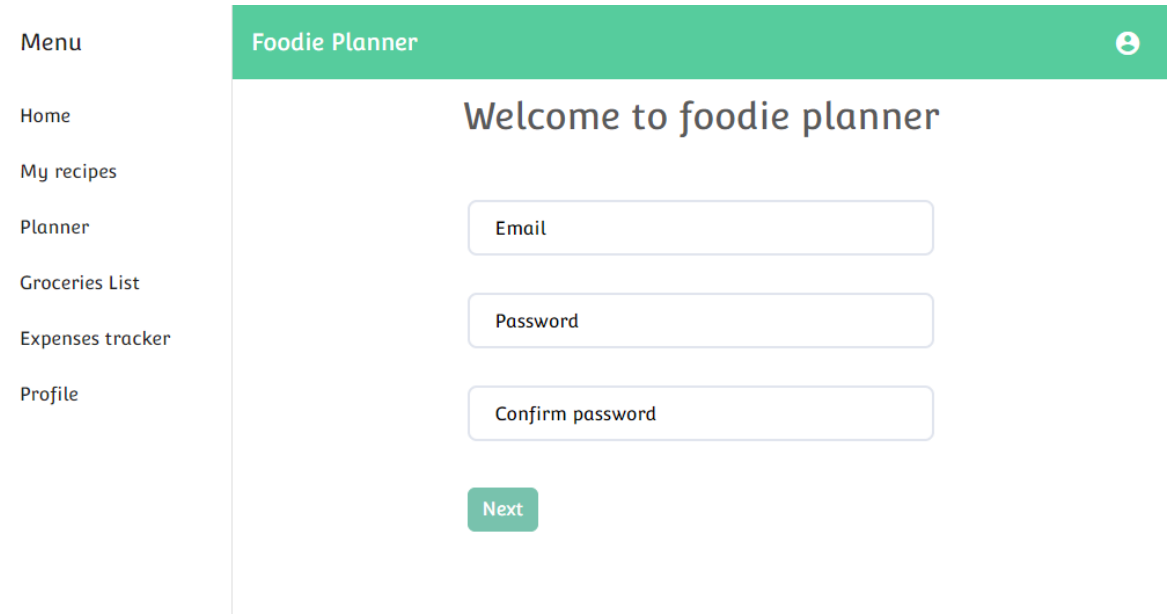


Figure 19: First registering form, source: own compilation

Once registered, the user can save from the homepage to the user's recipes book. The recipes book has similar features to the homepage, but a recipe can be removed instead of saved. New recipes can be added from this screen, and if the user is the creator, also edit them. Filtering and searching through the recipes as mentioned is also available.

The planner feature at first is empty with a button to add menus and a message. When a menu is added, a form to set a title or a description unfolds. Hence, an empty menu is present on the screen with the different days of the week. Then the meals are either selected from the form or defined in the database by the user profile. Each meal is presented in a card which can be edited by adding or deleting recipes. The recipe is also accessible from the name of the recipe. Once on the recipe, the ingredients can be added to the groceries list, or if attached, the video recipe is also available.

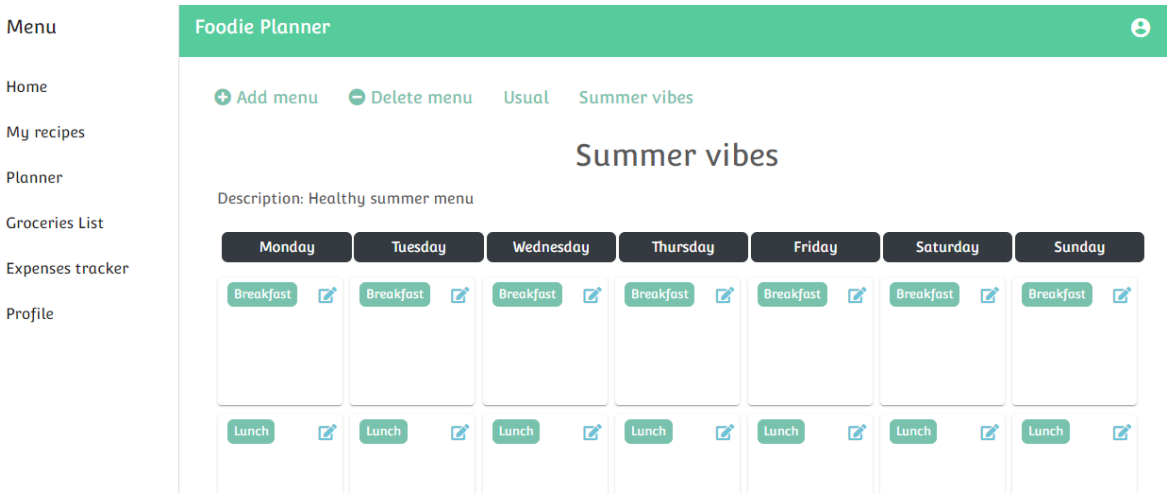


Figure 20: Planner screen, source: own compilation

Additionally, the groceries list presents two cards, the ingredients missing and the ingredients bought. There is a check or a cross button, if required, to pass data from one to the other. There is also a cross button in the missing ingredients to remove the ingredient from the grocery list. Lastly, the search bar informs if the ingredient is already on the list to add it or not.

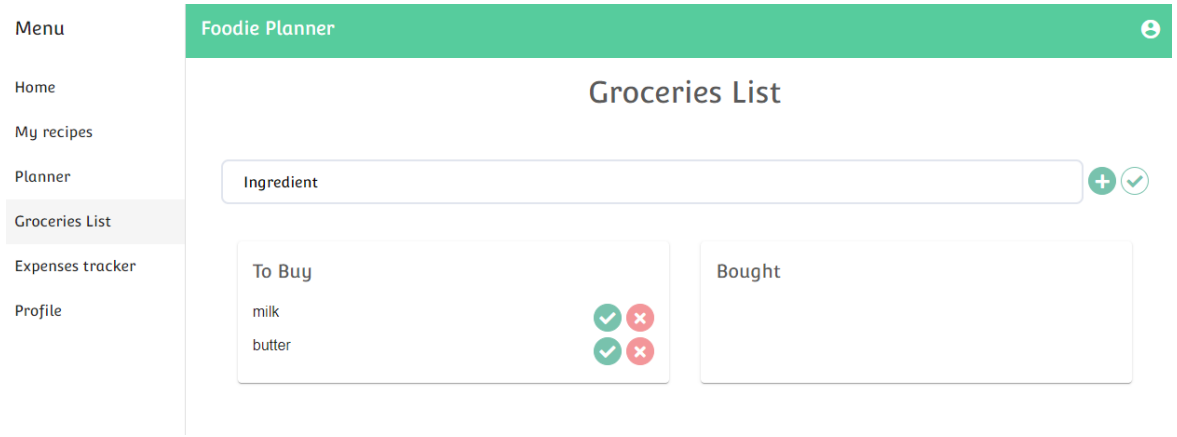


Figure 32: Groceries screen, source: own compilation

Another feature mentioned is the expenses imprint. Once the ingredients are bought, there is a button to check out the purchase. A modal enables the user to have a second thought. Once checked, an expense is automatically generated with the actual date and the ingredients. It can be modified to either state the expense or change any ingredient. Expenses can be added or deleted at any time. Lastly, the user can visualize and edit its profile anytime.

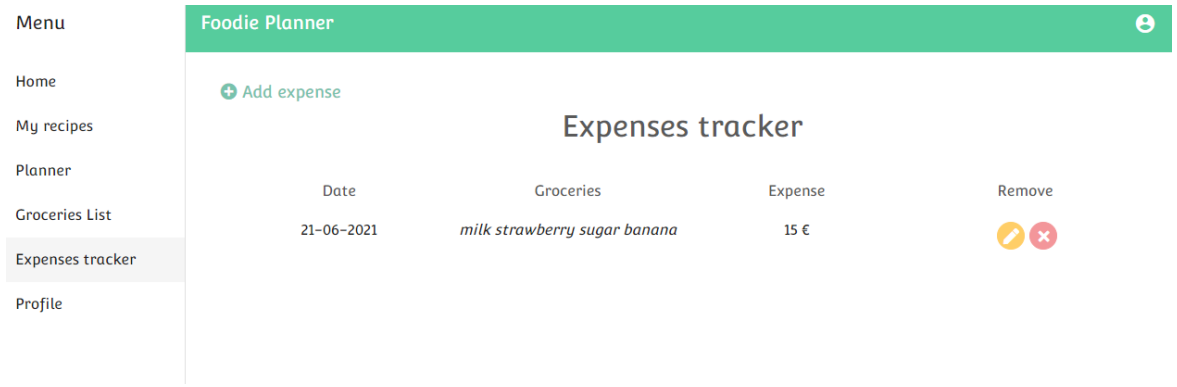


Figure 33: Expenses tracker screen, source: own compilation

Moreover, the following section presents the database results. It consists of a MongoDB Atlas database in an AWS cloud environment. The database has a persistent connection with the web service allowing constant requests of resources. This database has six different collections, as we can visualize in the subsequent figure. The collections will grow as users saves their menus or recipes.

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
days	14	315.7 B	4.3 KB	1	36.0 KB	
expenses	1	199.0 B	199.0 B	1	24.0 KB	
ingredients	194	105.2 B	19.9 KB	2	72.0 KB	
menus	2	153.0 B	306.0 B	1	36.0 KB	
recipes	2	555.5 B	1.1 KB	1	36.0 KB	
users	3	386.3 B	1.1 KB	2	72.0 KB	

Figure 23: MongoDB collections, source: own compilation

The last and maybe most important part of the project is the RESTful API. It gives cohesion to the website, bringing together the client-side and the data. As presented through the document, the different endpoints of the API allow retrieving resources from the database. Those endpoints are presented in the following tables. Three different permissions are defined to protect the data, either public, private or administrator. Being a public endpoint means that anybody can access that data. The private case is for registered users. Lastly, administrator permission is for managing endpoints. Then, the controller files handle the petitions to the database. They are triggered when either a post, get, put or delete HTTP request is sent to the API.

Methods	URLs	Actions	Access
GET	/api/	retrieve all Recipes	public
GET	/api/recipe/:rid	retrieve a Recipe by `:rid`	public
GET	/api/recipe/book/:uid	retrieve all user's Recipes	private
POST	/api/filter	retrieve Recipes with filters	public
POST	/api/recipe	create new Recipe	private
PUT	/api/recipe/:rid	update a Recipe by `:rid`	private
PUT	/api/recipe/save/:rid&:uid	save a Recipe by `:rid` by `:uid`	private
DELETE	/api/recipe/:rid&:uid	delete a Recipe by `:rid`	private

Table 4: Recipes' endpoints, source: own compilation

Methods	URLs	Actions	Access
GET	/api/auth	retrieve all Expenses	admin
POST	/api/auth	retrieve all user's Expenses by `:uid`	private

Table 5: Authentication endpoints, source: own compilation

Methods	URLs	Actions	Access
GET	/api/user	retrieve all Users	admin
GET	/api/user/:uid	retrieve a User by `:uid`	private
GET	/api/user/exists/:uid	check if User already exists by `:uid`	public
POST	/api/user	create new User	public
PUT	/api/user/:uid	update a User by `:uid`	private
PUT	/api/user/ingredient/:uid&:iid	save an Ingredient by `:iid` by `:uid`	private
DELETE	/api/user/:uid	delete a User by `:uid`	private

Table 6: Users' endpoints, source: own compilation

Methods	URLs	Actions	Access
GET	/api/ingredients	retrieve all Ingredients	admin
GET	/api/ingredient/:iid	retrieve an Ingredient by `:iid`	public
POST	/api/ingredient/filter	retrieve Ingredients filtered by name	private
POST	/api/ingredient	create new Ingredient	admin
PUT	/api/ingredient/:iid	update an Ingredient by `:iid`	admin
DELETE	/api/ingredient/:iid	delete an Ingredient by `:iid`	admin

Table 7: Ingredients' endpoints, source: own compilation

Methods	URLs	Actions	Access
GET	/api/menu	retrieve all Menus	admin
GET	/api/menu/user/:uid	retrieve a user's Menu by `:uid`	private
GET	/api/menu/:mid	retrieve a Menu by `:mid`	private
POST	/api/menu	create new Menu	private
PUT	/api/menu/:mid	update a Menu by `:mid`	private
DELETE	/api/menu/:mid	delete a Menu by `:mid`	private

Table 8: Menus' endpoints, source: own compilation

Methods	URLs	Actions	Access
GET	/api/day	retrieve all Days	admin
GET	/api/day/menu/:mid	retrieve all menu's Days by `:mid`	private
GET	/api/day/:did	retrieve a Day by `:did`	private

POST	/api/day	create new Day	private
PUT	/api/ingredient/:did	update a Day by `:did`	private
PUT	/api/meal/:mealid	update a day's meal by `:mealid`	private
DELETE	/api/day/:did	delete a Day by `:did`	private

Table 9: Days' endpoints, source: own compilation

Methods	URLs	Actions	Access
GET	/api/expense	retrieve all Expenses	admin
GET	/api/expense/user/:uid	retrieve all user's Expenses by `:uid`	private
GET	/api/expense/:eid	retrieve an Expense by `:eid`	private
POST	/api/expense	create new Expense	private
PUT	/api/expense/:eid	update an Expense by `:eid`	private
DELETE	/api/expense/:eid	delete an Expense by `:eid`	private

Table 10: Expenses' endpoints, source: own compilation

The server has been deployed in an Azure Cloud system. This environment displays the website on the Internet. As mentioned earlier it is important the load page speed. PageSpeed Insight has been used to analyze the performance of the website. The results are the following ones.

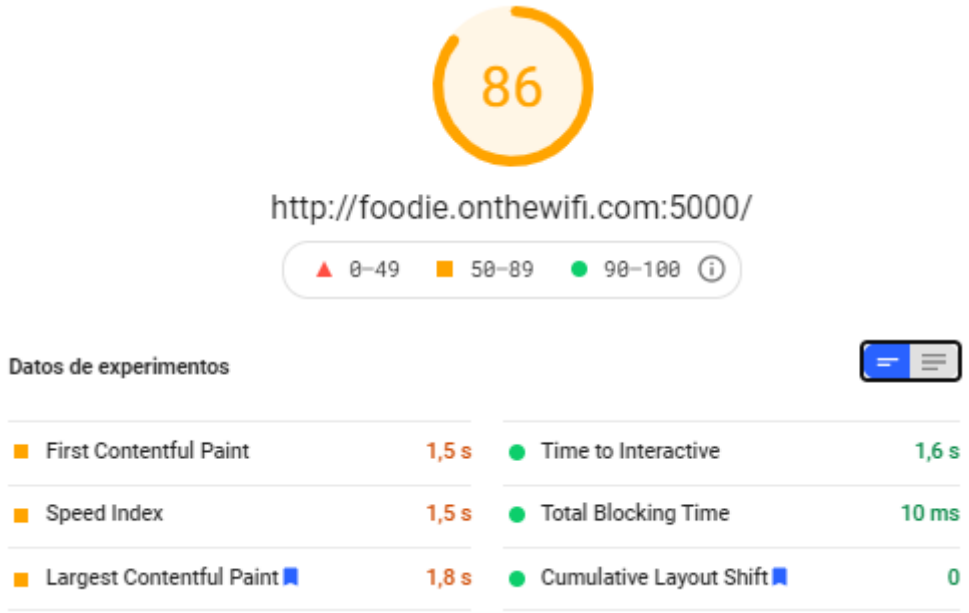


Figure 24: Page Speed Insight, source: https://developers.google.com/speed/pagespeed/insights/?url=http%3A%2F%2Ffoodie.onthewifi.com%3A5000%2F&tab=desktop

The most important data to analyze is the Time to Interactive parameter, it represents how long the user has to wait for the website to be fully functional. As mentioned before this timing had to be less than 5 seconds, and it has been achieved.

Overall, the project fulfils its purpose, to create a unified environment for cooking-related tasks. It has a user-friendly interface, highly responsive. The website adapts itself to any device screen. It is a multiplatform application

8. Budget

In this section, the budget is calculated. The analysis of the project's cost is for four months. Firstly, the working hours have been defined and then the material and services costs.

In table X, the total human cost for the project realisation is present. The annual salary assumed for a full-time junior engineer is 26000€. Therefore, a four-month, 25h/week salary adds up to approximately 5400€. The amount is calculated assuming twelve paychecks.

$$\frac{26000\ €/year * 25h * 4\ weeks * 4\ months}{40h * 4\ weeks * 12\ months} \simeq 5400€/year$$

In addition, social welfare has to be considered. It is a tax with a value of 33% of an employee salary that a company has to pay. It covers any medical injury that either occurs during working hours or on the journey to the workplace. The total amount is then 7182€.

Following, since the employee is working remotely, no furniture has to be taken into account. Nevertheless, a computer, a second screen, mouse and keyboard has been provided. The total cost of such devices is 1455€, which has a cost of 121,25€/month.

The development of the project first was on-premise. Therefore, both the database and the server were run locally, having no expense. Then to give access to the application, a cloud deployment is done.

The database is in a cloud cluster. This service consists of a MongoDB Atlas General M20 of AWS, Amazon Web Service. It has a storage of 20GB, with 4GB of RAM and a 2vCPU. The cost is 0,17€/h, assuming the cluster deployment is just a month, the total cost is 122,4€.

The server for the last month has been deployed in an Azure Cloud environment. The service is a basic dedicated environment, with 10GB of disk space, 1,75GB of RAM and one nucleus with a custom domain and hybrid connectivity. The pricing is 0,064€/h.

Concept	Time	Price	Amount
Salary	4 months	13,50€/h	7182,00€
Devices	4 months	121,25€/month	485,00€
Cloud database	1 month	0,17€/h	122,40€
Cloud server	1 month	0,064€/h	46,17€
Total			7835,57€

Table 11: Cost of the project, source: own compilation

9. Conclusions and future development:

People eat at least twice a day. Everybody needs to make time to cook. Abusing delivery or fast-food services is not a healthy attitude. Hence, the project presents a platform to help people organize their weekly meals and be conscious of their eating habits. A change in the alimentation can initiate significant transformations in peoples lives. For instance, sleep better, be more focused, boost the metabolism and many more.

Therefore, the web application permits creative people to share delicious recipes with those who desperately need inspiration. Planning weekly menus is not an easy task, but the website can streamline the process once a small community is created.

Moreover, grocery shopping without having a precise idea can be a tedious assignment. It leads people to repetitive patterns and eating daily alike. Instead, the user with a prepared menu can build a habit of purchasing the ingredients once a week. This habit prevents food waste, thus saving time and money.

The system performance is variable since it can be adapted to the server load. Azure enables server scalability, upgrading the characteristics of the cloud environment if required. The database can also increase its resources if more space is needed. Those changes depend on the system usage.

The SPA has a clean and straightforward interface to ease navigation and its interaction. In order to improve the user experience, reviews will be added to retrieve their opinion. Thus, the feedback enriches the service with perhaps additional features if demanded.

Consequently, the project's main contribution is a platform to have under control the food expense, the ingredients yearning in the fridge or the storage and most importantly, the food intake. Hence, helping people to have a more healthy and conscious life.

As future works present a system that extracts data from the user practice. The idea is to record user habits to include weekly menus, unusual recipes or even unknown ingredients based on the data extracted. This feature permits an analysis of the user's eating habits, enabling an improvement in his or her diet if desired. Eventually, having pre-design menus saves time. The nutritional value will be added to the ingredients, thus to the recipes. Therefore, the user can be more conscious of his or her food intake.

References

[1] G. Lim. *Beginning Angular 2 with Typescript*, 1st ed. CreateSpace Independent Publishing Platform, 2017.

[2] Microsoft. Typescript. [Online] Available: https://www.typescriptlang.org/. [Accessed: 14 June 2021].

[3] Mozilla and individual contributors. JavaScript [Online] https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Accessed: 24 March 2021].

[4] Google. Angular. [Online] Available: https://angular.io/. [Accessed: 20 June 2021].

[5] Facebook Inc. React. [Online] Available: https://reactjs.org/. [Accessed: 30 March 2021].

[6] MIT, Bootstrap. [Online] Available: https://getbootstrap.com/. [Accessed: 25 May 2021].

[7] Refsnes Data, HTML [Online] Available: https://www.w3schools.com/html/html_intro.asp. [Accessed: 8 June 2021].

[8] Refsnes Data, CSS [Online] Available: https://www.w3schools.com/css/default.asp. [Accessed: 20 June 2021].

[9] OpenJS Foundation. NodeJS. [Online] Available: https://nodejs.org/en/https://nodejs.org/en/about/. [Accessed: 24 May 2021].

[10] StrongLoop, IBM. Express. [Online] Available: https://expressjs.com/. [Accessed: 17 May 2021]

[11] npm Community. npm. [Online] Available: https://www.npmjs.com/about. [Accessed: 10 June 2021]

[12] MongoDB Inc. MongoDB. [Online] Available: https://mongodb.com. [Accessed: 25 May 2021]

[13] Codecademy. MVC: Model View, Controller. [Online] Available: https://www.codecademy.com/articles/mvc. [Accessed: 2 May 2021]

[14] restfulapi. What is REST? [Online] Available: https://restfulapi.net/. [Accessed: 10 May 2021]

[15] Stackify. What are CRUD Operations: How CRUD Operations Work, Examples, Tutorials & More. [Online] Available: https://stackify.com/what-are-crud-operations/. [Accessed: 10 May 2021]

[16] Auth0. JWT. [Online] Available https://jwt.io/. [Accessed: 30 May 2021]

[17] Qvault, Lane Wagner. Bcrypt Step by Step. [Online] Available https://qvault.io/cryptography/bcrypt-step-by-step/. [Accessed: 14 May 2021]

[18] MIT. Mongoose. [Online] Available: https://mongoosejs.com/. [Accessed: 12 June 2021]

[19] Guru99. Typescript vs JavaScript: What's the Difference?. [Online] Available: https://www.guru99.com/typescript-vs-javascript.html. [Accessed: 28 February 2021]

[20] Educba, Priya Pedamkar. ReactJS vs Angular2. [Online] Available: https://www.educba.com/reactjs-vs-angular2/. [Accesses: 2 March 2021]

[21] Bloomreach Inc. What Is a Single Page Application and Why Do People Like Them so Much?. [Online] Available: https://www.bloomreach.com/en/blog/2018/07/what-is-a-single-page-application.html. [Accessed: 25 May 2021]

[22] Yoav Einav, Amazon Found Every 100ms of Latency Cost them 1% in Sales. [Online] Available: https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/. [Accessed: 25 May 2021]

[23] Google. Google Keep. [Online] Available: https://keep.google.com/. [Accessed: 26 June 2021]

[24] Google. Google Calendar. [Online] Available: https://calendar.google.com/. [Accessed: 26 June 2021]

[25] Notion Labs, Inc. Notion. [Online] Available: https://www.notion.so/. [Accessed: 26 June 2021]

[26] Hello Fresh SE. Hello Fresh. [Online] Available: https://www.hellofresh.com/. [Accessed: 28 March 2021]

[27] webedia. Directo al Paladar. [Oline] Available: https://www.directoalpaladar.com/. [Accessed: 29 March 2021]

[28] Red Link To Media, S.L.. Recetas Gratis. [Online] Available: https://www.recetasgratis.net/. [Accessed: 29 March 2021]

[29] Realfooding. Realfooding. [Online] Available: https://realfooding.com/. [Accessed: 30 March 2021]

[30] Google. Firebase. [Online] Available: https://firebase.google.com/. [Accessed: 15 March 2021]

[31] Jelvix,Vitaliy Ilyukha. Differences Between Relational and Non-Relational Database. [Online] Available: https://jelvix.com/blog/relational-vs-non-relational-database. [Accesses: 14 March 2021]

Appendices

A.1. Work Packages

Project: WEB development	WP ref: 1
Major constituent: Learning technologies	Sheet 1 of 1
Short description: Analyze and learn the different technologies that will be used to develop the WEB application. And structure the project.	Planned start date:19/02/2021 Planned end date:19/04/2021
	Start event:19/02/2021 End event:19/04/2021
Internal task T1: Learn Typescript and program the first scripts to fully understand the technology. This will be done following a tutorial. Internal task T2: Learn Angular 2 with a book to understand the framework in which the web application is going to be developed. Internal task T3: Learn NodeJS and MongoDatabase to implement the back end of the web application. Internal task T4: Define the web design, requirements and specifications	

Project: WEB development	WP ref: 2
Major constituent: Frontend development	Sheet 1 of 1
Short description: Design and implement the front end of the web application.	Planned start date:19/04/2021 Planned end date:14/06/2021
	Start event:19/04/2021 End event:14/06/2021
Internal task T1: skeleton of the web. Implement the basic structure of the web application. Which means having a server, connected to a database, and a home page. Internal task T2: Program the different views of the SPA (Single Page Application). Internal task T3: Create a navbar to navigate through the views, and program the Angular routing. Internal task T4: Log in, Sign in and authentication programming. Internal task T5: Program the style of the application.	

Project: WEB development	WP ref: 3
Major constituent: Backend development	Sheet 1 of 1
Short description: Program a web server and create and design the database.	Planned start date:10/05/2021 Planned end date:07/06/2021
	Start event:10/05/2021 End event:07/06/2021
Internal task T1: Design the Mongo models required. Internal task T2: Program the request to the database. Internal task T3: Program the connections to the database.	

A.2. Database collection schemas

User schema:

Name	DataType	Description	Relation
_id	Id	Key	
email	String	Email (unique)	
password	String	Encrypted password	
firstname	String	Name	
lastname	String	Last name	
birthyear	Number	Birth year	
role	String	Role (‘admin’,’user’)	
meals	String[]	Meals (‘Breakfast’, ‘Snack’, ‘Lunch’, ‘Dinner’)	
diet	String	Diet (‘Omnivorous’, ‘Vegetarian’, ‘Vegan’)	
recipes	Id[]	Saved or owned recipes Id	Many to many
groceries	String[]	Missing ingredients name’s	Many to many

Table 12: user , source: own compilation

Recipe schema:

Name	DataType	Description	Relation
_id	Id	Key	
name	String	Name	
timing	Number	Duration of recipe cooking	
guest	Number	Number of portions	
meal	String[]	Kind of meal (‘Breakfast’, ‘Snack’, ‘Lunch’, ‘Dinner’)	
diet	String	Diet (‘Omnivorous’, ‘Vegetarian’, ‘Vegan’)	
ingredients	RecipeIngredient[]	Name of the ingredients required	Many to many
image	String	Image of the recipe	
videoRecipe	String	Link to videoRecipe	
creator	String	Email of the user that has written the recipe	One to many
saved	String[]	Email of the users that have saved the recipe	Many to many

Table 13: recipe, source: own compilation

Ingredient Schema:

Name	DataType	Description	Relation
_id	Id	Key	
name	String	Name	
diet	String	Diet (‘Omnivorous’, ‘Vegetarian’, ‘Vegan’)	

Table 14: recipe, source: own compilation

Menu schema:

Name	DataType	Description	Relation
_id	Id	Key	
_user	String	Email of the owner	One to many
title	String	Title	
description	String	Explanation	

Table 15: menu, source: own compilation

Day schema:

Name	DataType	Description	Relation
_id	Id	Key	
_menu	Id	Email of the owner	One to many
day	String	Title	
meals	Document[]	Explanation	
meals._id	Id	Key of embedded document	
meals.meal	String	Meal of the day	
meals.recipes	Id[]	Recipes of the meal	Many to many

Table 16: menu, source: own compilation

Expenses schema:

Name	DataType	Description	Relation
_id	Id	Key	
_user	Id	Email of the owner	One to many
ingredients	String[]	Ingredients name’s bought	
date	DateTime	Date of purchase	
expense	Id	Amount of the purchase	

Table 17: menu, source: own compilation