BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE ENGLISH

DIPLOMA THESIS

Hybrid Frameworks for Web and Mobile Applications

Supervisor
Lect. Dr. Cojocar Dan

Author
Răducu Vlad-Rareș

2022

ABSTRACT

In the fast-paced environment that we currently live in, every resource is becoming scarcer, especially time. Thus, similar to other domains, methods of making the time we have more efficient and more profitable are constantly being developed, the community of software developers came up with their own solutions. One of those solutions is the use of hybrid frameworks for deploying the same projects on multiple platforms with the promise of only one codebase.

The aim of this paper is to present how this solution was inevitable to come and to show the current options a developer has if one chooses to take this path. Across multiple chapters, this thesis provides a detailed view of the internal architecture of the most popular hybrid frameworks at the moment as well as present important similarities and differences between them. In addition, it gives recommendations about important factors to take into account when choosing between different options of technologies as well as showcase how multiple building blocks come together in a practical application in the form of an online shop with the help of the Ionic framework .

Contents

Chapter 1

Introduction

The massive number of software applications nowadays need to be available on different operating systems [SB16] and there is a very high demand for developing fast and reliable apps in a very short period of time. First-movers[1] can gain an advantage through sustainable leadership in technology [LM88] and thus it is of utmost importance to know the advantages and disadvantages when choosing a tech stack when developing a new application.

1.1 Motivation

In the most recent years, the number of software tools and technologies which solve similar problems dramatically increased. There is an overwhelming amount of available information out on the internet to be filtered when deciding which tools should be best to use when a professional wants to develop a new software application. The major disadvantage in the information age is that one has to spend a lot of effort gathering the relevant data for their use case. This process of choosing between multiple available options may consume some of the self's limited supply of energy, thereby rendering the resource less available for further demands [VBS$^+$14]. This thesis wants to facilitate this decision for the professional.

1.2 Aim and Scope

Considering the motivation written above, the aim of this paper is to provide a detailed overview of multiple available hybrid frameworks on the Internet and help an individual to make an educated choice while deciding on what technology to use in future projects. We believe that the way in which the information was structured,

[1] A first mover is a service or product that gains a competitive advantage by being the first to market with a product or service [Tar20]

as well as the real-world examples throughout this paper, should greatly facilitate this choice for the experienced developer, together with providing a good understanding for the non-technical decision-makers within a project/business on what factors are taken into account when choosing a given framework for developing a project.

1.3 Approach and Outcomes

The paper is structured into five big chapters, the first one being the introduction, the one you are reading at the moment. After being accustomed to what this paper has the focus on, the following chapters have the aim to build the information for a better understanding of the discussed topic.

The second chapter, 'Hybrid frameworks' aims to familiarize the reader with the concept of a framework in regards to software development as well as make him understand why the apparition of hybrid frameworks was needed in the context of mobile and web development. The sections of this chapter aim to introduce the web environment to the reader and how it relates to the mobile one. For the outcome of this chapter, the reader should understand the differences between these platforms as well as understand their role within the same ecosystem.

The third chapter, 'What makes a hybrid framework', is the core of the theoretical part of this thesis and its aim is to present the actual technicalities of how written code is transformed into information that can be understood by the operating system on mobile platforms and on web platforms as well. It also provides information about how the hybrid frameworks had developed a bridge for this "transformation" of code such that once written code can be understood by different operating systems of different platforms. Following this, a presentation of the most popular frameworks' architecture at the moment of writing this paper and a list of important factors which should be taken into account when deciding on a framework for a given project lead to the outcome stated in the 'Aim and scope', namely an educated decision on the chosen technology.

The fourth chapter, 'Practical application', provides a detailed description of a project built with the help of the hybrid Ionic Framework. Its main outcome is to present how different technologies and programming techniques can be assembled together in order to achieve a working application on multiple platforms.

In the final chapter, 'Conclusion', we addressed some general thoughts about hybrid frameworks, the current sentiment about those together with our opinion about the future of these technologies.

Chapter 2

Hybrid Frameworks

2.1 Short History

The last decade showed a lot of changes and innovations in the technological sector. Since the beginning of the Internet, there was a strong correlation between browsers and mobile phones. As numerous events that happened on this planet had an economic reason, the case of hybrid frameworks for web and mobile applications is no exception.

Ever since the first mobile phones had access to the Internet, a trend was formed from which the world did not and will not escape. The narrative started to become ″speed and portable″, the technological advancements of the industry consolidating this mantra. Around the year of 2009, the ITU(International Telecommunication Union - United Nations sub-organization) had reported that there were more than 3,4 billion unique users of mobile phones. Thus, a new market had opened for the developers, businesses, and software sellers that could be captured.

This historic milestone was also related to a new major breakthrough in the mobile phone industry: the apparition of smartphones. What seems trivial at the time of writing this thesis, at that time it was an incredible advancement: a portable device that can be fit in a pocket that combines the functions of a mobile telephone with the computational powers of a personal computer into one unit. In order words, smartphones facilitated a wider variety of software. Those slowly become indispensable in everyday life because of their offering of a multitude of options for information, communication, education, and entertainment. Their hardware made it possible to have internet through Wi-Fi or cellular networks, multimedia functionality(music, video, camera, or gaming), and even GPS-based navigation, on top of the core phone functions such as voice calls or text messaging. [HCK$^+$15]

The apparition of smartphones also brought along two, soon to become at that time, giants in the mobile industry: (Apple)iOS and (Google)Android. Even though the iOS operating system was released first in the United States on June 29, 2007, in the form of the first iPhone, Google was not discouraged and released its own operating system in the form of Android 1.0 as a beta version (launched for developers) on November 5, 2007, the first Android smartphone being only later announced on September 2008, in the form of T-Mobile G1 (also known as HTC Dream). [Joh]

As time passed and the two communities grew, so did the developers which worked on each of the newly launched operating systems. More and more businesses wanted to be part of the opportunities these two new platforms had to offer and they allocated the necessary resources for it, but soon they realized that mobile development was getting harder as years passed by, each year new devices being launched on the market with new sensors and hardware capabilities, as well as the developing and designing paradigms being changed, on top of the rapidly increasing competition. Soon, these businesses realized that maintaining at least 3 dedicated platforms (web, Android, and iOS) does not generate the highest yield, so they started to think of possible alternatives.

Similar to Java's promise of "write once, run anywhere", something of the same mantra had to be developed for mobile platforms as well. With that in mind, the first solutions to solve the issue of having to maintain multiple platforms of the same products appeared in the programming community in the form of hybrid frameworks (the term cross-platform framework is used as well) as we know them today. We will explore this concept in the following chapters.

2.2 What is a Framework?

The software technologies have been developing at a rapid rate in the last decade and numerous frameworks have been released for the developers to be used at will. Starting in high school, any student who aimed to pursue a career in the development of software applications had coded a lot of similar building blocks when creating an application.

Frameworks (in the space of computer science) are of key importance for developing large-scale object-oriented software systems. They promise higher productivity and shorter time-to-market through design and code reuse. Therefore, one can infer that a software framework is an abstraction of a general use case that can be selectively modified in order to achieve domain-specific solutions for the given business requirements. [Rie00] In order not to rewrite the same type of code for each piece of software that serves the same functionalities, communities of programmers have worked together in order to build such templates that can be used almost as boilerplate code[1] in many situations.

[1]In Information Technology, a boilerplate is a unit of writing that can be reused over and over

Depending on the area in which these frameworks are used, these provide a standard way to build and deploy applications and expose reusable code libraries as well as other support programs, compilers, and toolsets.

2.3 Hybrid frameworks

In the scope of this thesis, we will analyze how hybrid frameworks help in the process of developing software applications and what are the best choices in this regard. Before we start this discussion, we first have to present and understand the other available options when developing applications for web and mobile platforms (the topic or this paper).

2.3.1 Native Mobile Applications

When developers say they have a native solution when writing the code for a mobile application they infer that the application has been written using the native development language and tools for that specific platform. An application that would be run on an android device would have been developed using Java or Kotlin while a native iOS application would have been built using the iconic Swift.

The main perks of using a native solution for developing a mobile application are the high accessibility that programmers have to the resources of the given device (such as sensors, the contact list, location, etc.) as well as the smoothness with which the application runs. On top of that, by building a mobile application with native technologies, one can have access to all of the native user interfaces and layouts available on that device, thus a better user experience overall.

The downside of developing a native application is that it cannot be reused on other platforms. Supposing a programmer develops an Android application, it cannot be run on an iOS device or be deployed as a web application. The whole project has to be written again with new code, new technologies, and spending more resources.

2.3.2 Web Applications

Web applications are the traditional software projects written in HTML, CSS, and JavaScript and that can be opened in any browser. These kinds of applications are more static and do not have all the capabilities that native applications possess. It is very hard to access at the maximum capability the resources of a device by using a

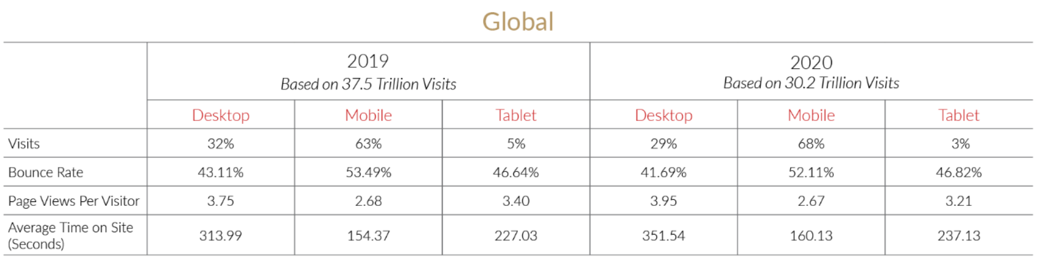
without change. By extension, the idea is sometimes applied to reusable programming, as in "boilerplate code." [Zav18] web application and most of the time it is not convenient at all to do. Even though deploying a web application provides wider and more stable access to the users and more control by the owners (you do not have to depend on the policies of some mobile application stores to spread your application), the limitations of a browser do not appeal so much for a mobile user.

Initially, web pages were a little more than simple textual documents with limited user interaction capabilities. As time went on, more and more tools had been developed in order to make those more interactive and support advanced graphics and animations [MT08]. Even though at the time of this writing web applications offer a wide variety of options for mobile users and developers, the capabilities still do not match their native counterparts.

2.3.3 Hybrid Applications

The features and options offered by smartphones through mobile applications are immense and many web applications would benefit a lot if there was a possibility to make use of those [VJ17]. On the other hand, mobile application users and developers would highly prefer if there was a web version of the same application (for using the perks that a laptop/computer offers). The latter would also fancy this possibility without having to rewrite the whole codebase.

As seen in the table from Figure 2.1, there is a very balanced proportion between the desktop and mobile visitors who accessed web pages in the years 2019 and 2020. This strongly suggests that developers and web owners should think about how to satisfy both types of users when creating a product. The question shifted in the past years from "is my business model more suitable for web/mobile users" to "how can I satisfy both mobile and web users when I implement my business model".



Global

	2019 *Based on 37.5 Trillion Visits*			2020 *Based on 30.2 Trillion Visits*		
	Desktop	Mobile	Tablet	Desktop	Mobile	Tablet
Visits	32%	63%	5%	29%	68%	3%
Bounce Rate	43.11%	53.49%	46.64%	41.69%	52.11%	46.82%
Page Views Per Visitor	3.75	2.68	3.40	3.95	2.67	3.21
Average Time on Site (Seconds)	313.99	154.37	227.03	351.54	160.13	237.13

Figure 2.1: Table showing the results of a research on desktop vs mobile vs tablet users accessing web applications [Eri]

Hybrid mobile applications are developed by using standard web technologies (i.e. HTML, CSS, and JavaScript) and all service requests to the Platform API are mirrored by a cross-platform JavaScript API. In this context, a hybrid development framework (e.g., Apache Cordova) can be defined as a software component that allows developers to create a cross-platform web-based mobile app by providing (i) a native wrapper for containing the web-based code, and (ii) a generic JavaScript API that bridges all the service requests from the web-based code to the corresponding platform API [MRST15]. Therefore, hybrid frameworks are integrating tools that the programming community developed in order to shorten the gap between web and mobile technologies and solve the rising demand for a solution that can satisfy the needs of both types of users.

Chapter 3

What Makes a Hybrid Framework

Before taking a deeper look and deciding on which of the multiple frameworks available on the world wide web might fit best for the use case in hand, the first step that one has to understand is how does the architecture behind such a framework does function. The second part would be to understand what are the challenges each of these frameworks is facing and how they aim on solving those.

3.1 Overview

The first thing we have to understand is that hybrid frameworks are templates for building the front-end [1] (client-side) part of an application. Since the main focus is on the presentation layer, all the frameworks focus on providing a library of UI [2] elements that are compatible with multiple platforms: web, mobile, and desktop and all the relation logic which comes with this part of the development process: authentication, navigation/routing, layout, themeing, testing, storage and so on.

Most of the available hybrid frameworks on the web are based on existing web frameworks and technologies and in order to achieve the aforementioned parts written in the above paragraph they need to find an efficient way of converting the web codebase into platform-specific codebase for each of the platforms we intend to run the code on (e.g. Android, iOS). There are special cases such as Flutter (which has its own programming language – Dart) where you need to convert the code for both web platforms as well as mobile platforms.

[1]In the modern days of software engineering, most of the applications have a **front-end** (or client-side rendering). In this type of architecture, the browser will be in charge of rendering the final form of the application, the one which the user will see (thus the name of client-side, since the rendering happens on the user's computer). This also implies that most of the logic which was involved in the process of presenting the results (the outputs) to the user (this layer also being known as the presentation-layer) is also handled on the client-side[Mar].

[2]A user interface (UI) is the visual representation of the respective program together with the place where the user can interact with the program at hand in terms of inputs and outputs [SJWM05]

After the conversion is done, everything is in place. This is the point to decide and analyze which of the existing frameworks fits best with the business model and requirements we are trying to implement.

There are many questions we would like to ask ourselves in order to decide what would be the best choice. Do we have a complex model which needs a lot of customization and refining? Then, a framework that provides a wide variety of customizable widgets that have many interfaces which can be extended. Do we want a generic application to be developed in a fast period of time ? We would guide ourselves to a framework that has as much of out-of-the-box code as possible in order for a fast deploy of the application. Is the business we are trying to build of long duration and with many expected maintenance changes? We would like the framework we chose to have a numerous and active community in order to help with the potential roadblocks we may face.

3.2 Architecture

The aim of this subchapter is to discover how a hybrid written application can make once written code on multiple platforms. In order to understand this, we need to learn about how human-written code is run on the given platform, at least on the surface level. Since most of the most popular frameworks use web-based frameworks, we will start to understand how code is being interpreted and run on a browser.

Every browser that is present on a computer has a built-in engine which understands JavaScript code, contrary to many backend programming [3] languages which have to be compiled in order to run. The process of converting the human written code to code which the computer understands is pretty simple: the browser takes the source code and it inserts it into the pipeline of an interpreter. For example, in the case of the Ignition Interpreter (which is one of the most popular used since its release in 2017), the source code is inserted into a baseline compiler which transforms the programming code into bytecode [4] . The byte code is then inserted into

[3]Back-end development refers to the business logic of an application, usually located on a server, outside of the reach of the client. Following object oriented programming and separation of concerns, most of the computing is done in this layer of the architecture: price calculations, filtering, logins etc. Also this is the layer responsible for communication with databases and storing information. [Mat]

[4]Bytecode is the intermediate step between the source code (written by the programmer) and machine code (understood by the computer). It is designed in an efficient way such that after it is inserted into an interpreter, it will result in runnable machine code. Bytecode is compact numeric codes, constants, and references (normally numeric addresses) that encode the result of compiler parsing.

an interpreter [5] which converts it into binary code which can be understood by the machine (Figure 3.1).

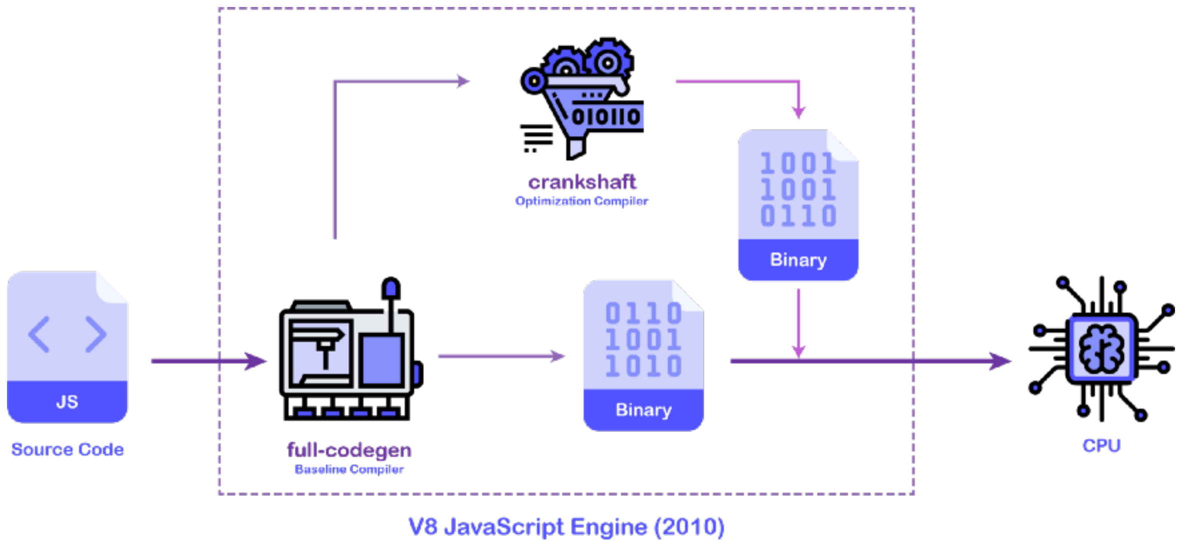


Figure 3.1: 2010 V8 JavaScriptEngine [Uda]

The process of running code on mobile phones is very similar. In the case of Android, the supported native languages are Java and Kotlin. In order to run these two, the process is the following: the source code is compiled by javac (the Java compiler), the javac generates a Java byte-code file which is sent to the JVM (Java Virtual Machine); the JVM understands byte-code and it converts it into machine code using the JIT (just in time compiler) and then the computer can execute the desired commands [Ban]. The only difference in this process when deploying the code to a mobile device is the part where the Java byte-code is sent to the JVM. In the case of Android mobile devices, there is a DEX compiler that takes the Java byte-code and transforms it into a Dalvik byte-code. The latter is then sent to the DVM[6] (Dalvik Virtual Machine), a specific virtual machine created for mobile devices. The described processes can be followed in Figure 3.2 and Figure 3.3 below this text.

A very similar process is used for compiling iOS applications. Applications for

[5] In computer science, an interpreter is a program that accepts any program (the source program) expressed in a particular language (the source language) and runs that source program immediately. [WB00] For program execution, an interpreter often employs one of the following strategies:

1. Immediately perform the instructions which were obtained after parsing the source code;
2. Execute the object code or some efficient intermediate representation of a high-level source code following a given protocol.
3. Execute the stored compiler-generated bytecode after matching it with the interpreter's Virtual Machine code.

[6] The Dalvik Virtual Machine (DVM) is an android virtual machine optimized for mobile devices. Android runtime must support: limited processor speed, limited RAM, no swap space, battery-powered, diverse set of devices, and sandboxed application runtime. Thus, the DVM was designed such that the aforementioned factors are optimized [Ehr10]

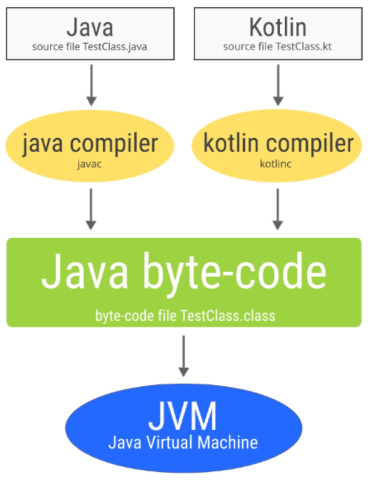


Figure 3.2: Java and Kotlin code compilation [Ban]

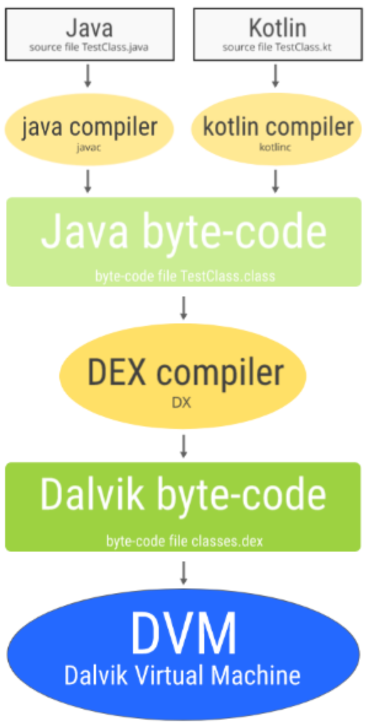


Figure 3.3: Java and Kotlin code compilation for Android devices [Ban]

iOS are written in Swift [7] or Objective-C [8] and they follow the same principles of classic C family language compilation: preprocessing, compilation, assembly, and linking. In recent years, the Xcode [9] developing platform was developed which came with big efficiency improvements in the process of code compilation [Ger18]. For the purpose of this thesis, it is not of the utmost importance to dig into the details of this process, but rather to see the overview: the source files are interpreted and built by the Xcode build system, and they are sent to the Swift and Clang compilers, the object files are then linked together by the linker and then converted later to machine code which can be understood by iOS devices (Figure 3.4).

[7]Released in June 2014 during the annual Worldwide Developers Conference, Apple unveiled a new programming language for developing iOS and Mac OS applications under the name of Swift. Swift is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc. and the open-source community. It was developed with the purpose of replacing the old and uncomfortable Objective-C which was constrained upon the developer by Apple in the past. [GM15]

[8]The Objective-C language is a simple computer language designed to enable sophisticated object-oriented programming. Objective-C is defined as a small but powerful set of extensions to the standard ANSI C language. Its additions to C are mostly based on Smalltalk, one of the first object-oriented programming languages. Objective-C is designed to give C full object-oriented programming capabilities, and to do so in a simple and straightforward way[App].

[9]Xcode is Apple's integrated development environment (IDE) and it is aimed squarely at developing, testing, and packaging OS X and iOS applications, utilities, and plugins. [KN13]

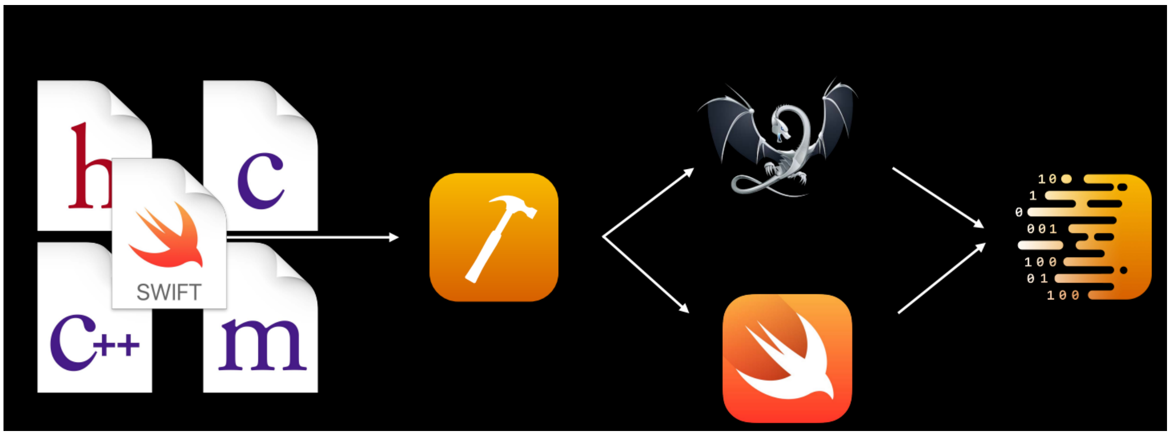


Figure 3.4: Xcode Build Process [Ger18]

3.3 The Hybrid Solution and Popular Hybrid Frameworks

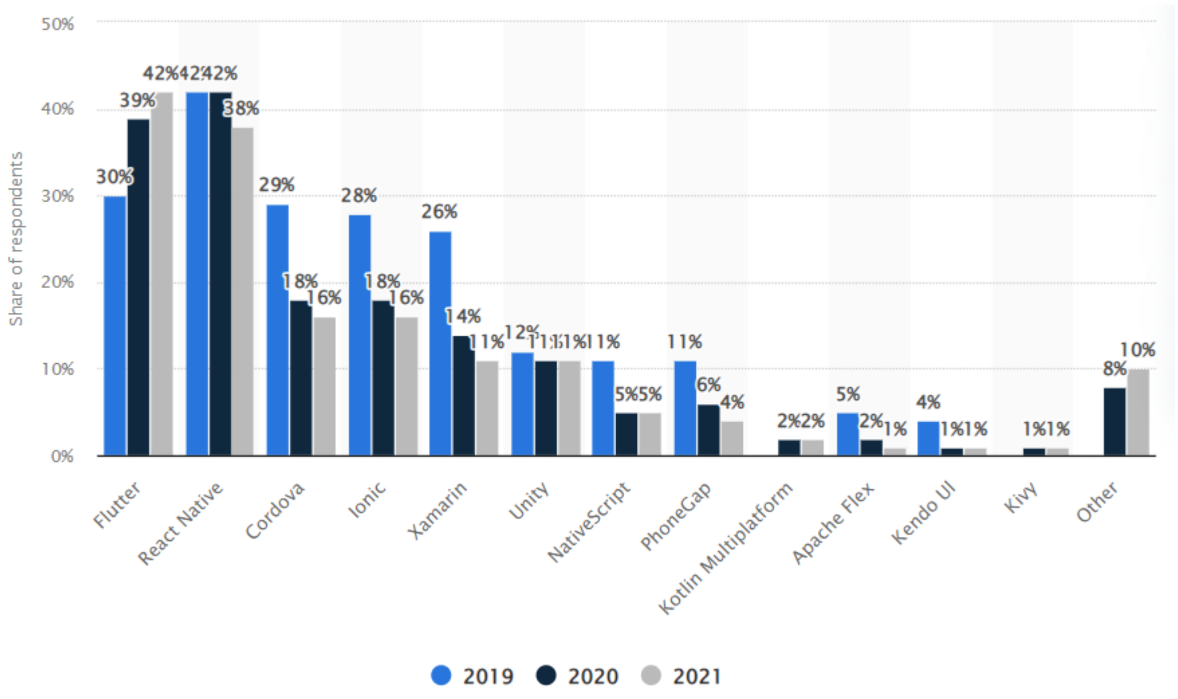


Figure 3.5: Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021 [Lio]

As we have clearly seen in the previous section, the main issue one has to solve when trying to deploy code written on one platform to multiple platforms are hardware and software compatibility problems. Different types of operating systems have different requirements and the machine code which is sent from the application to the operating system has different meanings across multiple platforms. Thus, the main problem a hybrid framework has to solve is to provide such tools that once a code was written on one platform, it can be run across multiple operating systems.

In order to achieve this, each hybrid framework solution has to provide a protocol, a bridge, between the once written code and the machine code. Instead of having just one compiler which takes the written code and transforms it into machine code suitable for one platform, the hybrid solutions have to do a series of transformation in order to covert the HTML, CSS, and JavaScript code into different machine codes, one for each of the platforms the framework provides.

Within this section, an overview of different architectures of the most popular hybrid frameworks which were used between 2019 and 2021 is presented. According to a statistic released in June 2021 by Statista, the most popular hybrid frameworks are: Flutter, React Native, Cordova, Ionic, and Xamarin (Figure 3.5). Their approach to handling this mapping of code across multiple platforms is analyzed in the following subsections.

3.3.1 React native

In 2015 React Native was launched by Facebook (now Meta) as an open-source cross-platform (another term used in the industry for hybrid applications) framework used in mobile app development. On their official site, it is advertised as ″written in JavaScript - rendered [10] with native code″, React Native relies on the work of their ″sibling″ React and its large library, thus having access to many features available to ″normal″ React developers who build web applications.

JavaScript is the building block of the React framework. In order to make it possible that JavaScript can be run on mobile devices, the React community had to come up with an architecture such that the available tools provided by the mobile platforms would fit in, so they chose to use JavaScriptCore(JSC) directly, mostly because of the Apple's iOS specifications which state that in order to be able to run JavaScript code on their platform, one has to use the WebKit JavaScript framework for achieving that. In the same previous architecture, they relied on a ″bridge″ between the resulted JavaScript code interpreted from the React Native Framework which would be translated into native code. The bridge would use asynchronous JSON messages between the JavaScript part and native part, without the ends being aware of one another.

[10]Within the React framework, speaking in broad terms, ″rendering″ is the process of ″painting″ (or repainting) the DOM. React is a JavaScript-driven library and it maintains a ″virtual DOM″ which is periodically compared with the real DOM of a webpage. A react application is, at its core, a JavaScript object which holds in one of its attributes (props) a tree-like structure of all the nodes of the application (which resemblance the DOM of a webpage). The process in which the React engine performs a scan through this virtual DOM collecting the current state, structure, and desired changes of the UI is called **rendering** [Ank]
″**The Document Object Model (DOM)** is the data representation of the objects that comprise the structure and content of a document on the web.″ [MDN]

The new architecture (which is used at the time of writing this paper) proposed a clean and robust solution for all the shortcomings of the previous iteration (such as a slow channel of sending JSON serialized messages or a relatively inflexible JavaScript engine for translating code on other platforms).

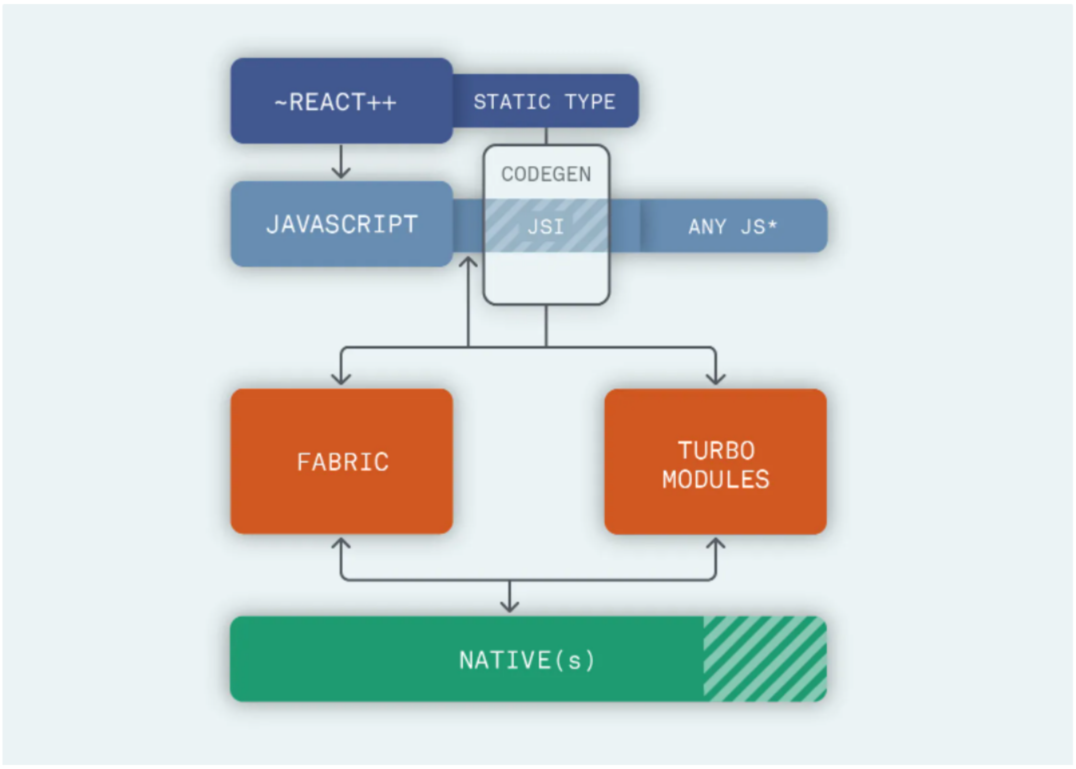


Figure 3.6: High level view on the React Native underlying architecture [Lor]

The first major improvement of the new architecture is the addition of a static type checker (at the moment React Native supports Typescript or Flow) which enabled the framework to create a tool similar to a code generator to ease the compatibility between the JavaScript and the native side. This addition is very important because, with the help of a static type checker, the generator is able to define the interface of the files needed by Fabric and TurboModules to send the messages between the two, speeding up the channel of communication at the same time since validation of the data is not necessary every time.

The second part of the framework architecture is how it consumes the code written in React Native. This layer is similar to an interpreter of native programming languages and it is composed of the following elements: the minified JavaScript code produces from converting the React Native source, the JavaScript Interface (also known as JSI), and JavaScriptCore (JSC, a JavaScript engine). The JSI is exactly what the name suggests: an interface; with this addition, the framework gained more flexibility as well as compatibility, since this structure allows the integration of different JavaScript engines and also allows for holding references to C++ Host Objects as well as invoking methods on those. Since C++ has been one of the few possible ways of sharing code between the Android and iOS without including JavaScript in the process, the fact that the framework included JSI and allowed C++ code to be part of the architecture enables numerous changes to the structure and it leaves room for the opportunity for writing more C++ code for applications.

The third layer of this architecture consists of two components, Fabric and TurboModules, and this is the layer that has the role of being the channel between the JavaScript and the native part. With the help of Fabric, the previously "Shadow Tree" (a tree-like resemblant structure that has the role of memorizing the structure of the UI components) is now created straightly on C++ (code that can be run directly on the native platforms), thus reducing the number of steps across the two realms and greatly improving the responsiveness of the user interface. On top of that, having access to JSI, Fabric is able to expose the UI operations to the new Shadow Tree (the structure which determines what is shown on the screen), thus allowing straight interaction from both ends. Moreover, the TurboModules play a very important role in terms of performance (a popular disadvantage of hybrid applications to native ones) and it permits the JavaScript code to load each module only when it is really needed holding a reference to it, unlike the previous iteration when all the modules had to be initialized in the startup phase of the application.

React Native wants to be "agnostic" [11] to its native platform and since their last iteration of the underlying architecture, they can say they managed to achieve this goal. Since the Facebook team does not own either the iOS or the Android platform, the last block had the job to enable the code on the native platforms (basically a lot of packages that held Android/iOS UI elements such as ScrollView, WebView, PushNotification, etc.) was within the same package/repository with all the other elements of the framework. The last part of this architecture is now called "Lean core" and its aim is to take all the code present in the React Native codebase and extracts it into separate packages/repository and be enabled depending on the platform on which the application will be executed on. In this way, the last part becomes closer to a "horizontal" layer rather than a "vertical" one, thus considerably reducing the size of the codebase. Also, this inclusion allowed the community to further maintain the UI elements which are not used by the Facebook team and were receiving less "care" in the past. [Lor]

[11] A person who does not believe in the existence of God. In this context, not being dependent on another entity/set of entities

3.3.2 Flutter

From the official documentation of this framework, "Flutter is a cross-platform UI toolkit that is designed to allow code reuse across operating systems such as iOS and Android, while also allowing applications to interface directly with underlying platform services." Different from other frameworks, the Flutter framework has its own programming language: Dart.

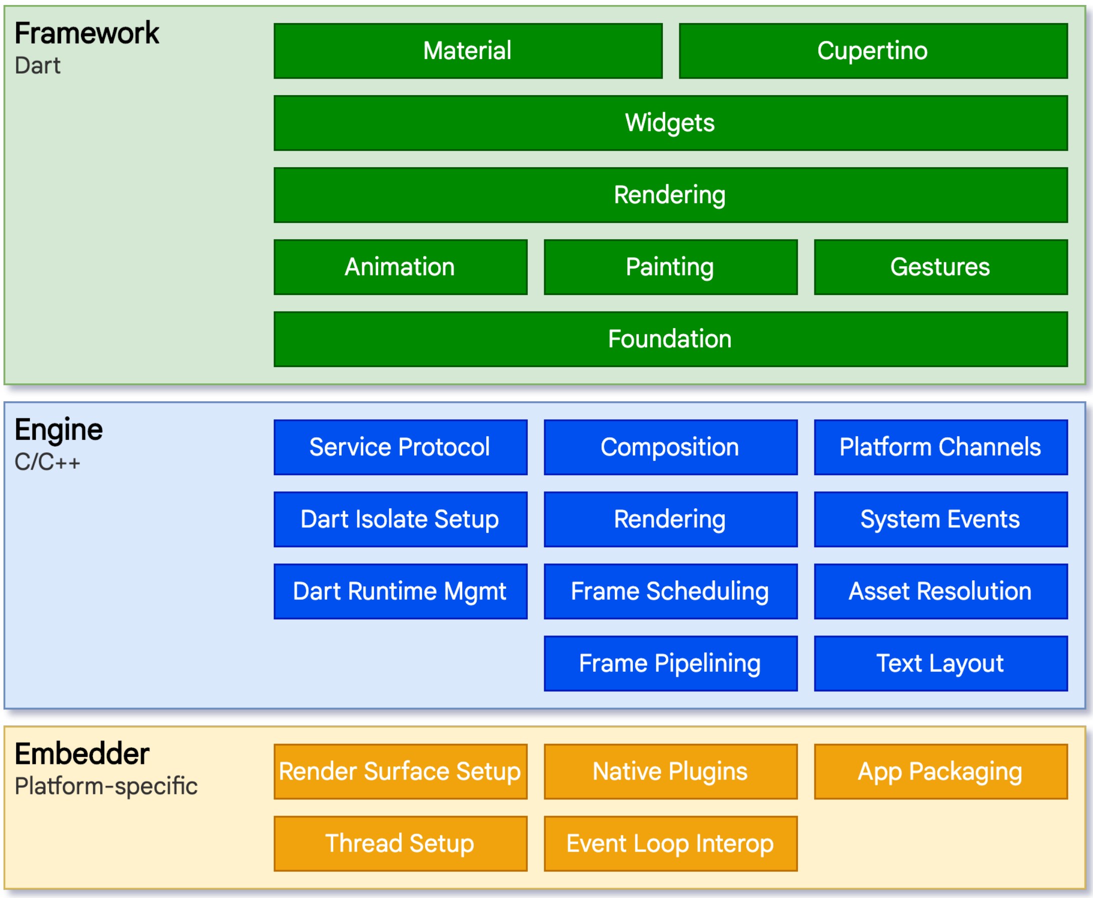


Figure 3.7: Flutter architectural overview - Flutter architectural layers [Flu]

Following coined software engineering principles, the Flutter framework is structured as an onion-like architecture, each layer consisting in a series of independent libraries, each of those depending on the underlying layer. As in classic layered architecture, no layer has the option of accessing data below itself and every part of the framework level is constructed to be optional and replaceable.

From the point of view of the operating system on which a Flutter application would run, the packaging of the application mimics a native one: a platform-specific embedder provides an entry point and it communicates with all the platform-specific services available on that platform: rendering surfaces, accessibility, input, etc. With the help of the embedder, Flutter code can be integrated into an existing system as a module or it can also be the whole content of the application.

The most important part of this architecture and what makes the Flutter framework unique is the Flutter engine. It is written in C++ and it is responsible for rasterizing [12] composite scenes whenever a new frame needs to be painted. On top of that, the engine is exposed to the Flutter framework through "dart:ui" which wraps the underlying C++ code into Dart classes. This library exposes the lowest-level primitives, such as classes for driving input, graphics, and text rendering subsystems.

The building block of this architecture is the Flutter framework. Through this layer, the developers have access to a modern and reactive framework written in the Dart language. This is the block that provides all the necessary components for designing a user interface: foundational classes as well as services such as animations, painting or gestures; a rendering layer that allows the developer to create a tree of renderable objects; hand in hand with the rendering layer is the widget layer: each rendered object in the rendering layer has a corresponding class in the widget layer, the latter allowing the programmer to define the combination of classes to be used or reused (following the paradigm of reactive programming [13]). The Flutter framework also provides support for the Material and Cupertino libraries which offer a wide variety of controls that use the widget layer's composition primitives for implementing the Android or iOS design languages.

As previously described, all the "work" is done within the Flutter framework itself. Thus, in order to be able to do that, the Flutter team created the engine such that it is platform-agnostic and it exposes a stable ABI (Application Binary Interface) that enables the platform embedder to set up and use Flutter. The embedder is the native platform's operating system application that acts as a host for all Flutter elements and acts as the channel between the host operating system and Flutter. When a Flutter application is started, the embedder provides the entry point, initializes the Flutter engine, obtains threads for UI and rastering, and creates a texture that Flutter can write to. The embedder is also responsible for the app lifecycle, including input gestures (such as mouse, keyboard, touch), window sizing, thread management, and platform messages. Flutter also includes platform embedders for Android, iOS, Windows, macOS, and Linux.

Another really important feature of the Flutter architecture is the possibility to

[12]Raster-scan displays are most commonly driven by a frame buffer, a memory that stores the color value for every picture element on the screen and refreshes the display continuously. The process of generating these picture clement values from a geometric description of the image is known as rasterization. [GGSS89]

[13]Reactive programming has been proposed as a viable alternative to the 'Observer' design pattern in developing reactive applications such as graphic user interfaces, animations, and event-based systems. The idea behind reactive programming is to support language-level abstractions for signals – time-changing values that are automatically updated by the language runtime. In reactive programming, programmers specify the functional dependency of a signal on other values in the application, and changes are automatically propagated when it is required.[SM16]

integrate Flutter with other code. Flutter offers a variety of interoperability mechanisms, from accessing code of APIs written in other languages such as Kotlin or Swift or calling a native C-based API to fully embedding native controls in a Flutter app or embedding Flutter into an existing application.

Moreover, if a developer wants to call custom code directly from the platform's native language (in the case of desktop or mobile application), this can be achieved with the help of platform channels. This mechanism works as a communication channel between the Dart code and the platform-specific code of the host application. By encapsulating a name and a codec, one can send and receive messages between Dart and a platform component written in a language like Kotlin or Swift. Data is serialized from a Dart type like Map into a standard format and then deserialized into an equivalent representation in Kotlin (such as HashMap) or Swift (such as Dictionary). This process is shown in Figure 3.8: [Flu]

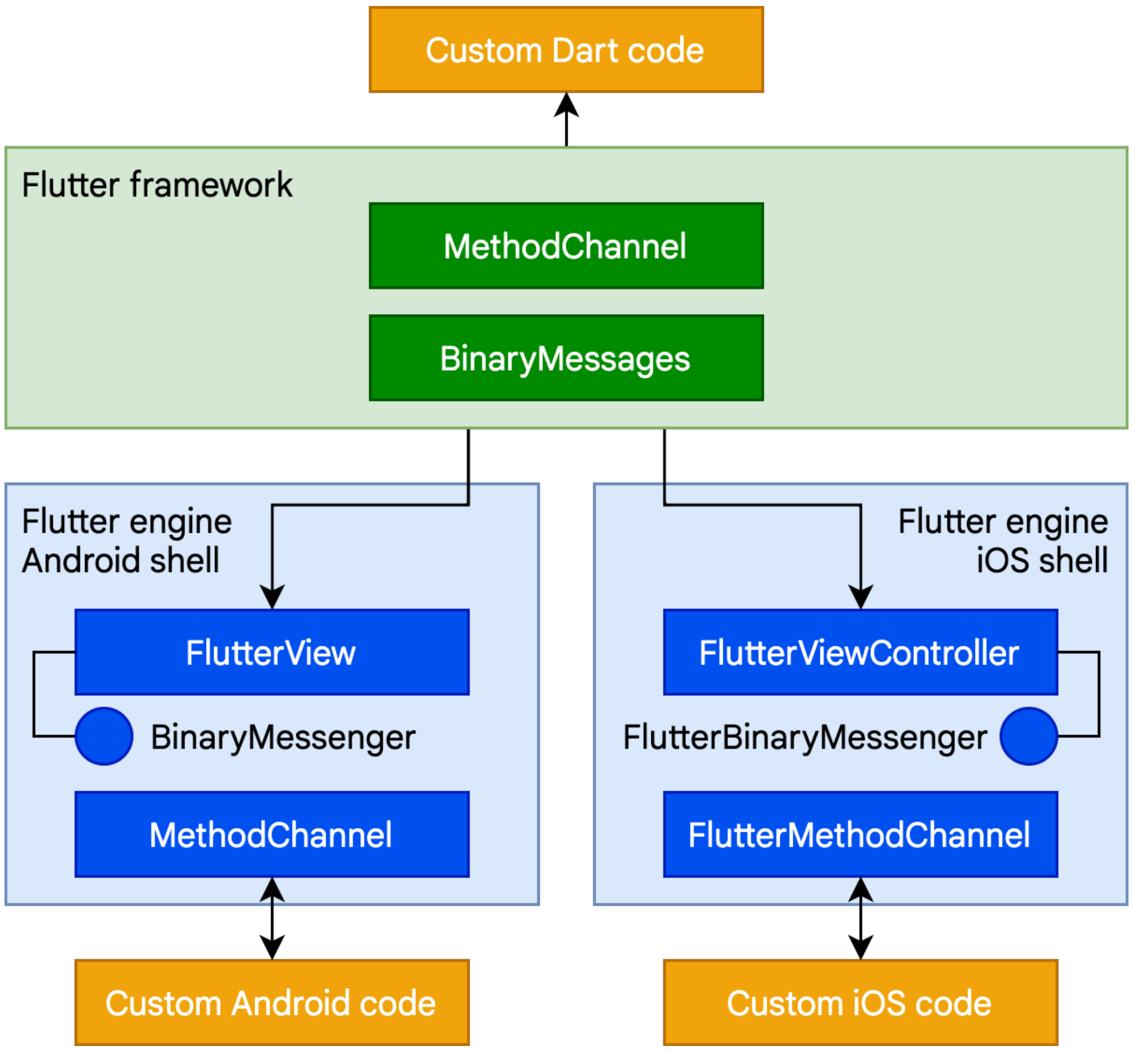


Figure 3.8: Flutter architectural overview - Flutter platform channels [Flu]

3.3.3 Xamarin

Xamarin was launched in 2011 and it was bought by the Microsoft corporation in 2016 and it serves the purpose of building applications for Android, iOS, macOS, and Windows. Xamarin is structured as an open-source cross-platform framework that uses the C# programming language, the .NET framework, and Visual Studio as primary tools for creating the aforementioned applications.

The first difference that is present in Xamarin versus the other hybrid frameworks is the fact that Xamarin does not provide an out-of-the-box way of developing web applications. In order to achieve this, the developer has to operate with Ooui, a small cross-platform UI library that enables the use of web technologies. Also, different from the other frameworks, Xamarin aims to be more of a back-end hybrid framework, rather than a front-end one. On its starting page of documentation, it is advertised that ”you to create native UI on each platform and write business logic in C# that is shared across platforms”, thus suggesting that the main focus of the framework is on having a shared back-end logic, without having to rewrite it for each individual platform.

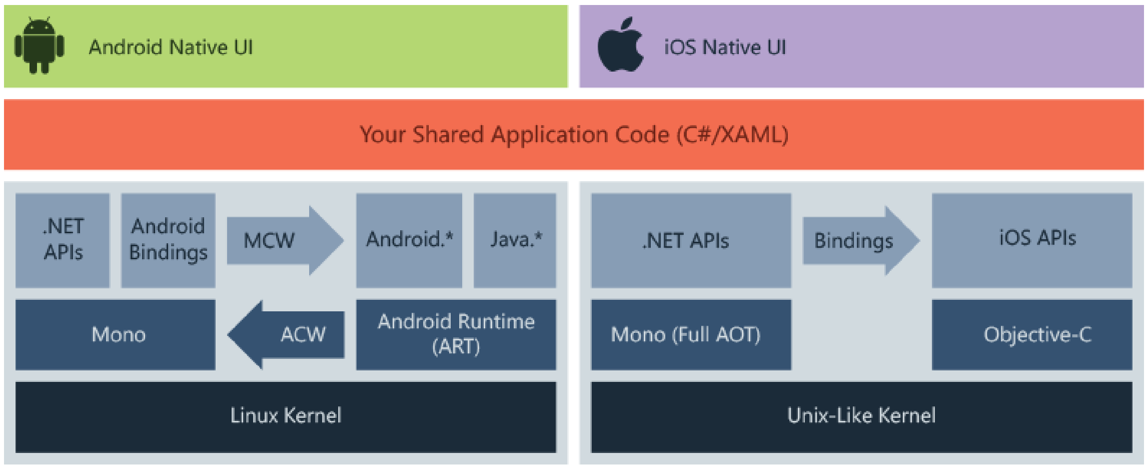


Figure 3.9: Overall architecture of a cross-platform Xamarin application [Jus]

As seen in Figure 3.9, the native UI elements have direct access to the shared C# code within the Xamarin framework. From that point on, the framework will take charge and will do a complete binding depending on the underlying SDK of the native platform. Similar to Flutter, Xamarin facilitates directly invoking Objective-C, Java, C, or C++ libraries with the help of the COM Interop (a technology included in the .NET Framework Common Language Runtime which allows the interaction between the Component Object Model objects and .NET objects).

The process of compiling the Android applications is the following: the C# code is compiled into Intermediate Language which is then Just-in-Time (JIT) compiled to a native assembly when the application launches, Xamarin. Android applications run within the Mono execution environment, side by side with the Android Runtime (ART) virtual machine. Xamarin provides .NET bindings to the Android.* and java.* namespaces. The Mono execution environment calls into these namespaces via Managed Callable Wrappers (MCW) and provides Android Callable Wrappers (ACW) to the ART, allowing both environments to invoke code in each other.

Xamarin.iOS applications are fully Ahead-of-Time (AOT) compiled from C# into native ARM assembly code. Xamarin uses Selectors to expose Objective-C to managed C# and Registrars to expose managed C# code to Objective-C. Selectors and Registrars collectively are called ″bindings″ and allow Objective-C and # to communicate. [Jus]

3.3.4 Cordova

After Adobe corporation acquired the PhoneGap (one of the first hybrid solutions for deploying applications written in HTML, CSS, and JavaScript to mobile devices) project in 2011 and then discontinued in 2020, Apache Cordova was forked as an open-source framework used for building web applications for Android, iOS and Windows, using web technologies.

In order to understand the architecture of Apache Cordova, firstly we have to know what it is composed of. It is structured in three main blocks: the web app, the WebView, and the Cordova Plugins, each having an important role in the process of transforming the HTML, CSS, and JavaScript code into code that is understood by mobile operating systems.

The web app is the ″visible core″ of the framework. Having the same composition as any other plain HTML, CSS, and JavaScript web application starting from the ″index.html″, the web app is the part where the developers will write the code of the application. The main difference is the existence of the crucial ″config.xml″ file which holds important information about the application and specifies parameters about its behavior. The XML file is platform-agnostic (a repeating feature that most of the frameworks want to have) and is based on the W3C's Packaged Web Apps (Widgets) specification [14] and its aim is to specify what core Cordova API features, plugins, and platform-specific settings should the application use.

Cordova applications rely on browser-based WebView(s) within the native mobile platforms for displaying the whole user interface which was built in the web app part of the framework. With the help of API bindings, the framework was able to make the appropriate mappings from the web app to the WebView and then from the WebView to the mobile operating system.

[14] W3C's Packaged Web Apps (Widgets) specification are a set of standardized specifications for packaging format and metadata about packaged apps and widgets. [W3C]

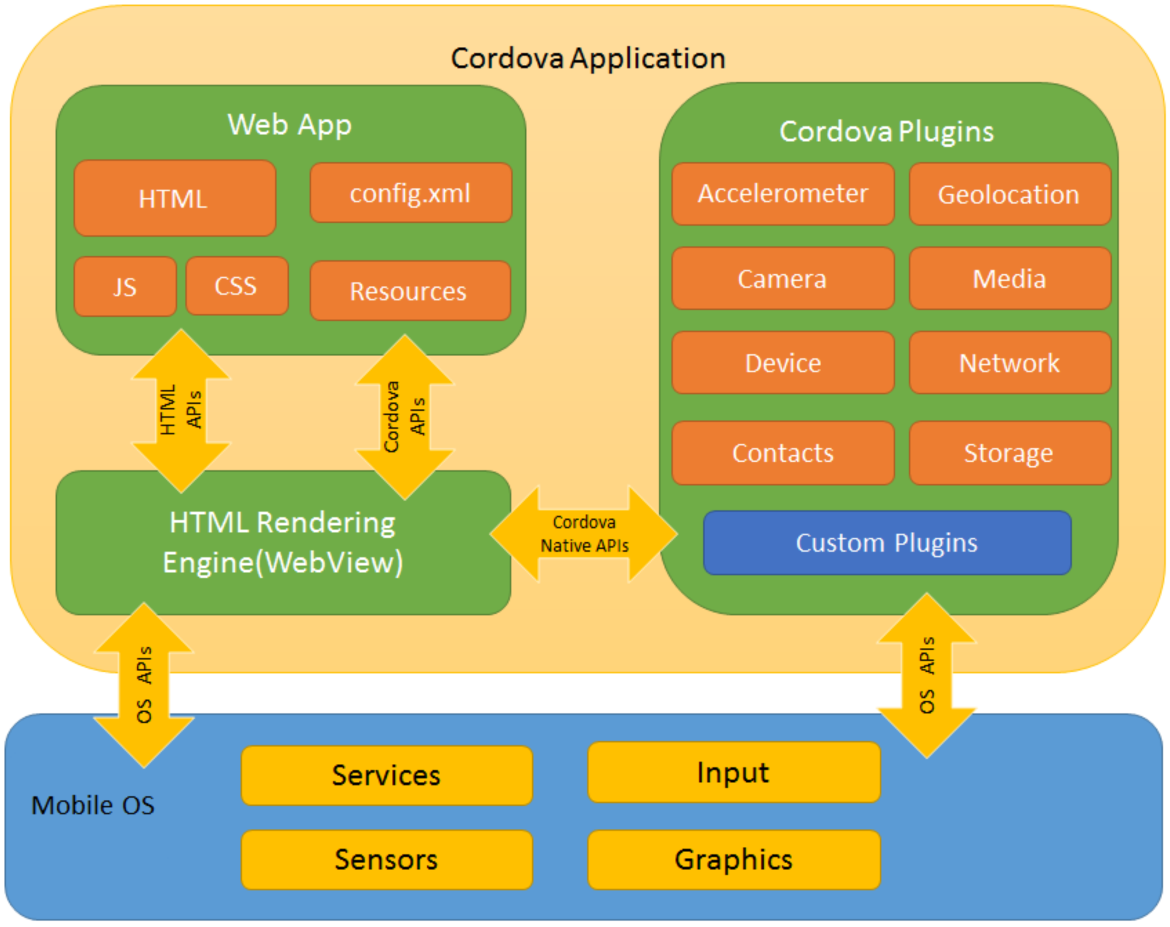


Figure 3.10: High-level view of the Cordova application architecture [Cor]

Probably the main feature of this ecosystem is the Cordova Plugins part. This part provides the interface between the Cordova application and the native components and enables them to communicate with one another as well as providing standard device APIs bindings. Also, the Apache Cordova project possesses a large library of plugins called "Core Plugins". Those plugins have the main purpose of accessing the device services and capabilities, such as battery, location, contacts, or camera. On top of that, Cordova allows for custom creation of plugins in case there is a need for communication between Cordova and some custom native components (when there would not exist a default plugin for such communication).

Cordova is a very powerful framework and it can be integrated within other hybrid frameworks (e.g. Ionic), mainly because of its vast plugin collection and its CLI (command-line interface). The CLI is a high-level tool that allows the developer to focus on building applications for multiple platforms at once without having to worry about the low-level shell scripts. [15] The CLI is in charge of copying common web assets into subdirectories for each mobile platform as well as making the obligatory configuration changes for those and running building scripts for generating the application binaries. [Cor]

3.3.5 Ionic

Ionic framework was released in 2013 as an open-source framework that can leverage web technologies such as HTML, CSS, and JavaScript. The piece that made Ionic popular is the fact that it can integrate other web frameworks on top of itself, such as Angular, React, or Vue.

As seen the Figure 3.11, the overview of the Ionic Framework is pretty simple. A developer can take any of its back-end solutions (e.g. Amazon Web Services, Azure, Firebase) and then connect those to their favorite front-end web technologies, be it Angular or React, Vue or Stencil, or even plain JavaScript. Depending on the chosen front-end framework, the programmer can use the Ionic CLI and generate an Ionic application based on the choice. After that, a native API will be needed for the translation of web Ionic elements to native elements. There are 2 choices: Capacitor, the cross-platform native runtime created by the Ionic team, or Apache Cordova, or rather, the plugins library offered by Cordova, described in the previous subsection.

[15] A shell script is a small program run by the Unix shell and it usually involves file manipulation, write/read accessing rights, and printing text.

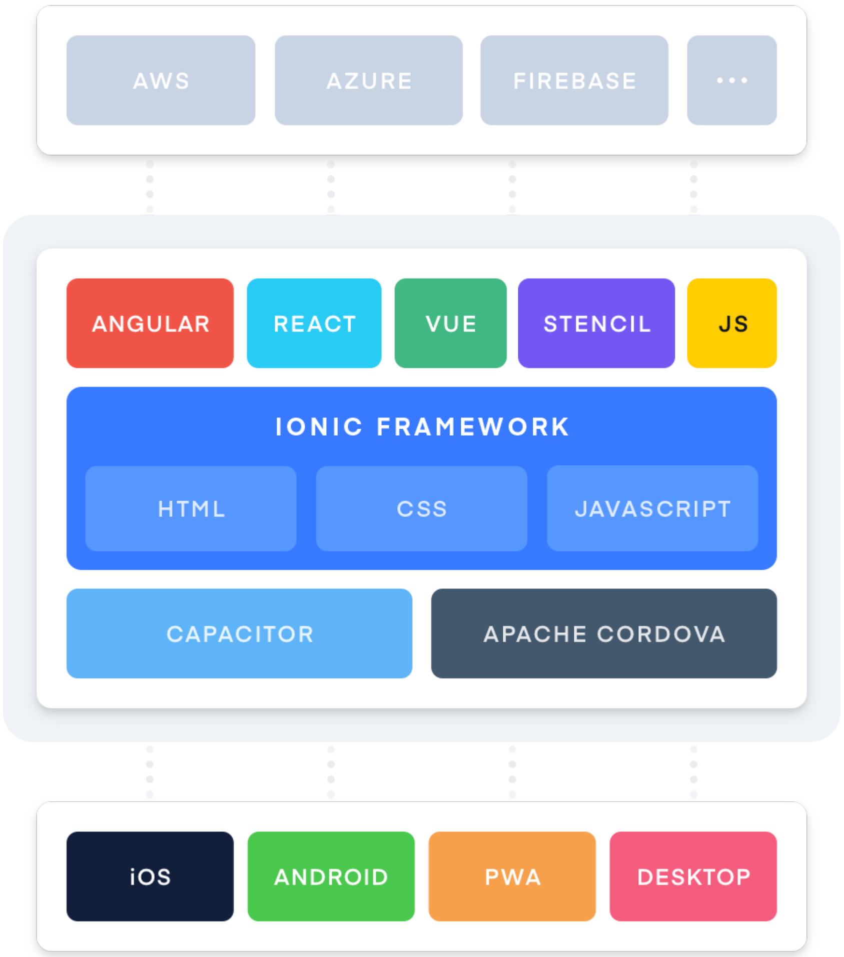


Figure 3.11: High-level view of the Ionic application architecture. [And]

3.4 Important Factors when Choosing a Framework

The world of software development is a living organism and it evolves at a very rapid pace, from year to year. True to nature's principles, whoever is not strong enough to grow and improve itself, is left behind and the relatively new hybrid frameworks are no exception to this rule. The following sections provide our exploration of some important factors one has to think of when choosing the framework technology for their business.

3.4.1 Community Size

It is not a secret that many tools and products that developers and even regular individuals use on a day-to-day basis are of open-source provenience. Considering the way that modern software is written now, namely numerous usages of external libraries and accessing third-party services, the community behind a software is crucial to the lifespan of a given piece of software.

Across this thesis, we have accentuated the idea that nothing is static and everything is changing and we do it again. Since almost nothing is built from scratch now, the future quality of the software one wishes to build in present is directly dependent on the involvement of the community who maintains the libraries that were used in the process of creating that software program. The whole purpose of the open-source software is "to sit on the giant's shoulders" and not reinvent the wheel. These open-source resources also come with the risk of a library being abandoned (programmers mostly use the term 'deprecated') in favor of a new alternative that is expected to be thriving in future environments. It is of high importance for thoughtful business owners as well as ambitious programmers to make some research on the communities of the frameworks they would like to choose and make an educated guess on whether those will be relevant in the future. Migrating from one technology to another has a high cost in terms of spent resources and might lead to negative results for a business if it reaches a point which that is the only viable solution for the survival of their product.

Reusing the same statistic used at the beginning of the section 'The hybrid solution and popular hybrid frameworks' from this chapter (Figure 3.5), we can clearly see that the most popular framework at the time of writing this thesis is Flutter, very close to React Native, both having close to the same share of developers who are using those. At the same time, we can see that Flutter was in an ascending trend of popularity from 2019 to 2021, while React Native stagnated at the same levels and Ionic, Cordova, and Xamarin lost a significant number of developers.

We have seen in the previous sections that most of the frameworks are open-source applications, thus all of them have a public Github page with the code they are maintaining. Our recommendation is that before sticking to a definite choice, a brief look over the Github activity of a given framework should be a mandatory step before proceeding with the development in that given framework.

3.4.2 Ease of Development

This criterion is related to the quality of the documentation and the learning curve for using a certain framework.

The quality of the documentation can be evaluated by taking into consideration some of the following points: there are enough code examples for the presented API, there exist links to similar problems in case of issues/errors or there is a multitude of user comments on popular threads of the framework's most common questions (this is also strongly related to the previous factor of community size, finding answers on sites such as Stack Overflow or the Github (issues) page for a given framework).

By learning-curve one can understand the subjective progress of a developer during his first examination of a framework. Intuitive concepts bearing resemblance to already known paradigms allow for fast success. This can have a significant impact on how fast new colleagues can be trained and how much additional, framework-specific knowledge a developer needs to acquire. [HHM12]

Considering the above-stated reasons, my recommendation would be that in the process of deciding on picking a framework for a given project, the project leader should at least briefly take a look at the framework's documentation. He should look for framework-specific features as well as shared programming/architectural paradigms across multiple technologies, such that he can make a proper decision that might fit best the project at hand.

For instance, taking Ionic as an example, which supports 3 different frameworks as underlying frameworks, namely Angular, React, and Vue, has higher ease of development than let us say, Flutter, which has its own programming language and it is more likely adjacent to a mobile native programming paradigm, rather than a web paradigm, in the case of the former.

3.4.3 User Interface Persistence

While the general appearance of an app can be influenced during development, it does matter whether a framework inherently supports a native look and feel or whether its user interface looks and behaves like a website. [HHM12]

Living in times where people's attention is the most valuable thing businesses want to target, one can agree that the user interface of a given software product has increased significantly in importance. Due to this factor, frameworks have to support the specific usages and UI components lifecycles that might occur while using a native app. For instance, applications are used for a short amount of time, then those are interrupted by another action (e.g. a phone call), and then the app is resumed again. When returning to the app, would be of negative impact if the user would had to fill a form again or redo all the steps he previously completed within that specific use case.

Thus, before making a choice for the framework they intend to choose for a given business model, the programmers should research if the framework at choice has any related issues with possible use cases that might appear while using the app in regards to user interface elements, customization or application interrupts.

3.4.4 Closing Thoughts

There are many other factors that can be taken into account when making this kind of decision. If we were to take this decision, these three factors would be the first on our checklist. Depending on each individual case, there might be other factors with other priorities weighted to each of those.

Chapter 4

Practical application

As years pass at a rapid pace, almost all things that we can think of in the real world have a mapping onto the Internet. In the past when a business had an online presence, it was seen as a surprise, as an outstanding move that would generate a lot of capital. In recent times, if a business does not have an online presence, it is seen as an anomaly and a certain way to lose potential customers and funds.

For the purpose of the practical work of this thesis, it is presented in the following sections how a hybrid framework is useful and might be the key technology for many businesses for developing their companies, mainly because hybrid frameworks offer the vital flexibility many small to medium enterprises strive for in the early stages of their lifespan.

4.1 Short Description

In this practical chapter of this thesis, we chose to build a general template for how a business might develop an online shop for the products they wish to sell in the virtual environment using a hybrid framework. For this purpose, we selected Ionic without any particular reasons, mostly because of our familiarity with this framework.

For this application, the main use case is selling pizza(s) (or any other product/service a business would want to sell). The complexity of this project is not necessarily in the logic and heavy computation that are taking place, but rather in the design and the architecture of the application, as well as finding fast and reliable ways of producing a high-quality application using as few resources as possible.

The following sections contain a presentation of the architecture of the application, main use cases, the design in terms of responsiveness (web vs mobile users), the technologies used in the process + (software engineering) design patterns, and helping tools that played an important role for finishing this application.

4.2 Implementation and Design

4.2.1 Architecture

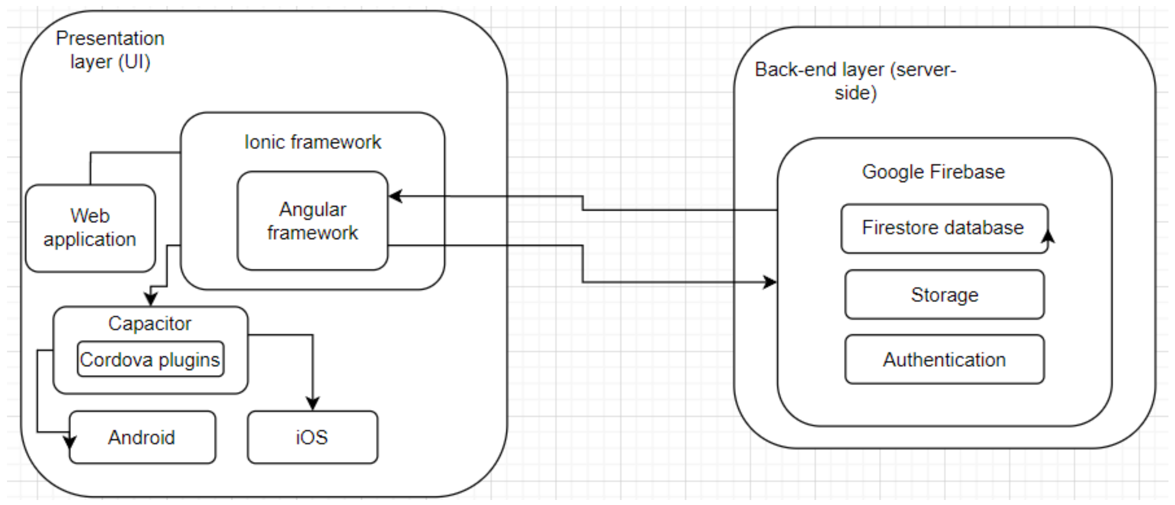


Figure 4.1: General architecture of the application

Studying Figure 4.1, we can see the general architecture of the application. It consists of two main parts: the presentation layer (UI or client-side) and the back-end layer(server-side). For this application, we chose to use Firebase as a solution for back-end operations, thus it will act as an API in this application, all the additional logic being done in the Angular services within the Ionic application (small adaptation to classical client-server applications in which the front-end is only responsible for showing data, without doing any mutations to it). More details about the Google Firebase solution are provided in the ”Technologies” section.

The front-end communicates with the back-end through asynchronous API calls to the Google Firebase platform. The latter provides a solution for authentication and managing the login status of each of the users, as well as a nonsequential database for storing relevant information and a storage service for saving larger data (such as files or photos).

In the front-end, the choice was Ionic which has Angular as the underlying web framework (the web version of the application can be started out of the box because of this property, just by running `ionic serve`, the command which starts the ionic application). For the user interface, we used Ionic components and then used Angular-specific design patterns and templates for building the front-end logic for easy maintenance and changes to the code.

In order to be able to run the Ionic/Angular written code on mobile devices, we needed the help of the Capacitor plugins API, the native solution proposed by the Ionic team for accessing mobile-specific services and hardware (such as storage or camera). Also, in the cases in which a plugin was not available in Capacitor, it also had the capability to integrate singular Cordova plugins for fitting the purpose.

After implementing the plugins with Capacitor/Cordova, we used the Capacitor CLI for generating the code for the mobile platforms (Android/iOS) and then just run it on their specific platform (on an emulator or physical device).

4.2.2 Use Cases

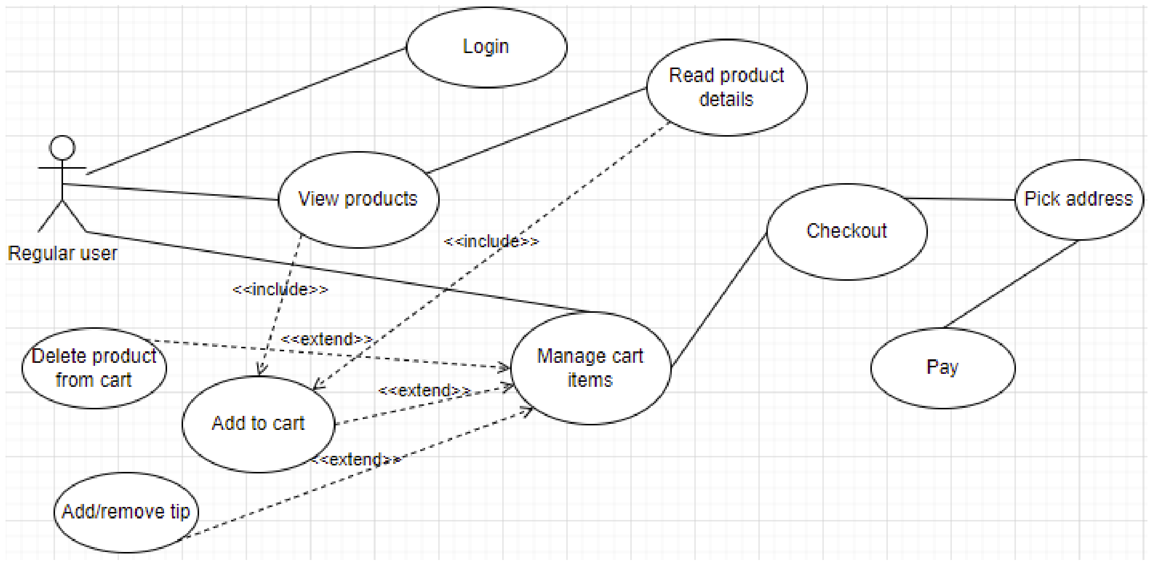


Figure 4.2: Use cases of the application

As shown in the Figure 4.2, there are multiple use cases for this application.

The main use case can be seen by observing the following path: the actor views all the available products, adds a few of them to the cart, he adds or removes a tip for the raider (the person who is in charge of delivering the goods) and then proceeds to checkout. After this, the actor selects an address and a method of payment and then the delivery information will be sent to the business provider.

Other use cases are adjacent to the main one and are pretty logical. An actor can see the details of a product if so wishes or remove products from the cart, as well as increase/decrease the tip for the delivery man. There are some functionalities that are hidden and can be accessed by authenticating into the system.

If a user logs into the system, he becomes an admin (as seen in the Figure 4.3. with the extended use case), provided he has valid credentials to do so (the credentials are given by the system administrator - a person who has access to Firebase authentication platform and can set/generate new credentials). An admin has the ability to add a new product, where he can upload an existing photo of the product or take a new picture of it and enter the details, as well as delete or edit existing ones. With this distinction between a normal user/an admin, the business can require the

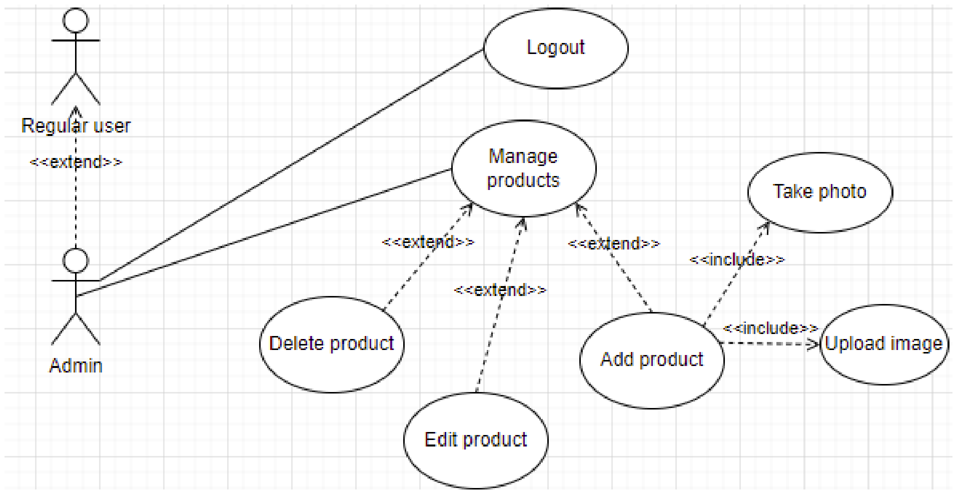


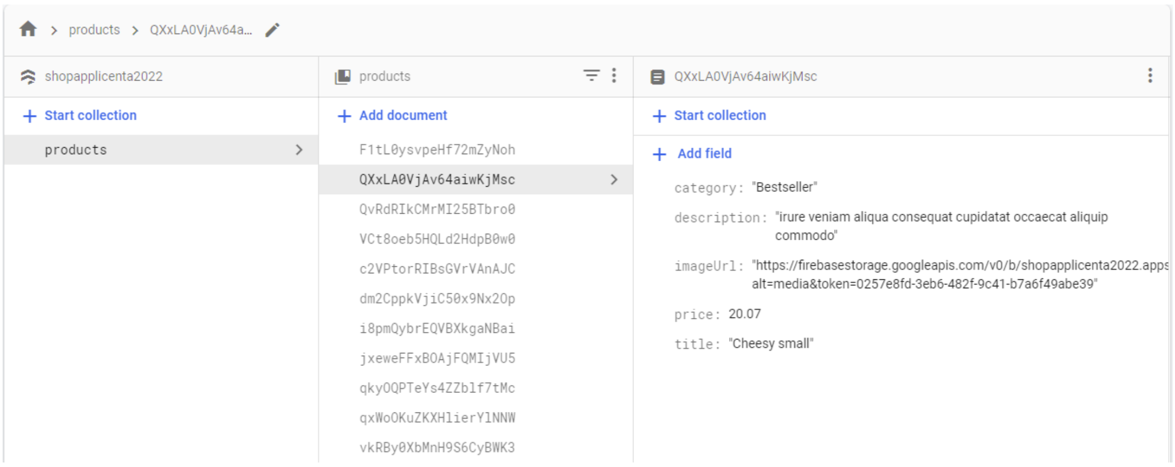
Figure 4.3: Use cases of the application

developers to add extra management features for the admin type, such that the developers will not be needed for normal administrative changes within the business since the admins will be able to change them within the application.

4.2.3 Used Technologies

Google Firebase Database

Firebase is a platform developed by Google and used in the creation of web and mobile applications. The platform offers services for the back-end tasks of most applications, such as authentication, database, storage, and hosting, as well as for analytics and other important metrics.



```
products
```

```
F1tL0ysvpeHf72mZyNoh
QXxLA0VjAv64aiwKjMsc
QvRdRIkCMrMI25BTbro0
VCt8oeb5HQLd2HdpB0w0
c2VPtorRIBsGVrVAnAJC
dm2CppkVjiC50x9Nx2Op
i8pmQybrEQVBXkgaNBai
jxeweFFxBOAjFQMIjVU5
qky0QPTeYs4ZZblf7tMc
qxWoOKuZKXHlierYlNNW
vkRBy0XbMnH9S6CyBWK3
```

```
category: "Bestseller"
description: "irure veniam aliqua consequat cupidatat occaecat aliquip
             commodo"
imageUrl: "https://firebasestorage.googleapis.com/v0/b/shopapplicenta2022.apps
          alt=media&token=0257e8fd-3eb6-482f-9c41-b7a6f49abe39"
price: 20.07
title: "Cheesy small"
```

Figure 4.4: Products collection in Firebase database

In Figure 4.4 one can see the user interface of the Firestore Database. It is designed as a NoSQL Database, which has a lot of optimizations and different attributes compared to the traditional relational databases. The main perk is that allows flexible rules on how the data should be structured in order to achieve security and flexibility.

As seen in the above figure, Firebase is a database that is stored as JSON objects, making it easier to use than some SQL databases because of the way that data can be handled as a tree. The ”entities” which we know in traditional SQL are ”collections” (in our case the collection ”products”) which have many ”documents” (or table entries in the relational databases). Usually, the name of the document is the unique id that defines the object. In the end, each document has its fields, the characteristic attributes which hold the information about a given object. Each field can be of type: number, boolean, map, array, null, timestamp, geopoint, or reference. Also, a very important feature of a document is the ability to create another ”collection” within it, thus allowing complex linking between the database objects, depending on the problem domain.

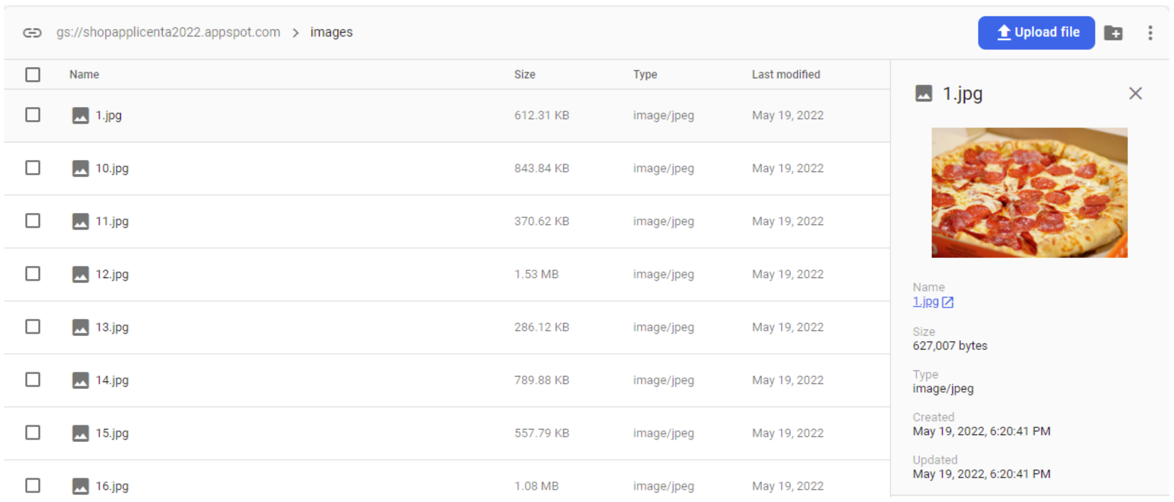
In this application, we have only one ”collection”, namely ’products’. It has 5 fields: category(string), description(string), and imageUrl(string) - this is an url with a reference for the Firebase Storage, the location where the uploaded images are hold within the application, price(number) and title(string).

```
import {doc, Firestore, collection, setDoc}
    from '@angular/fire/firestore';
private firestore: Firestore;
const PRODUCTS_COLLECTION_KEY = 'products';
async addProduct(newProduct) {
    const productsReference = collection(this.firestore,
    PRODUCTS_COLLECTION_KEY);

    await setDoc(doc(productsReference), newProduct);
}
```

The interface on the front-end is quite simple as well, as seen in the above code snippet: provided we have a JSON object in the ’newProduct’ variable, we just need to create a reference to the collection we wish to add the new object, we add a new document to it by calling setDoc() and giving as parameters the reference to the collection and the new object. The result is a ’Promise’, thus we have to (a)wait for it (or implement some logic for the time which is spent while making this call).

Google Firebase Storage

From their official site, the Google Cloud Storage for Firebase is advertised as a powerful, simple, and cost-effective object storage for scaling. Its main purpose is to serve the developers who need to store and handle user-generated content, such as files, photos, or videos.


gs://shopapplicenta2022.appspot.com > images

Name	Size	Type	Last modified
1.jpg	612.31 KB	image/jpeg	May 19, 2022
10.jpg	843.84 KB	image/jpeg	May 19, 2022
11.jpg	370.62 KB	image/jpeg	May 19, 2022
12.jpg	1.53 MB	image/jpeg	May 19, 2022
13.jpg	286.12 KB	image/jpeg	May 19, 2022
14.jpg	789.88 KB	image/jpeg	May 19, 2022
15.jpg	557.79 KB	image/jpeg	May 19, 2022
16.jpg	1.08 MB	image/jpeg	May 19, 2022

1.jpg

Name
1.jpg

Size
627,007 bytes

Type
image/jpeg

Created
May 19, 2022, 6:20:41 PM

Updated
May 19, 2022, 6:20:41 PM

Figure 4.5: Google Firebase Cloud Storage User Interface

As seen in the Figure 4.5 the storage is nothing spectacular. We used it for storing the photos which the admin can upload when he adds a new product to the store and the storage provides back a link to the newly added item. The main advantage of this storage is that it is bundled together with the other services, making it very convenient for developers to use it since most applications also have to provide a way to store large data. In addition, Google Firebase Cloud Storage also provides rules which the programmers can set in order to guard the data and define who has access to it.

As there does not exist a direct link between the storage and the database within the Firebase services, we had to make this connection in one of the services within the application.

```
const FIRESTORE_BASE_PATH = '/images';
private firestore: AngularFireDatabase;
private firebaseStorage: AngularFireStorage;

uploadWithUserImage(file: File) {
    const filePath = FIRESTORE_BASE_PATH + '/' + file.name;
    const storageReference = this.firebaseStorage
        .ref(filePath);
    const uploadTask = this.firebaseStorage
```

```
        .upload(filePath, file);

    this.sharedService.uploadProgress$ = uploadTask
        .percentageChanges();

    uploadTask.snapshotChanges().pipe(
      finalize(() => {
        storageReference.getDownloadURL()
            .subscribe(downloadUrl => {
              this.imageToUploadUrl = downloadUrl;
              const newProduct = this.buildNewProduct();
              this.productService.addProduct(newProduct);
              this.router.navigateByUrl('/products',
                {replaceUrl: true});
            });
      })
    ).subscribe();
}
```

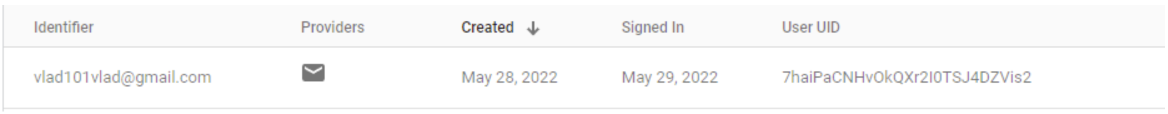
As seen in the above code snippet, provided we have a reference to the storage with a path chosen by us, as well as one to the Firebase database, we can make the synchronization happen. We have implemented this mechanism in the following way: provided that the `uploadWithUserImage()` method receives a file (which is also converted within the application from the base64 information of an image), we can create an upload task to the storage reference with the provided file.

Having the upload task, we can call `snapshotChanges()` on it which will give us back an `Observable` on the item which will be uploaded in the near future. With the help of the `pipe()` method, we can stub the observable and make use of the `rxjs` function `finalize()` which will be called right after the upload task is finished, thus allowing us to run the code which we write within the aforementioned `finalize()` method.

Being assured that the upload is finished, we will make use of the previously defined reference to the storage with the path defined by use to get the download URL of the object which resides there. Having now the download URL, all we have to do is to include the newly created product in the Firebase database, as seen in the previous subsection. In this way, we can be certain that there are no inconsistencies between the data which are persisted on the cloud.

Google Firebase Authentication

Google Firebase Authentication is one of the most powerful services that the Firebase bundle has to offer. It provides an out-of-the-box solution for authentication. It is well known within the programming community that security tasks are quite complex and require a lot of time to properly set.



Identifier	Providers	Created ↓	Signed In	User UID
vlad101vlad@gmail.com	✉	May 28, 2022	May 29, 2022	7haiPaCNHvOkQXr2I0TSJ4DZVis2

Figure 4.6: Example of simple Firebase authentication dashboard

The best thing about Google Firebase Authentication is that it provides a lot of authentication methods with the most popular social media accounts as well as any authentication-related processes, such as: email address verification, password reset, email address change, or SMS verification.

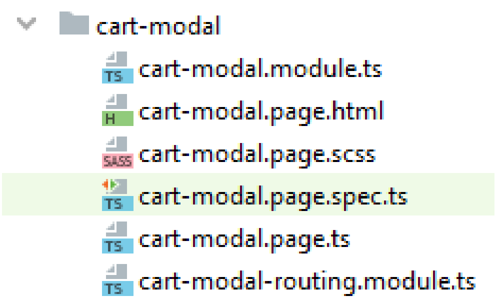
```
import {setPersistence} from '@firebase/auth';
import {Auth, browserLocalPersistence,
    signInWithEmailAndPassword} from '@angular/fire/auth';
private firebaseAuth: Auth;

setLocalAuthenticationPersistance(): Promise<any> {
    return setPersistence(this.firebaseAuth,
        browserLocalPersistence);
}

async loginAdmin({email, password}): Promise<any> {
    return this.setLocalAuthenticationPersistance()
        .then(_ => signInWithEmailAndPassword(
                this.firebaseAuth,
                email,
                password
            ));
}
```

Another very important feature of the authentication interface is the fact that we can set the expiration time of the login of a certain user. One of the requirements of Firebase applications is that you have to be 'connected' to it (or rather register the application within the Firebase console), thus allowing the Firebase to track the state of user authentications. Having this property, is very easy and convenient for developers (and users as well) since they do not have to store the "authentication tokens" in local storage or cookies. As seen in the above code snippet, having an instance of the Firebase Authentication, we set the persistence of it to be 'local', so the user will still be logged in, even if he closes the application. After that, we have to do a simple call to Firebase Authentication to `signInWithEmailAndPassword()` and if all the information is valid, the user will be logged into the application.

Angular

From their site, Angular is a development platform, build on TypeScript [1] and it includes a component-based framework for building scalable web applications, and a collection of well-integrated libraries that cover a wide variety of features, such as routing, forms management, client-server communication, and many more. [Ang]

We have chosen Angular for this application mainly because of reasons of familiarity with the framework as well as because of the possibilities and structure it gives to a project.



```
cart-modal
    cart-modal.module.ts
    cart-modal.page.html
    cart-modal.page.scss
    cart-modal.page.spec.ts
    cart-modal.page.ts
    cart-modal-routing.module.ts
```

Figure 4.7: Angular file structure

As seen in Figure 4.7, Angular has a very good solution for the separation of concerns and logic of building a web application. The building block of the angular framework is the component. It will be rendered in the DOM as any other HTML tag, but the difference here is that the programmer has the ability to build this tag. Any component contains an html (template), a scss (stylesheet), and a ts class, making it very easy for the developer to organize and maintain the code, any major change being effortless on the programmer side. Other than components, Angular also supports modules, further increasing the capability of organizing the code. In addition to that, each module has a routing file where all the routes can be declared and secured based on various parameters.

```
<ion-item lines="none">
  <ion-input
```

[1] Typescript is (always intuitive from the name) a programming language that was developed with the purpose to offer a typed version of JavaScript. It is an open-source project developed and maintained by Microsoft and it is compiled in JavaScript.

```
        [(ngModel)]="password"
        [type]="showPassword ? 'text' : 'password'"
        placeholder="Password"
      ></ion-input>
      <p class="pw-toggle" slot="end"
        (click)="showPassword = !showPassword">
        {{ showPassword ? 'HIDE' : 'SHOW' }}
      </p>
    </ion-item>
```

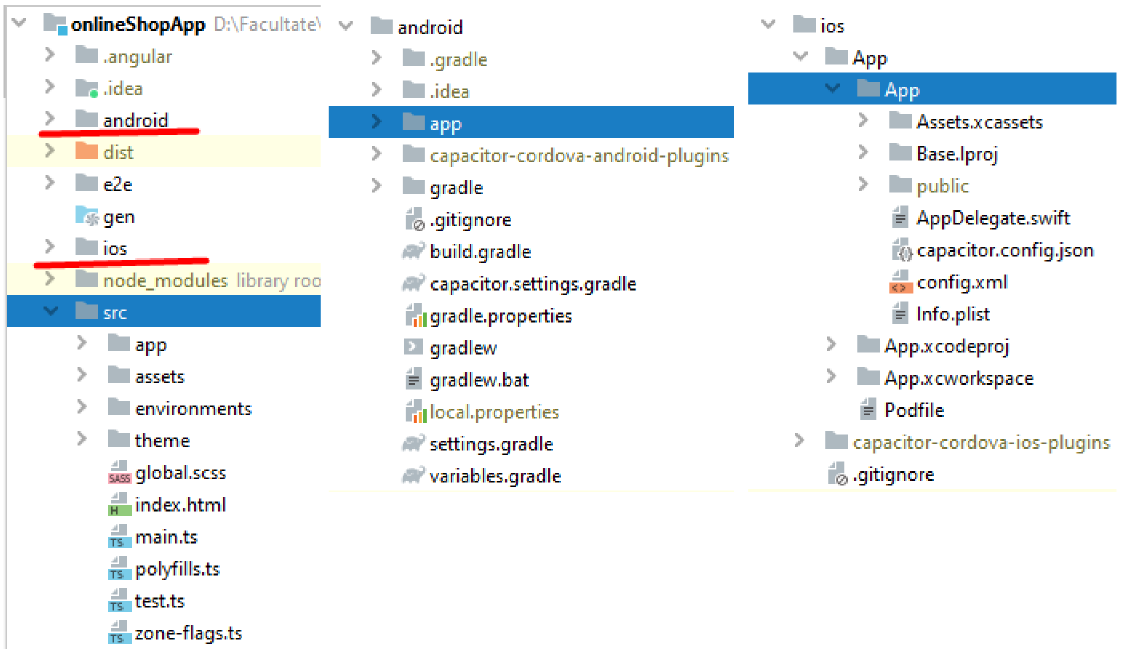
Also, a feature that cannot be omitted and is present within the Angular framework is template binding and customization through property bindings or directives. In the above code snippet, we have provided an example of how useful these features can be while writing the template of a page. We can see in the case of an input, we can dynamically bind the value that one enters with the `password` attribute through the help of `[(ngModel)]` binding. In addition to that, with the help of the style directive `[type]`, we can dynamically set to show or hide the password (setting the type to `'text'` or `'password'`, respectively) depending on the value of the field `showPassword`, a button which can be clicked and which toggles its value between true and false, depending on the user's choice. What would be a dreaded document reference and then setting those properties through JavaScript functions in a 'js' file (and many other operations for persistence assurance between objects and attributes) in the case of a plain JavaScript project, with the help of the Angular templates, can be obtained in a short and elegant way.

Another important feature of the framework is dependency injection. Dependencies are services or objects that a class needs to perform its function. Contrary to plain JavaScript projects, Angular offers dependency injection, a design pattern in which a class request dependencies from external sources rather than creating them. In the case of Angular, this injection happens to a class upon instantiation. [Ang]. In the case of this application, we have used Angular services for API calls to the Firebase back-end as well as for accessing shared computations/methods between multiple-page or components.

Ionic Capacitor

The Capacitor is a cross-platform runtime created by the Ionic team that allows one to target different types of native platforms like iOS, Android, or the desktop (using Electron), as well as the web. [Day] Its main purpose is, as many other hybrid bridge solutions, to facilitate the mantra of "write once, run everywhere"; basically allow code written within the Ionic stack to be run on the aforementioned platforms, without having to set any special parameters or make any changes to the code.

In order to enable the Capacitor within the Ionic application, one has to enter the following command: `ionic integrations enable capacitor`. After that, the developer has to initialize the capacitor with information about the application that it will run on with: `npx cap init [appName] [appId]`, where `appName` is the name of the application and `appId` is the domain identifier of the app (e.g. ro.ubbcluj.app). This will generate the `capacitor.config.json` file which holds information about the Capacitor configuration. Being already present within the application, the programmer has to use the Capacitor in order to add other platforms to the project, using the command: `npx cap add ios npx cap add android`. This will generate two real iOS and android application projects within the main Ionic application, as seen in the Figure 4.8.



```
onlineShopApp D:\Facultate\
  .angular
  .idea
  android
  dist
  e2e
  gen
  ios
  node_modules library roo
  src
    app
    assets
    environments
    theme
    global.scss
    index.html
    main.ts
    polyfills.ts
    test.ts
    zone-flags.ts
```

```
android
  .gradle
  .idea
  app
  capacitor-cordova-android-plugins
  gradle
  .gitignore
  build.gradle
  capacitor.settings.gradle
  gradle.properties
  gradlew
  gradlew.bat
  local.properties
  settings.gradle
  variables.gradle
```

```
ios
  App
    App
      Assets.xcassets
      Base.lproj
      public
      AppDelegate.swift
      capacitor.config.json
      config.xml
      Info.plist
    App.xcodeproj
    App.xcworkspace
    Podfile
  capacitor-cordova-ios-plugins
  .gitignore
```

Figure 4.8: Ionic Capacitor snippet with android and iOS subprojects

Thus, in order to use the camera, the internal storage or access the file system for each of the platforms the code will end up being run, we had to use the imports that come from the Capacitor.

```
import {Storage} from '@capacitor/storage';
const CART_KEY = 'active-cart';

removeProduct(toRemoveProduct: ProductDTO): Promise<any> {
    return Storage.get({key: CART_KEY}).then((response) => {
      const cartDto = CartService
        .mapCartDtoStringyfiedToCartDTO(response.value);
        cartDto.products = cartDto.products
          .filter((cartProduct) =>
          toRemoveProduct.id !== cartProduct.id);

        this.cartItemCount
          .next(this.computeCartNumberOfItems(
              cartDto.products));

        return Storage.set({key: CART_KEY,
          value: JSON.stringify(cartDto)});
    });
}
```

For the cart in which the customers would add their choices of the products they wish to buy, we have chosen to implement it in the internal storage of the platform the application would be run on (the reasoning is that if they wish to leave the site/app, they would not have to add again all the products; database storage would not make much sense, since users might find inconvenient to create an account for ordering a product). We can see in the above code snippet the way is written in the code. In this example, the user wishes to remove a product from the cart. Provided we have a reference to the product they wish to remove, we get the cart from the internal storage of the platform and we remove it by filtering the current products from the cart such that their id(s) is different from the one we wish to remove. After the filtering is done, we set the new list of products to the key which is set for the cart within the internal storage.

```
import {Camera, CameraResultType, CameraSource, Photo}
    from '@capacitor/camera';

currentImageToBeUploaded: Photo = null;

async changeImage(cameraSource: CameraSource) {
    const image = await Camera.getPhoto({
      quality: 90,
      allowEditing: false,
      resultType: CameraResultType.Base64,
      source: cameraSource
    });
    if (image) {
      this.imageSourceWasChanged
        .next(this.b64toSrcFormat(image.base64String,
            image.format));

      this.currentImageToBeUploaded = image;
    }
}
```

In the second example which can be seen in the above code snippet, we can see the implementation of using the camera/file system for uploading an image to the application. With the help of Capacitor, all the information we need to provide to the `Camera.getPhoto()` is the source we want the Capacitor to take the image from (platform camera/ file system), in what format we want the result (base64, URI, blob, etc.), the quality of the image and whether it can be edited or not.

4.2.4 User Interface and Responsiveness

In the times when the average attention span of a person is getting shorter and shorter, the visual elements of a software application are very important to the overall impression of that application. In order to retain the users, the application has to have a pleasant user interface and an intuitive user experience and the Ionic framework has the necessary UI components to facilitate this goal.

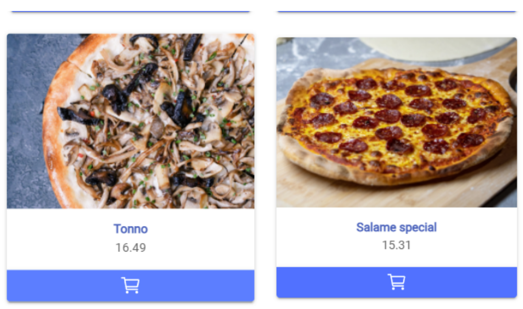
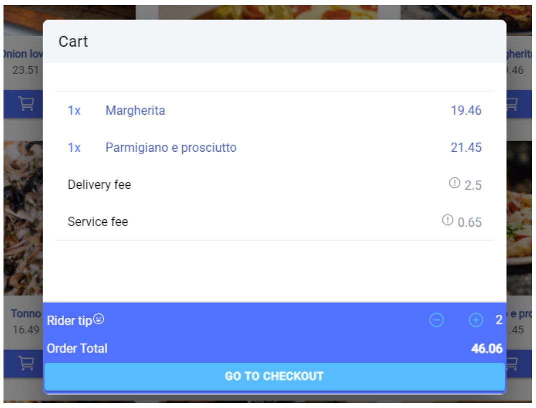


Figure 4.9: Ionic cards with image and button


Cart

1x	Margherita	19.46
1x	Parmigiano e prosciutto	21.45
Delivery fee		2.5
Service fee		0.65
Rider tip		2
Order Total		46.06

GO TO CHECKOUT

Figure 4.10: Ionic modal

As seen in the Figure 4.9 and Figure 4.10, for the basic customization we tried, the result is decent and any user will have an easy time using the application. The elements used in the Figure 4.9 are grids, rows, columns, rows, buttons, and icons. Each of the aforementioned elements can be customizsed by normal css as well as by accessing the in-built attributes that were exposed by the team who created the UI components. In the Figure 4.10 we can observe how an Ionic modal looks over the main page, as well as how other elements can be used for showing a view, such as `ion-header; ion-footer; ion-toolbar`, etc.

(Design) Responsiveness

The term 'responsiveness' has many meanings in the software world, most of the time referring to a specific ability of a system or a functional unit to complete an assigned task within a given time. Even though the responsiveness of a certain hybrid application compared to a native application is important and should be studied, it was not included in the points of focus for this practical part.

One of the parts we focused on while building the application was "Responsive (Web) Design". This term is related to the ability of a front-end application to adapt its design/user interface depending on the type of device it is opened on. Even though the design responsiveness can be taken into account depending on multiple factors such as CPU power, RAM memory, display resolution, and so on, we only focused on the responsiveness depending upon the screen size. Our presentation on responsive design with regards to the screen size can be obtained, as well as show some examples in the following rows.

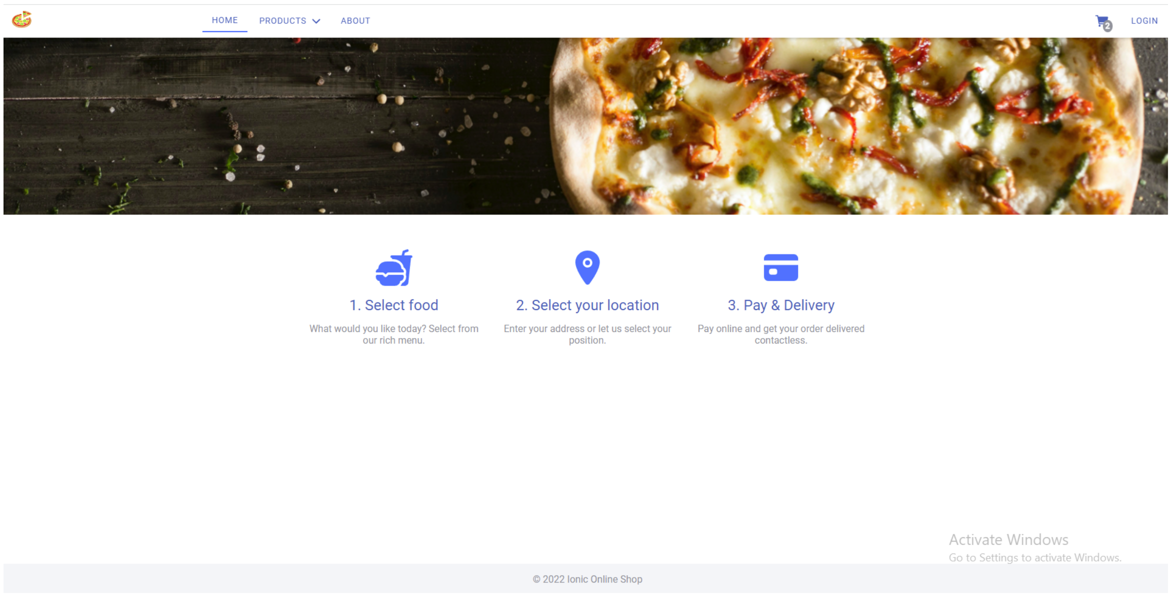

HOME
PRODUCTS
ABOUT
LOGIN
1. Select food
What would you like today? Select from our rich menu.
2. Select your location
Enter your address or let us select your position.
3. Pay & Delivery
Pay online and get your order delivered contactless.
Activate Windows
Go to Settings to activate Windows.
© 2022 Ionic Online Shop

Figure 4.11: Browser view of the home page

There are some notable differences between the figures 4.10. and 4.11, even though the content is the same. Those consist in how the `ion-grid` is computed depending on the screen min/max-width. Also, we can spot that the header has different elements based on the same width property, and in the case of the smaller screen, some of the elements of the header are transferred into a side menu which can be slid to the right. Also on the smaller screen, the grid has a max of one column per row, as opposed to three items per row in the case of the large screen. All of those was possible with the help of css `@media` rule as well as Angular's structural directives.

```
@media (min-width: 768px) {
  .mobile-header {
    display: none;
  }

  .header {
    display: flex;
    background: var(--ion-toolbar-background);
    padding-left: 40px;
    padding-right: 40px;
  }

  :host {
    box-shadow: 0px 1px 6px 0px rgb(0 0 0 / 24%);
    z-index:1;
  }
}

@media (max-width: 768px) {
  .mobile-header {
    display: block;
  }
  .header {
    display: none;
  }
}
```

Short and elegant, we can create a pair of `@media` rules for `min-width` and `max-width` respectively, with the same value and then decide what do to with the elements within those. Having different content for the `mobile-header` and `header`, we decided that we display or hide each of those entirely depending on the screen size.

For the responsiveness of the grid, we made use of the in-built properties of the `ion-col`, namely `size-{'xl'/'l'/'md'/'sm'}` for setting the size of the columns depending on the size of the screen. This property is very common within the Ionic UI elements and it can be used on many occasions for designing the responsive UI between large, medium, and small screens.

```
<ion-grid>
    <ion-row>
        <ion-col offset-xl="3" size="12" size-md="4"
         size-xl="2" class="ion-text-center"...>
        <ion-col size="12" size-md="4" size-xl="2"
         class="ion-text-center"...>
        <ion-col size="12" size-md="4" size-xl="2"
         class="ion-text-center"...>
    </ion-row>
<ion-grid>
```

Another example of how we have used responsiveness can be seen on the products page. As seen in Figures 4.18 and 4.19, for the small to medium screens, there is a `ion-fab-button` which opens on clicking a filter modal for filtering the products based on some chosen category.

The previous button is not available on the web version of the user interface (or

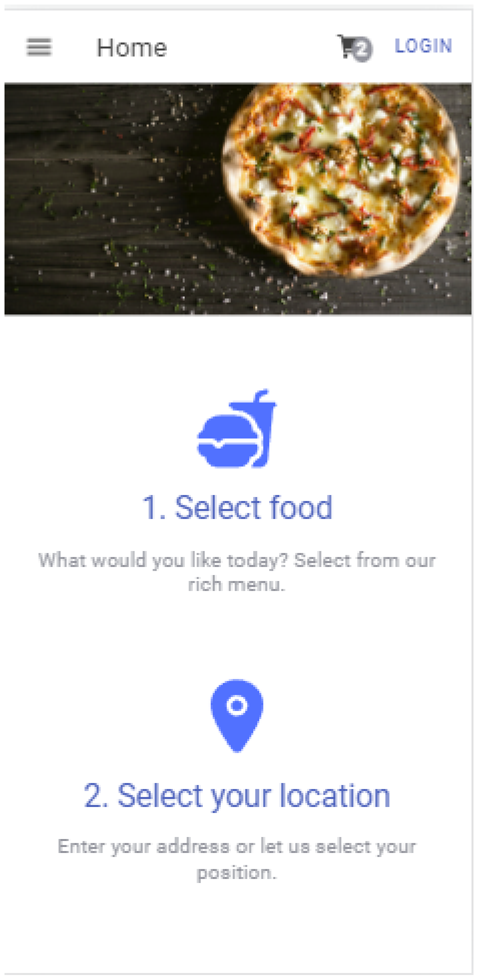

Home
2
LOGIN
1. Select food
What would you like today? Select from our rich menu.
2. Select your location
Enter your address or let us select your position.

Figure 4.12: Mobile view of the home page

large screens), the function of filtering being included in the header in that case. This was achieved with the help of a custom, responsive Ionic css class, namely `ion-hide-{screen-size}-{direction}`. Ionic provides by default such custom styling classes solely for the purpose of responsive designing of the UI components.

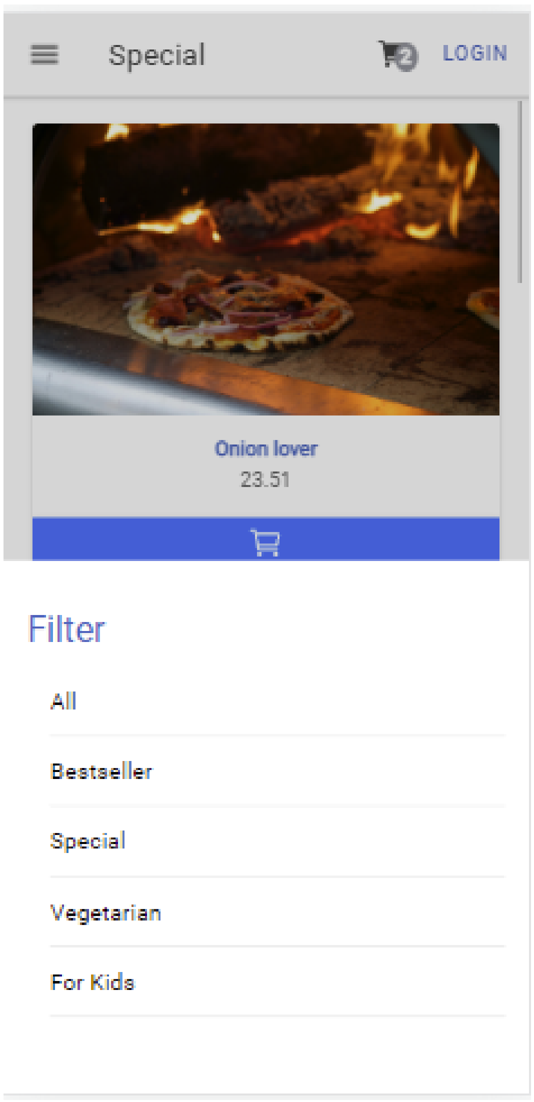


Figure 4.13: Ionic filter modal

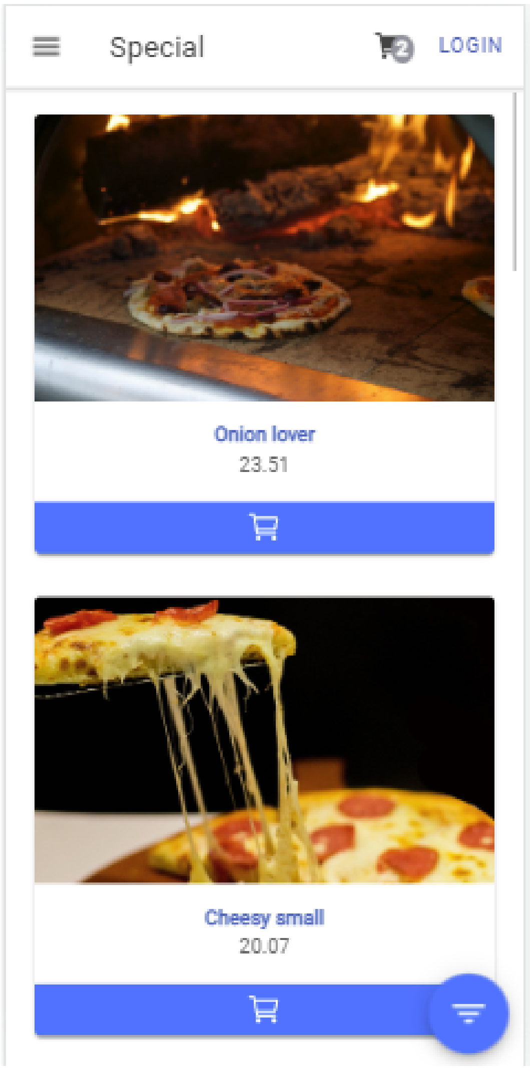


Figure 4.14: Mobile version of products' page

4.2.5 Design Patterns

In the process of implementing this application, multiple design patterns were used.

Observer

The observer design pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The way it is usually implemented is by encapsulating the core components into a 'Subject' abstraction and the variable components in an 'Observer' hierarchy. [Sou]

```
export class CartService {
  private cartItemCount = new BehaviorSubject(0);
  private cartNeedsRefresh =
    new BehaviorSubject<boolean>(null);

    getCartRefresh(): Observable<boolean> {
        return this.cartNeedsRefresh.asObservable();
    }

    notifyCartNeedsRefresh() {
        this.cartNeedsRefresh.next(true);
    }
}

loadCardInformation() {
    this.cartService.getCart().then((response) => {
      this.cart = CartService
        .mapCartDtoStringyfiedToCartDTO(response.value);
      this.computeCartSum();
    });
}

subscribeToRefresh() {
    this.cartService.getCartRefresh().subscribe(() => {
      this.loadCardInformation();
    });
}

addProduct(product) {
    this.cartService.addProduct(product)
        .then((addedProduct) => {
          this.cartService
            .showToast('Product was added to the cart');
          this.cartService.notifyCartNeedsRefresh();
        });
}

removeTip() {
    this.cartService.updateTip(-TIP_AMOUNT).then(() => {
      this.cartService.notifyCartNeedsRefresh();
    });
}
```

One can see the way we implemented the 'Observer' pattern for automatic cart refresh in the above code snippet. In order to construct this pattern, we have declared a variable `cartNeedsRefresh` in the `CartService` class. This will act as an observer to the state of the cart and whether it needs to be refreshed or not. After converting it to the `Observer<boolean>` type with the help of the `getCartRefresh()` method, we can set what behavior should happen when it triggers. In this case, when there is a change in the cart information (it needs to be refreshed), the `loadCartInformation()` method will be called, updating the information of the cart.

How does the observer know when to trigger a refresh ? The method `notifyCartNeedsRefresh()` takes care of this task by updating the value of the `BehaviourSubject<boolean>` variable, thus triggering the the behaviour written in the `subscribe(() => {...})` block, namely loading again the cart information. All there is to do is to call the `notifyCartNeedsRefresh()` in the methods which change the state of the cart, as seen in the above code snippet, for the methods `addProduct()` or `removeTip()`.

Adapter

In any project which is working with multiple libraries and APIs, this design pattern is indispensable. This design pattern can be found in many different forms, but the core principle is the same and pretty obvious from the name: to make objects fit together, or rather, their interfaces. The intent of this design pattern is to convert the interface of a class into another interface clients expect. The adapter pattern lets classes work together that could not otherwise because of incompatible interfaces.

```
export interface ProductDTO {
  id: string;
  title: string;
  description: string;
  category: string;
  price: number;
  imageUrl: string;
}

export interface CartProductDto {
  amount: string;
  id: string;
  title: string;
  price: string;
}

private static mapProductNewToCartProduct(product
    : ProductDTO): CartProductDto {
    return {
      id: product.id,
      price: JSON.stringify(product.price),
      title: product.title ,
      amount: '1'
    };
}

static mapCartDtoStringyfiedToCartDTO(cartDtoStringyfied)
    : CartDto {
    return JSON.parse(cartDtoStringyfied);
}
```

For example, in this application, the model for displaying the information in the products page (and how it comes from the back-end) is different than in the model the products are stored in the cart. Such differences led to the solution of implementing the adapter pattern through the help of mapper methods, which takes the interface of the products as they are represented in the products' page and converts it to an interface which is compatible with using the internal storage.

4.3 Possible Future Improvements

This practical work was just an application for showcasing general features of the Ionic framework and how a hybrid framework would be in practice. There is a lot of room for improvement for the current application, currently, it is a good starting point as a template for any customized shop for a real-world business.

Depending on the scope one wants to follow, there are many branches on continuing the work on this application program. If one wants to continue to explore the other functionalities of this hybrid framework, he can start by implementing testing with the help of 'Jasmine' on top of the default Angular unit testing. Also, there are endless plugins in the Capacitor, as well as in Cordova, for accessing the latest hardware capabilities of the newest devices.

The next logical steps for this exact application would be to integrate a third-party payment method (or even a dedicated one) for the user or select from a list of couriers which would be the best choice based on the location from the customer and the shop, for the admin. From there on, the application can be extended up to the creativity of the programmer who wishes to develop it and to the requirements of the business owner.

After finishing this thesis, we will try to achieve the aforementioned improvements and feature extensions and we will present it to some local businesses for facilitating their online selling.

Chapter 5

Conclusions

Even though hybrid app development is relatively new within the programming world, more and more applications are being developed using these types of frameworks. The popularity of the hybrid community is increasing as the weeks pass and the trend does not seem to stagnate in the near future.

Nevertheless, hybrid frameworks are a great tool to have for small and medium-sized projects. Even though one of the most common issues across all solutions is scalability to complex applications, proven apps such as Twitter, Instagram, Gmail, or Evernote showed that is indeed possible to make a quality product by choosing the hybrid path.

The communities are constantly growing around these frameworks and the non-believers are getting less vocal as months pass in the software developing world. We are optimistic about the future of these technologies and we are expecting a growing demand for such applications as well as for specialized programmers within one of the described hybrid frameworks within this thesis.

Bibliography

[And] Andrew Haire. What is Ionic: Learn the essentials of what you can do with Ionic and how it works.). https://ionic.io/resources/articles/what-is-ionic. Online; accessed May 2022.

[Ang] Angular Io. What is Angular? https://angular.io/guide/what-is-angular. Online; accessed May 2022.

[Ank] Ankush. Understanding Rendering Behavior in React. https://geekflare.com/react-rendering/. Online; accessed May 2022.

[App] Apple Developer Documentation. The Objective-C Programming Language. https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html. Online; accessed May 2022.

[Ban] Ban Markovic. Process of compiling Android app with Java/Kotlin code. https://shorturl.at/eEG57. Online; accessed May 2022.

[Cor] Cordova Contribuitors. Overview. https://cordova.apache.org/docs/en/latest/guide/overview/. Online; accessed May 2022.

[Day] Dayana Jabif. Ionic Capacitor Tutorial - Getting Started with Capacitor. https://ionicthemes.com/tutorials/native-cross-platform-web-apps-with-ionic-capacitor. Online; accessed June 2022.

[Ehr10] David Ehringer. The dalvik virtual machine architecture. *Techn. report (March 2010)*, 4(8):72, 2010.

[Eri] Eric Enge. Mobile vs. Desktop Usage in 2020. https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage. Online; accessed April 2022.

[Flu] Flutter. Flutter architectural overview. `https://docs.flutter.dev/resources/architectural-overview`. Online; accessed May 2022.

[Ger18] Jake Petroules; Jürgen Ributzka; Devin Coughlin; Louis Gerbarg. 2018 apple worldwide developers conference. In *Behind the Scenes of the Xcode Build Process*, pages 1–285. Apple, 2018.

[GGSS89] Nader Gharachorloo, Satish Gupta, Robert F Sproull, and Ivan E Sutherland. A characterization of ten rasterization techniques. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 355–368, 1989.

[GM15] James Goodwill and Wesley Matlock. The swift programming language. In *Beginning Swift Games Development for iOS*, pages 219–244. Springer, 2015.

[HCK$^+$15] Severin Haug, Raquel Paz Castro, Min Kwon, Andreas Filler, Tobias Kowatsch, and Michael P Schaub. Smartphone use and smartphone addiction among young people in switzerland. *Journal of behavioral addictions*, 4(4):299–307, 2015.

[HHM12] Henning Heitkötter, Sebastian Hanschke, and Tim A. Majchrzak. Comparing cross-platform development approaches for mobile applications. In *WEBIST*, 2012.

[Joh] John Callaham. The history of Android: The evolution of the biggest mobile OS in the world. `https://www.androidauthority.com/history-android-os-name-789433/`. Online; accessed June 2022.

[Jus] Justin Johnson, Hemant Arya, David Britch, Craig Dunn. What is Xamarin? `https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin`. Online; accessed May 2022.

[KN13] Maurice Kelly and Joshua Nozzi. *Mastering Xcode: Develop and Design*. Peachpit Press, 2013.

[Lio] Lionel Sujay Vailshery. Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021. `https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/`. Online; accessed April 2022.

[LM88] Marvin B Lieberman and David B Montgomery. First-mover advantages. *Strategic management journal*, 9(S1):41–58, 1988.

[Lor] Lorenzo Sciandra. The New React Native Architecture Explained: Part Four. `https://formidable.com/blog/2019/lean-core-part-4/`. Online; accessed May 2022.

[Mar] Marcelo Pastorino. Frontend vs. backend: what's the difference? `https://www.pluralsight.com/blog/software-development/front-end-vs-back-end`. Online; accessed May 2022.

[Mat] Matthew Martin. What is Backend Developer? `https://www.guru99.com/what-is-backend-developer.html`. Online; accessed May 2022.

[MDN] MDN contributors. Introduction to the DOM. `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction`. Online; accessed May 2022.

[MRST15] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. Hybrid mobile apps in the google play store: An exploratory investigation. In *2015 2nd ACM international conference on mobile software engineering and systems*, pages 56–59. IEEE, 2015.

[MT08] Tommi Mikkonen and Antero Taivalsaari. Web applications–spaghetti code for the 21st century. In *2008 Sixth international conference on software engineering research, management and applications*, pages 319–328. IEEE, 2008.

[Rie00] Dirk Riehle. *Framework design: A role modeling approach*. PhD thesis, ETH Zurich, 2000.

[SB16] Andrea Sánchez Blanco. Development of hybrid mobile apps: Using ionic framework. 2016.

[SJWM05] Debbie Stone, Caroline Jarrett, Mark Woodroffe, and Shailey Minocha. *User interface design and evaluation*. Elsevier, 2005.

[SM16] Guido Salvaneschi and Mira Mezini. Debugging for reactive programming. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 796–807. IEEE, 2016.

[Sou] Source Making. Observer Design Pattern. `https://sourcemaking.com/design_patterns/observer`. Online; accessed June 2022.

[Tar20] Evan Tarver. First mover deifinition, 2020.

[Uda] Uday Hiwarale. How does JavaScript and JavaScript engine work in the browser and node? `https://medium.com/jspoint/how-javascript-works-in-browser-and-node-ab7d0d09ac2f`. Online; accessed May 2022.

[VBS$^+$14] Kathleen D Vohs, Roy F Baumeister, Brandon J Schmeichel, Jean M Twenge, Noelle M Nelson, and Dianne M Tice. Making choices impairs subsequent self-control: a limited-resource account of decision making, self-regulation, and active initiative. 2014.

[VJ17] Tena Vilček and Tomislav Jakopec. Comparative analysis of tools for development of native and hybrid mobile applications. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1516–1521. IEEE, 2017.

[W3C] W3C. Packaged Web Apps (Widgets) - Packaging and XML Configuration (Second Edition). `https://www.w3.org/TR/widgets/`. Online; accessed May 2022.

[WB00] David Anthony Watt and Deryck F Brown. *Programming language processors in Java: compilers and interpreters*. Pearson Education, 2000.

[Zav18] Meet Zaveri. What is boilerplate and why do we use it? necessity of coding style guide, 2018.