Advanced Implementation of Mobile Applications

Gabriel Batista de Almeida

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Fernando Henrique Côrte-Real Mira da Silva

Examination Committee

September 2022

Acknowledgments

I would like to thank my thesis supervisor Prof. Fernando Mira da Silva for his guidance, support, and advice that helped me throughout the whole thesis.

I would also like to express my gratitude towards the DSI team that offered me the best support one could have asked for in the development of the proposed solution for this thesis. I would like to thank mainly Marco Cerdeira for always being available to assist me with anything, especially concerning the Fenix backend, Catarina Cepeda for helping me with the app development, decision making, and any other frontend related matter, and finally, Rita Severo for providing most of the design, allowing me to focus more on this thesis and the solution development.

I would like to extend my gratefulness to my parents as well, for always trying to provide the best education possible, for all the advice and help throughout the years, and for being there for me in the good and bad times in my life.

Lastly but certainly not least, to my close friends for providing feedback on this work, for their constant support, for giving me happy memories, and for always being there for me.

To each and every one of you – Thank you.

Abstract

Mobile applications are becoming more and more popular, to the point that most people prefer using mobile applications instead of their desktop counterparts. The development of native mobile applications can be very challenging due to the costs and expertise associated with them since each platform has its separate codebase built using different technologies. With this in mind, a cross-platform development framework is a possible solution to this problem, since this kind of approach allows sharing a majority of the codebase between all supported platforms, eliminating the need to develop and maintain two or more separate codebases. Such approaches are described and compared in this thesis, particularly React Native, Flutter, and Progressive Web Apps, due to their recent success and their vast number of features. After comparing the aforementioned frameworks and using Instituto Superior Técnico as our case study, we considered the best choice currently to be React Native due to its performance, native UI looks, big and active community, and largely thanks to the use of a programming language known by many developers, JavaScript. The whole development process was registered in detail, namely the requirements gathering and discussion, taking into consideration the feedback and suggestions gathered from a survey, the design, implementation and evaluation. Final tests and evaluations showed high responsiveness and an excellent user experience, at a moderate cost of higher resource usage relative to its native counterparts.

Keywords

Cross-Platform Development; React Native; Flutter; Progressive Web Apps; Mobile App Development; Academic Applications.

Resumo

As aplicações móveis têm vindo a tornar-se cada vez mais populares, ao ponto de a maioria das pessoas preferir o uso de aplicações móveis aos seus websites equivalentes. O desenvolvimento de aplicações móveis nativas pode ser muito desafiante devido aos custos e competências associados a elas, visto que cada plataforma tem uma base de código independente, desenvolvida usando diferentes tecnologias. Com isto em mente, uma abordagem de desenvolvimento cross-platform é uma solução possível para este problema, visto que esta abordagem permite partilhar a maioria da base de código com todas as plataformas suportadas, eliminando a necessidade de ter que desenvolver e manter mais que uma base de código. Tais abordagens são descritas e comparadas nesta tese, nomeadamente o React Native, o Flutter, e as Progressive Web Apps, devido ao seus recentes sucessos e grande número de funcionalidades. Depois de comparar as alternativas anteriores e usando o Instituto Superior Técnico como um caso de estudo, consideramos o React Native como a melhor escolha atualmente, devido ao seu desempenho, interface nativa, grande comunidade, e especialmente devido ao uso de JavaScript. Todo o processo de desenvolvimento foi registado em detalhe, nomeadamente a definição e discussão dos requisitos, tendo em conta o feedback e as sugestões obtidas de um inquérito realizado, o design, a implementação e a avaliação. Testes e avaliações finais mostraram uma alto nível de desempenho e uma excelente experiência de utilizador, a um custo moderado de um maior uso de recursos relativamente às respetivas aplicações nativas.

Palavras Chave

Desenvolvimento Cross-Platform; React Native; Flutter; Progressive Web Apps; Desenvolvimento de Aplicações Móveis; Aplicações Académicas.

Contents

List of Figures

List of Tables

Listings

Acronyms

ADB	Android Debug Bridge
AP	Access Point
API	Application Programming Interface
ARM	Advanced RISC Machines
CI/CD	Continuous Integration and Continuous Delivery
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DOM	Document Object Model
DSI	Direção de Serviços de Informática
FPS	Frames per Second
GPU	Graphics Processing Unit
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IST	Instituto Superior Técnico
JSI	JavaScript Interface
JSX	JavaScript Syntax Extension
NDA	Non-Disclosure Agreement
OAuth	Open Authorization
OS	Operating System
PKCE	Proof Key for Code Exchange

PWA	Progressive Web App
RAM	Random Access Memory
SDK	Software Development Kit
SSL	Secure Sockets Layer
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
VS Code	Visual Studio Code

1 Introduction

Contents

Since the unveiling of the first iPhone in 2007, smartphones are becoming an increasingly crucial part of our daily lives, and with that, the demand for mobile apps is increasing. According to recent statistics, the current number of smartphone users is 6.3 billion and is estimated to be 7.5 billion in 2026 [4], an annual increase of approximately 3.5%. Furthermore, according to recent studies [5], more than 54% of web traffic originated from mobile devices, and the number of mobile applications downloaded in 2020 alone was more than 200 billion [6], stating that the use of mobile apps is more popular than their equivalent desktop websites.

This rise in popularity, however, caused a problem for companies and software developers that want to create an app for mobile devices. For an application to reach as many users as possible, it is important to decide for which platforms is it going to be distributed, and as of today, Android and iOS dominate the smartphone industry with 72.44% and 26.75% of market share respectively [7]. If both platforms have to be supported, the use of separate codebases and different technologies for each platform increases the development and maintenance costs.

Therefore, a unified solution that reaches both platforms is needed to reduce costs and increase time-to-market. A cross-platform framework that reuses most of its codebase for all of its target platforms, simplifying the development process, is an obvious solution for this goal. However, it also implies several technical challenges and may raise performance issues.

1.1 Motivation

In 2015, Facebook, launched React Naive, a cross-platform framework that fixed most of the disadvantages of previous cross-platform approaches, allowing the development of an application that looks and feels exactly like a native app, while sharing most of its codebase with their target platforms. Since then, the popularity and use of this kind of approach have been increasing, giving birth to new cross-platform frameworks, like Flutter and Progressive Web Apps Progressive Web Apps (PWAs), both created by Google.

With several frameworks available, it is hard to decide the best approach to build a mobile application that both reaches the maximum number of users possible and cuts costs at the same time, both in time and money. Furthermore, the type of application that is going to be developed also plays a relevant role in the framework selection, so it is essential to be aware of the advantages and disadvantages of each framework to perform the best choice possible.

Deciding on the framework, however, is not everything, and there are a lot of other aspects that need to be taken into account, like the steps and phases needed to create an optimised mobile application and the challenges and complexities of doing so.

In order to investigate these points in a more practical case, the Instituto Superior Técnico (IST) mobile application was used. It was chosen since it was developed using native code, so a new application that is easier to maintain and develop is needed to replace it.

1.2 Objectives

The objective of this work is to find the best current alternatives to native development, their advantages, disadvantages, and depending on the requirements and the type of application, which is currently the best overall approach to develop an application from scratch. Moreover, the best practices to develop a mobile application and the challenges and complexities of developing them are going to be enumerated and analysed.

At the same time, it was set as a goal to validate this work with the development of a new application for IST, containing features to help the daily lives of students, teachers, and staff, which corrected the flaws of the existing one and replace it in the future. This case study and the development context were also taken into account in the framework selection process.

1.3 Document Structure

The remaining of this thesis is structured as follows: Chapter 2 describes the current state of the art and similar work to the case study that is going to be used. Chapter 3 defines the requirements for the application that is going to be developed, and based on those requirements, the solution found that best suits the case study. Chapter 4 details the development experience and the implemented requirements. Chapter 5 describes and analyses the evaluation methods and their results. Finally, Chapter 6, discusses the conclusions of this work and proposes some future work.

2 State of the art

Contents

To find the best approach to develop a mobile app, the different types of application methodologies need to be understood. Based on the upsides and downsides of each approach, and the purpose and requirements of the target app, it is possible to discuss the best approach.

2.1 Current App Taxonomy

The most common taxonomy for mobile applications, is the one described by Xanthopoulos and Xinogalos [8], which divides applications into *Native Apps*, *Web Apps*, *Hybrid Apps*, *Interpreted Apps*, and *Generated Apps*. Except for native apps, all of the other types of applications are cross-platform approaches. Cross-platform approaches have been widely used, particularly since the unveil of React Native [9], created by Facebook. Later on, Flutter [10], developed by Google, also contributed to increase the popularity of cross-platform frameworks.

2.1.1 Native Apps

Native apps are developed using the official Software Development Kit (SDK), for the target platform, following the operating system design language and guidelines. The Integrated Development Environment (IDE) for iOS is Xcode [11] and for Android is Android Studio [**?**]. Apps are developed using the recommended native programming language for the platform, namely, Objective-C or Swift for iOS [12] and Java or Kotlin for Android [13]. Although multiple programming languages are supported, Apple and Google recommend using Swift and Kotlin, respectively. These IDEs provide features to help develop, test, debug, and emulate the application.

The advantages of native apps are that they support the latest software and hardware features, and they provide the intended native User Experience (UX) that makes them look like they belong to the native ecosystem. Moreover, they have near-optimal performance, since they are compiled directly to binary code, not needing any further translation.

The greatest downside that makes most companies look for a cross-platform approach is the fact that if the application is available on multiple platforms, more than one codebase needs to be maintained, increasing the cost of development and maintenance. Other factors are the expertise needed to develop a native app, as well as the need to learn a native language and IDE.

2.1.2 Web Apps

Web apps are applications that run in an internet browser. In other words, it is a website that is especially made for mobile devices. They are developed using common web technologies like HyperText Markup Language (HTML) and JavaScript, and sometimes they are used alongside frameworks like VueJS, Angular, Ruby on Rails and so on [14]. Since they act as websites, they are built using a three-tiered architecture [15] composed of the following layers:

- The **presentation tier**, that communicates with the other tiers through Application Programming Interface (API) calls, displays information, and receives user input;
- The **application tier**, which contains the logic of the application and is usually stored on a server;
- The **data tier** contains the database for the application.

Despite having the advantage of being a cross-platform approach and having the ability to run without being installed on the mobile device, web apps do not have access to many hardware features, they are slower, since they are rendered in a browser, and the main logic is not present on the device itself, so each web page needs to be downloaded. Moreover, they do not work offline, which can be a disadvantage to some applications that do not require constant internet access to function.

Web apps are quite simple to develop, which is a major positive factor. However, given their limitations, they are only suited to simple apps that do not have strict performance requirements and only need to access basic hardware features.

2.1.3 Hybrid Apps

Hybrid apps are a mix of native apps and web apps, where a web page, like a web app, is built using HTML and JavaScript and runs on a native container through a web view component. This component renders the page like a browser, and the native container provides more access to the device's hardware than traditional web apps. This access is done through API calls that are then translated to the appropriate one for the platform being used [14]. Since it runs on a native container, hybrid apps can be installed on the device and they do not need to obtain the source code for the app on a remote server. Apache Cordova, Ionic, and Adobe PhoneGap are popular hybrid app frameworks. PhoneGap was the first cross-platform approach, being released on 2009 and was later bought by Adobe in 2011, that donated the source code to Apache, creating Cordova. However, Adobe PhoneGap was discontinued in 2020.

Even though hybrid apps have the advantages of web apps, like being cross-platform, and resolving some of the issues present in them, they still have some disadvantages when compared to native apps. For instance, the number of device-specific features that they can access is limited, and the graphics quality is inferior to native apps [16], performance is not as good, since it basically runs on a browser, and their hardware resource efficiency is far worse than native apps. In a study comparing the same application developed using a native and hybrid approach [17], it was found that hybrid apps use on average twice the Central Processing Unit (CPU) load of native apps, which can be a problem in low-end devices. This high CPU consumption can also lead to higher energy consumption, which is always a critical design factor in mobile apps.

2.1.4 Interpreted Apps

Interpreted apps have a layer between the application itself and the platform, called *the bridge*, that enables the use of native user interface components and hardware features [18]. While keeping an abstract model of the platform, interpreted apps can be programmed using a variety of programming languages, like JavaScript, Java, and others. Currently, React Native is the most used interpreted approach.

Their main advantages are to use native user interface components, to adopt well-known programming languages, which reduces cost and increases efficiency, to have performance levels close to native apps, and to use almost all of the hardware features available. Hansson and Vidhall [19] compared a React Native application with a similar native app in both Android and iOS and showed that both applications have equivalent user experiences.

A downside of this design is that apps occupy more storage and, despite having similar performance, they can use more resources when compared to their native counterparts. This can be an issue, especially on low-end devices, since the main goal of cross-compiled approaches is to reach the widest range of different platforms and devices.

2.1.5 Generated Apps

Generated apps use the same code base to compile and generate machine code for each platform. The natural consequence is the higher performance of these apps, which can be on par with native apps, since no further translation is needed. One example of this approach is Flutter, which uses Dart as its main language and generates native Advanced RISC Machines (ARM) machine code.

Another approach for building generated apps is using a Model-Driven approach. In this kind of approach, a model of the application is constructed with its specifications and features. Then, from this model, native code is generated for each target platform [8].

2.2 An Alternative Taxonomy

Recently, a new alternative taxonomy for mobile applications was proposed by Nunkesser [20], which attempts to overcome some issues found in the conventional taxonomy [8] adopted in this document. Nunkesser states that ”the current state of cross-platform development does not really suit the taxonomy", giving the example of Xamarin, which is developed using C# and compiles applications into binary code, that does not fit into any category. The taxonomy proposed by Nunkesser divides the mobile app categories more clearly in the following manner [20]:

- **Endemic Apps** are applications that use the operating system's SDK, for instance, Swift in Xcode for iOS;
- **Pandemic Apps** use programming languages that are supported by most Operating Systems (OSs), like HTML, Cascading Style Sheets (CSS), JavaScript, C and C++. This category contains 4 subcategories:
  - **Web Apps** correspond to the web apps described in Section 2.1.2;
  - **Hybrid Web Apps** are web apps encapsulated in an endemic shell, corresponding to the hybrid apps in Section 2.1.3;
  - **Hybrid Bridge Apps** use endemic JavaScript engines and can use endemic User Interface components as React Native does;
  - **System Language Apps** are apps that are built using C or C++.
- **Ecdemic Apps** are applications that are developed using a programming language that is not understood by the operating system.

Despite fixing some issues and introducing an improved classification, this taxonomy still has some issues. For example, it assigns the *Hybrid Bridge Apps* category to Flutter, which is debatable, since it does not use any JavaScript bridge or engine. In our view, it fits better into the *Ecdemic Apps* category, since its programming language, Dart, is not understood by most OSs. Also, the use of terminologies like *Endemic, Ecdemic and Pandemic* are not easily understandable and are not widely used. For these reasons, the taxonomy of Xanthopoulos and Xinogalos [8] was followed in this document.

2.3 Main Cross-Platform Frameworks

The most used cross-platform approaches today are React Native and Flutter, which in 2020 were used by 42% and 39%, of the developers using a cross-platform framework [21]. Recently, Progressive Web Apps (PWAs) have also become a viable alternative, due to their simplicity, In the following section, these frameworks are described and discussed in more detail.

2.3.1 React Native

React Native was launched by Facebook in March of 2015 as an open-source framework, and it started out as an internal hackathon back in 2013 [22]. Being open-source meant that anyone could contribute to the project. Shortly after its launch, Microsoft and Samsung decided to help develop React Native and use it in their operating systems [23]. The fact that React Native was also based on Facebook's ReactJS made most developers feel right at home when using it, decreasing its learning curve. Another feature to ease the development is the support for hot reloading, which allows seeing the changes performed in the application files being applied live in the emulator, or in the device itself, without losing the state of the app.

All of this made React Native incredibly popular, to the point that its own GitHub repository is one of the most starred projects of all time. Another consequence is that there are numerous custom-made components and libraries made by the enormous community. If a developer needs something not available in React Native chances are that there is already a community-made component or library for it.

Currently, many popular applications are using React Native, namely: Facebook, Instagram, Discord, Skype, Bloomberg, and Tesla, among others [24].

2.3.1.A Rendering

React Native uses the Virtual Document Object Model (DOM) to render the components, which is similar to a browser's DOM. The Virtual DOM is a node tree that keeps all view elements and their properties, acting like a snapshot of the state of the app. When a change of state happens in the application, the Virtual DOM plus a diffing algorithm is used to only re-render the necessary parts and thus improving the performance. To perform this update, the following steps are executed [18]:

- Update the state or the properties of the affected components,
- Create a new Virtual DOM tree with the updated components,
- Find the difference between the new tree and the old one, using a diffing algorithm,
- Re-render only the difference between the two trees with the minimum amount of changes.

This process of diffing the trees and selecting which nodes in the Virtual DOM need to be updated or replaced is called *reconciliation* [25].

2.3.1.B The Bridge

Since React Native is an interpreted app, it uses a JavaScript bridge to access the native APIs. The access to these APIs gives React Native the ability to access all the platform's features, and if a feature is not available, the developer can modify the bridge by creating native modules[1] to add the missing

[1] Native Module – https://reactnative.dev/docs/native-modules-android

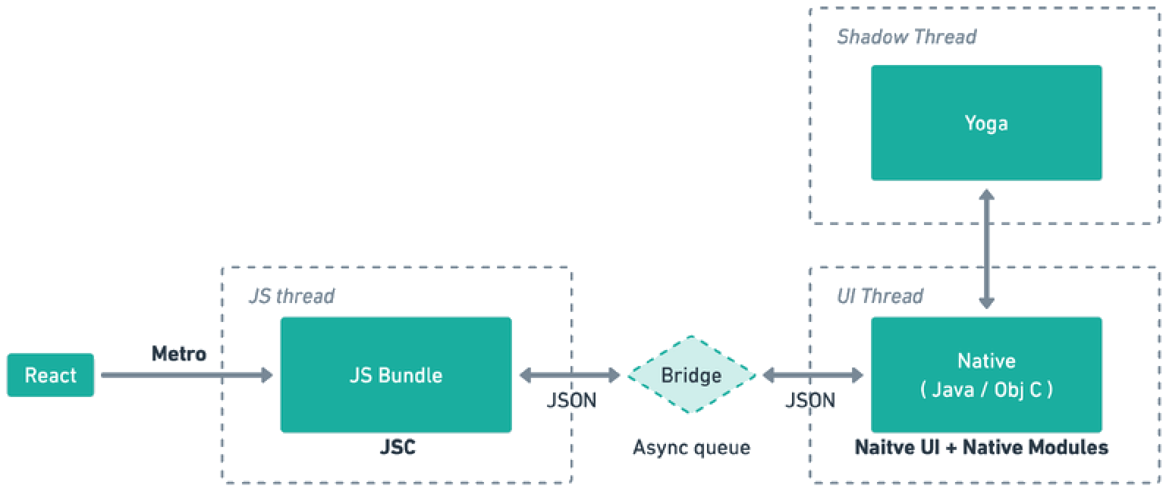


Figure 2.1: React Native architecture. Adapted from [1]

feature. Native modules allow the JavaScript side of React Native to access a module created using native code. The bridge not only provides native features, but it allows access to native rendering methods and elements, giving it an User Interface (UI) indistinguishable from a native application [18]. To further increase the performance, this bridge runs on a different thread from the main UI thread, so the UI always feels responsive [19]. This bridge can communicate asynchronously both ways between the main UI thread and the JavaScript thread (Figure 2.1), which places the messages in a queue that is also running on a separate thread [26].

Over the years, however, some issues were found with this asynchronous design. Being asynchronous makes the implementation of new features harder to implement, it is not possible to cancel events and makes it difficult to introduce JavaScript code into native modules while expecting a synchronous response, among other problems [27].

A simple example of this problematic behaviour is the *onScroll* event on any item list. In the edge case where a list has many items, while a scrolling action is being performed, a message is sent from the main thread to the JavaScript thread to get the items that need to be rendered. If the scrolling speed is high and the list is large, it is possible to scroll faster than the JavaScript thread can respond to the main thread. When this happens, React Native renders empty space instead of the list items, creating a visual gap between the interface and the intended action.

2.3.1.C The New React Native Architecture

In order to overcome the limitations of the asynchronous design, the React team is working on a new architecture that is currently deployed on the Facebook app and is starting to be implemented on existing libraries [28]. One of the issues of the current architecture is the need for the bridge to serialize

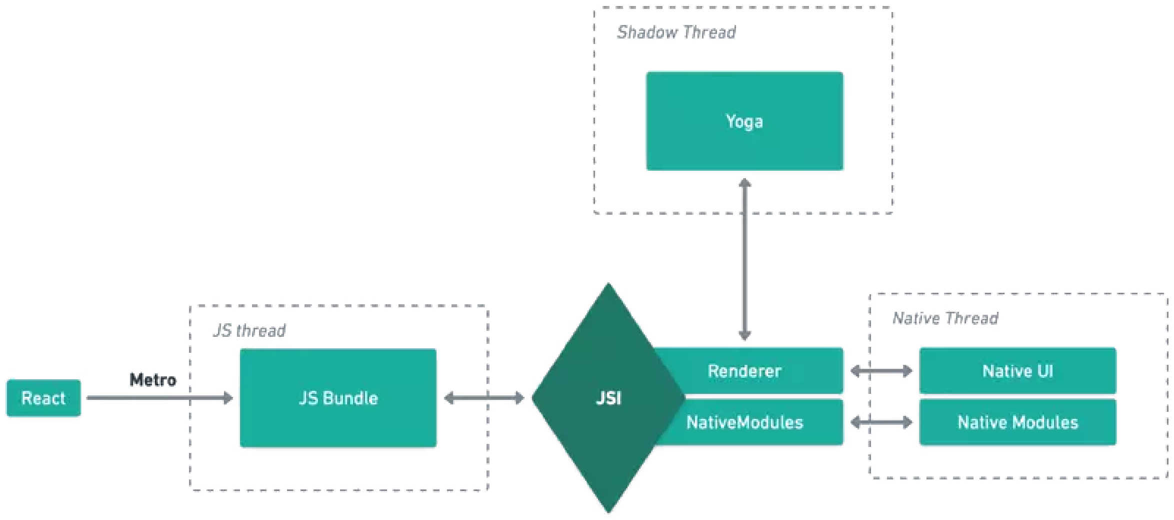


Figure 2.2: React Native new architecture. Adapted from [1]

everything to JSON in order to communicate between the JavaScript realm and the native one. This can easily become a bottleneck and also introduces unnecessary operations, which take more time. To remove the bridge entirely, the new architecture has a similar approach to how ReactJS works in a browser, containing two important components to overcome the aforementioned issues:

- The **JavaScript Interface (JSI)**, which now integrates the bridge functions;
- The new UI manager, **Fabric**.

When a node is created in a browser, the browser returns an object that contains a reference to the corresponding native element. JSI works similarly to this, returning an object containing a reference to the appropriate host object, written in C++, allowing the use of native methods directly using this reference. With this, instead of using a bridge, JSI would be used to communicate between the JavaScript and native threads (Figure 2.2). This change also made the use of other better-performing JavaScript engines possible, since React Native is no longer dependent on its engine, the JavaScript Core engine.

The new UI manager, *Fabric*, allows the use of both synchronous and asynchronous UI operations, resolving the other issues presented above, like the *onScroll* event. The use of synchronous operations is possible due to the fact that *Fabric* is able to block the UI thread and run operations on it, resuming the asynchronous operations afterwards.

2.3.1.D Components

Everything in React Native is built using components. These components are programmed using JavaScript Syntax Extension (JSX). JSX is a syntactic extension of JavaScript [29] that instead of separating technologies, enables to integrate the logic, the UI, and styling for a component on a single file. Separation of concerns is used instead of a separation of technologies [22]. JSX allows components to be developed as classes, making code re-utilisation possible with class hierarchies (Listing 2.1). Instead of classes, they can be developed as functional components (Listing 2.2).

Listing 2.1: Class component in React Native

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = {aSimpleState: true};
    }

    render() {
        return {
            <View>
                <Text>Hello World!</Text>
            </View>
        }
    }
}
```

Listing 2.2: Functional component in React Native

```
export default function MyComponent(props) {
    const [aSimpleState, setASimpleState] = React.useState(true);

    return (
        <View>
            <Text>Hello World!</Text>
        </View>
    );
}
```

Components can receive external properties, called *props*, have internal state, and a render method. Built-in components can either be cross-platform or platform specific. For instance, the *View* component, which is the most basic component and equivalent to the *div* in ReactJS, that in turn is HTML, is rendered as a *View* component in Android and *UIView* in iOS [18]. An example of a platform specific component is the date picker, that in Android is *DatePickerAndroid* and in iOS *DatePickerIOS* [22]. It is also possible to develop platform specific components. The files need only to have a different extension, for example, for Android *myComponent.android.js* and for iOS *myComponent.ios.js* [22].

2.3.1.E Props and States

As mentioned before, props are properties of a component given by its parent, and they are immutable. In other words, the child cannot change the values of those properties, only its parent (Listing 2.3) [25].

Listing 2.3: Setting a value for a prop in a React Native component

```
<MyComponent aSimpleProp={value}/>
```

The state is initialised whenever a component is mounted, and the syntax used to create it is different depending on the type of the component (Listing 2.1 and Listing 2.2). They can hold any time of information deemed to be useful.

Both of them are useful for creating dynamic components since changing the values of props and states causes a re-render of the component on its affected parts.

2.3.1.F Styling

Although styling in React Native is not done through CSS, its syntax is quite similar and contains many of the CSS features used frequently. Styling can be done in three ways [30]:

- Passed as a style prop in the component;
- Used as a JavaScript Object;
- Created using the *Stylesheet.create* method, which is the recommended method [22].

2.3.1.G Performance

A study made by Danielsson [26] compared the same application developed for both native Android and React Native, testing CPU, Graphics Processing Unit (GPU) and Random Access Memory (RAM) utilisation and power consumption. In their first test, they tested both applications at idle (opened and then kept in the background). In their results, the CPU, GPU, and power consumption were similar. However, React Native clearly had higher RAM requirements.

The difference between both was more apparent in the other tests, where some user interaction was involved. In those tests, the GPU usage was the only one that remained similar between them. The average CPU utilisation for native Android was around 36.1% and for React Native 41.6%, representing an increase of approximately 15%, which can be noticeable on some devices. Lastly, both the power consumption and memory were very similar, but slightly higher in React Native.

Another study made by Hansson and Vidhall [19] obtained similar results to the one performed by Danielsson [26]. Their experiments, however, showed a larger difference in the CPU utilisation during the startup of the application, where React Native had a CPU utilisation of more than double of the one used by the native app, averaging 14% and 6% respectively. This is due to the fact that React Native has to start the JavaScript thread besides the main thread, create the Virtual DOM in the JavaScript thread, and send it through the bridge to the main thread, so it can render the application.

Their results also show that React Native had a larger app size compared to Android, which is expected since it needs to store the framework in the application itself.

Overall, it is clear that the CPU utilisation is higher in React Native compared with native apps, as would be expected. The other metrics, however, were mostly similar.

2.3.2 Flutter

Flutter is an open-source cross-platform framework that is developed by Google and was first released in 2017. Flutter requires Dart, a programming language created by Google as well, to develop its applications. Like React Native, Flutter can be used to develop applications for both iOS and Android, but it can also build apps for Fuschia, Google's next-gen mobile operating system [25].

Besides being able to develop apps for mobile devices, Flutter can be also used to create web and desktop applications [31]. To help the development of these apps, Flutter supports hot reloading, pre-made widgets to give a native look and feel, and a high-performance rendering engine.

Similarly to React Native, there are many applications and companies currently using Flutter, like BMW Group, Alibaba Group, eBay, Tencent, Toyota, and many others [32].

2.3.2.A Dart

The main goal of Dart is to build high-performance scalable web applications. It can be used to develop both the frontend and the backend of the application. It is being used by Google internally since JavaScript could not provide the features and, above all, the performance needed for their web apps [18].

Google created Dart with the intent of replacing JavaScript, so naturally, it supports most of JavaScript's features. However, Dart has a Java-like syntax [25], and it is not similar to JavaScript.

JavaScript allows for various unpredictable programming errors in runtime due to its ”little declarative syntax” [18], allowing the assignment of wrong types for instance. To fix this, Google made Dart a type-safe language or, in other words, it has a *sound type system*. This means, for example, that a variable of type *String* always remains as a *String*. Type soundness is possible due to the combination of static checking and runtime checking [33]. Some of the advantages of this approach mentioned in Dart's development guide [33] include: "more readable code", "more maintainable code", and "type-related bugs are revealed at compile time".

2.3.2.B Widgets

Everything in Flutter is built using widgets. For instance, styling elements like margins, padding, and so on, are built using widgets. Unlike React Native, Flutter does not use native UI elements. Instead, it offers widgets with the look and feel of Android and iOS, named Material and Cupertino widgets, respectively [18]. Widgets also inherit properties from their parent widgets.

Regarding their state, widgets can either be stateless or stateful. Stateless widgets do not have a mutable state and they extend the class *StatelessWidget*. Stateful widgets extend *StatefulWidget*, and as the name suggests, they have a mutable state, however, since widgets are immutable, when the state is changed, a new widget is created to replace the old one [34].

2.3.2.C Rendering

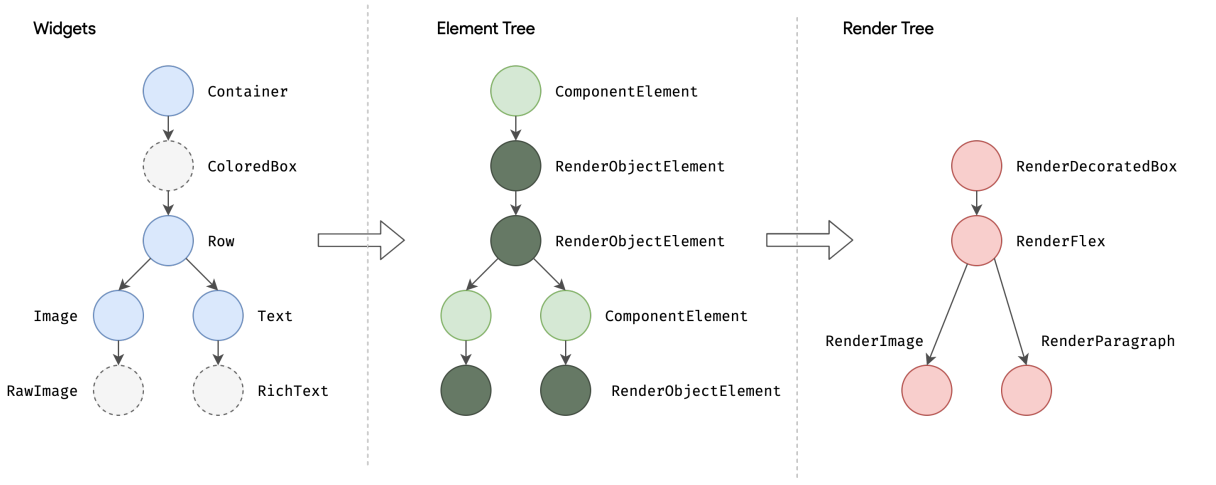


Figure 2.3: Flutter trees diagram. Image retrieved from [2]

To render its widgets, as shown in Figure 2.3, Flutter's widgets are organised in three different trees [2]:

- The **Widget tree** contains all widgets necessary to create the user interface;
- The **Element tree** is made of elements that were created from the widgets used in the Widget tree by calling a specific method on the widget. Elements are an instantiation of a widget that is mutable, unlike widgets. Since they are mutable, one of the main purposes of this tree is to manage the widgets' state;

- The **Render tree** for each element on the Element tree, Flutter calls a method to create a *render object*.

When a widget is changed, a new one is created and replaces the old one in the Widget tree, since they are immutable. Then if the new one is of the same type, the *element* can just be updated with the reference of the newly created widget and the *render object* can be updated as well, instead of creating a new *render object*, which is computationally expensive. Otherwise, a new *element* and a new *render object* are created [18].

Additionally, the widgets instead of being rendered at the system level, are rendered at the application level, which makes them more customizable but with the trade-off of the larger application size [25].

2.3.2.D Performance

According to Flutter's own main page, Flutter has achieved near-native performance, mainly due to its Dart compiler that uses both a Dart Virtual Machine with just-in-time compilation, which compiles the code during the execution of the program, which provides support for hot-reloading, and an ahead-of-time compiler that compiles the code into a lower level one (ARM in this case) before execution [2].

This high performance although, is not all due to the compiler. The use of three trees, promotes separation of concerns, that in turn increases performance. Pre-included widgets are also tuned with performance in mind. The *ListView* widget for instance, only renders the items in the list that are currently visible [35]. It can not only run at 60Hz, as a native application, but also at 120Hz if the device supports it, augmenting its visual performance.

In a study made by Fentaw [31], the performance between Flutter and React Native on both Android and iOS was tested and compared, providing interesting results. In all tests, the values registered resulted from the mean of five test runs for each of the 4 different tests performed.

In Android, no framework stood particularly out in the case of CPU usage, since in half of the tests React Native had a better CPU usage and in the other was Flutter. As for memory, both of them had similar results, except for one test case, where Flutter had clearly better memory management.

In iOS, on the other hand, results were drastically different in both CPU and RAM consumption. In all CPU tests, Flutter always had better efficiency than React Native. In memory usage, the results were the opposite, where React Native in some tests had five times less memory usage than Flutter. In GPU, however, both frameworks obtained similar results.

2.3.3 Progressive Web Apps

In an attempt to overcome the web apps' disadvantages and promote their use, Google created a new concept called Progressive Web Apps, which had the goal of closing the gap between web apps and native apps combining the best features of each one. In order to accomplish this, PWAs have the following features [36–38]:

- **Homescreen installation**: PWAs can be installed on the device and, in some operating systems, like Android, the system treats them as a normal native app, even during the installation process. The size of PWA apps, when compared to other approaches is much smaller, as shown in [36], where the size of the PWA app was approximately 157 times smaller than React Native. This is mainly due to its web app similarities, where the application is rendered using a browser, so the application does not need a framework or a layer translating the source code;
- **Background synchronisation**: This kind of application can use background synchronisation as any other native app can;
- **Offline support**: With the capability to be installed on the device, once downloaded and installed, these apps can run offline, since the elements of the web pages are cached on the device, which can be useful for apps like games, or simple apps that do not require access to the internet;
- **Push notifications** are available, but they are limited to browsers that support it;
- **Trying the app before installing**: Another important feature that originates from its web app similarity is the ability to try the app on the browser before deciding whether or not to install it;
- **Available in app stores**: PWAs can also be available in app stores. As of now, only Microsoft Store for Windows and Google Play Store for Android allow PWAs to be published to them, not being supported by the Apple App Store, which is a major downside for cross-platform solutions.

For a browser to be able to support some of the features aforementioned above, it needs to support the Service Workers API. This API runs on a separate browser thread and is responsible for providing features present in native apps, namely [38]: background synchronisation, push notifications, caching mechanisms, and interception of network requests [36].

Another key functionality besides the Service Workers is the App Shell, which caches offline all interface elements needed like HTML, CSS, and JavaScript to decrease the number of Hypertext Transfer Protocol (HTTP) requests sent, to increase performance [39]. Although its performance is not as good as native apps, it is close to it, but only if the browser is running in the background, otherwise its performance can be up to three times worse, as shown in [36].

Despite all of these improvements over traditional web apps, PWAs still have some downsides. For instance, not all browsers have support for the Service Workers API equally, so features can vary from browser to browser, and they are not available in all app stores, like the iOS App Store. As mentioned before, if the browser is not running in the background, it is much slower. Additionally, despite being

Table 2.1: Comparison between PWAs, React Native and Flutter

Features	PWAs	React Native	Flutter
Language	Web technologies	JavaScript	Dart
Hot reload	Depends on the framework used	✓	✓
Availability	Can run on a browser, cannot be submitted to the App Store	All app stores	All app stores
Hardware features	Partial support	All features can be used and implemented when missing	All features can be used and implemented when missing
UI	Native like UI can be obtained using third party libraries	Native elements	Elements that mimic native ones
Performance	To perform well the browser needs to be running in the background	Similar to Flutter in Android, but worse efficiency on iOS	The fastest overall, but not as fast as native apps
Size	Extremely small	Slightly less than Flutter	The largest

able to support more features, there are still some features that native apps have that are not available in PWAs.

An example of a progressive web app is *Google Maps Go*[2], which is a lighter version of the original *Google Maps* app that occupies ”100 times less space”, as stated in the app description, made especially for low-end devices. This version of *Maps* is even pre-installed on *Android GO*[3], a lightweight version of Android made for such devices.

2.3.4 Choosing the Right Framework

Unless time and money are not an issue, native apps always provide the best experience possible and are the best approach. Otherwise, cross-platform frameworks, like React Native, Flutter, or Progressive Web Apps, provide a native-like experience, both in terms of looks, features, and performance, whilst reducing development costs by a large amount.

Choosing between these frameworks also depends on the budget and the type of application that is going to be developed. Progressive Web Apps are a good choice for simple apps, especially for low-end devices, since most developers are familiar with web development, they use few resources, and they are small in size. However, the lack of support from the Apple App Store is a major drawback for a cross-platform solution. Even when installed from a web browser, the iOS does not treat it like a normal application, not showing in the application drawer, for instance.

React Native and Flutter both support most hardware features, have a native-like UI, and have great

[2]Google Maps GO – `https://play.google.com/store/apps/details?id=com.google.android.apps.mapslite`
[3]Android GO – `https://www.android.com/versions/go-edition/`

performance. If the fastest performance possible is one of the requirements of the app, Flutter is possibly the best choice, as seen in the study performed by Fentaw [31]. This is especially true in iOS, where the CPU efficiency was drastically better than React Native, and GPU performance in Frames per Second (FPS) was also better. This is due to the fact that the code is directly compiled to ARM machine code. However, this comes at the cost of being developed using Dart, which most developers are unfamiliar with, indirectly increasing the development cost of the application. If performance is not an issue, React Native is the best choice overall to develop a new application, due to the use of JavaScript, Native UI elements, support for all hardware features, and a big development community.

Table 2.1 summarises and compares the characteristics and supported features of these 3 approaches.

2.4 Mobile App Development

Developing mobile applications is not only about just choosing a framework, developing the app, and shipping it. There are several other factors that contribute to the success or failure of mobile apps, and developers and companies should be aware of them in order to build a successful app.

2.4.1 Best Development Practices

One should always strive to build the best possible product, and the same goes for mobile applications. To achieve this, [40] describes what should be avoided and the best practices in order to develop a successful application. Among the mentioned practices, the following general principles should be followed:

- Adhering to user standards, which the most relevant are:
  - Good performance;
  - Intuitive and good-looking user interface, that also complies with the OS's design language;
  - Strong security;
  - Robust code, avoiding spurious crashing;
  - Support of similar features on all platforms (fair use for all platforms);
  - Should not drain a lot of battery, which is essential in a mobile device.
- Following the development life cycle phases. Complying with this life cycle ensures that the most important aspects of software development, like requirements gathering and testing, are not overlooked, and so, most of the common issues can be avoided. This cycle is going to be described in more detail in Section 2.4.2;

- Developers should take user feedback into consideration. User feedback can be obtained from user testing, before launching the application, through surveys and reviews on the OS's application store. It is also important to build into the application itself, a way for the users to give feedback;
- Avoiding tedious and difficult app registration process, which can turn users away from using the app any further, eventually leading to uninstalling the app itself;
- Balanced use of notifications. Too many may lead to turning off notifications or uninstalling the app, too few may lead to forget the application;
- Testing and maintaining the application properly, namely, coping with the OSs' evolutions, can prevent several bugs and security issues from compromising the user experience.

Failing to comply with these good practices can have both short and long-term consequences. In the short term, revenue is going to be lost, due to bad reviews on the platform's application store, which users normally verify before downloading a new app. In the long term, users might not want to install a new app made by a company that previously made an undesirable app, so companies can struggle to attain success in future projects.

There are other aspects that can lead to the success of mobile apps that are outside the scope of a developer, like marketing, which is paramount these days for the success of applications, but this factor is not the focus of this thesis.

2.4.2 Development Life Cycle

As discussed in [40], traditional software development life cycles, like the ones described in [41], containing planning, analysis, design, implementation, testing, and maintenance phases, can be used successfully for the development of mobile applications, helping increase their overall quality.

In a survey performed by [42], mobile app companies, developers, researchers, stakeholders, and experts were asked what were the best and common methods and practices to apply during mobile app development. The conclusion reached from this survey and others performed in the past was not far from the aforementioned traditional software development phases.

Since the traditional software development life cycle can be applied to the development of mobile applications successfully, the following phases should be followed throughout the product development:

- **Requirements Gathering or Analysis**: this is possibly one of the most important ones, where both the functional and non-functional requirements are gathered. Functional requirements describe the features and services of the application, and the business requirements, which detail purpose, target audience, goals, and data requirements, among other aspects. In sum, they can answer the question of *what* the application can and cannot do. Non-functional requirements, on the other hand, can answer to *how* does the application accomplish what it does. These can be performance, security, usability, and so on;

- **Planning**: In the planning phase, as the name suggests, a plan of the entire product life cycle is made, with the main purpose of estimating time and cost. Risk assessment can also be done in this phase, where multiple project plans can be developed to replace the original one if needed;
- **Design**: The goal of this phase is to build mock-up screens and wireframes showing the interaction between the application's screens or even prototypes for client approval. This is done before the development, so the time and cost needed to change something during development are reduced;
- **Development**: This is the longest phase of the life cycle, and is where the code starts to get written, ultimately turning the design mock-ups into reality, while following the requirements established before. This phase should be performed with an agile approach, Scrum [43], for instance, where, in each spring functionalities or smaller modules are implemented one at a time, and tested at the same time to fix bugs as early as possible, reducing costs. This testing is often overlooked by developers and often leads to "buggy" and unreliable applications. Regression testing is essential while using this approach, making sure that new functionalities do not break previously developed ones;
- **Testing**: After the development of the application, it should be carefully tested against the requirements and designs obtained from the previous phases. In addition to the unit tests, regression tests, and implementation tests done during the iterative development phase, with an agile approach, user acceptance tests and system tests should be performed to ensure that the final product meets the client's expectations. Besides these tests, tests that verify the non-functional requirements, like performance testing should also be done [44];
- **Deployment**: After being certain that the app complies with the requirements through testing, it is time to release the app into the compatible operating systems' application stores;
- **Maintenance**: After the release of the application to the public, it is likely that some users are going to encounter some issues with the application, so it is important to perform maintenance on the application, releasing updates regularly to fix those issues. The development of new features can also be important to keep the users engaged in the app.

2.4.3 Challenges

Even when following the recommendations above, mobile app developers still encounter some issues and challenges, as demonstrated in surveys performed by [45] and [42]. Some of the main issues encountered were:

- The existence of multiple platforms, like iOS and Android, can be an issue, especially if they tend to follow divergent paths for their interfaces. Developers have to keep up with the different APIs and design languages from both platforms. Even inside the same operating system, which is the case of Android, a multitude of different devices can exist, with different screen sizes, resolutions, memory, CPU, and so on. This implies extra work for testing and optimising the app for multiple devices, especially low-end ones. And, even so, it does not guarantee the intended experience for all devices;

- Testing, although essential, can also be a big challenge, since not only do apps need to be tested on emulators and then on real devices, but also the tools available for doing so are not as robust as the ones for desktop software. In one of the studies in [45], the participants complained about the limited support for automation testing in mobile applications, stating that the tools and emulators lacked good analysis tools to give insight into the device's metrics, and some of the features needed for testing were not available, like sensors;

- Security can also be difficult to guarantee with the plethora of platforms and devices, due to the different ways they manage data;

- User expectation is getting higher and higher, so delivering mobile applications with good quality can be difficult. Requirements gathering and testing can help increase the quality of apps and meet user expectations;

- For the ones used to develop software for desktop platforms, designing and building the UI/UX for a mobile device can be quite challenging as well, so it is important to validate them with users;

- The application maintenance itself can also prove to be a challenge. Developers need to keep up with APIs, comply with new requirements, and improve the overall experience through fixing bugs or issues, for instance, publishing updates whenever required [46].

2.5 Similar use cases

The subsequent subsections describe similar applications to the case study of this thesis.

2.5.1 Current IST App

As mentioned in Section 1.2, a new academic mobile application for Instituto Superior Técnico (IST) is going to be developed to validate this work. The current one [47] was originally developed in 2013 as a Master's Thesis by Barata [48], in native Android. A year later, the development of the application for the iOS platform started, through native development again, by both Barata and Direção de Serviços de Informática (DSI), which is responsible for the development and maintenance of IST's digital infrastructure. Since it was built through native development again, the application was developed from scratch using Swift, which is a programming language used exclusively for developing apps for the Apple ecosystem, and a slightly different design to conform with the iOS design patterns.

Since DSI has many other projects, and the team is relatively small, resources need to be optimised. Since only a few developers can program in native iOS and Android, the application had few updates. This is clearly visible, for instance, in the news section, where sometimes the news and courses' names are in Portuguese, even though the application's language is in English. This is one example among other issues, like duplicated courses in the curriculum feature, push notifications not working, and others that were already reported by users.

A cross-platform solution will enable to have only one one codebase for both platforms instead of two, and will ease the overall development effort.

The current app supports the following features, although some of them are currently working with limited functionality:

- Login into the user's account using the Open Authorization (OAuth) protocol;
- News feed with school news and courses' announcements. The feed is also personalisable;
- Cafeteria's menu for both campuses;
- Schedule of the shuttle that connects both campuses;
- Available capacity for the parking lots;
- Possibility to withdraw a queue ticket from the university services for on campus assistance;
- Courses' information;
- Evaluation schedule and enrolment;
- Curriculum information, which courses are completed, their grade, the student's average grade, completed credits and years;
- The teachers can publish their classes' summaries;
- The Staff can report and review their assiduity;
- Outstanding and paid payments;
- University's contacts;
- Synchronisation of the student's schedule with their smartphone's calendar;

- Both Portuguese and English language support;
- Help set up the university WiFi, by installing certificates if needed and logging into the users' accounts.

2.5.2 IST GO

Along with the pandemic, came many contact tracing apps like STAYAWAY COVID [49], among others. IST Go [50] was one of these apps, and as the name suggests, it is being developed for the IST community. However, this app was not like the other contact tracing apps that mainly used Bluetooth and GPS to determine the location of the user. Instead, it analysed the number of devices connected to the Access Point (AP) in each university building to determine its occupation. Unlike the other apps, the information was not sent to another entity but shown to the user.

The application was developed using Flutter, and has the following main features:

- Using a map, it shows how many people are in each building and in each room inside it;
- Since it is much harder to detect how many people are inside each classroom using the APs, users can report how many people are inside it and can request other users to report that information;
- Users can make a reservation for a classroom, indicating the time and number of people;
- There is also a survey area to facilitate their proliferation among the community;
- To encourage students to use the app, gamification was introduced. Each action performed, like answering a survey, gives the user experience points, which makes a user level up and climb its way up the leaderboard.

2.5.3 Other Academic Applications

Although there are a great number of apps for universities, there are only a few papers that document this process [51]. In one of them, similar to our case study, a new mobile application for the LNU university (Linnaeus University) was needed [52]. Before developing the new application, the requirements for the new one had to be gathered. To do this, some steps were taken, the most important ones being:

- Survey students about the quality of the existing app and what features they considered important to have
- Investigate other universities' applications

The results from their second step can be interesting and helpful for our case study. The features supported by other universities, listed from the most common to the less common features are:

- Academic schedule and map, around 50% of universities offered them in their apps;
- School's information;
- Events/news;
- Cafeteria menu;
- Staff info;
- Feedback;
- Transportation, information on how to get to the school, either by train, metro or other means of transport;
- Reminders and Exam results, with only a 7% adoption rate. The latter, as mentioned in [52], would not be feasible, due to the amount of changes needed in the backend and the school's API to allow such a feature.

In [52], it is advised to develop the application in an iterative way, instead of an incremental way, for the same reasons aforementioned in section 2.4.2. The importance of feedback is also emphasised and should be dealt with in the maintenance phase. Besides user feedback, the collection of user data can also prove to be useful.

Similarly to [52], in [51] a study of features present in other universities' apps was also performed, revealing similar results. Besides this, the architecture of their application is described as a simple interface that performs API calls to the school's server to retrieve information. To reduce the number of API calls, in order to increase performance, the app contains a local database that stores user preferences and other information that does not change so frequently, like the student's courses. Although the application was only released for iOS initially, it was developed using a cross-platform framework, PhoneGap, which makes it easier to release the app to other operating systems in the future.

3 Proposed Solution

Contents

After describing the different types of mobile applications and discussing the best alternatives to native apps, the definition of the application requirements is a crucial step to be able to make a good decision on the best development approach.

3.1 Requirements

There are multiple methods to gather requirements that can vary in both time and money. As described in [53], the most relevant ones are:

- ”**interviews**”: allows to obtain information from individual users through interviews;
- ”**surveys**”: questionnaire to the target audience to obtain information, such as user needs;
- ”**wants and needs analysis**”: is a brainstorming activity between the team members, the product owner, stakeholders, and others involved in the requirements definition, that seeks to create a priority list of the potential users’ desires and needs to validate the current requirements and find new ones;
- ”**group task analysis**”: records the steps taken by a user to complete a predefined task under analysis to later on be able to improve it if necessary.

Due to the time constraints of this work, only part of these methods were adopted, namely ”surveys” and ”wants and needs analysis”.

3.1.1 Original App Requirements

Since the new application is meant to replace the current one, its requirements and features should also be taken into account and analysed. The original functional requirements for the application, defined in [48] are divided into four categories, depending on the user role, and ordered by their priority (highest priority first). The original requirements were the following:

- **General**: ”news feed, available courses, institutional contacts, find points of interest nearby, finding buildings and rooms, schedule and bus-stops of the IST shuttle”;
- **Students**: ”course announcements, class schedule, academic calendar, personal curriculum, fees and payments’ information, courses, classes, and groups enrolments and exams registrations, synchronise digital calendar with the IST calendar, and available rooms”;
- **Professors**: ”academic schedule, academic calendar, synchronise digital calendar with the IST calendar, show and add courses’ announcements, show and add classes’ summaries, show students/groups, and information about received documents”;

- **Alumni**: ”personal curriculum, and fees and payments’ information”.

Comparing the features available in the current application (Section 2.5.1) with the original requirements, some features initially deemed to be a high priority, such as students’ class schedules and the academic calendar, among others, are missing from the current application, and need to be reconsidered if whether or not they are still going to be kept as requirements.

3.1.2 Survey

Following the approach described in [52] and in [53], a user survey was made (Appendix A) and disseminated to the IST community through the IST website and social media. The goal of the survey was to gather feedback on the current application, user satisfaction, features that need to be improved, and suggestions for new ones. 167 responses were collected from students, 16 from professors, six from staff, five from alumni, and two from investigators, in a total of 175 responses. Note that it is possible to have more than one role at the university.

Concerning the non-functional requirements (Figure A.3, Figure A.4, Figure A.5, and Figure A.6), the feedback was overall positive, averaging 3 out of 4 in all of them, except for the User Experience (UX) (Figure A.3). This can be explained by the fact that there were many features that were reported as not working properly, as it can be seen in Figure 3.1.

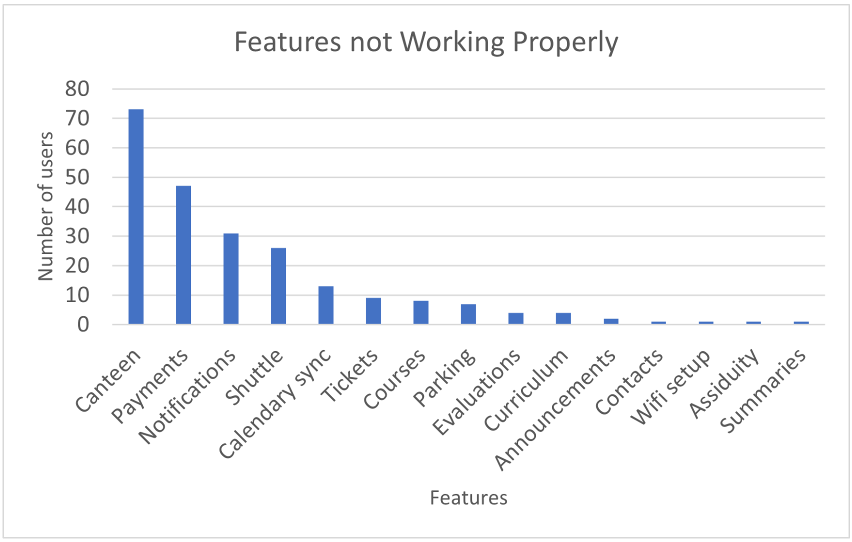


Figure 3.1: Features not working properly in the current app

Most of these responses were expected since some of the original features are not working properly due to changes in the Fenix API that were not updated in the app, or the API itself needs updating. This large number of unavailable or deficient features also leads to user dissatisfaction, which is visible in Figure A.8. Over 50% of the responses, when asked about the deficiencies of the application and reasons for not using it, identified some features not working as they should (Figure A.12). This overall dissatisfaction is also shown in the overall application usage (Figure A.2), where a third of the users answered almost never using the app and the other third answered only using it sometimes.

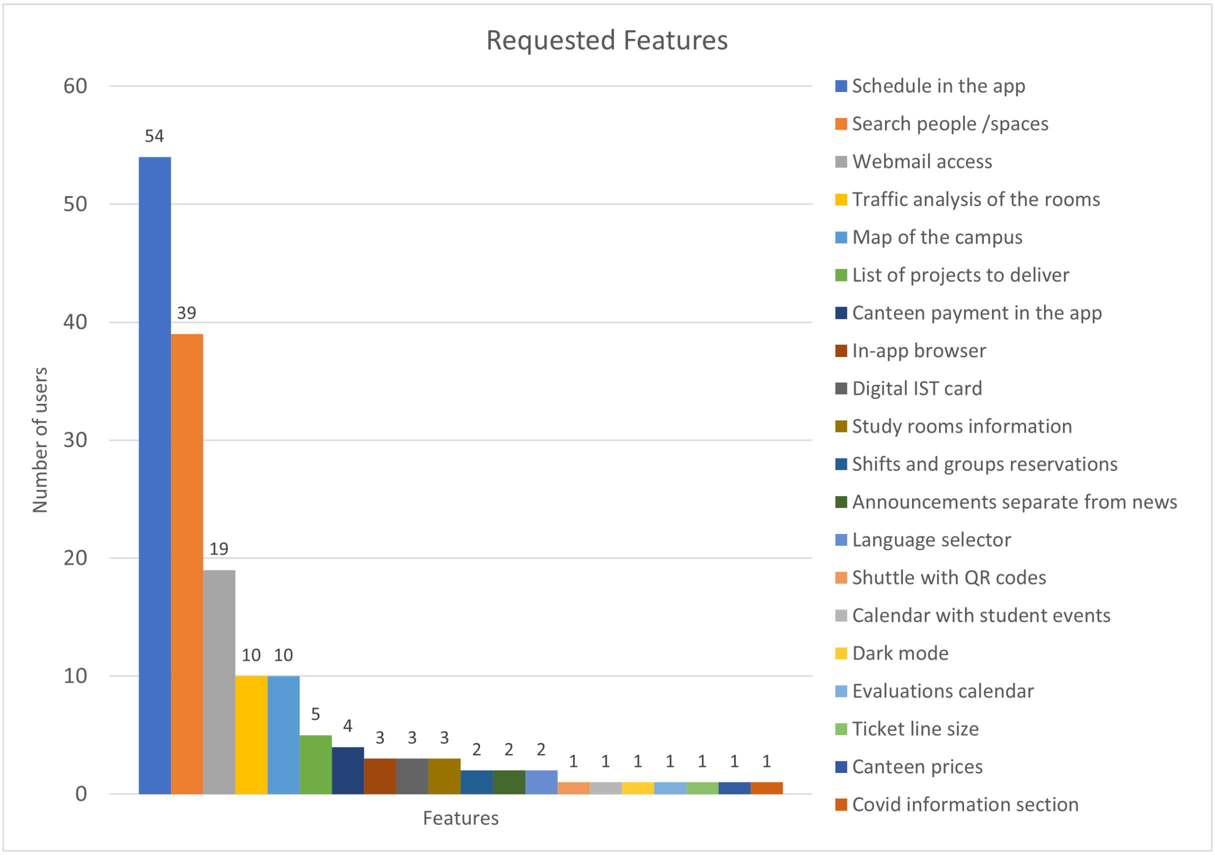


Figure 3.2: Feature requests for the new app

As for the suggested new features (Figure 3.2), the two most requested ones were the student and teacher class *schedule in the application*, and the ability to *search people and spaces*. These requirements were already identified in the original application, as described in Section 3.1.1. In particular, the *schedule in the application* was listed as the highest priority requirement for faculty and the second highest for students. Some of the other suggested features show a correlation with the disliked aspects of the application (Figure A.8), namely as the *in-app browser*, the *language selector*, *webmail access*, *search people/spaces*, and *dark mode*.

3.1.3 Defined Requirements

Following the "*wants and needs analysis*" method described in [53], a meeting was held with some members of the DSI team to discuss the aforementioned survey results (Section 3.1.2) and to define the application requirements. In this meeting, the disliked aspects (Figure A.8), the features not working properly (Figure A.10), and the suggested features (Figure A.11), were predominantly discussed and analysed, to decide whether or not they should become a requirement.

3.1.3.A Functional requirements

The first and most important requirement established was that the new application should support all the features of the current application, as described in Section 2.5.1, except for the *Assiduity* feature for the university staff, since the system that manages human resources moved to a new platform, for which the API is still not available. Besides supporting the current features, the issues identified by the users in the survey needed to be addressed and fixed as well, mainly the most reported ones, for instance: the canteen feature is not showing any information regarding the meals, the payments feature is not showing up to date information, notifications are not working, the shuttle is not displaying any information, and fixing the calendar sync with the users' mobile device (Figure A.10).

In order to define what needs to be improved, the features suggested by the respondents (Figure A.11) were analysed. Since the two most requested features, the *schedule in the application* and the *search people/spaces*, represented 57% of the requests and, were initially part of the original requirements, they were considered part of the requirements of the new application as well. Aside from these, and based on the requests from users, some new features also became part of the requirements:

- An in-app browser to avoid going out of the application to open Uniform Resource Locators (URLs), especially URLs belonging to the IST domain, making the in-app navigation easier;
- An option to select the desired application language on the settings page. By default the application should choose Portuguese if the platform's language is in Portuguese, and English otherwise;
- Dark mode for the application tied with the device's dark mode preferences;
- Using the students affluence already registered by the libraries, show the occupation of the libraries, their location and business hours;
- Show the location and business hours of the study rooms;
- Separate courses' announcements from news, to facilitate their visualisation and consultation.

Besides these, considering that DSI has a guide available on how to add the user's university email account to an email client, it was concluded that adding a feature to read the user's email would require to develop or emulate an email client, what would be complex and out of scope, duplicating already available apps. So, only a reference to that guide or instructions on how to do it is required.

3.1.3.B Non-functional requirements

As for non-functional requirements, based on the list of non-functional requirements in [54], the following ones were defined:

- **Performance**: as described by Nielsen [55], for the UI interaction to feel responsive and the application to react instantaneously, the response time should be of 0.1 seconds. If an immediate response is not possible, the application should respond in one second or less, without the need of showing any feedback. For loading times, they should not exceed 10 seconds, and should always provide some feedback to the user, indicating that a task is being performed by the system;
- **Usability**: the application should be at least, as intuitive as the current application is (Figure A.4), and should provide a better user experience than the current one (Figure A.3), since the issues identified in the current application will be resolved;
- **Compatibility**: the application must be supported on older devices, to reach as many users as possible. In iOS, due to its better upgradability when compared to Android, this is not a problem since most devices (approximately 85% as of January of 2022 in Portugal) are running the latest or previous version of iOS [56]. With this in mind, at least iOS 14 would need to be supported. To decrease the number of unsupported devices, iOS 12 should be supported. As for Android, more than 94% of Android devices in Portugal are running Android 8 or newer [57], so at least this version needs to be supported;
- **Security**: user accounts should be securely stored on the device through encryption. Any other data that is not private should not be accessible by other means besides the application itself. To reduce the possibility of attacks, like Man in the Middle attacks, only secure connections should be used, such as Hypertext Transfer Protocol Secure (HTTPS). Furthermore, the application should not be able to retrieve information from any domain not belonging to the university. Users should only be able to perform actions and access features they have permission for through their roles.

3.1.3.C Fault Model

As part of the application's requirements, the types of faults that it can sustain should also be established. The following use cases should be supported:

- If there is no internet connection available, the application must show the latest information retrieved, depending on the feature being used, so the user can still access it even without an internet connection. Otherwise, feedback that no information could be retrieved should be provided to the user;

- Similarly to the previous use case, if an API request fails to return valid information, the previous retrieved information, or feedback that it was not possible to retrieve it, should be provided to the user;

- When a user is editing information, for instance, writing a class summary, and the request is not completed successfully, feedback should be provided to the user, and the already filled information should persist to enable to complete the operation when connectivity recovers, without having to retype the whole content.

3.2 Choosing the Cross-Platform Framework

To choose between the frameworks described in Section 2.3, various aspects were taken into consideration. PWAs could have been a good choice, due to their small size, the use of familiar programming languages, good hardware feature support and their ability to be installed on the device. Although the latter is true, the fact that PWAs cannot be submitted to the App Store makes this choice unfeasible. Also, despite having support for the most commonly used hardware features, some features like syncing the calendar without user input, which the current application supports, are not currently possible on PWAs, leaving only React Native and Flutter.

Since performance is not a critical requirement for this type of app and the fact that DSI, which will be maintaining the application in the future, uses mainly JavaScript for their frontend projects, React Native was considered the best choice overall to develop IST's new academic application, due to the use of JavaScript, Native UI elements, and a large development community.

3.3 Application Architecture

To keep the application as light as possible, and to be easily run on a multitude of devices, the app is going to depend mainly on the faculty academic management platform, Fenix. Fenix API provides an API [58], which supplies most of the essential information needed, namely: information about the user (courses, curriculum, payments, schedule, evaluations), the courses, the university's spaces, and contacts, among others. This API is also publicly available. Although, due to recent changes in the backend, some endpoints were not returning the most updated information, as mentioned previously. To remedy this, for the most part, the Fenix API is currently being updated, except for the *Shuttle* feature, where a new API was used. Similarly to the ongoing application, the *Ticketing* feature, which allows to withdraw tickets from the university's services, is supported by another API as well. The endpoints for the APIs being used are listed in Appendix B.

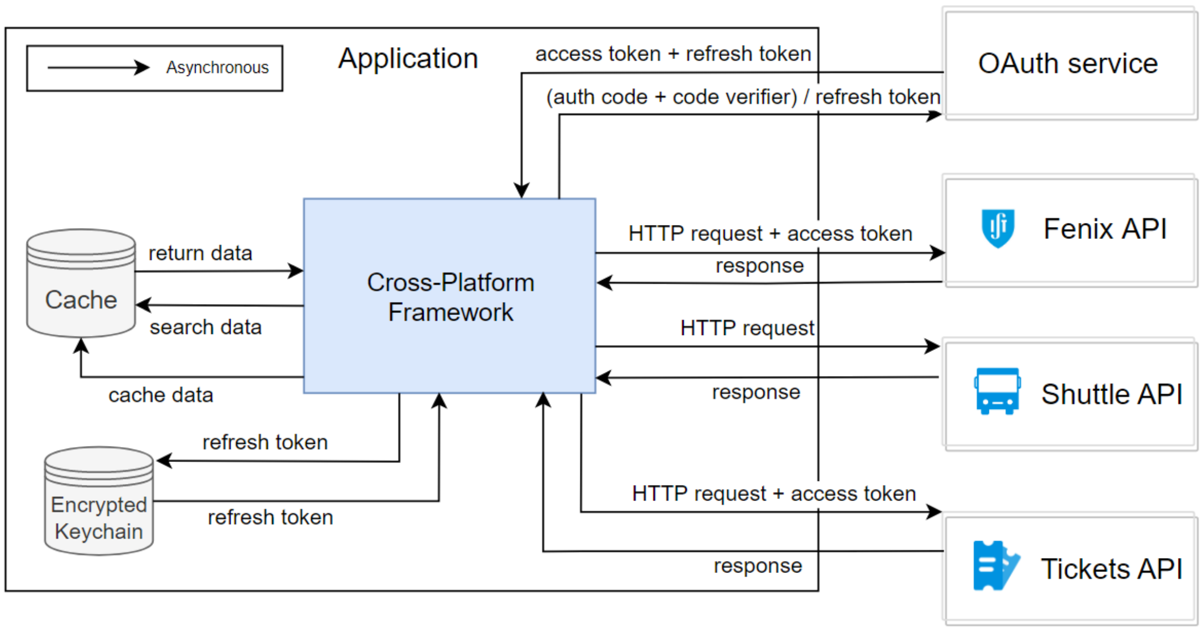


Figure 3.3: Simplified application architecture

In order to have access to user-protected information, the Fenix and Tickets APIs need, in some way, to use the user's account. For this purpose, the Fenix backend supports authentication through OAuth [59], for which a specific OAuth protocol flow was implemented. After the user signs in, it is important to understand what kind of roles the user has, in order to show different features depending on the role. For instance, users with the *teacher* role need to have the *summaries* feature to write courses' summaries, but the rest of the roles do not have permission to do so, so this feature should not appear to them. It is also important to note that users can have multiple roles.

Relying solely on APIs to get the necessary information means that the application would only work with an internet connection. To overcome this restriction, in a similar fashion to [51], a cache was implemented to provide the last information retrieved whenever it exists, instead of not showing any information at all. In most cases the cached information would be the same as the current one, since most of it does not change frequently, apart from the news and courses summaries. A simplified version of the application's architecture is shown in Figure 3.3.

3.4 Authentication

OAuth is an open standard authorisation protocol that provides services or apps the option to allow secure authorisation by using tokens instead of credentials [60].

In an OAuth flow, after a user authenticates and authorises the service, an access token is used instead of the user account, improving security and privacy at the same time. Since it was going to be implemented on a mobile application, which was going to be public, it could be exploited through authorisation code interception, where a malicious app is running alongside the legitimate application and intercepts the authorisation code, allowing it to obtain the access token afterwards, and impersonating the user. To prevent this, an OAuth with Proof Key for Code Exchange (PKCE) [3] mechanism was used.

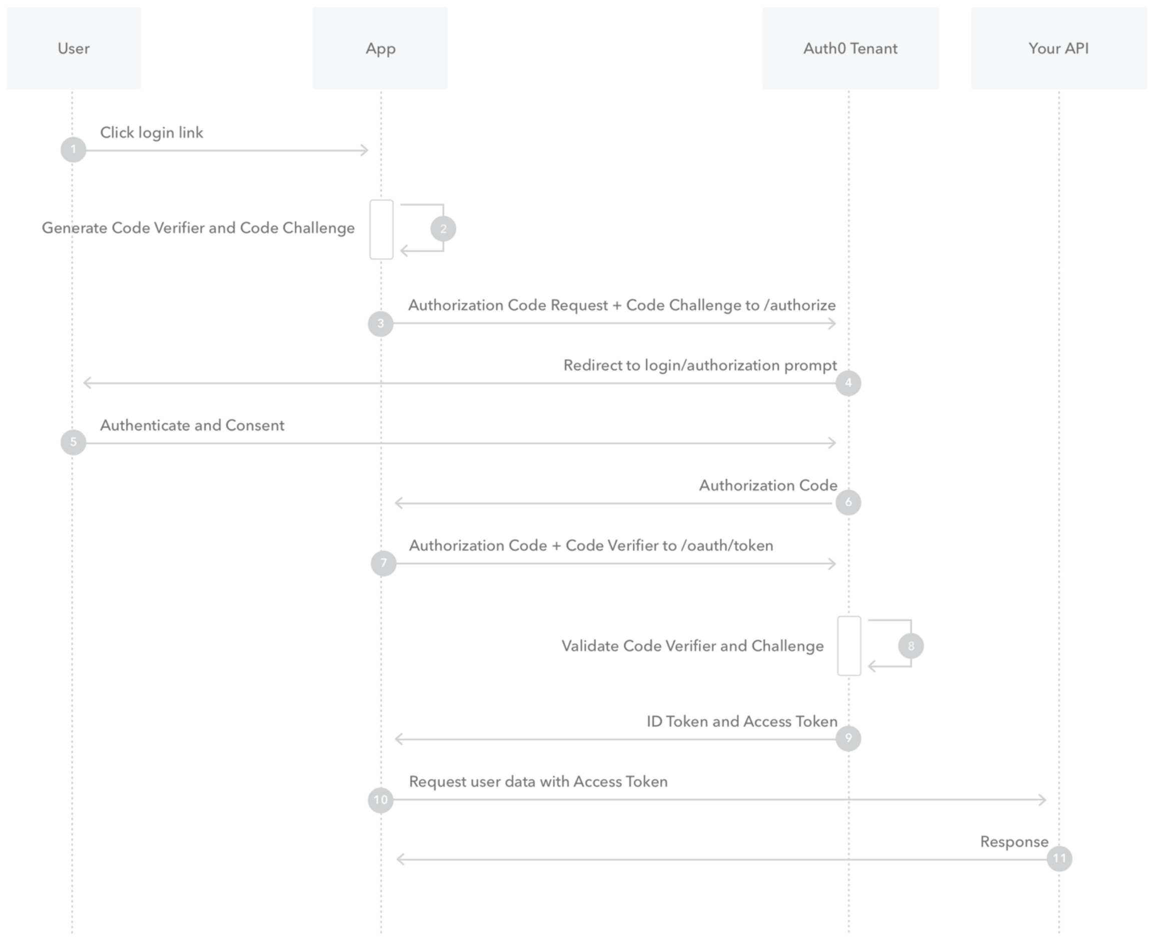

User
App
Auth0 Tenant
Your API
1 Click login link
Generate Code Verifier and Code Challenge 2
3 Authorization Code Request + Code Challenge to /authorize
Redirect to login/authorization prompt 4
5 Authenticate and Consent
Authorization Code 6
7 Authorization Code + Code Verifier to /oauth/token
Validate Code Verifier and Challenge 8
ID Token and Access Token 9
10 Request user data with Access Token
Response 11

Figure 3.4: Authorisation code flow for OAuth with PKCE. Retrieved from [3]

This variant instead of only sending the authorisation code request, it creates a random string, called the *code verifier*, that is usually encrypted with SA256, creating the *code challenge*. This code challenge is then sent alongside the authorisation code request. After the user authenticates and authorises the service, an authorisation code is returned. To prevent the aforementioned attack, then the application sends the authorisation code with the code verifier in plain text, for the authorisation server to verify the user. Using this approach, even if the authorisation code and the code challenge are intercepted, it is not possible to obtain the access token, since it is not possible to obtain the code verifier from the code challenge. With this, authentication with OAuth with PKCE provides a simple way for secure access to user private information or role-based protected activities. The diagram of this flow is shown in Figure 3.4.

To prevent the user from having to log in every time the application is used, a *refresh token* is sent with the access token. This token can be stored securely on the device and allows one to obtain a new access token without performing the aforementioned process. A certain duration is assigned to the refresh token, so when a new access token is requested, a new refresh token is provided as well.

3.5 Interface Design

The web design team at DSI had already been working on a design for a new mobile application for some time before the start of this thesis. This new design is based on the new design language that is currently being implemented on every service provided by DSI. Similarly to other design projects, it is being developed using *Sketch*[1], which allows for multiple users to work on the same project, build wireframes, and prototypes, and share design elements like icons, fonts, and others, between projects. When the design of a screen, or feature, is complete, the designs are uploaded to *Zeplin*[2], which gives developers easy access to templates, which are organised by features, allowing them to access design items, like icons, fonts, images, the spacing between elements, and others, with ease. Furthermore, Zeplin also provides extensions that allow code generation for design elements, allowing developers to focus more on developing the logic of the application and not the design, as demonstrated in Figure 3.5, where Zeplin generated the style for a text component in React Native.

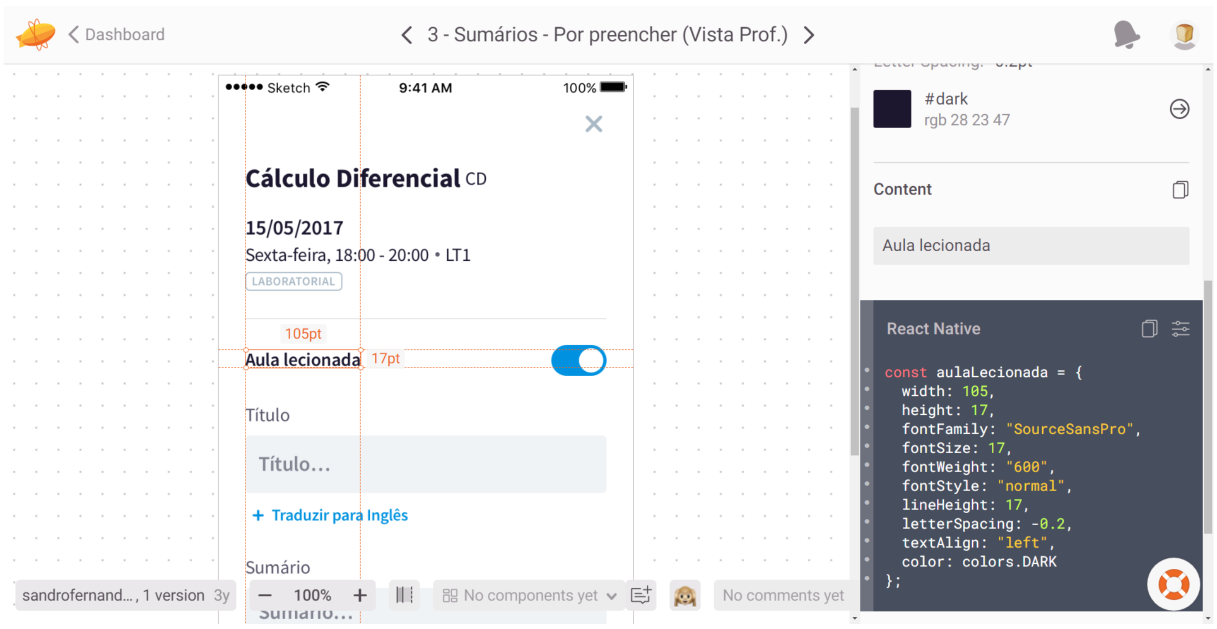
Since new features were going to be implemented, their design needed to be drawn. To do so, the remaining features were designed based on:

- The new existing designs;
- The design language being used;
- The design goals described in [61], namely: ”it makes it easy to recover from mistakes”, ”it has uniformly designed interface elements, but leverages irregularity to create meaning and importance”, or ”it conforms to users’ mental model of what it does”.

It is important to note that since the same codebase is shared between multiple platforms, it is of high importance to create designs that fit both the iOS and Android design languages and make sure that they do not feel out of place.

After doing so, the design needed to be evaluated against the requirements [62]. Considering that user acceptance tests are going to be performed on a later stage of the application development (Section 5.2), a simple test with users that only evaluated and validated the UI was made.

[1] Sketch – `https://www.sketch.com/`
[2] Zeplin – `https://zeplin.io`



#dark
rgb 28 23 47

Content

Aula lecionada

React Native

```
const aulaLecionada = {
  width: 105,
  height: 17,
  fontFamily: "SourceSansPro",
  fontSize: 17,
  fontWeight: "600",
  fontStyle: "normal",
  lineHeight: 17,
  letterSpacing: -0.2,
  textAlign: "left",
  color: colors.DARK
};
```

Figure 3.5: Zeplin interface with React Native code generation of design elements

In this user test, the designs for each functionality were shown to the participants, using the prototype generated by *Sketch*, which used the design's wireframes. After the UI was presented to the participants, a small survey (Appendix C) about it was performed. This test was made with only seven users, mainly students, which are the main target for the application. In spite of the design being made by an experienced team, this test still proved to be very useful. Most of the users rated the UI with the maximum score available (Figure C.3), and all of them stated that it had surpassed their expectations (Figure C.5). Moreover, every participant provided constructive feedback on how they thought some aspects of the design could be improved, for instance, allowing the user to sort the curriculum courses by academic year or alphabetically, or only showing the shuttle trips that have not started yet. This allowed gathering some feedback that would only be received in the user acceptance tests, which would be performed at the end of the development (Section 5.2), much earlier.

4 Implementation

Contents

After defining the requirements (Section 3.1.3) and validating the design (Section 3.5), the implementation details are described in this chapter.

4.1 Application Implementation

At the beginning of the development period, there were three key factors that were considered concerning React Native development: what language to use, what emulator to use, and how many third-party libraries should be used.

4.1.1 Development Language

Although JavaScript is the programming language used to program in React Native, a superset of it, *TypeScript*, can be used. *TypeScript* is developed by Microsoft and in its essence is just JavaScript, but with types [63]. Lately, it has been widely used for this reason. This way, some of the issues inherited by JavaScript for not using types could be avoided altogether. But due to its slightly different syntax, it could make the application maintenance more difficult, so it was decided that it would be better to use plain JavaScript instead. To compensate for this lack of types throughout the codebase and to ease the future maintenance of the app, detailed documentation was made, identifying the types of the function arguments and component properties, for instance.

4.1.2 Emulation

The emulator provided by Android Studio, which is the one recommended in the React Native documentation, proved to be extremely slow and hard to work with, in the main development environment used, running Ubuntu 20.04 LTS. This, however, was not an issue in the Mac OS development environment, even though the machine was much older than the one running Ubuntu. Thankfully, there are alternatives that provide most of the same features as the Android Studio emulator and have better performance, like *Genymotion*[1], the chosen alternative. *Genymotion*, besides its superior performance, provided a simpler UI to create and edit emulators, and connected to the application's development server like the recommended emulator, due to being recognised as an Android Debug Bridge (ADB) device.

4.1.3 Third-party Libraries

To keep the app as light as possible, the use of third-party libraries was avoided as much as possible. However, in the end, many were still used. This is mainly because React Native does not include some essential UI elements, or features, like a toggle component, in order to have a small footprint. Most

[1] Genymotion – https://www.genymotion.com/

of these features were initially provided by React Native and are currently being developed and maintained by the React Native community. This ultimately, increased the number of external dependencies, expanding the local project size. This was worrying at the beginning since it was not desirable for the app to have a large size. In the end, this proved to not be a problem in terms of size, as discussed in Section 5.1.5. Despite this, it still can be a problem when it comes to maintenance since it is important to keep track of each dependency and keep them always updated, especially to avoid any security concerns.

4.2 API Constraints

During the development process, some issues with the Fenix API [58] were identified. Some of these issues conditioned the development of new features. The following issues were identified:

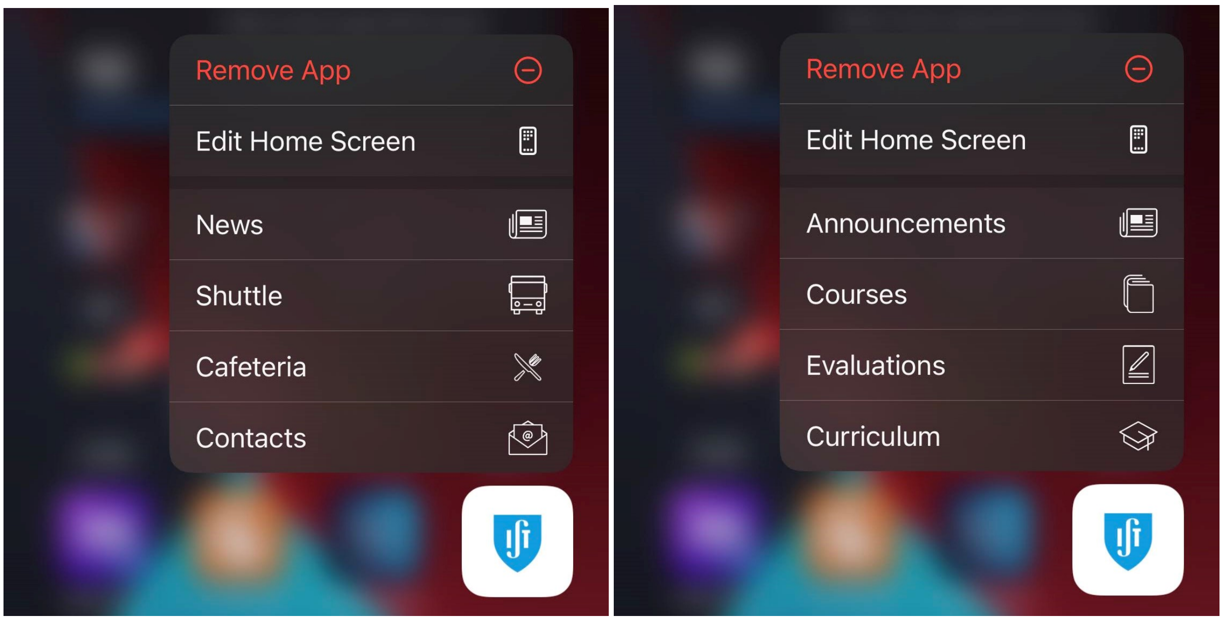
- **/contacts**: endpoint has a lot of missing contacts when compared to IST web page;
- **/person/payments**: a recent update to the Fenix backend changed the way payments are handled and processed. Due to this change, payments' information for the present academic year, 2021/2022, are not returned by this endpoint;
- **/canteen**: no information at all is returned, making it unfeasible to implement the canteen feature without having to use fake information. Besides not returning any information, more information should be returned compared to when it was working, like more dietary information about the menu, ingredients, and meal energetic value. Of course, keeping this information up to date depends also on external entities.

Furthermore, push notification endpoints in the Fenix API [58] were not working and need to be fixed in order for these to work. Initially, a workaround was implemented, where a background task would run each hour on the device and fetch course announcements to compare with the ones stored in cache to understand if there were any new announcements. If there were new announcements, then local notifications were created. The downside of this initial implementation was the fact that iOS could only run background tasks if the application was already opened once and running in the background. Android, on the other hand, could schedule the background tasks when the device booted up, without having to open the application.

Besides the problems in the existing endpoints, there were are also endpoints that had to be created in order to support some of the newly implemented features. For example, the *Search* feature was one of these, which already has all the code needed in the application to search spaces, people, and courses, but which could only search people with the previously available endpoints.

4.3 Implemented Requirements

Despite all the issues identified in the used APIs, all the currently available features in the ongoing application and still required were implemented. Further to these, all the newly established requirements (Section 3.1.3.A) were also implemented.



(a) Quick actions for users when they are not logged in **(b)** Quick actions for users with the *Student* role

Figure 4.1: Quick actions in the new iOS app

Besides the established functional requirements, there were also new functionalities that were added to the application throughout its development. The new features are:

- User account page that shows the user's roles, personal information, namely full name, birthday, and gender, and the user contacts and web pages;
- Quick actions for both iOS and Android that depend on the role of the user, as shown in Figure 4.1. If the user is not logged into an account, the shortcuts available are the ones in Figure 4.1(a), news, shuttle, and contacts. If the user is a student, the quick actions are announcements, courses, evaluations, and curriculum (Figure 4.1(b)). If the user is an alumnus, payments, curriculum, and news. If the user is a teacher, the shortcuts are curriculum, shuttle, news, and summaries. Finally, if the user is a staff member, the quick actions are the same as when a user is logged out. Since most of the users are students, their quick actions were based on the responses to the ninth question of the survey (Figure A.9), which are the same as shown in Figure 4.1(b), except for the

announcements, since the news and announcements were split into two different functionalities to facilitate their visualisation;

- An option for the users to customise the feature that is loaded once the application starts. By the default it is the *news* feature;
- An option to select the closest shuttle stop using the user's location, if permission is given by the user to access location services;
- In order to improve the overall security of the Fenix system, support for two-factor authentication was implemented in the application. This was done through developing notifications, a page displaying the generated code, which also allows to request a new code as well. With this, two-factor authentication is ready to be used with few changes required once it is implemented in the backend;
- User interaction logs were added and are sent automatically, if the user allows it, when a crash occurs, due to the use of *Crashlytics*[2]. This tool automatically gathers information about the device of the user and sends it along side the source of each crash. This makes it easier to analyse and understand what was the source of the crash.

As for the non-functional requirements defined in Section 3.1.3.B, all of them were met. To achieve the *compatibility* requirements, the application is able to run on iOS devices running iOS 12 or later and on Android 6.0 or later, exceeding the minimum requirement established by a couple Android versions. To comply with the *security* requirements, no HTTP connections are allowed. Moreover, as mentioned in Section 3.3, the application stores the *refresh token* provided by the OAuth mechanism in a secure way. In the application this is done through a popular and regular maintained third party library, *react-native-keychain*[3], that provides access to the native *iOS Keychain*[4] and *Android Keystore*[5], which are the recommended and most secure way to store authentication information on both platforms.

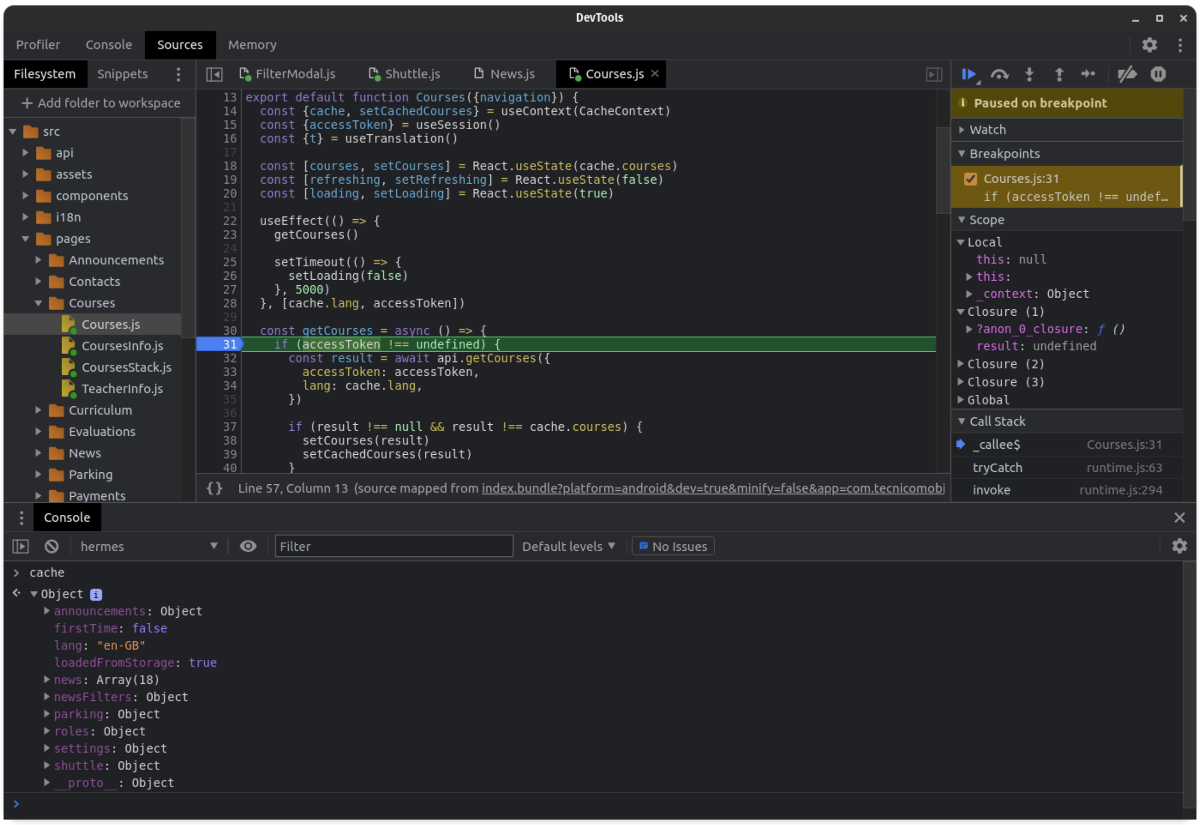
For the remaining non-functional requirements, *Performance* and *Usability*, performance testing and user acceptance tests were performed and their results are going to be analysed in detail in Chapter 5.

4.4 Debugging

Similarly to traditional software development, to fix the issues encountered during development, debugging had to be performed. Fortunately, React Native already provides guides and tools for this purpose.

[2]Crashlytics – https://firebase.google.com/products/crashlytics
[3]React Native Keychain – https://www.npmjs.com/package/react-native-keychain
[4]iOS Keychain – https://developer.apple.com/documentation/security/keychain_services
[5]Android Store – https://developer.android.com/training/articles/keystore

If the default JavaScript engine is being used, the development menu can be used to start the debugging server. This will open *http://localhost:8081/debugger-ui* on Chrome, allowing the debugging of the application using Chrome's developer tools, like a website or a NodeJS application. The aforementioned tools allow browsing the application's files, introducing multiple breakpoints, querying the application using the console, observing the value of the variables used in each breakpoint, analysing errors and warnings, and CPU and memory profiling.



```
export default function Courses({navigation}) {
  const {cache, setCachedCourses} = useContext(CacheContext)
  const {accessToken} = useSession()
  const {t} = useTranslation()

  const [courses, setCourses] = React.useState(cache.courses)
  const [refreshing, setRefreshing] = React.useState(false)
  const [loading, setLoading] = React.useState(true)

  useEffect(() => {
    getCourses()

    setTimeout(() => {
      setLoading(false)
    }, 5000)
  }, [cache.lang, accessToken])

  const getCourses = async () => {
    if (accessToken !== undefined) {
      const result = await api.getCourses({
        accessToken: accessToken,
        lang: cache.lang,
      })

      if (result !== null && result !== cache.courses) {
        setCourses(result)
        setCachedCourses(result)
      }
```

```
{} Line 57, Column 13 (source mapped from index.bundle?platform=android&dev=true&minify=false&app=com.tecnicomobi
```

```
Courses.js:31
  if (accessToken !== undef…

▼ Local
    this: null
  ▶ this:
  ▶ _context: Object
▼ Closure (1)
  ▶ ?anon_0_closure: f ()
    result: undefined
▶ Closure (2)
▶ Closure (3)
▶ Global

_callee$      Courses.js:31
tryCatch      runtime.js:63
invoke        runtime.js:294
```

```
> cache
<· ▼ Object i
    ▶ announcements: Object
      firstTime: false
      lang: "en-GB"
      loadedFromStorage: true
    ▶ news: Array(18)
    ▶ newsFilters: Object
    ▶ parking: Object
    ▶ roles: Object
    ▶ settings: Object
    ▶ shuttle: Object
    ▶ __proto__: Object
>
```

Figure 4.2: Debugging a React Native application running on Hermes, with Chrome developer tools

Another method to debug the application, is through Visual Studio Code (VS Code) [6], the IDE used to develop the application. To do so, firstly the *React Native Tools*[7] extension, provided by Microsoft, needs to be installed. Having done so, depending on the React Native configuration, the *launch.json* file needs to be configured accordingly to the extension's setup guide. After setting up, to start debugging, the play button needs to be pressed on VS Code's debug tab. This extension provides most of the features in Chrome developer tools, with the exception of CPU and memory profiling.

Having tested all of these methods, the easiest one to use, was VS Code, due to its simplicity (only a button needs to be pressed to start debugging after setting up). Also, being able to introduce breakpoints,

[6]Visual Studio Code – `https://code.visualstudio.com/`
[7]Visual Studio extension – `https://marketplace.visualstudio.com/items?itemName=msjsdiag.vscode-react-native`

query the console, and check variables' values inside the development IDE, instead of using another application to do it, makes it simpler and more productive.

4.5 Testing

During the development process, with the main purpose of catching errors as soon as possible, *unit* and *integration testing* were considered. These, however, did not test the interactions of the user with the created components, which is their most important characteristic. To test these interactions, according to React Native's documentation [64], *component testing* is the most adequate method.

Besides automated testing, manual testing of different aspect ratios on different devices, and different accessibility options, such as bigger fonts, were also tested, which proved to be a time-consuming task.

4.5.1 Component Testing

To start component testing, as recommended by the documentation [64], the *React Native Testing Library*[8] should be used. This third-party library is based on *Jest*[9], which is a powerful, yet simple, JavaScript testing framework, compatible not only with React Native, but also with VueJS, AngularJS, Node, and TypeScrypt, among others.

However, soon after, component testing started to reveal some problems. The biggest one was the fact that when a change of state happened in a component, it caused the component to be unmounted during testing, which made it impossible to test anything after that state change, leading to an error stating that the component had been unmounted. This made component testing unfeasible since most components have an internal state and depend on them to work properly. So, another approach had to be used.

4.5.2 End-to-End Testing

The last method described in React Native's documentation [64] that allowed testing the interaction with components, was *end-to-end testing*. In this type of testing, instead of testing an application component, the whole application is used for testing on a real or emulated device. They also emulate well the interaction of a user with the application, since the actual elements of the application are being interacted with.

To perform end-to-end tests, the React Native team recommends two frameworks, *Detox*[10], and *Appium*[11]. The first one was the chosen framework, due to the use of *Jest*, since it was already used

[8] React Native Testing Library – https://callstack.github.io/react-native-testing-library/
[9] Jest – https://jestjs.io/
[10] Detox – https://wix.github.io/Detox/
[11] Appium – http://appium.io/

when trying to perform component testing, and it has easier to read and understand documentation. Setting up *Detox* was a time-consuming, yet simple process, due to its comprehensive documentation. This long setup process was mainly due to the steps required to make end-to-end tests work on Android. For this platform, both changes in the native code and the application's Android properties files had to be made. In iOS, the only setup needed was to configure the *Detox* property file that contains information about the devices that are going to be used for testing. In Android, this configuration also needs to be done, and besides supporting the *Android Studio emulator*, it also supports *Genymotion* (the emulator being used, for the reasons aforementioned in Section 4.6.2).

For the tests to be consistent, the APIs needed to return always the same response or return an empty response to test empty states. To achieve this, mocks for the APIs were developed. Mocks replace components or functions with a different implementation or fixed response for testing purposes. Besides the APIs, the OAuth login flow also needed to be faked, since *Detox* cannot interact with web views, which are required to log in. Since it would not be safe from a security standpoint to have the credentials for multiple accounts (one for each role) registered in plain text in the test files, a simple component to select the role of the user was made. Each role in the mocked component would set the appropriate roles in the app and cache information about the user, which in this case is a made-up user, similarly to the real login.

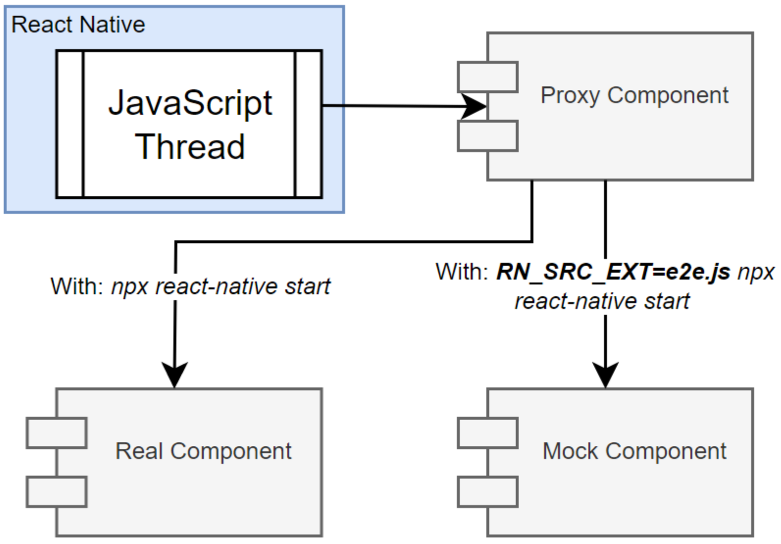

React Native
JavaScript Thread
Proxy Component
With: *npx react-native start*
With: ***RN_SRC_EXT=e2e.js*** *npx react-native start*
Real Component
Mock Component

Figure 4.3: How a real component or a mock component is selected using Detox

Detox's mocks, when compared with the ones developed when component testing, were much simpler. To do so, as illustrated in Figure 4.3, only three steps need to be taken:

- Create a proxy component that only imports the real component and exports it;

- Create a fake component, like a normal component, in the same directory as the proxy with the *.e2e.js* file extension and the same name as the proxy;
- launch the Metro server (the application server) with the *RN_SRC_EXT*[12] variable set. This way when the application is running, the server will use the mock components instead of the real ones.

Listing 4.1: End-to-end test using Detox

```
1  describe('News screen', () => {
2    beforeAll(async () => {
3      await device.launchApp()
4    })
5
6    it('should display news component', async () => {
7      await expect(element(by.id('newsList'))).toBeVisible()
8
9      await expect(element(by.id('newsItem0'))).toBeVisible()
10     await expect(element(by.id('newsItem0-title'))).toHaveText('A title')
11     await expect(element(by.id('newsItem0-cat'))).toHaveText('Events')
12   })
13 })
```

After mocking all the needed components, tests can be easily written using *Jest*, like the one shown in Listing 4.1. The test files need to be located in the *e2e* directory with the *.e2e.js* file extension, similarly to the mock components. In this and other tests, methods from *Jest*, like *describe* (line 1), to group tests, or *beforeAll* (line 2), code to run before all tests, are needed to write them. Besides using the *Jest* library, *Detox* also offers its own library to ease testing, with methods to tap a component, write in a text box like a user would, sliding, pinching, assessing methods (line 7 to 11), among others. Besides these methods, it also offers a *device* object with methods that allow uninstalling, installing, restarting, terminating, and launching the application (line 3), sending a notification, setting the device location, and many others. Also, it is important to note that to find the components with ease, their *testID* can be used like in line 7 of Listing 4.1, which is a property that can be given to all React Native components.

End-to-end testing a functionality proved to be quite useful due to the fact that a considerable amount of errors were found, like missing empty states, toggles not working correctly, or roles having access to features that they should not have, and the small amount of time they took to write. After developing all the functional requirements listed in Section 4.3, a total of 165 tests were developed and, after some

[12] https://wix.github.io/Detox/docs/guide/mocking/#triggering

corrections, all of them run successfully. These tests can also be integrated into a Continuous Integration and Continuous Delivery (CI/CD) pipeline, making sure that any update to the code passes all end-to-end tests.

Although end-to-end tests are the best method to test user interaction with the application, the React Native team identifies some of their disadvantages: they are more cumbersome to write, slower to run, and, sometimes without any changes to the source code, a test can either pass or fail (flakiness) [64]. The last one is due to the application being tested on the release version, so cached data, for instance, can affect the outcome of tests, if they are not accounted for. The fact that they are slower is true since the test is being run in a real or emulated device. End-to-end tests, when compared with component testing, contrary to what was stated by the React Native team, were simpler to write, so more tests could be written in the same amount of time. As for the flakiness, it did not prove to be true as well while developing the tests.

4.6 React Native Analysis

Throughout the application development, both positive and negative points of React Native were found and are described in the following subsections.

4.6.1 Positive Factors

Right from the start of the development, it was clear that the environment setup, although more complex on Linux than on Mac OS, was straightforward due to the detailed and comprehensive documentation provided by the React Native team. This detailed documentation can be seen everywhere in React Native, especially in components' documentation, which lists and explains every *prop* the component can use, exemplifies the use of the component through an example that can be tested and interacted with through the web browser, and has a list of methods provided by the component. Besides component documentation, there are testing, debugging, performance, and design guides, among many others. Comprehensive documentation can also be found in many third-party packages offered by the community.

The big and active community of React Native also eased the development of the application due to two factors. The factor stems from its popularity, if a problem in React Native or in a popular third-party component is found during development, it is highly likely that other developers had already encountered a similar issue and clear solutions are already available on internet forums. The second factor is the fact that if something is needed that is not currently provided by React Native, it is almost certainly provided by a third-party library developed by the community. For example, the *in-app web browser* used in the

IST app, the calendar synchronisation, HTML renderer, among others, are community components or libraries.

Another way to avoid some of the issues already identified in React Native, especially the ones that come with the use of the *Bridge*, as mentioned in Section 2.3.1.B, is to use another JavaScript engine. For the development of the application, *Hermes* [65], an engine also developed and made open source by Facebook, was used. This engine was mainly chosen due to the use of JSI, which brings many advantages, such as the ones described in Section 2.3.1.C, its smaller application size and memory usage, and faster application startup.

Not only can the application run on emulators, but it can also run on real devices and be connected to the application server (the Metro server), allowing testing the application on real devices at the same time the application is being developed. When the application is running on a real device, it offers the same debugging features available in emulators, like an element inspector, and live performance metrics, among others.

Lastly but not least, the use of JavaScript, together with hot-reloading, contributes to simplify the use of React Native. As previously described, the use of hot-reload allows changes in the application to be applied live both in the used emulators and the real devices, eliminating the time needed to compile the application, which can take time in slower machines.

4.6.2 Negative Factors and Issues

Despite all the positive aspects described in the previous section, there were some issues found while developing the application that are relevant to all React Native applications. As mentioned previously, almost anything that is missing in React Native can be provided by a third-party package. However, even basic features that are used in all applications, like a splash screen, are missing in React Native. This means that the developer has to spend time selecting all the available custom packages and deciding which is the best to use, eventually leading to multiple packages being tested, raising maintenance concerns.

In the case of the aforementioned custom splash screen, it leads to another downside of React Native: although JavaScript is the language used to program in React Native, eventually more programming languages are used to make a React Native application. In this case, as well as others, like setting up notifications, changing the application icon, allowing HTTP requests to test internally, and changing the application name, among others, imply changes to the native code and property files for both platforms. Ultimately, it may be less of a problem than it seems since most third-party packages have detailed explanations and step-by-step setup instructions for these cases. However, it is important to note that changes in native code will most likely happen. A bigger problem arises when a feature is not currently provided by React Native nor the community. In these cases, the only way to implement it is through the creation of a native module.

Lastly, a major inconvenience found while developing the application was the fact that, on several occasions, React Native did not provide a useful stack trace to identify the root of a problem, which lead to extra time spent on finding the source of the error.

5

Solution Evaluation

Contents

To evaluate the new solution, the current IST application was used as a baseline, in order to compare and evaluate the non-functional requirements, such as performance and usability, against the new React Native application.

5.1 Performance Testing

Since React Native applications are interpreted apps, it is expected that the use of resources, namely CPU, RAM, and power consumption, will be higher when compared with its native counterparts. To check this, and to evaluate how many resources the new application requires, the following subsections describe the methodology and results obtained from testing both the performance of the original application and the new one, for each supported platform.

On both platforms, two tests were performed. The first one launched both applications for the first time, logged in, and went through all the common features between the two. The second test evaluated the performance of the applications in standby (both with it opened and running in the background).

In Android, to monitor in real-time the device's metrics while conducting the aforementioned tests, Android Studio provides an application profiler that allows to observe the CPU consumption, RAM usage, and power usage, as shown in Figure 5.1. The profiler also allows to connect to any device, so a real device, or an emulator, can be used to test the application. To conduct the tests in Android, an emulator running Android 10 was used.

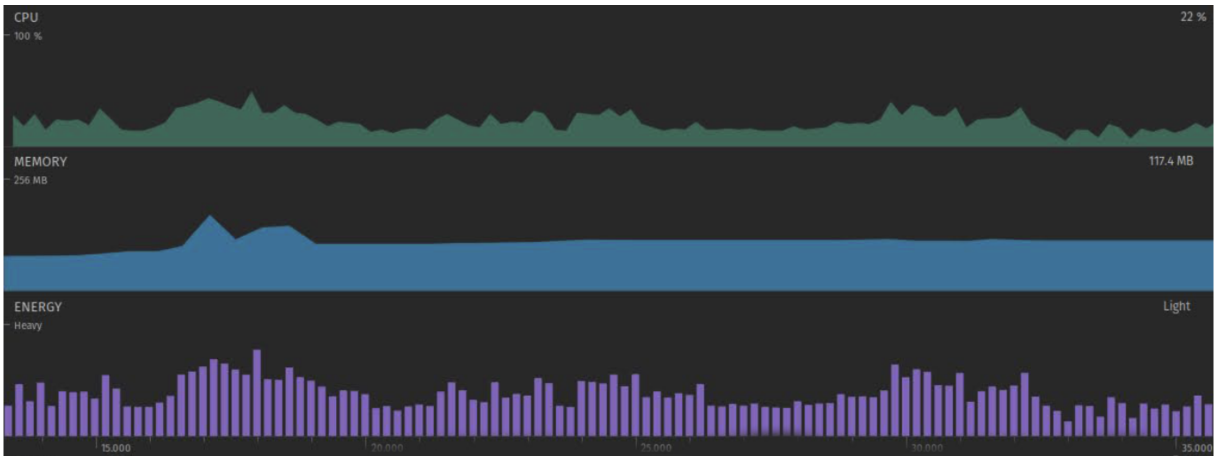


Figure 5.1: Device's metric shown in Android Studio while performing the first test case

In iOS, Xcode can be used to monitor the device's metrics as well (Figure 5.2). Despite Xcode offering a more complete profiling tool (*Instruments*) with more metrics and customizability than Android Studio, it proved to be quite challenging to use. Although both emulators and real devices can be profiled, some metrics, including some of those captured during the aforementioned tests, can only be monitored using real devices and not emulators. For this reason, instead of an emulator, like in Android, the tests were performed on a real device. The test device was an iPhone Xs running iOS 15.0, which represents a good middle ground between newer and older iOS devices.

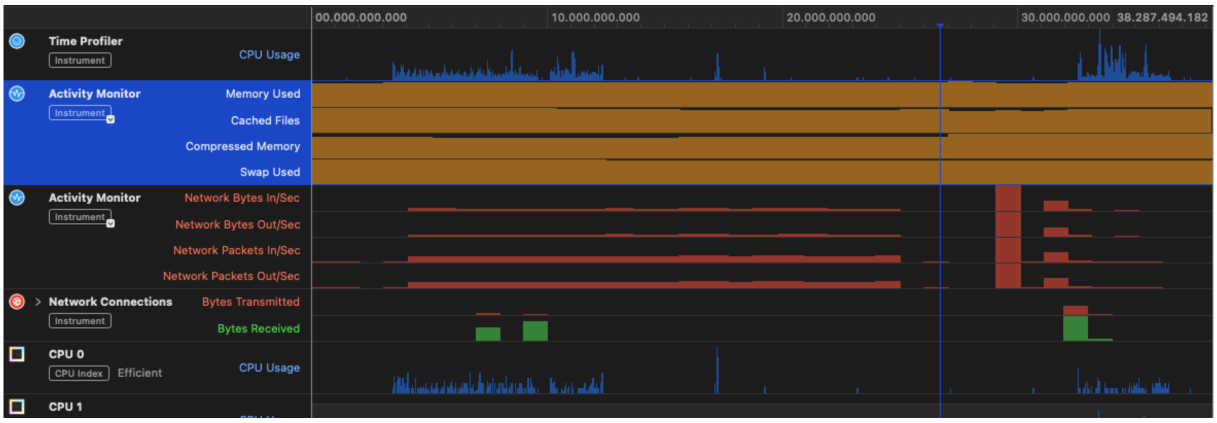


Figure 5.2: Device's metric shown in Xcode while performing the first test case

5.1.1 CPU

During the first test on both platforms, the CPU performance tests yielded similar results to the ones described in Section 2.3.1.G. For the first test case, the user interaction test, the React Native application had at startup approximately two times the CPU consumption of the original native Android application, namely 55% and 27% respectively (Figure 5.3), and 70% for React Native and 50% for native iOS (Figure 5.4). These results are consistent with those obtained by Hansson and Vidhall [19]. This behaviour was expected for the same reasons mentioned in Section 2.3.1.G.

For the remaining of the test, the CPU usage proved to be higher in React Native than its native counterpart, having an average in Android of approximately 14% for React Native and 10% for native Android, and in iOS of 49% in React Native and 41% in the native application. In Android, despite the average being smaller than expected, it still represented a 40% increase over native. In iOS on the other hand, although the average CPU usage was higher, the increase was significantly smaller, being about 19.5%. The spikes in each graph coincided with the loading of a new feature.

This higher CPU usage can be explained not only by the use of React Native, but also result from a more complex UI. This is mainly due to the elaborate animations used, which were almost nonexistent in the original application. Moreover, some features, such as the *shuttle*, were not working on the current application, so they were only displaying an empty state.

Lastly, for the second test case, in Android, both applications had similar CPU consumption of 12% while idling and 0% while running in the background. iOS, on the other hand, had a 5% CPU consumption in the native app while idling and 0% in the React Native app. As for background utilisation, both applications used 0%.

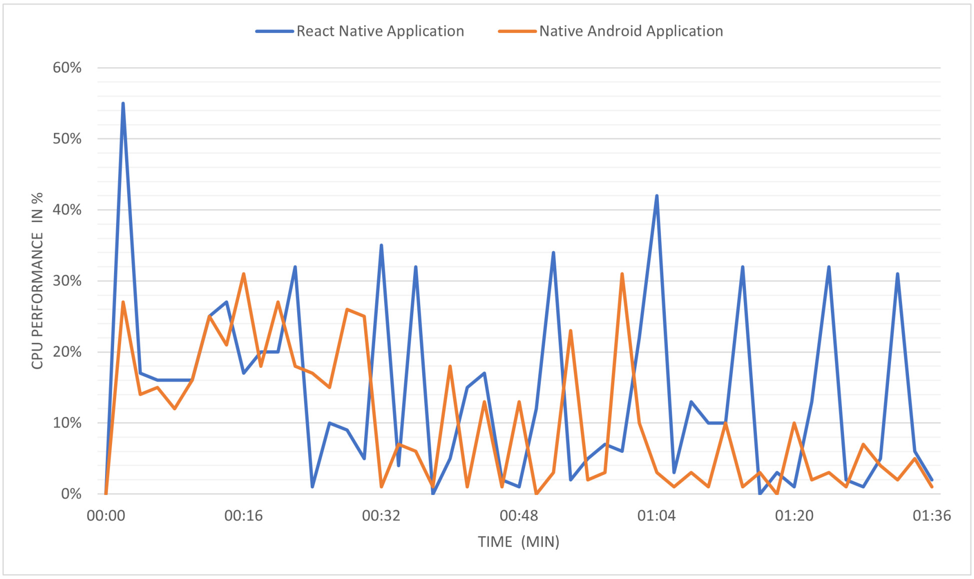


Figure 5.3: CPU usage on an Android emulator for the first test case

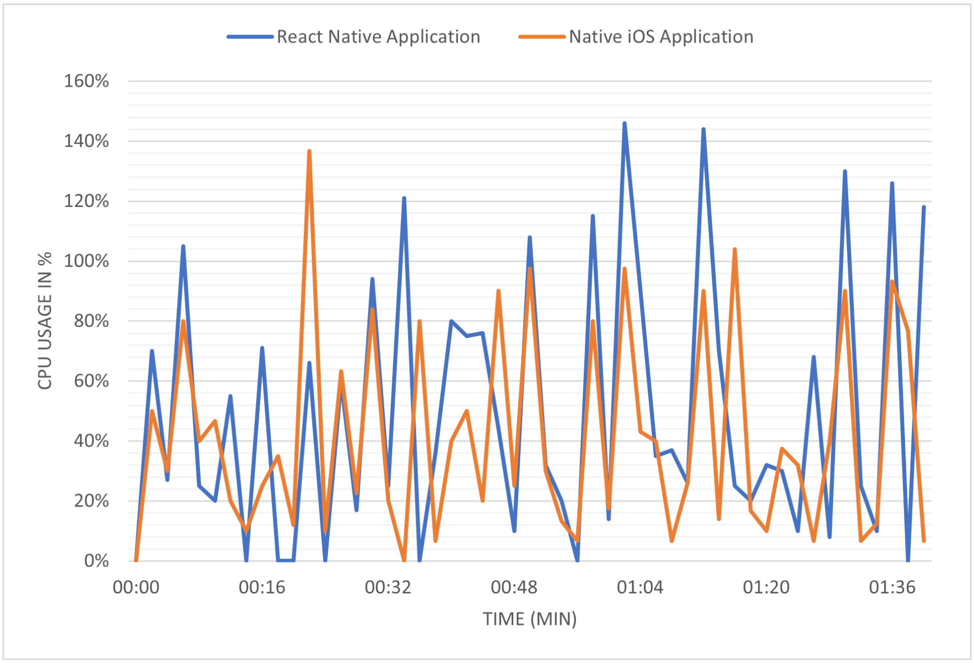


Figure 5.4: CPU usage on an iOS device for the first test case

5.1.2 RAM

Regarding memory usage, the new application required on average 30MB more in Android (Figure 5.5) than the original one, and 17MB more in iOS (Figure 5.6). This represented a 29.4% increase in Android and 35.8% in iOS. This can be explained by the same reasons mentioned for the increase in CPU usage.

In a similar fashion to the CPU usage, the RAM usage can also decrease in a day to day use, since it is expected that each user will only use regularly a handful of features. This is shown in the survey answers (Figure A.9). This coupled with the fact that *react navigation*[1], the library used to navigate through the screens, only loads a screen after the first time it is needed, justifies that the amount of used RAM will be lower on a daily basis.

In the second test case, the RAM usage on Android was more than 100% higher on the React Native app, 118MB for the new application while idling and 103.6MB while in the background, and 49.8MB for the current one in both cases. The situation in iOS was similar, with an idle consumption of 53.4MB and 48.6MB while in the background, compared with 20MB and 10MB respectively of the native iOS application.

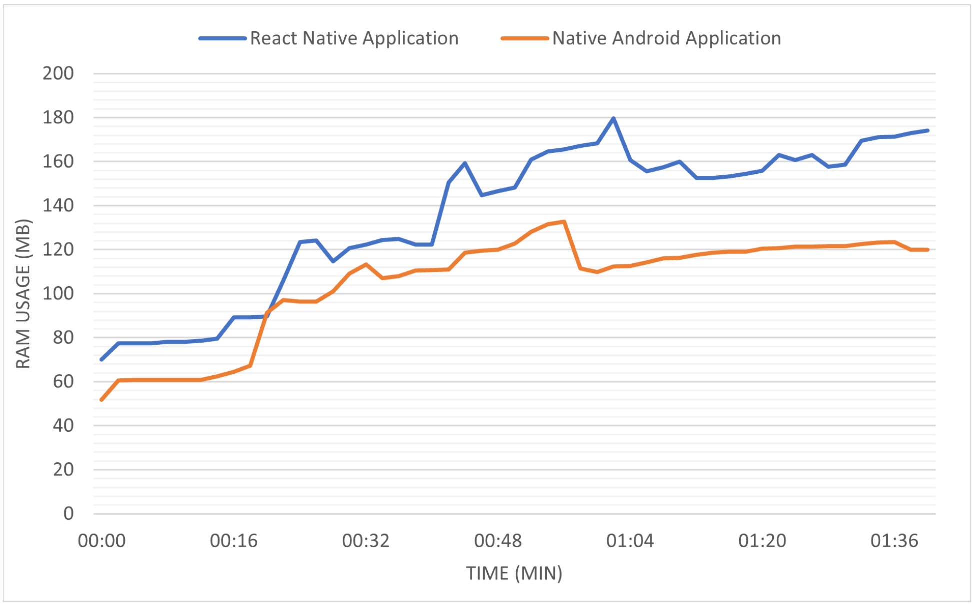


Figure 5.5: RAM usage on an Android emulator for the first test case

[1] React Navigation – `https://reactnavigation.org/`

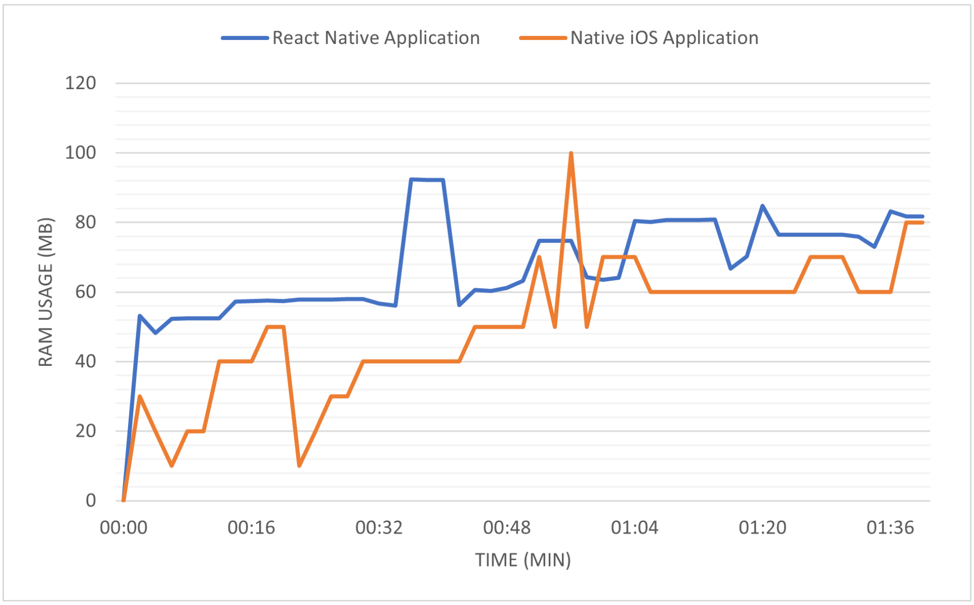


Figure 5.6: RAM usage on an iOS device for the first test case

5.1.3 Network Requests

To monitor the number of network requests and their latency in Android, Android Studio provides a network profiler, but unfortunately, it was not working in the current version[2]. Therefore, third-party alternatives had to be used. For React Native, *reactotron*[3] proved to be extremely useful, not only showing network requests but *AsyncStorage* calls as well. Unfortunately, it only worked for the React Native App. A more global solution was to use *Charles*[4], an HTTP and Secure Sockets Layer (SSL) proxy tool, that allows to perform a *man in the middle* attack and observe every request sent from the device or emulator. In iOS, since it was not possible to build the native app on Xcode, due to the great amount of produced errors, *Charles* needed to be used as well, since Xcode does not show detailed network information of external apps.

After analysing the results, the average network call duration in React Native was 95.28ms for Android and 138.43ms for iOS, which are similar to their respective native counterparts, as it was expected. Besides their duration, it was interesting to compare the number of requests the new application performed compared with the original one, which was much less in the React Native application than the current native one: 25 and 47 requests respectively. This difference can be explained by the fact that the original application performs all the requests once the application loads, which decreases the load time of each feature, but it also performs API requests in each feature. With this approach, most requests are

[2]Android Studio Bumblebee – 2021.1.1 Patch 2
[3]Reactotron – https://infinite.red/reactotron
[4]Charles – https://www.charlesproxy.com/

being performed twice. This shows that the new application is more efficient than the current one. Note that in iOS, the React Native application each couple of minutes performs a network request to verify its internet connection, which does not happen in Android, but these were filtered out during the network testing.

5.1.4 Energy Comsumption

In both tests running on Android, despite the React Native application consuming more energy, among the three Android Studio's energy categories, "light", "medium", and "heavy", both applications still average a "light" consumption. In iOS, the results were similar to the ones obtained in Android.

5.1.5 Storage Size

Besides testing both applications, their storage size after the tests was also analysed in order to better simulate the state of the application after some use. In Android, the new application occupied 54.05MB, while the original one was only 28.62MB. In iOS, the new application had a size of 23.7MB while the original one had 21.1MB. The increase in the Android app size was expected since a React Native application needs to contain the React Native framework. In iOS, on the other hand, the size was almost the same, which was anticipated, but not to this extent. This smaller size in iOS is because applications generated for Apple devices are more optimised for each individual device, whilst on Android, apps are only optimised for a given architecture, since it needs to work for a greater number of devices. In any case, storage on mobile devices increased drastically compared to the devices available at the time the current app was developed, so this increase does not pose any problem.

5.1.6 Overall Discussion

Despite the overall higher resource consumption in the React Native app, the increase in most cases is minor and justifiable, considering that the app is more computationally intensive, due to its modern animations and the number of features being greater. React Native itself contributes to this as well. In general, this increase in resource usage was similar to the results obtained in [19] and [26], discussed in Section 2.3.1.G. This gap would probably be even smaller, if the current native applications were similar to the newly developed one, similarly to the aforementioned studies.

5.2 User Acceptance Tests

According to [66] there are three types of user acceptance tests:

- **behaviour** or **scenario** based acceptance tests: evaluates the system based on the perspective of the users, by analysing the external behaviour of the system;
- **black-box** acceptance tests: tests the functional requirements of the product;
- **operation-based** acceptance tests: testing is based on the probability of occurrence and each test case is different depending on the profile of the user.

To choose the best approach, the identified drawbacks of each one of them were considered. The first one has limited user involvement and it does not specify the acceptance criteria from the user's point of view. Since the users should be involved the most in this process to validate the product, this approach is not the best one. The second one, the *black-box* approach is the most widely used one and does not have any major downside. The last one is a new framework presented in the aforementioned article and seems to be promising, but it requires more previous analysis and it was not tested in a real application with users, so it could have unknown disadvantages. Having this in mind, the *black-box* approach was the chosen one.

5.2.1 User Test Cases

The next step was defining the test cases to be performed by the users. Hetzel and Hetzel [67] state that the test cases should be based on major functional requirements, and in [66] that they should be decomposed into single condition cases. This being said, the following test cases for each of the major functionalities available to all user roles were considered:

- **Setup screen**: Open the app and go through the initial setup;
- **News**: Read the first article of the "Campus and Community" category;
- **Cafeteria**: Check the ingredients of the soup being served for dinner on Tuesday for the Alameda campus;
- **Shuttle** Check the bus stops for the first shuttle trip tomorrow from Taguspark to Alameda;
- **Search**: Find the location of room FA1, including its location on the building floor;
- **Parking**: Check how many hours a day the Alameda car park is open;
- **Contacts**: Find the phone contact of DSI;
- **Tickets**: Get a ticket for the "General Service" queue in the academic service;
- **Payments**: Share the payment information of the first outstanding payment;

- **Account**: Discover the roles of the logged user;
- **Calendar sync**: Disable the evaluation calendar sync;
- **Custom initial feature**: Change the start-up screen to the *Shuttle* feature;
- **Email**: Check the email setup guide.

For participants with the *student* role, the following additional test cases were performed as well:

- **Announcements**: Read the first announcement of the ”Master Dissertation” course;
- **Courses**: Find out the teachers of the ”Communication Skills” course;
- **Schedule**: Figure out the student schedule for tomorrow;
- **Evaluations**: Discover the rooms for the next incoming student evaluation;
- **Curriculum**: Discover the student grade of the first course alphabetically of the Master's degree;
- **Study**: Find the most empty library in the ”Pavilhão Central” building.

For the users with the *teacher* role, only the *schedule* test case, together with a new test case for the *Summaries* feature is needed: edit the first filled summary, adding a Portuguese title translation. For the *Alumni*, the *Curriculum* test case described above was used. Lastly, participants with the *staff* role only needed to do the first mentioned test cases.

5.2.2 Results

After defining the test cases, and using the acceptance criteria presented in [66], ”No major problems found”, the user acceptance tests were performed. In these tests, the users interacted with the application running on their own devices, allowing to gather feedback and observe the application running on a multitude of different devices, more importantly, different screen aspect ratios. It is also important to mention that the application used for testing was a slightly modified version of the developed application that used mocks for logging in and to gather most of the information, allowing to have more consistent tests among each participant.

Nielsen [68] argues that five users are enough to perform user tests and provide the best return on investment. Also, according to [69], the participants should either be a user of the product, or a participant with similar needs and background to the ones that do use the product. For these reasons, ten users participated in the user acceptance tests, since there were no costs besides the time involved with testing. Of the ten, three were students, but not at IST, allowing to simulate the experience of new students and foreign exchange students enrolling at the university and interacting with the application for the first time. Three were students at IST, one alumnus, and three were teachers. Also, one of the students and the alumnus were also teachers before, so they also performed the test cases assigned to teachers. Users with the *staff* and *investigator* roles were not tested, since there were no exclusive features for those users. Before starting testing, the participants needed to sign a Non-Disclosure Agreement (NDA) to guarantee that the current version of the application is not leaked. Throughout the duration of the tests, the users were encouraged to give feedback on the test case being performed, as described in [69]. After completing the test cases, the users answered a survey (Appendix D), where they could leave their feedback anonymously about the application as a whole.

Afterwards, the feedback provided from the final survey was compiled and analysed, and it is available in Appendix D. Overall, the participants considered the new application very good, despite some suggestions that could improve the UX of the application. This can especially be observed in the responses got from the participants when asked to rate the overall user experience of the application, where every participant answered with the maximum score (Figure 5.7). This result was a substantial improvement over the original application (Figure 5.8), and allowed to fulfil the usability requirements established in Section 3.1.3.B.

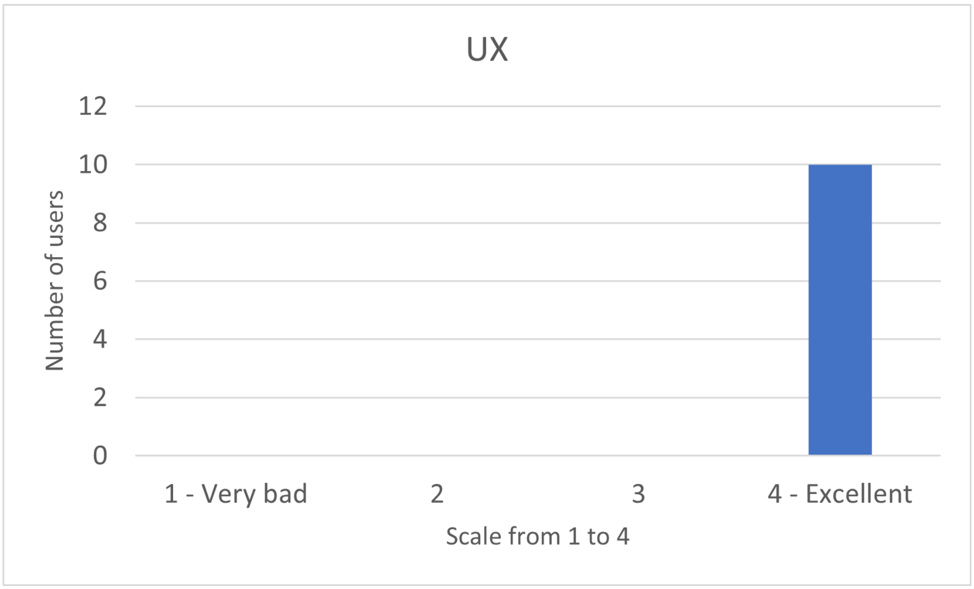


Figure 5.7: UX rated by users during user acceptance tests

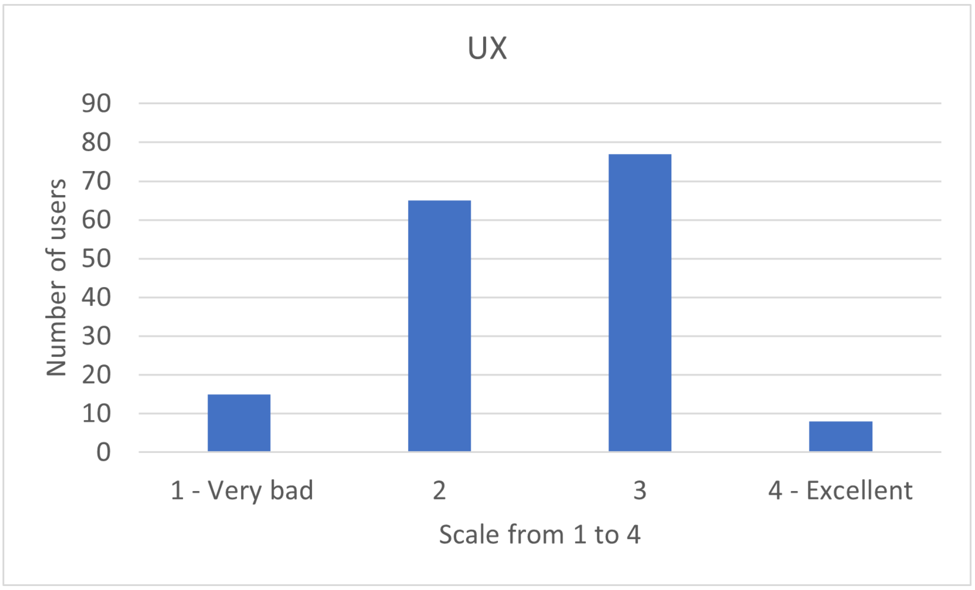


Figure 5.8: UX rated by users during the survey related to the current app

As for the other metrics, UI (Figure D.3) and intuitiveness (Figure D.2), both of them achieved the same results overall as the UX metric. Intuitiveness, however, had one participant who did not rate it with the maximum score. This was due to a design decision that most users did not agree with: the *search* feature and the *study spaces* one are separate features, where they could be just one since the *search* functionality can search for spaces in the university's campus, it will also find the study spaces. Also, it was not clear what could be searched, and the location of the search menu item in the menu itself was not the best one. Despite this, another indicator of how intuitive the application was is the time taken on average by each participant to accomplish a task or a test case. On average, more than 84% of the test cases were completed in less than 15 seconds, as shown in Figure 5.9. Unsurprisingly, the tasks that had a bigger time disparity were the ones related to the *search* and *study* features, which, as mentioned before, were the ones that the users thought could be improved the most.

Besides the aforementioned issue, the users identified small improvements that could be done to further ease the use of the application, such as: moving the help section from the settings to the menu drawer for easy access, a more conventional date picker to consult the *shuttle* schedule, adding the day of the month bellow the day of the week on the user's schedule and the canteen menu to easily identify which week it is, and move the study spaces outside the student menu so everyone could use it. Apart from improvements, additional features were also suggested, like: adding more information to the courses' pages (the most recent announcement and a link to consult all of them, summaries, among others), in a similar fashion to students, teachers could have their own menu with more features (such as information on the courses they are teaching, their announcements), and for the students that are also teachers, have a separate schedule for their student and teacher roles. The last ones are of

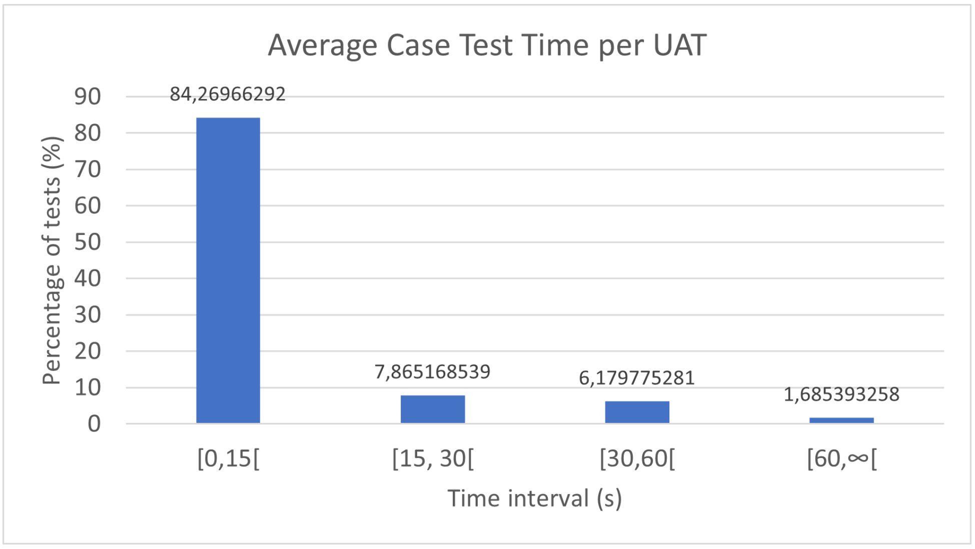


Figure 5.9: Time took on average for each test case in user acceptance tests

high importance since only a few teachers had answered the survey regarding the current application (Section 3.1.2), so there were no specific features for teachers mentioned in the answers that users would like to have. This way, it was possible to discuss directly with the participants their needs and analyse if they would be a good addition to the application.

Despite the overall feedback being exceptionally positive and the design being already validated by users, as described in Section 3.5, the user acceptance tests still proved to be useful. These tests allowed the verification of the design once more, since it suffered some alterations during the development of the application. It also verified the application itself, if it was responsive and if no major bugs were found, which was the established acceptance criteria. Most important of all, the feedback gathered will make it possible to further improve the product before its release, since no major problem was found during testing.

6 Conclusion

Contents

The goal of this work was mainly to find a viable alternative to native mobile development using the IST app as a use case. After understanding the different types of applications, and the different types of cross-platform approaches, the three main ones were described and compared, namely, React Native, Flutter, and Progressive Web Apps. To choose between them, however, the requirements of the application needed to be gathered, to find the best fit for the new application. The environment where the application was going to be developed and maintained also took a major role in the decision. The final decision was to use React Native, which was identified as the best overall cross-platform solution, mainly due to the use of native UI elements, hardware and software feature support, and the use of JavaScript.

After developing the application two metrics were used to verify if the goal was accomplished: the time taken to develop the application from start to finish, and the results obtained from testing the non-functional requirements. As for the first one, the whole application was developed in only five months by only one developer with the help of one designer, working on average only four to five hours a day. This could even be reduced to almost half the time if a full-time job schedule were to be used (around eight hours a day), or if more developers worked in the project. This demonstrates how simple it is to develop a mobile app using a cross-platform framework, especially using one with a known programming language. As for the second metric, the results from the performance testing show that the application uses slightly more resources than the native ones. This is a trade-off worth to be taken, due to the feedback obtained from the user acceptance tests, which showed better results, especially in the UX of the application, than the ones obtained from the survey of the current application.

Also, to help achieve these better results, it was crucial to involve the target audience as much as possible during the development of the application. This was done through a survey that allowed the users to express what they thought should be improved and also what new features they would like to have in the application. Besides the survey, testing the application design with users before starting development, helped improve even further the design, and helped to achieve better results in the user acceptance tests.

6.1 Future Work

In the future, the feedback received from the user tests should be applied, resulting in the modification of the UI and the addition of new features. Also, a new round of user testing should be done after implementing all the necessary changes and before publishing the application, to validate the application once more with users. To improve the application itself and its development environment even further, there are a couple of aspects that could be further performed, such as:

- The use of TypeScript instead of JavaScript, to enforce the use of types, preventing many type-related errors in JavaScript, since it does not have a sound type system;

- Integrate the end-to-end tests in a CI/CD pipeline, to assure that any new alteration to the application code does not break any previous feature;
- Customizable quick action, allowing the user to decide which shortcuts the application should have according to their preferences;
- Customizable notifications, giving the user the ability to choose, for instance, which courses send notifications, and to turn on or off payments' notifications. For this, changes in the backend and in the API would also need to be done.

Bibliography

[1] "React Native will be re-architecture in 2020," Jun. 2020, (visited on 14/12/2021). [Online]. Available: https://itzone.com.vn/en/article/react-native-will-be-re-architecture-in-2020/

[2] Flutter, "Flutter architectural overview," 2022, (visited on 24/10/2021). [Online]. Available: https://flutter.dev/docs/resources/architectural-overview

[3] Auth0, "Authorization Code Flow with Proof Key for Code Exchange (PKCE)," 2022, (visited on 02/03/2022). [Online]. Available: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-proof-key-for-code-exchange-pkce

[4] "Smartphone users 2026," (visited on 16/10/2021). [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[5] "Desktop vs Mobile vs Tablet Market Share Worldwide," (visited on 16/10/2021). [Online]. Available: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide

[6] "Annual number of mobile app downloads worldwide 2020," (visited on 03/11/2021). [Online]. Available: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/

[7] "Mobile Operating System Market Share Worldwide," (visited on 16/10/2021). [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[8] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proceedings of the 6th Balkan Conference in Informatics*, ser. BCI '13. New York, NY, USA: Association for Computing Machinery, Sep. 2013, pp. 213–220.

[9] "React Native · Learn once, write anywhere," (visited on 17/10/2021). [Online]. Available: https://reactnative.dev/

[10] "Flutter - Beautiful native apps in record time," (visited on 17/10/2021). [Online]. Available: https://flutter.dev/

[11] "Xcode," (visited on 16/10/2021). [Online]. Available: https://developer.apple.com/xcode/

[12] "Swift - Apple Developer," (visited on 16/10/2021). [Online]. Available: https://developer.apple.com/swift/

[13] "Kotlin and Android," (visited on 16/10/2021). [Online]. Available: https://developer.android.com/kotlin

[14] K. Shah, H. Sinha, and P. Mishra, "Analysis of Cross-Platform Mobile App Development Tools," in *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, Mar. 2019, pp. 1–7.

[15] S. Helal, J. Hammer, J. Zhang, and A. Khushraj, "A three-tier architecture for ubiquitous data access," in *Proceedings ACS/IEEE International Conference on Computer Systems and Applications*, Jun. 2001, pp. 177–180.

[16] A. Hrivas and A. Pardeshi, "IMPLEMENTATION OF CROSS-PLATFORM MOBILE APPLICATION USING PHONEGAP FRAMEWORK," Nov. 2019. [Online]. Available: https://www.researchgate.net/publication/337623551_IMPLEMENTATION_OF_CROSS-PLATFORM_MOBILE_APPLICATION_USING_PHONEGAP_FRAMEWORK

[17] N. Anggraini, R. Fajriansyah, N. Hakiem, I. Munawar, T. Rosyadi, and L. K. Wardhani, "Development of mobile academic information system (AIS) UIN Syarif Hidayatullah Jakarta based on Android with performance evaluation based on ISO/ IEC 25010," in *Proceedings of the 18th International Conference on Advances in Mobile Computing & Multimedia*, ser. MoMM '20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 215–218.

[18] E. Hjort, "Evaluation of React Native and Flutter for cross-platform mobile application development," Master's thesis, Åbo Akademi University, 2020, accepted: 2020-12-01T07:34:58Z Publisher: Åbo Akademi. [Online]. Available: https://www.doria.fi/handle/10024/180002

[19] N. Hansson and T. Vidhall, "Effects on performance and usability for cross-platform application development using React Native," Master's thesis, Linköping University, 2016. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-130022

[20] R. Nunkesser, "Beyond web/native/hybrid: a new taxonomy for mobile app development," in *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 214–218. [Online]. Available: https://doi.org/10.1145/3197231.3197260

[21] "Cross-platform mobile frameworks used by global developers 2021," (visited on 17/10/2021). [Online]. Available: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[22] B. Eisenman, *Learning React Native: Building Native Mobile Apps with JavaScript*. "O'Reilly Media, Inc.", Dec. 2015, google-Books-ID: 274fCwAAQBAJ.

[23] "React Native: A year in review," Apr. 2016, (visited on 03/11/2021). [Online]. Available: https://engineering.fb.com/2016/04/13/android/react-native-a-year-in-review/

[24] "Who's using React Native? · React Native," (visited on 14/12/2021). [Online]. Available: https://reactnative.dev/showcase

[25] W. Wu, "React Native vs Flutter, Cross-platforms mobile application frameworks," Bachelor's Thesis, Metropolia University of Applied Sciences, 2018, accepted: 2018-05-15T10:36:08Z Publisher: Metropolia Ammattikorkeakoulu. [Online]. Available: http://www.theseus.fi/handle/10024/146232

[26] W. Danielsson, "React Native application development : A comparison between native Android and React Native," Master's thesis, Linköping University, 2016. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-131645

[27] "State of React Native 2018 · React Native," Jun. 2018, (visited on 14/12/2021). [Online]. Available: https://reactnative.dev/blog/2018/06/14/state-of-react-native-2018

[28] "React Native in H2 2021 · React Native," Aug. 2021, (visited on 14/12/2021). [Online]. Available: https://reactnative.dev/blog/2021/08/19/h2-2021

[29] "Introducing JSX – React," (visited on 21/10/2021). [Online]. Available: https://reactjs.org/docs/introducing-jsx.html

[30] J. Warén, "Cross-platform mobile software development with React Native," 2016.

[31] A. E. Fentaw, "Cross platform mobile application development : a comparison study of React Native Vs Flutter," Master's thesis, University of Jyväskylä Faculty of Information Technology, 2020. [Online]. Available: https://jyx.jyu.fi/handle/123456789/70969

[32] "Showcase - Flutter apps in production," (visited on 14/12/2021). [Online]. Available: https://flutter.dev/showcase/

[33] "Dart overview," (visited on 25/10/2021). [Online]. Available: https://dart.dev/overview.html

[34] Flutter, "Introduction to widgets," 2022, (visited on 24/10/2021). [Online]. Available: https://flutter.dev/docs/development/ui/widgets-intro

[35] T. Tran, "Flutter Native Performance and Expressive UI/UX," Bachelor's Thesis, Metropolia University of Applied Sciences, 2020, accepted: 2020-05-07T05:35:35Z. [Online]. Available: http://www.theseus.fi/handle/10024/336980

[36] A. Biørn-Hansen, T. A. Majchrzak, and T.-M. Grønli, “Progressive Web Apps: The Possible Web-native Unifier for Mobile Development,” Jan. 2017, pp. 344–351.

[37] D. Fortunato and J. Bernardino, “Progressive web apps: An alternative to the native mobile Apps,” in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, Jun. 2018, pp. 1–6.

[38] K. Behl and G. Raj, “Architectural Pattern of Progressive Web and Background Synchronization,” in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, Jun. 2018, pp. 366–371.

[39] “The App Shell Model | Web Fundamentals,” (visited on 19/10/2021). [Online]. Available: https://developers.google.com/web/fundamentals/architecture/app-shell

[40] V. N. Inukollu, D. D. Keshamoni, T. Kang, and M. Inukollu, “Factors Influencing Quality of Mobile Apps:Role of Mobile App Development Life Cycle,” *arXiv:1410.4537 [cs]*, Oct. 2014, arXiv: 1410.4537. [Online]. Available: http://arxiv.org/abs/1410.4537

[41] N. B. Ruparelia, “Software development lifecycle models,” *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, pp. 8–13, May 2010. [Online]. Available: https://doi.org/10.1145/1764810.1764814

[42] H. K. Flora, H. K. Flora, X. Wang, and S. V. Chande, “An Investigation into Mobile Application Development Processes: Challenges and Best Practices,” *International Journal of Modern Education and Computer Science*, vol. 6, no. 6, pp. 1–9. [Online]. Available: https://www.academia.edu/8106746/An_Investigation_into_Mobile_Application_Development_Processes_Challenges_and_Best_Practices

[43] K. Schwaber, “SCRUM Development Process,” in *Business Object Design and Implementation*, J. Sutherland, C. Casanave, J. Miller, P. Patel, and G. Hollowell, Eds. London: Springer, 1997, pp. 117–134.

[44] G. D. Everett and R. McLeod, *Software Testing: Testing Across the Entire Software Development Life Cycle.* Wiley-IEEE Press, 2007. [Online]. Available: https://ieeexplore.ieee.org/servlet/opac?bknumber=5201507

[45] M. E. Joorabchi, A. Mesbah, and P. Kruchten, “Real Challenges in Mobile App Development,” in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Oct. 2013, pp. 15–24, iSSN: 1949-3789.

[46] J. Gao, L. Li, T. Bissyandé, and J. Klein, “On the Evolution of Mobile App Complexity,” Nov. 2019, pp. 200–209.

[47] T. Lisboa, “Técnico Mobile App,” 2022, (visited on 27/11/2021). [Online]. Available: https://tecnico.ulisboa.pt/en/campus-life/services/tecnico-mobile-app/

[48] R. S. Barata, “Mobility and Location in Academic Information Systems,” Master’s thesis, Instituto Superior Técnico, Lisbon, Portugal, Oct. 2013. [Online]. Available: https://fenix.tecnico.ulisboa.pt/cursos/meic-a/dissertacao/2353642465450

[49] R. Oliveira and J. M. Mendonça, “STAYAWAY COVID. Contact Tracing for COVID-19,” *INESC TEC Science&Society*, vol. 1, no. 1, pp. 58–61, Dec. 2020. [Online]. Available: https://science-society.inesctec.pt/index.php/inesctecesociedade/article/view/33

[50] “Tecnico Go!” (visited on 18/11/2021). [Online]. Available: http://istgo.tecnico.ulisboa.pt/#/

[51] M. Chacón-Rivas and C. Garita, “Mobile Course: Development of a mobile app to access university courses information,” in *2013 XXXIX Latin American Computing Conference (CLEI)*, Oct. 2013, pp. 1–6.

[52] A. Gündogdu, “Designing a Better App for Universities,” Bachelor’s Thesis, Linnaeus University, 2017. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-60512

[53] C. Courage and K. Baxter, *Understanding Your Users: A Practical Guide to User Requirements Methods, Tools, and Techniques*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[54] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Springer Science & Business Media, Dec. 2012.

[55] J. Nielsen, *Usability Engineering*. Morgan Kaufmann, Oct. 1994, google-Books-ID: 95As2OF67f0C.

[56] S. GlobalStats, “iOS Version Market Share Portugal,” 2022, (visited on 24/02/2022). [Online]. Available: https://gs.statcounter.com/ios-version-market-share/all/portugal

[57] ——, “Mobile & Tablet Android Version Market Share Portugal,” 2022, (visited on 24/02/2022). [Online]. Available: https://gs.statcounter.com/android-version-market-share/mobile-tablet/portugal

[58] FenixEdu, “FenixEdu™ API Endpoints,” 2022, (visited on 27/11/2021). [Online]. Available: https://fenixedu.org/dev/api/

[59] R. Boyd, *Getting Started with OAuth 2.0*. ”O’Reilly Media, Inc.”, Feb. 2012, google-Books-ID: qcsoLHusAFsC.

[60] Oauth, “OAuth 2.0 — OAuth,” 2022, (visited on 12/09/2022). [Online]. Available: https://oauth.net/2/

[61] R. H. Jr, *Designing the Obvious: A Common Sense Approach to Web & Mobile Application Design*. Pearson Education, Nov. 2010, google-Books-ID: KNIeucozRpoC.

[62] N. Z. b. Ayob, A. R. C. Hussin, and H. M. Dahlan, "Three Layers Design Guideline for Mobile Application," in *2009 International Conference on Information Management and Engineering*, Apr. 2009, pp. 427–431.

[63] "JavaScript With Syntax For Types." (visited on 12/09/2022). [Online]. Available: https://www.typescriptlang.org/

[64] "Testing · React Native," (visited on 30/03/2022). [Online]. Available: https://reactnative.dev/docs/testing-overview

[65] "Using Hermes · React Native," (visited on 07/03/2022). [Online]. Available: https://reactnative.dev/docs/hermes

[66] H. K. Leung and P. W. Wong, "A study of user acceptance tests," *Software Quality Journal*, vol. 6, no. 2, pp. 137–149, Jun. 1997. [Online]. Available: https://doi.org/10.1023/A:1018503800709

[67] B. Hetzel and W. Hetzel, *The complete guide to software testing*, 2nd ed. QED Information Sciences, 1988. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/42384

[68] J. Nielson, "Discount Usability: 20 Years," Sep. 2009, (visited on 28/06/2022). [Online]. Available: https://www.nngroup.com/articles/discount-usability-20-years/

[69] K. Moran, "Usability Testing 101," Dec. 2019, (visited on 28/06/2022). [Online]. Available: https://www.nngroup.com/articles/usability-testing-101/

A Requirements Survey

We are making a new application for our university and we would love to hear your opinion of the current one, as well as your suggestions for new features, to help us build a better experience for the IST community. The survey should only take 5 minutes, and your responses are completely anonymous.

The development of the new application is within the scope of the thesis ”Advanced Implementation of Mobile Applications”, oriented by Professor Fernando Mira da Silva, which has the goal of finding the best practices to develop a mobile application, the challenges and complexities of developing them, and most importantly what are the best current alternatives to native development, their advantages, disadvantages, and which is is currently the best overall approach.

The survey will be open to answers until 23 of January. If you have any questions feel free to contact us at gabriel.almeida@tecnico.ulisboa.pt.

1. What are your roles at the university?

    - ☐ Student
    - ☐ Professor
    - ☐ Staff

◻ Alumni

◻ Investigator

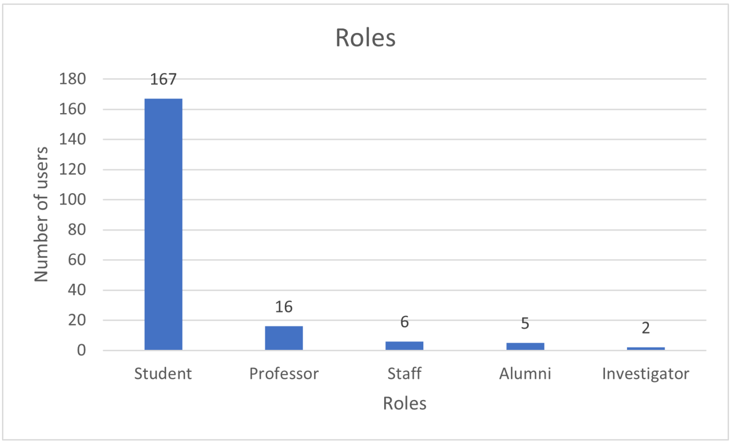


Figure A.1: Roles of the participants from the current app survey

2. How often do you use the application?

   ○ Every day

   ○ Almost every day

   ○ Sometimes

   ○ Almost Never

   ○ Never

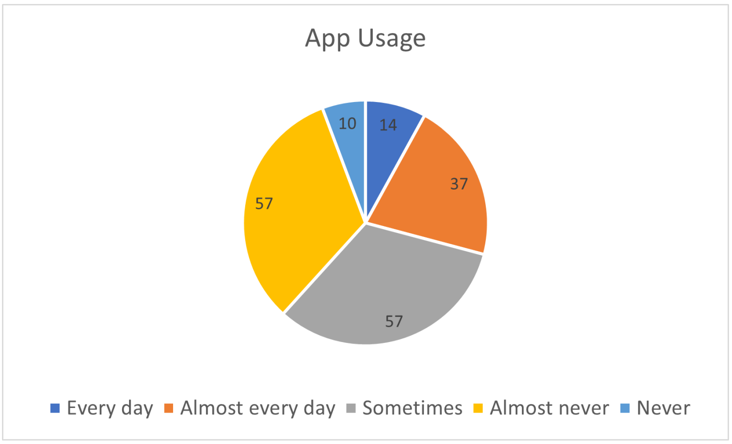


Figure A.2: Current application usage by the users

3. On a scale of 1 to 4, how would you rate your overall experience with the application?

Very bad	1	2	3	4	Excellent

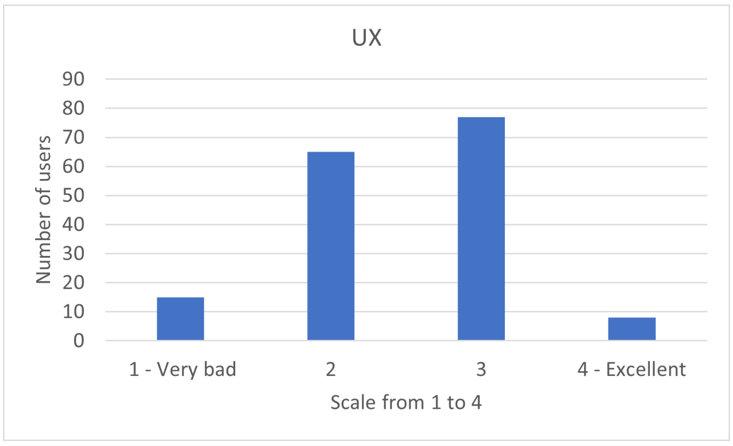


Figure A.3: User satisfaction with the UX of the current application

4. On a scale of 1 to 4, how intuitive is the application?

Not intuitive at all	1	2	3	4	Very intuitive

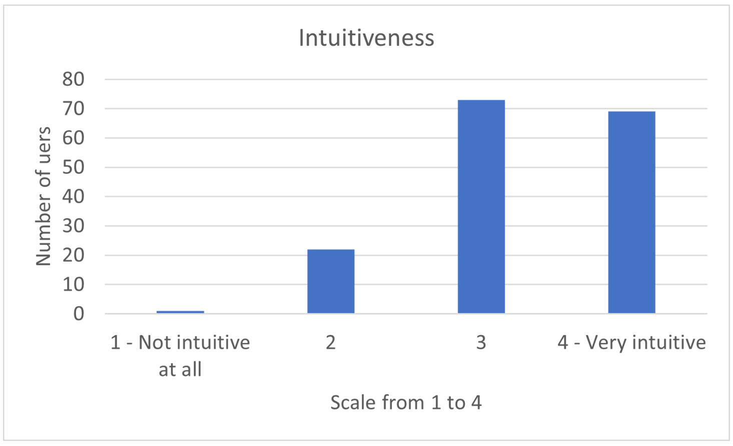


Figure A.4: User satisfaction with the intuitiveness of the current application

5. On a scale of 1 to 4, how would you rate the user interface?

Very bad	1	2	3	4	Excellent

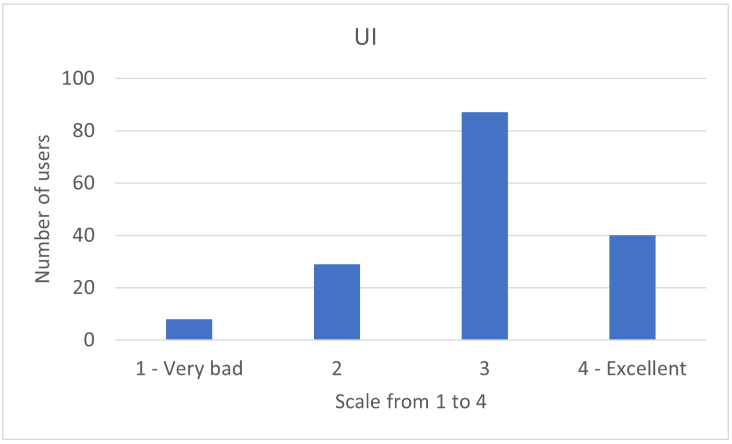


Figure A.5: User satisfaction with the UI of the current application

6. On a scale of 1 to 4, how fast is the mobile app for you?

Very slow	1	2	3	4	Very fast

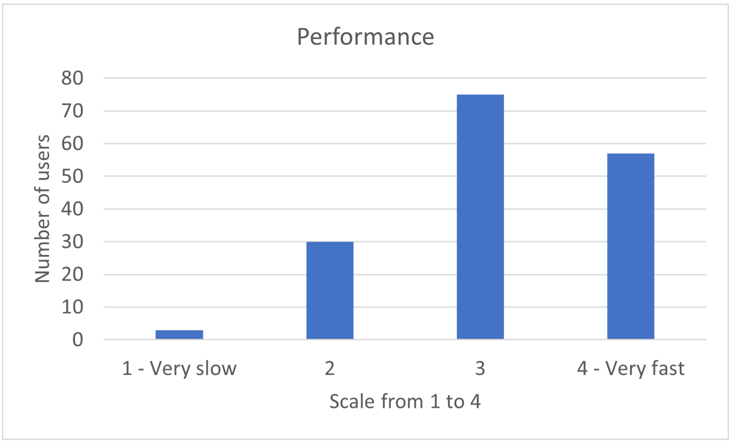


Figure A.6: User satisfaction with the performance of the current application

7. What do you like the most about the Técnico mobile app?

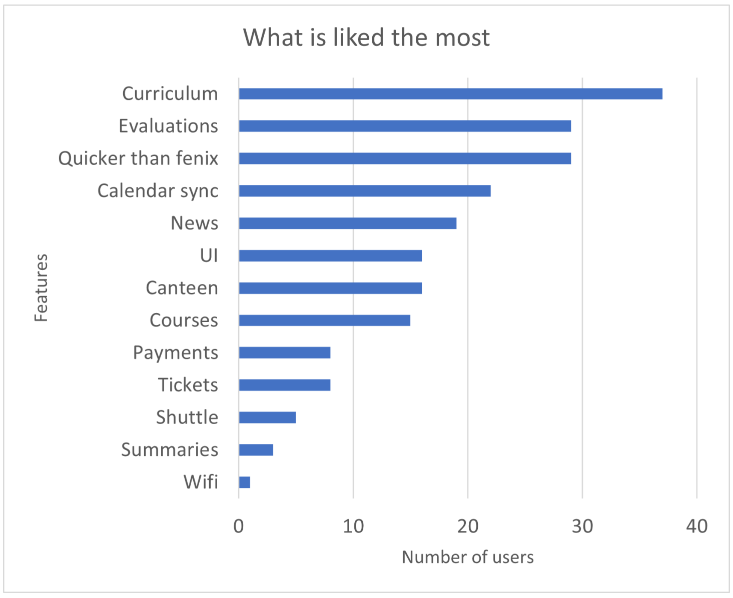


Figure A.7: What is liked the most in the current app

8. What do you like the least about the application?

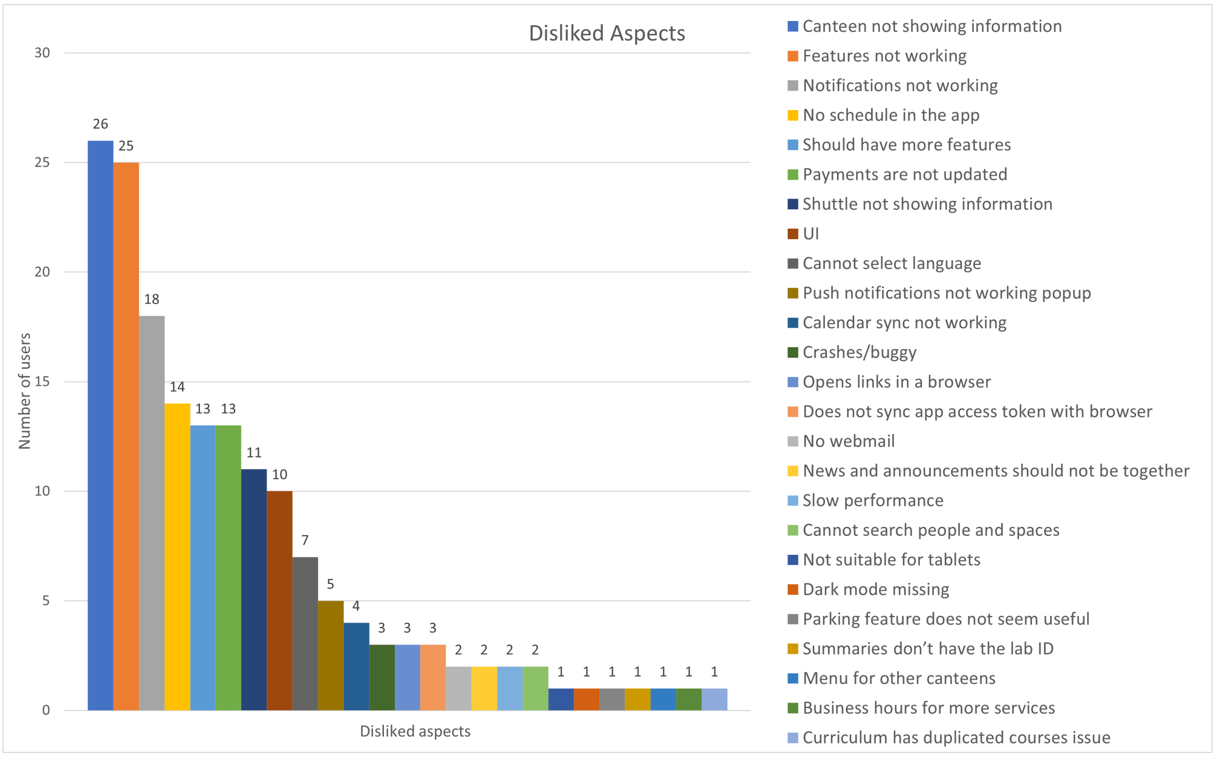


Figure A.8: Disliked aspects about the current application

9. When you use the application, which features do you tipically use?

- ☐ News
- ☐ Cafeteria
- ☐ Shuttle
- ☐ Parking
- ☐ Tickets
- ☐ Courses
- ☐ Summaries
- ☐ Evaluations
- ☐ Curriculum
- ☐ Payments
- ☐ Contacts
- ☐ Assiduity
- ☐ Other...

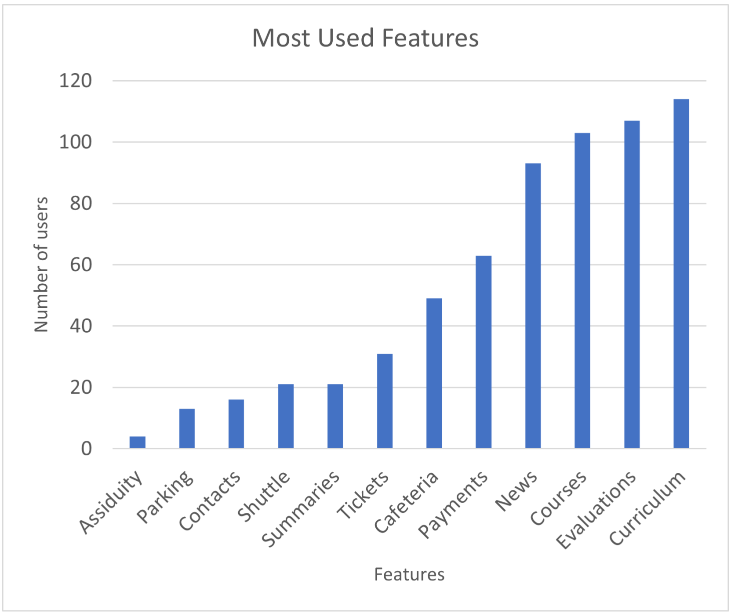


Figure A.9: Most used features on the current application

10. Are there any features that did not work as expected? If so, which ones and why?

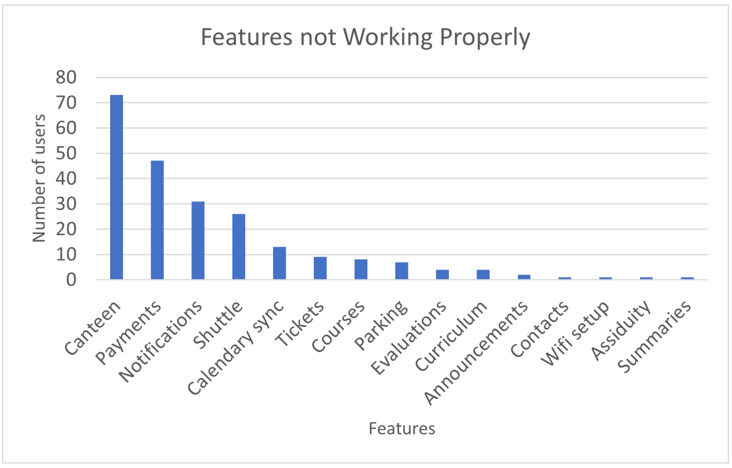


Figure A.10: Features not working properly in the current application

11. Are there any missing features in the app that could be useful for your daily life at Técnico?

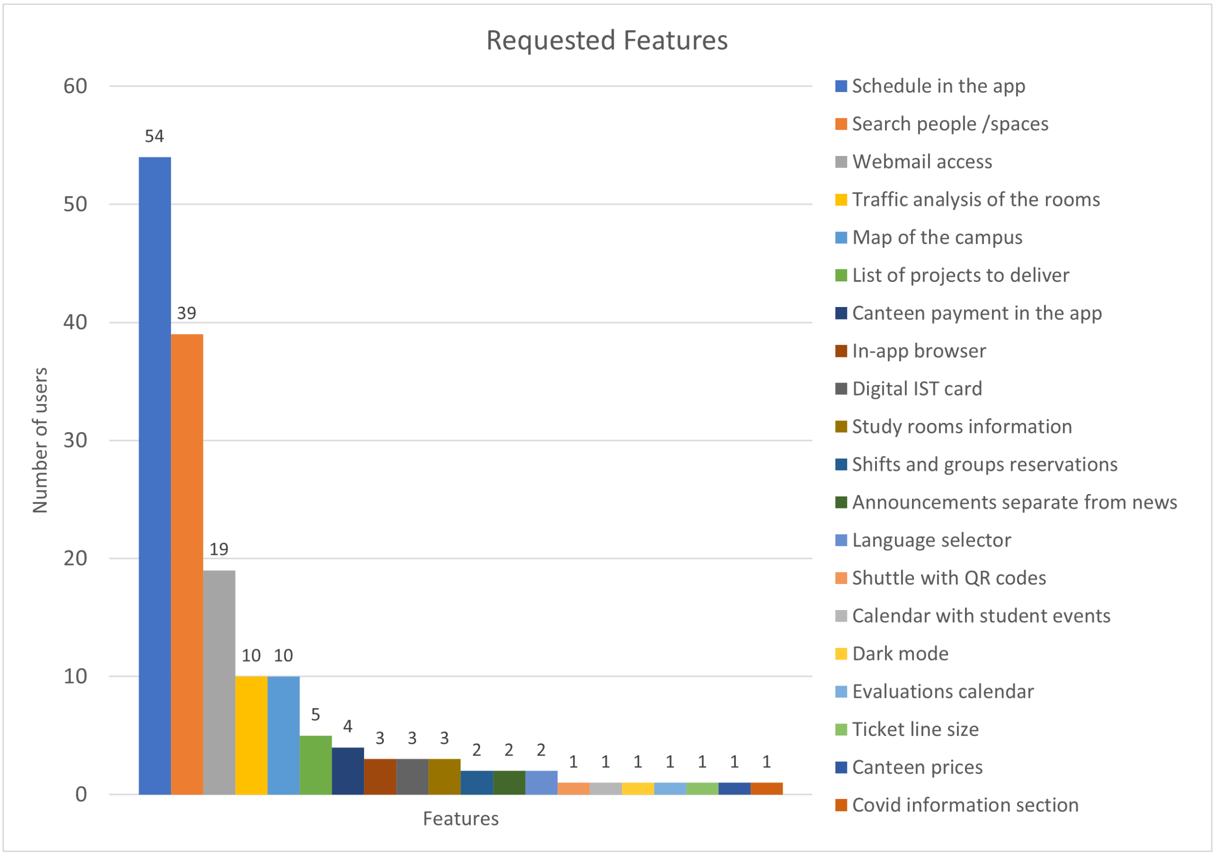


Figure A.11: New suggested features for the application

12. Do you feel there is room for improvement in Técnico's mobile app? If so, what?

(Similar answers to the previous question)

13. Is there any reason for never having used the application? (Only if answered ”Never” in question 2)

____

____

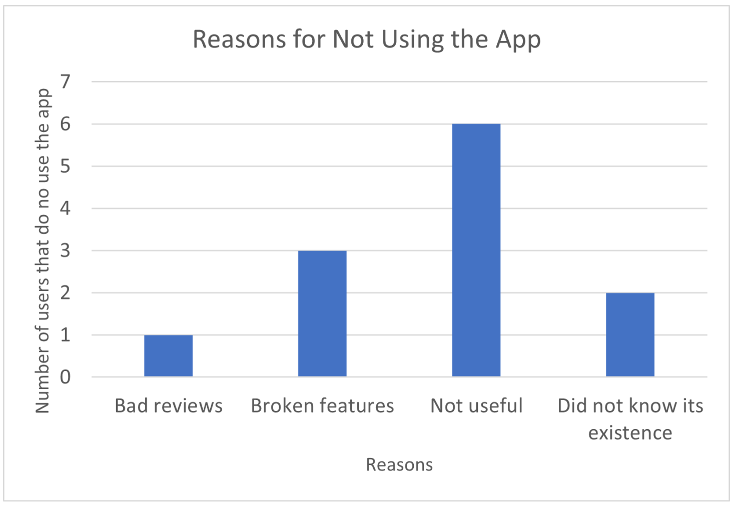


Figure A.12: Reasons for not using the current application

Thank you for your feedback!

B
API Endpoints

The new *shuttle* API being used is located at `https://shuttle.tecnico.ulisboa.pt/api`, and contains the following endpoints:

- /**stops**: contains information about all the stops;
- /**stops**/{**id**}: returns information about a given stop;
- /**routes**: shows every route the shuttle can take;
- /**routes**/{**id**}: returns information about a specific route;
- /**trips/schedulers**: contains all the planned trips for the shuttle service, with their respective time and date. This endpoint can also receive a *start* and *end* parameters to choose the start and end date respectively.

As for the *ticketing* API, it is hosted at `https://fenix.tecnico.ulisboa.pt/api/ticketing`, and provides the following endpoints:

- /**services**: provides information on all the services that support ticketing;

- **/tickets**: needs authentication and returns all the user's tickets;
- **/services/{serviceId}/queues/{queueId}/tickets**: *POST* request that assigns a new ticket, from a given queue and service, to an authenticated user.

Lastly, the Fenix API is still being updated, and besides the endpoints listed in [58], new ones are being added:

- **/search/spaces**: allows to search spaces within the IST campuses, and to filter which type of spaces it searches, like libraries, study spaces, classrooms, among others;
- **/search/people**: allows to search the members of the IST community;
- **/search/courses**: allows to search the courses lectured at IST.

C
UI Survey

We are making a new application for Instituto Superior Técnico and we would love to hear your opinion about the new design we are building for it. The survey should only take 2 to 5 minutes, and your responses are completely anonymous.

The development of the new application is within the scope of the thesis ”Advanced Implementation of Mobile Applications”, oriented by Professor Fernando Mira da Silva, which has the goal of finding the best practices to develop a mobile application, the challenges and complexities of developing them, and most importantly what are the best current alternatives to native development, their advantages, disadvantages, and which is is currently the best overall approach.

If you have any other suggestions feel free to contact us at gabriel.almeida@tecnico.ulisboa.pt.

1. What are your roles at your current university?

   ☐ Student

   ☐ Professor

   ☐ Staff

   ☐ Alumni

☐ Investigator

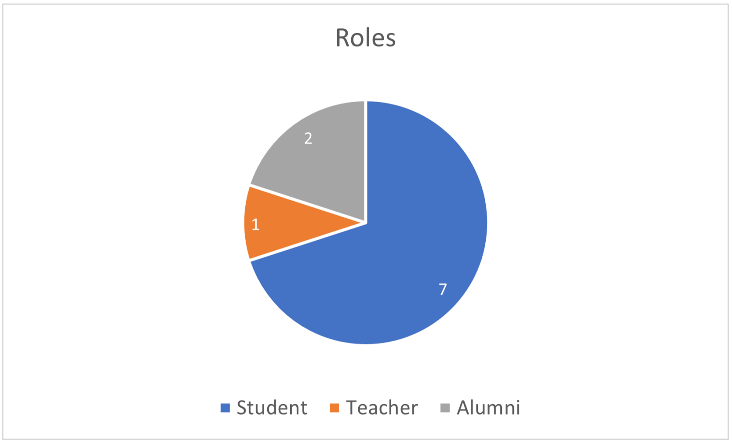


Figure C.1: Participants roles of the UI test survey

2. On a scale of 1 to 4, how intuitive is the user interface?

Not intuitive at all	1	2	3	4	Very intuitive

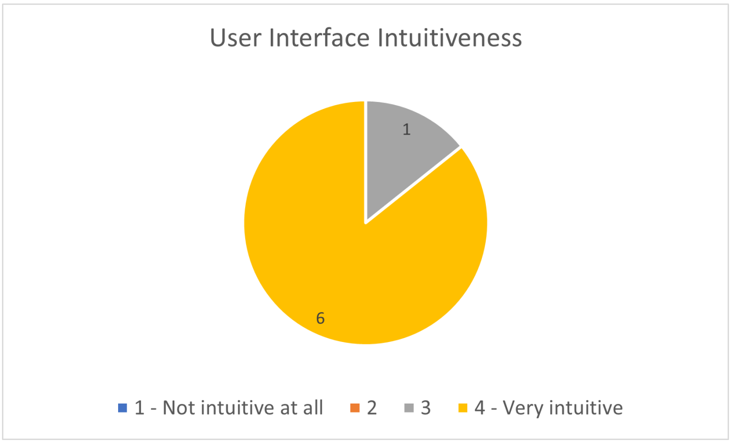


Figure C.2: User satisfaction with intuitiveness of the new user interface

3. On a scale of 1 to 4, how would you rate the user interface?

Very bad	1	2	3	4	Excellent

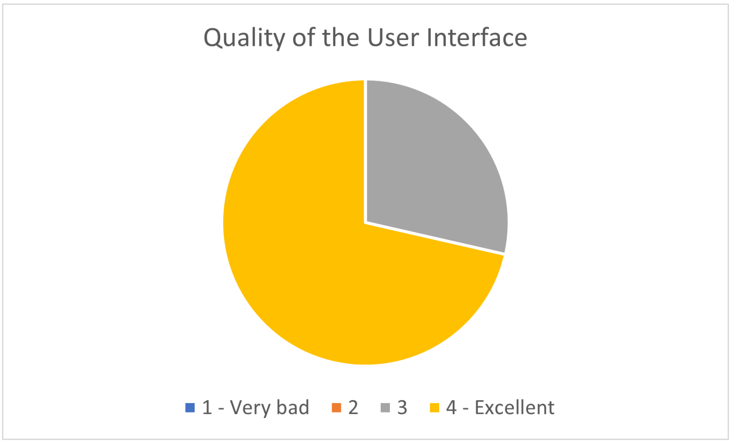


Figure C.3: User satisfaction with quality of the new user interface

4. On a scale of 1 to 4, how appropriate do you think the UI is to this kind of application?

Not appropriate at all	1	2	3	4	Very appropriate

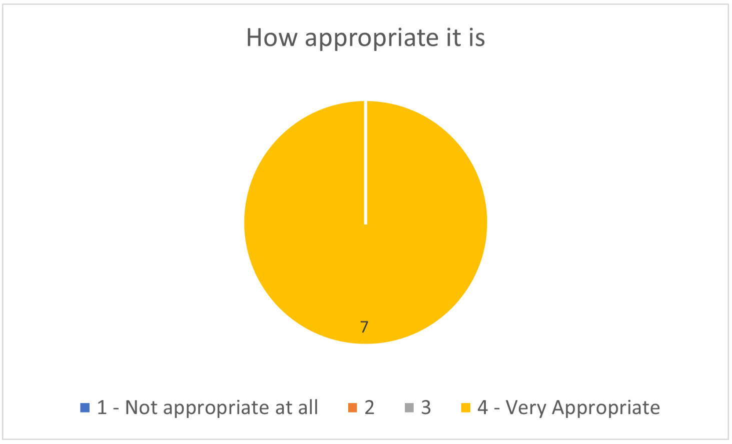


Figure C.4: How appropriate the users thought the new design was for new application

5. Did this UI exceed your expectations when you think about academic applications?

   ○ Yes

   ○ No

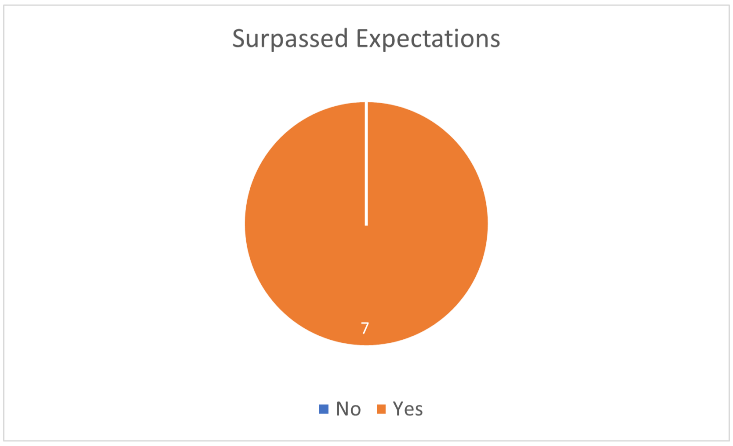


Figure C.5: Participants opinion on whether or not the new design surpassed their expectations

6. Do you have any suggestions? If so, what are they?

D User Acceptance Tests Survey

Thank you for participating the user testing for the new application for Instituto Superior Técnico. We would love to hear your opinion about the new application as a whole. The survey should only take 2 to 5 minutes, and your responses are completely anonymous.

The development of the new application is within the scope of the thesis ”Advanced Implementation of Mobile Applications”, oriented by Professor Fernando Mira da Silva, which has the goal of finding the best practices to develop a mobile application, the challenges and complexities of developing them, and most importantly what are the best current alternatives to native development, their advantages, disadvantages, and which is is currently the best overall approach.

If you have any other suggestions feel free to contact us at gabriel.almeida@tecnico.ulisboa.pt.

1. What are your roles at your current university?

   ☐ Student

   ☐ Professor

   ☐ Staff

   ☐ Alumni

- ☐ Investigator

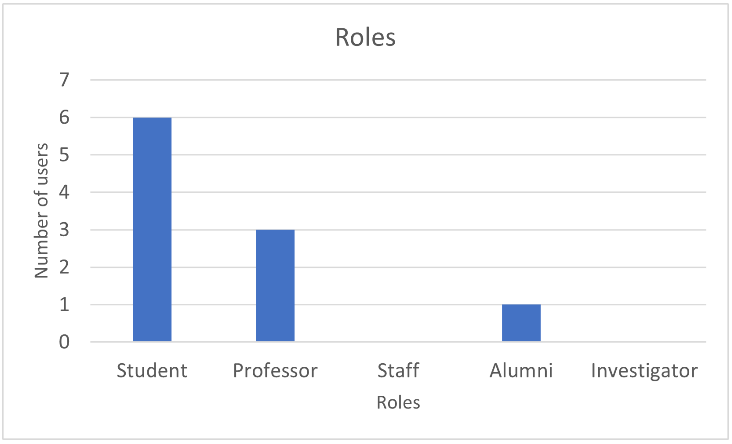


Figure D.1: Roles of the participants from the user acceptance tests

2. On a scale of 1 to 4, how intuitive and easy to use was the user interface?

Not intuitive at all	1	2	3	4	Very intuitive

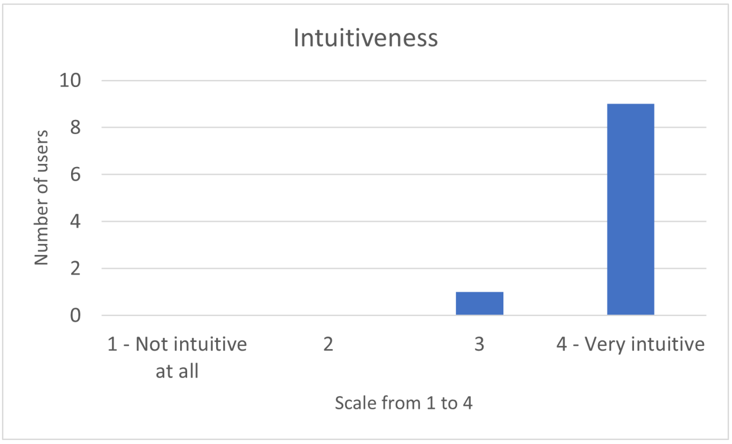


Figure D.2: User satisfaction with the intuitiveness of the new application

3. On a scale of 1 to 4, how would you rate the user interface?

Very bad	1	2	3	4	Excellent

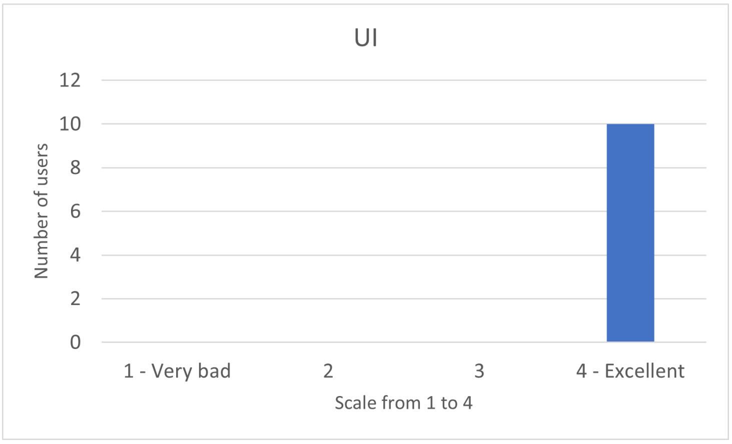


Figure D.3: User satisfaction with the UI of the new application

4. On a scale of 1 to 4, how would you rate your overall experience with the application?

Very bad	1	2	3	4	Excellent

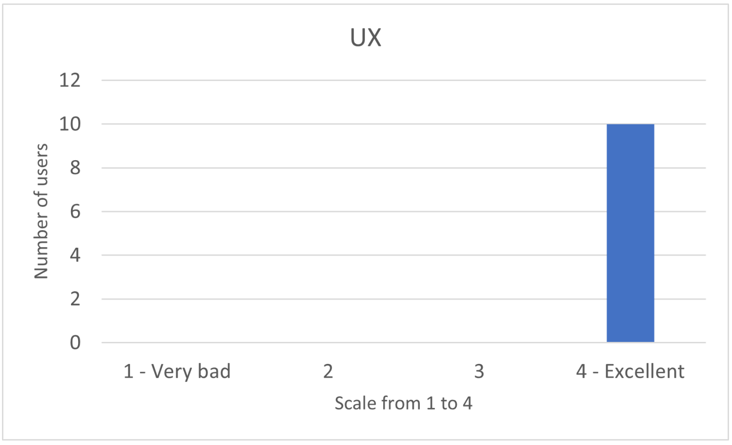


Figure D.4: User satisfaction with the UX of the new application

5. Was there anything unexpected or that made you confuse in the application? If so what?

○ Yes

○ No

6. Do you have any suggestions on how it could be further improved? If so, what are they?