NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
OF
COMPUTER SCIENCE

JOÃO MIGUEL VIEGAS MURGEIRO

Master in Computer Science

WEB-IDE FOR LOW-CODE DEVELOPMENT IN OUTSYSTEMS

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
November, 2021

DEPARTMENT OF
OF
COMPUTER SCIENCE

WEB-IDE FOR LOW-CODE DEVELOPMENT IN OUTSYSTEMS

JOÃO MIGUEL VIEGAS MURGEIRO

Master in Computer Science

Advisers: Miguel Carlos Pacheco Afonso Goulão
Associate Professor, NOVA University of Lisbon
Carlos Sousa
Head of Product Architecture, OutSystems

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
November, 2021

Web-IDE for Low-Code Development in OutSystems

Acknowledgements

I would like to start by thanking Faculdade de Ciências e Tecnologia from NOVA University and, in particular, the Informatics Department for, over the course of the last five years, providing me all the necessary skills and knowledge that will need for my professional career. Thank you also to OutSystems for offering me this opportunity and scholarship.

Fundamental to this thesis's success, I want to thank both my advisers Carlos Sousa and Miguel Goulão for all the patience and guidance you have given me throughout these months. The fact that you were always available to help me, and your incredible dedication in explaining and discussing the various aspects of the work, motivated me to perform the best I could but and provided a vital contribution. This work is also yours. Thank you also to everyone at OutSystems who contributed to this thesis, your friendliness has been much appreciated.

To all my colleagues, but in particular to my mates Duarte Almeida, Miguel Lourenço, Tomás Farrajota and Tomás Pereira for always being there throughout the entirety of the course. Not only for all the group projects and help you have given me but also for the unforgettable times I got to share with you guys.

Last, but most importantly, I need to thank my parents: Ana and Alexandre. I very much value all the sacrifices you have made to provide me with the best conditions and opportunities for me to thrive and achieve my goals. Your hard work, attention and love have inspired me to be the person I am today, and I will be eternally grateful for that. To my sister Catarina, grandmother Salete, grandfather Orlando and aunt Regina for the unconditional support and, in particular, a special thank you to my grandmother Margarida, who I know would be incredibly proud in seeing me achieve this life objective. Finally, to my beautiful Carolina for your wholehearted support and for always believing in me.

"The pessimist resembles a man who observes with fear and sadness that his wall calendar, from which he daily tears a sheet, grows thinner with each passing day. On the other hand, the person who attacks the problems of life actively is like a man who removes each successive leaf from his calendar and files it neatly and carefully away with its predecessors, after first having jotted down a few diary notes on the back. He can reflect with pride and joy on all the richness set down in these notes, on all the life he has already lived to the fullest." (Viktor E. Frankl)

Abstract

Due to the growing popularity of cloud computing and its numerous benefits, many desktop applications have been, and will continue to be, migrated into the cloud and made available through the web. These applications can then be accessed through any device that has access to a browser and internet connection, eliminating the need for installation or managing dependencies. Moreover, the process of introduction to the product is much simpler, faster and collaboration aspects are facilitated.

OutSystems is a company that provides software that enables, through an Integrated Development Environment (IDE) and a specific Low-Code language, users to securely and rapidly build robust applications. However, there are only available desktop versions of this IDE. For this reason, the objective of the proposed thesis is to understand what would be the best path for developing a Web-based version of the IDE.

To achieve this, it is important not only to understand the OutSystems Platform and, more specifically, the architecture of the Service Studio IDE, which is the component IDE provided by the product, but also to explore the state-of-the-art technologies that could prove to be beneficial for the development of the project.

The goal of this work is to debate different architectural possibilities to implement the project in question and present a conclusion as to what the adequate course of action, given the context of the problem. After distinguishing what are the biggest uncertainties and relevant points, a proof of concept is to be presented accompanied with the respective implementation details.

Finally, this work intends to determine what would be a viable technological architecture to build a Web-based IDE that is capable of maintaining an acceptable performance, similarly to Service Studio IDE, while also insuring that the this system is scalable, in order to be able to provide the service to a large amount of users. That is to say, to present a conclusion regarding the feasibility of the project proposed.

Keywords: OutSystems, Integrated Development Environment, OutSystems IDE, Web-IDE, Web Development, Online IDE, Cross-platform, Cloud Computing

Resumo

Devido ao aumento de popularidade de tecnologias de computação *cloud* e as suas inúmeras vantagens, aplicações *desktop* estão e vão continuar a ser migradas para a *cloud* para que possam ser acedidas através da *web*. Estas aplicações podem ser acedidas através de qualquer dispositivo que tenha acesso à *internet*, eliminando a necessidade de instalação e gestão de dependências. Além disso, o processo de introdução ao produto é simplificado, mais rápido e a colaboração é facilitada.

A OutSystems é uma empresa que disponibiliza um *software* que faz com que utilizadores, através de um IDE e uma linguagem de baixo nível, possam criar aplicações robustas de forma rápida e segura. No entanto, atualmente só existem versões deste IDE para *desktop*. Como tal, o objetivo da tese proposta é perceber qual será a melhor forma de desenvolver uma versão do IDE sobre a *Web*.

Para alcançar isto, é importante não só compreender a Plataforma OutSystems e, mais especificamente, a arquitetura do Service Studio IDE, que é o principal componente disponibilizado pelo produto, mas também explorar as tecnologias estado de arte que podem ser benéficas para o desenvolvimento do projeto.

O objetivo deste trabalho é debater diferentes arquiteturas possíveis para a implementação do projeto e concluir qual será o curso de ação adequado, dado o contexto do problema. Após distinguir quais são os maiores pontos de incerteza, uma prova de conceito é apresentada juntamente com os respetivos detalhes de implementação.

Finalmente, este trabalho tem como intenção detalhar uma arquitetura tecnológica viável para construir um IDE na *web* capaz de manter uma performance aceitável, semelhante à do Service Studio IDE, e garantir a escalabilidade do sistema, de forma a conseguir oferecer o serviço a um número elevado de utilizadores. Por outras palavras, apresentar uma conclusão em relação à viabilidade do projeto proposto.

Palavras-chave: OutSystems, Ambiente de desenvolvimento integrado, OutSystems IDE, Web-IDE, Desenvolvimento Web, Online IDE, Plataforma Cruzada, Computação em nuvem

Contents

List of Figures

List of Tables

Acronyms

4GL	Fourth-generation programming language i, 6
API	Application Programming Interface i, 14, 19, 22, 44, 45, 48, 51, 52, 60, 63, 71, 72
AWS	Amazon Web Services i, xii, 19, 28, 29, 46, 47, 48, 49, 50, 51, 52, 58, 60, 61, 62, 63, 67, 68, 70, 71, 72, 74, 78, 79, 81, 82, 84, 85
CDK	Cloud Development Kit i, 46, 47, 48, 55, 63, 79
CDN	Content Delivery Network i, x, 71, 72
CI/CD	Continuous Integration/Continuous Delivery i, 73, 74
CLI	Command Line Interface i, 47, 48, 63
CORS	Cross-origin Resource Sharing i, 48
CRDT	Conflict-free Replicated Data Type i, xii, 24, 25, 26
CSS	Cascading Style Sheets i, 17, 35, 79
DOM	Document Object Model i, 35
GSD	Global Software Development i, 14
HTML	HyperText Markup Language i, 17, 35, 36, 79
IDE	Integrated Development Environment i, vii, viii, xii, 1, 2, 3, 5, 7, 8, 9, 10, 11, 12, 14, 21, 22, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 39, 40, 44, 52, 54, 55, 62, 63, 65, 67, 68, 71, 72, 76, 77, 78, 80, 81, 83
IT	Information Technology i, 19, 27
JDK	Java Development Kit i, 2
JSON	JavaScript Object Notation i, 37, 38, 55

MVC	Model-View-Controller i, 17
OML	OutSystems Markup Language i, 81
OS	Operating System i, 12, 19, 30
OT	Operational Transformation i, xii, 24, 25
PC	Personal Computer i, 30
POC	Proof of concept i, xiii, 3, 4, 17, 31, 32, 33, 35, 36, 37, 39, 45, 46, 47, 50, 51, 52, 55, 56, 57, 60, 63, 65, 67, 70, 76, 77, 79, 81, 82, 83
SDK	Software Development Kit i, 61
SVG	Scalable Vector Graphics i, 35
TCP	Transmission Control Protocol i, 60
UI	User Interface i, 6, 8, 9, 11, 12, 16, 32, 63
UML	Unified Modeling Language i, 19
UUI	Universally unique identifier i, 36, 37, 57
UWP	Universal Windows Platform i, 12, 16
VDI	Virtual Desktop Infrastructure i, xii, 5, 17, 18, 19
VM	Virtual Machine i, 17, 18
VPN	Virtual Private Network i, 18
WPF	Windows Presentation Foundation i, 8, 9, 16
XAML	Extensible Application Markup Language i, 16, 17

1 Introduction

This chapter presents an introduction to the thesis, by contextualizing the problem and laying out the incentives and advantages of a Web-IDE. After this, the objectives of the thesis are detailed, as well as what are the contributions expected by the end of the project. Finally, the structure of the remainder of the document is defined.

1.1 Context and Description

Using a visual programming language, the OutSystems Platform allows developers to create state-of-the-art Web and mobile applications while having little concern for the complex technologies that implement those applications. Application development with OutSystems is much faster, up to 70% [1], and easier [2] than with traditional software development methods and is being increasingly adopted.

OutSystems enables teams of developers to rapidly create new applications and deploy them to production. The OutSystems platform is centered around different components and services that simplify the Low-code experience to the developers. Today the platform already provides an IDE tool for its developers. The IDE is called Service Studio and it is the development tool used to implement Low-code applications. This tool is a desktop-based application that is downloaded from the OutSystems Web site and then installed on the developer's computer, allowing it to run on Windows and Mac environments.

To increase even more the developers worldwide and simplify the usage of OutSystems in any system, it would be valuable to have a Web version of the IDE that allows developers to use the development environment using any device, from anywhere [3] and only the need to have a modern Web browser application to be able to develop rich Low-code applications.

The Service Studio Web IDE, inserted within the context of a Platform as a Service (PaaS) [4], should be built using the current state-of-the-art Web technologies and libraries and also support the top Web browsers across the different systems (Windows, macOS, Linux).

1.2 Incentive

Cloud computing, in this case, more specifically, Infrastructure as a Service (IaaS), has been emerging as an important computational model for satisfying the increase of resources and infrastructure that present services require [5]. It is specified as an on-demand service that can provide access to a pool of shared resources, information or software, that can be promptly released with minimal effort, according to the client's needs [6]. It does not require the end-user to worry about any type of management, configuration, or scaling, allowing an uncomplicated experience.

Cloud providers have data centers with monumental storing and computing power. Therefore, they can supply the users a better computing performance, at a reduced cost [7]. As a consequence, there has been a growing trend of adapting original desktop applications into an online context and provide them as a service, enabling users to swiftly access them on the go [8]. Such an example of this is the case of Google Docs [9].

It is not a trivial task to set up and maintain an efficient development environment, especially for initiate programmers [10]. If a user wants to develop, for example, a given Java project, it will be required to download a Java Development Kit (JDK) to be set up on their machines, and this development environment, in turn, will require the download, lengthy installation process and setting up of a compatible IDE.

This can be very inconvenient and, besides, for these operations to be successful, the user needs to ensure that its machine has the necessary memory space and computing power to efficiently execute the environment [11]. This process can become even more difficult if more than one environment is required to be installed in the same machine, for example, for developers that are required to work on separate projects, with distinct scopes. Furthermore, if the user decides to work on different machines, the extensive procedure must be repeated.

Thus, a cloud-based IDE provides a flexible and modern solution for the problems previously described. If the installation, configuration, and dependency management are done in a server, which in turn can be provided to the users through a service, the only requirements are access to the internet and the installation of a browser. As a result, the user doesn't need to go through the time-consuming installation process, or to allocate any memory space [12], since availability and scalability of the resources will be automatic and insured. Ultimately, this solution will not only simplify the experience and thereby attract new users but will also allow them to spend their limited time programming and being productive.

In fact, there have been empirical studies that demonstrate that the use of a cloud-based IDE will enhance the software development experience by reducing costs and overall complexity involved [13].

To conclude, an area where Web-IDEs has proven to be highly advantageous is in e-learning and distance learning, a subject that has never been more relevant. By providing an online and interactive environment for students to learn to program, accessible through a personal login, all the daunting installation steps are eliminated and interest is maximized [14] as they can, without any delay or obstacle, start experimenting and coding [15].

1.3 Objectives

The general objective is to increase the OutSystems developer community and reach even more developers worldwide. With this in mind, the Web IDE should also provide a similar experience and functionality as the Windows and Mac counterparts without the need to even install any application on the developers' computer. However, given the context of the thesis, it is not feasible to produce a fully functional product, but rather, a prototype that enables to arrive at a conclusion regarding the feasibility of the project.

For this reason, the goals of the proposed thesis are the following:

- Characterize the current architecture of the OutSystems platform. How the platform is built and how the systems and tools interact with each other.
- Distinguish the significant differences between the OutSystems Low-code development model and traditional code-based software development.
- Identify the major architecture quality attributes, non-functional requirements, and business cases that drive the design of the OutSystems Web-IDE solution.
- Evaluate what is the state-of-the-art in the industry in terms of Web technology frameworks/libraries and if and how they can be applied to the OutSystems context.
- Identify the main architectural challenges associated with creating a new Web-based IDE for OutSystems.
- Propose, prototype, and evaluate one or more solutions to those challenges, identifying the pros and cons of each approach.
- Present conclusions regarding the best path to creating a Web-based IDE to the major stakeholders of the Web-IDE initiative.

1.4 Key contributions

Regarding the key contributions that this thesis aims to produce, a functional POC of the IDE is constructed that allows to clarify the existing implementation uncertainties. The key contributions of this work are:

- How to implement the local persistence of the model inside the browser.
- Implementation of a validation method, based on semantic rules.

- Understand what is the best format to save the model in a server-side central system, respecting the cloud strategy of OutSystems.
- Detail the architecture of the cloud-based central system that can be deployed, in order to ensure large scalability and following good structural practices.
- How to correctly implement a real-time collaboration mechanism that enables to maintain a low latency experience.
- A practical POC that enables to arrive at a conclusion regarding the feasibility of developing a Web-based version of Service Studio, including a description of what are the main challenges and uncertainties of this solution, as well as future work required.

1.5 Structure

The remaining structure of this thesis is organized into the following chapters:

- **Chapter 2 - Background**: Investigation of technologies and paradigms that are relevant to the contribution of a proposed solution.
- **Chapter 3 - Related Work**: Here, use cases and pertinent related work are presented to help formulate a general idea of what a successful approach might be.
- **Chapter 4 - Requirements**: The system requirements are presented as well as a structured guideline of the different phases that are to be implemented.
- **Chapter 5 - Implementation**: Here, a discussion regarding the various alternative technologies to include in the POC are presented, as well as the various architectural decision that were taken and, moreover, the implementation details.
- **Chapter 6 - Evaluation**: This chapter presents the final product of the POC developed and, in addition, an evaluation process is also presented
- **Chapter 7 - Conclusion**: Finally, an synopsis of the thesis is presented as well as proposals for future work.

2 Background

A brief overview of the complete architecture of the OutSystems Platform will be presented as well as a detailed description of the Service Studio IDE; the reasons why it would be extremely useful for OutSystems to change the paradigm of its software from locally installed to Web-based. Next, the state-of-the-art technologies that are relevant within this context are detailed, namely, WebAssembly [16], popular cross-platform technologies, and VDIs, an alternative architecture considered.

Finally, the framework for the implementation of a diagram of the architecture is presented, including the reasons why the selected solution is considered to be the best choice.

2.1 OutSystems

OutSystems is a Low-code application development and delivery platform that enables fast, easy, and agile development of small and large applications, that is available as a Platform as a Service (PaaS) [17]. Not only does this platform allow to seamlessly deliver applications across most platforms and provides tools to accelerate and ease the integration of applications with existing clouds and local databases but also permits continuous development, delivery, and management.

2.1.1 Why OutSystems?

In 2001, the year OutSystems was founded [18], most companies had a fundamentally different way of approaching the development of an application. The planning stages of these projects were immense because the goal was to develop it right at the first time so that no further changes had to be added. However, this methodology demanded long periods of development, and previewing every single eventuality is just not feasible, hence, the risk of something failing added further delay.

For this reason, the agile manifesto [19] was introduced, with the goal of simplifying the software development paradigm by prioritizing early and continuous delivery, bringing both the costumer and the developer together and structure projects so that changes are always possible.

Within this context, Fourth-generation programming language (4GL)s [20] gained traction. The concept of 4GLs, developed from the 1970s through the 1990, aims to provide a high level abstraction language that is intuitive, friendly, versatile and yet powerful [21].

Nowadays, 4GL systems have emerged as Low-code development platforms, which are development environments used to create software with the aid of graphical User Interface (UI)s. The goal of these platforms is to reduce the amount of skill-set required to produce business applications, consequently increase productivity. The term "Low-code" was first used in 2014 by Forrester Research [22].

OutSystems is an example of a successful [23] Low-code development platform and was born out of the necessity of accelerating the development process and confer flexibility to it. OutSystems' approach is simple, instead of planning every detail and the changes costing enormous amounts of time, it would be simpler to implement a mechanism that enables cheap and fast changes, independent of the size of the application.

Nowadays the technological paradigm has changed and for a business to remain competitive, it needs to adapt and digitally transform its user interaction. Mobile has undoubtedly played a major role in this shift [24]. There is now a large group of young people that will not hesitate in saying that they prefer a mobile interaction and, logically, this tendency will only increase in the future. Not only this, but customers now want applications to be delivered in very short time frames and trend cycles happen at a much faster rate, consequently, in order to satisfy the demand, it is necessary to implement shorter delivery cycles [25].

OutSystems, as a company, understands these requirements and satisfies them by providing a complete, full-cycle Low-code application development platform that simplifies every stage of the app development and delivery process. The core characteristics of this platform are:

- **Rapid productivity**, through an intuitive visual model.
- **Multi-channel development**, that allows an application to run on a large variety of devices and platforms.
- **Open-Platform**, enabling integration with already developed back-end systems.
- **Reduced Cost**, automated dependency management, and holistic application life-cycle management significantly decreased management costs.
- **No lock-in**, applications are generated with standard architecture and no specific runtime interpreters or engines are required.

2.1.2 OutSystems Platform Architecture

In this section, the several components and tools that compose the OutSystems Platform, including the Platform Server, development, administration, and operational tools [26] are presented. These can be observed in figure 2.1. To clarify, among the several components of the OutSystems Platform, the only one that is to be installed locally by the users is the Service Studio. For this reason, the main focus of the thesis will directed towards this specific component.

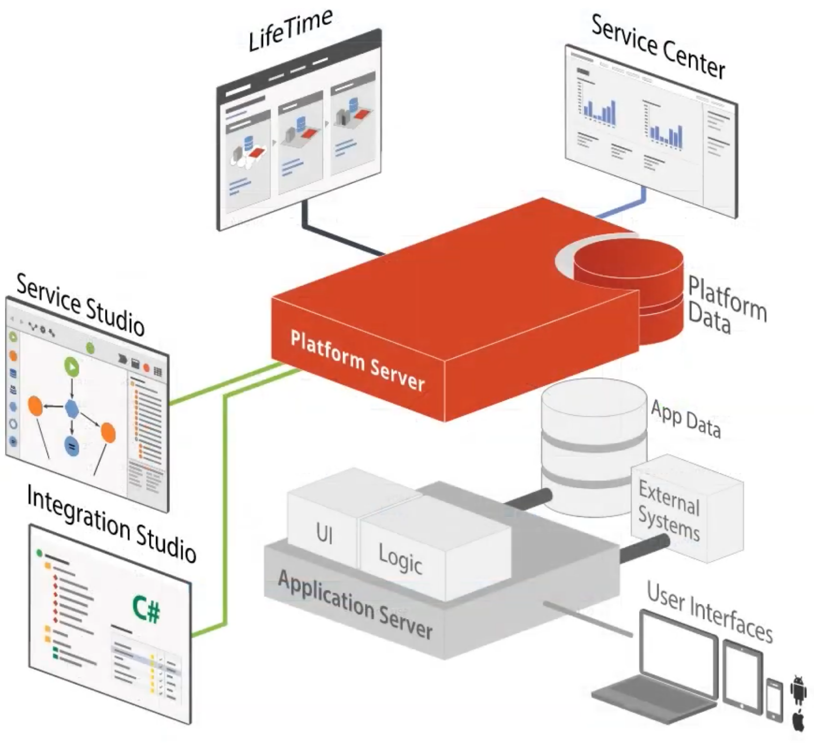


Figure 2.1: Architecture of OutSystems Platform [27].

- **Service Studio**: Is the visual IDE that users will exploit to create Web and Mobile Apps (figure 2.2). Here, it is possible to assemble and change these applications through a drag-and-drop interface, creating and publishing applications to the Platform Server.
- **Platform Server**: This is a set of servers that will compile, deploy, manage, run and monitor client's applications. It is composed of a code generator responsible for generating native optimized code from the application modeled in the IDE, the deployment services that will deploy the generated application to a standard Web application server, and application services that will manage the execution of scheduled batch jobs and provide logging services.
- **Platform Data**: Where the client's applications will be stored.
- **Integration Studio**: Is another development environment that was conceived to integrate external extensions with the OutSystems Platform. As part of the integration process, this tool creates representations of external artifacts inside the OutSystems

world. Once added, these extensions can be used and published as regular internal resources.

- **Application Server**: The Application Server will make use of traditional databases and external systems to run the applications created.
- **Service Center**: Is a Web Application accessible via a Web browser that consists of a platform server management and administration console. Here, it is possible to view and configure the platform server, inspect what applications are available and logs generated by the platform, monitor the server, and even configure environment settings.
- **LifeTime**: Whereas Service Center enables to manage an individual server or environment, LifeTime is another Web App that allows managing the full application lifecycle across multiple environments and not only manage users and infrastructure but also collect analytics.

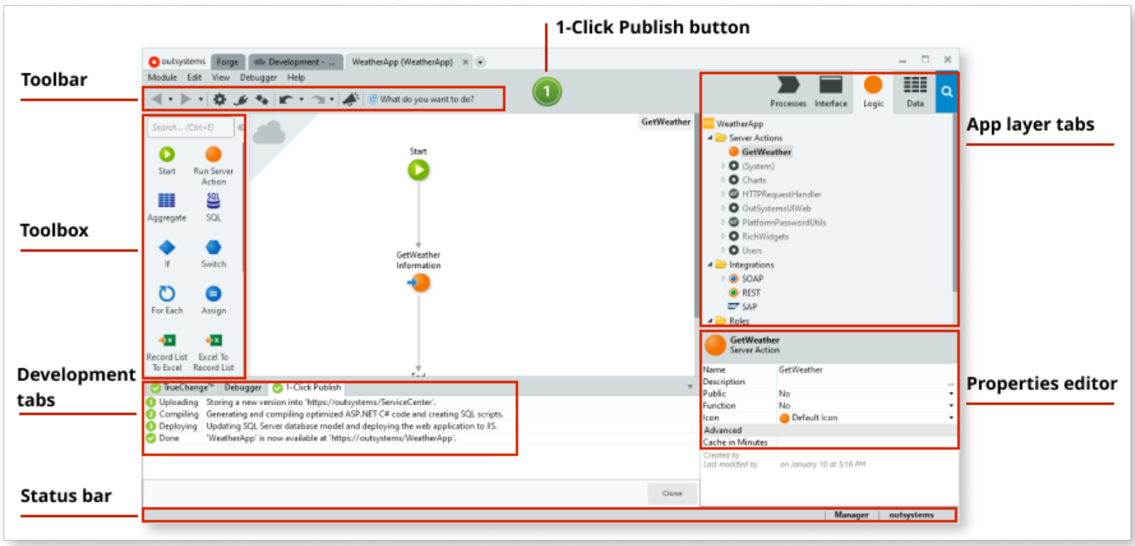
2.1.3 Service Studio Architecture

Service Studio, OutSystems' visual IDE (figure 2.2), allows users to create Web and mobile applications. This tool, through a drag-and-drop interface, allows users to create all pieces of the application stack, including the data model, application UI and logic, business logic and processes, security policies, and enables SOAP and REST Web Services [28]. Moreover, this environment also incorporates a "full reference-checking and self-healing engine" to ensure that the application being developed remains consistent.

Service Studio was constructed and released in the year 2009, based on the .NET platform [29]. Until 2017 no significant infrastructural developments were implemented, but this year, the IDE Value Path and UI Framework were added to enhance the quality of the product and user experience. In 2019, efforts were initiated to produce a cross-platform program, or "X-Platform", and bring the originally Windows-based program also to MacOS, which has been concluded in 2021. Finally, the future ambition will be, as this proposal describes, to adapt the program so that it can run on the Web.

Service Studio current status comprises more than 80 views, between them the UI Editor, Flow Editor, Properties and Login, 4 view frameworks, Windows Presentation Foundation (WPF) [30], React [31], Angular [32] and Avalonia UI Framework [33], 2 languages, C# and TypeScript [34], and 3 versions, the stable and beta versions based on .NET 4.7.2 [35] and the cross-platform one that is being developed with .NET Core [36].

The architecture of Service Studio can be separated into three distinct and well defined layers. The Model layer, which is where all the logic of operations executed over the data model is implemented, such as saving, validation, or editing of data. The View layer, which corresponds to the visual UI presented to the user and all inputs included. And lastly, the Presenter layer will act as an intermediary between the Model layer and the View layer, responsible for the management of actions.



```
TrueChange™   Debugger   1-Click Publish
Uploading   Storing a new version into 'https://outsystems/ServiceCenter'.
Compiling   Generating and compiling optimized ASP.NET C# code and creating SQL scripts.
Deploying   Updating SQL Server database model and deploying the web application to IIS.
Done        'WeatherApp' is now available at 'https://outsystems/WeatherApp'.
```

```
GetWeather
Server Action
Name          GetWeather
Description
Public        No
Function      No
Icon          Default Icon
Advanced
Cache in Minutes
```

Figure 2.2: Service Studio [37].

The fastest way to reach the “X-Platform” goal is a multi-step journey which includes the migration of all WPF views to cross-platform view technology, for instance, React as it is a widely popular language, including among developers inside the company and it is very easy to learn, developing a native Shell, port the .NET codebase to .NET Core and maintain a single code base as multiple versions would be unfeasible to manage.

To implement this particular solution, the “Hybrid” approach is being developed, based on .NET Core, where inside each window or tab in the application, there is a browser page. Within this context, Avalonia UI is responsible for the windowing system and React/TypeScript for the UI framework since it is easy to bootstrap, develop, maintain and allows for isolated views so that in case an error occurs with a particular view, it will not interfere in the others.

Nevertheless, all views share a common visual language. It is also relevant to note that every tab has its dispatcher queue that simulates the existence of one UI thread so that potential concurrency issues are avoided. The objective will be to fully substitute the current version of the software with the cross-platform version, once it’s fully developed.

2.2 Web-IDE motivation

At this point, the goal is merely to evaluate the possibility of developing a Web-IDE, this is not a certain project, in fact, some OutSystems senior developers are not interested in this particular solution because there is always an inherited delay observable in these Web-based solutions. This might be caused by some connection failure, too much time loading a certain resource, among other reasons, and will be translated into a non-productive experience for the developer.

On the other hand, assuming that the user experience requirements can be met, the main reasons why OutSystems should consider the development of a Web-IDE are friction, cross-platform and skill set and collaboration.

2.2.1 Friction

The main reason that is currently being discussed is the phenomenon known as friction.

Friction is the measure of difficulty encountered while completing a certain task, in other words, anything that prevents or dissuades customers from buying or using products or services [38]. In this case, the task would be to successfully engage and use the OutSystems Platform.

Nowadays friction is an extremely relevant topic because of the increasing availability of different solutions, in this case, the focus is a Low-code development tool, but this is also true for many other contexts. The attention span of a potential user is, increasingly, a highly valuable resource that must be capitalized upon.

The importance of friction can be illustrated through the following hypothetical scenario: If an average user is searching for the best technology to satisfy their needs, a simple query in a searching engine will likely provide him with numerous options, consequently, upon entering a website, he will then register and encounter himself on the main page.

Let´s now consider two distinct options that are available.

- In website A, there is a button available that after being pressed will begin the process of downloading a tool, however, this process will only take place if the operating systems of the downloadable artifact correspond to that of the user's, if the user has the required permission for such action and if the user's machine has the available disk space to store it. In case these conditions are satisfied, the user will then select the folder to store it and await the processes of installation, while this happens, in all probability, the not yet satisfied user will continue seeking other alternatives and might even find one before the process of installation is concluded, thereby canceling it.
- On the other hand, on website B, after registering, the user is automatically redirected to a ready-to-test environment, with all tools necessary to evaluate if this is the solution he intends to use. In this case, because a satisfactory solution is readily presented, the user will not continue to search for other alternatives.

In this example, the installation process is translated into a considerable amount of friction and demonstrates how crucial it can be in acquiring new users into the platform.

Perhaps a good and well-known example of how powerful the reduction of friction can be to achieve success is Uber [39]. Uber revolutionized the private transportation business thanks to a simple and intuitive mobile application that, among other attributes, simplifies the ordering process, reduces the uncertainty of when and what type of car will arrive, simplifies the payment process, makes the pricing transparent, and provides tracking and seamless on-demand availability. All these characteristics are effective ways of showing how this platform reduces friction that might exist for a potential customer. To illustrate this, in 2015, only in the city of San Francisco, Uber had a total revenue of $500 million, compared to the $140 million of the traditional taxi market, and tripling each year [40] .

According to a case study published on MarketingExperiments' blog [41], reducing friction leads to a much better user conversion rate and, consequently, business success. A simple experiment was made where two quote request interfaces were tested. One of them had a very poor user interface and the process required a large number of "clicks", which included the filling of numerous input variables and required navigation between pages. The other one was much simpler, with fewer input fields, information was provided to help fill the necessary inputs and the number of "clicks" was very low. It should not come as surprise to learn that the simpler interface outperformed the alternative by 262.3%, that is, the conversion rate jumped from 2.4% to 8.8% and therefore, simply by providing a simpler and friendly user experience, the user base of a particular platform or service could drastically increase.

With the previous experiment in mind, it is safe to assume that there is a large percentage of newly registered users that don't even press the download button, because the downloading process undoubtedly creates a considerable amount of friction, let alone experiment the program.

In a digital world with a wide range of options available, independently of the technology sector, friction-less experiences have a valuable role in determining the success of any business. Here is where the Web-based IDE can be exceptionally successful in decreasing the drop rate and promptly captivating the attention of users and, as a consequence, have a much higher success rate in recruiting them. This is because it is much more appealing and simple to promptly enter the IDE environment and quickly engage with the solution.

2.2.2 Cross-platform and Skill Set

In the world of software development, there are numerous distinct products with their own operating systems. Consequently, there are divergent development technologies for each of them, all with their language and characteristics. An example is the Swift language [42] which is exclusively compatible with iOS, and, identically, Koptin [43] with Android [44].

Unfortunately, in the past, if the goal of a company was to reach the highest percentage of users, that is, to reach users residing in different operating systems, it would be required to maintain distinct adapted versions of the same product, for each platform.

This is highly disadvantageous because, first of all, it is expensive to create and maintain multiple product versions [45]. Secondly, it is extremely costly for an organization that has the intent of publishing an application to develop the UI for distinct platforms and therefore, to have multiple teams of developers, especially when taken into consideration the lack of qualified software engineers [46], each with his/her skillset. For example, one is specialized in Android, another one that focuses on iOS and even a third one with Universal Windows Platform (UWP) [47], which undoubtedly yields very poor results. Finally, there is a compromise in uniformity, meaning that each platform has its UI, with its specific features and widgets, which means that users on one type of device will have a different experience than others, despite technically being the same product.

Because of these unavoidable setbacks, recently there has been a lot of push to cross-platform UI technologies. The concept of cross-platform is born out of the commodity of using only one technology to develop software compatible with more than one platform. An example of a widely used cross-platform technology is React Native [48] which allows the creation of native applications not only for Android but also for iOS.

The fact that the code can be developed in one technology and then translated simultaneously to several other platforms is helpful because now the same engineer can develop not only a Web application but also the mobile applications needed, despite the platform. As a result, expenses are reduced since there is only the need to develop and maintain one version of the product, companies are able to reach more users and interface consistency increases.

Currently, the OutSystems Platform is fully available to Windows users which corresponds to approximately 75% of the desktop operating system market share worldwide [49]. However, if considering only software developers, which is OutSystems' target audience, this percentage abruptly drops to about 45% (according to a survey done by Stack Overflow in 2020 [50]) and has been decreasing over the recent years.

OutSystems has recently developed a macOS (Apple, 16% of the desktop operating system market share worldwide [49] but 27% of software developers [50]) version of this platform, however, it is not as well supported as the Windows version.

This raises the first problem, despite both applications being very similar in the sense that most of the back components are reused, there are still more than one version live, meaning that an upgrade in an Operating System (OS) will inevitably lead to a rehabilitation of the platform, consequently highlighting the need of accompanying the technology paradigms to ensure that the platform is performing correctly. Moreover, despite the percentage of the remaining desktop operating systems being comparatively small (25%, therefore not representing as big of a problem as friction) it still a substantial problem as it represents a large number of potential users that simply don't have access to the platform.

A Web-based IDE would remove these issues [51] because as long as the user has access to a supported browser, it will also have access to the OutSystems Platform, regardless of the device and OS.

The fact that the application is Web-based also has the advantage of eliminating the need for the client to update it, the process can be automated, "behind the scenes", where the users will not have to worry about any action and consequently providing the user base, seemingly effortlessly, with the latest version of the product at any time, on any device.

2.2.3 Collaboration

Although the collaboration topic is not the focus of this thesis, it is still worth mentioning.

The use of collaboration tools is embedded in the daily life of every successful company and this is especially crucial in the context of software engineering [52].

Humans can have a great memory. Nonetheless, it can be overwhelming to try to manage a big programming project, remember all scopes and capabilities required for it, manage dependencies, artifacts, code, and even engineers [53] and it is extremely easy to lose track of progress. The benefits that collaboration provides can range from better communication between team members to organizing projects more effectively, stronger workflow management, creating a scheduled work environment, and even improving the required time for the elaboration of a project [54].

In fact, collaboration is at the heart of not only the Internet but also of countless applications and software because by making technology publicly available [55], the correct term to describe this is open-source, meaning that the code can be modified by whoever is interested in participating, and therefore a public community is formed, it improves the experience of all users by removing barriers between innovators and allowing the free exchange of ideas within a creative community. Moreover, companies not only share large portions of their code but even encourage users to find vulnerabilities within them, such as Google [56], because it is much more efficient to reward a programmer capable of finding such bugs than hoping that they don't exploit them for their own benefit.

Despite the OutSystems Platform not having included collaboration functionality, there are still numerous technologies that can help with this process such as GitHub [57], the most famous and most used collaboration tool [50] that allows developers to build, ship, and maintain their software. Thereby, enabling an OutSystems project to be easily integrated and managed in a collaborative environment.

However, the collaboration provided by GitHub does not give the user the sense of "close collaboration", in other words, with the example of Google Docs [9] in mind, if two clients open the same text document, it is possible to visualize in real-time the alterations made by each other and actively engage in the same piece of document as if physically present, which is an extremely relevant subject, especially during a pandemic. On the other side, with GitHub, this option is not available as separate branches of work must be created and later merged.

In addition, a Stanford study found that even a slight perception of collective work can greatly increase performance [58]. This study reported that to collectively engage in a project will decrease fatigue levels and not only increase engagement levels but also success rates [59]. *"The results showed that simply feeling like you're part of a team of people working on a task makes people more motivated as they take on challenges"* the researchers inferred. It can be concluded that a collaborative environment is highly valuable for providing the users with the necessary tools for the best performance possible but also to stimulate user engagement with the platform which will then translate into overall success.

Nowadays all successful real-time collaboration environments are Web-based, simply because it is much easier. It is certainly possible to build a system capable of synchronizing alterations made on a certain file by two clients simultaneously, however, not only will it be dependent on a central entity responsible for managing and propagating the changes but also the performance can be impacted by this model. On the other hand, Web applications inherently have these connections that hugely facilitate this process and, for this reason, online IDEs are directly associated with having a better collaboration experience and software produced [60, 61].

Despite collaboration being facilitated through a Web-based solution and being an extremely important element of a IDE platform, especially considering Global Software Development (GSD) [62] , the appropriate approach is still undefined and is a work in progress at OutSystems, hence, it does not carry the same weight as the points previously described.

2.3 State-of-the-art technologies

2.3.1 WebAssembly trend

WebAssembly [16] is a new type of code that runs in modern Web browsers. More precisely, a Web stack-based virtual machine or a processor that has been designed to easily compile to a lot of real architectures. Moreover, it is able to speedily transform into machine code [63]. It is a low-level language with a compact binary format that runs with near native performance [64].

In recent years this technology has been gaining a substantial amount of traction, not only because it provides languages such as C, C++ [65], and Rust [66] with a compiler target, enabling them to run on the Web but also because it is designed to run alongside JavaScript [67], the main technology of front-end development for decades. By using specific JavaScript Application Programming Interface (API)s, it is possible to load WebAssembly modules into a JavaScript application, hence, harnessing the expressiveness and flexibility of JavaScript while taking advantage of WebAssembly's performance and power, consequently allowing both technologies to complement each other.

When compiling a certain programming language such as C, C++, or Rust to WebAssembly, that code will be processed by the instruction register of that particular virtual machine and stored in a binary format, usually a ".wasm" file. These files can then be ingested by a runtime, which in our context will most likely be a browser that will turn the .wasm file into the actual machine code (figure 2.3). Moreover, WebAssembly was designed to make this process secure.

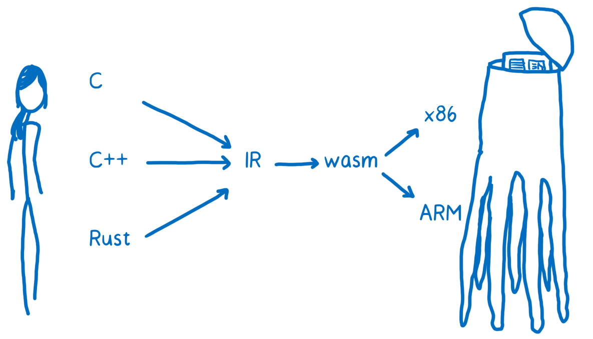


Figure 2.3: Web Assembly [68].

Another technology that has played a fundamental part in WebAssembly's success has been Emscripten [69, 70]. Within this context, Emscripten has the straightforward goal of being a replacement, for example, of the classic C compiler, so instead of compiling to the user's machine code, it will compile into WebAssembly.

As a result of this, old C or C++ libraries with a lot of already implemented functionalities can now be compiled into WebAssembly and be used on the Web browser, something that was never intended to be. Not merely does this new functionality open new doors in terms of the structural composition of a Web application but also enables to re-use an immense amount of code. Therefore, filling a gap in a Web platform that has been filled many times, just not in JavaScript, by adapting already constructed code to the Web, improving the Web platform experience.

At the present date, JavaScript and WebAssembly have approximately the same peak performance. Yet, there is one cardinal difference between both that is the fact that the first language is interpreted and the second is compiled, therefore delivering a more predictable performance, which is, generally speaking, preferred and will inevitably lead to better performance [71].

However, WebAssembly is still not a mature technology and for this reason, it might not be advisable to heavily rely on it, but rather to use it in synergy with JavaScript. Notwithstanding, upgrades are being produced such as multi-thread execution, garbage collection, or better debugging support that will further increase its performance, reliability, and overall value.

2.3.2 Cross-Platform development

Recently, there have been numerous technologies that have been surfacing within the context of cross-platform development [72]. There is not a universal solution as no platform can fully satisfy all types of requirements. Regardless, the following four technologies are the ones that have had the most success in recent times. A brief introduction and contextualization will now be provided for each one.

2.3.2.1 UNO platform

UNO platform [73] is an open-source project that is distinctive for enabling the possibility of building applications for iOS, Android, UWP, as well as integrating them in the browser, taking advantage of the already discussed technology of WebAssembly. Additionally, it also has some support for macOS.

It uses Extensible Application Markup Language (XAML) [74] for declarative UI and, by default, provides lookless controls, meaning that from scratch it holds the responsibility of rendering all the distinct platform controls, instead of relying on the ones specific to the native platforms, providing a parallel experience across supported platforms.

Nonetheless, this is a relatively new technology, which means that has a low level of maturity and could be a solution for some new experimental project but might lack the full support needed for implementing a big and important application.

2.3.2.2 .NET MAUI

MAUI [75], is an evolution of Xamarin.Forms [76] and is a technology for building native mobile and desktop apps using C# and XAML, therefore the main platforms are iOS, Android, and UWP. Currently, there is no browser support.

Contrarily to UNO Platform, it integrates the native look and feel by default, in other words, the layout will be similar across platforms, but the controllers will look like the native ones. Of course, altering the code so that the applications look identical across all platforms is also possible but generally, it is a time-consuming process.

Despite being very anticipated, this platform is still in the development stages hence it is only available in preview.

2.3.2.3 Avalonia UI

Avalonia UI [33] is an open-source technology predominantly dedicated to desktop an UI system, the evolution of WPF and allows developers to build native desktop applications across Windows, macOS, Linux, and in the future iOS.

Similar to UNO Platform, it also takes advantage of the XAML paradigm and produces an identical look and feel across platforms. But on the other hand, and despite being a mature technology, it does not support WebAssembly yet, and consequently browsers.

An advantage of this technology is that it is quite mature, which means there is comprehensive support available, particularly in MacOS, and as consequence, could be a great solution for the implementation of an extensive and reliable application.

2.3.2.4 Blazor

Blazor [77] started as an experimental technology but quickly gained a lot of traction.

Blazor's main distinguishing characteristic is that it is a browser-based technology that uses WebAssembly to bring C# or .NET code to the browser, alternatively to popular browser technologies such as Angular, React or Vue [78] that use TypeScript. It supports all modern browsers on all devices.

This option is particularly popular with “classic” single page application developers since it utilizes HyperText Markup Language (HTML) [79] and Cascading Style Sheets (CSS), instead of XAML. Alternatively, if the goal of a project is to develop a Model-View-Controller (MVC) style application, UNO Platform would be a more adequate solution.

The following table summarizes the previously described technologies:

	Browser	**Phones**	**Desktop**	**Idiom**
UNO Platform	Yes	Yes	Yes	XAML
.NET MAUI	No	Yes	Yes	XAML
Avalonia UI	No	Experimental	Yes	XAML
Blazor	Yes	No	No	Blazor, HTML/CSS

Table 2.1: Cross-platform technology’s characteristics.

2.3.3 Virtual Desktop Infrastructure

The end goal for this thesis, as has been described, is to develop a POC of a web version of the Service Studio, hence, it is important to keep an open mind regarding other possible alternatives of how this could be achieved, and note that the wanted solution does not necessarily imply a process of rebuilding the browser, it can be a question of how to distribute the application to be accessible through a browser. VDIs are an alternative consideration that fit into this description.

A VDI is a technology that recently has been gaining a lot of momentum and consists in creating a virtualized desktop environment that runs within Virtual Machine (VM)s on a remote centralized server setup. To put it simply, this is used to remotely access a virtual desktop [80].

The purpose of such infrastructure is to move the local applications and all the data to a secure dedicated and centralized data center with shared compute and storage capacity. Then, users are given authorized remote access to those applications and data from wherever and with whatever device, since all the heavy computation is done in the server context, the endpoint is not required to be a personal computer, it can equally be accessed through a mobile phone, tablet or thin client terminal [81]. The experience should be seemingly equal but instead of having the application running locally, it will be running remotely.

As observed in the figure 2.4, upon successfully establishing a connection to the VDI environment, a connection broker is responsible for attributing a virtual desktop from a resource pool for each client to connect. If the VDI deployment is persistent, subsequent accesses are redirected to the same virtual desktop, where changes are retained.

Contrarily, if it is non-persistent, the client will simply connect to an available option. A hypervisor is attributed to the vital role of creating, running, and managing various VM that encapsulate the virtual desktop environment.

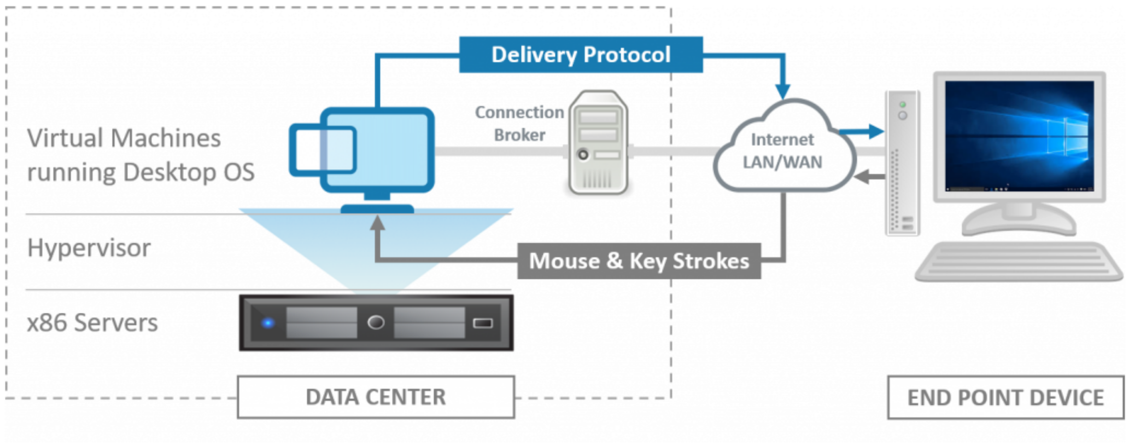


Figure 2.4: VDI Architecture [82].

This solution offers considerable advantages, such as:

- The fact that only necessary resources are allocated, meaning that the management of all resources necessary to complete the necessary computation is made automatically in the server, maximizing efficiency, reducing costs, and lowering hardware requirements. Moreover, seamless scalability is provided.
- It enhances mobility and remote access [83].
- There is no need to have the hassle of configuring an Virtual Private Network (VPN) connection as well as client-side installation or configuration, consequently managing dependencies, which can be a dull process, particularly for inexperienced users.
- Resources are always secured. In a VDI environment, because the storage of information is done in the server, the data is protected if an endpoint device is stolen or compromised.
- Not only does it allow for a homogeneous experience for every client that has access to a particular service, independently of the machine it is being accessed from, but also allows for a consistent experience across different devices.

However, there are also some limitations:

- It is imperative that the clients remain connected to the server to maintain access to the virtualized desktops. Any failure in this connection means that there is no possible way to access the service as no mitigation mechanism can be implemented.

- Performance might lag behind a regular OS and users might feel a delay that damages the fluency of the experience.
- VDIs require several components working together flawlessly to provide users with virtual desktops. If an issue is encountered, it could compromise the connectivity of all clients [84]. Besides, a third-party monitoring tool would likely be required, along with additional Information Technology (IT) staff for its management.

AWS [85], Citrix [86], Microsoft through Azure [87], and VMware [88] are some examples of Desktop as a Service (DaaS) providers.

2.4 Architecture Framework

The goal of architecture design is to identify technological components and management of responsibilities between components that build the system. For this goal, there is no standard universal solution. For this project, the C4 model will be used.

2.4.1 C4 Model

C4 model [89] is inspired by the Unified Modeling Language (UML) [90]. Nevertheless, it originated from the desire to simplify the designing process. This model is an "abstraction first" approach to diagramming software architecture and tries to approximate the process of designing a diagram to the thought process of the user so that it is simpler to produce.

The key logic behind this approach is to, instead of designing an architecture, with all of its complexity, in one single dimension, which most likely leads to an unintelligible drawing, is to decompose the diagram into four distinct and well defined dimensions, supported by straight-forward abstractions, being the first the most general and the last the most specific (figure 2.5).

To fully understand this approach, one first needs to understand the abstractions of this model. Among them is the Person abstraction, representing an actor, Software System, the highest level of abstraction, Containers, which represent a separately deployable thing or runtime environment, that is, an application or a data store such as Server-Side Web application, Mobile application, Database, File system and lastly a Component that encapsulates related functionality.

The four levels of diagrams are the Systems context diagram, Container diagram, Component diagram, and Code diagram, hence the name C4 model. The system Context diagram is responsible for providing the "big picture" of the software, containing the sub-systems that compose it and the user's interactions. Here, the detail is not relevant. The Container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it, some examples within the layer could be a mobile app, single page application, API application, or even a database. The Component

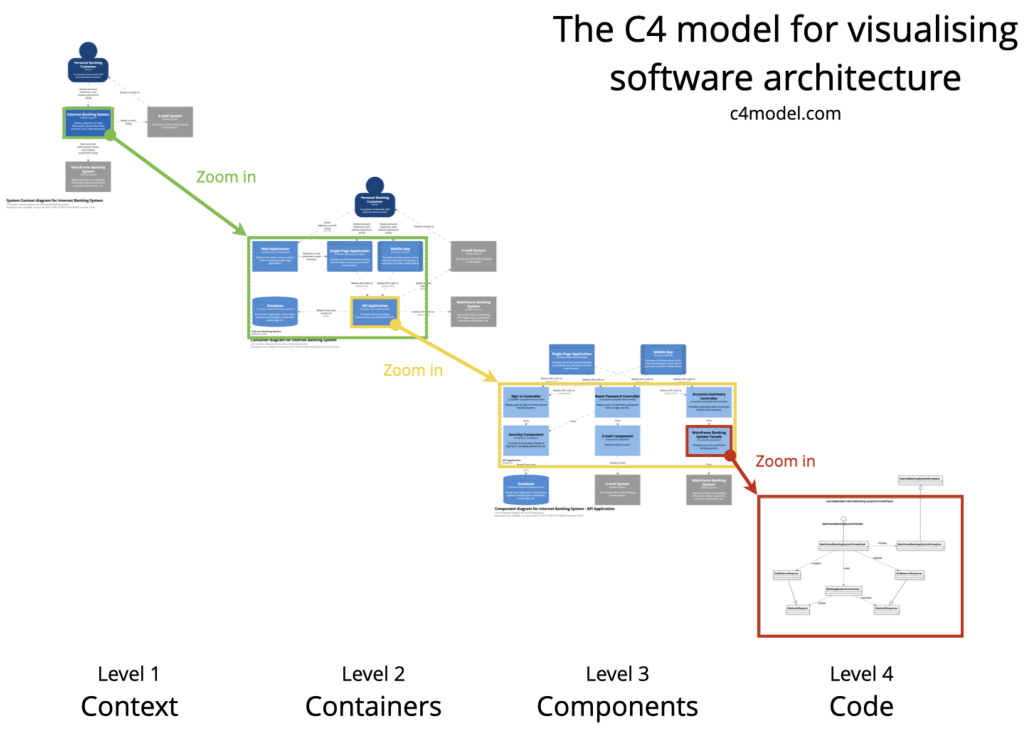


Figure 2.5: C4 model overview [89].

diagram decomposes the diagram further into its building blocks such as controllers, services, repositories, and so forth. Lastly, the Code diagram, responsible for the implementation of specific program logic, is generally not implemented due to the complexity and volatile nature of code development.

In addition to these core concepts, the C4 model also provides other useful features such as a Systems Landscape diagram, which depicts interactions between different systems, Dynamic diagram, that shows how collaboration at run-time between elements, deployment diagram, specifying the physical infrastructure required, notations, among others.

Therefore, the C4 model was chosen as the software architecture tool to be used because of its simplicity, intuitive nature, and very little overhead, especially for a non-proficient user.

3

Related Work

This chapter includes some work related to the technologies previously described to help guide and support future implementation decisions. It is, however, important to note that the related work here presented has the main goal of contextualizing the project developed and served as a source of inspiration, rather than a comparable implementation and, therefore, should not be used to draw parallels and comparisons.

The first example is a use case of how the engineers at AutoDesk [91] managed to successfully adapt a code originally intended for Windows to the Web. This is followed by a description of the system design of Google Drive as well as popular real-time collaboration patterns. After that, Power Platform and PowerApps are presented as an option for building a Web-IDE and also AWS Cloud9, an example of a successful real-time collaboration Web-based IDE. Finally, a brief description of ChromeOS is provided to further emphasize the growing relevance of Web technologies.

3.1 AutoCAD and WebAssembly

AutoCAD [92] is a software commercialized by AutoDesk that is widely used by architects for the construction of 2D and 3D models and is a great use case of a successful transition of a 30-year codebase to a browser experience, facilitated through WebAssembly. This codebase was originally Windows-based, has more than 15 million lines of code regularly growing, and has an ever-shifting nature.

First of all, it is relevant to note that the AutoCAD Engine is reusable across Desktop, Mobile, and Web, and because of that there are limitations regarding the efficiency of the Emscripten tool in translating to all three options.

To translate the necessary code to the Web, the engineers at AutoCAD started by rewriting it from scratch [93], using Flash [94] and later to C++ which would then be cross-compiled to Java and then to JavaScript using the Google Web Toolkit [95]. This process was unsustainable because it was too slow, rewriting the already constructed code felt like “reinventing the wheel” and it is very ineffective to maintain more than one codebase. Besides, cross-compilation can be a daunting task when applied over a large codebase due to high build and start-up times.

For this reason, the group decided to use the Emscripten compiler, which facilitates the process with a one-time compilation that improved build time and performance, and WebAssembly that greatly improved the time it took to compile JavaScript code.

With the adaptation from the Desktop to the Web, some challenges arose, including the adaptation from synchronous to asynchronous communication, lack of shared memory, unsupported concurrent memory accesses, and no support for exceptions.

AutoCAD software is heavily based on synchronous APIs and so, in order to adapt the code to an asynchronous paradigm, rewriting everything simply would not be a feasible solution. To solve this complication, the engineers decided to make use of a Web Worker who would act as an intermediate between the front-end Web user interface, by buffering incoming messages, and the actual AutoCAD back-end software, by intercepting network requests.

However, some disadvantages remained such as the need to manage the lifecycle of the Web Workers, the need to wait for them to initialize before each start-up, therefore impacting performance which is a crucial aspect of any Web-IDE's usability. To mitigate this, the initiation of the WebAssembly engine inside the Web Workers, which amounts to 90% of initiation time, was configured to "instantiateStreaming", supported by Google Chrome [96], that led to a 30% improvement and also, cache support would allow to previously initiate this process further improving performance.

The team also performed some code optimization, by simply reducing the code size, eliminating any unnecessary logic, and readjusting it, allowing for a boost in the start-up time of 1 second, which might not seem much but will enhance user experience.

3.2 System Design analysis of Google Drive

Google Drive [97] is used for uploading files into the cloud that then can be accessed through any device that has access to the internet [98]. This service should synchronize automatically between devices that the user is logged in, and between users with access to the same set of files.

If the intent is to upload a considerable-sized file, a lot of bandwidth will be required, and as a consequence latency will also increase. To mitigate this and to handle the transfer of files efficiently, these platforms will divide each file into smaller pieces, and then, it is only required to modify the correspondent smaller pieces of the data changed, instead of the entire file. Also, in case of communication failure, only the compromised parts of the file need to be re-uploaded or re-downloaded, which will help achieve a better response time.

The Metadata Server (figure 3.1) should include the record of the information relative to each file's parts, including permissions, and will be responsible not only for synchronizing the client's local database with the remote database, but also for the synchronization between different users' shared files. In turn, the Metadata Database will be responsible for storing the information about the users' files. After a client modifies or adds a file to the cloud storage, the Metadata Server will not merely commit the changes to the Metadata Database and reply with a confirmation message, to be returned to the client, but will also send the changes to the clients with access to the same file, with the help of a Notification Server.

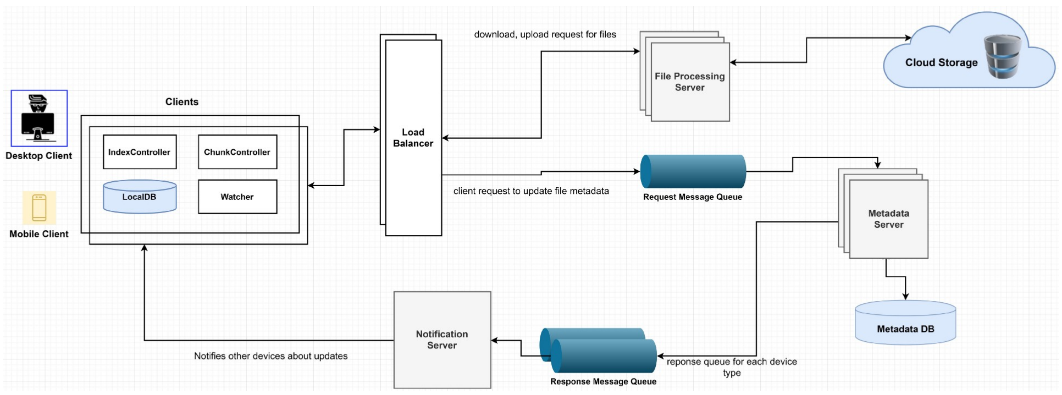


Figure 3.1: System Design of Google Drive [98].

For the service to handle a vast amount of requests from millions of users, an asynchronous communication protocol needs to be implemented. This protocol is based on a queue of messages, between each of the clients' devices and the server, that will provide temporary storage for when the destination program is busy or not reachable. This system does not require an immediate response to continue processing, conferring availability of the service. Two technologies that satisfy the requirements of such messaging queue are Apache Kafka [99] and RabbitMQ [100].

If by any reason the client becomes offline, a component named as the Watcher will observe client-side folders and if any change is executed by the user, it will notify the Index Controller, responsible for updating the data stored locally, about the action performed as it remains cached in the browser. Upon re-establishing the connection, and the metadata server receives an update/upload request, it first checks the consistency of the database, before performing the indicated action.

To ensure scalability for millions of users, each with multiple files divided into chunks, there is a need to partition the Metadata Database, which will also help to divide requests among different servers. A standard way of partitioning the files is to use a hash function that randomly generates a number, which will then correspond to the server that the files are to be stored.

Google Drive has become a critical tool for millions of people [101] and illustrates how advantageous a Web-based, friction-less experience can be.

3.3 Real-time collaboration patterns

Within the scope of real-time collaboration, handling concurrent editing in an environment where more than one user is operating simultaneously on the same document is very challenging. The problem arises due to the inherent latency that occurs between the propagation of changes that will result in distinct conflicting versions of a certain document.

To address this problem, a basic approach is serializing, or in other words, ordering all operations [102]. Yet, the delay caused by this solution is unfeasible for a real-time cooperating system. With that in mind, the most popular methods of implementing collaborative editing, enabling replication and consistency, are OT [103–105] and CRDT [106–108].

3.3.1 Operational Transformation

OT, originally implemented in 1989 [109], has been a widely used solution to address the problem of concurrency control. This pattern was adopted by Google Wave [110], a live collaborative tool that was later restructured into Google Docs, where it is still used, that transforms the operation applied to the divergent documents so that both states remain consistent.

To better understand the relevance of such a pattern, consider the following example:

It is assumed that a process A (figure 3.2) and a process B both start with the state "a,b,c" stored. The processes can be an illustrative example of two users performing changes in the same document.

Then, concurrently, process A performs an operation that deletes the character in position 2 while process B inserts a character "d" in position 0. This results in user A having stored the state "a,c" while user B has "d,a,b,c". Upon propagating both operations to the opposite process, the result will be "d,a,b" in process A and "d,a,c" in process B.

In conclusion, the execution of concurrent actions by distinct clients results in conflicting versions of the same document. For this reason, to resolve these conflicts, OT introduces a transform function that, based on the context is able to readjust the propagated operations so that the states in distinct documents remain the same.

Taking into consideration the previous example, in the case of process B, instead of deleting the character in position 2, OT takes into consideration that it has to add a position for its already added character "d", transforming the operation for the deletion of the character in position 3, resulting in the final state of "d,a,b". On the other hand, in process A, since the insertion of "d" in the position 0 does not alter the coherency of the final state, no transformation is required (figure 3.2) for both instances to be consistent.

It is relevant to note that this mechanism relies on a Client-Server architecture [112]. This is because the server will act as the source of truth, meaning that in the case that any

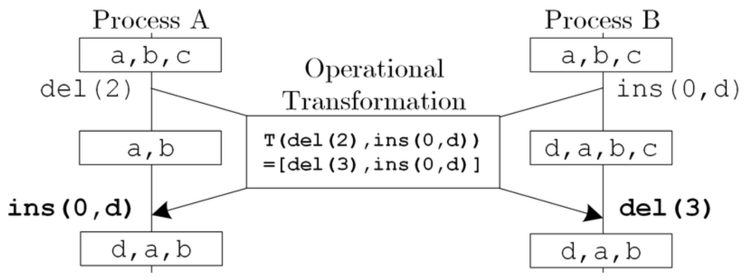


Figure 3.2: OT example [111].

of the clients (this pattern allows for an undetermined number of concurrent clients) fail, the server will be responsible for holding the accurate state definition.

In addition, the server forces the clients to wait for the acknowledgment of the operation that the client sent, allowing clients to remain consistent with the server and allowing for storage of a history of incremental operations rather than a copy of the state of each client. Meaning that there is only one copy of the document, stored on the server. Not only that but by having a single source of truth, with separate steps, it is easier to trace back which operations were executed by each client.

3.3.2 Conflict-free Replicated Data Type

OT can be a solid solution for handling concurrent editing. As proof of this, Google Docs still uses this pattern and it is extremely successful. However, for this process to remain consistent, the central server architecture is fundamental, in other words, all communication has to pass by the server, which can be a severe restriction.

For this reason, it has been thought that it would be interesting to generalize the algorithm to allow for any topology in the system [113]. So that, for example, if a user wants to synchronize both the phone and the laptop, the process would not require information to be sent and fetched from the server, possibly located thousands of kilometers away. The intend would be to re-centralize the pattern while maintaining its consistency, and when appropriate, propagate the changes to the server.

To satisfy these needs, CRDT has recently been gaining more attention, as it is a family of data structures that enable concurrent alterations on distinct nodes that will then automatically merge, despite the order and without requiring any synchronization [107]. Atom text editor [114] is an example of the integration of CRDTs.

CRDT gives each character a unique identifier. This identifier consists of a pair composed of a digit, the result of a simple counter, and a user ID, corresponding to the user that performed the alteration.

When these nodes exchange data, the alterations are based on unique IDs rather than the position of the letter, as in OT. This allows the operations to be commutative, meaning that any nodes that have seen the same set of operations, while maybe in a different order,

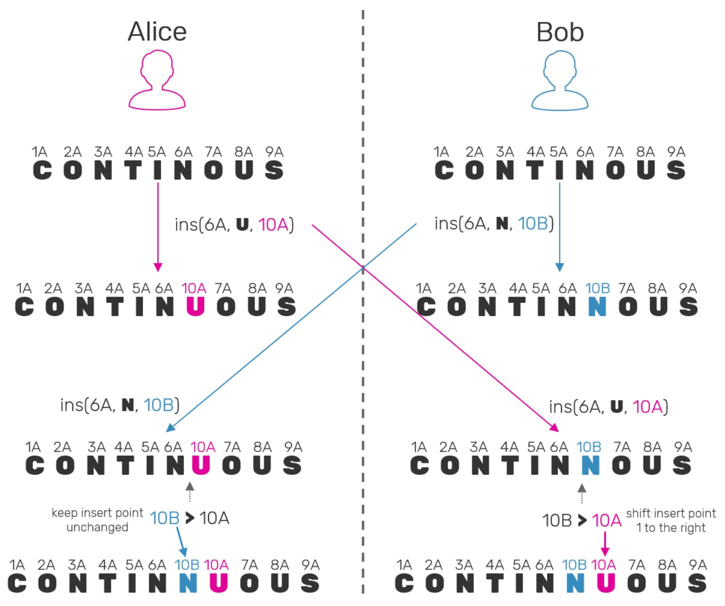

Alice
Bob
1A 2A 3A 4A 5A 6A 7A 8A 9A
CONTINOUS
ins(6A, U, 10A)
ins(6A, N, 10B)
1A 2A 3A 4A 5A 6A 10A 7A 8A 9A
CONTINUOUS
1A 2A 3A 4A 5A 6A 10B 7A 8A 9A
CONTINNOUS
keep insert point unchanged
10B > 10A
shift insert point 1 to the right
1A 2A 3A 4A 5A 6A 10B 10A 7A 8A 9A
CONTINNUOUS

Figure 3.3: CRDT example [115].

will have the same state. In essence, convergence is guaranteed and synchronization becomes trivial.

To better understand this pattern, consider the example of a document with the word “continous”, with two concurrent users performing alterations, pictured in figure 3.3. In case both users decide to insert a letter in the same position, for example after the letter with the ID “6A”, if the ID of the incoming operation is less than the inserting position, it will skip one position to the right and continue this shift process until it successfully inserts the characters.

This way, the pattern allows for consistency and a deterministic result, and even though Alice inserted the letter “u” at the same place Bob inserted the letter “n”, both users end up with equal versions of the document (“continnuous”) without needing any context interpretation.

3.4 Power Platform and PowerApps

The main goal of Microsoft’s Power Platform [116] is to enable digital transformation across organizations by empowering employees, engaging customers, optimizing operations, and transforming products by providing features such as anomaly detection, proactive maintenance, and simplified application building mechanisms. This solution not only enables the integration of an application in a Web context, but also facilitates and improves its management.

Power Platform can essentially be divided into three major parts: Power BI, a tool that provides business analytics; Flow, which facilitates the automation of the workflow; and finally, the one that is most relevant in the context of developing a Web-IDE, PowerApps, that makes it possible to build applications that users interact with on the platform. These components can then be integrated with a common data service.

PowerApps [117] is an example of a platform that enables the development of an online IDE by offering a set of tools that allow clients to build efficient custom applications [118]. By taking advantage of features already found in the Office 365 suite and Microsoft Platform, it is possible to integrate an already implemented process that includes, for example, Excel [119] or SharePoint [120]. Through PowerApps it is possible to build a standalone application that can be used on the Web or mobile, giving users access to the applications wherever they want, eliminating the need of installing software. This particular solution is directed towards an increase in business productivity [121], it is not a software development facilitator.

One of the goals of PowerApps is to bring the process of building an application closer to non-developers through intuitive drag-and-drop interfaces and by making available features and tools that do not require any programming. Amongst them are a collection of already functional applications, a large library of connectors that allow the integration of data and systems, and an active support community. Nonetheless, having background knowledge in code development is essential for the implementation of any relatively complex functionality.

There are two approaches to building an application using PowerApps. The first is referred to as “Canvas” where the structure is built from scratch, giving the user complete control over connections and implementation details. On the other hand, the model-driven approach enables the creation of an application on top of already developed examples, such as forms, data structures, and even business rules, further facilitating the creative process.

This is a platform that was built to develop simple yet efficient Web applications from scratch, with the focus on integrating parts of the Microsoft universe to increase automation, collaboration, and productivity. Examples include a rating system with input from a variety of users, a system to aid an IT department with tracking assets such as hardware, or even managing donation from a fundraise (figure 3.4).

PowerApps provides not only a Web platform for the application to be deployed but also the requirements for its management, maintenance, and improvement and therefore, this option deserves consideration. However, given that OutSystems already possesses fully operational software it might not be the most appropriate solution to re-integrate and re-connect all of its components inside a new platform, especially if there is no intention of taking advantage of Microsoft’s universe of services.

Also, Microsoft is a direct competitor to OutSystems in the field of Low-Code Development Platforms, hence, to have a system dependent on the Power Platform ecosystem might not be the most desirable course of action.

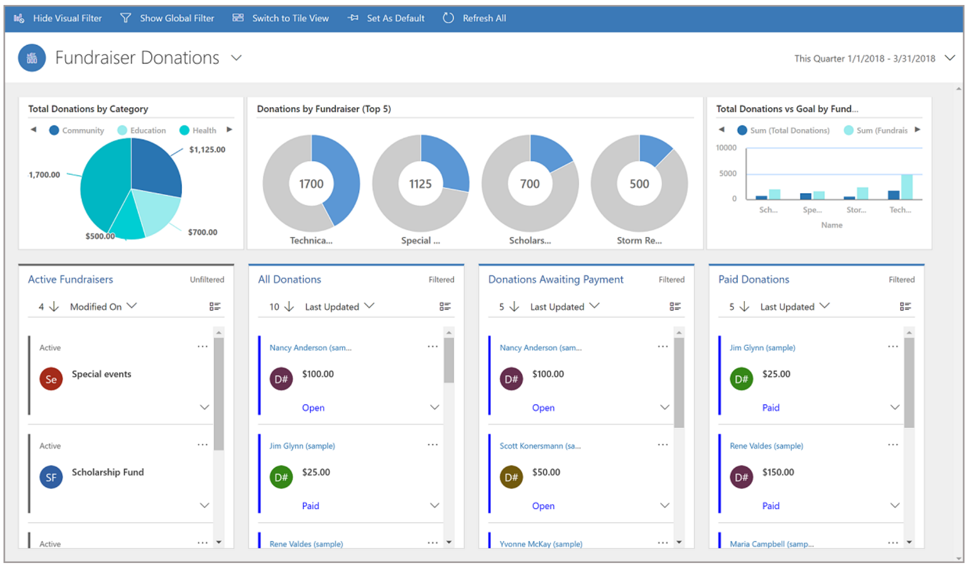


Figure 3.4: PowerApp example, Fundraiser Donations [122].

3.5 AWS Cloud9

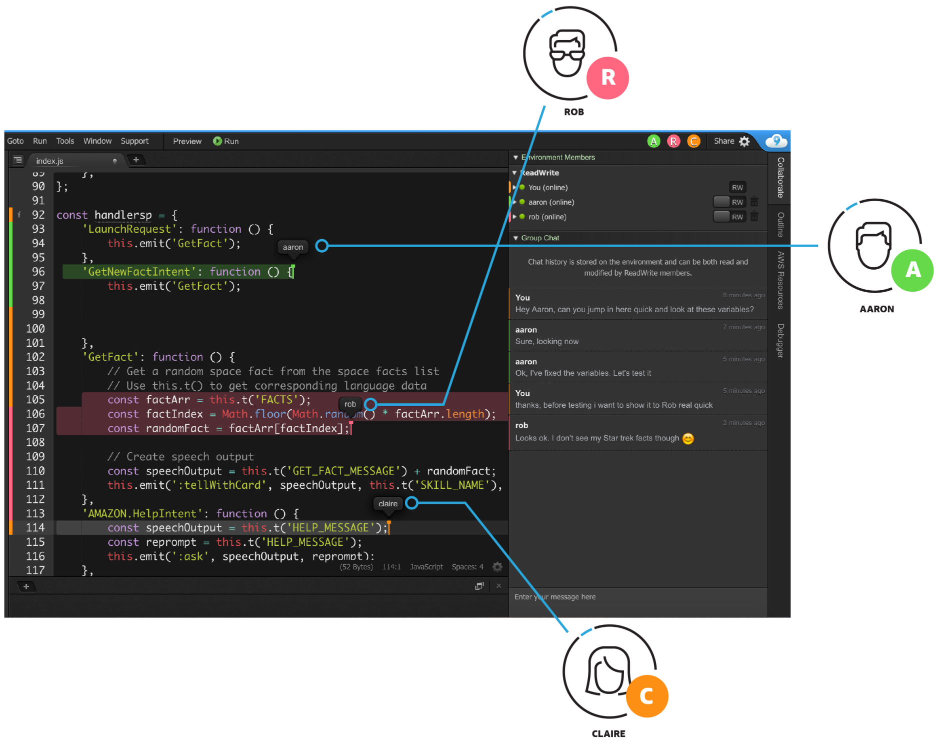
AWS Lambda [123] is an event-driven computational platform conceived to let users run code without having to provision or manage servers [124]. It's a serverless technology that allows clients to focus on implementing the business logic without worrying about the infrastructure to support it.

One of the major factors behind the widespread adoption of AWS Lambda has been the potential to reduce costs [125, 126], this is because, in this context, the user will only pay for the resources temporarily used, instead of having to permanently purchase the required equipment to perform the operations required.

However, one downside of this approach is the difficulty in debugging implemented functions. This aspect, combined with the fact that AWS already has a large group of cloud services and infrastructure, led Amazon to acquire and integrate the Cloud9 IDE [127] into its ecosystem. This made sense as a particular IDE would not only allow for an easier debugging of Lambda functions but would also complete the building blocks of the modern software development lifecycle and reinforce the consistency of the whole AWS ecosystem.

Cloud9 was founded in 2010 and was the first "Development as Service" platform that delivered an IDE in the cloud [128], which supports mainstream languages such as Python [129] and JavaScript [67]. By being an online IDE, it eliminates the need for downloading software, configuring environments, or setting up the IDE, which ensures that an enormous amount of time is saved, especially in a context where more than one project is being developed simultaneously by a large group of coders, as well as when developers in different teams utilize similar development environments.

AWS Cloud9 has strong and cost-effective integration with the AWS Services [130] by providing a command-line interface with direct access to the user's resources. It also allows for pair-programming (figure 3.5), or in other words, real-time collaboration and communication with other colleague developers. In addition to these advantages, the IDE also provides access to a AWS Resource Explorer and standard features such as Git integration and Docker support.



```
    },
};

const handlersp = {
    'LaunchRequest': function () {
        this.emit('GetFact');
    },
    'GetNewFactIntent': function () {
        this.emit('GetFact');



    },
    'GetFact': function () {
        // Get a random space fact from the space facts list
        // Use this.t() to get corresponding language data
        const factArr = this.t('FACTS');
        const factIndex = Math.floor(Math.random() * factArr.length);
        const randomFact = factArr[factIndex];

        // Create speech output
        const speechOutput = this.t('GET_FACT_MESSAGE') + randomFact;
        this.emit(':tellWithCard', speechOutput, this.t('SKILL_NAME'),
    },
    'AMAZON.HelpIntent': function () {
        const speechOutput = this.t('HELP_MESSAGE');
        const reprompt = this.t('HELP_MESSAGE');
        this.emit(':ask', speechOutput, reprompt):
    },
```

(52 Bytes) 114:1 JavaScript Spaces: 4

Environment Members
ReadWrite
You (online) RW
aaron (online) RW
rob (online) RW

Group Chat

Chat history is stored on the environment and can be both read and modified by ReadWrite members.

You 8 minutes ago
Hey Aaron, can you jump in here quick and look at these variables?

aaron 7 minutes ago
Sure, looking now

aaron 5 minutes ago
Ok, I've fixed the variables. Let's test it

You 5 minutes ago
thanks, before testing i want to show it to Rob real quick

rob 2 minutes ago
Looks ok. I don't see my Star trek facts though 😊

Enter your message here

Figure 3.5: AWS Cloud9 IDE, pair-programming feature [127].

AWS Cloud9 also provides organizations the added benefit of not needing to acquire a private server system, as they can store their development environment in the AWS cloud at a lower cost. Hence, it is true that this IDE can be very advantageous, as previously mentioned, but it is also a bid by Amazon for attracting developers to leverage its cloud infrastructure and integrate their products into the AWS ecosystem [131].

IDEs play a crucial role in driving developers' mindshare and adoption. An example of this is the fact that the success of the .NET platform can be greatly attributed to the popularity of Visual Studio Code [132] among developers. Amazon, by acquiring a Web-based IDE as service and integrating it with the already existing services is demonstrating its commitment and confidence in how cloud computing and Web technologies will play a leading role in the near future, including in the development environments, and, therefore, is trying to anticipate Microsoft, as it is yet to launch a browser-based version of Visual Studio Code that integrates with Azure [133], and confer future business success and longevity.

3.6 Chrome OS

Traditional OSs require a lot of hard drive space, locally installed programs, and manual management of updates and drives, whereas nowadays, the average Personal Computer (PC) user can execute most, if not all required tasks simply with access to the browser [134]. For example, checking the e-mail, working on some documents provided by the Office 365 ecosystem [135], listening to music, watching movies or checking social media are some of the various activities one can execute through the browser.

Chrome OS's intent is to take advantage of this paradigm shift and is a great example of how cloud computing models are gaining considerable momentum, by providing an OS that is completely based on the browser experience. Google Chrome OS was introduced by Google in 2009 and is an open-source cloud-based OS created on top of a Linux kernel, integrated with Chrome [136].

The product is commercialized integrated with a notebook with personalized hardware that enables the OS to run optimally. According to the company, the goal of this architectural approach is to present an overall better OS experience by eliminating the need of depending on local persistent storage and focusing on building a much simpler OS with improved performance and security [137].

Because Chrome OS supports only Web capabilities, it is not required to perform unnecessary systems checks and loading procedures that take a big toll on traditional OSs, boosting performance [138].

The notebook's interface is very similar to the browser and most of the common utilities of a PC are still available, such as a file system, Bluetooth, and peripherals. However, it's not possible to either install or uninstall any program, consequently, the user will be restrained to basic applications provided through the Web or the Google Web Store.

Chrome OS performs automatic background updates, sparing the user of unpleasing recurring notifications and one of the best advantages of the Google Chrome ecosystem is that no known viruses or malware can penetrate the system because it's a highly protected enclosed system by the user's personal Google account, making this operating system one of the most secure on the market.

This context might be more than enough for a vast majority of regular users, such-like students, or even in an organizational context where there aren't heavy technology requirements, since currently there is a wide range of Web applications able to provide basic services. On the other hand, it does not satisfy the needs of proficient users such as software developers, that require more demanding software to operate like IDEs, or event media artists that require, for example, heavy programs such as Photoshop, which are only available in desktop format.

In short, the goal of this Web-based concept is to enable an easier and more pleasant experience, with less friction, less frustration, more performance and endorses the leading role that Web technologies will have in the future.

4 Requirements

This chapter begins by highlighting what are the biggest difficulties in implementing the project in question.

After that, a discussion about what ought to be the requirements of the POC is presented, accompanied by an illustrative diagram, in order to identify what are the key points that enable to arrive at a conclusion in regards to the feasibility of this project.

Finally, a proposal of the structure of the POC is detailed. This is so that, despite possible future adjustments, a direction is clearly defined for the development of the practical work of the thesis.

4.1 Problems / Difficulties

Nowadays it is possible to observe a migration movement of technologies to the Web because, as described before, there are many advantages to this paradigm and this is a trend that is only in its early stages and will continue to grow. For this reason, it would be logical to assume that IDEs would also follow this path and become Web-based as well. However, this isn't the case as the most popular IDEs are still lagging behind in this pull towards the Web [139]. It is important to understand that complex IDEs such as Eclipse [140] and Visual Studio Code [132] incorporate a wide range of services and engineering tools and that to fundamentally change the architecture of this software to become Web-based is not a simple task [141].

In the case of a service that is based and runs on a server, and therefore is presented through the browser, as opposed to locally installed, there is the inherited disadvantage of much slower communication, the inevitable transformation process of adapting event-based functionalities to an HTTP request/response policy [142] and the added frustration of a limited number of client development languages.

It can be argued that this approach boosts the overall security of the files as these are remotely stored and saved, even if there is a problem of corruption or failure of any physical device. However, different precautions must be taken into account such as making sure that clients can only access their files, that it is not possible to perform changes over the server, and even supervising the resources used so that no instance over-consumes and damages other client's availability. Furthermore, server security and access restriction management are of extreme importance as a failure will compromise not a single client, but all of the clients' experience.

In addition, the cost of maintaining all of the client's files in the cloud can be considerable, especially in the case of a large user base such as OutSystems', and there is a dependence of internet connection [143]. This might not be fully incapacitating, but it still is a hugely relevant disadvantage that should be addressed to provide the users with the best possible experience.

4.2 Features

The vast majority of the requirements for the project to be developed were suggested, through several non-structured meetings, by the OutSystems adviser of the thesis, who is also the main stakeholder regarding this project, whose role inside the company is the Head of Product Architecture, hence, has a better understanding of what ought to be critical characteristics of the POC. Additionally, a brief meeting with the person responsible for the Product Design also occurred, to explain the main characteristics of the POC to be developed and, to receive some feedback.

Nonetheless, all requirements were further discussed and matured through a discussion and experimentation process involving both advisers and the student.

As is observable in figure 4.1, the first major requirement of the POC is to implement a resilient and appropriate visual model that represents the one provided by the official OutSystems' Service Studio IDE. To achieve this, it is fundamental to include an intuitive UI, local persistence and, additionally, a functionality that enables to upload custom files, to assist with both the development process as well as to facilitate testing.

Next, it is important to grant model manipulation, in other words, to incorporate a validation mechanism, either performed in the Server-side of the architecture or in the browser itself. Finally, include the auxiliary functionality of the browser cache to store the visual model the user is constructing.

The third and most complex set of requirements are relative to the Central System, or Server-side part of the architecture. This includes both the storage and retrieval of data with a fast enough response time. Moreover, it is also required to have a detailed discussion and description of the system architecture, as well as its implementation. This system must also be scalable and provide local data residency (data is only kept in the region the user is located at).

Furthermore, it is also required to include a real-time collaboration mechanism in the Web-IDE. To accomplish this, it is necessary to implement a subscription and broadcast system that enables various users to perform changes over the same visual model, at the same time, while also doing it with low latency, so that a pleasurable experience is provided.

Finally, the last requirement, although considered less relevant than the previous ones, is the inclusion of the possibility of off-line development.

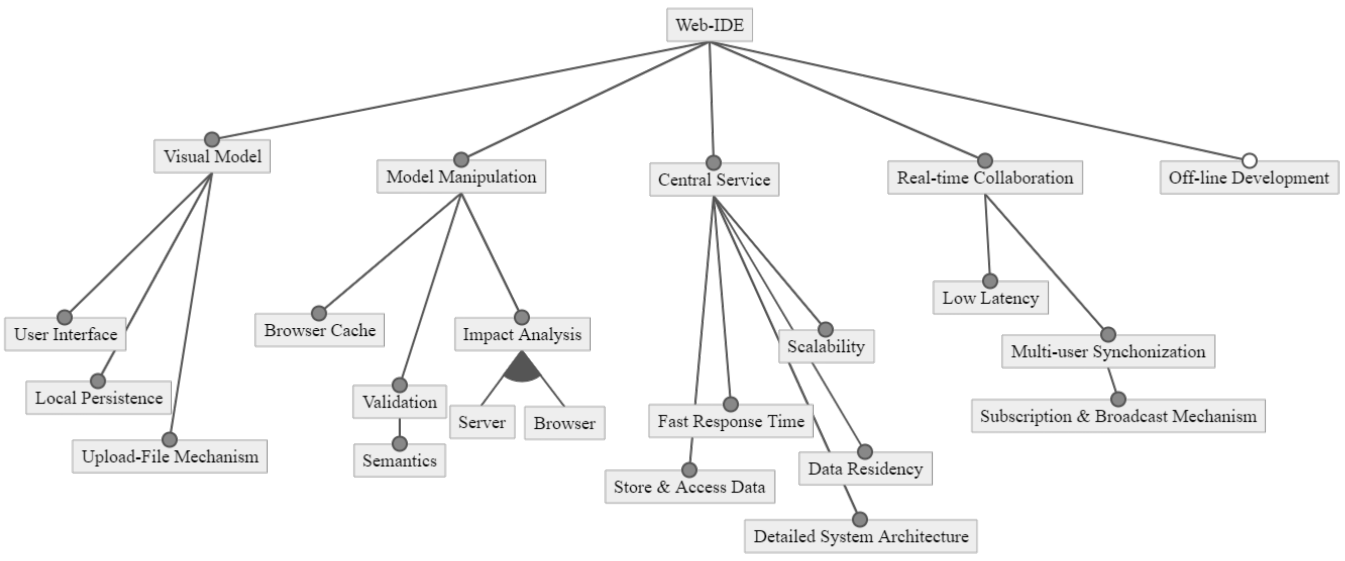


Figure 4.1: Feature Diagram.

4.3 Proof of concept

As has been mentioned before (section 1.4), a POC, or prototype, of the Web-IDE was developed during the thesis, to better understand the overall context of the project proposed. The goal of this prototype was not to produce a fully functional version of the Service Studio running on the Web, as this task would be unfeasible for the time available, but rather to emulate an example that is accurate enough to answer the most important questions in regards to the limitations of such architecture. For instance, how to implement the local model inside the browser and if it is possible to confer an experience that is fluid enough for real-time collaboration.

It was decided that the POC should have an incremental construction, to better define the structure of the project, conferring modularity, so that potential improvements are easier to apply later on. The POC to be developed could be divided into the following four incremental stages, however, it is important to mention that alterations could occur as these acted as a preliminary guideline.

- **Mockup**: The Mockup should focus on providing a subset of functionalities present in the Service Studio, for demonstration purposes.

  To this effect, an approach that could prove efficient would be to, rather than replicating all the different sections and tabs that are provided, concentrate on the central canvas grid interface (figure 2.2) that enables to move and connect the language components, therefore building simple graphs.

This part should not be a core focus of the project. Nonetheless, it is important not to neglect it as a visually attractive interface is inherently valuable.

- **Model manipulation**: The section should focus on giving some semantic to the graphs that are possible to build. The language used here should, at least at the beginning of the implementation, be an exemplification of the OutSystems language, to simplify the process.

  In other words, to guarantee that the implemented programs have coherence and that is validated locally, for example, an initial node can only have outwards dependencies and, analogously, a final node can only have inward dependencies.

- **Central Service**: A central data service must be constructed, with a programming appropriate language, to allow for the persistence of data, that is to say, to allow users to fetch their data every time they access the IDE Web page.

  To have a server responsible for storing all the data, it wouldn't be interesting to implement it locally as it wouldn't be useful to understand if the response time is fast enough to provide a good user experience. For this reason, it would be appropriate to store the data service in a Web Service supplier such as Azure to better replicate the structure of a Web-IDE.

- **Real-time collaboration**: Finally, a mechanism of real-time collaboration must be constructed to allow more than one user to interact with the same project at the same time.

  To enable this feature, the final structure should include a subscription and broadcast system that notifies and updates every client with access to a certain project that alterations were performed, therefore synchronizing all the information between them.

The work will be considered successful if it can satisfy the objectives proposed (section 4.3), if it can answer the uncertainties regarding the theme proposed, mainly, if it is feasible to build a Web-IDE version of the Service Studio. Moreover, if it is well constructed, consistent, and, above all, if it is able to provide future insight and value to engineers at OutSystems.

5 Implementation

5.1 Visual Model

In this section, a justification of why the selected visual library was chosen is presented as well as a contextualization of the technologies and concepts that are used within it.

Afterwards, all the implementation details relevant to the understanding of the visual model are discussed. Firstly, a brief explanation is presented of how the stored data is structured, in order to be easier to understand the visual model "under the hood". Secondly, the visual layout of the POC developed is presented, accompanied by a description of the available functionalities.

5.1.1 D3.js library

The first step in the implementation process was to create a visual representation of the logic models that are possible to create with the Service Studio.

It is important to note that the goal of this prototype representation is merely to act as visual and structural support for the POC, so that the main questions regarding the theme of this thesis can be satisfied, rather than an accurate copy of the IDE, as this would add additional complexity to the solution without contributing to its value and the scope of the project.

To produce a visual representation of the logic present in the Service Studio IDE, the D3.js [144] library was selected.

D3.js is a JavaScript library used for manipulating documents according to data and makes use of Scalable Vector Graphics (SVG), HTML, and CSS standards [145]. D3 library uses pre-built functions that allow to select HTML elements, create SVG objects and style them using CSS syntax. In addition, it is also possible to connect data to a Document Object Model (DOM), perform efficient transformations on it, add animated transitions and even dynamic effects for visually appealing representations. Due to the functional style of this library, there is available a large collection of community developed modules [146] that are easy to re-utilize. This proved to be particularly beneficial in this project, as it allowed for rapid construction of a simple yet consistent visual model.

An easy to understand example of the utility of this library is, for example, to generate an HTML table from a data set of numbers.

This library is generally used for data visualisation applications, such as animated treemaps, temporal force-directed graphs, histograms, scatter-plots, among others. However, it provides the necessary tools for the development of a node-based directional graph, which in this case is satisfactory to represent the implementation language present in the Service Studio Web-IDE.

5.1.2 Implementation details of the visual model

5.1.2.1 Local data storage

All the data relevant for the model is stored inside an object called "graph". This object contains:

id	A constant that is unique to the graph, used to identify it. This is a Universally unique identifier (UUI) that is randomly generated upon creation.
nodes []	An array that will contain all the metadata of the nodes present in the model.
links []	An array containing all the information of the links inside the graph.
data []	An array that contains all the data correspondent to the nodes inside the model.

Table 5.1: "Graph" object structure.

In the Service Studio, the data associated with each node can be altered accordingly to the operations performed. However, for the sake of this POC, the only alterations that are to be done to the nodes are of visual nature, such as insert, delete and move. Nonetheless, it is of interest to store a certain amount of data corresponding to each of the nodes, to better replicate the scenario in the Service Studio IDE.

Therefore, it makes sense to separate the information of a certain node into two components. The first one that contains its "metadata", trivial fields that are easy to assign and update, and a second one that contains "dummy" information, that is, the data associated to a certain node. But in this case the data will remain unchanged, contrarily to the Service Studio, and will only exist for demonstration and testing purposes.

This decision was fulled by the fact that, in the case of the Service Studio, each node can contain up to dozens megabytes of information, therefore, it would be very expensive to update even a simple "metadata" field, because it would require to update the whole node, leading to a nonoptimal solution, given that a fast and uncomplicated POC is desirable.

As an example, if the user decides to slightly change the coordinates of a node in the model, it will only be necessary to update two metadata fields inside the nodes array, the "X" and "Y" variables, enabling a fast and direct solution. Moreover, this design detail makes it so that only the required information, that is, the information containing the details of the nodes and links, excluding all the data of the nodes, is used for the rendering of the model, allowing for improved performance.

Each entry in the "nodes" array, that in this case represents the "metadata", will contain the following fields:

id	UUI of the node.
index	Position inside the array.
px	Horizontal coordinate in a two dimensional plan.
py	Vertical coordinate in a two dimensional plan.
fixed	This field is required for the rendering of the visualization model. D3.js enables nodes to move and have gravity depending on the use case and preference of the user. In this case, as in the Service Studio, the nodes are static and only move when the user drags it. For this reason, this variable is always set to true.
type	Category of the node. There are four different types of categories (Start, End, Assign and Function) and each of them will have its particular characteristics and use case, which will result in distinct verification scenarios, as explain ahead in the model manipulation section (5.2).

Table 5.2: "Node" object structure.

Each entry in the "links" array will contain the following fields:

id	UUI of the link.
source	Node object that corresponds to the source of the directional link.
target	Node object that corresponds to the target of the directional link.

Table 5.3: "Link" object structure.

It is also important to note that the links will change their length and direction, according to the source and target node.

Each entry in the "data" array will contain the following fields:

id	UUI of the node.
dummy	A field containing all the data corresponding to a node, in JavaScript Object Notation (JSON) format.

Table 5.4: "Data" object structure.

5.1.2.2 Visual aspect of POC and Toolbar functionalities

On the top part of the visual model it is possible to observe the following functionalities (5.1):

- **"Arrow" button (1 in the figure)**. When selected, allows the user to relocate any of the nodes present in the model.

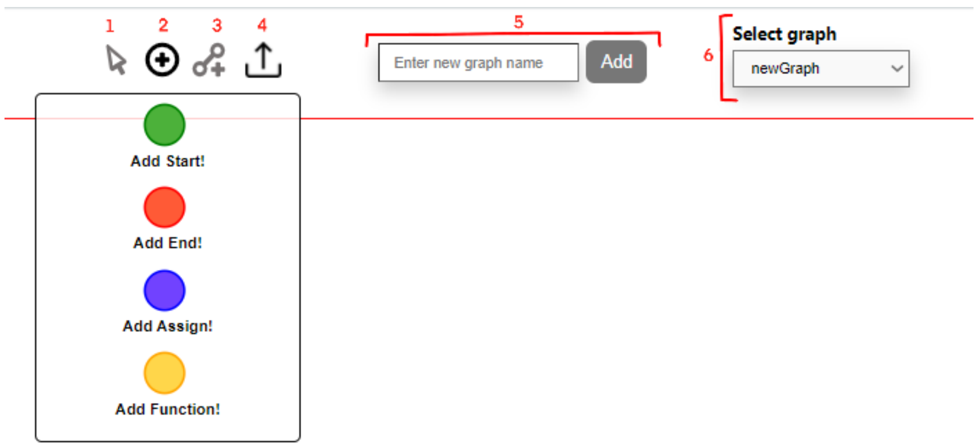


Figure 5.1: Visual model functionalities.

- **"Plus" button (2 in the figure).** When selected, will display a small library containing all four options of nodes that are possible to be introduced in the model. This library will be hidden once any node option is selected, and consequently, a new node is inserted at the centre of the model, or when the user clicks outside the open library. Once a new node is introduced to the visual model, it is possible to drag the nodes, in other words, once a node is added, the same behaviour as the "arrow" button will be available.
- **"Connection" button (3 in the figure).** Allows the user to connect any two nodes present in the model. In order to succeed, the user must first click on the node that should be the source of the directed link, and then drag the visual representation of an arrow and "drop" it on top of the target node. If the user fails to connect the link to a target node, the link is deleted. It is also not possible to create a link that has the same node as source and target.
- **"Upload" button (4 in the figure).** Allows the user to load a graph model from the local storage of his personal machine. After selecting the button, a pop-up will be displayed where the user will be able to search through his file system and select the JSON file containing the data of the model. Upon selection, and if the JSON file is valid, in other words, if it corresponds to the required input template, the data will then be uploaded into the visual model.
- **"Add graph" input (5 in the figure).** Generates an empty new graph model, with the name determined by an input label, that the user will now have access to. The empty model will automatically be displayed in the central canvas and the default name in the "Select graph" drop-down menu will be the name of the graph newly created.
- **"Select graph" drop-down menu (6 in the figure).** Displays all the names of the graphs that the user has access to. If the user has access to more than one graph

and wishes to change the current visual model, it will only be required to select the wanted graph. Once the graph is selected, the model will reload with the wanted graph.

Below the option buttons previously described two rectangular displays are presented. The first display, identifiable by a red border, is the central canvas where the visual model will be loaded and where any change to this model can be performed.

Underneath the main canvas, a display with a black border is presented, to better identify it, this display will also be addressed as "error box" . This display will present all the verification errors in the model that is currently loaded onto the canvas.

At the top of the page, a circle with a black border is observable. The colour of the circle will determine the verity of the model that is currently on the canvas.

The detailed logic of these errors, as well as the verity of the model, will be further detailed in the section on model manipulation. In the case that the model is verified, the displayed colour will be green. Conversely, if the model is not verified, the colour will be red.

Finally, the background image is a screenshot of the Service Studio IDE, so that a "real" feel is conferred to this POC.

5.2 Model Manipulation

With the visual model completed, this section focuses on defining how the visual model, and consequently the data within, are to be stored as well as to understanding what is the best way to confer it adequate semantic.

An examination is made on what is the best way to implement impact analysis within the scope of this project. After that, several browser cache options are considered to better understand, based on the application requirements, what ought to be the best alternative to integrate. Finally, there is a verification topic where the semantic rules applied to the visual model are detailed.

5.2.1 Impact Analysis

There is a considerable level of uncertainty regarding the performance, the fluidity of operations, and how to implement a functionality like an impact analysis within the context of a Web-IDE. The impact analysis will correspond to the identification of potential consequences that a change performed over the model might create, more specifically, to a process of conferring if the visual model is still verified.

It is important to note that in the standard OutSystems IDE, all necessary information for performing the operations available is loaded into memory, hence, when a user, for example, decides to delete a component and this operation needs to be verified, consequently, there is a process that needs to occur in order to make sure that the consistency of the file is maintained.

On the other hand, considering the Web-IDE solution, it might be a problem to maintain all information necessary in memory, and as consequence, these verification processes might create enough latency to compromise the fluency of the user's interaction, therefore damaging the user experience.

The second challenge is offline development. There are two available options, either the offline development is eliminated altogether, or an approach similar to Google Docs could be adopted where, in case of no internet connectivity, changes are kept in the browser and when connectivity is re-established these changes are then propagated.

The fact that Google Docs has such features means that it is possible to save, with good performance, complex information locally and that all logic that enables the user to modify and save a file is kept locally. For example, in the context of Google Slides, if a user presses the button "new slide", there is no need to fetch any information to a central system, all the logic is maintained locally and is independent. Nonetheless, changes are systematically being sent and saved, so that the latest version is always available for every user that has access to a certain document.

There is, however, one problem that is raised in the context of OutSystems that is the fact that if changes were to be stored and were to only be applied when the internet connection is resumed and if these newly added changes require verification, changes done in an offline context could be compromised, hence, lowering the quality of the user experience.

The following subsection will present a discussion concerning system alternatives that were considered.

5.2.1.1 Central System alternative

In an architecture where the central system is responsible for the impact analysis, the browser periodically dispatches portions of information, that correspond to changes performed over the model, so that it can be stored and consequently be gathered in order to be viewed by other IDEs. Here, the central system is responsible for making the impact evaluation that every action has, for example, checking if a certain component can be eliminated.

An obvious advantage of this approach is that the application is lighter and since there is no need to implement model logic, the information saved in the browser is simpler, therefore allowing faster distribution of information. Furthermore, there is also the added advantage that there is only the need to propagate changes made to the model.

On the other hand, not only could this option lead to added latency, especially if a verification process is required, but there is also the question of how to maintain coherence between the local model and remote model. If for any reason the information pipeline is compromised, the development cannot occur as the browser does not know to manipulate the model. This is a substantial disadvantage that leads to the conclusion that this architectural model would not be the best solution.

5.2.1.2 Browser solution

The solution adopted is based on a model where the browser is the one responsible for the impact analysis. That is, all logic required to apply any change to a component is stored locally in the browser, and not in the browser cache. Thus, the Web application knows how to implement every alteration, is responsible for managing the model and sends information to the Central System.

On one hand, it is important to note that not only is it necessary to implement the supported structure and model logic directly within the browser, which should not be trivial but also that this task will be limited to the languages available.

However, the self-evident convenience of this option would be that a low latency would be available as all necessary code for alterations is already locally saved. Besides, this model can mitigate temporary connection failure. In this case, as alterations are stored in cache inside the browser, whenever the connection is reestablished with the central systems, alterations can promptly be dispatched and therefore providing users continuous access to the platform.

For the reasons mentioned, we decided that this would be the best course of action.

5.2.2 Verification

Once the best option for impact analysis was established, the following step would be to implement it. In other words, add a verification mechanism to the visual and data model so that when a change is performed over the model, a routine or algorithm runs to guarantee its soundness.

To confer some semantic to the visual model previously described, as in the Service Studio, specific characteristics were attributed to each type of node. In this case, different types of nodes differ in the number of links that is acceptable for them to be connected to, more specifically, inward links and outwards links. As such, it was necessary to implement a function that returns the number of links that each node is connected to (“countSource”) and has connected to it (“countTarget”) (Algorithm-1) .

Algorithm 1: Calculate number of inward and outward links

```
Result: [countSource, countTarget]
var countSource = 0 ;
var countTarget = 0 ;
for (var j = 0; j < graph.links.length; j++) do
    if (graph.links[j].source.id == idNode) then
        countSource++;
    end
    if (graph.links[j].target.id == idNode) then
        countTarget++;
    end
end
```

With the previous algorithm implemented, it is now possible to use it to build a second, more elaborate, algorithm that is capable to determine the verity of a model (Algorithm-2), based on the following premises (rules):

- There can only be one link “leaving” a Start node (directed link outwards).
- There can be no links “entering” a Start node (directed link inwards).
- There can only be one Start node in the model.
- There can no link “leaving” a End node (directed link outwards).
- There can only be one End node in the model.
- There can only be one link “leaving” a Assign node (directed link outwards).
- There can only be one link “leaving” a Function node (directed link outwards).

5.2.3 Browser Cache

Once the impact analysis and, consequently, the verification process is implemented, the next step, within the model manipulation context, is to decide, among the different options available, which would be a more appropriate browser cache tool to be integrated into the project.

The browser cache is an auxiliary tool to store a verified, and up-to-date, copy of the data of the visual model that the user is currently working on. This component is of particular interest because it enables the users to develop their projects, verify it, save it and, according to the implementation and scope of the project, later quickly gain access to it rather than having to fetch the full data of the visual model from the server.

The considered options will now be presented.

Algorithm 2: Check if model is verified

```
Result: verification
var verification = true ;
var startCount, endCount = 0 ;
for (var j = 0; j < graph.nodes.length; j++) do
    var node = graph.nodes[j] ;
    var values = Algorithm-1(id.node);
    switch (node.type) do
        case 'Start' do
            if (values[0] > 1) then
                verification = false ;
                errorBox.append("Start can only have one outward link ! ") ;
            else if (values[1] != 0) then
                verification = false ;
                errorBox.append("Start can't have inward links ! ") ;
            startCount++;
        end
        case 'End' do
            if (values[0] != 0) then
                verification = false ;
                errorBox.append("End can't have outward links ! ") ;
            end
            endCount++;
        end
        case 'Assign' do
            if (values[0] > 1) then
                verification = false ;
                errorBox.append("Assign can only have one outward link ! ") ;
            end
        end
        case 'Function' do
            if (values[0] > 1) then
                verification = false ;
                errorBox.append("Function can only have one outward link ! ") ;
            end
        end
    end
    if (startCount > 1) then
        verification = false ;
        errorBox.append("Can only exist one Start ! ") ;
    else if (endCount > 1) then
        verification = false ;
        errorBox.append("Can only exist one End ! ") ;
    if (verification) then
        verificationCircle.fill = "green";
    else
    verificationCircle.fill = "red";
end
```

5.2.3.1 Local Storage

The first option considered was Local Storage, due to its high popularity.

Local Storage is a Web Storage API, which is a light-weight way to enable browsers to store key-value pairs [147]. The simplified API and good browser support make this option very advantageous as it is extremely easy to use.

A fundamental characteristic of this model is that the data stored in the browser has no expiration date, meaning that it will persist in the event that the browser is closed, making it adequate for storing data across an application.

The data stored in Local Storage is specific to the origin of the page, which is a combination of the protocol, host and port, and can be accessed across tabs or pages, only for that domain.

On the other hand, this option is quite insecure as the information can be accessed by any code on the web page, therefore it is unfit to store, for example, passwords.

Furthermore, there is one factor that renders this option unfeasible: it does not have enough space to store the necessary information. In the Service Studio IDE, a XML module that stores the information regarding a specific graph can contain dozens to hundreds of megabytes of information. Given that the Local Storage method can only store up to 5 megabytes [148] of information in storage, this option was discarded.

To further this, a graph object was constructed, with a size bigger than the 5 megabytes, by introducing large enough dummy variables, corresponding to the nodes present. Then, the upload button was used to import the graph onto the visual model and, as expected, an error occurred where the storage limit had been exceeded.

5.2.3.2 Session Storage

Session Storage is also a Web Storage API, and as a consequence, it shares most of the characteristics of Local Storage.

However, the main difference between this option and Local Storage is that all data stored in this session is cleared once the user closes the current tab. In addition, each tab has a separate session storage data [149].

Subsequently, this option also suffers from the same deficiency in the amount of data that it is capable of storing and, therefore, was also discarded.

5.2.3.3 WebSQL

WebSQL [150] was a third option also considered for the implementation of the browser cache. This option is also a web-based API that enables the storage of data in a relational database, which in turn can be accessed by using SQL queries.

However, a relational database would not be appropriate for this case. The information that we intend to store is not complex, simply the graphs and their corresponding items (nodes, links and data) which do not depend on distinct and correlated entities and therefore do not justify the adoption of a relational model.

Furthermore, this option has been deprecated and IndexedDB has been accepted as the standard, more efficient replacement.

5.2.3.4 IndexedDB

IndexedDB is another way to persistently store data inside a browser [151]. This option is supported by a wide range o web browsers and has the added benefit that web applications can access it in an off-line circumstance. It provides an asynchronous API that is connected to a client-side non-relational high-performance database specialized in storing large amounts of data. In fact, there isn't any limit to the data storage.

IndexedDB enables users to store and retrieve objects stored with a key and is built based on a transitional database mode [152], that is, every change that is performed on the database is within the context of a transaction. Once a uniquely identifiable object is created, it is possible to store multiple tuples inside, identifiable by an id.

This database does not have simplified support as Local Storage does. Furthermore, it does not support SQL. Instead, it uses a mechanism based on an index that produces a cursor which, in turn, needs to be iterated to find the desired result, which is not the most efficient solution if the goal of the application is to perform regular updates to small portions of data.

However, based on the advantages previously described, we decided that, given the circumstances and the requirements of the POC to be developed, IndexedDB would be the best option to integrate into the solution.

5.2.4 Implementation details

Considering the IndexedDB implementation, two objects were created, where the first stores the "graph" object (graph-object) and the second stores all the data corresponding to the nodes present in the graph (data-object). As previously addressed 5.1.2.1, this design is hugely beneficial as it drastically reduces the size of the "graph", allowing for fast updates to this structure and, when necessary, independently update the data corresponding to a particular node, as it requires a much bigger storage capacity.

It is also worth mentioning the only the last graph to be "seen" by the user is stored in the cache, meaning that when a user selects a different graph, the corresponding data is fetched to the server and, consequently, updates the cached data, simplifying the storage process and reducing the amount of data stored locally.

5.3 Central Service

Once the visual model is fully functional and the model manipulation is implemented, it is safe to say that the "client-side" of the POC is now completed. That is, both previous subsections were dedicated to the part of the POC or prototype that is responsible for the interaction with the users.

With this said, this next phase of the thesis is dedicated to the discussion and implementation of the Central Service required. In this context, this service can also be seen as the “server” component. Fundamental to any web-based application it will, not only will it be responsible for providing the users access to the platform, but also store, as well as propagate, all information relative to the visual models.

This part is inherently more complex as it requires the construction of a system composed of multiple components as well as a wide range of technologies. For this reason, this subsection starts by presenting an introduction to the Amazon Web Service ecosystem, AWS Cloud Development Kit (CDK), as well as AWS services relevant to the understanding of the system to be implemented.

With this clarified, a discussion regarding the system architecture is started where we explain how the different AWS services can be configured to interact with each other in order to produce a system capable of satisfying all the requirements necessary for the POC. Architectural decisions are described, in addition to a detailed explanation of the implementation details to better understand how the several components are designed to produce the desired outcome and, on top of that, how this Central System interacts with the client-side part of the application developed.

5.3.1 Amazon Web Services

For the construction of the Central Service component of the POC, we decided to use Amazon Web Services.

AWS is a scalable, reliable and low-cost cloud-based global infrastructure that provides cloud services (or microservices) in the format of building blocks, in other words, designed to work with each other. These blocks can subsequently be used to create, deploy and manage a wide range of types of applications in the cloud. AWS is currently used by hundreds of thousands of businesses all over the planet [153].

By enabling the construction of serverless Web applications, there is the added benefit that there is no need to maintain a server, the applications benefit from high availability and can be configured to be automatically scaled.

In addition, the users will only have to pay for the resources used, that is, there is no need to pre-calculate and determine what should be the resource allocation necessary to run the services and applications built.

5.3.2 AWS CDK

AWS CDK [154] is an open-source software development framework that helps users develop their cloud application using programming languages that they already know.

Configuring and setting up a cloud application can be a challenging task as it is required to perform several manual actions, write scripts and templates, as well as develop code in domain-specific languages.

For these reasons, CDK provides a high-level framework, based on the notion of Constructs, that enable to build components with popular programming languages, making this process much more accessible for new users. These constructs are based on tested and scalable configurations and, therefore, this architecture promotes the reuse of modules already developed by the community which, in turn, can easily be introduced in a safe and repeatable manner into new projects. This allows for rapid progression in starting new projects, helping the users to focus on specific implementation details of his application, instead of re-inventing common application structures.

This development approach is known as “Infrastructure as code”. Indeed, the components and respective configurations could also be built through the AWS Command Line Interface (CLI) using specific, and rather complex commands. However, it is much safer and easier to have a codebase that can be altered according to the users’ preferences and applications requirements that can then be re-built and promptly re-deployed into the cloud platform.

Given the advantages of this approach and anticipating a slightly complex central service architecture, that makes use of several different services, we decided to use AWS CDK as a framework for developing this part of the POC.

On the image 5.2 it’s possible to visualize how the structure of AWS CDK works.

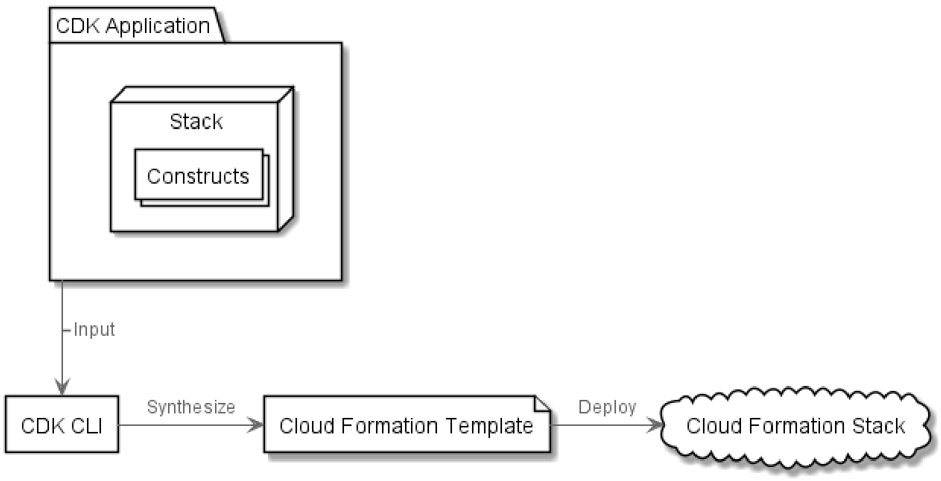


Figure 5.2: Structure of CDK.

There needs to exist a CDK application, developed by the user and that will represent the source code of the project. Inside it, the more general notion that exists is a Stack. A Stack is a root construct that represents a single CloudFormation stack which will correspond to a collection of resources and services that, in turn, build an application. Inside, each Stack will be built by Constructs, which are the basic building blocks of CDK applications and represent a particular functionality (workflow).

The CDK CLI will act as the synthesizer, that is, though a “synth” command, the source code from the CDK application will operate as the “input” and as a result will be synthesized into a Cloud Formation Template.

The Cloud Formation Template can be seen as the “skeleton” of the application or the assembly language. It consists of a template containing all configurations and settings necessary for the construction of the services that build the desired application.

Once the template is synthesized, it can be deployed to the AWS Cloud Formation functionality, in AWS, which will act as the processor and will produce all the components necessary to build the configured application. To update the application, the same process can be repeated, but this time, only the portions of the code that correspond to the alteration performed are to be deployed to the Cloud and, consequently, integrated into the application.

The understanding of the following services’ functionalities is fundamental in understanding the design of the architecture of the Central System.

5.3.3 Relevant AWS Services

With the introduction of both AWS and AWS CDK, the next logical step, to understand the building of the Central System application, is to explain what services it is comprised of, why these services were integrated, and how they interact with each other to produce the desired product.

For the implementation of this part of the project, several services were studied and considered. However, the following are the ones that were selected to integrate the solution, and, therefore, the ones that are relevant to understand.

5.3.3.1 API Gateway

Amazon API Gateway [155], as the name suggests, is an entrance point for the applications. It consists of a service that enables users to easily create, monitor and scale APIs.

This service not only processes up to thousands of concurrent API calls but also enables Cross-origin Resource Sharing (CORS) support, manages traffic and authorization configurations, throttling, version management, among other responsibilities.

There are two types of APIs available, the RESTful API, which is the standard architecture that makes use of HTTP requests, and the WebSocket API, that establishes a persistent connection between clients and the API and, therefore, enables real-time two-way communication. For now, only the first option is relevant.

5.3.3.2 Kinesis DataStream

Amazon Kinesis DataStream [156] facilitates the process of constructing a high throughput pipe that can collect and analyse data in real-time and at any scale. It is also possible to build real-time dashboards that help create a detailed actionable visual representation.

This is helpful because it enables the user to process specific types of events, highlight exceptions, find patterns, and respond accordingly to each of them.

A fair analogy to better understand the purpose of this structure is to compare it to a funnel, or even a filter that helps to process data.

The following concepts are fundamental to better understand this service:

- **Data producer**. A Data Producer is typically an application capable of producing a portion of data that, as it is being generated, is being emitted to a Kinesis DataStream.
- **Data stream**. A Data Stream is a set of shards. There are no limits to the number of acceptable shards within a Data Stream and all data will be retained in it for the default duration of 24 hours.
- **Data consumer**. A Data Consumer, generally a AWS service responsible for retrieving the most recent data generated by the Data Stream.
- **Shard**. A Shard is the fundamental unit of throughput and scale. Each shard contains an ordered sequence of records and can support up to 1000 PUT records per second. The number of shards used in a Kinesis DataStream can be specified or, alternatively, configured to be dynamically adjusted.
- **Data record**. Is the basic unit of storage in a Kinesis DataStream and is composed of a uniquely identifiable sequence number, partition key and data blob, corresponding to the data of the producer.
- **Partition Key**. Is an identifier that is specified by the Data Producer and will ultimately determine which shard ingests the data record.

5.3.3.3 Lambda Functions

AWS Lambda [157] is a serverless service that enables to effortlessly run code, that can be written in a wide variety of well-known languages, for any type of application without the need to manage or provision any type of server. It also automatically allocates computing power and manages the necessary scalability for running the code.

Lambda Functions are run based on events or triggers produced by any of the hundreds of services that are available in the AWS ecosystem. For these reasons, this service is one of the most fundamental as it often acts as a “glue” or “bridge” between different services.

This service provides numerous advantages. Among them is the fact that there is no need to set up any server or infrastructure, allowing the developers to swiftly run a piece of code. In addition, there is no need to waste time calculating the necessary resources as AWS Lambda automatically scales the application according to the workload, therefore, optimizing the overall cost necessary to run the application as it is only necessary to pay for the computation time used.

5.3.3.4 DynamoDB

Amazon DynamoDB [158] is a NoSQL database service inserted in the AWS ecosystem that was designed with the main goal of providing not only a highly available key-value storage system but also a fast and predictable performance [159].

A critical aspect of DynamoDB that is worth explaining is the fact that it is designed as a NoSQL database. NoSQL or non-relational databases were designed with the goal of producing a more efficient and cheaper way of storing data.

Among the motives that have increased the popularity of this database model, is the fact that it simplifies and removes unnecessary complexity that can be found in relational databases [160]. For some applications, ACID operations might be more than necessary and, especially in the case of Big Data applications, the priority is to be able to swiftly store large amounts of simplified data, rather than inserting it in an intricate table system, so that it can then be targeted by complex queries. Generally, one or a reduced number of data tables are required, minimizing the cost of retrieving information.

Another consequence of this simplified node-based model is that databases are designed to scale well horizontally, meaning that machines can be easily added without the same operational efforts that traditional relational databases require, such as sharding. This not only further reduces the overall complexity of the system but also the cost of setting up database clusters.

Therefore, this database model is optimized for the storage of simplified data, rather than efficiently retrieving complex information. It also allows for a flexible schema and a better scaling of the database.

The core characteristics of the DynamoDB service are:

- **Key-value data model**. Dynamo is responsible for the storage of several AWS services. The design of the database is based on a model where all data is stored in a document identifiable by a unique key. The fact that Dynamo is responsible for the storage of several AWS services and that these services can efficiently operate within this schema was a determining factor. However, this does mean that the only available operations are GET and PUT.
- **Eventual consistency**. It is possible to configure the read consistency model between eventually consistent reads and strongly consistent reads. However, as mentioned, Dynamo's main goal is to be highly available, therefore, to extract its full potential, there needs to be a compromise in reliability to achieve better performance. [161]. Besides, the first option provides a good enough performance taking into consideration that multi-region consistency below one second is not a requirement for the POC, and for this reason, it was selected as the appropriate setting for the database.
- **Decentralization**. Dynamo is a decentralized system, meaning that every node contains the same information as the others and that there is no single point of

failure, allowing for high availability of service, even in the case of failure of a database.

It also benefits from multi-region availability, automatic scalability, extremely high throughput of requests, low latency and data recovery functions.

The following concepts are fundamental for the construction of tables in DynamoDB [162]:

- **Item**. DynamoDB stores data as sets of items that are similar to rows or records in other databases and returns them based on a unique primary key value. Items are composed of a primary key and attributes, which are simple variables.
- **Partition Key**. Consists of a simple and unique primary key. Furthermore, DynamoDB will use the partition key's value to determine the partition, within DynamoDB, in which the item is stored. As a consequence, all items with the same partition key are stored together and can also easily be retrieved together, as a blob of data.
- **Sort Key**. Optional key that can be added to the Partition Key in order to form a composite primary key. This key will be used to order the items, within a certain partition.

5.3.4 System Architecture (Architectural Decisions)

With the relevant services well defined, the next stage of the project consists of integrating them, forming the System Architecture.

Before explaining the system design, it is essential to be familiar with the concept of graceful degradation, as it is at the foundation of the architectural decisions made throughout the elaboration of the system architecture. Graceful degradation [163], within the context of system architecture, is the notion that a system can still maintain most of, or the maximum possible of, its capabilities when one of its components fail.

It essentially consists of decoupling the components, or trying to implement them in a way that they can operate independently, so that their responsibilities and availability are separated, and therefore, confer a bigger resilience to the overall system. This is the reason why separate containers will be implemented within the architecture.

For the development of the Central System, an AWS Streaming Data Solution for Amazon Kinesis [164] was adopted as a solid starting point. It was then customized according to the needs of the specific solution of the POC.

This particular solution contains a pattern for capturing, storing, processing and delivering real-time streaming data, which is very adequate to include in the construction of the Central System, more explicitly, for capturing the changes made by the users on the visual models.

This pattern (5.3), uses an Amazon API Gateway as the first layer of abstraction, that allows capturing data from a non-AWS environment, the producer, which in this case

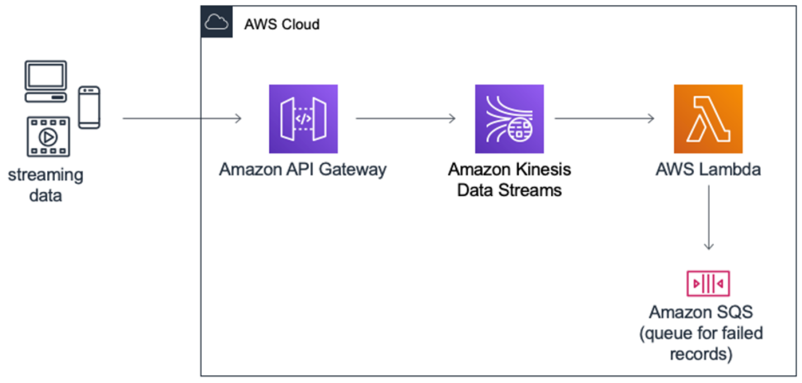

AWS Cloud
streaming data
Amazon API Gateway
Amazon Kinesis Data Streams
AWS Lambda
Amazon SQS (queue for failed records)

Figure 5.3: AWS CloudFormation template using Amazon API Gateway, Kinesis Data Streams, and AWS Lambda reference architecture [165].

will be the user, interacting through the browser, that is producing changes performed on the model.

The target of the data that passes through the API, which will merely act as an ingestion point, capable of capturing and storing large data volumes, is a Kinesis DataStream.

The Kinesis DataStream redirects all the data to the data consumer, that in this case, will be a AWS Lambda Function. Here, this function will simply read and analyse if the data has been propagated accordingly, without any error. Finally, there is an additional feature, the Amazon SQS which is a simple queue that stores any error.

This pattern is but a simple starting point for the development of the POC as it contains a ready-to-use solution with useful properties for the project, such as a mechanism that allows the ingestion and processing of data. With this in mind, all additions to the architecture were integrated “on top” of this basis.

Here (5.4), it is possible to observe the full system architecture.

The system can be divided into three separate parts. There is the IDE, which corresponds to the client-side program that the user will be interacting with. It is composed by the Flow Editor, the D3.js based editor described in the subsection 5.1.1, the impact analysis component 5.2.1, responsible for varifying the visual mode, and the browser cache (subsection 5.2.3), where the data of visual model that is currently being altered is to be stored. All operations performed over the flow editor, by the users, will be propagated to the browser cache so that all information is locally stored and is readily available.

Moreover, there is the Message System. This system is based on the pattern described above (5.3). that is, it is composed of the Event receiver API, an API-Gateway responsible for receiving all events related to alterations made by the user in the visual model, and the Message Broker, a Kinesis DataStream, responsible for redirecting the events received to the central system. The only slight difference is that the third component of the pattern, the Lambda Function is part of a distinct system. It is, however, still connected to the Kinesis DataStream.

Indeed, this architecture could also work without including the pre-built pattern and,

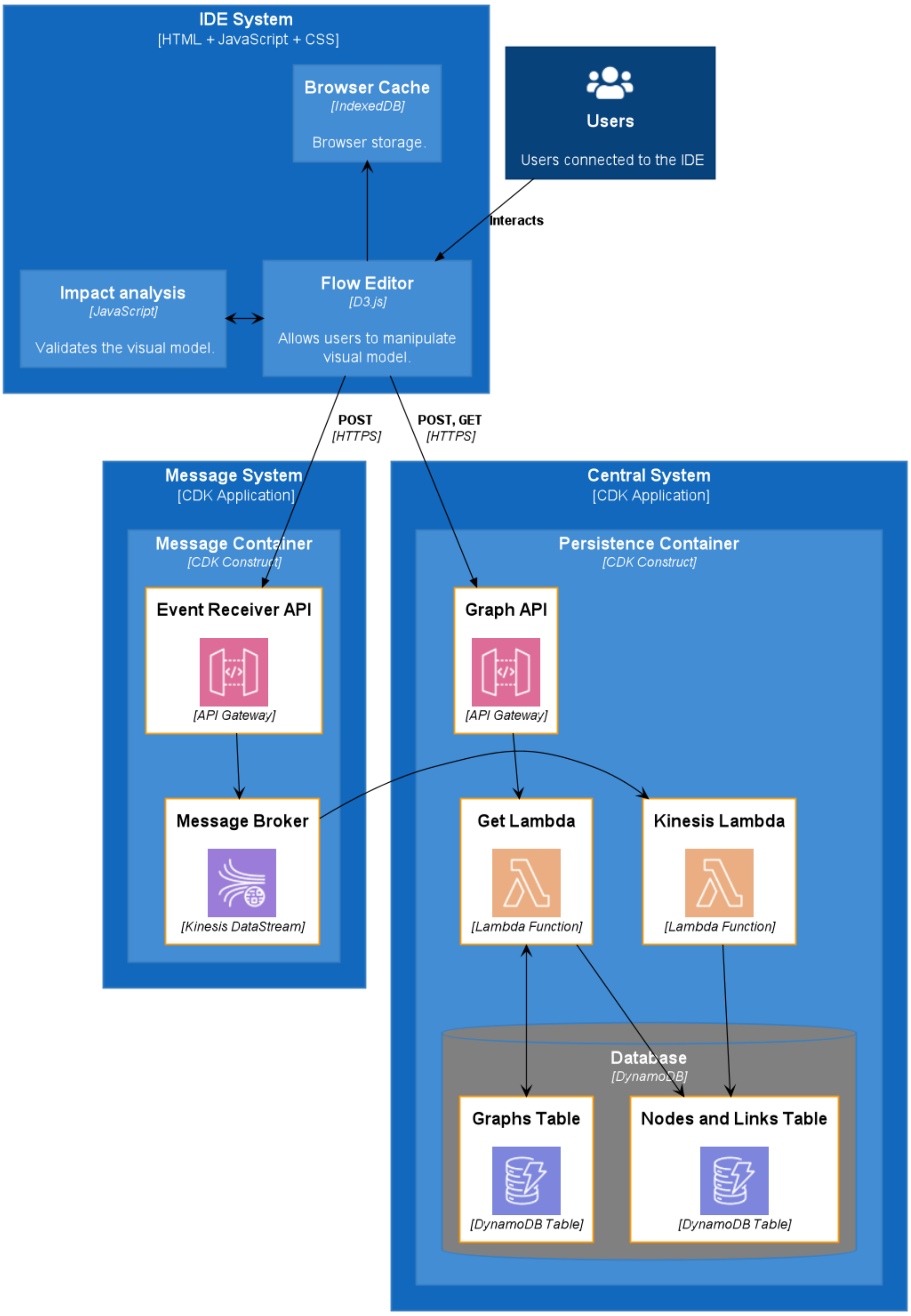


Figure 5.4: System Architecture (based on C4 model 2.4.1).

consequently, without the Message System, reducing it to a simple connection between the "Event Receiver API" and the "Kinesis Lambda". However, the Kinesis DataStream provides further robustness by decoupling the architecture in distinct containers. This is beneficial because, even if the Central System fails, the Message System is capable of continuing to receive the events produced by the IDE System, store them for up to a week and, once restored, propagate them accordingly to the Persistence Container. This allows the users to use the Web-IDE despite the state of the Central System, otherwise, the events would be lost and the whole IDE would simply block.

Lastly, there is the Central System which is the most complex of the three systems in this architecture. It is possible to observe a component by the name of "Kinesis Lambda" which concludes the third service present in the previous pattern.

In this case, this Lambda Function is not connected to any queue for failed records but is instead linked to a DynamoDB table. In the context of the project, the goal is to store data, or rather, store all changes performed on some visual model, so that it can then be retrieved in its totality, once a user enters the IDE. For this reason, the Lambda is responsible for the interaction (insertion and deletion) with the "Nodes and Links Table", contained in DynamoDB, where all the details regarding nodes and links of all models are stored.

Furthermore, the Central System contains yet another API-Gateway, the Graph API. This second API has a distinct functionality from the Event Receiver API. Instead of acting as a "busy" gateway, responsible for receiving every single alteration made to the model, it will act as an auxiliary API that processes less frequent requests. For this reason, it would not benefit from the advantages of integrating a Messaging System as previously addressed, thus, this API does not need to be connected to a Kinesis DataStream.

This secondary API will be responsible for:

- Retrieving all the information of a specific graph (nodes and links of a graph).
- Creating entries in the graphs table, that is, creating graphs and associating which users have access to it.
- Retrieving all the names of the graphs that a certain user has access to.

These actions will be performed by the "Get Lambda", which is, analogously to the "Kinesis Lambda", responsible for processing the requests of the Graph API, which are the required interactions with the database.

Recapitulating, and looking at the architecture "from above", first of all, all changes performed over the IDE will not only be stored locally but will also be translated and placed into an event. Once this event is triggered, is then propagated to the Message System, responsible for receiving, processing and redirecting it to the Central System. Within the Central System, the information contained inside a certain event will be stored accordingly, inside the DynamoDB. In addition, and to complement this workflow, the IDE makes use of the Graph API, which is essentially an auxiliary data flow used to retrieve and store specific information regarding the graphs model kept in the database that ensures the correct behaviour of the POC.

With this said, the core of the system architecture is complete, or in other words, all services required to receive, process and store all events of changes performed to the visual model, as well as the required mechanism of retrieval of data, are now built and integrated.

5.3.4.1 Implementation details

Regarding the IDE System, it is relevant to note that in a "real" architecture, such as the case of the Service Studio IDE, it would not be necessary to fetch the graph information from the server as the latest version would already be stored in the browser cache, either through model insertion or via a subscription system in the case of real-time collaboration. This is particularly beneficial in a scenario where a single graph contains a large amount of data, and, therefore, retrieving it from the server is time consuming. Or in a scenario that it is possible to maintain more than one flow "open" for alterations.

Nonetheless, it was decided that in this case, given the scope of the project, it is sufficient to fetch all the necessary information directly from the server, when needed, as it adds no significant delay and satisfies all the requirements. This means that the cache will only be used to store the data of the visual model that is currently being altered. Thus, even though this component is not a piece of the architecture that is being extensively used, it still is fundamental to expand this prototype into a scenery of bigger complexity.

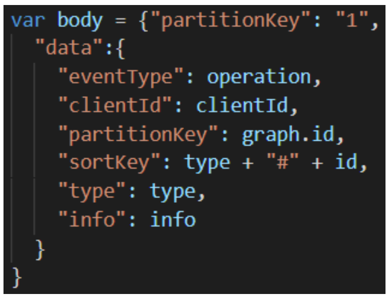
It is worth noting that, in this case, both the Messaging System and the Central System are defined and configured within the same CDK project. In a "real-life" project this would not be the case because both systems have distinct functionalities and as such, they would be developed separately, possibly by different teams. Only once fully operational, would the connection between them both be implemented. For the sake of simplicity, we decided that both systems would integrate the same CDK project so that potential additional errors or incompatibilities are eliminated.

When carefully observed (5.4), it is possible to note that the "Event receiver API" only receives POST requests, despite the fact that these events can contain both insertions (or updates), and deletions of nodes and links. This is because in streaming data patterns, or in event driven architectures, the goal is to have one single operation available in the API, a post-operation, capable of receiving all events emitted.

However, all events, which can also be seen as Data Records, consist of a JSON type file with a specific format containing a field where the operation is determined. In other words, in the eyes of the API, all requests will be POST requests so that they can be equally processed by the Kinesis DataStream, only when it "leaves" this service and is redirected to the Lambda Function, is the Data Record "deconstructed" and the operation to be performed over the database is identified, as well as all other required data.

The following image (5.5), obtained from the source code of the project, and table

(5.5) detail the structure of the Data Records and the purpose of each of the variables within it. The variables of the table are presented in the same order as observed in the image (from top to bottom).



```
var body = {"partitionKey": "1",
  "data":{
    "eventType": operation,
    "clientId": clientId,
    "partitionKey": graph.id,
    "sortKey": type + "#" + id,
    "type": type,
    "info": info
  }
}
```

Figure 5.5: Structure of an “Data Record” (screenshot from source code).

To clarify the design choices of the database schemes, in the “Graphs table”, the partition key corresponds to the “userId” and the “sortKey” to the “graphId”. The reasoning behind this schema decision is so that, by selecting a specific “userId”, it possible to return all the visual models associated with it. On the other hand, the only attribute is the name of the graph. This choice is useful for displaying all the names of the graphs the user has access to in the “Load graph” drop-down menu (5.1.2.2).

Regarding the “Nodes and Links Table”, the partition key is the “graphId” and the sort key is the “nodeLinkId”. On the side of the attributes two variables can be observed, the data variable, where all the data relative to a specific node, link or data is stored, and type, which determines the type of the variable. This schema enables the storage and retrieval of all variables that are associated with a certain graph.

The same system requirements could be satisfied by implementing just one DynamoDB table. However, this implies that with each change made to the model, and this change could simply be a small positional adjustment of a node, would result in the update of the entire graph tuple, which is not optimal. From a performance point of view, it makes much more sense to separate this table into two separate instances, one containing the names of the graphs, connected to the users that have access to it, and a separate one containing a tuple for each type of object (“node”, “link” or “data”). This way, changes are decoupled and lighter to perform.

5.4 Real-time collaboration

This section will cover the elaboration of the fourth and final phase of the POC, more concretely, a real-time collaboration mechanism.

Several architectural alternatives are considered in order to identify is the best solution to integrate into the project. Both disadvantages and advantages of each option are debated, followed by a conclusion.

partitionKey	As described (5.1.1), this variable will be used to select the Data Shard to be used. In this instance, a single Shard is capable of satisfying the throughput required for the prototype and, as such, the variable is hard-coded to "1", merely an exemplification.
eventType	Type of event that can be executed. As described, because the database in question is a NoSQL database, the operations available are both "put" and "delete".
clientId	Corresponds to the unique identifier (generated UUI) of the client that created the event. Usefull for the next phase of the POC.
partitionKey	Although this variable has the same name as the first one, here, Partition Key is in the context of DynamoDB, which also uses the same naming terminology. In this case, corresponds to the unique identifier of the graph that is being altered.
sortKey	Sort Key variable in the context of DynamoDB and corresponds to a concatenation of both the type of the object in question, as well as the it's unique identifier.
type	Corresponds to the type of the object in question. This variable can be one of the following options: "node" or "link". The reason why there is not a "data" option, despite being a valid type of object to be inserted in the database is, as already explained (5.1.2.1), it fulfills a demonstrative role as and such it cannot be altered.
info	This field will contain different information depending on the type variable. In the case it matches "node", if it is an insertion or update, it will contain both a "px" and "py" variables that corresponds, accordingly, to the horizontal and vertical coordinates within the visual canvas. In the case it corresponds to a "link", it will contain the both "source" and "target" variable that corresponds to the extremity nodes that compose the link. In both cases, if the operation is of deletion, this variable will be empty.

Table 5.5: Table to test captions and labels.

After that, a revised system architecture is presented, which integrates the developed subscription mechanism. Furthermore, relevant implementation details are presented as well as several thorough descriptions of how the various components within the final architecture interact with each other, in distinct contexts.

5.4.1 Alternatives considered

When evaluating possible solutions for the integration of a real-time collaboration mechanism in the POC, or in other words, how to implement a mechanism that enables the back-end, or server, to send, in real-time, the updates performed over a certain visual model, to all the connected clients that have access to it, the following architectural options were considered.

5.4.1.1 Polling

With polling, also known as "client pull", the front-end application has the responsibility of sending recurring requests to the server, separated by a fixed delay, to "ask" if there is any new data that can be retrieved. The smaller the interval between requests, the better the probability of handling fresh data [166].

There is also the option of Long Polling, based on the Comet model [167]. The difference in this version is that the requests are maintained open either until the required data is returned or when a time limit is reached.

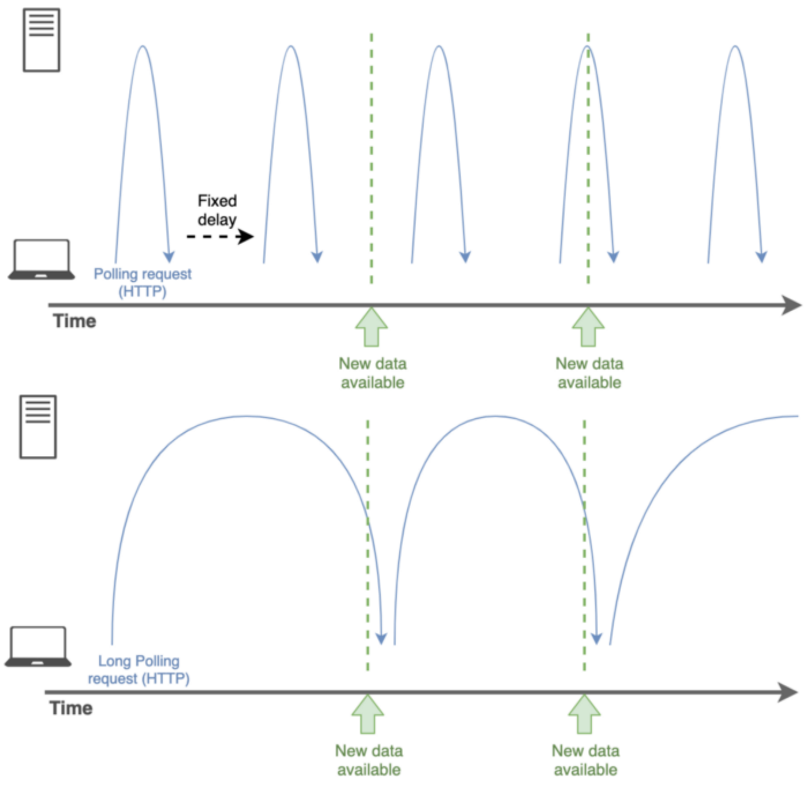


Figure 5.6: Polling (above) and Long Polling (below) [166].

Disadvantages:

- Polling is usually supplemented with a delay between requests so that the intensity of requests sent is not as high. As a consequence, this will most likely result in some delay in retrieval of the required data, making it so that a real-time response is unfeasible.
- It is necessary that, in the context of AWS, a Lambda Function is permanently kept alive to repeatably send requests, which results in a high amount of requests that simply do not return any data, leading to bad management of the computing resources and, as consequence, increased monetary costs.

Advantages:

- The only advantage that this option might present is its simplicity of implementation using simple HTTP requests.

This is not an optimal solution. As such, nowadays, it is not a pattern that is adopted.

5.4.1.2 Server-sent Events

With Server-Sent Events, also addressed as "server push", the front-end application needs to open a uni-directional connection, from the back-end to the front, with unspecified lasting duration [168] and, contrarily to Polling, in this case, the server is actively responsible for asynchronously pushing updates to the client. Moreover, whenever new data is available, it is then pushed to the front-end.

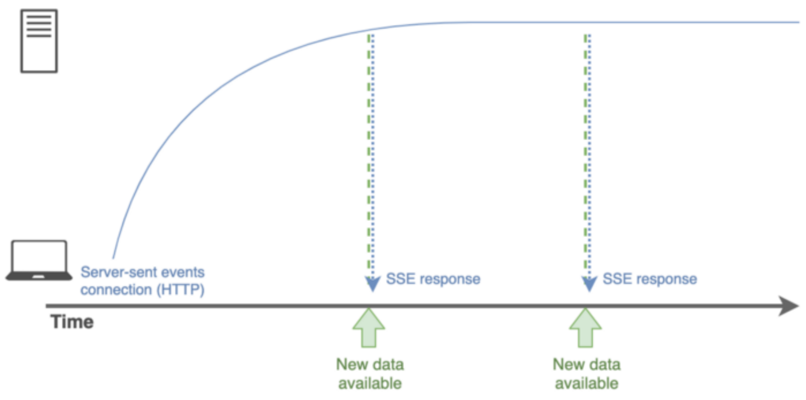


Figure 5.7: Server-sent Events [166].

Disadvantages:

- Identically to the previous option, it's necessary to keep a Lambda Function constantly active in case it is necessary to send any message to the client. As a consequence, the expenses cost increases in order to maintain a Lambda permanently active.
- This is not an optimal architecture. Nowadays it can be considered a "work-around" option as it cannot obtain the same performance as, for example, a WebSocket connection [169].

Advantages:

- Easy to implement and data efficient.
- Automatically multiplexed over HTTP.

5.4.1.3 AWS Cognito Sync

Yet another option that was considered as this Service enables to synchronize data between clients. However, this synchronization process is made with the aid of authentication, consequently, credentials are required, which is not a feature that is expected to be included in the POC as it falls out of the scope of the project.

5.4.1.4 WebSockets

WebSockets [170], contrarily to the options previously described, takes advantage of a full-duplex connection between the user's browser and the server, implemented over a single Transmission Control Protocol (TCP) connection. Furthermore, this connection configuration enables the back-end to send a message to the browser as soon as possible.

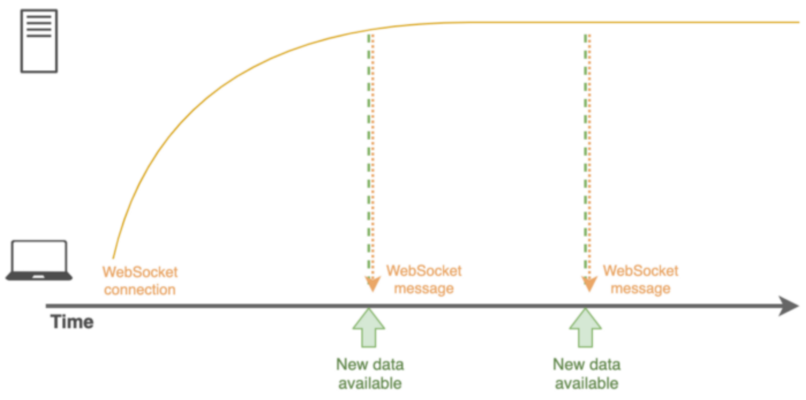


Figure 5.8: WebSockets [166].

Disadvantages:

- For this solution to be integrated into the project, the already defined APIs would have to replace by WebSocket APIs, which would lead to a refracting and cumbersome task rather than an improvement or addition.
- It would be necessary to manually manage and configure the necessary WebSocket connections, which are not automatically multiplexed over HTTP, define its scalability, implement fault-tolerance, etc...

Advantages:

- Flexible and standard solution.
- Full-duplex interaction in real-time between the client and the server.
- AWS already provides a WebSocketAPI that facilitates the handling of the connection process.
- It is a straightforward and cheap solution to implement.

5.4.1.5 AppSync

AWS AppSync [171], yet another AWS service, is a fully managed GraphQL API layer that has the goal of improving application performance by allowing to filter, select and combine data from multiple sources, such as Lambda or DynamoDB, and providing simplified data access through a unified API.

GraphQL [172] is a data language and is especially valuable when it is required to fetch information from more than one data source. This is because it enables clients to query for the portions of information desired, despite being distributed among more than one data source, therefore, retrieving only the needed data.

This is especially beneficial in a mobile environment with limited bandwidth because it is possible to select only the required fields from a certain data source, optimizing performance. It also has the added benefit that the information can be retrieved in the most suitable format.

AppSync takes advantage of GraphQL subscriptions, informing the service or client which data should be updated, to perform real-time operations by pushing data to clients that choose to listen to specific events from the back-end.

To complement the subscriptions, there is also the notion of GraphQL mutation. A GraphQL mutation is a call that is made through the AppSync API and once is performed, will alert all clients, that have a subscription in place with the specific GraphQL mutation, that a change was carried out and, consequently, this change, or mutation, is propagated to them by the AppSync service, with the final goal of integrating the change to the local model of the user and, with this, achieve a real-time collaboration architecture.

This service is designed to perform efficient queries to several data sources, which will not be the case as the goal of this project is to simply make use of the service DynamoDB, and therefore this feature would not be utilized. On the other hand, as previously mentioned, that is a quite complex service and as such, to not fully take advantage of all its features is not necessarily bad if it still can satisfy the requirements.

The connection management is handled automatically by the AWS AppSync client Software Development Kit (SDK) or AWS Amplify client using WebSockets as the network protocol between client and service. AppSync can be configured to be accessible through a user identification mechanism or a simple universal API key.

Disadvantages:

- Complex service, containing a lot of features, which could potentially lead to a steep learning curve.
- Requires some planning and effort defining how this service can be integrated with the rest of the already developed project.
- Not knowing if this option is capable of delivering a performance good enough to make real-time collaboration feasible.

Advantages:

- GraphQL facilitates the process of managing Lambda Functions and data sources as well as conflict management.
- Facilitates real-time collaboration between clients through broadcast features that enable propagating changes to all subscribed clients.
- Very advantageous solution in terms conflict resolution, offline development [173] and due to its serverless nature.
- Contains a server-side fully managed caching mechanism that by storing data, eliminates the need to directly access the data and proves both high availability as well as low latency.

5.4.2 System Architecture with Subscription Container (Architectural Decisions)

The most common solution would be to integrate the project with WebSockets, as it is flexible and overall simpler. However, it was decided that the AppSync service would be the best option to include in the project. The reason for this is that, besides the advantages previously mentioned, it consists of an alternative, less-known solution that confers individuality and interest to this particular architecture, thus, also increasing its overall value.

The revised system architecture, which includes the Subscription Container can be observed in figure 5.9.

Here, it is possible to note that the “Kinesis Lambda”, within the Persistence Container, has a new connection directed towards the AppSync API. This is because, in this new version, every time an event is triggered and directed to the Persistence Container, this Lambda function will also be responsible for converting this event into a GraphQL mutation so that an alteration to a visual model can then be sent to the other users that have access to the same model.

It is necessary to understand that within the AppSync service, there is a GraphQL API, responsible for receiving the mutation requests, and a Lambda function that corresponds to its data source. However, AppSync has the responsibility of propagating these mutations towards the Subscription client of each user. Furthermore, to complement the default workflow of an AppSync service, the “AppSync Resolver” function is a connection to a database that stores each mutation in tuples, which can be accessed in the future.

Finally, the last component to be added to the version is the Subscription client in the IDE System, which consists of a AWS Amplify Client, that is fundamental for the users to receive the necessary mutations. In this case, a user will create a GraphQL subscription for each visual model that he has access to and will receive all alterations made to the model via mutations, except for the ones made by the client itself.

For this project, we decided, although it is not the safest option, to access the AppSync API through its particular key. This is because the POC does not include the notion of users as it would not add any insight to the problem in question, and as such, the option chosen represents a simpler and direct solution.

To maintain a rigorous and consistent design of the system architecture, it would make sense, similarly to the messaging system, to include a Kinesis DataStream between the "Kinesis Lambda" and the "AppSync API", to process the events. However, to remove any unnecessary complexity of the system, it was decided that this detail would not be included as this pattern has already been included in the system and its advantages have been addressed.

Although both Containers (Persistence and Subscription) could be implemented as one, it was decided to decouple them. This is because, in the eventuality that, for example, the service of AppSync is temporarily down, and consequently the whole Subscription System is too, it does not compromise the rest of the Central System. In case this did happen, it would still be possible to use the IDE, except that the subscription to changes performed by other users to the same visual model would not be available. This decoupling process enables to increasing the overall resilience of the Web-IDE.

5.4.2.1 AWS Amplify

AWS Amplify [174] is a set of tools and services designed with the goal to aid the development and deployment of scalable mobile and web applications.

With the help of AWS CLI it is possible to automatically configure cloud back-ends, connect them to front-end applications and improve development workflows, such as sign-up features and even integrate UI components.

Within the context of the POC developed. AWS Amplify was useful to connect the back-end infrastructure CDK code with the client JavaScript application. By integrating an API to access Amplify in the client-side React project, and configuring the client to have access to the AppSync API, it became possible to create subscription channels that listen to any change made to useful data. In this case, to listen to any change made to a model that is being altered by more than one user at the time.

5.4.2.2 Implementation Details (Data Flow)

In this final version of the architecture, the "Kinesis Lambda" is not only responsible for storing the events, as previously described in the subsection 5.3.4.1, in the database, but also to convert them into mutations to be sent to the GraphQL API.

A mutation consists of a POST request sent to the GraphQL API, specifying a particular GraphQL mutation to be performed. In this case, every client will have two active subscriptions, one that is listening to mutations that correspond to any update/insertion action in the visual model, and another that corresponds to deletion actions. The type of

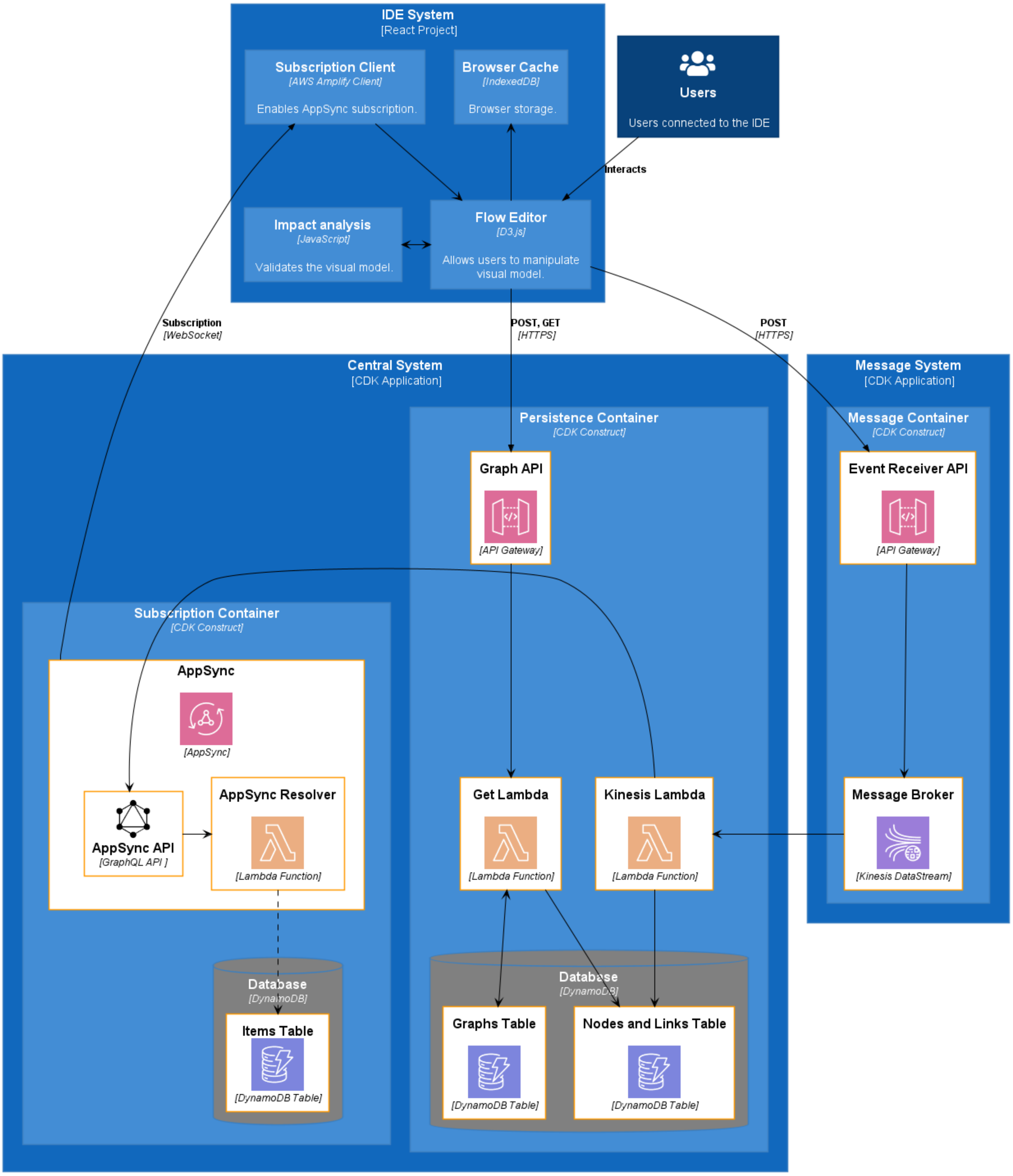


Figure 5.9: Real-time System Architecture (based on C4 model 2.4.1).

operations or mutations to be triggered will depend on the type of event produced when the user is manipulating the visual model.

The information contained in the events, more specifically, what type of operation is performed over which “node” or “link”, will be converted and propagated within the mutation to all the clients that are currently accessing a particular graph. Then, the subscription client is responsible for receiving and decoding the mutations and, consequently, determining what is the operation that the flow editor has to perform.

In conclusion, every time a user performs any action in the graph that’s currently being altered, an event is produced and propagated to the server which, in turn, is converted into a specific mutation that is then broadcast to every client that has an active subscription. As a result, every client will have the synchronized version of the visual model, allowing for real-time collaboration between users that have access to the same graph.

The event variable “partitionKey”, which is the identifier of the graph over which the operation is performed, will be used to configure the client subscriptions. The subscriptions will be limited to the mutations in which this variable is the same as the identifier of the graph that the users is currently altering. This is so that clients only receive the mutations over the visual model that they are currently working on.

The event variable “clientId” is helpful because the subscriptions in place will only receive mutations in which this variable differs from the ID of the client that is receiving it. In other words, every event will be identified by the client responsible and will only be propagated to different clients, as there is no need for the clients to be receiving mutations from the alterations performed by themselves.

Given that all four phases of the POC are implemented, the following sequence diagrams have the goal to better illustrate how the distinct elements present in the architecture interact and complement each other, within distinct contexts.

It should be noted that each of the actors in these diagrams, for reasons of simplification and given that the detailed interaction between the singular components has already been detailed, corresponds to either a container, system or actor, so that a broad yet clear notion of interactions is presented. Also, every arrow present corresponds to a synchronous call.

5.4.2.3 Start Workflow

The first diagram (figure 5.10) portraits the behaviour of the prototype when the user starts using the web application, in other words, what tasks need to be completed in order for the application to be ready for the interaction with the user.

Firstly, the user obviously needs to access the IDE. After that, two tasks are started simultaneously and run in a parallel fashion. On one hand, interaction with the persistence container is initiated involving both a request to retrieve the information of the default graph to be displayed in the visual model of the user and also a request to store all the names of the graphs that the particular user has access to. Both requests are performed with the help of the "Graph API". After this, the browser cache is also updated with the data of the graph retrieved. On the other hand, the real-time subscription mechanism is initiated where a subscription request is sent to the subscription container, regarding the default graph and, consequently, a WebSocket connection is put into place, which will continuously run so that the user can receive any possible alteration performed over the graph by any other user that has access to the same model. This connection will only conclude once the user decides to close the browser, that is the reason why, in the diagram, there is no specific stop to the loop section.

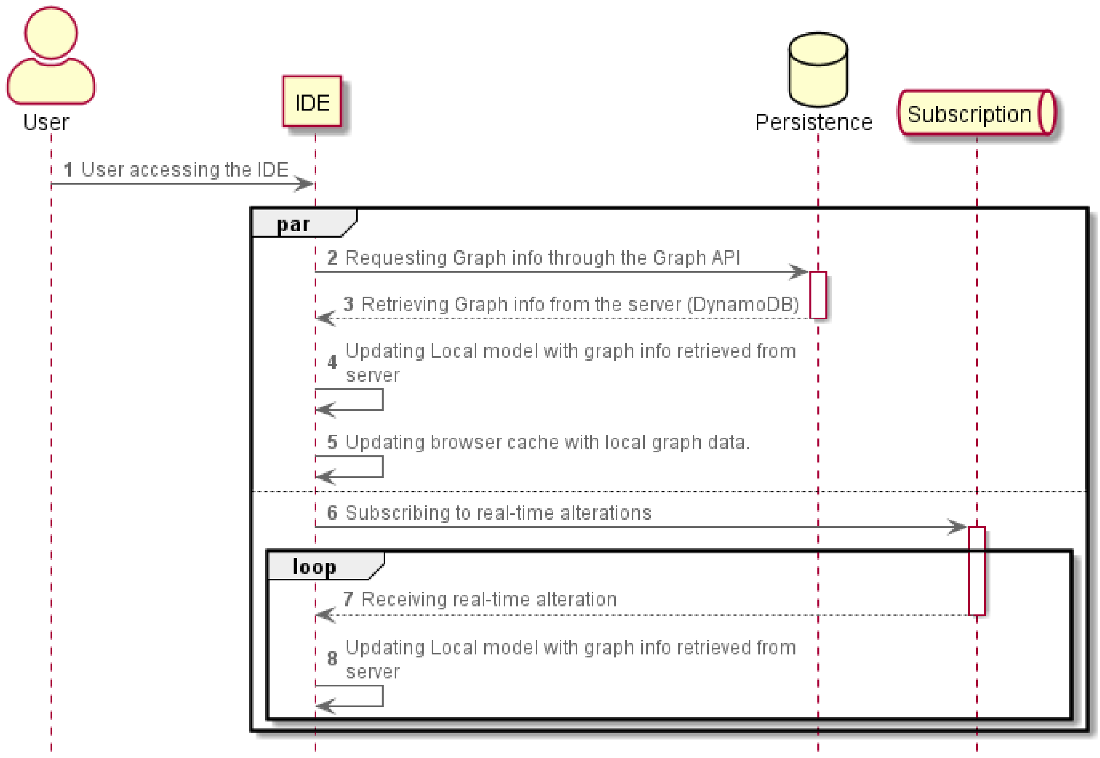

User
IDE
Persistence
Subscription
1 User accessing the IDE
par
2 Requesting Graph info through the Graph API
3 Retrieving Graph info from the server (DynamoDB)
4 Updating Local model with graph info retrieved from server
5 Updating browser cache with local graph data.
6 Subscribing to real-time alterations
loop
7 Receiving real-time alteration
8 Updating Local model with graph info retrieved from server

Figure 5.10: "Start" Sequence Diagram.

5.4.2.4 Add Graph Workflow

This second sequence diagram (figure 5.11) depicts what happens when a user selects the "Add Graph" input.

Initially, a new local graph object needs to be created. Afterwards, three distinct tasks are performed simultaneously. A new graph object is created in the database, more specifically, a tuple in the "Graphs Table" associating the new graph with the user in question, both the "Select Graph" drop-down and the visual model are updated and, finally, the cache is also updated with the new graph.

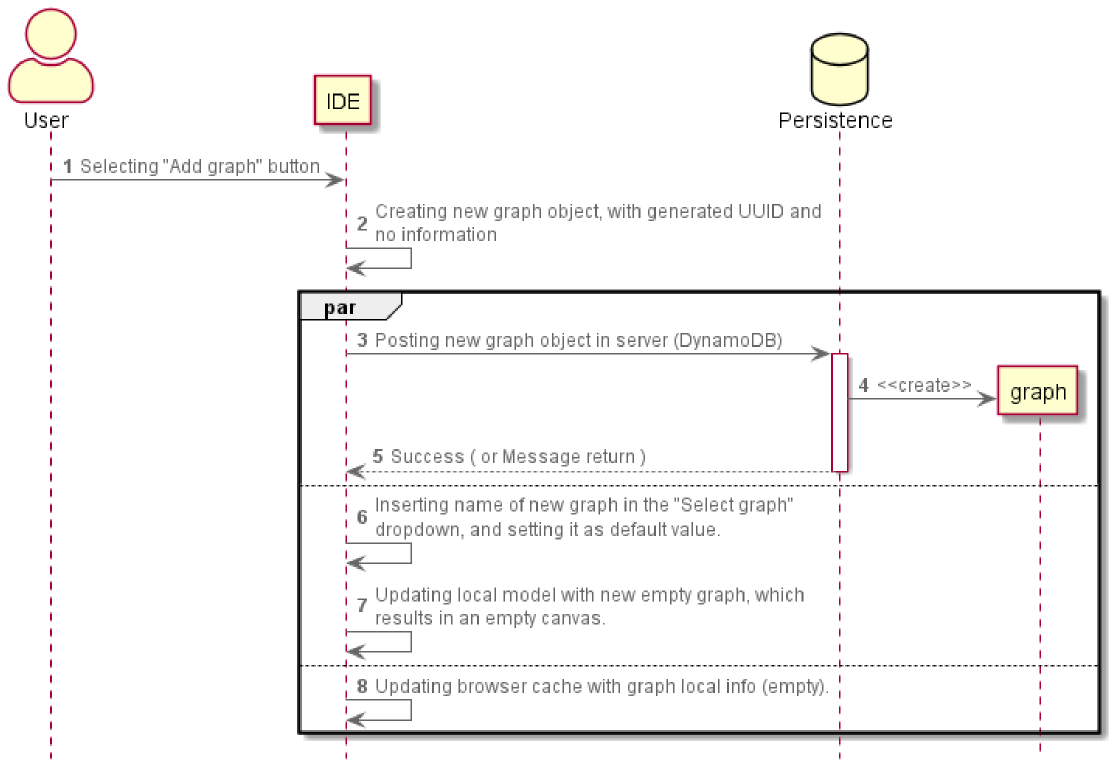

User
IDE
Persistence
1 Selecting "Add graph" button
2 Creating new graph object, with generated UUID and no information
par
3 Posting new graph object in server (DynamoDB)
4 <<create>>
graph
5 Success (or Message return)
6 Inserting name of new graph in the "Select graph" dropdown, and setting it as default value.
7 Updating local model with new empty graph, which results in an empty canvas.
8 Updating browser cache with graph local info (empty).

Figure 5.11: "New Graph" Sequence Diagram.

5.4.2.5 Select Graph Workflow

Lastly, the third sequence diagram (figure 5.12) delineates the behaviour of web-IDE when a graph from the "Select Graph" drop-down is selected.

First of all, the user needs to select the "Select Graph" drop-down and select the name of the graph that is intended to the loaded into the visual mode. After that, a request is sent through the "Graph API" with the purpose of retrieving the information stored in the database that corresponds to the graph in question.

Based on the received information, the local model is updated and, consequently, verified while "Select Graph" drop-down is updated to display the name of the newly selected graph and, also, the browser cache is updated.

5.5 Improvements

Once the four phases of the POC were implemented, it was noticeable that the overall latency of the web-IDE was not fluent enough (averaging a response time of 1.7 seconds) to provide the users with a fast and pleasurable experience as a real-time collaboration environment should.

As such, the system architecture and, consequently, the implementation of the AWS services was not only re-examined but also a few meetings were held with an AWS Technical Account Manager to find potential improvements regarding the overall latency of

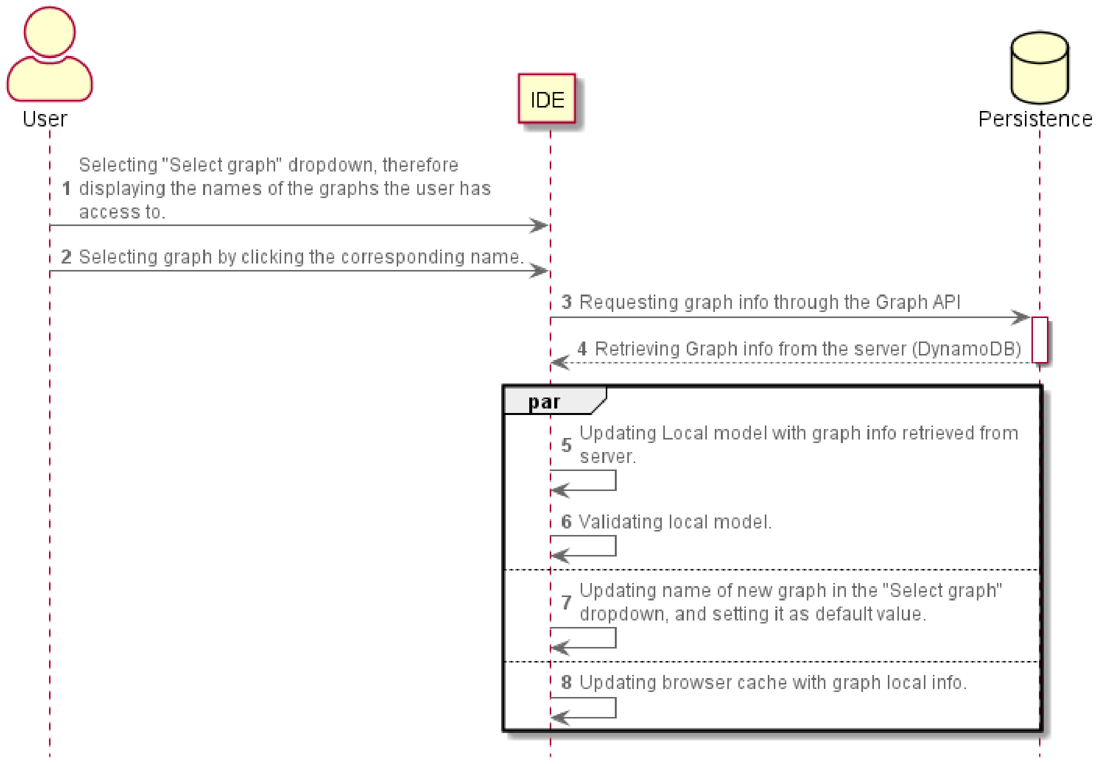

User
IDE
Persistence
1 Selecting "Select graph" dropdown, therefore displaying the names of the graphs the user has access to.
2 Selecting graph by clicking the corresponding name.
3 Requesting graph info through the Graph API
4 Retrieving Graph info from the server (DynamoDB)
par
5 Updating Local model with graph info retrieved from server.
6 Validating local model.
7 Updating name of new graph in the "Select graph" dropdown, and setting it as default value.
8 Updating browser cache with graph local info.

Figure 5.12: “Select Graph” Sequence Diagram.

the system.

The AWS web console was used to inspect, with a great level of detail, exactly how much time it took for a request to “travel” through each Service. Inside this console, there is a section for each of the distinct services and, within it, it is possible to find several dashboards or logs that detail the time, in milliseconds, at which the request first arrived at the service (for example, the API Gateway) or how much time it took for the service to fully process the request (for example Kinesis DataStream and Lambda Functions).

At the same time, a function, implemented in the client-side of the application, was used to record the exact time at which a request was sent to the Server-side and the time at which the response arrived. With this information, it was then possible to compare it to the time-stamps inside the AWS services and to determine how much time it took for the request to travel from the IDE System to the AWS cloud, and vice-versa.

Furthermore, to accurately measure the times shown in each of the following figures, and eliminate possible outliers or anomalies, an average of a set of ten requests was used. The figure 5.13 depicts how much time each stage takes. In some diagrams, the timestamps in the lower part may be rounded, in order to allow the tool used to produce visually intuitive diagrams and the times are easily comparable.

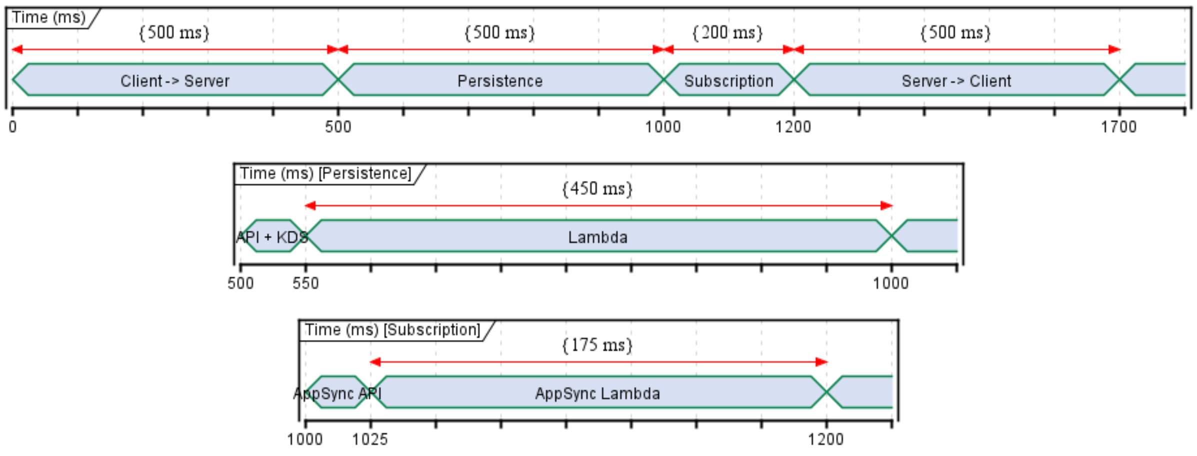


Figure 5.13: Latency of the system (Initial Version)

5.5.1 Persistence Container

Based on figure 5.13, it is possible to conclude the "Persistence Container" and, more specifically, the Lambda Function within it is taking a disproportionate amount of time compared to the rest of the components and the "Subscription Container".

With this being said, the first action that was taken to decrease the overall latency of the system was to remove the need to wait for the response of the insertions in DynamoDB tables.

In other words, the "Kinesis Lambda" was awaiting a success message from the DynamoDB database to trigger a message to the "App Sync API". This proved to be highly inefficient because, first of all, the insertion will only fail if the DynamoDB service is unavailable, which is unlikely, and secondly, even if this insertion temporarily fails, the events will continue to go through the Subscription Container and, consequently, propagate the alterations being made to the visual model by more than one user at the same time, not compromising the real-time experience.

This resulted in a significant improvement as the Lambda Function duration decreased by 44%

Furthermore, the next action taken was to increase the total memory allocated for the Lambda Function to run. Initially, the value was at 128 megabytes and was then increased to 512 megabytes. This translated into a further improvement in the run time of the function that culminated in the final duration of approximately 200 milliseconds, as noticeable in the figure 5.16. Conferring a total duration time of the Persistence Container to approximately 200 milliseconds and an overall improvement of 67% of the duration.

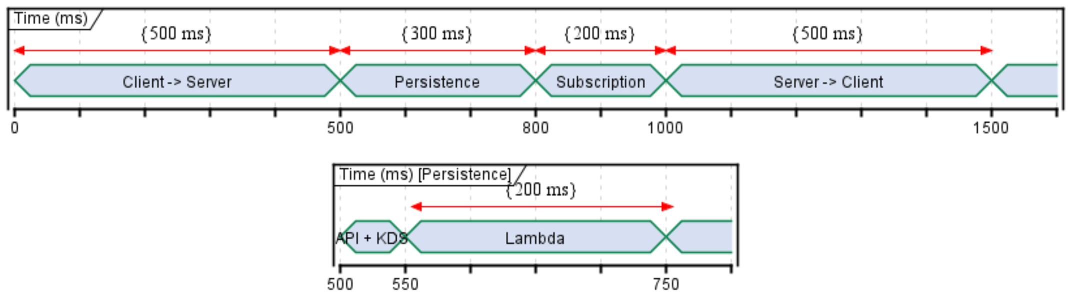


Figure 5.14: Latency of the system (Second Version)

In addition, and despite not having a direct impact on the performance of the system, the DynamoDB tables were initially set with a provisioned amount of resources. And so, in order to ensure that scalability and the optimal architectural design is implemented, this definition was changed to provide as many resources needed ("On-Demand") at any time so that in a possible busy time frame, where a larger than usual amount of request is made to the service, the insertions still take place as expected.

5.5.2 Subscription Container

The second point of improvement is associated with the AppSync Service. Off-line development was initially seen as a relevant aspect to include in the POC. However, as development progressed and more detailed discussions of the requirements to implement took place, this requirement descended towards a useful addition, rather than a focal point in the architecture.

This is because AWS Amplify provides an Amplify DataStore that serves as a local persistence repository that then automatically synchronizes the data with the cloud (AppSync service). Not only that, but it also provides versioning, conflict detection and off-line development by allowing the mutations, and queries, to be stored locally and, in case there is no internet connection, wait for it to be re-established and propagate them to the AWS Cloud [175], where incompatibilities are solved and the information is stored.

With this being said, it is true that AWS AppSync allows for off-line development. However, this is based on the premise that it is directly connected to the database where the graph-related information is stored. By analysing the System Architecture, this is not the case. The graph information is instead located inside the Persistence Container, meaning that in order to include this offline development feature, a complex re-design of the system architecture would be required. This is why this particular feature ultimately became part of future work for this project.

Furthermore, and with figure 5.9 in mind, it is possible to observe that the connection between the "AppSync Resolver" Lambda Function and the DynamoDB is dashed. The reason for this is that the Database within the Subscription, given the circumstances, has no real utility, in fact, it even adds further delay to the response and so, it was decided to not include this service in the final system architecture. This second alteration to the system resulted in an improvement in the latency by approximately 85 milliseconds.

Finally, and identically to the previous subsection, the memory allocated to run the "AppSync Lambda" was increased from 128 megabytes to 512 megabytes, resulting in a further improvement of the duration. As a result, as seen in the figure 5.15, the duration that the "Subscription Container" takes to process a request is extremely small, averaging at only 20ms, which corresponds to an improvement of around 90%, when compared to the initial version.

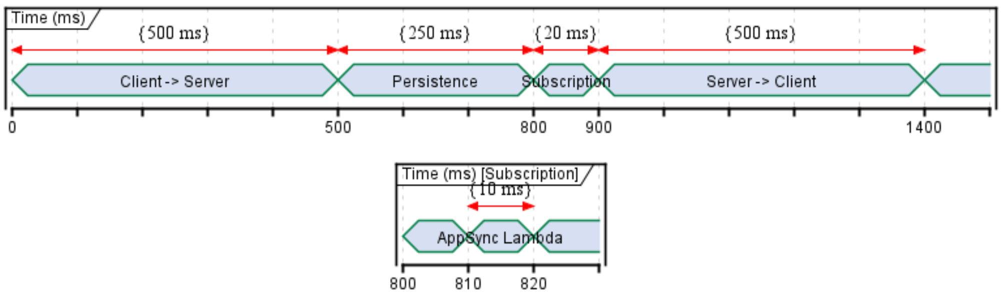


Figure 5.15: Latency of the system (Third Version)

5.5.3 CDN

With the previous improvements in mind and looking at the timing diagrams, it is clear that the main obstacle regarding the overall latency in the interaction with the Web-IDE is the time the messages take to travel between the IDE System and the Server-Side components, and vice-versa.

This is because, in OutSystems, the default region for the deployment of applications, within the context of AWS is North Virginia, in the USA. North Virginia is approximately 5,800 kilometres from Lisbon and as result, the time it takes to propagate any message to a server, and back, in this location will always hinder the speed performance of the web application.

To try to mitigate this, an improvement that was considered was to integrate a CDN, which in the context of AWS would correspond to the CloudFront service [176], in the system architecture, between the client-side of the application and the various APIs present.

A CDN is a service that accelerates internet content delivery or simply put, it makes websites, or web applications faster. Essentially, what it does is it replicates the contents of the servers where the content is stored to locations that are geographically closer to the end-user (to replica servers) that is requesting it [177], drastically reducing the amount of time it takes to deliver that content.

Furthermore, there is also the indirect benefit that this structure reduces the amount of traffic that reaches the main servers, distributing it through the edge locations where the CDNs are stored, reducing the amount of capacity required by the main server to serve all its users.

However, when investigating the implementation of this addition to the system, it was noticed that the responses of the requests made from the client-side to the server were made via CloudFront. The reason for this is that, curiously, the API Gateway service benefits from edge-optimized API endpoints [178], which routes the requests to the nearest CloudFront Point of Presence.

In fact, this is a default configuration which makes the introduction of a CDN redundant and, for this reason, no further actions were taken as the service is fundamentally already integrated into both the API Gateway components and, therefore, its advantages incorporated.

On the other side, regarding the Subscription Container, options to implement edge optimization (CloudFront distribution) when processing AppSync GraphQL subscriptions were also investigated. Upon researching this matter and soliciting support from the AWS Support Center, we concluded that GraphQL subscriptions are not supported by CloudFront distribution at this moment [179].

This is, since CloudFront can only change HTTP headers, while AppSync real-time subscription operation encodes the authentication (e.g. X-API-KEY or Authorization) and the hostname in its query payload with base64 format. Unlike Query and Mutation operations, CloudFront cannot tweak the encoded information in the query payload, though they are required in the initial handshake for Web Socket communication.

5.5.4 Deployment

It is true that a CDN can mitigate a significant portion of the latency. However, it still causes some delay and because, given the scope of this prototype, the goal is to reproduce an ideal scenario, to provide the best experience possible and to achieve optimal performance, the best option is to deploy the Web-IDE in a server physically closer to Lisbon. Upon further examination, taking advantage of a AWS latency test [180], we concluded that a server located in Europe, more specifically in Paris, would be the best solution to deploy the application, in the interest of reducing latency.

Both the Central System and the Message System were then deployed in Paris, resulting in the latency between the user and the Server-side becoming nearly negligible (approximately 50 milliseconds).

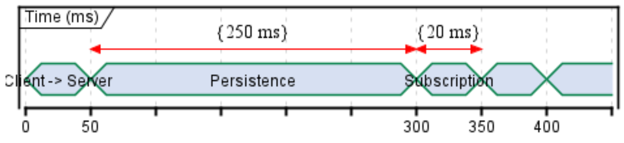


Figure 5.16: Latency of the system (Final Version)

Conclusively, these improvements managed to shrink the overall latency of the updates to the model from approximately 1.7 seconds to 400 milliseconds, an unobtrusive and very acceptable delay. This represents a reduction of latency by 76%, which ultimately results in a very fluent experience, adequate for real-time collaboration.

A further possible improvement that was also considered was to implement geo-replication, that is, to maintain copies of the data in distinct geographical locations.

The first reason why this option would be beneficial is that it reduces the latency between the users and servers [181] where all the data is being stored, meaning that users can benefit from a low latency service, regardless of their location and, as a consequence, the collaboration between users located in different regions is also possible. Furthermore, this is also the added benefit of disaster tolerance, meaning that the data storage has the ability to tolerate any eventual unplanned failure of data centres.

On the other hand, from an engineering point of view, geo-replication adds substantial complexity to the project as replication algorithms need to be implemented. Given the scope of this project, it was decided that it would be reasonable to not implement this particular feature and to restrict the data residency of the graphs to the physical location where the user accesses it from.

5.5.5 CI/CD

The last, but nonetheless important improvement performed on the project was the integration of a Continuous Integration/Continuous Delivery (CI/CD) pipeline. Despite not providing any additional benefit in terms of latency, it is a widely used practice that adds productivity and structural value to the project.

CI/CD is a delivery method that aims to add automation and continuous monitoring to the app development process [182] so that the developers can focus the product implementation rather than wasting time in the merging, testing and deploying process every time a change is added to the project.

The "CI" in CI/CD stands for Continuous Integration. This means that changes are regularly merged to a repository containing the code of a project, and also that the resulting code is automatically built and tested. This is particularly beneficial in a context where multiple developers are working simultaneously in distinct features as it enables an easy coordination process and mitigates potential errors or incompatibilities that most likely lead to a loss of time and money. In summary, “CI” makes it easier to identify potential bugs.

The “CD” in CI/CD refers to Continuous Delivery and/or Continuous Deployment. The difference between these two definitions is that whereas Continuous Delivery means that the changes to an application are automatically tested, released to a repository, and then verified by a team before being deployed, Continuous Deployment automatically releases the changes performed on the project directly into production, where it is used by the users.

Both of these options are dependent on a repository that has the code built and tested, which is provided by the “CI”. In the scope of our project, the term that best describes the pipeline implemented is Continuous Deployment, which allows the integration of changes in the project and deploys them within a question of minutes.

In this case, given that project is already based in the AWS cloud ecosystem, we decided to incorporate this method with the tools available in it.

Namely, by making use of AWS CodeBuild [183], which is a service responsible for the “CI” part of the project, that is, it is used for compiling the source code, running tests and producing software ready to be deployed. The service enables a direct connection to the repository of the source code, which in this case is GitHub. Given the nature of the project, no relevant tests are required and so, this optional phase was not included.

After this, AWS CodeDeploy [184], as the name suggests, automates the software deployment into a compute service. Finally, AWS CodePipeline is an agglomeration service, as seen in figure 5.17, that enables to build a full CI/CD pipeline, based on the services previously mentioned. That is, it allows the automation of the build, test and deploy phases, every time a change is made in the source code of the project.

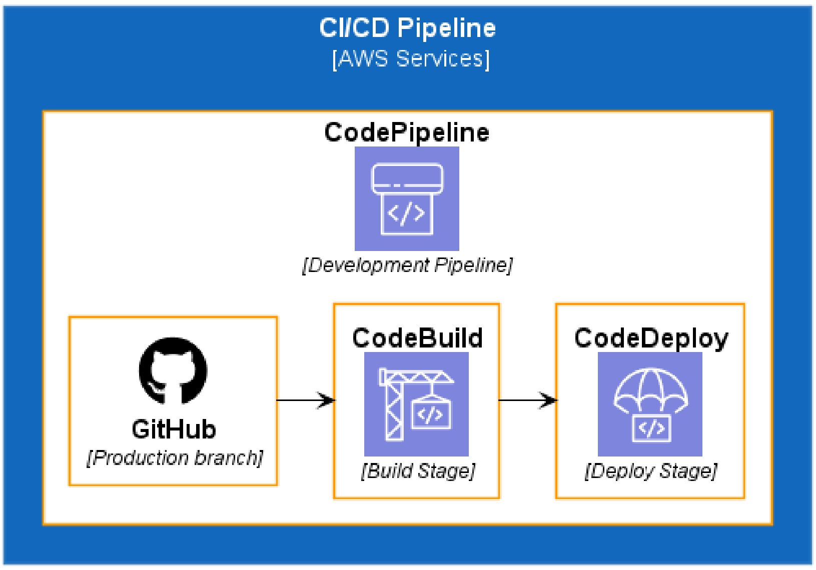


Figure 5.17: CI/CD

To understand, in practical terms, how these components interact with each other it is first important to note that the project repository will be required to have two distinct branches. The development branch, where all alterations and additions are to be integrated and tested, and the production branch.

Once new code is incorporated in the development environment, a simple merge action of the development branch into the production branch will trigger a GitHub notification function that, in turn, is received by AWS CodeBuild, which will then proceed to make a new build of the updated project and, once finished, will order AWS CodeDeploy to re-deploy the latest build.

Therefore, in a matter of seconds, changes made in the project can be incorporated into the production environment, without requiring any effort from the user whatsoever.

6 Evaluation

6.1 Execution

This subsection is used to exhibit the final result of the work developed. Figure 6.1 contains the final version of the POC of the Service Studio IDE, based on the web.

Additionally, a presentation is provided with the intent of providing a palpable demonstration of how the developed tool can be used: Demonstration link.

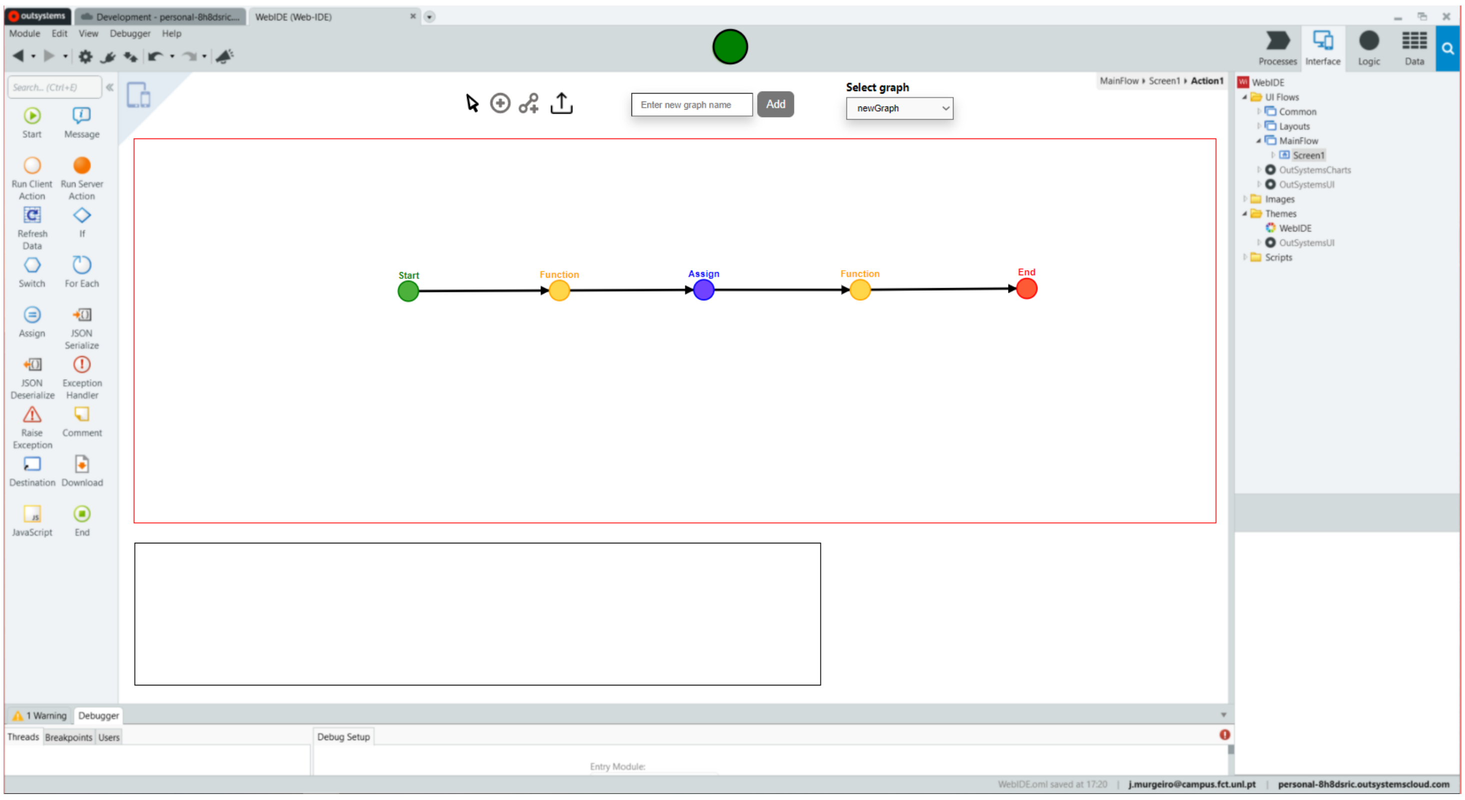


Figure 6.1: Final result of POC (screenshot).

In this video, a brief contextualization of the overall theme of the thesis, as well as the several phases in the development of the proof o concept are presented, followed by a commentated "walk-through" of all functionalities present in the project and the actions that are possible to perform.

6.2 Interviews with stakeholders

The goal of this thesis is to determine the architectural viability of the solution, rather than evaluating the usability of the tool developed. The POC constructed is based on a visual model with functionalities mimicking the ones present in the Service Studio. Nevertheless, these usability aspects are not the center of the project. The architectural solution that functions as a vehicle for us to test and arrive at a conclusion regarding the soundness of the solution.

For this reason, upon deliberation with the coordinators of the thesis, we decided that the best course of action, in order to produce an adequate evaluation process, would be to conduct interviews relevant stakeholders of the Service Studio IDE. This way, it is possible to integrate an external, yet competent evaluation process into the work.

These semi-structured interviews ought to be of qualitative nature, with the purpose of having clear questions, reaching distinct topics, that enable the interviewee to provide their opinion.

The interview process was preceded by a brief presentation of the development of the project, as well as technologies used and architectural decisions.

The questions asked are separated into three distinct categories. The first set of three questions are within the context of the feasibility of the architecture, followed by three other questions regarding the performance of the system and finally two questions concerning scalability, from both the server-side and the client-side. In each of the questions asked, a quick discussion was held that enabled the stakeholders to clear any possible doubt or question, communicate their concerns and express their opinions. The fifth stakeholder present in the table 6.1 is simultaneously an advisor of this thesis.

Stakeholder Name	**Position**	**Code**
Vasco Pessanha	Product Management | App Development	S1
Nuno Maurício	R&D | Phoenix IDE Alpha | Team Lead	S2
Tiago Simões	Product Design - Frontend at R&D	S3
Ricardo Alves	Product Manager @ UX/UI	S4
Carlos Sousa	Head of Product Architecture	S5

Table 6.1: Stakeholders table.

6.2.1 Threats to the validity of the evaluation

Before presenting the questions asked, two aspects regarding the validity of the evaluation ought to be mentioned.

- The evaluation process is based on a reduced number of stakeholders (Table 6.1), consequently, it is necessary to admit that the same set of questions, if presented to different a group, could potentially result in different feedback. However, the stakeholders asked to participate in this evaluation phase have leading roles in the current development of the product and, more specifically, in the Service Studio IDE. Yet, most of them work in different areas, thereby providing complementing points of view.
- One of the stakeholders included in this evaluation section coincides with the coordinator of this thesis. For this reason, it is important to understand that his inclusion should not focus on whether or not he agrees with the decisions taken, given that he participated in them, but rather to provide the unreplaceable insight of a leader in the area of architecture at OutSystems.

For each of the following questions asked, an answer will be provided, based on the general opinion gathered though the interviews, and illustrated via verbatim quotes by the interviewed stakeholders to better characterize their opinion, where appropriate.

Does the architecture of the system fit the Cloud strategy of OutSystems?

"OutSystems has a public strategic partnership with AWS, meaning that AWS is the cloud provider of reference." - S5.

Given that the system architecture was built using exclusively AWS services, combined with a client-side system, the architecture fits well into the Cloud strategy at OutSystems

Do the services used to build the system architecture respect good structural practices?

For the development of the Central system, we decided to use the AWS microservices (see subsection 5.3.1). Microservices enable to build a streamlined development environment where the responsibilities of each service are well segregated, which allows for better fault isolation and greater business agility.

In addition, this paradigm is highly effective not only in terms of high availability, automatic scalability and easy deployment but also in a monetary aspect as the user only has to pay for the resources used.

At the core of the Messaging Container, a streaming pattern (see subsection 5.3.4.1), or event-based pattern, was used to aid the propagation of events produced by the user, when interacting with the visual model. This pattern is beneficial because not only does it allow to collect and analyse the streaming data in real-time, but also guarantees that the events are kept in memory, providing the system with fault tolerance.

Another concept that was fundamental in the construction of the system architecture is graceful degradation (see subsection 5.3.4). By decoupling the services implemented into separate containers, it is possible for them to operate independently and, therefore, have distinct responsibilities and availability, maximizing the system resilience in case of some failure.

Considering that all the previous points were respected while designing and implementing the system architecture, we argue that the services used to build the system architecture respect good structural practices by design.

Are the technologies used adequate considering OutSystems' strategy?

Here, the term technologies, is referring to the coding languages used to implement the project.

The first version of the client-side system was composed by a JavaScript file, where the vast majority of the code is and where all the code relative to the D3.js library can be found, accompanied by a simple HTML and CSS file. Upon the forth phase of the development of the POC, the necessity of integrating a AWS Amplify Client, which is only made available through the implementation of a React project, arrived. Therefore, in order to accomplish this solution, it was necessary to incorporate the first version of the client-side system into a React project, resulting in the final version.

The server-side of the system was implemented using the AWS CDK, benefiting from the Infrastructure as Code approach, facilitated by a custom language provided by AWS. However, the Lambdas used are also implemented using JavaScript.

"JavaScript and React are two core technologies for OutSystems, in which there is a large skill-set of developers.." - S5. For the reason presented in the previous citation, the technologies used are perfectly aligned with OutSystems' technological direction.

From the point of view of feedback from the verification mechanism, is 1.5 milliseconds, on average, acceptable?

The reduced latency time is due to the fact that all the verification process is done locally, at the client-side.

The verification process in the POC consists of a single loop (see subsection 5.2.2) responsible for verifying all inward and outwards connections in each node, which can be seen as a somewhat "empty" scenario, as this process requires very few conditions, when in comparison with a real-life scenario of an OutSystems project.

In the latter, it is possible to have dozens of screens and entities. To delete, for example, an entity, can result in a outburst of errors anywhere in the application. Hence, it is safe to say that the verification process is much heavier than the one present in this POC.

"To correctly address this question, I would like to see a verification process of a real application, as this application is practically empty. Nonetheless, the latency of the feedback is more than enough." - S1.

For this reason, and while having the question in mind, 1.5 milliseconds is a very acceptable latency. However, it is unfair to compare both verification processes as the computing power required to run each one of them is very disproportionate. For the requirements of this POC, the verification mechanism is extremely rapid, conferring performance to the solution. Nevertheless, to build truly comparable scenario and understand the verification latency in a Web-IDE of Service Studio, further semantics as well as numerous other components would have to be added.

"Given the circumstances and the context of the work developed, it's an excellent latency." - S2.

In conclusion, the fact that the use case had to be simplified in order to match the requirements and test the architecture, means that the project is not complex enough to be a rigid parallel to the Service Studio.

In a collaboration scenario, is a latency of 400 milliseconds, on average, from the moment a user performs an alteration, up until the moment it is seen by other users, acceptable?

"I'm OK with the action of moving a node taking 400ms, but what would happen if two users decided to move it simultaneously?" - S4.

"I don't necessarily think that a model of live collaboration, such as pair-programming, would be something that OutSystems' developers would want." - S1.

The real-time collaboration mechanism is part of the requirements determined at the start of the thesis development. The goal here is to demonstrate a viable way of constructing such a mechanism. This, however, does not eliminate the need to understand what is the best policy for the collaboration to be available.

It makes sense that, for example, when a user is altering a specific line of code present in a node, no other user has write access to it, otherwise it could be a disadvantage, instead of an advantage. The same logic can be applied in the action of moving a node.

For this reason, and in order to further complement the real-time collaboration mechanism, a refinement of the possible actions within the real-time context would be necessary, which will also be addressed in the future work subsection, as well as how the events are propagated. In this case, each events is propagated independently, but they could also, for instance, be placed in a queue and then loaded into the model in a group.

"I would say that in a real-time collaboration environment, a delay of up to 1 second, and a maximum of 2 seconds is acceptable, more than that would create a lot of confusion. So yes, the delay is very acceptable." - S2.

In summary, a latency of 400 milliseconds to propagate alterations in the visual model between users with access to the same graph is very acceptable. However, it is crucial to understand that, in this case, this corresponds to the standard model and, therefore, could be subject to alterations to better fit the requirements of a real-life scenario of a web version of the Service Studio.

Is 500 milliseconds an acceptable latency to download a graph from the server-side containing a total of 10 megabytes of information?

It is essential do comprehend that, in this question, the focal point is to understand the difference in latency in downloading a file with a certain amount of information, not necessarily to load it into the visual model.

This is due to the fact that in the Service Studio, the download process will retrieve a zip object. This object will then be extracted and consequently rendered into a substantially bigger OutSystems Markup Language (OML) file, therefore, adding a further step into the process of loading a graph that does not exist in the POC.

However, given that the question is merely focused on the size of the object that is downloaded, a comparison could be made.

"Half a second to open a "real" flow is acceptable. But to make a true comparison, you would have to measure how much time it takes to download a flow of this size, using the Service Studio IDE." - S1.

Upon testing the scenario mentioned in the previous comment, using the Service Studio IDE, a 10 megabytes file was saved to the server-side and, consequently, downloaded. Therefore, it is possible to conclude that the process of downloading a file of 10 megabytes, in the Service Studio, will amount to a total of 4 seconds. Although, this time includes the unzipping process.

With this in mind, it is reasonable to conclude that 500 milliseconds is an acceptable latency to download a graph from the server-side containing a total of 10 megabytes, as it translates to a significant improvement in performance, when comparing to the Service Studio.

Is a system capable of supporting 10 million requests per second, per region, sufficient?

Netflix is capable of supporting up to millions of requests per second [185], and so, as an example point of reference, the questions asked to the stakeholders included the number of 10 million requests per second, to account for any possible peak in requests that could happen in the context of the OutSystems IDE.

As previously mentioned, the server-side of the applications is composed by services provided by AWS. For this reason, the scalability of the part of the architecture is dependent on the scalability of each of the services that comprise it.

All of the services that compose the workflow that the system architecture provides have a threshold, or "soft-limit" that corresponds to the maximum of resources possible to allocate, in order to improve performance. As examples, the "soft-limit" of API Gateway is 10.000 requests per second [186], Lambda is 1000 concurrent executions [187] and Kinesis DataStream is 500 shards [156].

However, these "soft-limits" are only put in place to protect the average user from unintentionally exceeding the normal allocation of resources, which could lead to very expensive monetary consequences. As such, it is possible to request AWS to lift these restrictions and increase the maximum capability of the services. In theory, there is not real "hard-limit" to any of the services, and the limits can always be increased to match the requirements of any given application, as long as the cost is taken into consideration.

It must be said, tough, that in order to be sure that the system can indeed reach high levels of scalability, a proper testing procedure would have to be conducted. However, in the case of the POC, this would not be an simple endeavour and, because it is not part of the requirements of the systems, we decided to not include it.

In conclusion, and also according to the technical support of AWS, the services that compose the system architecture would have no problem in scale in order to sustain up to 10 millions requests per second, as long as their capacity limit is raised. However, this is not a proven conclusion as no testing process was conducted.

Is it acceptable to operate over a visual model containing up to 500 megabytes of information?

"Yes, it is very acceptable, however, I'm not sure the the performance would be the same if the all the information would correspond to millions of nodes." - S2.

With the previous comment in mind, the scenario of having a very large number of nodes in the visual model was also tested. Using the "upload button", various scenarios, with varying number of nodes were tested. We arrived at the conclusion that the visual model is capable of supporting 1000 nodes without damaging the performance, but 10.000 nodes exceeds the limit. This, however, is not an impediment as in a "real-life" scenario, a graph should not surpass 1000 nodes. Hence, the focus of the testing should be in having a common amount of nodes, each containing data, rather than having an unreasonable number, each containing close to no information.

"Be careful because what you are using to reach the 500 megabytes is the payload, but in a real scenario, a simple operation would require a heavy verification process over all the information present in the application." - S1.

It is also important to understand that the value of 500 megabytes was achieved by adding several nodes to the visual model, each containing a "dummy" variable composed by an large String. This, however, is not the same as 500 megabytes OutSystems applications, which would be composed by numerous screens and entities and so, to operate over such a model, as previously mentioned, would require a much heavier verification process, which could impact the overall performance.

"This is completely acceptable as we don't have models with 500 megabytes, that's a very large number." - S3.

"The bigger models I've seen had a total amount of information of around 60/70 megabytes." - S1.

And so, the answer to the question is affirmative, but it is still important to keep in mind that the context of this POC makes it impossible to perform a real and honest comparison with the Service Studio, which has a much more complex nature.

7 Conclusion

OutSystems is a Low-code platform that aims to aid users to develop and manage robust applications. The main component of this product is the Service Studio, which is an IDE that facilitates the development of applications by making use of an intuitive interface.

Currently, there is only available a downloadable and locally installable version of this IDE, which could be disadvantageous going into the future. This is because there is a growing trend of migrating desktop applications to the cloud and providing them via a service through the web, to harness the various advantages of this model.

For this reason, the theme of the thesis was to try and get to a conclusion as to whether it would be feasible to develop a web-based version of the Service Studio IDE.

As a result, this thesis had the objective of developing a prototype that represents a feasible solution for the proposed project. Further elaborated, this includes understanding as to whether the project developed consists of a viable, and scalable, technological architecture that is capable of maintaining acceptable performance.

In order to respond to these points of uncertainty, a POC was developed, which was divided into well defined distinct phases, to ensure a structured, cumulative development process. For each step, a discussion regarding the possible technologies to include is presented, as well as a clarification of architectural decisions that were taken, and a detailed description of relevant details of the solution.

After this, an improvement process of the POC took place intending to enhance the performance of the system and add structural value to its overall architecture. Finally, an evaluation process with experts in the Service Studio was conducted to validate the soundness and feasibility of the solution.

With this being said, it is possible to affirm that the requirements established at the beginning of the thesis were met and that the goals determined were accomplished. Thereby, the prototype developed consists of a feasible solution for the problem presented and that, therefore, OutSystems could, if desirable, adapt the Service Studio into a web-based paradigm without having to lose performance while still maintaining the scalability of the system.

7.1 Contributions

- **Local Persistence inside the browser**. In order to achieve local persistence inside the browser, the tool D3.js was used to implement a visual canvas mimicking the Service Studio Low-code language. After that, a regular JavaScript script was used to construct a data structure capable of storing all information necessary for the visual model within the browser storage, while also storing it in the browser cache, using IndexedDB. In addition, an algorithm was also implemented that is responsible for validating the semantics of the visual models done by the users.
- **Scalable, cloud-based Central System**. For the Central System, an architecture composed of multiple AWS Services was designed. These services can be configured to use as many resources as necessary to need the client's requirements, thus, ensuring the scalability of the whole system. These services were implemented, using good structural practices, to produce a pattern capable of not only propagating all the changes the clients perform over the visual models but also storing and retrieving them accordingly.
- **Real-time collaboration mechanism**. For the real-time collaboration mechanism, the Central System was complemented with the service AWS AppSync that enables the system to not merely create publish and subscribe pattern but also to establish a Web-Socket connection with all the clients, allowing for a fluid collaboration environment, in real-time.
- **Viable technological architecture**. With the previous points in mind, this work included various discussions regarding architectural decisions that contributed to a viable technological solution to build a scalable and performance-driven Web-IDE.

7.2 Future Work

The objectives that were proposed for the development of this thesis have been achieved and, therefore, the main questions regarding this subject have been addressed and answered.

Still, as with any complex problem, there are various additions that, despite not being part of the scope of this thesis, can be investigated and implemented to continue the research process. These topics fall under the area of future work:

- **Synchronization**. One limitation of D3.js is that it is not possible to define the difference between a simple click on a node, and the action of moving a node, in both cases a selecting of the node occurs. Furthermore, To delete a node, it is required to first select it, and then delete it. This triggers two events, an update location and a delete, because the events are asynchronous, in the current visual model, it is possible that, especially if the server-side is deployed in a different region than the user that is accessing it, the delete event arrives first to the IDE of another user, instead of the update. The result is that the node is deleted, and then updated, meaning that it is not visually deleted.

This exemplifies how important a synchronization mechanism would be for a detailed real-time collaboration framework. This detail could be solved by implementing an algorithm such as Lamport's Logical Clock [188].

- **Off-line development**. As previously mentioned, the integration of an off-line development mechanism would further improve the quality of the solution as it would allow users to make use of the platform, if there is no internet connection established. A possible solution for this could be to re-design the current system architecture to take advantage of the off-line development feature the service AWS AppSync provides.
- **Collaborative development**. Through the project developed it has been proven that more than one user can perform changes over the same visual model in real-time. However, this might not correspond to the desired behaviour of the Web-IDE, that is, it might be decided that, for example, only one user can change a node at a given time, to avoid any direct conflict. In other words, the configuration of the concurrency model and collaborative access policies can be detailed and, as such, are part of possible future work.

Bibliography

[1] *DNB Bank Modernizes its Bespoke CRM System 70% Faster Without "Ripping and Replacing" Legacy Banking Systems*. 2021. URL: https://www.outsystems.com/case-studies/dnb-bank-polska-modernizes-crm-system/ (cit. on p. 1).

[2] *Lendr Builds 5 Apps in Under 12 Months with Just 2 Developers*. 2021. URL: https://www.outsystems.com/case-studies/lendr-accelerates-app-dev/ (cit. on p. 1).

[3] G. Sofonea, L. Ciovica, and L. Ciovica. "Web-IDE". In: *Proceedings of the 12th WSEAS international conference on Computers*. 2008, pp. 314–316 (cit. on p. 1).

[4] L. Morgan. *What Is Platform as a Service?* 2019. URL: https://www.outsystems.com/blog/posts/what-is-platform-as-a-service/ (cit. on p. 1).

[5] S. S. Manvi and G. K. Shyam. "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey". In: *Journal of network and computer applications* 41 (2014), pp. 424–440 (cit. on p. 2).

[6] A. N. Ansari et al. "Online C/C++ compiler using cloud computing". In: *2011 International Conference on Multimedia Technology*. IEEE. 2011, pp. 3591–3594 (cit. on p. 2).

[7] K. M. Garawad and S. Girish. *.NET Compiler using Cloud Computing*. 2014 (cit. on p. 2).

[8] L. M. Gadhikar et al. "Implementation of Browser Based IDE to Code in The Cloud". In: *International Journal of Advances in Engineering & Technology* 5.1 (2012), p. 336 (cit. on p. 2).

[9] *Google Docs*. 2021. URL: https://www.google.com/docs/about/ (cit. on pp. 2, 13).

[10] J. Jenkins, E. Brannock, and S. Dekhane. "JavaWIDE: innovation in an online IDE: tutorial presentation". In: *Journal of Computing Sciences in Colleges* 25.5 (2010), pp. 102–104 (cit. on p. 2).

[11] M. Dutta, K. K. Sethi, and A. Khatri. “Web based integrated development environment”. In: *International Journal of Innovative Technology and Exploring Engineering* 3.10 (2014), pp. 56–60 (cit. on p. 2).

[12] R. Hegde and K. Karunakara. “Improved Interaction in Web-Based Cloud IDE”. In: *Int. J. Innov. Technol. Explor. Eng.* 4.4 (2014), pp. 16–20 (cit. on p. 2).

[13] A. Yousif, M. Farouk, and M. B. Bashir. “A Cloud Based Framework for Platform as a Service”. In: *2015 International Conference on Cloud Computing (ICCC)*. IEEE. 2015, pp. 1–5 (cit. on p. 2).

[14] B. Yulianto, H. Prabowo, and R. Kosala. “Comparing the effectiveness of digital contents for improving learning outcomes in computer programming for autodidact students”. In: *Journal of e-Learning and Knowledge Society* 12.1 (Jan. 2016). ISSN: 1826-6223. URL: https://www.learntechlib.org/p/171429 (cit. on p. 3).

[15] S. Mohammed and A. M. Abdelazziz. “WIDE an interactive Web integrated development environment to practice C programming in distance education”. In: *2013 1st International Conference of the Portuguese Society for Engineering Education (CISPEE)*. IEEE. 2013, pp. 1–6 (cit. on p. 3).

[16] *WebAssembly*. 2021. URL: https://webassembly.org/ (cit. on pp. 5, 14).

[17] *Deploying OutSystems in the cloud*. 2021. URL: https://www.outsystems.com/evaluation-guide/deploying-outsystems-in-the-cloud/ (cit. on p. 5).

[18] OutSystems. *OutSystems Evaluation Guide*. 2021. URL: https://www.outsystems.com/evaluation-guide/why-outsystems/ (cit. on p. 5).

[19] M. Fowler, J. Highsmith, et al. “The agile manifesto”. In: *Software Development* 9.8 (2001), pp. 28–35 (cit. on p. 5).

[20] J. Martin. *Application Development without Programmers*. USA: Prentice Hall PTR, 1982. ISBN: 0130389439 (cit. on p. 6).

[21] *Fourth-generation programming language*. 2021. URL: https://en.wikipedia.org/wiki/Fourth-generation_programming_language#cite_note-6 (cit. on p. 6).

[22] *Low-code development platform*. 2021. URL: https://en.wikipedia.org/wiki/Low-code_development_platform (cit. on p. 6).

[23] *The Best Low-Code Development Platforms*. 2018. URL: https://www.pcmag.com/picks/the-best-low-code-development-platforms (cit. on p. 6).

[24] K. Schwertner. “Digital transformation of business”. In: *Trakia Journal of Sciences* 15.1 (2017), pp. 388–393 (cit. on p. 6).

[25] S. Mäkinen et al. "Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises". In: *Information and Software Technology* 80 (2016), pp. 175–194. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2016.09.001. URL: https://www.sciencedirect.com/science/article/pii/S0950584916301434 (cit. on p. 6).

[26] OutSystems. *OutSystems Evaluation Guide*. 2021. URL: https://www.outsystems.com/evaluation-guide/architecture/ (cit. on p. 6).

[27] *OutSystems Overview*. 2021. URL: https://www.outsystems.com/training/lesson/2183/components-and-tools (cit. on p. 7).

[28] *OutSystems development and management tools*. 2021. URL: https://www.outsystems.com/evaluation-guide/development-and-management-tools/ (cit. on p. 8).

[29] *.NET Free. Cross-platform. Open source*. 2021. URL: https://dotnet.microsoft.com/ (cit. on p. 8).

[30] *Get started with WPF*. 2018. URL: https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2019 (cit. on p. 8).

[31] *React A JavaScript library for building user interfaces*. 2021. URL: https://reactjs.org/ (cit. on p. 8).

[32] *The modern web developer's platform*. 2021. URL: https://angular.io/ (cit. on p. 8).

[33] *Avalonia UI*. 2021. URL: https://avaloniaui.net/ (cit. on pp. 8, 16).

[34] *Typed JavaScript at Any Scale*. 2021. URL: https://www.typescriptlang.org/ (cit. on p. 8).

[35] *Download .NET Framework 4.7.2*. 2021. URL: https://dotnet.microsoft.com/download/dotnet-framework/net472 (cit. on p. 8).

[36] *Download .NET Downloads for .NET Framework and .NET Core, including ASP.NET and ASP.NET Core*. 2021. URL: https://dotnet.microsoft.com/download (cit. on p. 8).

[37] *Service Studio Overview*. 2020. URL: https://success.outsystems.com/Documentation/11/Getting_started/Service_Studio_Overview (cit. on p. 9).

[38] C. BUSINESS. *Reducing Business Friction to Deliver Better Experiencesb*. 2019. URL: https://business.comcast.com/community/browse-all/details/reducing-business-friction-to-deliver-better-experiences (cit. on p. 10).

[39] *Uber*. 2021. URL: https://www.uber.com/pt/en/deliver/ (cit. on p. 10).

[40] E. Europe. *How Uber Used a Simplified Business Model to Disrupt the Taxi Industry*. 2021. URL: https://www.entrepreneur.com/article/286683 (cit. on p. 11).

[41] M. Experiments. *B2B Lead Gen: A/B split test helps increase quote requests 262%*. 2012. URL: https://marketingexperiments.com/lead-generation/b2b-increase-quote-requests (cit. on p. 11).

[42] *Swift The powerful programming language that is also easy to learn.* 2021. URL: https://developer.apple.com/swift/ (cit. on p. 11).

[43] *A modern programming language that makes developers happier.* 2021. URL: https://kotlinlang.org/ (cit. on p. 11).

[44] H. Heitkötter, S. Hanschke, and T. A. Majchrzak. "Evaluating cross-platform development approaches for mobile applications". In: *International Conference on Web Information Systems and Technologies*. Springer. 2012, pp. 120–138 (cit. on p. 11).

[45] C. Scott D'Ambra. *What is Cross Platform Development?* 2018. URL: https://www.cleart.com/what-is-cross-platform-development.html (cit. on p. 11).

[46] J. Wood. *The Shortage of Software Engineers in 2020*. 2020. URL: https://www.kms-technology.com/blog/the-shortage-of-software-engineers-in-2020.html (cit. on p. 12).

[47] *What's a Universal Windows Platform (UWP) app?* 2020. URL: https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide (cit. on p. 12).

[48] *React Native Learn once, write anywhere.* 2021. URL: https://reactnative.dev/ (cit. on p. 12).

[49] StatCounter. *Desktop Operating System Market Share Worldwide*. 2021. URL: https://gs.statcounter.com/os-market-share/desktop/worldwide (cit. on p. 12).

[50] S. O. D. Survey. *Developers' Primary Operating Systems*. 2020. URL: https://insights.stackoverflow.com/survey/2020#overview (cit. on pp. 12, 13).

[51] S. Emani et al. "Web Based 'C' IDE: Approach". In: *Department of Information Technology, STES's Sinhgad Academy of Engineering* (2012) (cit. on p. 12).

[52] F. Lanubile et al. "Collaboration Tools for Global Software Engineering". In: *IEEE Software* 27.2 (2010), pp. 52–55. DOI: 10.1109/MS.2010.39 (cit. on p. 13).

[53] J. Whitehead. "Collaboration in Software Engineering: A Roadmap". In: *Future of Software Engineering (FOSE '07)*. 2007, pp. 214–225. DOI: 10.1109/FOSE.2007.4 (cit. on p. 13).

[54] *The Importance of Implementing Online Collaboration Software*. 2020. URL: https://blog.runrun.it/en/online-collaboration-software/#:~:text=With%5C%20the%5C%20use%5C%20of%5C%20collaborative,save%5C%20time%5C%2C%5C%20and%5C%20boost%5C%20productivity (cit. on p. 13).

[55] B. Commerce. *What is open source, and why is it important?* 2020. URL: https://www.bigcommerce.com/ecommerce-answers/what-open-source-and-why-it-important/#:~:text=Open%5C%20source%5C%20licensing%5C%20encourages%5C%20innovation,for%5C%20the%5C%20past%5C%20few%5C%20decades (cit. on p. 13).

[56] *Google Vulnerability Reward Program (VRP) Rules.* 2020. URL: https://www.google.com/about/appsecurity/reward-program/ (cit. on p. 13).

[57] *GitHub.* 2020. URL: https://github.com/ (cit. on p. 13).

[58] P. B. Carr and G. M. Walton. "Cues of working together fuel intrinsic motivation". In: *Journal of Experimental Social Psychology* 53 (2014), pp. 169–184. ISSN: 0022-1031. DOI: https://doi.org/10.1016/j.jesp.2014.03.015. URL: http://www.sciencedirect.com/science/article/pii/S0022103114000420 (cit. on p. 13).

[59] A. Gaskell. *New Study Finds That Collaboration Drives Workplace Performance.* 2017. URL: https://www.forbes.com/sites/adigaskell/2017/06/22/new-study-finds-that-collaboration-drives-workplace-performance/?sh=6cbc485e3d02 (cit. on p. 13).

[60] A. van Deursen et al. "Adinda: A Knowledgeable, Browser-Based IDE". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2.* ICSE '10. Cape Town, South Africa: Association for Computing Machinery, 2010, pp. 203–206. ISBN: 9781605587196. DOI: 10.1145/1810295.1810330. URL: https://doi.org/10.1145/1810295.1810330 (cit. on p. 14).

[61] M. Goldman, G. Little, and R. C. Miller. "Collabode: Collaborative Coding in the Browser". In: *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering.* CHASE '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 65–68. ISBN: 9781450305761. DOI: 10.1145/1984642.1984658. URL: https://doi.org/10.1145/1984642.1984658 (cit. on p. 14).

[62] J. D. Herbsleb. "Global Software Engineering: The Future of Socio-technical Coordination". In: *Future of Software Engineering (FOSE '07).* 2007, pp. 188–198. DOI: 10.1109/FOSE.2007.11 (cit. on p. 14).

[63] A. Haas et al. "Bringing the web up to speed with WebAssembly". In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation.* 2017, pp. 185–200 (cit. on p. 14).

[64] M. W. Docs. *WebAssembly.* 2020. URL: https://developer.mozilla.org/en-US/docs/WebAssembly (cit. on p. 14).

[65] *C++.* 2021. URL: https://isocpp.org/ (cit. on p. 14).

[66] *Rust A language empowering everyone to build reliable and efficient software.* 2021. URL: https://www.rust-lang.org/ (cit. on p. 14).

[67] *JavaScript.* 2021. URL: https://www.javascript.com/ (cit. on pp. 14, 28).

[68] L. Clark. *Creating and working with WebAssembly modules.* 2017. URL: https://hacks.mozilla.org/2017/02/creating-and-working-with-webassembly-modules/ (cit. on p. 15).

[69] *Emscripten.* 2021. URL: https://emscripten.org/ (cit. on p. 15).

[70] A. Zakai. "Emscripten: An LLVM-to-JavaScript Compiler". In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion.* OOPSLA '11. Portland, Oregon, USA: Association for Computing Machinery, 2011, pp. 301–312. ISBN: 9781450309424. DOI: 10.1145/2048147.2048224. URL: https://doi.org/10.1145/2048147.2048224 (cit. on p. 15).

[71] *WebAssembly for Web Developers (Google I/O '19).* 2019. URL: https://www.youtube.com/watch?v=njt-Qzw0mVY&ab_channel=GoogleChromeDevelopers (cit. on p. 15).

[72] C. Findlay. *Cross-Platform C# UI Technologies Part 2.* 2020. URL: https://christianfindlay.com/2020/06/24/csharp-crossplatform-2/ (cit. on p. 15).

[73] *Uno Platform.* 2021. URL: https://platform.uno/ (cit. on p. 16).

[74] *XAML overview (WPF .NET).* 2020. URL: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/fundamentals/xaml?view=netdesktop-5.0 (cit. on p. 16).

[75] *MAUI.* 2021. URL: https://visualstudiomagazine.com/articles/2020/05/19/maui.aspx (cit. on p. 16).

[76] *Xamarin.Forms An open-source framework for building iOS, Android, and Windows apps.* 2021. URL: https://dotnet.microsoft.com/apps/xamarin/xamarin-forms (cit. on p. 16).

[77] *Blazor.* 2021. URL: https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor (cit. on p. 16).

[78] *Vue, The Progressive JavaScript Framework.* 2021. URL: https://vuejs.org/ (cit. on p. 17).

[79] *W3C HTML.* 2021. URL: https://www.w3.org/html/s (cit. on p. 17).

[80] P. Chrobak. "Implementation of Virtual Desktop Infrastructure in academic laboratories". In: *2014 Federated Conference on Computer Science and Information Systems.* IEEE. 2014, pp. 1139–1146 (cit. on p. 17).

[81] *What is VDI (virtual desktop infrastructure)?* 2021. URL: https://www.citrix.com/glossary/vdi.html (cit. on p. 17).

[82] *What is VDI?* 2021. URL: https://subscription.packtpub.com/book/virtualization_and_cloud/9781789802375/1/ch01lvl1sec10/what-is-vdi (cit. on p. 18).

[83] *What is VDI (Virtual Desktop Infrastructure)?* 2021. URL: https://www.vmware.com/topics/glossary/content/virtual-desktop-infrastructure-vdi (cit. on p. 18).

[84] M. Rouse. *What is virtual desktop infrastructure? VDI explained.* 2020. URL: https://searchvirtualdesktop.techtarget.com/definition/virtual-desktop-infrastructure-VDI (cit. on p. 19).

[85] *Amazon Web Services.* 2021. URL: https://aws.amazon.com/ (cit. on p. 19).

[86] *Citrix Virtual Apps and Desktops.* 2021. URL: https://www.citrix.com/solutions/digital-workspace/virtualization-vdi.html (cit. on p. 19).

[87] *Windows Virtual Desktop.* 2021. URL: https://azure.microsoft.com/en-us/services/virtual-desktop/ (cit. on p. 19).

[88] *VMware.* 2021. URL: https://www.vmware.com/ (cit. on p. 19).

[89] *C4 Model.* 2021. URL: https://c4model.com/ (cit. on pp. 19, 20).

[90] *What is Unified Modeling Language (UML)?* 2021. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#:~:text=UML%5C%2C%5C%20short%5C%20for%5C%20Unified%5C%20Modeling,business%5C%20modeling%5C%20and%5C%20other%5C%20non%5C%2D (cit. on p. 19).

[91] *AUTODESK.* 2021. URL: https://www.autodesk.com/ (cit. on p. 21).

[92] *AUTOCAD - CAD software to design anything—with time-saving toolsets, web, and mobile apps.* 2021. URL: https://www.autodesk.com/products/autocad/overview?term=1-YEAR&support=null (cit. on p. 21).

[93] K. Cheung. *Autocad Webassembly: Moving A 30 Year Code Base To The Web.* 2021. URL: https://www.infoq.com/presentations/autocad-webassembly (cit. on p. 21).

[94] *Getting started with Flash development.* 2010. URL: https://www.i-programmer.info/programming/other-languages/1604-getting-started-with-flash-development.html (cit. on p. 21).

[95] *GWT - Productivity for developers,performance for users.* 2021. URL: http://www.gwtproject.org/ (cit. on p. 21).

[96] *Google Chrome.* 2021. URL: https://www.google.com/chrome/ (cit. on p. 22).

[97] *Google Drive.* 2021. URL: https://www.google.com/intl/en_in/drive/ (cit. on p. 22).

[98] A. Chakraborty. *System Design Analysis of Google Drive*. 2020. URL: https://towardsdatascience.com/system-design-analysis-of-google-drive-ca3408f22ed3 (cit. on pp. 22, 23).

[99] *APACHE KAFKA - More than 80% of all Fortune 100 companies trust, and use Kafka*. 2021. URL: https://kafka.apache.org/ (cit. on p. 23).

[100] *RabbitMQ*. 2021. URL: https://www.rabbitmq.com/ (cit. on p. 23).

[101] S. Wodinsky. *Google Drive is about to hit 1 billion users*. 2018. URL: https://www.theverge.com/2018/7/25/17613442/google-drive-one-billion-users#:~:text=Google%5C%20Drive%5C%2C%5C%20the%5C%20company's%5C%20flagship,by%5C%201%5C%20billion%5C%20people%5C%20worldwide. (cit. on p. 23).

[102] L. Lamport. "Time, clocks, and the ordering of events in a distributed system". In: *Concurrency: the Works of Leslie Lamport*. 2019, pp. 179–196 (cit. on p. 24).

[103] C. Sun and C. Ellis. "Operational transformation in real-time group editors: issues, algorithms, and achievements". In: *Proceedings of the 1998 ACM conference on Computer supported cooperative work*. 1998, pp. 59–68 (cit. on p. 24).

[104] D. Sun et al. "Operational transformation for collaborative word processing". In: *Proceedings of the 2004 ACM conference on Computer supported cooperative work*. 2004, pp. 437–446 (cit. on p. 24).

[105] D. Sun and C. Sun. "Context-based operational transformation in distributed collaborative editing systems". In: *IEEE Transactions on Parallel and Distributed Systems* 20.10 (2008), pp. 1454–1470 (cit. on p. 24).

[106] M. Shapiro et al. "Conflict-free replicated data types". In: *Symposium on Self-Stabilizing Systems*. Springer. 2011, pp. 386–400 (cit. on p. 24).

[107] N. Preguiça et al. "A commutative replicated data type for cooperative editing". In: *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE. 2009, pp. 395–403 (cit. on pp. 24, 25).

[108] N. Preguiça. "Conflict-free replicated data types: An overview". In: *arXiv preprint arXiv:1806.10254* (2018) (cit. on p. 24).

[109] C. A. Ellis and S. J. Gibbs. "Concurrency control in groupware systems". In: *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*. 1989, pp. 399–407 (cit. on p. 24).

[110] *Google Wave: Live collaborative editing*. 2009. URL: https://www.youtube.com/watch?v=3ykZYKCK7AM&hd=1&ab_channel=Google (cit. on p. 24).

[111] C. Boelmann et al. "Application-Level Determinism in Distributed Systems". In: Dec. 2016, pp. 989–998. DOI: 10.1109/ICPADS.2016.0132 (cit. on p. 25).

[112] S. Agarwal. *How Real-Time Collaborative Editors work?* 2017. URL: https://www.srijanagarwal.me/2017/04/collaborative-editing/ (cit. on p. 24).

[113] *InfoQ - CRDTs and the Quest for Distributed Consistency*. 2021. URL: https://www.youtube.com/watch?v=B5NULPSiOGw&ab_channel=InfoQ (cit. on p. 25).

[114] *Atom text editor*. 2021. URL: https://atom.io/ (cit. on p. 25).

[115] B. Sypytkowski. *Operation-based CRDTs: arrays (part 1)*. 2020. URL: https://bartoszsypytkowski.com/operation-based-crdts-arrays-1/ (cit. on p. 26).

[116] *Power Platform - Adapt. Now more than ever*. 2021. URL: https://powerplatform.microsoft.com/en-us/ (cit. on p. 26).

[117] *Power Apps - The world needs great solutions. Build yours faster*. 2021. URL: https://powerapps.microsoft.com/en-us/ (cit. on p. 27).

[118] J. Perry. *What is PowerApps and how can I use it?* 2019. URL: https://www.contentformula.com/blog/what-is-powerapps-and-how-can-i-use-it/ (cit. on p. 27).

[119] *Excel*. 2021. URL: https://www.microsoft.com/en-us/microsoft-365/excel (cit. on p. 27).

[120] *Share Point*. 2021. URL: https://www.microsoft.com/en-us/microsoft-365/sharepoint/collaboration (cit. on p. 27).

[121] J. Karnes. *Microsoft PowerApps: What is it? What does it do? Is it easy to use?* 2017. URL: https://centricconsulting.com/blog/microsoft-powerapps-introduction_portal/ (cit. on p. 27).

[122] *Example of model-oriented applications*. 2020. URL: https://docs.microsoft.com/pt-pt/powerapps/maker/model-driven-apps/overview-model-driven-samples (cit. on p. 28).

[123] *On Cloud9: Lambda Development in AWS's cloud-based IDE*. 2021. URL: https://aws.amazon.com/lambda/ (cit. on p. 28).

[124] B. Cutler. *On Cloud9: Lambda Development in AWS's cloud-based IDE*. 2018. URL: https://medium.com/slalom-technology/on-cloud9-lambda-development-in-awss-cloud-based-ide-937654c1a13a (cit. on p. 28).

[125] M. Villamizar et al. "Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures". In: *Service Oriented Computing and Applications* 11.2 (2017), pp. 233–247 (cit. on p. 28).

[126] M. Villamizar et al. "Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures". In: *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE. 2016, pp. 179–182 (cit. on p. 28).

[127] *AWS Cloud9 - A cloud IDE for writing, running, and debugging code*. 2021. URL: https://aws.amazon.com/cloud9/ (cit. on pp. 28, 29).

[128] J. MSV. *The Master Plan Behind Amazon's Acquisition of Cloud9 IDE*. 2016. URL: https://www.forbes.com/sites/janakirammsv/2016/07/18/the-master-plan-behind-amazons-acquisition-of-cloud9-ide/?sh=4b588d5d72c1 (cit. on p. 28).

[129] *Python*. 2021. URL: https://www.python.org/ (cit. on p. 28).

[130] L. Ciortea et al. "Cloud9: A software testing service". In: *ACM SIGOPS Operating Systems Review* 43.4 (2010), pp. 5–10 (cit. on p. 28).

[131] T. Blazeclan. *AWS Cloud9: All You Need to Know*. 2020. URL: https://www.blazeclan.com/blog/aws-cloud9-all-you-need-to-know/ (cit. on p. 29).

[132] *Visual Studio Code*. 2021. URL: https://code.visualstudio.com/ (cit. on pp. 29, 31).

[133] *Microsoft Azure*. 2021. URL: https://azure.microsoft.com/en-us/ (cit. on p. 29).

[134] A. Wright. "Ready for a Web OS?" In: *Communications of the ACM* 52.12 (2009), pp. 16–17 (cit. on p. 30).

[135] *Office 365*. 2021. URL: https://www.office.com/ (cit. on p. 30).

[136] *What is Google Chrome Operating System?* 2020. URL: https://www.geeksforgeeks.org/what-is-google-chrome-operating-system/ (cit. on p. 30).

[137] N. Chandler. *How the Google Chrome OS Works*. 2020. URL: https://computer.howstuffworks.com/google-chrome-os.htm (cit. on p. 30).

[138] D. Tweney. *Google Chrome OS: Ditch Your Hard Drives, the Future Is the Web*. 2009. URL: https://www.wired.com/2009/11/google-chrome-os-ditch-your-hard-drives-the-future-is-the-web/ (cit. on p. 30).

[139] J. Lautamäki et al. "CoRED: Browser-Based Collaborative Real-Time Editor for Java Web Applications". In: *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*. CSCW '12. Seattle, Washington, USA: Association for Computing Machinery, 2012, pp. 1307–1316. ISBN: 9781450310864. DOI: 10.1145/2145204.2145399. URL: https://doi.org/10.1145/2145204.2145399 (cit. on p. 31).

[140] *Eclipse IDE*. 2021. URL: https://www.eclipse.org/ide/ (cit. on p. 31).

[141] L. C. Kats et al. "Software Development Environments on the Web: A Research Agenda". In: *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. New York, NY, USA: Association for Computing Machinery, 2012, pp. 99–116. ISBN: 9781450315623. DOI: 10.1145/2384592.2384603. URL: https://doi.org/10.1145/2384592.2384603 (cit. on p. 31).

[142] L. Wu et al. “CEclipse: An Online IDE for Programing in the Cloud”. In: *2011 IEEE World Congress on Services*. 2011, pp. 45–52. DOI: 10.1109/SERVICES.2011.74 (cit. on p. 31).

[143] T. Gaikwad et al. “Web Based IDE”. In: *International Journal of Research in Information Technology (IJRIT)* ISSN 2001-5569 (2014) (cit. on p. 32).

[144] M. Bostock. *Data-Driven Documents*. 2020. URL: https://d3js.org (cit. on p. 35).

[145] Wikipedia. *D3.js*. 2021. URL: https://en.wikipedia.org/wiki/D3.js (cit. on p. 35).

[146] M. Bostock. *d3js.org*. 2020. URL: https://observablehq.com/@d3/gallery (cit. on p. 35).

[147] N. Obaseki. *localStorage in JavaScript: A complete guide*. 2020. URL: https://blog.logrocket.com/localstorage-javascript-complete-guide/#sessionstoragevslocalstorage (cit. on p. 44).

[148] *HTML Web Storage API*. 2021. URL: https://www.w3schools.com/html/html5_webstorage.asp (cit. on p. 44).

[149] Y. Chavan. *Everything you need to know about HTML5 local storage and session storage*. 2019. URL: https://javascript.plainenglish.io/everything-you-need-to-know-about-html5-local-storage-and-session-storage-479c63415c0a (cit. on p. 44).

[150] *Web SQL Database*. 2010. URL: https://www.w3.org/TR/webdatabase/ (cit. on p. 44).

[151] *IndexedDB*. 2021. URL: https://developer.mozilla.org/pt-BR/docs/Web/API/IndexedDB_API (cit. on p. 45).

[152] A. Zlatkov. *How JavaScript works: Storage engines + how to choose the proper storage API*. 2018. URL: https://blog.sessionstack.com/how-javascript-works-storage-engines-how-to-choose-the-proper-storage-api-da50879ef576 (cit. on p. 45).

[153] S. Mathew and J. Varia. “Overview of amazon web services”. In: *Amazon Whitepapers* (2014) (cit. on p. 46).

[154] *AWS Cloud Development Kit*. 2021. URL: https://aws.amazon.com/cdk/ (cit. on p. 46).

[155] *Amazon API Gateway*. 2021. URL: https://aws.amazon.com/pt/api-gateway/ (cit. on p. 48).

[156] *Amazon Kinesis Data Streams*. 2021. URL: https://aws.amazon.com/kinesis/data-streams/ (cit. on pp. 48, 81).

[157] *AWS Lambda*. 2021. URL: https://aws.amazon.com/lambda/ (cit. on p. 49).

[158] *Amazon DynamoDB*. 2021. URL: https://aws.amazon.com/dynamodb/ (cit. on p. 50).

[159] G. DeCandia et al. "Dynamo: Amazon's highly available key-value store". In: *ACM SIGOPS operating systems review* 41.6 (2007), pp. 205–220 (cit. on p. 50).

[160] C. Strauch, U.-L. S. Sites, and W. Kriha. "NoSQL databases". In: *Lecture Notes, Stuttgart Media University* 20 (2011), p. 24 (cit. on p. 50).

[161] S. Gilbert and N. Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". In: *Acm Sigact News* 33.2 (2002), pp. 51–59 (cit. on p. 50).

[162] *Choosing the Right DynamoDB Partition Key*. 2021. URL: https://aws.amazon.com/blogs/database/choosing-the-right-dynamodb-partition-key/ (cit. on p. 51).

[163] *graceful degradation*. 2021. URL: https://www.pcmag.com/encyclopedia/term/graceful-degradation (cit. on p. 51).

[164] *AWS Streaming Data Solution for Amazon Kinesis*. 2021. URL: https://aws.amazon.com/solutions/implementations/aws-streaming-data-solution-for-amazon-kinesis/?nc1=h_ls (cit. on p. 51).

[165] *AWS Solutions Implementations, Implementation Guide, Option 1*. 2021. URL: https://docs.aws.amazon.com/solutions/latest/aws-streaming-data-solution-for-amazon-kinesis/option1-api-kds-lambda.html (cit. on p. 52).

[166] X. Lefèvre. *Asynchronous Client Interaction in AWS Serverless: Polling, WebSocket, SSE or AppSync?* 2020. URL: https://medium.com/serverless-transformation/asynchronous-client-interaction-in-aws-serverless-polling-websocket-server-sent-events-or-acf10167cc67 (cit. on pp. 58–60).

[167] *Comet (programming)*. 2021. URL: https://en.wikipedia.org/wiki/Comet_(programming) (cit. on p. 58).

[168] *Using server-sent events*. 2021. URL: https://developer.mozilla.org/pt-BR/docs/Web/API/Server-sent_events/Using_server-sent_events (cit. on p. 59).

[169] Ø. R. Tangen. "Real-Time Web with WebSocket". MA thesis. 2015 (cit. on p. 59).

[170] *The WebSocket API (WebSockets)*. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (cit. on p. 60).

[171] *AWS AppSync Features*. 2021. URL: https://aws.amazon.com/appsync/ (cit. on p. 61).

[172] *A query language for your API*. 2021. URL: https://graphql.org (cit. on p. 61).

[173] *AWS AppSync Features*. 2021. URL: https://aws.amazon.com/appsync/product-details/ (cit. on p. 62).

[174] *AWS Amplify*. 2021. URL: https://aws.amazon.com/amplify/ (cit. on p. 63).

[175] F. D. Ed Lima. *AWS AppSync offline reference architecture – powered by the Amplify DataStore*. 2020. URL: https://aws.amazon.com/blogs/mobile/aws-appsync-offline-reference-architecture/ (cit. on p. 70).

[176] *Amazon CloudFront*. 2021. URL: https://aws.amazon.com/cloudfront/ (cit. on p. 71).

[177] G. Peng. "CDN: Content distribution network". In: *arXiv preprint cs/0411069* (2004) (cit. on p. 71).

[178] *Choose an endpoint type to set up for an API Gateway API*. 2021. URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-endpoint-types.html (cit. on p. 72).

[179] R. D. Adrian Hall. *Use a custom domain with AWS AppSync, Amazon CloudFront, and Amazon Route 53e*. 2018. URL: https://aws.amazon.com/blogs/mobile/use-a-custom-domain-with-aws-appsync-amazon-cloudfront-and-amazon-route-53/ (cit. on p. 72).

[180] *AWS latency test*. 2021. URL: https://ping.psa.fun (cit. on p. 72).

[181] V. Balegas et al. "Geo-replication: Fast if possible, consistent if necessary". In: *Bulletin of the Technical Committee on Data Engineering* 39.1 (2016), p. 12 (cit. on p. 73).

[182] *What is CI/CD?* 2018. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd#overview (cit. on p. 73).

[183] *AWS CodeBuild*. 2021. URL: https://aws.amazon.com/codebuild/ (cit. on p. 74).

[184] *AWS CodeDeploy*. 2021. URL: https://aws.amazon.com/codedeploy/ (cit. on p. 74).

[185] B. Ganesan. *How Netflix Handles Up To 8 Million Events Per Second*. 2015. URL: https://www.slideshare.net/AmazonWebServices/bdt318-how-netflix-handles-up-to-8-million-events-per-second (cit. on p. 81).

[186] *Amazon API Gateway quotas and important notes*. 2021. URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html (cit. on p. 81).

[187] *Lambda quotas*. 2021. URL: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html (cit. on p. 81).

[188] *Lamport Clock*. 2021. URL: https://pt.wikipedia.org/wiki/Rel%C3%B3gios_de_Lamport (cit. on p. 85).