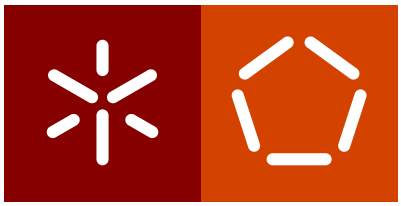


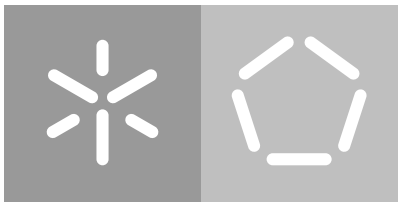
Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ricardo Ribeiro Ferreira

JxAppDev Framework for Hybrid Applications

Hybrid Desktop/Web Application Development

October 2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ricardo Ribeiro Ferreira

JxAppDev Framework for Hybrid Applications

Hybrid Desktop/Web Application Development

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
Rui Couto
José Creissac Campos

October 2018

ACKNOWLEDGEMENTS

This work was only possible due to the people that have been supportive and kind to guide me during this journey. Here I leave my gratitude for everyone that has contributed to this thesis.

I would like to thank my supervisors which have always been available to help me when needed and to guide me through my work.

To Professor José Creissac for the opportunity of working with him, his availability and all the support.

To Rui Couto for all the support and guidance when writing this document and providing me with a proper path to achieve our goals, many thanks.

Finally, I must express my very profound gratitude to my parents and specially to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

ABSTRACT

In modern days, it becomes more and more common for software solutions to focus on mobile and web technologies, therefore the current desktop market has been shrinking. Due to the big impact that web technologies are having on the market and user's daily basis it has become impossible for developers to neglect this evolution.

Nevertheless, in some cases it is difficult to justify the development of some web applications since the benefits are to small and the costs to high.

Due to this problem and some other small inconveniences, there are some emerging technologies that try to close the gap between desktop and web applications by trying to combine the best of both worlds.

There are some well-known technologies such as Java Applets, which are mainly Java applications that can be executed on the browser. Even though these technologies are very interesting and in some specific cases very useful, companies avoid taking this path since this kind of software applications raise some problems, which are making sure that these new technologies are abandoned and forgotten (e.g. some security problems with regard to plugin installation).

With this project, we intend to create a Framework, which main goal is to ease the hybrid application development.

This framework allows users to develop native Java SE applications, that can be accessed as normal Desktop applications, but at the same time it is possible to access the same content through a regular Web Browser, using common well-known technologies such as HTML, JavaScript and CSS.

With this solution, it is possible to avoid high costs on Web application development, and avoid other small problems such as security problems when installing plugins that can be found in the current existing solutions.

This way it is possible to develop a single Desktop application that is reusable on the Browser if needed.

The idea is not to allow the user to create a new application that can be accessed on both platforms, but on the contrary it aims Java applications that have already been developed or that will be developed with no intention of making them accessible on the Web, but at some point the urge to port the application appears and the user won't need to rebuild everything from scratch, but he will simply need to invest some time developing the new User Interface for the Web version that he wants to provide.

RESUMO

Nos dias de hoje é cada vez mais comum as soluções de software que se encontram no mercado serem feitas à volta de tecnologias mobile ou web, o que tem criado uma diminuição no mercado de aplicações nativas desktop. Devido ao grande impacto que as tecnologias web têm tido no mercado e nos utilizadores, tem sido impossível para os desenvolvedores de aplicações, negligenciar esta evolução.

Mesmo assim, em alguns casos específicos torna-se muito difícil justificar o desenvolvimento de aplicações web, sendo que os benefícios obtidos são muito baixos e os gastos de produção muito altos. Devido a este tipo de problemas entre outros pequenos detalhes, tem surgido novas tecnologias que tentam encurtar a diferença entre aplicações desktop e web, tentando combinar o melhor dos dois mundos.

Existem algumas tecnologias muito conhecias tais como Java Applets, que são no fundo aplicações Java que podem ser executadas dentro de um browser. Mesmo estas tecnologias sendo muito interessantes e até mesmo muito úteis em certos casos, grandes companhias tendem a evitar o uso destas devido a alguns problemas que têm aparecido ao longo do tempo e que podem por em jogo a segurança e duração de vida dessas aplicações. Um grande problema encontrado neste ramo, são falhas de segurança na instalação de plugins, que esta a fazer com que este tipo de tecnologias esteja a ser abandonada.

Com este projeto criamos uma Framework que permite e facilita o desenvolvimento de aplicações híbridas. Esta Framework permite que os utilizadores desenvolvam aplicações em Java SE, que podem ser acedidas como aplicações normais, desktop, mas ao mesmo tempo é possível aceder a algum do conteúdo dessas aplicações a través de um simples Navegador de Internet. Isto tudo criando uma nova camada composta de tecnologias Web tais como HTML5, CSS e JavaScript sem ter de recriar a aplicação.

Com esta solução é possível evitar grandes custos no desenvolvimento de novas aplicações Web, e ao mesmo tempo evitar pequenos problemas de segurança como os que já foram mencionados antes em soluções já existentes. Desta forma é possível criar aplicações nativas que podem ser reutilizadas como aplicações Web caso necessário. A ideia por trás deste projeto não se concentra em permitir criar novas aplicações híbridas, sendo que já se encontram soluções para este tipo de problemas. Pelo contrário o principal problema que tentamos resolver com esta solução é permitir a aplicações Java que já existem ou que estão a ser desenvolvidas como simples aplicações nativas, possam no futuro ser portadas para o domínio Web sem que seja necessário a recriação de uma versão completa Web, e ao

mesmo tempo limitar o custo desse processo a um mínimo, requerendo apenas que uma nova UI para a versão Web seja desenvolvida.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

Nowadays, the software industry is vast and grows very fast, with it the choice of technologies for software development is always growing. Technologies that might be adequate for some projects today, might not represent the same stability in the future. Every day it becomes more and more complex to decide between different technologies. Whether a single programming language can tackle all challenges and even worse the decision of which software to develop, native or web and to which platform, in order to target as much users as possible. Nevertheless, some technologies get more attention than others such as Java being one of the most popular programming languages (Garbade) due to its flexibility to tackle different software markets such as Web and Native applications as well as mobile devices. Even though these technologies are very popular and always evolving, in some cases it is difficult to create software for all existing platforms using a single programming language without having to make some compromises such as targeting specific operating systems, specific users or even specific types of software (e.g. target windows devices or Mac OS devices and being able to offer the same user experience on both, or even decide between Desktop devices and Mobile devices). Due to these difficulties, Web applications have been growing since they can be developed using multiple programming languages, but mostly for their flexibility to target almost any device containing an Internet connection.

The solution adopted by most companies is to target a small number of platforms or even a single platform in order to reduce development and maintenance costs, but at the same time avoid problems such as having two teams building the same application for different platforms, which can in some cases cause problems regarding their look and feel between platforms, the difficulty to offer the same functionalities between operating systems (Ashwini, 2017), as well as the absence of specific libraries and frameworks between the multiple programming languages.

To solve these problems some technologies have emerged, which offer the possibility to create new applications that can be ported between multiple devices with different operating systems. These types of applications are called hybrid applications. In general the technologies offer the possibility to target a maximum of two operating systems or two devices such as for example Windows and Linux for desktop operating systems or Android and iOS for mobiles. Even though these solutions help migrate the problem, they require

some compromises, such as for example the use of specific programming languages (in most cases Web technologies) and the need to create applications from scratch without the possibility to adapt older applications.

This thesis focuses on the development of a new Framework for Hybrid application development. This framework tries to tackle some of the problems encountered with other similar technologies, by offering the possibility to create applications using the Java programming language together with a new Web user interface that can be ported between different operating systems. Additionally, it offers the possibility to adapt already existing applications, making them cross-platform. For better integration between systems we decided to use Java since it already offers the possibility to create application for multiple platforms and different operating systems, which can in some cases be difficult with other programming languages. This framework also gives developers, the possibility to make those same cross-platform applications, accessible through the web using a web browser by adding some extra code to the native application.

1.1 MOTIVATION

Java is one of the most well-known if not the most popular programming languages, competing with a vast world of similar technologies that offer comparable functionalities, making Java applications very common and vastly distributed. TIOBE[1], a company focused on software quality measurement, has listed the most popular programming languages in the last decade including specific values for the last two years. This list presents *Java* in first place for the most popular programming language with a leading rate of 13.2% compared to *C* which has the second place with a rating of only 10.2% for the last month of 2017. Figure 1 shows the evolution of programming languages in the past seventeen years. In this figure we observe a common issue between all the programming languages, it being a decrease in popularity that has a greater impact on those having a higher rate.

1 https://tiobe.com/

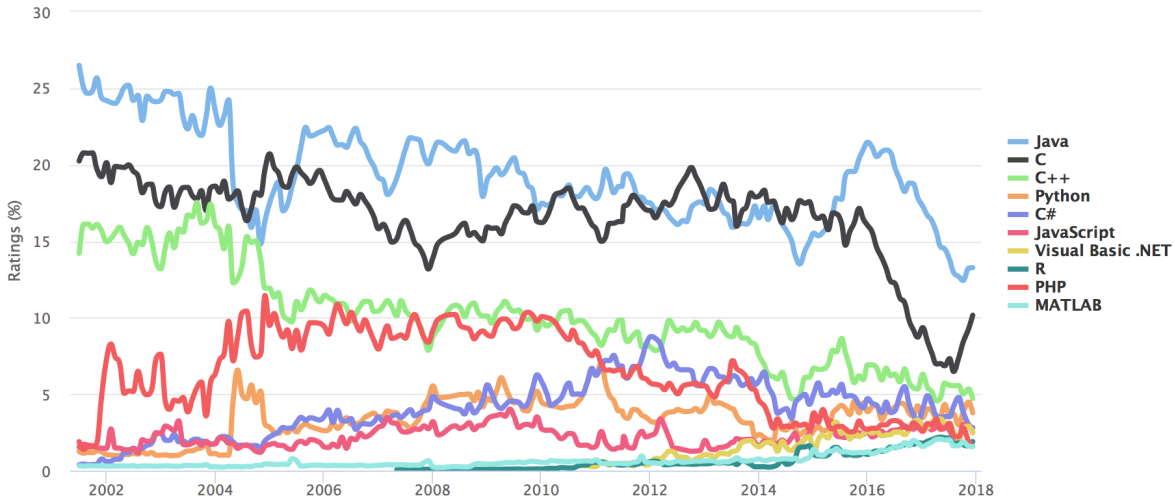


Figure 1.: Programming languages popularity since 2002, by (TIOBE).

Java is vastly used due to the cell phone industry market leader operating system (Android) which requires applications to be developed in Java, the vast world of Web applications developed in Java, but also due to the frequent use of those new technologies in schools and universities. Even though applications developed with this technology are very common, there are some well-known weaknesses and flaws that require some attention.

One of the main problems faced in the Java world is the current trend that focuses on Web Applications development. These are easier to use, and most of the time do not require much effort by the end-user to install and configure, making Web technologies very appealing. User habits tend to follow the direction of what makes their life easier, which led to the adoption of Web Application very quickly, due to their flexibility and availability. The Web Application market is huge, and offers the possibility for users to access their favorite applications by the means of a simple browser in any kind of device, anywhere in the world.

Even though Web Applications are becoming more and more relevant, native applications continue to represent a huge role in the application development industry. One of the main problems found in native applications is the single user approach. Only the person in front of the computer, where the application was installed, can have access to it.

The current multiuser trend is having a huge impact in the world of Application Development, forcing companies to develop their application multiple times from scratch to make them available on all platforms. This has a huge impact on legacy applications, but also on new applications, which don't take this multiuser paradigm into consideration while developing and have to be adapted later on.

In order to make their Java Applications available on multiple platforms, not only do developers need to create a new *user interface* with Web Technologies, but they also need a

completely new Back-End for those applications to work. This strategy is time consuming and can sometimes be very costly. In some cases those costs are so high compared to the small benefits the companies make. This leads companies to give up on this kind of approach.

In order to deal with this problem, *Hybrid Applications* have emerged and are being adopted by multiple companies. Hybrid applications have in most cases their front-end and back-end developed with Web technologies. This permits to have a multi-platform applications that uses the same Web user interface between different devices. These application work like any other common application that is installed on a machine and runs locally, but their interface is displayed inside a specific browser window (container) that makes them look like native applications. This way companies can create a single application that runs on multiple operating systems.

1.2 PROBLEM

The objective of this thesis is to solve the problem of developing multiple applications for multiple platforms, in this specific case to break the boundaries between *desktop Java applications* and *Web Applications* by creating a framework for hybrid web/desktop application development.

We aim to provide an easy solution for Java Developers to make their applications available on the Web without having to create two complete different applications for multiple platforms. Furthermore, we aim to offer the possibility for developers to provide their already created Java applications on the Web by simply using this framework together with a new Web Interface that they design, without needing to create a new back-end. The proposed framework provides a connection between both worlds Desktop (Java) and Web applications that aims to reduce time and costs of software development.

To reach these goals we prepared a new architecture that takes in consideration some of the already existing frameworks for hybrid application development to which we made some modifications to better suit the proposed solution. This architecture is compared to the more general Native and Web application architectures in Figure 2.

On the left side of the figure we can see a general architecture of Web applications which is composed of four layers. The first layer resides on the machine where we want to interact with the application, through the means of a web browser that interacts itself with the operating system (for specific tasks such as access local files). Then we have a second layer, the Web Server. This layer will serve the web pages to be displayed by the Web Browser. When a request comes from the browser to the Web Server, it will either respond with a new web page, or send the information to the third layer if some task needs to be performed. The Application Server layer is where the application's logic resides. When a task is performed

it will communicate back to the Web Server that will then respond to the Web Browser with some data. Finally we have a Database Server which contains all databases. This is where the data of the application is stored, and is only accessed by the application server when some data is required to perform some task or when there is new data to be stored.

On the center of the figure we can see a general architecture of Native applications. Here we have a single layer that interacts with the operating systems and at the same time contains the logic of the applications as well as the capability to display the user interface. Native applications work locally and the database with which they communicate is in general installed on the same computer as the application. These applications also provide a way to display the UI which does not require external help such as a browser like the one used in Web applications.

On the left side we have the architecture we planned for our hybrid framework. As we can see here we have two components, one local and another one external. The local component is installed on one machine where the logic of the application lays. The installed application is capable of locally serving a Web interface inside a native UI. This Web UI transmits information to the rest of the application using specific communication mechanisms that compose the Communication Layer. In general the database used for the application is also installed on the same machine, but in some cases it can be located on a Database Server as in the Web Application architecture example. Finally, the external component that runs on other machines uses a Web browser to communicate with the Java application. Since the UI displayed on the local component is a Web interface, it can also be used by the browser. The main difference here is the communication mechanism used between the browser and the java application which is established using HTTP. All the functionalities available on the local component are also present in the external component.

We can also see that the Web Application does not support the same type of interaction between the application and the operating systems. This is due to the Web browser not being capable of interacting with some system components in the same way that local applications can (an example of this type of behaviours, are cameras that in most cases only work in local applications).

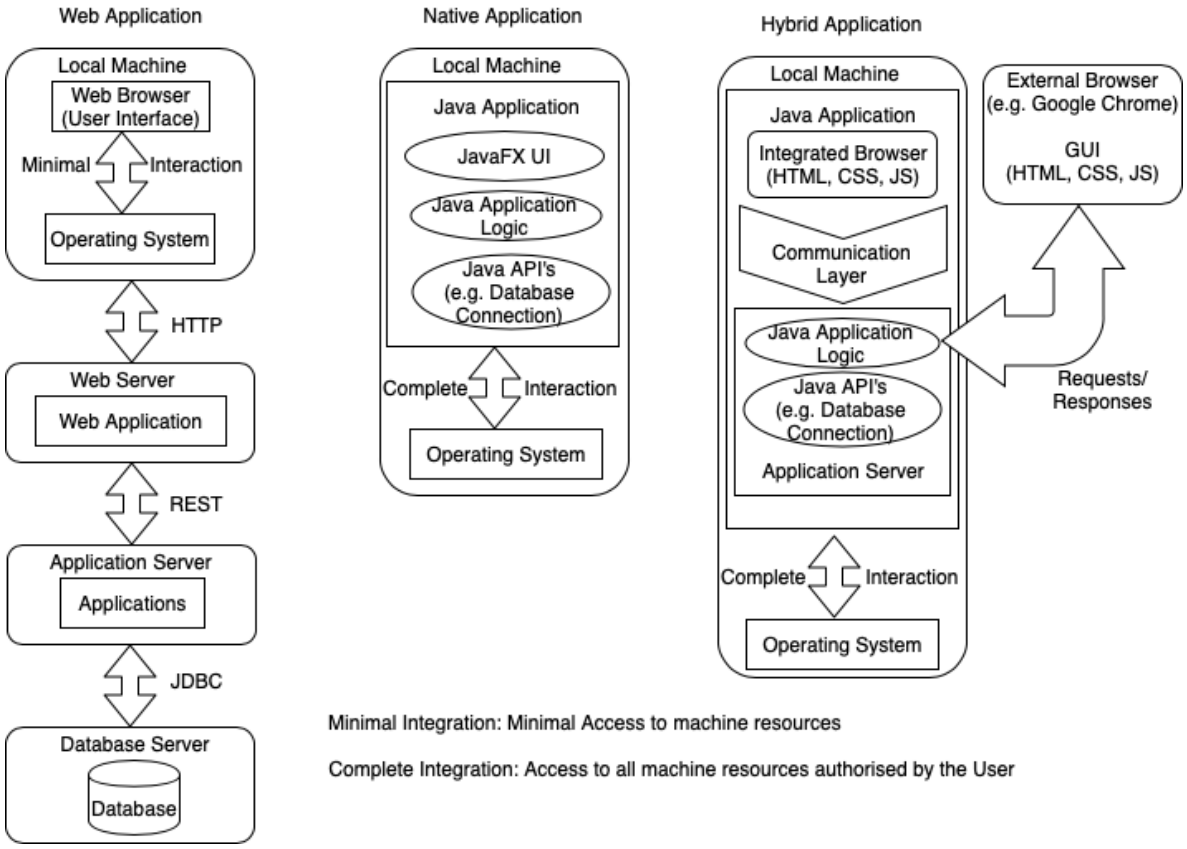


Figure 2.: Web Applications vs Native Applications vs Hybrid Applications stack comparison

1.3 OBJECTIVES

This work consists in developing a framework called *JxAppDev* (**J**ava Cross-Platform **App**lication **Dev**elopment), which main goals are to facilitate and speed up the process of porting native Java applications between different operating systems, as well as making those applications available on the Web.

This framework will provide an easy way to serve functionalities from native applications on the Web, without requiring to re-write the back-end of the application. This work will focus on porting to the Web, applications that were already developed and for which the developer will only be required to create a new front-end, using modern Web technologies such as HTML5 (Freeman, 2012), CSS (McFarland, 2012) and JavaScript (Duckett et al., 2014). In some cases such as stateful applications some extra changes might be required for the facades to work properly with the JxAppDev.

The main goal is not to create a Framework that makes all current Java functionalities work, but on the contrary, we will focus on a predefined set of features and common tasks

that Java application can provide, such as Button actions, navigation and other simple controls.

The framework does not offer a large number of functionalities. Instead it focuses on facilitating portability of applications between operating systems without having to completely rewrite those applications for different platforms with multiple programming languages.

The framework focuses on a specific type of applications since it will be based on asynchronous communication between the different components, and it will only support applications based on a multitier architecture. Taking this into consideration, the framework will not support real-time applications nor intensive graphical applications.

At the end of this work, a test application will be developed using this framework as a *use case* to better explain the capabilities of the framework, how it works and what limitations are present at that point. This will demonstrate the capabilities of the framework to create hybrid application that can be ported between different platforms.

A second *case study* will be presented, which consists of transforming an already existing native application, which will then be accessible through the Web.

This document will be a guide through the work that will be done, but at the same time it will serve as user guide to explain how the framework is supposed to work.

In order to achieve our goal, the following set of objectives is proposed:

- The first objective of this master's thesis is to investigate similar frameworks, technologies and approaches. After gathering enough information we want to elicit the most appropriate data on related work to facilitate the development of applications according to the proposed approaches, as well as select technologies that can be incorporated into our framework in order to facilitate its development. This will help us better identify potential problems with which we are faced, as well as provide guidelines to solve those problems.

- The second objective is to design the proposed approach, based on the previously collected information in order to obtain a concrete architecture for our framework.

- The third objective is to implement the Java Framework using possible frameworks or technologies that facilitate our work, according to the established requirements.

- The last objective is to perform two *use cases* using our framework. The first one consists of developing a small Java application from scratch and the second one consists of adapting an already existing application. The purpose of these *use cases* is to illustrate all the features and limitations of the framework.

For this framework to work there is a set of requirements that must be properly met by the Java application used or developed to be integrated with the JxAppDev.

- Applications must be built using a multitier architecture
- Applications must offer all the desired functionalities that they want to provide (e.g. applications that need a login, must already provide a login mechanism)
- Stateful application must provide the proper facades for the integration with the JxAppDev
- Applications must not require real-time processing (e.g. streaming application)
- Applications must not require high demanding graphical processing (e.g. games)

1.4 DOCUMENT STRUCTURE

This document is composed of 6 Chapters. The first Chapter is composed of the introduction which addresses the problems and objectives of this work.

Chapter 2 presents the related work which consists of technologies that tackle similar problems to the ones listed in this document, and detailed information about technologies that are required to develop the JxAppDev framework.

Chapter 3 contains details about the proposed solution and presents the JxAppDev detailed implementation together with some explanations on the approaches that were taken, the decisions that were made and their purposes.

Chapter 4 presents two case studies which demonstrate the frameworks usability, capabilities and limitations.

Chapter 5 contains a study aimed at validating the framework and the derived applications.

Finally, in Chapter 6 we conclude with a discussion of the results, as well as some potential directions for future work to improve the current solution.

2

RELATED WORK

This Chapter introduces concepts, software and libraries that are required to understand this thesis. We will discuss the available technologies related with this work, together with technologies that are similar to what we are trying to achieve. We will compare all the presented possible solutions and describe them to have a better understanding of the problem and find technologies that suit better this work.

2.1 BACKGROUND

Companies are sometimes forced to choose between native and web application, as well as decide which programming language suits best the tasks at hand in order to avoid having multiple teams working on multiple applications for the same purpose. These problems are forcing companies to adopt a new trend in the past few years, which consists in developing hybrid applications (Lynch, b). Hybrid applications in this context, are a new type of software that is implemented using conventional Web technologies but at the same time includes a local component executing on a mobile/desktop device (Martin et al., 2014). Using *HTML, CSS and JavaScript* technologies for the Web component of these applications, makes them portable and therefore usable in any platform. These applications become cross-platform and can be deployed on any Desktop or Mobile OS. The second component of these applications has a very interesting aspect. Since hybrid applications are based on a local component (e.g Java Back-End), it becomes now possible for the Web component to access local storage, local documents or even other things such as *Hardware* (e.g. Camera) on mobile devices.

An extensive research concerning related technologies and works was performed, addressing books, papers and websites. Not a large number of results were found. As presented next we focused our research on subjects that work around the same problems.

Within the found results we tried to extract relevant technologies that support the development of the JxAppDev. We will start by describing those technologies and the solutions that were discovered during this research phase.

One main problem that occurs in current hybrid application is the security and user rights problem. Since hybrid applications are based on two completely different software components, some security breaches may occur. Those breaches are due to the difference in security measures taken by Web applications which use *Access Control Policy's* (Sandhu et al., 1996) based on the HTTP/HTTPS (Gourley et al., 2002) *Same Origin Policy* (SOP) system, while local resources access is governed by the OS's *Discretionary Access Control Policy* (Jordan, 1987) that is very different form the Web security model. Due to the differences on these two security models, sometimes security gaps are found which allow access to local resources by the means of Web foreign-origins. A good example of this problem in Hybrid application is Android which is based on *user permissions*. A simple application usually requests for access to the device components such as the camera. When a hybrid application makes the same request, the access should be granted to the application itself, but not to foreign-origins content included in the application such as advertisements.

In the specific case of advertisements on websites, the user might grant the permissions to the application, which trust the ads broker, but neither of them fully trusts the ads content creator. Since ads tend to create HTML objects (*iFrames*) which execute scripts (JavaScript), it becomes very easy for attackers to use their own ads to inject malicious code and therefore cause arm to the end user of the application. Due to these problems the use of iFrames has decreased. As mentioned by (Martin et al., 2014) in their paper, Hybrid Frameworks require special attention in order to grant protection against these kind of problems, which can only be achieved by "gluing together" multiple layers and security policies belonging to the different components of the hybrid application.

A second problem with current solution to develop hybrid applications, is the need to create applications from scratch. This can be a problem since many companies have legacy software that is available on some platforms and needs to be ported to other systems or even worse to the Web. With the current solutions a complete new application needs to be developed and sometimes even more than one application since none of the found technologies allows porting local application to the Web.

Some more aspects that are important when taking into consideration hybrid applications development are: the requirement for developers to learn web programming languages, the complexity of making one application perform identically in multiple platforms and track the performance costs in multiple platforms, since in some cases hybrid applications can be more resource demanding and therefore perform worse in some operating systems, compared to native applications.

2.2 SIMILAR TECHNOLOGIES & PROPOSITIONS

During our research we found no plausible solution available to solve our problems without diverging from our goals. To the best of the author's knowledge, the current solutions focus on web technologies without offering the possibility for developers to use other programming languages, such as Java in this specific case. None of the found technologies offer the possibility to convert current developed applications into cross-platform applications, nor offer the possibility to convert local application into web applications without completely recreating those applications.

The mobile application domain was a source of inspiration since the idea behind hybrid Web/Mobile application is quite the same. We could find a considerable amount of work created around this topic, which was used as groundwork for our proposed approach, but also to illustrate similar solutions and technologies.

2.2.1 *Hybrid Mobile Applications*

Progressive Web Apps (PWA) are a relatively new type of mobile applications that was presented by Google in 2015. These applications take benefits of Web applications and bring those functionalities to mobile devices by the mean of web technologies such as HTML, CSS and JavaScript. The main idea behind PWA is to offer a mechanism that allows Web applications to run on mobile devices with almost all the benefits that native applications offer. This is possible through the current advancements in Web browsers, together with a new technology called service workers. These service workers lie between the network and the device itself, they are scripts that serve the content to the application and provide new functionalities such as push notifications and caching. With these workers, the PWA can provide offline content by using the caching mechanism to keep important content to be served while there is no network connection.

Nevertheless, this type of applications has some drawbacks. At the moment not all operating systems support these applications, such as for example iOS. Most of Android devices support PWA but in some cases not all sensors can be used like for example bluetooth, finger print readers.

PhoneGap (Ghatol and Patel, 2012) is an *Open Source* framework whose main goal is to facilitate the development of Hybrid applications for multiple Mobile operating systems such as iOS (Joe Conway, 2012) and Android (Dixit, 2014). This makes it unsuitable for our objectives, since we are focusing on Desktop Application. This framework is developed by Adobe Systems and based on Apache Cordova (Nachimuthu, 2017). Its community shows the interest invested on this type of technologies.

PhoneGap can be used to create applications from scratch, by simply using Web technologies such as HTML5, CSS and JavaScript. In some specific cases, such as when using external libraries, the programmers are required to have some basic knowledge of the programing language of the specific target mobile operating system to which they are developing. Apart from being Mobile Driven, this Framework had another disadvantage, since it only allows to create applications from scratch and not convert already finished Java Application to be available on the Web. Even though it cannot be used on this project, this is a good example of a framework whose main focus is to facilitate the development of Hybrid Applications.

We can see in Figure 3 how the mobile application stack is composed compared to a Hybrid solution. A lot of details about mobile hybrid applications can be found together with descriptions and presentations of multiple frameworks created for this purpose, including PhoneGap. As we can see in this figure, the native application architecture is much less complex since all API's are directly connected, both to the operating system as well as to the native application itself.

On the other hand, hybrid applications require some extra layers to work. The HTML layer is only in contact with the rendering layer (Webview) that is capable or drawing the graphical user interface. This Webview layer interacts with a layer of plugins that can be used by the web component to interact with the API's layer (same in the native version) through a bridge. As in the native version, this layer of API's communicates with the operating system. All the layers are required for the HTML5 application to interact with the operating system, but at the same time this extra complexity allows for the application to interact with multiple operating systems, which is not possible in most cases with native applications since they focus in general on a single system.

Apart from PhoneGap, during our research we were able to find multiple other technologies for Mobile Hybrid Applications that resemble this last one. Among those technologies some, such as Xamarin (Hermes, 2015), Ionic Framework (Phan, 2015), Framework7 (Kharlampidi) and Mobile Angular UI (mcasimir), have good supporting communities. These technologies focus on facilitating the development of Hybrid Mobile Applications by making them as cross-platform as possible. Considering their resemblance in objectives and functionalities, we decided to simply focus on PhoneGap in order to be more specific and clear as well as to not repeat ourselves since they all share the same advantages and limitations .

This set of technologies establishes a new paradigm for building native application using web technologies, which weren't widely explored until now. Next we present some technologies which are more related with our objectives, that are inspired on the ones just mentioned.

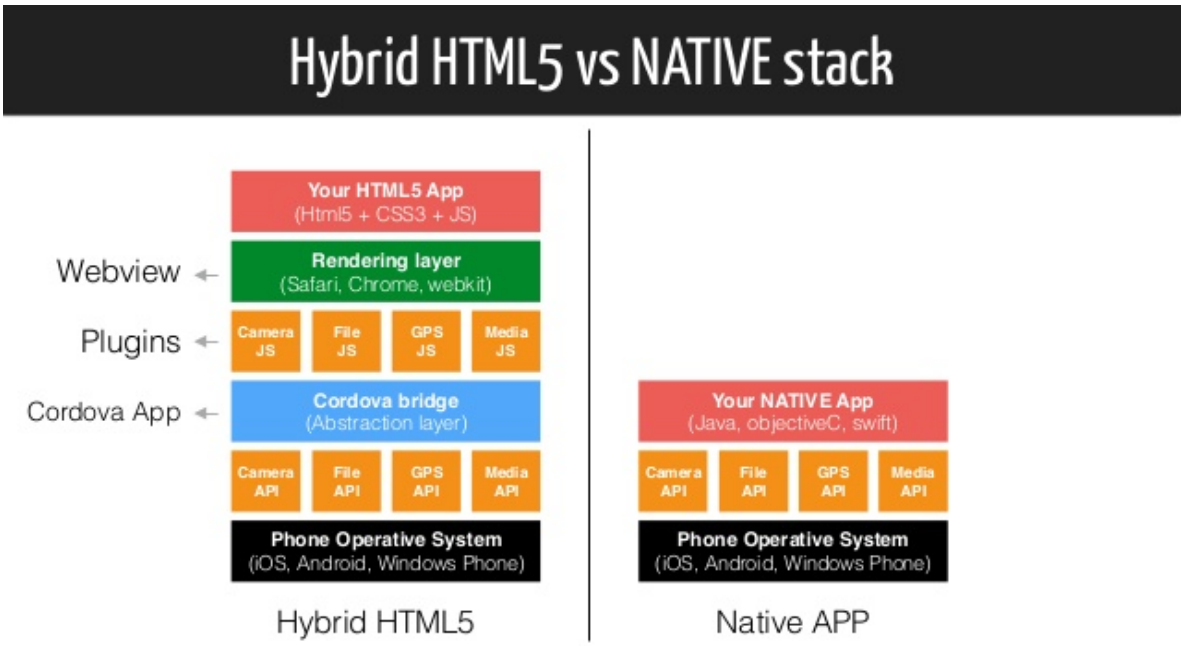


Figure 3.: Comparison between Hybrid Application Stack and Mobile Application Stack by (Rodrigues, 2015)

2.2.2 *Cross Platform Hybrid Frameworks*

This Section will cover three Hybrid Frameworks which address some of the problems that we intend to solve. These frameworks are based on JavaScript to develop Desktop applications and are used by multiple Software Development companies such as Microsoft, GitHub and many more.

Electron

The first framework worth mentioning is Electron (Griffith and Wells, 2017), one of the most popular frameworks for cross-platform application development. Some of its functionalities are very close to the ones we want for JxAppDev but its main goals diverge from our needs.

Electron is an open source framework developed by *GitHub*. This framework allows developers to create applications from scratch that are purely based on HTML5, CSS and JavaScript. Due to the use of Web Technologies these applications can run on multiple desktop platforms.

The solution proposed by Electron is based on a combination of Chromium and Node.js (Syed, 2014) into a single runtime environment.

Chromium is a very light-weight open source browser developed by Google, that packs the most important functionalities found in Google Chrome, one of the most advanced Browsers that exist. Chromium has a minimalist user interface, and was developed to mainly offer the users as much flexibility as possible, such as running as a Shell for the Web, something that is not possible with Google Chrome.

Node.js is a runtime environment for JavaScript, built on Chrome's V8 JavaScript Engine. The main goal of Node.js is to allow the execution of JavaScript code as Sever-Side scripting, bringing flexibility to the creation of dynamic Web content before the pages are sent to the browser.

With the conjunction of these two very popular technologies, Electron is able to offer the user a unique experience where he can develop desktop applications which run very similar to their browser counterpart, without losing a lot of performance or providing very bad user experience. We can see in Figure 4 a picture of a simplified Architecture of an Electron Application. In this figure we can see that there is a JavaScript file that is always running inside the main process of the application. All the interaction with the Operating System passes through this main process that uses native API's for the interaction. We can also see that every Window has its own rendering process which means that we can create as many windows as desired at the same time without them affecting each other. The communication between the different windows and the main process is done through Inter Process Communication (IPC) which allows processes to communicate with each other and synchronize their actions.

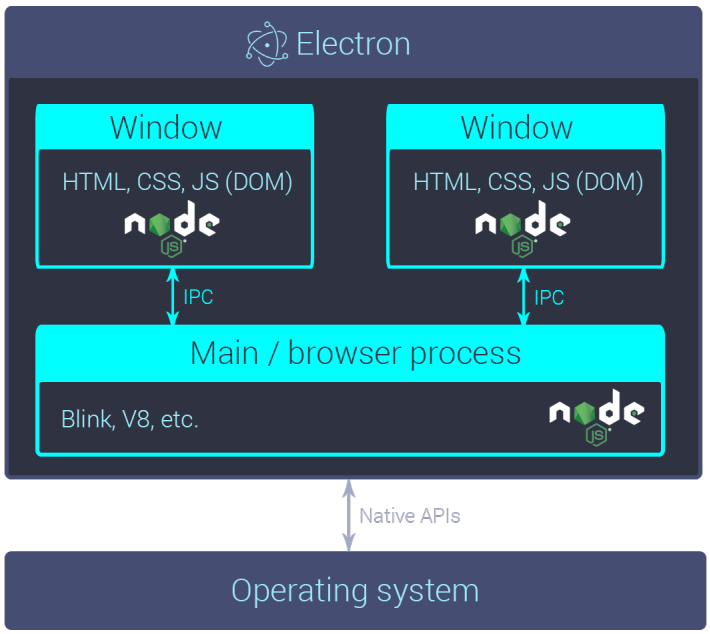


Figure 4.: Electron application architecture by Adam Lync. (Lynch, a)

The Electron Framework provides various advantages when developing application from scratch since it allows developing a single native application using Web technologies, that can later run as a single desktop application, offering the exact same functionalities that are present on the browser. There are a lot of well-known applications that were developed using this library such as Slack or GitHub Desktop.

The reasons for this framework not to meet all the proposed requirements are the following. First, this technology is only usable when developing application from scratch, otherwise we must re-write the complete application. This can be a problem since we mainly want to create a bridge between native applications that exist, and require to be ported to other platform and possibly the Web.

Second, users must directly develop applications in HTML5, CSS and JavaScript, which is quite different from what we want to achieve. We want to be able to develop applications in Java (back-end), and offer the possibility to create a new front-end using the same technologies found in the Electron Framework but only for the user interface.

Finally, compared to Java which is a very present and established programming language, Electron uses JavaScript which has other disadvantages such as few development libraries and even less support tools compared to the vast world of Java.

Meteor

Meteor (Vogelsteller et al., 2016) is a framework for full-stack hybrid applications development. Compared to Electron, this framework not only offers the possibility to create desktop hybrid applications, but it also allows to create mobile application for multiple mobile operating systems such as iOS and Android. Applications created with Meteor are based on JavaScript as programming language and Node.js servers. They can run on application servers, Web Browsers or even Mobile devices. Meteor uses a system called *Data on the Wire* which basically focuses their applications on sending pieces of data instead of complete HTML pages, that will then be rendered on the client side. This is a full-stack JavaScript platform that is based on the Node.js API and build tools. Since Meteor creates Node.js based applications, it requires Node.js hosting to be deployed. Due to this last requirement, Meteor recommends their on-hosting service called Galaxy (Inc.), which is based on Amazon Web Services (AWS) (Bermudez et al., 2013). Since Meteor is based on JavaScript, it supports integration of the most popular JS libraries such as Angular (Murray et al., 2018) and React (M and Sonpatki, 2016). For database, Meteor is fully integrated with MongoDB (Chodorow and Dirolf, 2010) out of the box without any further manipulations. As we can see, this framework shares the limitations found in the Electron framework, and even further it does not focus on Java which is a requirement for this project. Nevertheless, this framework is open source, well maintained and is used by companies such as Mazda, IKEA and many more.

NW.js

NW.js (Benoit, 2015) previously known as *node-webkit* is a framework developed by Intel that has the same purpose as Electron. It also focuses on the development of Desktop applications using Web technologies. These applications can be accessed locally but also through the Web. NW.js is also based on chromium but instead of using the *libchromium-content* (Electron) library to access Chromium's content API as Electron, NW.js focuses on using chromium as a complete package. Due to this specific chromium usage, the entry point of NW applications is a web page or a JavaScript Script (previously specified in the application) that opens inside a browser window. In comparison, Electron applications entry point is a JavaScript file, where a browser window has to be manually created in order to be able to load the desired HTML content. There are many other small differences between these two frameworks that can be seen in Table 1. Nevertheless, we should take in consideration that NW.js lost some of its users since the transition between *node-webkit* and NS.js, which caused quite a lot of problems making some users change to other technologies.

After this research, we concluded that none of the above solutions offers the complete set of desired functionalities for this work, such as to convert native applications that already exist into cross-platform applications and some other points that can be seen in Table 1. If we take a deeper look into this table we can see that there are some frameworks that offer similar objectives such as using Chromium on the front-end, develop application for multiple platforms (web, desktop, mobile) or even develop local and web application with single framework. Nevertheless, none of them allow the development of Java application which is an important objective of the JxAppDev, neither do they offer the possibility to transform already existing application into hybrid application. Even more, none of them offer the possibility to develop a single application that can be available on all platforms. These findings reinforced the need to create a new framework.

	PhoneGap	PWA	Electron	Meteor	NW.js	JxAppDev (proposal)
Device compatibility	Mobile	Mobile	Desktop	Web, Mobile, Desktop	Desktop	Web, Desktop, (Mobile)
Programming language	HTML5, CCS3, JavaScript	HTML5, CCS3, JavaScript	HTML5, CCS3, JavaScript	HTML5, CCS3, JavaScript	HTML5, CCS3, JavaScript	Java, HTML5, CCS3, JavaScript
Adapt Existing Applications	No	No	No	No	No	Yes
GUI Container	WebView	WebView	Chromium		Chromium	Chromium
UI Libraries	Angular, Backbone, Bootstrap		Angular, React, Reactive, Photon, Bootstrap	Angular, React, Blaze, Bootstrap	Angular, React, Bootstrap	Angular, React, Bootstrap
Back-End	Applicaion Server	Application Server	Node.js	Node.js	Node.js	Application Server
Application Server Communication	RESTful XML, JSON, SOAP, Sockets	HTTP/HTTPS		HTTP/HTTPS		RESTFul API, JSON, Sockets
Communication	Ajax	JavaScript	Inter Process Communication (IPC)	JavaScript	JavaScript	Ajax, JavaScript
Local Application	No	No	Yes	Yes	Yes	Yes
Online Application	Yes	Yes	No	Yes	No	Yes

Table 1.: Framework comparison Table

2.3 TECHNOLOGIES

In this section, we will discuss the existing technologies for the different components required to develop the JxAppDev framework, according to the objectives previously set and the abstract architecture we previously defined (Figure 5).

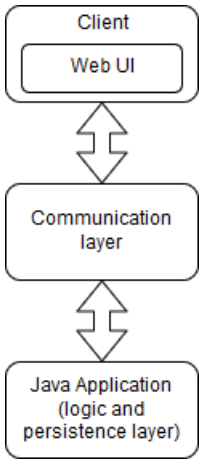


Figure 5.: JxAppDev framework abstract architecture

The architecture of the JxAppDev will require a component that is capable of presenting web content inside a Java Application, a layer of communication between the user interface and the back-end as well as a back-end component capable of understanding the requests coming from the user interface and interact with a possible existing Java application that will treat those requests. To be able to create these components we gathered information on topics that are important for this development and which will be presented in the following order.

First, we will present different Java GUI (*Graphical User Interface*) toolkits that can integrate a Web component to present the hybrid applications user interfaces. Second, we will mention some IDE's (*Integrated Development Environment*) that can integrate those Java toolkits. Third, we will specify some technologies that are used on Java toolkits to present Web interfaces. Fourth, we will discuss about the different types of applications that are glued together to create hybrid applications, such as Web Applications, Desktop Applications along with others. Fifth, we will present some technologies that can be used to create a communication layer for the different components of an hybrid application. Finally, we will discuss about some Java features that will facilitate the integration of already existing applications into the framework.

2.3.1 *Java GUI toolkits*

We start the process by analysing a list of technologies that use Java Programing Language, see what they offer, compare them and then choose which one is the most appropriate for our objectives and at the same time find the most future proof technology between them.

Abstract Window Toolkit (AWT)

AWT Java's first GUI toolkit, was almost directly discarded from our list since it is quite old and does not offer as much flexibility as the others that we decided to take a look at, and it was also replaced by Swing. We should nevertheless take into account that more recent solutions such as *Swing* still use AWT's interface to communicate with the operating system, regarding the management of windows and events. Thus, the need to take this technology into account.

Swing

Swing (Loy et al., 2002) was one of the most important picks for this list due to its legacy being the only Java GUI toolkit developed by Oracle for multiple years. It offers a lot of flexibility since it was built on top of AWT and components are much lightweight. Even though it seems very legit to use Swing in this project, Oracle has been working on JavaFX and we found many articles mentioning that JavaFX should become the successor of Swing. We even found a question inside Oracle's overview documentation for JavaFX, where they mention this specific problem, and where it is clearly written that JavaFX was created in order to replace their previous solution Java Swing (Oracle, b). It is also worth noticing that Swing does not support the Webviews from the new WebEngine that is present in JavaFX which is one of the core components that might be required to accomplish this project.

We were able to find some open source projects (Jackson), (Oracle, e) that demonstrate the use of Webviews in Swing, but those solutions are based on using JavaFX components inside Swing applications. To make this project as simple as possible and focus on the implementation of JxAppDev, we think that there is no valid reason to use both Swing and JavaFX in a single application, making the implementation of our approach more complex.

The Standard Widget Toolkit (SWT)

SWT (Harris and Warner, 2004) is an open source widget toolkit which main goal is to offer an alternative to Java integrated toolkits (AWT/Swing). It is a free software licensed under the *Eclipse Public License (EPL)* and developed by the Eclipse Foundation developers of the Eclipse IDE, one of the IDE's in our list (see Section 2.3.3). One advantage of this toolkit compared to others, is that for every element, it uses native GUI libraries of the operating

system where the applications is running. This makes SWT implementations adjust to each OS, offering a native "look & feel" and the native "performance" for widgets in contrast to other solution like Swing. Though in terms of design SWT is a better choice than Swing or any other technology in our list, we decided not to use SWT due to it requiring additional native libraries for the different operating systems. This problem occurs due to the use of native GUI looks and feels. For this to work, SWT requires external native libraries which can sometimes cause unusual behaviours on different platforms. Since this project focuses mainly on application portability and that we want to limit framework complexity as much as possible, we decided to avoid this type of risks and therefore look for other technologies that might suit better. Nevertheless, we acknowledge possible usability issues arising from not adopting OS specific elements.

JavaFX

Putting together the lack of WebViews support with the possibility of Swing being replaced, we decided to take a deeper look at JavaFX (Grinev, 2018), and analyse what benefits it could bring to this project and whether its use would be justifiable.

After analysing JavaFX documentation, we found out that this technology brings new updates to the table such as the WebView integrated in their WebEngine. As a first glance, JavaFX looks ideal for our project since it is well presented and seems to be the best and more complete technology for the approach we want to adopt.

Though JavaFX's API is the technology that seems more relevant for this project and might contain most of the functionalities we will need to develop our framework, we decided to take a look at the list of updates that have been published by Oracle until now. We were very disappointed in the results we could find since Oracle has been making very little updates to the JavaFX library and even worse, most of those updates haven't included any news for the WebView component that we require.

Since JDK 8 (*Java Development Kit*) first released in 18 March 2014, which included updates for JavaFX's WebView (Oracle, c), there was no WebView update, at least none that were relevant as we can see in the Oracle's Documentation (Oracle, a), until the new JDK9 was released. Oracle has been making some efforts to compensate the previous lack of updates. One of the big changes coming with the latest version, Java SE 11 is that JavaFX will be detached from the JDK and will be a separate component that can be updated and altered separately. This means that new updates can come to JavaFX without the need to completely update the current JDK.

To better illustrate this problem, we included in Table 2 all the updates that can be found on Oracle's Web Site.

Version	Release Date
JavaFX 1.0	04/12/2008
JavaFX 1.1	12/02/2009
JavaFX 1.2	02/06/2009
JavaFX 1.3	22/04/2010
JavaFX 1.3.1	21/08/2010
JavaFX 2.0	10/10/2011
JavaFX 2.1	27/04/2012
JavaFX 2.2	24/08/2012
JavaFX 8	18/03/2014
JavaFX 9	21/09/2017
JavaFX 10	20/03/2018
JavaFX 11	25/09/2018*

Table 2.: JavaFX Updates

*At the time of writing this document, a new version of JavaFX is about to be released, which might bring new updates regarding the use of WebViews and their functionalities. However, this work was developed taking into account the available technology at the time of writing.

This research provided us with some information about the JavaFX framework. Before the version 2.0, JavaFX was based on a Scripting language that was dropped with this new release that introduced the common Java Language to which we are used today. This decision was taken in order to avoid developers having to learn another scripting language and so facilitate application development. Until the 2.2 Version, JavaFX was standalone, but then at the launch of the JavaSE 7 update 6, Oracle decided to integrate the JavaFX libraries into the Java JDK and the Java JRE. After February 2013, Oracle stopped supporting the JavaFX standalone version and it could only be found in the Java JDK and Java JRE.

Since JDK9 there have been some extensive updates and new functionalities added to JavaFX which might be improved in the future, due to JavaFX becoming a standalone component again in the most recent version.

Nevertheless, the lack of previous updates which were concerning and Java SE 11 being a too recent update and all the changes are currently happening, raised a certain level of incertitude towards the technology. We decided to look deeper and see if there was some other technology capable of offering the same advantages or even bring something extra to this project.

JxBrowser

The JxBrowser library (TeamDev, 2015) integrates Chromium into JavaFX and Java Swing applications. This offers more flexibility since we can choose between both Java toolkits and

the use of Chromium to replace the JavaFX WebViews. Chromium offers the possibility to use very recent technologies such as HTLM5, CCS3 and JavaScript to display web pages inside a Java application as well as manipulate those pages.

Other than offering a suitable solution for our problem, this library is quite well maintained and is frequently updated to guarantee that the most recent Web Standards are supported.

There is a small downside when using this library since it is a paid project that is maintained by a team of developers. This required us to obtain a license, making the usage of JxAppDev limited to users possessing a license for this product.

Due to the great flexibility of JxBrowser and since we were able to obtain a license, it is the most suitable solution for the approach we decided to take. This library is well supported and updates have been very frequent. The possibility to develop using JavaFX and Swing with the integration of Chromium makes it the most appropriate for this project giving us much flexibility.

2.3.2 *Embedded Web Views*

As mentioned before, this project requires a technology that offers the possibility to display web pages inside Java applications. During our research, we found two technologies that work with Java and have a good support team working frequently on them.

The first one are the WebViews that we encountered while reading about the JavaFX framework. The WebView is a Browser API based on WebKit (Raasch, 2011), which is an Open Source Web browser engine very popular and used by multiple MacOS applications such as Safari, the App Store as well as some Linux applications. The WebKit can also be integrated in JavaFX applications which will then have the capability to use WebViews, and contain all the core functionalities of a simple browser such as load web pages, apply CSS Style Sheets into those web pages and at the same time run JavaScript.

Even though this technology looks very promising, it has a flaw, since it only allows the management of a single page at a time.

The WebEngine is composed of two main classes that together will offer all functionalities mentioned above. The first class is the WebEngine class which is used to handle simple user interaction actions such as link navigation and HTML forms submission. This class main goal is to load web pages, access the DOM (*Document Object Model*) and execute JavaScript. The second is the WebView class which encapsulates the WebEngine class and provides properties and methods which help provide effects and transformations on web pages.

Due to the restrictions previously mentioned, an alternative to WebViews is the use of chromium, a light version of Google's Chrome browser.

The actual Chromium ((Google)) project contains two components. The first one is called Chromium, a light browser that aims at offering all users a better, more stable and possibly safer way to navigate the Web. The second component of this Chromium project is called the Chromium OS which was created to offer a fast, simple and more secure computing experience for users that spend most of their time on the web and don't really require all the functionalities of a common OS. This Chromium OS is generally used on Google's Chromebook devices. Chromium is also based on the same WebKit as JavaFX's WebViews, but it offers its own rendering implementation. This implementation is the same used in Google Chrome which, guarantees that pages are properly displayed as if it was a common web browser.

Nevertheless, here we are only interested in Chromium browser. As we saw before, it is used in Electron, a very well-known framework for hybrid applications development which as a result of its popularity seems to work well.

Chromium is also present in the JxBrowser library, the toolkit used in this project that replaces the other choices such as Swing and JavaFX, which is very well maintained and therefore shows the quality of this open source project that multiple companies decided to use, instead of other technologies such as the WebViews from the WebKit previously mentioned.

2.3.3 *IDE's*

In this subsection we present two IDE's that can be used to develop the JxAppDev, Eclipse IDE and IntelliJ IDEA. We mainly focus on these two choices since the list of IDE's is too vast and some of the technologies we decided to use were developed having in mind these two IDE's which facilitates the process of developing and integrating these technologies in our framework. All the technologies we intend to use in this project have been properly supported and their integration is well documented. Eclipse was selected due to its SWT integration and the support for other Java toolkits. IntelliJ IDEA was selected since JxBrowser, the more recent Java toolkit that we decided to use on our project, is well documented and presents a simple way to be integrated into IntelliJ with a lot of support in that matter.

Eclipse IDE

Eclipse (Holzner, 2004) is an extensible IDE that can be used for software development in a large variety of programing languages, the most common one being Java. Eclipse uses SWT as Java toolkit. Eclipse is composed of a workbench and a very interesting plug-in infrastructure that gives it an enormous amount of flexibility and customization, together with OSGI runtime environment (Osgi). It is a free software released and maintained by the Eclipse Foundation under the Eclipse Public License.

IntelliJ IDEA

IntelliJ IDEA (Essentials, 2014) is one of the many IDE's developed by JetBrains. It is a cross-platform Java IDE that supports the integration of many tools and frameworks such as *git, Sprint* and many more. This IDE is available in two different versions, one free of charge that is supported by the Apache 2.0 License (Apache), and a commercial version called Ultimate Edition that uses the same code base while adding new features like support for Java EE, Spring, SQL, Android and many other useful functionalities. The base GUI toolkit used by IntelliJ IDEA is Swing.

For this thesis, we will only focus on IntelliJ IDEA since we have access to its commercial version, but mainly because it is supported by the JxBrowser community. There are plenty of documents that facilitate the integration of projects into IntelliJ IDEA (TeamDev).

2.3.4 *Application Back-End*

Since we propose to combine two types of applications and create a bridge between them, we decided to analyse how those applications are composed, and compare them to better understand what we are trying to achieve. In this subsection, we will focus on the structure of the different types of applications, but mostly on the back-end since it will be the main point of connection between the different components of the JxAppDev as well as the possible exiting Java applications that will be adapted using the framework.

Web Applications

The structure of web applications is quite complex since it is based on a full-stack architecture. In this part we won't go into details about a full-stack architecture but there are normally at least three components present, the user interface that will be displayed in the browser, the *Web Server* where the logic stays and a Database where persistence resides.

Here we are interested in the Web Server since it is the component where the main changes can be observed in comparison to native applications.

The user interface is built with web technologies such as HTML, CSS and JavaScript and usually runs on a Browser. A similar approach is proposed in our hybrid applications since user interfaces will also be built using those same technologies, the main difference being that the UI will be displayed inside the desktop application instead of the web browser.

On the other hand the Web Server layer is not going to be the same on both sides. Most important, we should take note that desktop applications do not generally use Web Servers to handle the logic of the applications. Web applications are based on services. When a user clicks a button, a request is sent to the Web Server from the UI and it will then be treated and

served back to the user interface where it becomes possible to see the expected outcome. In order for this to happen an Internet connection is required. This type of behaviour is not present on native applications since they are based on local access. Generally the application is a complete package written in a single programming language, where logic, UI and persistence live together.

These multiple layers can all exist separately on different machines, without interfering with each other. Every layer may be written with a different programming language, the only requirement being that developers make the effort for all those components to properly work together in the end.

There is a huge list of Web Servers available, but there are three of them, TomCat (Vukotic and Goodwill, 2011), Glassfish (Kou, 2009) and JBoss (Marrs and Davis, 2005) which are the most popular, well maintained and that properly work with Java web applications.

It is also important to mention that there is a version of Java applications that suits better when used with Web Servers. These applications are called Java EE applications. Java EE is designed for companies, due to it being better when creating scalable distributed systems, but also the additional libraries that are included to support database access, remote method invocations, web services, XML processing and many more.

Desktop Applications

Desktop applications are very different from Web Applications. Mostly standard desktop applications are installed in a single machine where all the different parts of the application lay together without requiring an Internet connection to communicate between the different components, although they might communicate with external services to perform certain actions. In this specific case of Java applications we have again a three component architecture. The components present in native applications are the user interface layer, the logic layer and finally the persistence layer. Every component of a native application is in general written in a single programming language, in this specific case Java. All components are usually glued together except for the Database that can in some cases be running in a separate machine. Nevertheless, the communication to the database is made from the application itself using Java programming language (together with JDBC that communicates itself using network protocols). When looking closer, we can see that the UI layer and the logic layer are usually linked and that there is no need for an Application Server to handle the requests made from the UI since all the requests are locally served by the application itself.

Hybrid Applications

This Framework aims to close the gap between desktop native applications and web applications. To solve this problem we require a communication mechanism between the web

user interface created using web technologies and the logic layer of the Java application that will be installed on the desktop computer. In comparison to desktop applications, this new hybrid application created with the JxAppDev will be composed of a local logic layer in Java and a user interface created using only web technologies. This new communication system allows the new web interface to interact with the rest of the Java application, which by itself will allow access to device functionalities and file system, something that was not possible on common web applications. To accomplish such a task, a separate new communication layer needs to be developed in order to properly handle all messages transmitted between the different components.

Embedded Web Servers

An interesting alternative to the integration of a web server with Java applications is the use of an embedded web server. It would facilitate the process of setting the different components of the application in order for them to work properly. There are many well-known embedded web servers, such as Embedded Tomcat (Heroku) and Jetty (Eclipse). In theory, the idea of using an embedded web server seems a viable solution. There are however a few inconveniences that made us drop this idea for the moment being. First, embedded web servers require specific source code to be set and sometimes it takes quite a lot of time to make them work properly. Secondly, it would force us to choose a specific web server, which would be very difficult to replace later due to the constraint to completely change our communication layer in order to make everything work properly. Thirdly, embedded web servers can be very practical for single applications but when we have multiple applications running on the same embedded server, or when we have multiple clients accessing the same server, they can become very slow and sometimes unresponsive. External Web Servers are more optimized to server static content which can be very interesting in our case. Finally, and probably the most important issue that caught our attention was that if we do not pay attention, embedded web servers can cause several troubles regarding *ports*. If a web server is installed and configured on a machine to use an IP or Port that is already set for another application or server, some problems may occur and a new configuration is required.

Due to these complications, we decided to avoid using embedded web servers for the moment. Nevertheless, not using embedded web servers for this project does not mean it was a worse choice for our approach, we simply avoided the extra work to integrate a web server since it wasn't justifiable at this point.

Native (Desktop)	Web	Hybrid
Developed in platform specific language	Developed using HTML, CSS and JavaScript	Developed using HTML, CSS and JavaScript + Java
Separate code for each platform	Run in Server Side	Write once run everywhere
Fastest and most responsive experience to users	Speed depending on server, Natural felling	Medium performance compared to Native
Higher investment of time, talent and resources	Save time and money	Save time and money
Higher costs and development time	Faster development requires two teams for front-back end	Faster development cycle, requires knowledge of Java Language
Only locally accessible	Only web accessible	Accessible locally and from the web
Can be installed from the different Stores	Only accessible on the web	Can be installed from the different Stores
Internet Connection for some applications	Requires Internet Connection	Internet Connection for some applications

Table 3.: Comparison between different application types

After this first analysis some conclusions were taken regarding the approach we decided to follow to develop a framework for hybrid applications.
There are some costs while developing hybrid applications such as for example mixing programming languages which requires in some cases learning new technologies, or the loss of some performance if we don't take the proper care to adapt the application for different operating systems. These drawbacks should be taken into consideration, but at the same time this is a solution that should only be used when developing applications that need to be available on different platforms. Since these applications need to be available on multiple platforms, there is already a chance that the application will be developed using multiple programming languages. On the other hand we should also take note that in most cases the development cycle of hybrid application is smaller and can therefore reduce costs compared to the development of multiple applications.

One of the most important reasons to choose hybrid application compared to native and web application, is the possibility to make a single application available not only locally but at the same time through the Internet. We can see in Table 3 some benefits of developing hybrid applications with this framework, such as using Java for the back-end, the reduced time of developing and the development of application which don't require internet connection to run locally.

2.3.5 *Communication Layer*

As mentioned before, our framework will require a communication layer that will handle the interaction between the user interface, that will be developed using Web technologies, and the Java back-end that will contain the logic. There are two possibilities to build this layer. First we can use a Web Service system that will serve the requests coming from the UI. The second possibility is the use of Web Sockets for the data transfer.

Web Services

Web Services are a communication system that usually uses XML (Dick, 2003) as standard communication language to transfer messages over the Internet. This system is built over the HTTP protocol to handle the communication. There are two main types of Web Services.

First we have *SOAP Web Services* (Snell et al., 2001) which are in reality a communication protocol. SOAP has an interface described in a machine-readable format (WSDL) (Graham et al., 2004). The communication uses SOAP-messages which have a specific format and that are sent in XML to specific methods previously defined. In SOAP the request-response mechanism is based on a *Remote Procedure Call* system that is not tied to any communication protocol even though in general HTTP is used. Usually the Web Server is capable of coding and decoding received SOAP messages in POST HTTP messages that will be handled, and a response will be sent back to the client.

Second we have *REST Web Services* (Masse, 2011), a different approach since it is not based on defining a protocol but rather an architecture. This architecture makes resources available through specific URI's. In general, the HTTP protocol is used to perform the required actions over those resources. In some cases, all the HTTP methods are available but sometimes a minor set of those methods can be made available like for example GET and POST methods. All the data transferred is usually in one of the three main formats: HTML, XML or JSON (Bassett, 2015) which offers more flexibility than SOAP solutions at the cost of losing standardization in the communication process.

Web Sockets

Web Sockets have some common aspects to Web Services since their main purpose is to establish communication between a client and a server. Web Sockets are a protocol which offers continuous TCP connections with full duplex communication between client and server. When using a Web Socket there is a first phase of communication called handshake where a synchronization between client and server is established, and only if this handshake is completed can there be a real communication between the two parties. Web Sockets allow the transfer of binary data, but nowadays people tend to prefer using JSON Serialization to convert data into Strings and then send those Strings through Sockets. On the one hand,

it becomes easier to read and handle the data transferred between all the parties involved. On the other hand, Web Sockets are less structured than the REST architecture due to the possibility of sending any type of data using a single URI as access point.

Web Service	WebSocket
HTTP Server	API
Sits on top of HTTP	Sits alongside HTTP
Single Direction Communication	Bidirectional Communication
HTML/XML/JSON	Any Type of Data
Request/Response	Any Direction Communication
Server Response follows Client Request (Only)	Server can send content to Client (at any time)
Higher overhead 10547 milliseconds for 10000 messages	Lower overhead 1019 milliseconds for 10000 messages
”Stateless” (REST)	Stateful
Need Cookies or Other Solutions to Save Connection State	Simpler to Save Connection State
Resource Reference via URI	No Resource Reference
Scales better in some circumstances	Difficult to scale due to continuous connections
Better for low request demand	Low requests demand requires multiple connection open/close
Allows Caching, Routing, Multiplexing, Gzipping	Requires extra definition on top of the WebSockets
More structured Code at start	Requires complete code restructure and planning at start

Table 4.: Web Services vs WebSockets comparison

The functionalities intended for the JxAppDev framework imply a combination of Web Services and Web Sockets to guarantee that the user has a more complete experience. All requests coming from the different clients should be handled by a REST API that will transmit those requests to the proper component capable of handling them. This API will also be in charge of sending the corresponding responses to the different clients. A Web Socket system is needed to handle more specific tasks such as allowing the back-end to send information to the different clients even when no request was made. This can be useful when multiple clients access the same data and therefore notifications can be sent to all clients in order to update themselves. In Table 4 we have a comparison of both technologies where we see the benefits of Web Sockets for bidirectional communication, the stateful connections and the possibility to transfer any type of data. On the other hand we have the benefits of Web Services regarding the use of JSON together with a more structured code, the better scalability and the possibility to reference resources via URI’s.

2.3.6 *Interaction Layer*

Since one of the main goals of this project is to facilitate the integration of a new Web interface with a Java back-end by the intermediary of a framework, we anticipated that the developer should be required to make as few changes ad possible to the back-end code, in order to establish communication between the different components. To solve this problem we searched for possible technologies that could allow our framework to make calls to the Java application previously created without the need to completely define how it works and how it is developed.

Java Reflection

Java Reflection (Forman and Forman, 2005), developed by Oracle (d), has multiple applications in Java. Between all the possible usages two of them are very important. First, Java Reflection offers the possibility to inspect object from a Java program. It is possible to inspect classes, interfaces, fields and methods at runtime without knowing them at compile time. The second and most important functionality offered by the Java Reflection, is the possibility to instantiate objects and invoke their methods, by simply knowing their names. With this last functionality we are able to pass the name of methods and classes that need to be called on the Java application back-end, by simple passing their names from our Web front-ends. This makes the code simpler and decreases the chances of developers to be required to change their code in order to connect the two components of their new hybrid applications. This is a functionality capable of facilitating the integration of two separate components if the developer knows their structure, their architecture and their functionalities.

2.4 ANALYSIS

After researching about the different technologies for all components of the JxAppDev we took some decisions that helped us design a stable architecture.

First, the analyses of the different types of applications, permitted us to clarify that only an hybrid solution would offer the amount of portability and flexibility that we require to make application run on a maximum number of platforms but mostly on the web which wouldn't be possible in any other scenario.

Second, after some time spent reading documentation and testing the different Java GUI toolkits, we decided to use the JxBrowser library. This library is very powerful and flexible. Since we are focusing on having as much flexibility as possible in this project, the API

offered by JxBrowser is a good choice. It offers the possibility to write code in Swing and JavaFX, while having the power of chromium to properly render web pages in any device. The documentation is vast and the support team is very active.

Third, since the integration of JxBrowser with IntelliJ IDEA is well documented and properly maintained we decided to use this IDE for our project.

Forth, in this section, we have seen the two most popular communication technologies that can be integrated in our work. We decided not to look deep into the negative aspects of each one of these technologies since they both work differently, and at the same time they both work very well in different scenarios. At first, we decided to choose Web Services for this project in order to obtain a more structured solution. Web sockets tend to be less organized to implement due to the larger flexibility that they offer as data transfer. Nevertheless, after some tests we were forced to use both technologies in this framework. There are two specific case scenarios to which one of the technologies will benefit this work and therefore we decided to implement each component using a different technology. During the implementation phase this might look too complex, but the end-user will never be in contact with the source code, therefore he will never notice any difference when using the multiple services of the JxAppDev.

The first communication component of this framework is a Web Service that will handle all requests for HTML pages coming from the user interface. This is a unidirectional communication where requests are made to specific URI's and the proper page is sent to the user interface. This maintains a more organized way of requesting web pages and at the same time gives us more flexibility to expand our work later, in case it is necessary. The second component is a Web Socket service that will allow communication in both directions, meaning that the back-end will be able to send information to the user interface even when there is no request made. This will be used in two specific occasions. The first is when we want to navigate between web pages, and the second one when the back-end needs to send information about possible changes made from one user of the applications that should be propagated between users. The choice to use Web Sockets on this specific case is due to it being more efficient when sending big quantities of messages in short periods of time (Gupta) but also due to the flexibility to later change the type of messages transferred by simply changing the way they are treated on the back-end.

Since we will be using two technologies that both work with JSON (*JavaScript Object Notation*), which is one of the most popular data transfer formats, due to its flexibility and it being very light, we decided that for the moment being all messages transmitted by the JxAppDev will be in JSON format.

Finally, after all these decisions we started testing some examples to see how the communication between the components would be done. Since the structure of the messages transmitted from the different clients to the Java Application vary for every application,

it would require the developer to give enough information about his application, for the framework to understand those messages. It would also require the framework to be able to directly interact with the different classes and methods of the different applications. Since this problem persisted for every application, we decided to take a look at Java Reflection which helped us solving this problem. With Java Reflection all this integration was reduced to a simple specification of the existing classes of the applications. The developer is also required in most cases to develop an extra layer of adaptation between his application and the framework, but at the end the communication is much more transparent and flexible than before.

3

JXAPPDEV

3.1 METHODOLOGY

This thesis focuses on developing a framework for hybrid Java desktop applications. This framework will focus on easing the development of multi-platform Java desktop application that can also be accessed via Web browsers. We also focus on facilitating the multi-platform paradigm for already built applications. This way developers can create a single desktop application which can then be easily installed in any system, and by creating a Web front-end, not only those applications will have the same look and feel through the different platforms but it will also be possible to access them via any Web browser from any device without having to completely recreate the complete application multiple times.

This chapter contains the requirements for this project, as well as a detailed specification of the framework with the challenges we encountered. We will present the selected technologies and the approaches that were taken to develop the framework. We will also discuss the approach taken to reach and validate the solution.

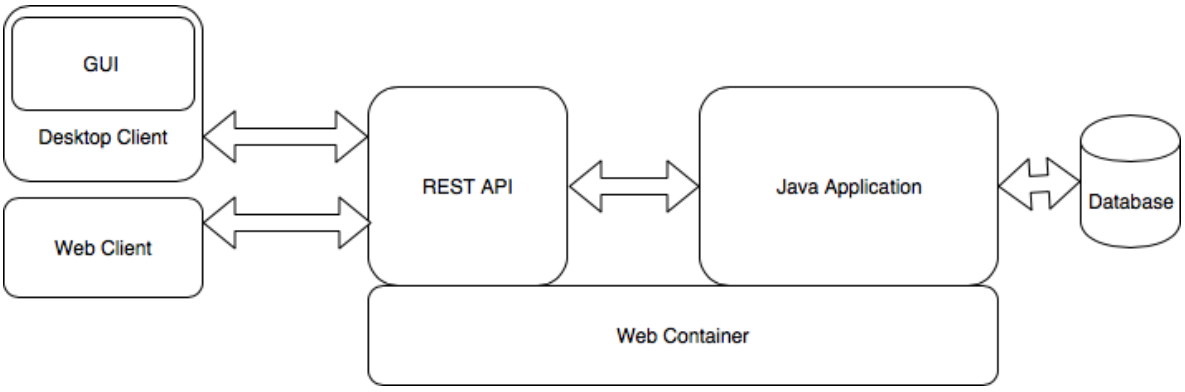


Figure 6.: Framework General Architecture

3.2 SPECIFICATIONS

The JxAppDev framework consists of multiple layers, as presented in Figure 6. To choose which technologies compose the different layers and decide the final architecture for this framework, a complete research of the available technologies was previously made and multiple test cases were developed in order to decide the final path to take.

Before proceeding to the implementation phase, we had to decide on a more general architecture for our framework. This way we were able to focus on missing technologies or even missing components that at first we had never taught were needed or could benefit our development.

Finally, after some tests we decided that this framework is composed of three layers, each representing a specific component required during the development process.

The first layer of the framework is a Java component that is used to display the Web UI inside a Native Desktop application (Desktop Client). This layer is also in charge of handling any compatibility problems between the Native version and the Web version, such as handling sessions for different users.

The second layer is in charge of the connection between the user interface and the Java application back-end. After some analysis, it was concluded that two methods were required to fulfil a normal flow of the application. A REST API for requests between the user interface and the back-end logic, and a Web Socket based solution for a notification system that allows communication between different Application clients.

Finally, the third layer is an Application Server, used to provide access to the back-end.

Alongside these three layers a JavaScript library was developed to facilitate the development of the Web interfaces, but most importantly this library will help in the communication between the front-end and the back-end of the derived hybrid applications. This is due to having two completely different client sources making the same requests to the back-end: the web version which will use the JavaScript library in order to reach the REST API and the Web Sockets, and the native version that will use the JxBrowser library for those same interactions.

All applications developed using JxAppDev will be integrated with all of these components. To these layers we add the Java component where the application's logic and persistence are contained, and the Web technologies based user interface to create the final resulting application. This Web UI will be used for both types of possible deployment, as a native installation, or as a Web Application served by a Web Server.

3.3 REQUIREMENTS

Prior to the development phase, we had defined the requirements needed to support the previously defined objectives, and achieve a viable solution.

First we have a set of requirements for the framework which specify the functionalities of the framework as well as its behaviours.

- The framework must allow to develop Desktop Applications.
- The framework must allow to develop Web Applications.
- The framework must allow to adapt Desktop Applications into Web Applications
- The Framework must support the development of Java applications
- The Framework must contain a JavaScript library to facilitate Web front-end development
- The front-end JavaScript library must facilitate communication between the user interface and the back-end
- Java application integration should not require the user to manipulate any framework layer
- The Framework Middle Layer must be composed of a Java REST Web Service and a Web Socket system
- The desktop version of the frond-end must be developed in Java using the JxBrowser library
- The framework must be capable of distinguishing calls from the Web browser and the desktop application

Since the scope of the JxAppDev is very specific, we are not supporting all types of application. Some requirements were necessary to specify which types of application this framework supports and what characteristic they should have.

- Applications developed with the framework must use asynchronous communication
- Applications developed with the framework must be created using a multilayer approach
- The framework does not support real-time applications, such as streaming services
- The framework does not support intensive graphical applications such as games
- Applications developed with the framework must provide the proper facades for the functionalities they want to provide

3.4 ARCHITECTURE

Before deciding how to build the different components of the JxAppDev framework we had to do some preliminary tests with the approaches and technologies analysed in the previous chapter in order to see which ones were the most efficient and suited best our approach. After some attempts, we were finally able to decide which ones were the most convenient do develop the JxAppDev and established a base architecture that can guarantee the minimal requirements previously established.

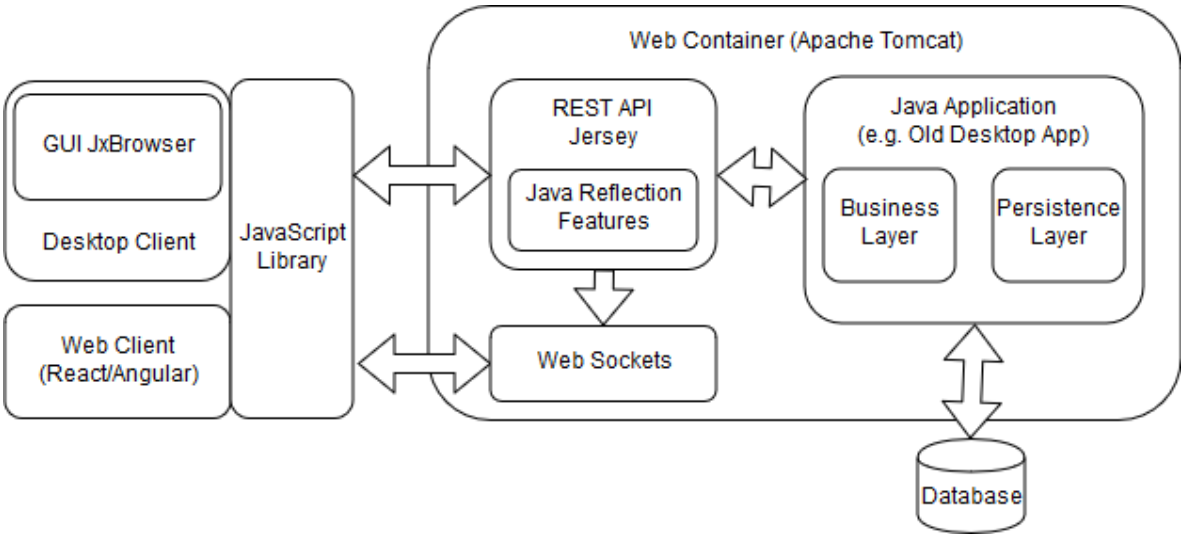


Figure 7.: JxAppDev framework detailed architecture

The architecture of choice is composed of three layers, each representing a specific component required during the development process. All application developed using JxAppDev will be integrated with all of these components. To these layers we add the Java component where the application's logic and persistence are contained, and the Web technologies based user interface to create the final resulting application. This Web UI will be used for both types of possible deployment, those being, as a native installation, or as a common Web Application served by a Web Server.

The server used is Apache Tomcat 7, a stable and well supported Java application server. Its role is to provide access to the REST API.

This architecture can be seen in Figure 7 and all its components are listed above where we detail their implementation, the functionalities they offer as well as the role they play, both inside the framework and the derived hybrid applications. If we take a look at the architecture in Figure 7 we can see the relation to the previous architecture in Figure 6. On the back-end we still have the Java Application which contains the business logic of the application as well as the connection to the possible databases. We also have the Web

container that will provide the application. Finally, on the back-end we can see two changes. First, the REST API now contains the Java Reflection feature that is used to interact with the Java Application. Second we have a separate component that will handle the Web Socket notification system. On the front-end we still have two clients, the first one being the Web client that does not change, and the second one being the Desktop client which now contains the JxBrowser component to server the GUI. Finally on the front-end we have the new JavaScript library that will be used to communicate with the back-end.

3.5 USER INTERFACE

There are two alternatives considered to deploy the user interface. The first, and more conventional one, is by the means of the Web Server (Figure 7, Web Client). This is done by installing a Web server and properly set the server to display the UI. This works as a Web application. Using this method, we are able to access the application from any Web Browser, which will normally make requests to the Web server for new web pages.

The second consists in deploying the interface in the desktop (Figure 7, Desktop Client). In this case the application is installed locally and the JxBrowser will serve the required pages. JxBrowser is capable of serving those pages inside a Web component that we create when the application is launched. All the requests for a new page come directly to the JxBrowser application that will select the proper page to be displayed and send it to the Web container.

In both cases the use of jQuery, Bootstrap and other libraries is automatically handled, either by the browser or JxBrowser.

3.6 JAVASCRIPT LIBRARY

The communication with the REST API can be done from two different sources: the Web Browser or the Java application created with JxBrowser. To facilitate the use of the JxAppDev framework, a JavaScript library (Figure 7, JavaScript Library) was developed. This library allows all requests between components to be as transparent as possible.

This way, when the user interface wants to communicate with the back-end, it only needs to call the proper methods from the JavaScript library.

The developer needs to add a ”div” component on every page with the id ”UserAgent” (Figure 8), that will be used to distinguish the source of the request.

```html
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<div id="UserAgent"></div>
</body>
</html>
```

Figure 8.: User Agent inside an HTML page

The JS Library receives a generic request from the user interface, analyses the origin using the User-Agent. Depending on the value found, which is different on both versions of the application, we can distinguish if the request comes from the Web Browser or the Native Application (Figure 9).

```javascript
function createRequest(json, class_name, method_name, callback, notificationGroup, notificationName){
    // Retrieve the User-Agent value from the Web page
    var userAgent = document.getElementById('UserAgent').value = navigator.userAgent;
    // Create a JSON object with the Class and Method name to be called on the back-end
    var jsonOb = null;
    jsonOb = '[{"className" : "' + class_name + '", "methodName" : "' + method_name + '"},';
    // Add the JSON parameters received from the UI that are required by the back-end methods
    if(json.length !== 0){
        jsonOb = jsonOb + json + ',';
    }else{
        jsonOb = jsonOb + '{},';
    }
    // In case a specific group needs to be notifyed we will add that information to the JSON object
    if(notificationName.length !== 0 && notificationGroup.length !== 0){
        jsonOb = jsonOb + '{"notificationGroup" : "' + notificationGroup +
        '", "notificationName" : "' + notificationName + '"}';
    }else{
        jsonOb = jsonOb + '{}';
    }
    // Close the JSON object
    jsonOb = jsonOb + ']';
    // Verify in wich case we are and call the proper method to send the JSON object
    var response = "";
    if(userAgent === "WebView"){
        response = sendFromWebView(jsonOb, callback);
    }else{
        response = sendFromWeb(jsonOb, callback);
    }
    // Restur the response to the JavaScript method that initiated the call
    return response;
}
```

Figure 9.: Generic function that redirects requests using the proper method

After analysis, the request is sent using the proper method: directly to the REST API in the web browser scenario, or to the JxBrowser application (Figure 10) which will then communicate with the REST API in the native application scenario. This process offers extra transparency to the system itself and facilitates the development of new Web Interfaces since all requests are always transmitted to the JavaScript library and never directly to the multiple components of the system. We can also see in Figure 9 that this generic request receives a call-back parameter. This is the name of a call-back method that should be called when the JS library receives the response from the REST API. An example of such use case is when the developer wants to update some component on the UI, after the request is handled by the back-end. The developer just passes the name of the method that will handle the UI update as a callback parameter, and it will be automatically invoked after the requests is treated.

```
1  /*
2   * This method is in charge of calling the proper method
3   * inside the JxBrowser Application
4  */
5  function sendFromWebView(jsonOb, callback){
6      var response = window.java.callMethod(jsonOb);
7      if(callback != null) {
8          callback(response);
9      }
10 }
11 /*
12  * This method is in charge of using an Ajax call to send
13  * the request directly to the REST API on the Back-End
14 */
15 function sendFromWeb(jsonOb, callback){
16     var response = "";
17     $.ajax({
18         type: "POST",
19         url: "https://ea8b3bdf.ngrok.io/request",
20         data: jsonOb,
21         contentType: "application/json; charset=utf-8",
22         dataType: "json",
23         success: function(data){
24             if(callback != null){callback(data);}
25         },
26         failure: function(errMsg) {alert(errMsg);}
27     });
28 }
```

Figure 10.: Request origin distinction

For a better understanding of the flow of an application developed using the JxAppDev framework, we included two sequence diagrams that illustrates two clients making calls to a back-end application where concurrency is available. These example should illustrate the path made by a request coming from the front-end until the response comes back, as well as demonstrate how the application can inform other clients about the changes

that were made. If we compare both sequence diagrams we can see that they are almost identical. In the native application flow (Figure 11) the communication starts with an initial request that goes from the UI to the JavaScript library. Then this message is transmitted to the JxBrowser that will itself transmit the content to the REST API. Finally, the REST API extracts all the data from the request and makes the proper calls on the Java application (containing the applications logic). In comparison, the Web version (Figure 12) skips the JxBrowser components going directly from the JavaScript library to the REST API. The rest of the communication process remains the same.

We can also see in both diagrams two users registering to the notification systems with a request that goes from the JS library to the Web Socket component. This is used after the main request is treated by the Java application. The REST API will inform the Web Socket systems that there is a notification to be sent to a specific group. When this information returns to the JavaScript library it is transmitted to the user that can perform some tasks with the received data, such as update the UI.

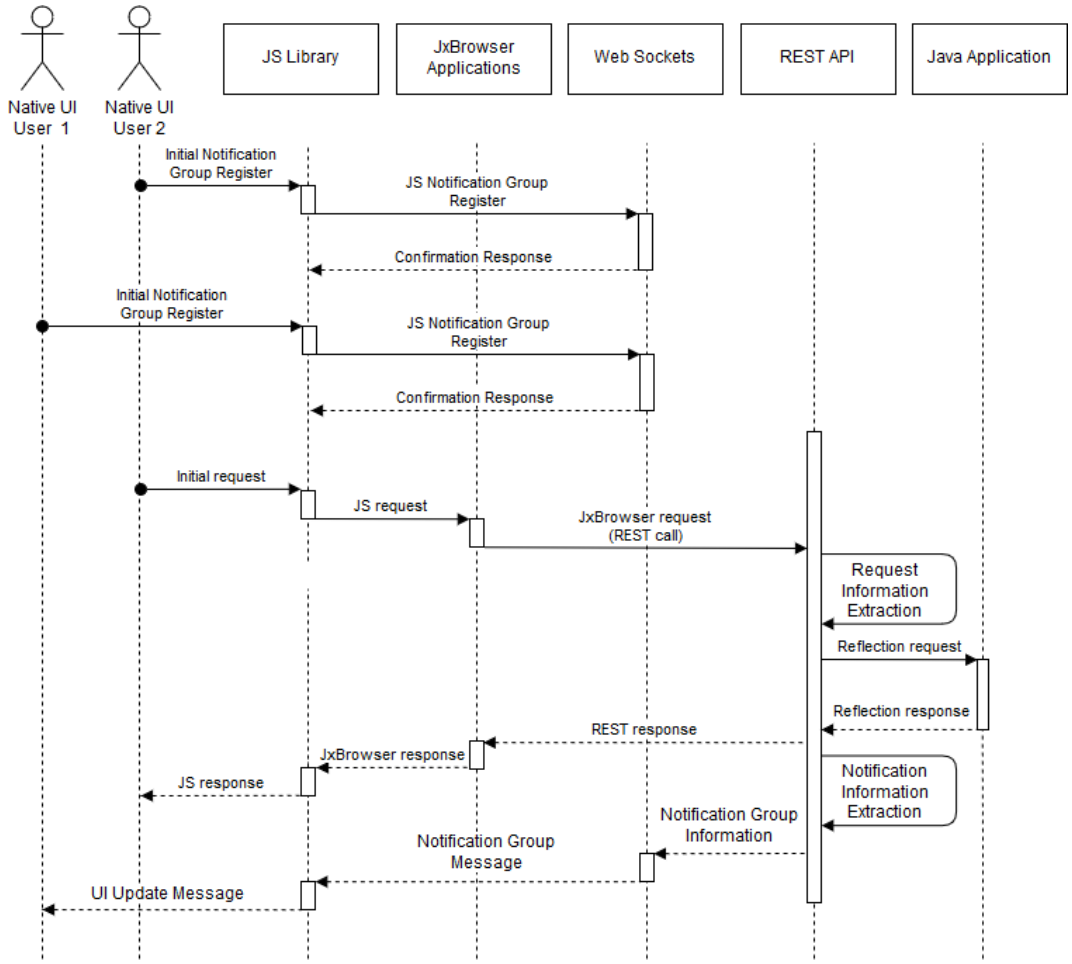


Figure 11.: Sequence diagram of a Native application flow

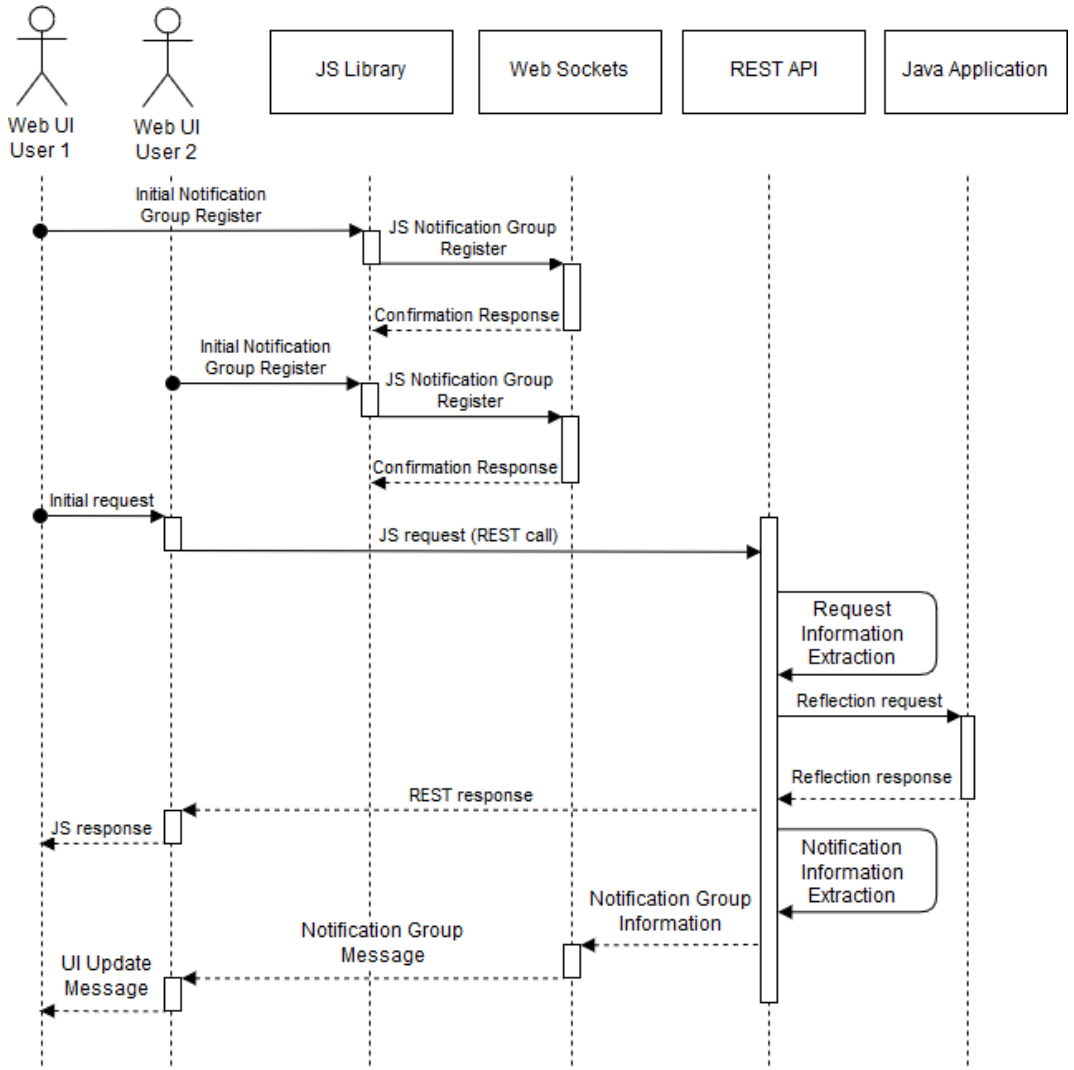


Figure 12.: Sequence diagram of a Web application flow

3.7 REST API

The framework offers an API (Figure 7, REST API Jersey) composed of two essential parts. First we have a component that will handle the requests coming from the front-end. Since all the communication is made using JSON, this component extracts the information from the different JSON Objects inside every request. These objects should contain the name of the classes and methods to be used on the back-end (Figure 7, Java Application), the parameters required by those methods as well as a notification message and a notification group if required by the request (Figure 13 lines 2-6). Second, we have a component whose main goal is to use the information extracted from the JSON object to call the proper methods on the developers application (Figure 7, Java Application) using Java Reflection. This

component facilitates the integration of existing applications into the JxAppDev back-end since it creates an extra transparency layer. The developer will never interact or change the code on the REST API, since he only needs to know the classes and methods he created on his Java application. The back-end of the JxAppDev framework receives calls coming from different clients, these requests are analysed and through the means of the Java Reflection feature we are able to instantiate the classes that we need on the Java Application, list their methods and use them to properly treat the requests. When the Java Application finishes treating a request, the response is sent to the JxAppDev back-end that will be in charge of transmitting it back to the different clients. Without the use of Java Reflection we wouldn't be able to make calls on the Java Application by simply having the methods name, and a more complex integration between the application and the framework would be required.

Nevertheless, to properly integrate a Java application with the framework, the developer needs to create an extra layer on his application that will serve as entry point between the Java Reflection and the classes on his application. This new layer will not only facilitate the communication between the two components but at the same time it will avoid that extra changes have to be made to the Java Application, that can in some cases be an already existing application and therefore changes might be complicated and undesired.

After the REST API extracts the information inside the JSON object coming from the front-end, we use Java Reflection to instantiate the classes from the new layer created on the Java application and call the proper method to complete the task at hand (Figure 13 lines 7-14). When the method terminates, it will respond back to the REST API that will itself send the information to the front-end to be displayed.

The REST API will also handle verifying if any group message needs to be sent after all tasks are completed and will then call the proper methods for group notification if necessary (Figure 13 lines 15-18).

The most important objectives were to allow the User Interface to communicate with the back-end with minimal implementation and deployment costs. Using Java Reflection features (Figure 7, Java Reflection Features), the Class and Method names to be invoked on the back-end, together with the required parameters for those methods (if necessary), are sent by the JS library to the REST API, which instantiates the required classes and calls the appropriate methods. There are two requirements for the REST API to work properly. First the Java Objects on the back-end must be converted to JSON objects, before being sent to the front-end. This can be considered the major change required from developers to the Java Application. The second requirement is that the developer must specify in a configuration file all the packages and classed of the Java application. This will allow the Java Reflection mechanism to find the required Classes and Methods. This second requirement can be avoided, by passing the complete path (package + class name) to the JS Library instead of only passing the class name. This second method that requires the developer to input

```
public String genericRequest (final JSON json){
    String class_name = json.get("className").getAsString();
    String method_name = json.get("methodName").getAsString();
    String parameters = json.get("parameters").getAsString();
    String notif_group = json.get("not_group").getAsString();
    String notif_name = json.get("not_name").getAsString();
    Reflections refl = new Reflections("Paths.txt", new SubTypesScanner(false));
    Set<Class<?>> classes = refl.getSubTypesOf(Object.class);
    Class clasz = Class.forName(class_name);
    Constructor constructor = clasz.getConstructor();
    Object instance = constructor.newInstance();
    Method[] methods = clasz.getMethods();
    Method methodz = clasz.getDeclaredMethod(method_name, String.class);
    String json_response = (String) methodz.invoke(instance, parameters);
    if(notification_group){
        NotificationService.sendGroupNotification(notif_name, notif_group);
    }
    return json_response;
}
```

Figure 13.: Generic back-end request received and treated by the Java Reflection

more information is more flexible, enabling for example, the possibility of having two classes with the same name in different packages. Something not supported when using the configuration file, since the framework will not be able to distinguish between the different classes with the same name. We should also take into consideration that when using our REST API together with Java Reflection, we are opening a security breach. Users can use brute force by sending multiple requests using different URI's until they reach a valid end-point of the API, being so able to execute some functionality of the application. This problem can be solved when using the configuration file. This file can serve as a filter for the request coming from the front-end and therefore exclude invalid requests. This solution is not implemented in the JxAppDev framework, but since we are passing a token for every user request we can use this same token together with the configuration file to filter all requests and therefore avoid calls to other methods or classes.

3.8 WEB SOCKETS

In some cases, there is a need for the business logic to send information to the user interface (e.g. Observer-Observable pattern) such as notifying the user inter-face about changes made by some client (e.g. concurrency). To solve this problem, the framework supports a Web Socket system (Figure 7, Web Sockets) capable of handling these situations by maintaining a connection that is established through the JavaScript library and which offers the possibility for the Server to communicate with the Client. We created a system where the client can use

the JavaScript library to open a connection and furthermore register to specific notifications groups. The JavaScript library provides methods to register to specific notifications groups (Figure 14) as well as some methods to handle those notification groups on the front-end using the Client local storage (Figure 15).

```
// Register to Single Notification Group
function registerToGroup(groupName){
    client.send("{\"type\":\"register\",\"group\":\"" + groupName +
    "\",\"token\":\"" + token + "\"}");
    addGroupToLocalStorage(groupName);
}

// Unregister from Single Notification Group
function unregisterFromGroup(groupName){
    client.send("{\"type\":\"unregister\",\"group\":\"" + groupName +
    "\",\"token\":\"" + token + "\"}")
    removeGroupFromLocalStorage(groupName);
}

// Register to Multiple Notification Groups
function registerToGroups(groups){
    var jsonString = "{\"type\":\"multiregister\", \"groups\" : [";
    var i;
    for (i = 0; i < groups.length; i++) {
        jsonString = jsonString + groups[i] + ",";
        addGroupToLocalStorage(groups[i]);
    }
    jsonString = jsonString.substring(0, jsonString.length - 1);
    jsonString = jsonString +  "], \"token\":\"" + token + "\"}";
    client.send(jsonString);
}

// Unregister from Multiple Notification Groups
function unregisterFromGroups(groups){
    var jsonString = "{\"type\":\"multiunregister\", \"groups\" : [";
    var i;
    for (i = 0; i < groups.length; i++) {
        jsonString = jsonString + groups[i] + ",";
        removeGroupFromLocalStorage(groups[i])
    }
    jsonString = jsonString.substring(0, jsonString.length - 1);
    jsonString = jsonString +  "], \"token\":\"" + token + "\"}";
    client.send(jsonString);
}

// Unregister from all Groups
function unregisterFromAllGroups(){
    var jsonString = "{\"type\":\"unregisterall\", \"token\":\"" + token + "\"}";
    client.send(jsonString);
    removeAllGroupsFromLocalStorage();
}
```

Figure 14.: Methods to handle groups for notifications

```
1  // Add group to local storage
2  function addGroupToLocalStorage(newGroup){
3      var storedGroups = JSON.parse(localStorage.getItem("groups"));
4      if(storedGroups == null){
5          storedGroups = [];
6      }
7      if(storedGroups.indexOf(newGroup) === -1){
8          storedGroups.push(newGroup);
9          localStorage.setItem("groups", JSON.stringify(storedGroups));
10     }
11 }
12
13 // Remove group from local storage
14 function removeGroupFromLocalStorage(oldGroup){
15     var storedGroups = JSON.parse(localStorage.getItem("groups"));
16     var index = storedGroups.indexOf(oldGroup);
17     while (index > -1) {
18         storedGroups.splice(index, 1);
19         index = storedGroups.indexOf(oldGroup);
20     }
21     localStorage.setItem("groups", JSON.stringify(storedGroups));
22 }
23
24 // Remove all groups from local storage
25 function removeAllGroupsFromLocalStorage(){
26     localStorage.setItem("groups", JSON.stringify(""));
27 }
28
29 // When moving to another page, register to groups again
30 function registerToPreviousGroups(){
31     var storedGroups = JSON.parse(localStorage.getItem("groups"));
32     if(storedGroups != null){
33         registerToGroups(storedGroups);
34     }
35 }
```

Figure 15.: Methods to store groups information on the front-end

When sending requests, the developer can specify an extra parameter that will be extracted on the back-end REST API (Figure 13). On the Server side when the REST API receives a request that contains a notification parameter, it will automatically extract that information, perform the action corresponding to the request and before communicating back to the Client, it will perform the corresponding notification action which will be transmitted to the registered clients through Web Sockets. This system can be used in cases where there is a multi-client connection. When some data is changed by one user interface, all the clients are notified and can therefore update their UI.

3.9 ACCESS CONTROL

Most Web Applications manage multiple users using the notion of session while Desktop applications mostly focus on single users. Even when authentication mechanisms are used, they are very different from the cookies and tokens used on the Web. To solve this problem the framework has support for users to authenticate using a token system such as JWT (JSON Web Tokens) ((Blokdyk, 2018)). Some methods were added to the framework which allow the native version of the resulting applications to handle JWT as the Web Browser version. This will possibly force the user to develop new authentication systems, but will provide the same type of authentication for all applications, which was not possible before.

3.10 PAGE NAVIGATION

To avoid the need to distinguish between page navigation on the Web browser (achieved through links), and on the Java counterpart, where it must be done through the JxBrowser API, the framework implements a unified mechanism that can be used on both cases. This mechanism is composed of three parts. First, a parser that is used to fetch data from a specific file (Figure 17) created by the developer where the page navigation for the application is defined (Figure 16).

```
{
    "page-navigation": [{
            "from-page": "Login",
            "navigation-rules": {
                "login" : "Notes_Page",
                "register" : "Register",
                "cancel" : "/"
            }
        },
        {
            "from-page": "Notes_Page",
            "navigation-rules" : {
                "back" : "Login",
            }
        },
        {
            "from-page": "Register",
            "navigation-rules" : {
                "cancel" : "Login"
            }
        }
    ]
}
```

Figure 16.: Example of a navigation file

Second, we added some methods in our JavaScript library that will detect user clicks on specific HTML button, anchors and input fields (Figure 17). When a click is detected, the user is redirected to the corresponding page according to the data that was previously extracted from the navigation flow file. With this system the developer has two extra changes to do on the application. First, create the navigation file which should be added to the JxAppDev framework and secondly specify the proper button names inside the HTML pages he created for the UI, the same button names that he defined on the navigation file. Finally, the JavaScript Library also features methods to programmatically perform navigation between pages (Figure 17). These methods simply require the name of the page to which the user should be redirected and they will handle the navigation for both versions of the application.

```
1  // Load the JSON file containing navigation information
2  function parseNavigationFile(){
3      $.getJSON("navigation/navigation.json", function(json) {
4          parseJSON(json);
5      });
6  }
7  // Parse the navigation file and create a map
8  function parseJSON(json){
9      var page_navigation = json["page-navigation"];
10     for (var key in page_navigation) {
11         var navigation_rules = page_navigation[key]["navigation-rules"];
12         for (var key1 in navigation_rules) {
13             if (navigation_rules.hasOwnProperty(key1)) {
14                 pageNavMap[key1] = navigation_rules[key1];
15             }
16         }
17     }
18 }
19 // Navigate to page when button clicked
20 $(document).click(function(e) {
21     if(e.target.name in pageNavMap){
22         navigateTo(pageNavMap[e.target.name]);
23     }
24 });
```

Figure 17.: JavaScript methods that handle page navigation

3.11 SUMMARY

This framework offers the possibility to develop hybrid applications using a Web UI front-end and a Java back-end. This Web UI can be used inside a native Java application or a web browser allowing it to be cross platform as well as be served as a Web application.

Most of the communication is established by the JxAppDev with some few tweaks required by the developer.

First, all request between the Client and the Java application must be done through a JavaScript library that will communicate with a REST API on the back-end using JSON.

Second, the navigation between pages inside the user interface must be done through the JavaScript library using a navigation file provided by the developer. This will allow both versions of the application to properly navigate through the different pages without requiring to identify which version we are using.

Third, the requests coming from the UI need to contain the name and methods of the classes that will be used by the logic of the Java back-end application. For this to work, the developer needs to create an integration layer that will help Java Reflection interact with his application and which will also handle the transformation of JSON Objects that he sends from the UI to Java Objects required by the business logic.

Forth, the developer needs to create a file containing all the packages for the classes of this new integration layer. This will help Java Reflection interact with those classes and methods.

Finally, when developing a multithread application where concurrency between clients is possible, the developer needs to use the Notification system integrated in the JavaScript library. This system will allow information to be transmitted between clients after every task is performed on the back-end.

4

APPLICATION EXAMPLES

As proof of concept we decided to create two application that allow us to test and illustrate the use of the JxAppDev framework. First we have a practical case of hybrid application development. Here we have created an application that contains all the CRUD (Create, Read, Update and Delete) functionalities to demonstrate the capabilities of the framework while developing new applications. Then we have another practical example of an existing Java application that is transformed into a hybrid application. This will help us illustrate the capability of the framework to adapt already existing applications.

4.1 NEWLY DEVELOPED APPLICATION

Our first test case consists of developing a Memo Pad in Java. This application allows authenticated users to access a previously created set of Notes (Figure 20). The user can then see those notes and make three operations on them, being those create, eliminate and modify notes. We started by creating a database to store the different user information as well as their corresponding notes. Then we created some Java classes that handle the communication to the database. These classes are capable of handling the different operation such as create/delete users as well as create/delete/update notes. Since the framework is based on a JSON communication between the front-end and back-end, we decided to create another class that is in charge of serializing the objects extracted from the database and create the corresponding JSON object that can handled by the user interface.

We can see in Figure 18 a class diagram of the Memo Pad application back-end that is divided in two parts. First, a package marked in blue which contains the classes that belong to the framework. Second, marked in red we have a package which contains the classes that belong to the application. We can also see that in this class diagram there are no classes that handle the UI. This is due to the UI being presented by the framework front-end.

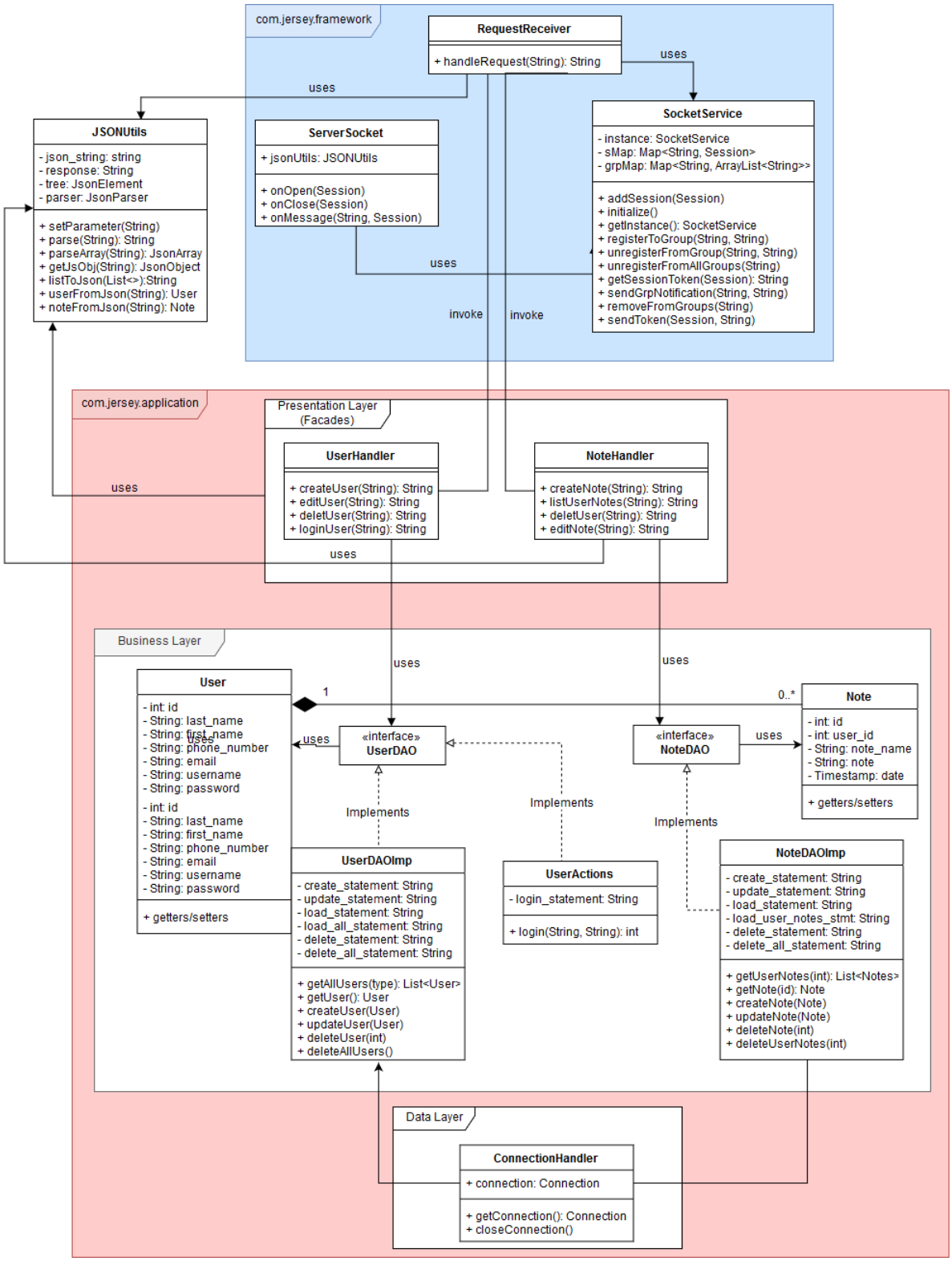


Figure 18.: Memo Pad class diagram

4.1.1 *Framework Usage*

To reach the desired goals we started by creating a user interface using Web technologies. This user interface consists of three HTML pages, a first page for the login (Figure 19), a second one for user registration (Figure 21) and a final one that well list the user notes (Figure 20) as well as offer the possibility to manipulate those notes. The next step was to use the new Web interface together with the JavaScript library of the JxAppDev framework. All requests coming from the UI will call a JavaScript method to which we pass the name of the classes and methods that we desire to invoke on the native Java back-end application.

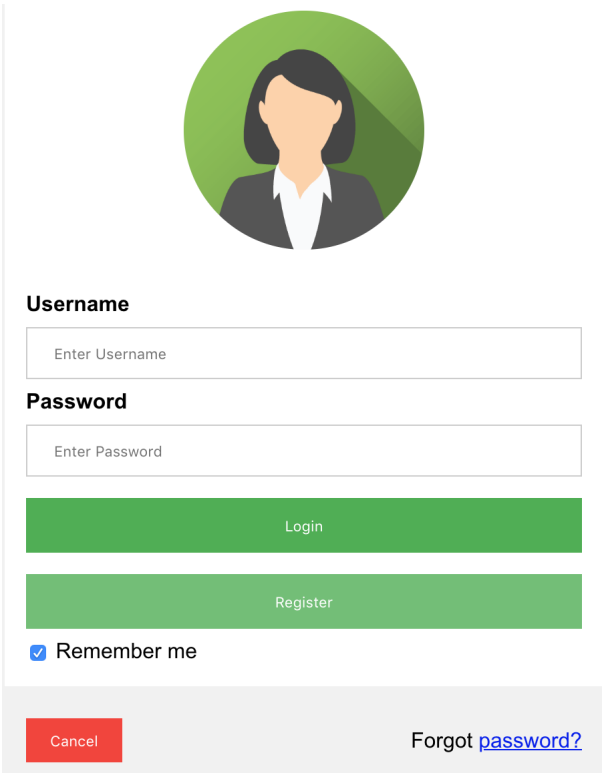


Figure 19.: *Login page* for the Memo Pad application

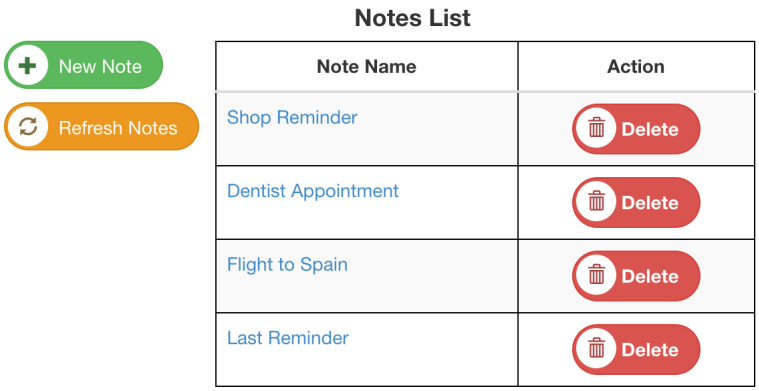


Figure 20.: *Notes list page* for the Memo Pad application

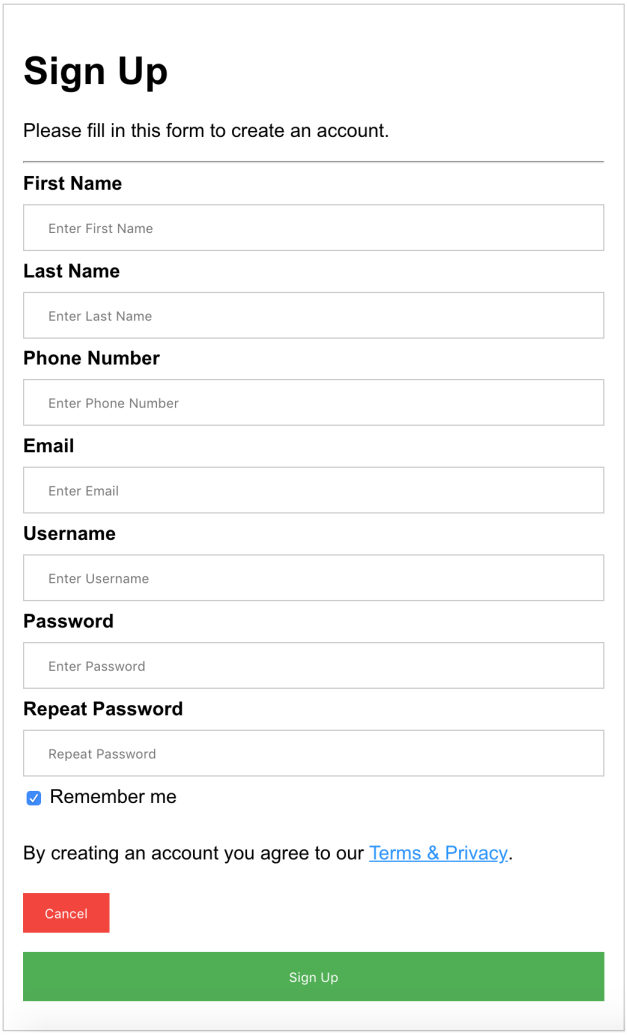


Figure 21.: *Register user page* for the Memo Pad application

The process is illustrated in (Figure 22) where the user simply calls the generic `createRequest()` method from the JavaScript library passing all the arguments such as the class to be instantiated on the back-end, method to be called and a JSON containing the parameters for the method call. There are three other arguments that can be used: a call-back function to be invoked after the request is completely processed by the system, a group name to be notified and the corresponding notification message to be transmitted to all registered clients. This information is communicated from the user interface to the business logic by the framework.

```
1  // List all notes by User
2  function listNotes(){
3      var json = '{"user_id": "' + user_id + '"}';
4      createRequest(json, "NoteHandler", "listNotesByUser",
5      handleTable, "", "");
6  }
7  // Update some table
8  function handleTable(response){
9      var table = document.getElementById("notetablebody");
10     clearTable();
11     for (x in response) {
12         addRow(table, response[x]);
13     }
14 }
```

Figure 22.: JavaScript library generic request call by a JavaScript method (e.g. List Notes)

On the back-end, the business logic of the application was integrated with the JxAppDev framework. Three tasks were required to guarantee the transfer of data from the UI to the business logic and the proper flow of the derived applications. First, all Java objects need to be converted into JSON. This way the framework is capable of transferring the data to the UI where it can be read and treated by the developer. To solve this first problem, we used the Jackson API which allows to convert Java objects into JSON objects. Any technology that performs this conversion can be used. The only requirement is that all the communication between the business logic and the user interface must be done using JSON objects, since it is the only data type that can be understood by all framework components.

The second task was to give enough information to the framework, so that Java Reflection can access all the classes and methods from the business logic. This can be done in two different ways. The first one, used in this example, consists in creating a ”paths.txt” file where all the packages containing the different classes from the business logic are listed. This allows Java Reflection to detect those classes and therefore use them as well as their methods. A simplified example of using such file is illustrated by the ”List Notes” request mentioned before (Figure 13, page 42). All the information is extracted from the received JSON, the proper class is instantiated using the ”paths.txt” file (Figure 23) and the proper

method is invoked. This solution works for small applications, but can in some cases cause trouble. If for some reason there are two classes with the same name in different packages, it can induce error. To avoid this problem it is also possible to send from the user interface, the complete path to the class that should be instantiated (including packages) by Java Reflection. This avoids the use of the "paths.txt" file, but requires the developer to specify the proper packages, on the UI, for every request, which can sometimes be less convenient. In both cases after receiving the response from the business logic, the framework will perform an extra task. It will verify if there was any notification group specified in the received JSON object (Figure 13, page 42). If so, the corresponding notification message is sent to all group members.

```
1  com.jersey.app.database
2  com.jersey.app.notes
3  com.jersey.app.user
4  com.jersey.app.helpers
```

Figure 23.: Example of a paths.txt file

The third and final task consisted in adding a navigation file (Figure 16, page 47). This file contains a list of all buttons in the user interface that perform page navigation, together with the name of the corresponding pages that should be displayed when clicking each of the buttons. In this example the list was composed of only three buttons, used to navigate between the login, register and notes pages.

4.2 ADAPTING EXISTING APPLICATION

The second test case consists of adapting an already existing application using the JxAppDev framework. The application we chose is called IVY Editor which is a component of the IVY Workbench application (HASLab). IVY Workbench is a model based tool for the analysis of interactive systems design. This tool contains a text Editor that allows the user to describe models and which will then compile these models and verify their validity. The main functionalities of the editor are to describe a model using text and manipulate the text with simple functionalities such as undo/redo actions, open/save files as well as a compile function that will analyse the described model and display the result of the analysis in a separate text box. In this example we used the IVY Workbench application that was developed in Java as back-end, and created a new text editor component. The existing text editor was developed using JavaFX and is only accessible locally in a computer where it was previously installed. The objective of this example was to transform this application, making it accessible from a Web browser lying on the machine where the native application was installed, but also through other machines. There are two main goals that we achieve while

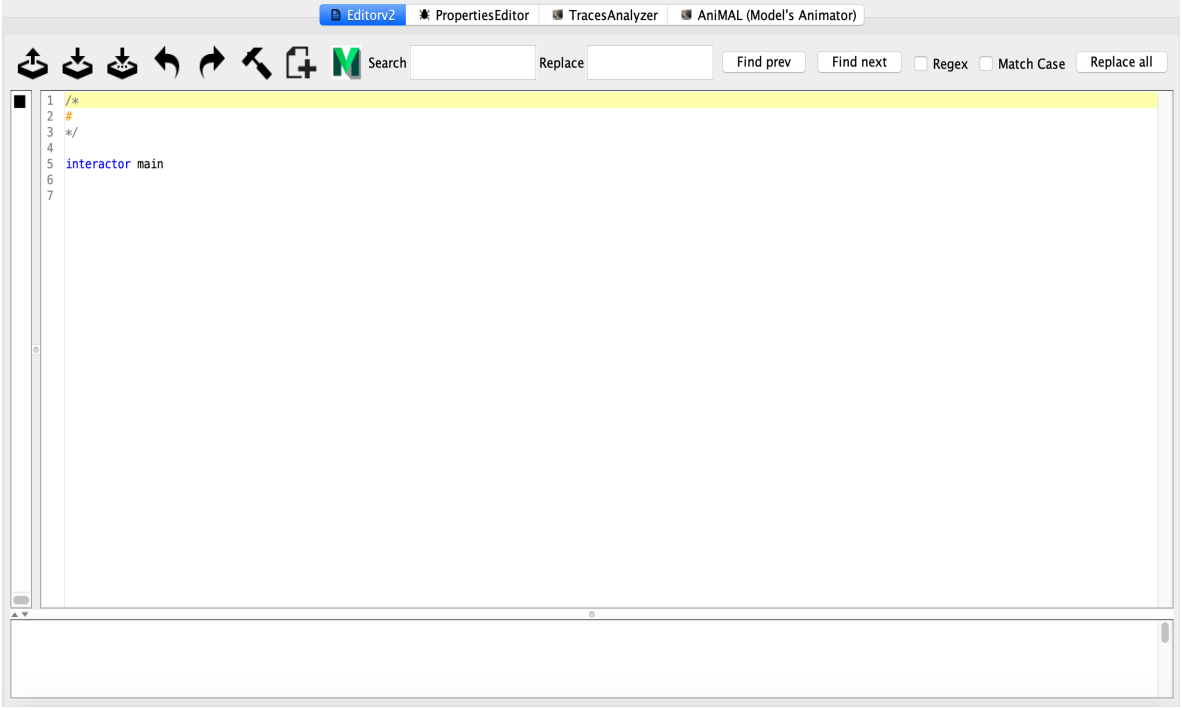


Figure 24.: IVY Editor original Java Swing user interface

developing this application. First we wanted to be able to locally access the application using a web browser. Second, we wanted to be able to port the application to a different machine, from where we will manipulate the data that was only accessible locally with the native application.

4.2.1 *Framework Usage*

The IVY Editor, previous native UI was developed with Java Swing (Figure 24). As in the previous use case we started by creating a new interface using Web technologies (Figure 25). This new interface represents a similar version of the user interface, but will now be used for both versions of the application, native and web.

Since we are developing a text editor we decided to use a JavaScript code editor on the front-end that offers functionalities such as opening files, saving files, undo some action, search and replace words. This editor is called ACE (Cloud9) and is maintained by Cloud9 in collaboration with Mozilla. It offers the main operation that we need for our IVY editor and is very flexible in terms of customization. With this Web code editor we were able to create a new Web UI for the IVY Editor without spending too much time on details that do not contribute to demonstrate the capabilities of the JxAppDev framework.

The next step was to use the new Web interface together with the JavaScript library of the JxAppDev framework. The major change is that instead of directly invoking Java methods

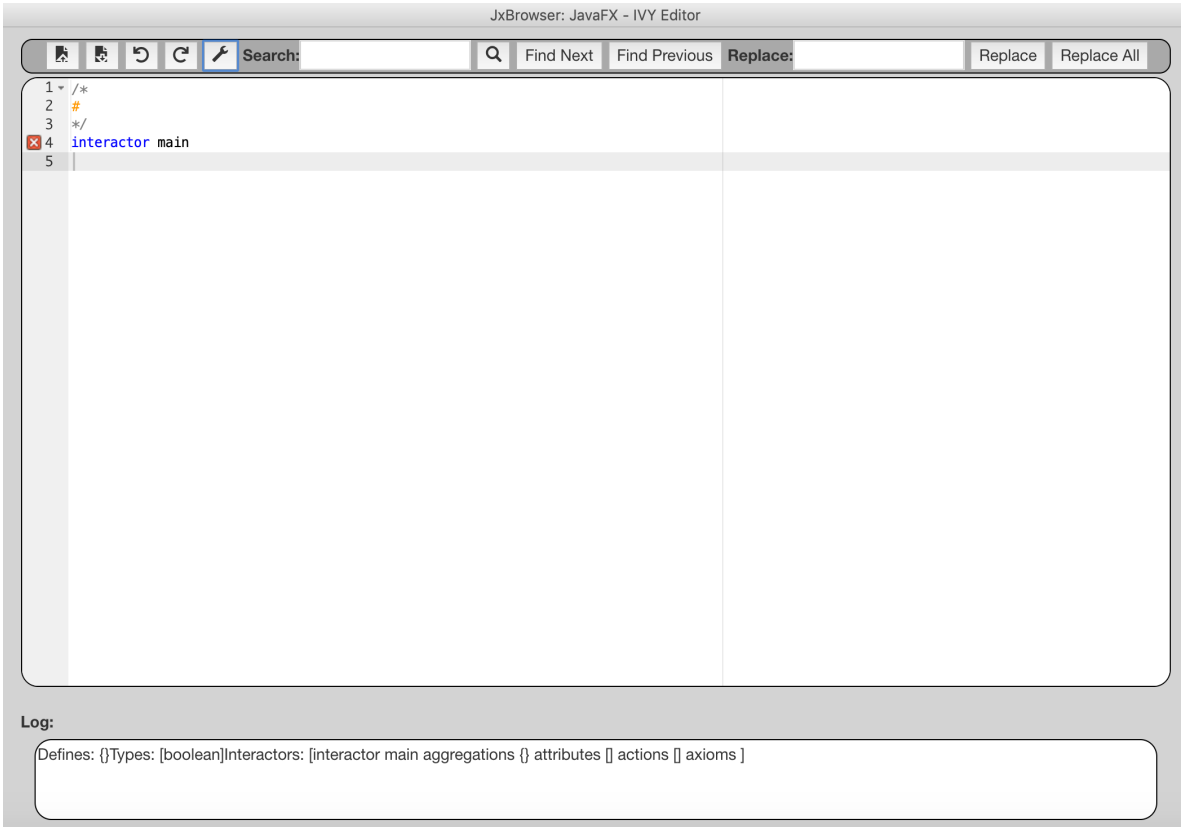


Figure 25.: IVY Editor user interface for both version JxBrowser and Web

from the Java business logic, we rather call a JavaScript method to which we pass the name of the classes and methods that we desire to invoke on the native Java back-end application.

As in the previous example we can now make calls to the generic `createRequest()` method from the JavaScript library (Figure 26) and pass the required arguments such as the class and methods to be invoked on the back-end, as well as the call back method "setLog()" that will handle the response from the compiler and display the response message in the appropriate text box.

On the back-end, the business logic of the application was integrated with the JxAppDev framework. Two main changes were required to guarantee the transfer of data from the UI to the business logic and the proper flow of the derived applications. First, we started by creating a new class ("ModelCompiler" Figure 27) on the Java back-end that will separate the "compile" functionality of the IVY Workbench but also handle the transformation between Java Objects that are used by IVY to JSON objects that can be used by the JxAppDev and the other way around.

```javascript
/*
 * This method will extract the text from the Editor Window,
 * create a JSON object with that value and finlly call the
 * function that will send a request to the back-end.
 */
function compileModel(textModel){
    var json = '{"text_model":"' + textModel + '"}';
    createRequest(json, "ModelCompiler", "compile_model", setLog, "", "");
}
/*
 * This method will update the log box on the UI
 * with the received response from the back-end compiler.
 */
function setLog(response){
    document.getElementById("textarea").value = response.result;
}
```

Figure 26.: JavaScript request for Model Compilation

```java
package com.jersey.app.ivyWorkbench;

import java.util.Scanner;

import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.jersey.app.ivyWorkbench.CoreSystem.ivycompiler.Compiler;

public class ModelCompiler {

    public static String compile_model(String received_json){
        String response = "";
        JsonParser parser = new JsonParser();
        JsonObject jsonObject = parser.parse(received_json).getAsJsonObject();
        String string_model = jsonObject.get("text_model").getAsString();
        Scanner model = new Scanner(string_model);
        try{
            Compiler comp = (Compiler) new Compiler().compileModel(model);
            response = comp.model.toString();
        }catch(Exception e){
            e.printStackTrace();
            response = e.toString().replaceAll("\"", "'");
        }

        return "{\"result\": \""+response+"\"}";
    }
}
```

Figure 27.: ”ModelCompiler” class on the back-end application

The second change was to give enough information to the framework, so that Java Reflection can access all the classes and methods from the business logic. For this we created again a ”paths.txt” file (Figure 28) where we added the package containing the Java class

that we just created on the IVY Workbench. With this change we can again use Java Reflection to instantiate that class and call the compile method.

```
1   com.jersey.app.ivyWorkbench
```

Figure 28.: Paths file containing the package to the IVY Workbench

In this case since no other UI client needs to be informed about the task that was just performed, the framework won't send any notification to other clients, and will simply respond back to the UI with the result of the performed compilation.

Since this application consists of a single window with a text editor there is no need to create a navigation file.

After all the changes, we decided to calculate the number of lines of code that were added, to better estimate the cost of using the JxAppDev in terms of work. On the back-end, the IVY workbench went from 7634 lines to 7662 lines (an increase of 0.37%). We should take into consideration that the code base includes now the framework's back-end (421 more lines of code). On the front-end, the old Java Swing version contains 180 lines of code, compared to the new version which contains a total of 231 lines of code (an increase of 28%). There are 406 more lines of code, but those belong to the JavaScrip library and the JxBrowser. We should also consider that these numbers do not include both libraries used for code editing on the Java application as well as its JavaScript counterpart on the hybrid application. We can see a comparison of the number of lines before and after the integration of the IVY Workbench in the new editor in Figure 29.

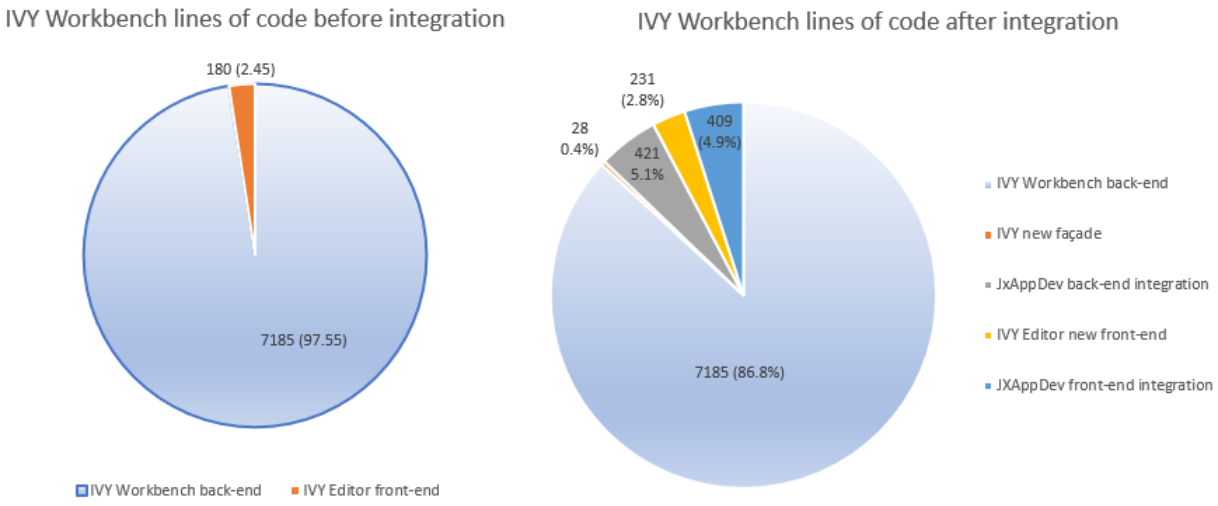


Figure 29.: Lines of code comparison before and after integration of the IVY Workbench

4.2.2 *Application flow*

With these changes the application is complete, we can launch the new application and edit the models we want. After editing a new model, we click the compile button that will invoke a JavaScript method which will extract the content from our editor and invoke the ”createRequest()” method from the JxAppDev JavaScript library. This method will create a JSON with all parameters and will send this information to the REST API. On the REST API we will extract the class and methods to invoke on the IVY compiler as well as the text coming from our UI editor that comes as an extra parameter inside the JSON. Finally with Java Reflection we will instantiate the ”ModelCompiler” class and invoke the ”compile_method()” with the text that we just extracted from the JSON coming from the UI.

The ”compile_method” will use the IVY compiler to analyze and verify the model. Depending on the result, it will create a returning JSON response that will be passed back to the REST API.

When receiving the response, the REST API can send the information to the front-end JavaScript library that will invoke the call-back method ”setLog()” that was specified when creating the request. This method will then take the response and insert it into the Log box on the user interface.

A sequence diagram illustrating these steps can be seen in Figure 30.

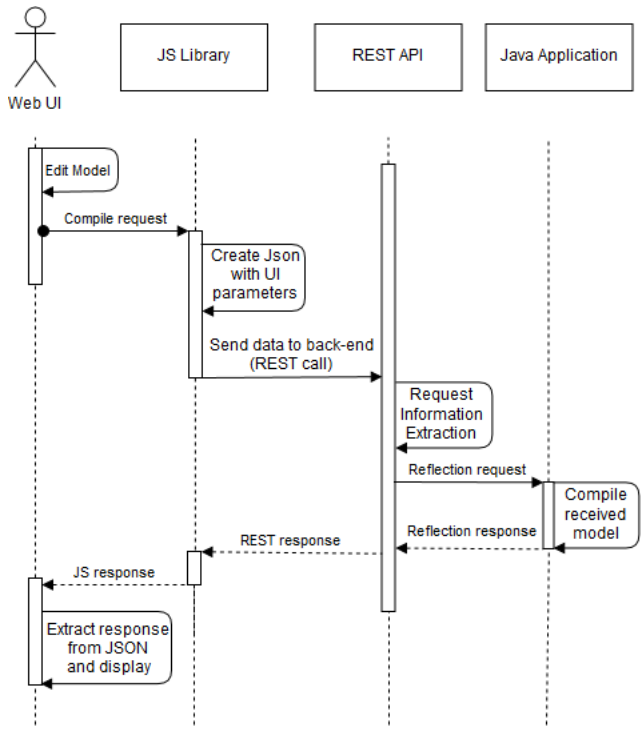


Figure 30.: Sequence diagram for the IVY Editor on the Web version

4.3 PERFORMED TESTS

After developing both use cases, we decided to perform some tests that helped us verify that everything worked as expected and that all functionalities were delivering the desired results.

4.3.1 *Use case New Application*

After developing the first application (Memo Pad), we gathered a set of four tasks (login, create/delete/edit multiple notes), that were performed on the different versions of the application. This set of tasks was performed locally on the desktop application, then we proceeded to perform those same tests on the web browser version of the application, and finally we created a ”jar” file that was exported to a different machine where the same set of tests was again performed. These tests demonstrated that portability between machines and to the web browser worked as expected. By performing the login task on all versions we concluded that the navigation system was working properly since we were always redirected to the next page of the application. Finally a test was performed to verify that the notification system worked. For this test we logged in with the same user account into the web browser and the remote machine at the same time. When performing basic actions on the application such as deleting or creating notes, the information was propagated to the other client which was demonstrated by an update on the user interface. This way we were able to prove that all the minimal requirements of the JxAppDev were achieved and that everything was functional in the described scenario.

4.3.2 *Use case Adapted Application*

After adapting the second application (IVY Editor), we gathered a set of three tasks (edit model, compile valid model, compile faulty model), that were performed on the old application before being adapted, on the new local application as well as the web browser version. These tests allowed us to compare the old version with the new one to see if the results were the same, and at the same time they allowed us to test if the native version as well as the web version reacted the same way to the different tasks. By performing the edit model task on all versions we were able to see that the different UI editors used were working as expected. Then we proceeded to both ”compilation” tasks that as expected demonstrated that when the model was valid there was a valid response being displayed on the log field, as well as an error message when we tried to compile a faulty model. This allowed us to demonstrate that the minimal requirements for all applications were achieved and that the tasks used in this scenario worked as expected. We can see in Figure 31 the load of a model

in both versions of the application, in Figure 32 the compilation of a valid model and in Figure 33 the compilation of a faulty model. In all the Figures we have on top the native Java Swing version and on the bottom the new Web version of the application.
We should take into consideration that all tests that were performed in both cases are preliminary end-to-end tests that were mainly used to guarantee that the main functionalities were maintained between applications.

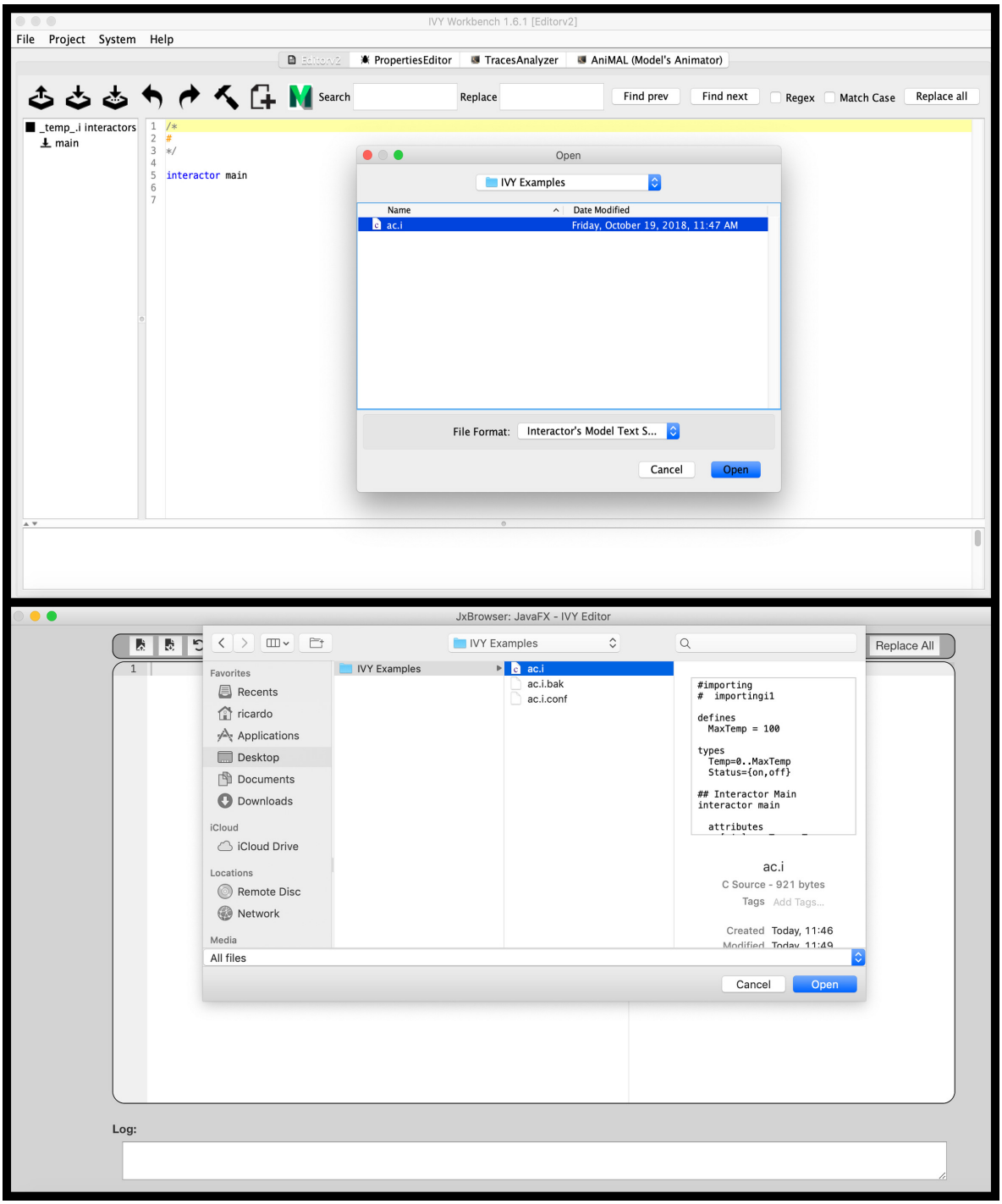


Figure 31.: Sequence of images demonstrating IVY usage

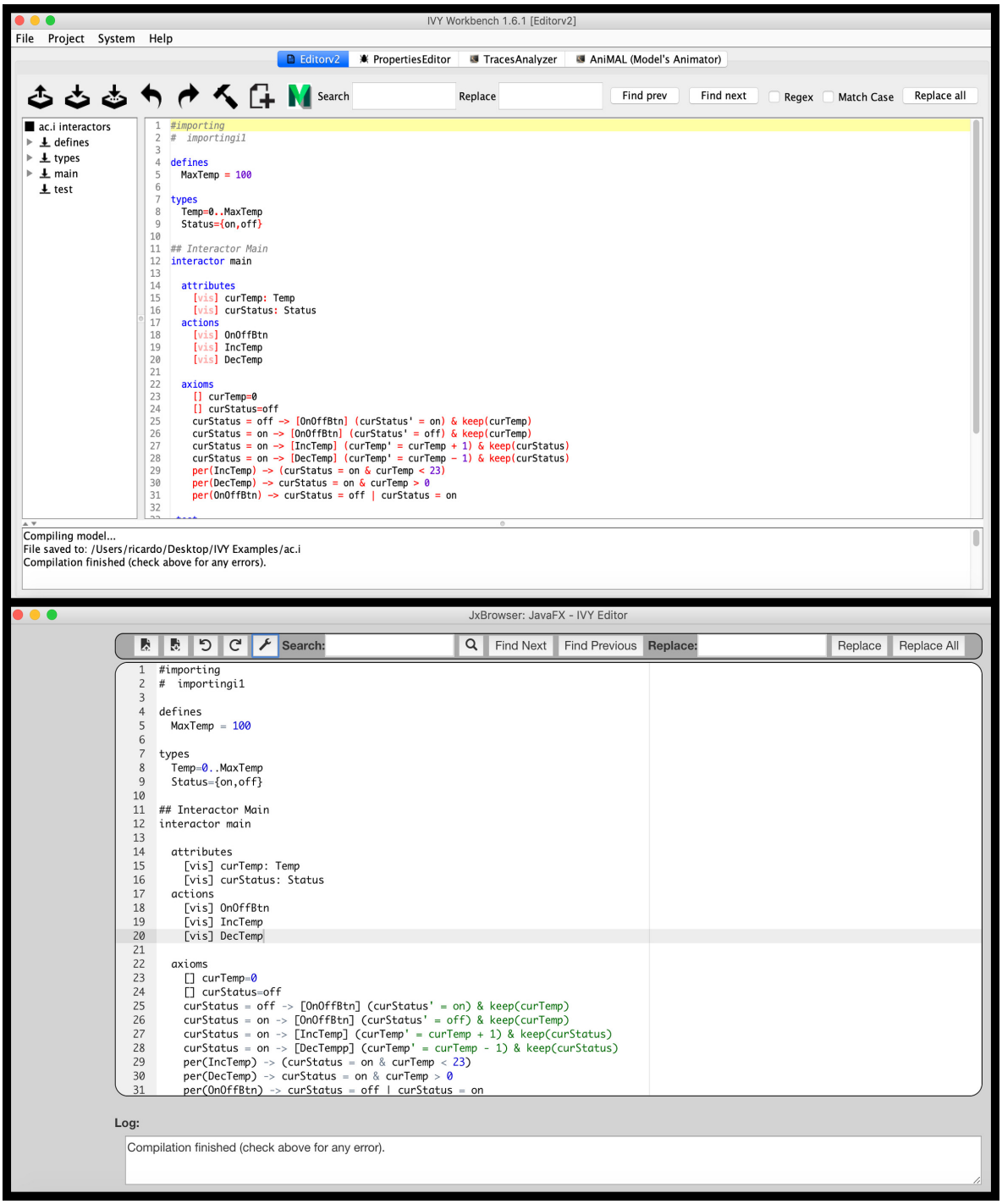


Figure 32.: Sequence of images demonstrating IVY usage

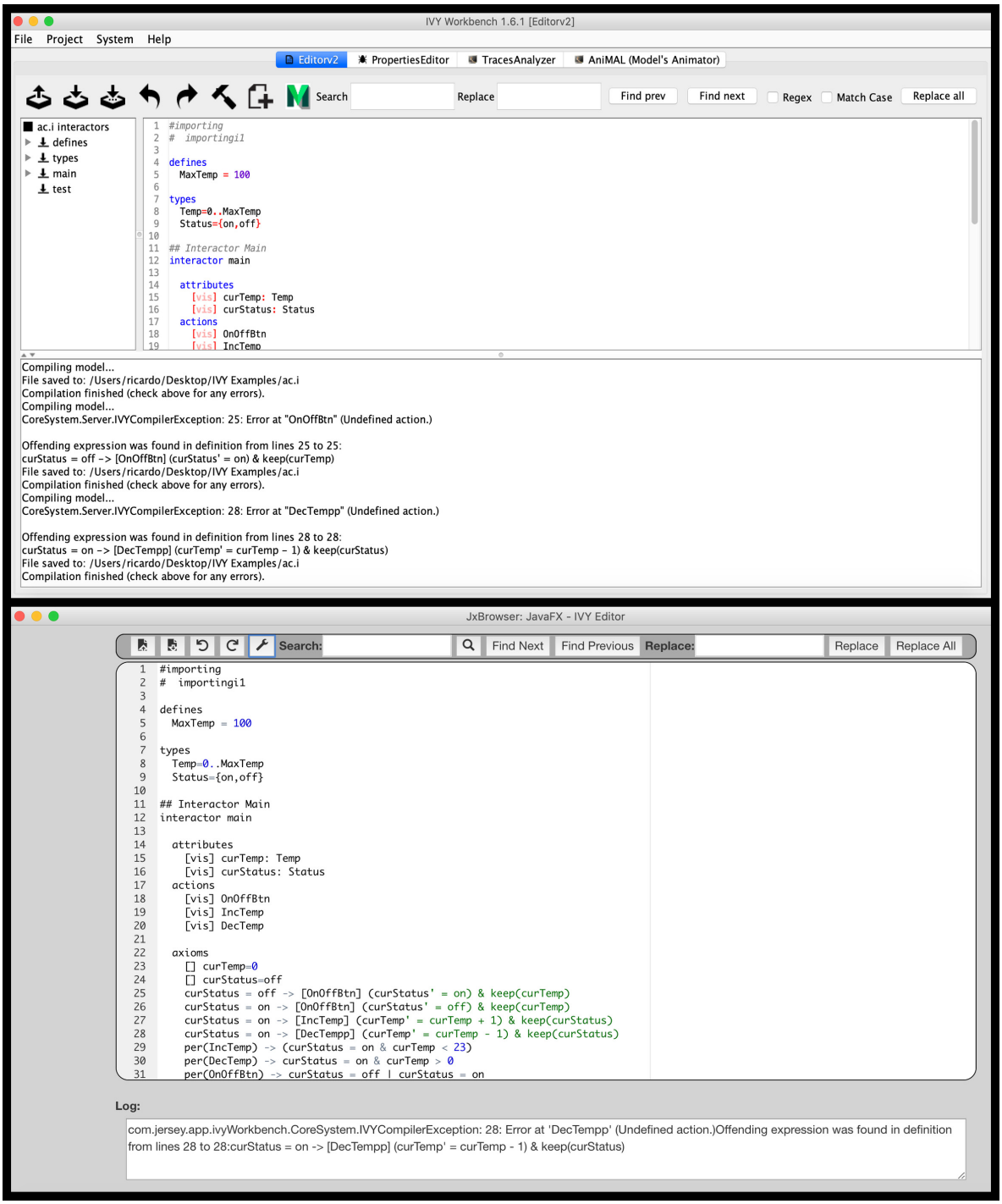


Figure 33.: Sequence of images demonstrating IVY usage

4.4 SUMMARY

These two examples helped us verify the viability of the JxAppDev. The Note Pad application helped as demonstrate that it is possible to create CRUD application using the JxAppDev, and that the derived hybrid application can be accessible in various machines while offering a stable user experience. We were also able to test and refine some of the functionalities implemented in the framework, such as the Web Socket system used for notifications. With the second example, using the IVY Editor, we were able to demonstrate that it is possible to adapt already existing applications (not all types of applications), and provide a user experience that resembles the one obtained with the original version of the application. We also concluded that it is possible to adapt applications if they are based on a multi-tier architecture, but it some cases it is necessary do develop the proper facades for the functionalities we desire. Finally, we were able to estimate the changes required to adapt an existing application as well as determine the extra work required to make the new application work as expected. These two examples serve as a base experiment that can help improve the framework in the future.

5

VALIDATION

While developing our framework several questions arose regarding its usability and the usability of the derived applications created using the framework. A first set of questions regarded the usability of the resulting applications. The questions consisted on whether those applications were practical to use, if there was a big difference between the native application and the new derived ones, but mostly if the costs of developing or upgrading a native application using JxAppDev framework was beneficial enough to justify the extra work. A second set of questions regarded the usability of the framework itself, the required changes on the native applications, but also finding out if there is any interest from developers to use this kind of technologies to help them improve application portability to multiple platform and visibility to a higher number of clients. To address these issues we carried out a three stage validation of the framework and the resulting applications.

5.1 THE STUDY

Before starting the study we defined a set of objectives that should be met at the end of the validation process in order to help us make a proper evaluation. We also properly defined the three validation stages, which consisted of a first stage where the participants interact with all three applications to realize 6 tasks on each one of them. A second stage which consisted in giving a survey, regarding the usability of all applications and their possible differences. A final stage composed of a presentation about the JxAppDev, followed by another survey regarding the interest of the participants in using such development platforms.

STAGE 1 aimed to validate the usability of the different applications.

Objective 1 Provide application usage with minimal effort for the users.

Objective 2 Minimize complexity differences between the different applications.

Objective 3 Reduce the learning curve between the multiple applications.

STAGE 2 aimed to identify possible inconsistency between those applications.

Objective 4 Minimize differences between the applications.

Objective 5 Provide smoothness and good integration of the different framework components.

STAGE 3 evaluated the usability and mostly the interest of using a tool such as JxAppDev.

Objective 7 Target interested public and usages for the framework.

After preparing our list of stages with all the objectives, we proceeded by choosing the subjects for our experiment. The set of test subjects was based on a group of 7 participants all male, with ages between 24 and 27 that performed all three evaluation stages. They were all students from a master's course on Software Engineering at the University of Minho. They had an average of 2 years of industry experience in software development. All test subjects had previously worked on projects consisting in developing native and web applications, and none of them had previous contact with the JxAppDev framework or any similar technologies used for Hybrid application development.

5.1.1 *Stage 1 and 2*

For the first two stages we started by defining a typical use case for a Memo Pad application usage composed of 6 simple tasks that could be performed by the different test subjects. These tasks consisted in the user registering to the system, login in, create one note, create a second note, update one of the notes and delete both notes. Then we created a script for the participants, containing the set of tasks and instruction on how to perform these tasks. We also randomly assigned an order for every version of the application to be tested, native, web and hybrid. A task consists in some goal the user should achieve in the applications (e.g. create a new note). All the applications were tested by completing all the tasks in each one of them.

We also created a questionnaire to evaluate the experience of the participants with the applications and find their interest on using a framework such as this one. The questionnaire was divided in three parts, two distributed to the participants after completing all the

tasks and the final part only distributed after an explanation about how the applications were created, and some information about the JxAppDev framework.

The first part of the questionnaire was composed of three SUS (System Usability Scale) (Brooke, 1996) questionnaires, one for each of the different versions of the application. The second part of the questionnaire contained a set of questions that focused on the differences in usage and complexity of the various applications.

5.1.2 *Stage 3*

After completion of those two parts of the questionnaire, a small presentation of the JxAppDev framework was made followed by some questions and answers to explain its functionalities and about the different applications they had just been testing and how they were developed. Then we gave all the participants the last part of the questionnaire, which was composed of some open questions where they were asked to express their opinions about hybrid application development and the usage of this type of tool to create applications.

The complete Questionnaire can be found in Appendix A.

5.2 SETUP OF THE STUDY

To perform the study, we gathered all the participants in one room. All participants were informed that the maximum duration of time to perform the set of tasks was 1 hour, and that they could abandon the study at any time and even leave any task if they weren't able to finish it. Each participant was asked to reproduce the tasks in their scripts, handed in a printed page. A laptop with all three versions of the application installed was given to each participant. None of the applications contained information on which version it was, since the order was randomly assigned for every participant and written on their scripts. A printed page containing the first two parts of the questionnaire was handed to every participant after they completed the different tasks on all applications. Finally the presentation was made and the last part of the questionnaire was distributed in another printed page. Most of the participants finished the tasks as well as the first two parts of the questionnaire in 30 minutes, plus 5 minutes on average for the last part of the questionnaire after the presentation was terminated. Some questions arouse for the last part of the questionnaire since most of the participants could not elaborate the answers regarding their personal opinion about hybrid applications since they had little experience with the subject.

5.3 Results of the experiment

First we gathered all SUS surveys and calculated the score for each one of them. The results from all test subjects were then added together and the average result for each version of the application was calculated (Figure 34). The obtained results by application helped us conclude that even though the different versions of the application should be identical in usage, the candidates were capable of identifying minor differences between them. The JavaFX version results are slightly higher (83.5) than the results obtained for the other two applications (77.5 for the native version and 78.5 web version) which are practically the same between them (this was expected due to the applications being the same with the simple difference on how they are deployed). According to (Bangor et al., 2008) all three interfaces have good usability.

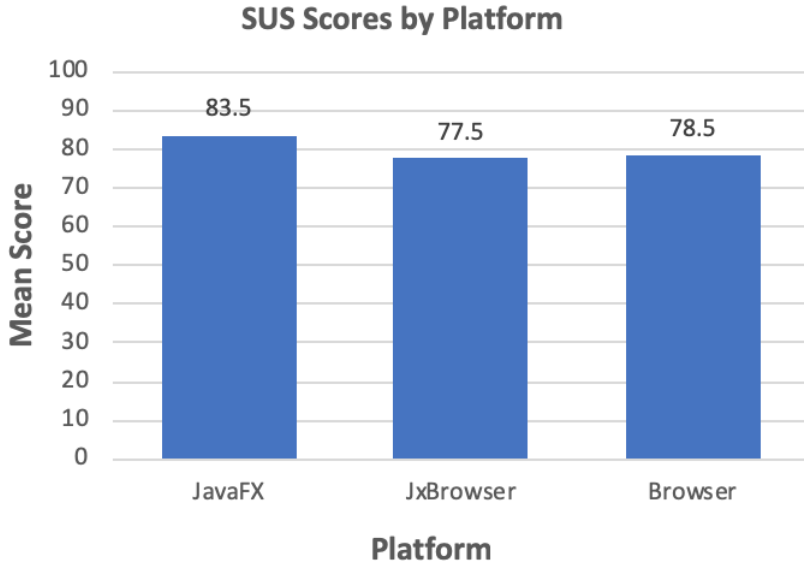


Figure 34.: SUS survey results. Mean score from all participants by platform.

The second part of the questionnaire focused on determining the differences in appearance, usability and functionalities of the different applications. The scale used for these questions was from 0-7 with 0 meaning no difference and 7 meaning that those application were completely different. For every question we calculated the average between the different participants to determine an overall evaluation. The obtained results were between 1.4 and 2.4 see Figure 35, meaning that most of the participant found almost no differences between the applications. The results with higher values were mostly referring the differences between the multiple User Interfaces.

Finally the third component of our questionnaire focused on the general opinion of developers on using such frameworks for development as well as their personal opinion on

	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5	Total Score	Average Score
Question 1	2	2	2	2	2	10	2
Question 2	2	3	2	3	2	12	2,4
Question 3	3	2	1	2	3	11	2,2
Question 4	3	1	2	2	1	9	1,8
Question 5	1	2	1	2	1	7	1,4

Figure 35.: Results of the second part of the questionnaire

the subject of hybrid applications. The presented results, plus the participants' feedback indicate that most of them would try and spend some time learning how to use a framework such as JxAppDev, but only if the learning curve was small. They also implied that they saw benefits on developing hybrid applications due to the diminished costs of development and the more pleasing UI that can be developed using Web technologies. On the other hand they all were sceptic about the performance of the derived applications, and argued about the possibility of a more complicated debugging system since possible bugs and errors would propagate to all the applications instead of being specific to a single platform.

5.4 SUMMARY

After performing this study we concluded that the applications derived from the framework have good performance and they work as expected, allowing to perform the same tasks as the original JavaFX version. The participants were able to finish all the tasks and therefore we conclude that the applications were alike and posed no particular problem to the users. This was also confirmed by the responses that were given by the participants during the questionnaires.

Nevertheless, we should take two things into considerations. First, the test subjects' sample was small and therefore cannot be used to extract conclusive results. Second, we should also notice that this sample was only used to test the derived applications, and therefore we have no data that demonstrates the usability of the framework itself. This is something that can only be achieved with extensive tests to a larger sample of participants and which will require them to test the framework in multiple scenarios.

6

CONCLUSION AND FUTURE WORK

Software development methods are always changing, leading companies to decided between different technologies, and multiple platforms to target. This multitude of choices is in general handled by developing the same software multiple times for multiple platforms using the corresponding programming languages. To avoid these problems some companies are diverging to the development of hybrid applications. The software used to develop this applications focus mainly on mobile devices or creating new application for native platforms. Nevertheless these solutions do no resolve the problem of porting native applications between platforms an well as the web, and neither do they handle already existing application that must be completely re-developed from scratch.

In this paper we present the JxAppDev, a framework that allows the development of cross platform hybrid applications. The framework allows the development and adaptation of native Java applications to Java applications based on a Web user interface, which can be accessed on various native platforms but also through the means of a Web Browser. This framework is composed of an Integration layer for Java applications being this the back-end, and a Java layer where we can integrate a Web user interface. In the front-end we used the JxBrowser framework that supports the integration of Web pages inside a Java native application.

We also created a JavaScript library that facilitates the process of communication between the front-end and the back-end, due to the possible use of a Web Server to provide the user interface. In the back-end we used two connection systems, a REST API that receives the requests coming from the front-end, analyses those requests and transfers them to the Java business logic using Java Reflection which invokes the proper classes and methods. Moreover, we used Web Sockets that oversee accepting registrations from clients to specific notifications and transfers those notifications to the respective registered clients. The back-end is supported by an Apache TomCat Web Container. The framework also contains a Web page navigation system which allows to upload a navigation file that will be analysed and through the help of specific names on HTML buttons, will redirect the user to the proper locations. Regarding the common problems found in hybrid applications, described in Section 2.1, in JxAppDev they were addressed as follows. First, since we offer the possibility

to adapt existing applications, there is no longer a need to rewrite legacy applications. Second, since our back-end is developed in Java, the Web technologies are only required to develop the UI which is not the case with other hybrid solutions where the application is completely developed using Web technologies. This also allows the possibility to have two teams working on separate components of the application, one in Java and the second one developing the UI. Finally, to solve the resource access permissions, we added a token that is used by every user of any application. All requests integrate this token for identification. If the token is not present we can discard the request (due to time constraints, the token is already implemented in the system for the notification system, but not yet used to discard requests).

Based on the tests performed, it is possible to argue that an application developed with this framework can be ported from a native platform to the Web with a minimal set of modifications to the base application. It was also possible to demonstrate that the base functionalities of a Java application are preserved in its hybrid counterpart.

Nevertheless, we recognize the limitations of the realized tests. A study with 7 participants was performed, which helped us evaluate the usability of hybrid applications and find out at which point developers are interested in such technology. This study was an initial validation due to the small number of participants, but we intend to extend the study and collect more information. We still have to test the framework with more complex applications, since it was not possible to properly determine the efficiency of developing and maintaining a single hybrid application, against the development of multiple different applications, despite the positive indications. Although based on the responses we got from the participants of our survey, the results obtained with both use cases that were capable of performing all the tasks we established, and the minimal number of modifications that were required to adapt and develop these two applications, we can conclude that the results were satisfactory and therefore the work on this framework should proceed to obtain better results.

As future work we pretend to extend the number of tests to understand the limits of using external libraries, as well as to properly determine the level of benefits from developing hybrid applications. We want to replace our Web Server by an integrated version, so that there is no need for manual installation and configuration of a new web server for every application. We would like to create a system that automatically converts Java objects into JSON objects to avoid the need to manually create specific classes for this task for every application that we want to adapt using the JxAppDev. We want to create a system that automatically generates the extra files required by the framework, such as the navigation file for the navigation system and the paths file for the Java Reflection calls. We also pretend to extend the functionalities of the JxAppDev in order to be able to develop Android hybrid applications. Finally, we want to add a system that discards all requests coming from

unknown sources, when they don't include the proper token. This will insure that resources are only accessed by users with the proper permissions.

BIBLIOGRAPHY

Apache. Apache license=Website. URL https://www.apache.org/licenses/.

Amit Ashwini. Cross platform human interface guideline problems, December 2017. URL https://medium.com/swlh/what-to-keep-in-mind-while-designing-a-cross-platform-application-fe887dc79869.

Aaron Bangor, Philip T. Kortum, and James T. Miller. An empirical evaluation of the system usability scale. *Int. J. Hum. Comput. Interaction*, 24:574–594, 2008.

Lindsay Bassett. *Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON*. O'Reilly Media, Inc., 2015.

Alessandro Benoit. *NW.js Essentials*. Packt Publishing Ltd, 2015.

Ignacio Bermudez, Stefano Traverso, Marco Mellia, and Maurizio Munafò. Exploring the cloud from passive measurements: the amazon aws case. *IEEE INFOCOM*, pages 230–234, 7 2013.

Gerardus Blokdyk. *Json Web Token*. CreateSpace Independent Publishing Platform, 2018.

John Brooke. *"SUS-A quick and dirty usability scale." Usability evaluation in industry*. CRC Press, June 1996. URL https://www.crcpress.com/product/isbn/9780748404605. ISBN: 9780748404605.

Kristina Chodorow and Michael Dirolf. *MongoDB: The Definitive Guide*. O'Reilly Media, 2010.

Cloud9. Ace - the high performance code editor for the web =Website. URL https://ace.c9.io/.

Kevin Dick. *XML: A Manager's Guide*. Addison-Wesley Professional, 2003.

P.K. Dixit. *Android*. Vikas Publishing House, 2014.

Jon Duckett, Gilles Ruppert, and Jack Moore. *JavaScript and JQuery: Interactive Front-End Web Development Hardcover*. John Wiley & Sons, 2014.

Eclipse. Embedded jetty example=Website. URL https://www.eclipse.org/jetty/documentation/9.4.x/embedded-examples.html. last checked: 01.01.2018.

Electron. Chromium lib=Website. URL https://github.com/electron/libchromiumcontent. last checked: 28.11.2017.

IntelliJ IDEA Essentials. *Jarosław Krochmalski*. Packt Publishing Ltd, 2014.

Ira R. Forman and Nate Forman. *Java Reflection in Action*. Manning, Shelter Island, NY, 2005.

Adam Freeman. *The Definitive Guide to HTML5*. Apress, 2012.

Dr. Michael J. Garbade. Top 3 most popular programming languages in 2018 (survey)=Website. URL https://hackernoon.com/top-3-most-popular-programming-languages-in-2018-and-their-annual-salaries-51b4a7354e last checked: 01.9.2018.

Por Rohit Ghatol and Yogesh Patel. *Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5*. Apress, New York, US, 2012.

Google. Chromium project=Website. URL https://www.chromium.org/.

David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, and Sailu Reddy. *HTTP: The Definitive Guide*. O'Reilly Media, Inc., 2002.

Steve Graham, Glen Daniels, Doug Davis, Yuichi Nakamura, Simeon Simeonov, Peter Brittenham, Paul Fremantle, Dieter Koenig, and Claudia Zentner. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Sams Publishing, 2004.

Chris Griffith and Leif Wells. *Electron: From Beginner to Pro*. Apress, New York, US, 2017.

Sergey Grinev. *Mastering JavaFX 10*. Packt Publishing Ltd, Birmingham, United Kingdom, 2018.

Arun Gupta. Rest vs web socket comparison and benchmark=Website. URL http://blog.arungupta.me/rest-vs-websocket-comparison-benchmarks/. last checked: 02.01.2018.

Robert Harris and Rob Warner. *The Definitive Guide to SWT and JFace*. Apress, 2004.

HASLab. Ivy workbench website=Website. URL http://ivy.di.uminho.pt/. last checked: 20.6.2018.

Dan Hermes. *Xamarin Mobile Application Development: Cross-Platform C and Xamarin.Forms Fundamentals*. Apress, 2015.

Heroku. Embedded tomcat example=Website. URL https://devcenter.heroku.com/articles/create-a-java-web-application-using-embedded-tomcat. last checked: 01.01.2018.

Steve Holzner. *Eclipse: Programming Java Applications*. O'Reilly Media, Inc., 2004. ISBN 0596552629, 9780596552626.

Meteor Development Group Inc. Meteor hosting service=Website. URL https://www.meteor.com/hosting. last checked: 28.11.2017.

Andy Jackson. Swing webview integration example=Website. URL https://gist.github.com/anjackson/1640654. last checked: 01.01.2018.

JetBrains. Jetbrains=Website. URL https://www.jetbrains.com/.

Aaron Hillegass Joe Conway. *iOS Programming: The Big Nerd Ranch Guide*. Addison-Wesley Professional, 3rd edition, 2012.

Carole S. Jordan. *Guide to Understanding Discretionary Access Control in Trusted Systems*. DIANE Publishing, 1987.

Vladimir Kharlampidi. Hybrid mobile framework - framework7 =Website. URL https://framework7.io/.

Xuekun Kou. *GlassFish Administration*. Packt Publishing Ltd, 2009.

Marc Loy, Brian Cole Robert Ecksteinand Dave Wood, and James Elliott. *Java Swing*. O'Reilly Media, Inc., California, US, 2002.

Adam Lynch. Electron application architecture=Website, a. URL https://www.smashingmagazine.com/2017/03/beyond-browser-web-desktop-apps/. last checked: 29.12.2017.

Max Lynch. Ionic dev survey says... there's no better time to be a web developer=Website, b. URL https://blog.ionicframework.com/ionic-dev-survey-says-theres-no-better-time-to-be-a-web-developer/. last checked: 20.6.2018.

Vipul A M and Prathamesh Sonpatki. *ReactJS by Example - Building Modern Web Applications with React*. Packt Publishing Ltd, 2016.

Tom Marrs and Scott Davis. *JBoss at Work: A Practical Guide*. O'Reilly Media, Inc., 2005.

G. Martin, J. Suman, and S. Vitaly. Breaking and fixing origin-based access control in hybrid web/mobile application frameworks. *NDDS Symp.*, pages 1–42, 2 2014.

Mark Masse. *REST API Design Rulebook*. O'Reilly Media, Inc., 2011.

mcasimir. Hybrid mobile framework - mobile angular ui=Website. URL http://mobileangularui.com/.

David Sawyer McFarland. *CSS3: The Missing Manual*. O'Reilly Media, Inc., 2012.

Nathan Murray, Felipe Coury, Ari Lerner, and Carlos Taborda. *Ng-Book: The Complete Guide to Angular*. CreateSpace Independent Publishing Platform, 2018.

Nanda Nachimuthu. *Mastering Apache Cordova*. Packt Publishing, Limited, 2017.

Oracle. Javafx update list=Website, a. URL https://www.java.com/en/download/faq/release_changes.xml.

Oracle. Javafx qa=Website, b. URL http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html. Question Number: 6.

Oracle. Javafx recent releases list=Website, c. URL https://www.java.com/en/download/faq/release_dates.xml.

Oracle. Oracle's web site=Website, d. URL https://www.oracle.com/index.html. last checked: 02.01.2018.

Oracle. Swing webview example=Website, e. URL https://docs.oracle.com/javafx/2/swing/SimpleSwingBrowser.java.htm. last checked: 01.01.2018.

Osgi. Osgi=Website. URL https://www.osgi.org/.

Hoc Phan. *Ionic Cookbook*. Packt Publishing, 2015.

Jon Raasch. *Smashing WebKit*. John Wiley & Sons, 2011. ISBN 1119999138, 9781119999133.

Hugo Rodrigues. Hybrid html5 apps=Presentation, April 2015. URL https://www.slideshare.net/hugorodriguespt/hybrid-html5-apps. last checked: 28.11.2017.

Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *Computers*, pages 38–47, 2 1996.

James Snell, Doug Tidwell, and Pavel Kulchenko. *Programming Web Services with SOAP: Building Distributed Applications*. O'Reilly Media, Inc., 2001.

Basarat Syed. *Beginning Node.js*. Apress, New York, US, 2014.

TeamDev. Jxbrowser - intellij idea integration=Website. URL https://jxbrowser.support.teamdev.com/support/solutions/articles/9000012869-jxbrowser-in-intellij-idea.

TeamDev. Jxbrower=Website, 2015. URL https://www.teamdev.com/jxbrowser. last checked: 28.11.2017.

TIOBE. Programming language comparison=Website. URL https://www.tiobe.com/tiobe-index/. last checked: 29.12.2017.

Fabian Vogelsteller, Isaac Strack, and Marcelo Reyna. *Meteor: Full-Stack Web Application Development*. Packt Publishing Ltd, Birmingham, England, 2016.

Aleksa Vukotic and James Goodwill. *Apache Tomcat 7*. Apress, New York, US, 2011.

A

QUESTIONNAIRE

In the following pages we can fin the questionnaire used during the Validation phase of this project.

System Usability Scale

Instructions: For each of the following statements, mark <u>one</u> box that best describes your reactions to the application you tested first.

	Strongly Disagree				Strongly Agree
1. I think that I would like to use this application frequently.	☐	☐	☐	☐	☐
2. I found this application unnecessarily complex.	☐	☐	☐	☐	☐
3. I thought this application was easy to use.	☐	☐	☐	☐	☐
4. I think that I would need assistance to be able to use this application.	☐	☐	☐	☐	☐
5. I found the various functions in this application were well integrated.	☐	☐	☐	☐	☐
6. I thought there was too much inconsistency in this application.	☐	☐	☐	☐	☐
7. I would imagine that most people would learn to use this application very quickly.	☐	☐	☐	☐	☐
8. I found this application very cumbersome/awkward to use.	☐	☐	☐	☐	☐
9. I felt very confident using this application.	☐	☐	☐	☐	☐
10. I needed to learn a lot of things before I could get going with this application.	☐	☐	☐	☐	☐

Overall Evaluation

How different were the multiple tested applications?

Equal ☐ ☐ ☐ ☐ ☐ ☐ ☐ Different

In case you would have to recommend these applications, were there some which you would recommend less than others?

Same Recommendation ☐ ☐ ☐ ☐ ☐ ☐ ☐ Different Recommendation

Was there some noticeable difficulty in using one application compared to the others?

Minor ☐ ☐ ☐ ☐ ☐ ☐ ☐ Great

Did you see any smoothness and navigation problems in any of the applications that were tested?

No Problem ☐ ☐ ☐ ☐ ☐ ☐ ☐ Multiple problems

Did you encounter crashed or bugs during any of the tests?

None ☐ ☐ ☐ ☐ ☐ ☐ ☐ A lot

General questions and opinions

Do you think cross platform application can offer some benefits in application development?

Would you learn to use a different development platform if it offered the possibility to turn simple native application into cross platform web and native applications.

If offered the possibility to use a Framework which allowed you to upgrade any older local application into the Web, with the cost of developing a new UI and some minor changed to the application, would you consider using it?

Can you list 3 positive and 3 negative aspects of creating cross platform applications?

-
-
-
-
-
-

Can you list 3 positive and 3 negative aspects of the cross-platform applications you tested compared to the native application.

-
-
-
-
-
-

Could you list some benefits or disadvantages of using Web Technologies to develop UI for native applications?

-
-
-
-
-
-
-
-
-
-
-