Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technologies

Bachelor Thesis

Development of mobile applications for online shopping

Zhuman Aruzhan

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Aruzhan Zhuman

Informatics

Thesis title

Development of mobile applications for online shopping

Objectives of thesis

The thesis will focus on the topic of mobile application development. The primary goal is to assess and compare methods of developing mobile applications in e-commerce and to select a suitable approach for practical implementation.

Partial objectives:

- to investigate several approaches to developing mobile applications using available literature and online sources
- to analyze selected approaches and choose the optimal option for practical implementation
- to develop a prototype mobile application using the chosen method and evaluate the results

Methodology

The thesis will comprise theoretical and practical parts. The theoretical part will be based on the analysis of professional literature and scientific resources. In the practical part, chosen development methods will be compared, and the optimal method will be selected using multiple criteria decision analysis. The selected development method will be analyzed based on the practical implementation of a prototype application. Conclusions will be formulated by combining the results of the theoretical and practical parts.

The proposed extent of the thesis

40-50

Keywords

mobile application, e-commerce, multiple criteria decision analysis, application development, user interaction

---

Recommended information sources

Iversen, Jakob, and Michael Eierman. Learning Mobile App Development: A Hands-on Guide to Building Apps with IOS and Android. Addison-Wesley, 2013.

McWherter, Jeff, and Scott Gowell. Professional Mobile Application Development. John Wiley & Sons, 2012.

Mohapatra, Sanjay. E-Commerce Strategy: Text and Cases. Springer Science & Business Media, 2012.

Panhale, Mahesh. Beginning Hybrid Mobile Application Development. Apress, 2015.

Reynolds, Janice. The Complete E-Commerce Book: Design, Build & Maintain a Successful Web-Based Business. CRC Press, 2004.

Salz, Peggy Anne, and Jennifer Moranz. The Everything Guide to Mobile Apps: A Practical Guide to Affordable Mobile App Development for Your Business. Simon and Schuster, 2013.

Expected date of thesis defence

2023/24 SS – PEF

The Bachelor Thesis Supervisor

Ing. Jan Pavlík, Ph.D.

Supervising department

Department of Information Technologies

Electronic approval: 4. 7. 2023

doc. Ing. Jiří Vaněk, Ph.D.

Head of department

Electronic approval: 3. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 01. 03. 2024

Declaration

I declare that I have worked on my bachelor thesis titled "Development of mobile applications for online shopping" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break any copyrights.

In Prague on 11.03.2024 _____

Acknowledgement

I would like to thank my supervisor Ing. Jan Pavlík, Ph.D. for his advice and support during my work on this thesis.

Development of mobile applications for online shopping

Abstract

The rise of e-commerce has been accelerated by the widespread use of mobile devices and mobile apps, which enable consumers to access e-commerce platforms from anywhere, at any time. A huge variety of present development tools leads to a need to choose between them carefully considering many constraints.

The thesis aims to find the best development method among Swift programming language, React Native framework, Flutter Flow platform and App My Site app builder for an e-commerce mobile application. This goal will be attained by conducting a literature review and MCDA which in its turn will be done according to a given scenario.

Based on MCDA results the optimal solution will be developed in the form of an app prototype that will have a completed user interface but with limited functionalities to perform. The real results will be evaluated contrasting them with the expectations described in the theoretical part and MCDA criteria assessment.

Keywords: mobile application, e-commerce, multiple criteria decision analysis, application development, prototype, Swift, React Native, Flutter Flow, App My Site.

Vývoj mobilních aplikací pro online nakupování

Abstrakt

Vzestup elektronického obchodu byl urychlen rozšířeným používáním mobilních zařízení a mobilních aplikací, které spotřebitelům umožňují přístup k platformám elektronického obchodu odkudkoli a kdykoli. Obrovská škála současných vývojových nástrojů vede k potřebě si mezi nimi pečlivě vybrat s ohledem na mnoho omezení.

Práce si klade za cíl najít nejlepší metodu vývoje mezi programovacím jazykem Swift, frameworkem React Native, platformou Flutter Flow a tvůrcem aplikací App My Site pro mobilní aplikaci elektronického obchodování. Tohoto cíle bude dosaženo provedením rešerše literatury a vícekriteriální analýzy variant (VAV), která bude provedena pro potřeby daného scénáře.

Na základě výsledků VAV bude vyvinuto optimální řešení ve formě prototypu aplikace, který bude mít dokončené uživatelské rozhraní, ale s omezenými funkcemi. Reálné výsledky budou vyhodnoceny v kontrastu s očekáváním popsaným v teoretické části a posouzením kritérií VAV.

Klíčová slova: mobilní aplikace, e-commerce, vícekriteriální analýza variant, vývoj aplikací, prototyp, Swift, React Native, Flutter Flow, App My Site.

Table of Contents

1 Introduction 10

2 Objectives and Methodology 11

2.1 Objectives 11

2.2 Methodology 11

3 Literature Review 12

3.1 Definition and growth of mobile e-commerce 12

3.2 Significance of mobile applications in e-commerce 13

3.3 Types of mobile applications 14

3.3.1 Native App 14

3.3.2 Cross-platform App 15

3.3.3 Hybrid App 16

3.3.4 Progressive Web App 16

3.3.5 Comparison of mobile app types 18

3.4 Application life cycle 19

3.5 Mobile application development approaches and tools 20

3.5.1 Swift 21

3.5.2 React Native 22

3.5.3 Flutter Flow 23

3.5.4 App My Site 24

3.6 Distribution to app stores 25

3.7 Revenue models 26

4 Practical Part 28

4.1 Research questions and scenario 28

4.2 MCDA of mobile application development tools 29

4.2.1 Criteria assessment 29

4.2.2 Calculations 32

4.3 Development of mobile application prototype 35

4.3.1 Planning 35

4.3.2 Design 36

4.3.3 Development 37

5 Results and Discussion 39

6 Conclusion 41

7 References 42

8 List of figures, tables and images 45

8.1 List of figures 45

8.2 List of tables 45
8.3 List of images 45
9 Appendix 46

1 Introduction

Over the past few decades, advances in the development of the Internet and devices have led to a change in many human activities including shopping. The introduction of e-commerce has made shopping flexible for consumers and has opened new opportunities for entrepreneurs. Making the process of purchasing goods online as convenient as possible was also thanks to mobile e-commerce.

Selling and buying goods and services using mobile phones can be done through websites and mobile applications. This thesis will focus on the mobile applications sector. There are four types of mobile applications: native apps, hybrid apps, cross-platform apps and progressive web apps. Each business can choose one of these apps taking into account business requirements and development resources. Each mobile app type in its turn has its own development tools. For example, Android Studio or XCode for native apps and Flutter or Xamarin for cross-platform apps. However, if we take it in general, there are three ways in which mobile applications can be built: traditional development, no-code and low-code approaches. These methods are classified based on the coding involvement level, so in the first method, developers build the app by manually writing the source code while in the last two methods, users create the apps in the graphical user interface and the code is generated behind it. Different means of app development have different drawbacks and benefits which will be described in this bachelor thesis.

2 Objectives and Methodology

2.1 Objectives

The thesis will focus on the topic of mobile application development. The primary goal is to assess and compare methods of developing mobile applications in e-commerce and to select a suitable approach for practical implementation.

Partial objectives:

- to investigate several approaches to developing mobile applications using available literature and online sources
- to analyze selected approaches and choose the optimal option for practical implementation
- to develop a prototype mobile application using the chosen method and evaluate the results

2.2 Methodology

The thesis will comprise theoretical and practical parts. The theoretical part will be based on the analysis of professional literature and scientific resources. In the practical part, chosen development methods will be compared, and the optimal method will be selected using multiple criteria decision analysis. The selected development method will be analyzed based on the practical implementation of a prototype application. Conclusions will be formulated by combining the results of the theoretical and practical parts.

3 Literature Review

3.1 Definition and growth of mobile e-commerce

Apart from the familiar e-commerce sector mobile e-commerce is gaining popularity as well. On a global scale, mobile devices account for 72% of e-commerce sales, and as shown in Figure 1 forecasts suggest that the mobile e-commerce share will surpass 87% by the year 2026.

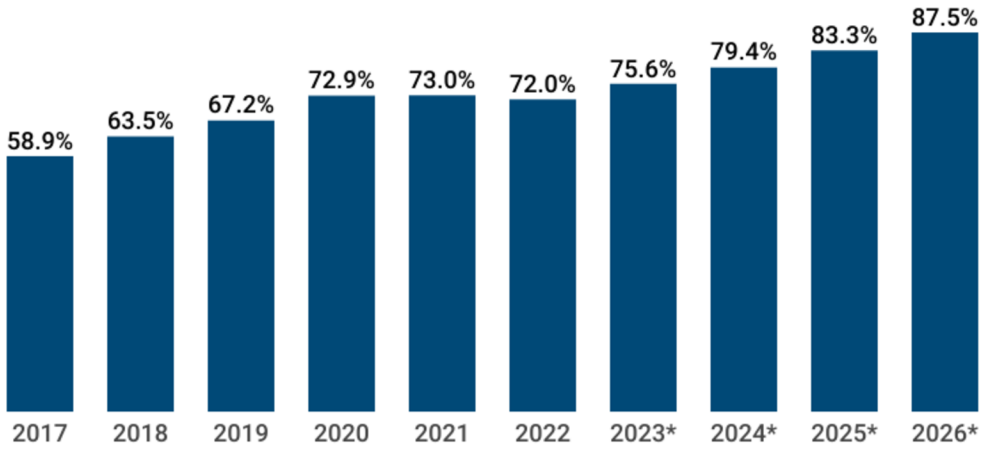

Figure 1: Mobile share of global e-commerce revenue, Source: capitaloneshopping.com, 2024

However, opinions differ as to whether mobile commerce is part of e-commerce or a separate type of trading activity. Furthermore, the classification of devices used in these two types of commerce also varies. Thus, there are several definitions of electronic commerce and mobile electronic commerce:

1) E-commerce and m-commerce together make up digital commerce which indicates transactions held over the Internet. While e-commerce includes transactions made on desktop computers or laptops, m-commerce takes place on mobile phones or tablets. Moreover, it is important to mention that m-commerce is not a derivative of e-commerce (Swilley, 2015, p. 135, p. 15).
2) E-commerce refers to the digital trading and transfer of information, products, services, and financial transactions through telecommunications networks. Mobile e-commerce, alternatively known as mobile commerce or m-commerce, encompasses any actions associated with a (possible) business transaction carried out using wireless devices such as mobile phones and laptop computers through communication networks (Tarasewich, Nickerson, Warkentin, 2002, p. 42).

3.2 Significance of mobile applications in e-commerce

In contrast to traditional e-commerce, which revolves around a company's online store, with m-commerce consumers have the option to shop on their mobile phones from brands and retailers through various channels including mobile websites, mobile commerce apps and social media. Out of these three options mobile applications are mostly known for their higher user satisfaction due to better performance. In fact, individuals look through approximately 4.2 times more products per session within mobile apps than on mobile websites (jmango360.com).

One of the main advantages of mobile apps is that they can access the device's native features like a camera, which is why it leads to more user engagement and opens up a gate for new app functions. For instance, the integration of augmented reality and virtual reality is transforming online shopping from 2D product catalogs to 3D (Baker, Abu Bakar, Zulkifli, 2020 as cited in Ain, Missen, Prasath, 2023, p. 88). In addition, it is found that AR-mediated m-commerce has a positive impact on customer confidence and perception of innovation (Stoyanova, Gonçalves, Brito, Coelho, 2013 as cited in Ain, Missen, Prasath, 2023, p. 88).

Another beneficial feature that can be used in m-commerce applications is making payments through biometrics. Based on a recent Juniper Research report, biometrics are projected to verify over $3 trillion in payment transactions by 2025, a significant increase from the roughly $404 billion recorded in 2020. As mobile payments become progressively dominant in the payment landscape, biometric methods such as fingerprint, iris, voice, and facial recognition are becoming crucial for enhancing user experiences within mobile apps (Ain, Missen, Prasath, 2023, p. 88).

Nevertheless, it is important to deliberate some subtleties of developing mobile commerce apps. According to the research findings, firstly, there is a significant difference in the behavior of males and females when ranking mobile shopping apps since women are more involved in online shopping. Secondly, it turns out that the usability scores of shopping apps for iOS and Android are very dissimilar. Moreover, due to the cultural divide and resistance to technological advances, people in rural areas hesitate to buy online (Ain, Missen, Prasath, 2023, p. 82).

Many business owners are unwilling to include m-commerce in their strategy. Often the reason for this is the involvement of higher costs, but despite that fact, the benefits then can overweight the costs. Therefore, if businesses want to succeed, they need to be ready to

allocate resources and funds towards both technological advancements and marketing works to establish a solid and prosperous mobile commerce platform (Swilley, 2015, p.15).

3.3 Types of mobile applications

There are various mobile application types to choose from when it comes to developing a mobile solution. Here "mobile application type" does not refer to the functionality provided to the user, but to the methodology and structure used in building the application (azure.microsoft.com).

In this research, the most common types of mobile app development will be reviewed, each of which has its own features, benefits and capabilities:

- Native apps
- Cross-platform apps
- Hybrid apps
- Progressive Web Apps (PWA)

The choice of development type depends on various factors such as project requirements, budget, timeline, and the need to access specific device features. Each of these types has its own advantages and limitations, and developers should choose the appropriate type according to the needs and requirements of their project.

3.3.1 Native App

While not requiring an online connection, native apps have excellent visual appeal. For each platform and version you want to target, you will need to create and manage an app depending on the audience. That is why native apps are usually expensive and time-consuming to build and maintain, but they compensate for these drawbacks with high performance.

App stores, both platform-specific and independent, are used to distribute native software. A 30 percent revenue split and the usage of their own billing system are requirements for the majority of app shops. It is crucial to learn how to make the app noticeable among the other millions of apps available because one of the main difficulties faced by app stores is discovery (Salz, Moranz, 2013).

Many various platforms were in competition in the early years before 2010, but now there are just two major platforms (Majchrzak, Biørn-Hansen, Grønli, 2018, p. 5736). Java or Kotlin are the languages that developers use to construct Android apps, with Android

Studio serving as the official IDE (integrated development environment). While iOS developers build apps in XCode using the languages Objective-C or Swift (Thomas, Devi, 2021, p. 117).

3.3.2 Cross-platform App

Cross-platform mobile development is the process of creating software programs that are interoperable with a variety of mobile operating systems. All cross-platform app development frameworks have the following fundamental quality: code is created just once (Majchrzak, Biørn-Hansen, Grønli, 2018, p. 5737).

Code written in a single language that can be utilized by several platforms has its pros and cons. On the one hand, because only one app needs to be created to support all platforms, maintaining and updating apps is much more convenient. On the other hand, the debugging process might be more difficult since some problems might only exist on certain platforms (Jayvir, 2023).

Based on the results of the survey conducted among the developers from the year 2019 to 2022 (Figure 2), the list of the most used cross-platform frameworks was formulated:

1. Flutter
2. React Native
3. Cordova
4. Ionic

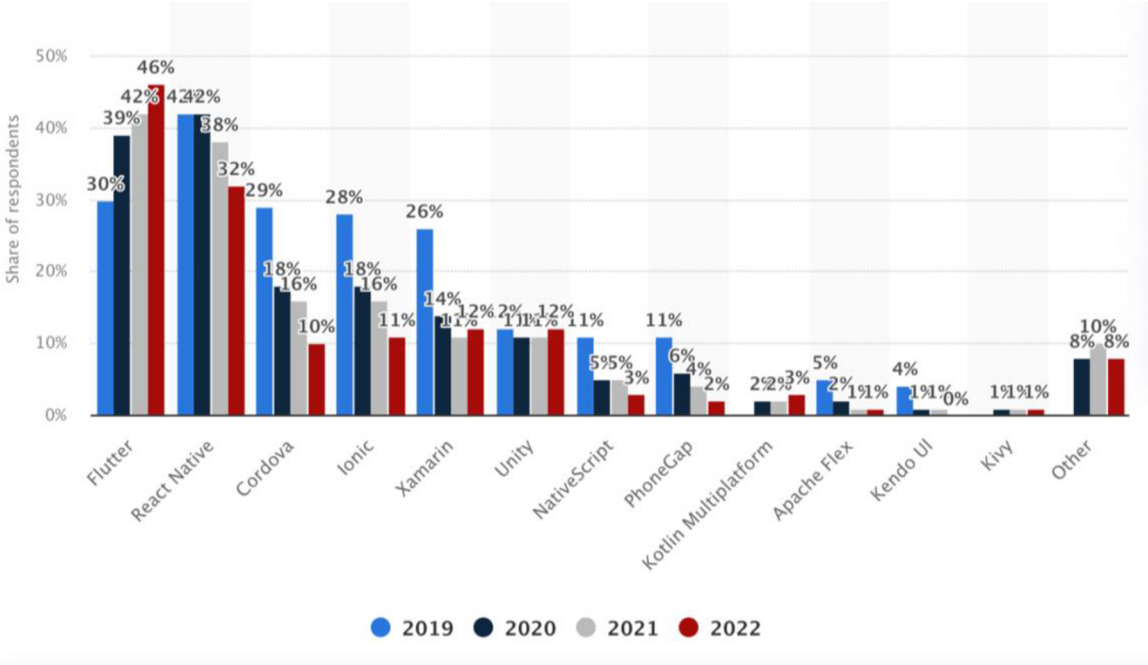

Figure 2: Most used cross-platform frameworks, Source: www.statista.com

Cross-platform development is especially beneficial for small projects because the charges for the application will be lower due to the need for fewer developers, and the development process itself will be much faster. Applications that do not demand a lot of resources from smartphones, like a basic social network or an online store, are less likely to need to intervene in low-level activities. Simple apps will not be negatively impacted by this in any manner in terms of quality or performance. However, this approach pales in comparison to native development when it comes to projects that require lengthy development cycles, ongoing enhancements, and feature expansion (Shevtsiv, Striuk, 2020, p. 76).

3.3.3 Hybrid App

A hybrid application is created using web technologies and then covered in a shell tailored to the target platform. The native shell gives the program a native appearance and, more crucially, qualifies it for app store submission. Additionally, app developers may incorporate some native functionality into a hybrid app, enabling it to utilize hardware elements like built-in sensors and location as well as some native APIs (Application Programming Interfaces) (Salz, Moranz, 2013). Hybrid apps are powered by a single codebase allowing them to run on any platform. Usually, they are written in a well-used programming language like JavaScript, Java, HTML, or CSS (www.ibm.com).

The benefits of hybrid apps extend to content management. Without needing to update the app in app stores, developers may make changes to the app's content directly on the server. As a result, businesses can swiftly adjust to changing user or market demands.

Hybrid apps and cross-platform apps are very similar since they both share a single codebase. However, they are still treated as different development approaches as each of them has its own pros and cons. Firstly, the time that it takes for the development of a hybrid solution is less compared to a cross-platform one. Secondly, access to native features is limited in hybrid apps. In general, hybrid applications are inferior to cross-platform ones in terms of similarity to native apps and the choice between the two mostly depends on business requirements.

3.3.4 Progressive Web App

To get around the limitations of mobile browsing and native programs, in 2015 Google developed Progressive Web Apps (PWA). Similar to how one would launch native

applications, PWA may be accessed simply by clicking on an icon on the device's home screen (Chavan, Bhatkar, Muley, 2022, p. 207).

PWAs are very safe since they only operate over HTTPS (Hypertext Transfer Protocol Secure). However, because PWA is a relatively new technology, it is currently only supported by a small number of browsers (Adetunji et. al, 2020, p. 95). For instance, it is stated on the Google Chrome Help page that there is no PWA support in Chrome for iOS (support.google.com), which is considered as an obvious disadvantage.

Since PWA lies on the border of mobile applications and web apps, the distinctions between them are hazy. Therefore, several research works were done comparing these approaches to answer the question of whether PWA can become a replacement.

Push notifications, offline capacity, and other native app-like capabilities cannot be delivered by responsive web applications. To increase user engagement and conversions, these elements are essential from a marketing viewpoint. Because of this, organizations tend to replace responsive mobile websites by adopting progressive web apps (Chavan, Bhatkar, Muley, 2022, p. 209). However, it is still too early to determine if PWA will be able to displace current cross-platform development methodologies or whether if it can be claimed that they now can meet many requirements for unified multi-platform development. Moreover, even though PWAs aim to support many platforms from a single code base (inherently being web-based), they may not have been recognized as a cross-platform development technique up until this point (Majchrzak, Biørn-Hansen, Grønli, 2018, p. 5742, p. 5737).

3.3.5 Comparison of mobile app types

Table 1: Comparison of mobile app types

	Native apps	Cross-platform apps	Progressive web apps	Hybrid apps
Number of codebases	One per platform	One, but compiled for each platform	One total	One for the app, another for the container
Languages and frameworks	Native only	Team's choice	Web only	Web and native
Access to SDKs and APIs	Yes	Yes	No	Limited
Performance	Highest	High	Lowest	Low
Access to device hardware	Complete	Most	Very little	Some
Responsiveness to user input	Good	Good	Worst	Poor
Interactivity	High	High	Lowest	Low
Device resource use	High	High	Low	Medium
Requires connectivity	No	No	Yes	Yes
Cost to build and maintain	Highest	High	Lowest	Lower
Where the app is stored	Device	Device	Server	Device and server
Deployed through	Marketplace	Marketplace	Browser	Marketplace
Requires outside approval	Yes	Yes	No	Yes

Source: azure.microsoft.com

Given four types of mobile applications differ in every aspect required to be considered in the development process. Thus, each of them will be very specific to the business requirements and it is well seen in the summarized Table 1 with different feature comparisons. The list below is created based on the performance power and consequently also required costs because the higher the performance, the more resources are needed. From Table 1, the application types can be listed in descending order:

1. Native apps
2. Cross-platform apps
3. Hybrid apps
4. Progressive web apps

High quality and capability, also known as the performance of the app, are achieved through having such possibilities as access to SDKs (Software Development Kits) and APIs (Application Programming Interfaces), access to device hardware, responsiveness to user input and interactivity. All of these make devices use more resources but at the same time, it leads to better user experience, and therefore, higher user satisfaction. In this case, building a native app would be an ideal solution, however, in the real world, there are a lot of nuances that no one could easily neglect. That is why other application types were created over time which are enough to attain smaller business needs.

In terms of app deployment PWAs stand out compared to others since it is a web app. So, while the other three types are deployed through marketplaces and require external approval, PWAs are simply published through browsers and do not need to be involved in any approval processes. These web applications are stored on servers, but native and cross-platform applications take up some memory on phones or tablets, and hybrid ones are located both on the device and server. It also means that the applications that require servers do not run without a network connection.

3.4 Application life cycle

The application life cycle represents a sequence of steps that help in the successful delivery of the product. However, the technical team does not have to follow the steps in a strict order, meaning that they can always step back and alter something. Especially when it comes to the testing phase because to resolve the defects found in the testing phase the cycle needs to go back to the development phase. Also, requirements may be changed, or new requirements may be added to the scope of the project. The application life cycle includes:

1. Planning. The business unit defines the objectives that need to be attained by delivering the application to the customers. Technical and non-technical requirements are gathered from the stakeholders. Resources such as financial resources and human resources are analyzed. Timelines for each cycle phase are estimated.
2. Design. Application architecture, user interface and prototypes are settled.
3. Development. Actual building of the backend and frontend of the application.
4. Testing. This is the most important step before going live as all the issues with the app should be investigated during testing real-life scenarios.

5. Deployment. The final version of the application is released and published on the selected platforms.
6. Maintenance. Since the testing cannot cover all possible use cases or simply because some issues might be overlooked, the users may face some problems with the app, so support should be provided. Apart from that the performance of the application is monitored and new improvements are implemented.

In the end, when the decision to retire the app is made, all the documentation and data should be safely archived and that would be the end of the application life cycle.

3.5 Mobile application development approaches and tools

There have only ever been two options for business owners looking to develop applications: either purchase already-made apps from a third-party vendor or create their apps from the start with the help of knowledgeable developers and programmers. Today, however, we are witnessing the emergence and advancement of low-code/no-code development options that enable individuals with little or no programming skills to access the potential of application development (www.sap.com). Thus, three different approaches to software development can be distinguished:

- traditional development (also called pro-code)
- low-code
- no-code

The classical approach to software development based on hand-writing source code is known as traditional programming. Developers use programming languages and tools such as IDE and cross-platform frameworks to create applications from scratch. This method has a high degree of flexibility and control, allowing developers to fully tailor each part of the program to the needs of the project. Traditional programming is often used to develop complex and vital applications that require high performance, security, and reliability. However, since developers must write all the code by themselves, this method is time and resource intensive. It can lead to a higher chance of errors and slow down the development process, especially in complex projects. In addition, finding and attracting developers with the right skills to work based on traditional programming can be difficult.

The problem of the discrepancy between the rising demand for apps and the lack of developers available to satisfy this need led to the launch of low-code development

platforms (Alsaadi et. al, 2021, p. 123). The use of minimal coding in low-code software tools has influenced the popularization of citizen developers, i.e., developers with little or no expertise in software engineering (Oltrogge et. al, 2018, p. 634), for instance, employees from company departments other than IT who use low-code platforms to enhance digital transformation. Both terms "citizen developers" and "low-code platforms" are usually used with regard to enterprise organizations or companies. Examples of low-code platforms are Appian, Microsoft PowerApps and Mendix.

Low-code and no-code approaches are very similar as in both visual development environments and intuitive interfaces are used that allow the creation of applications by dragging and dropping ready-made components, modules, and functions. The key advantages of these two alternatives to traditional programming are enhanced performance and a less complicated development process. The main difference as the names of approaches already suggest is that no-code does not require any programming knowledge while in low-code some code lines may be entered by the user. Less code involvement means more restrictions in app customization, therefore, such platforms in general will be suitable for smaller and simpler projects.

There is a huge variety of tools used in each of the development approaches. In order to make the comparison of development tools less complicated and healthier in the practical part, some restrictions will be applied to the selection of development methods that will be analyzed. Firstly, because low-code development platforms are mostly used in enterprises, only no-code and traditional approaches will be considered. Secondly, two types of tools will be reviewed for each approach to make it fair. Development tools are selected due to their well spreading in the market:

Traditional development:

- for native apps – Swift programming language
- for cross-platform apps – React Native framework

No-code development:

- from scratch – Flutter Flow
- converting from website – App My Site

3.5.1 Swift

Swift is a programming language introduced by Apple in 2014 to develop applications for its ecosystem. Swift 3 was available as open source in 2016, making it the

first significant version that allows anybody interested in the language to work on its development. Since then, Swift has expanded outside of the Apple environment at an even quicker rate. There are now several platforms apart from Apple's that support Swift's development tools and the programs built in the end can be deployed on these platforms. According to Swift's official website, these platforms include:

- Windows
- Ubuntu
- Amazon Linux
- CentOS (www.swift.org)

Swift is a new programming language, but it has many similarities with Objective-C which is a superset of C in terms of object-oriented paradigm and syntax (www.geeksforgeeks.org, 2020). Swift stores values in variables that may be referred to by a name in the same way as it is done in C. All essential C and Objective-C data types, such as Int for integers, Double and Float for floating-point numbers, Bool for boolean values, and String for textual data, are available in Swift as well (docs.swift.org). Moreover, developers have the option to implement Swift in the same project with Objective-C and C++ files and have access to those languages' APIs (developer.apple.com). Such features lead to faster learning and make the transition to Swift easier.

Despite being a still developing young language, Swift is considered a modern approach to building applications. Thanks to its unique features such as automated memory management, initialization of variables before use and checking for overflow in arrays and integers, Swift enables safer programming. It is 8.4 times quicker than Python and 2.6 times faster than Objective C (www.apple.com) which indicates a great performance. In conclusion, Swift is a good choice for developing native applications.

3.5.2 React Native

React Native is an open-source cross-platform framework initially established by Facebook in 2015 (Hjort, 2020, p. 12). Many businesses apply it, including Airbnb, Tesla, Walmart, and Uber (Barros et. al).

React Native applications can be developed in one of the several development environments including:

- Expo
- React Native CLI
- Visual Studio Code
- Android Studio and XCode (Nathan, 2023, p. 6)

According to the research conducted by Barros et. al, it was found that despite the generally quicker functionality of native apps developed in Swift or Java, as an example, saving and retrieving data from local storage on Android and iOS devices falls under the category of situations where React Native apps perform statistically equally well. With this technology, iOS and Android applications can be written in JavaScript based on React JavaScript library to access the platform's API. React JS or React allows the development of the app's UI components through JavaScript XML (JSX) which is a syntax for encasing XML within JavaScript (Fentaw, 2020, p.21).

Given that JavaScript is often used by web developers, it is safe to say that using it is one of React Native's major benefits because instead of learning a new language and development environment for each platform, it enables the creation of mobile apps using existing well-known methodologies (Hjort, 2020, p. 12). However, when it comes to new joiners or other developers, the knowledge of JavaScript acts as a prerequisite for starting to work with React Native. Another thing to consider is that since React Native has React library as a dependency, one should also learn React fundamentals first before jumping right into React Native app development. On the other hand, the developer community for React Native is huge offering a lot of information and assistance for creating apps (Nathan, 2023, p. 3), so it is easier to get the support if needed.

3.5.3 Flutter Flow

Flutter Flow is a comparatively new platform used for the visual development of iOS and Android mobile applications and web applications. It allows quick creation of apps based on Flutter which is an open-source framework established by Google to develop cross-platform apps with the same codebase.

The main principle of operation is to create a user interface using a drag-and-drop system and on the basis of this the program code is generated. The interface can be created by users themselves or they may use ready-made templates, which will significantly speed up the work process. Flutter Flow creates a number of opportunities for the most efficient

and convenient work on projects. This includes integration with external servers and APIs, real-time preview of the application and team collaboration. Flutter Flow is especially useful for those who want to take advantage of the power and flexibility of Flutter but may not have much coding experience or want to just simplify the app development process. However, like many no-code app builders, Flutter Flow can help create only the basic structure of an app, meaning that more complex and custom functionality still requires traditional Flutter coding. It is not a drawback but rather a factor that must be considered before choosing the development tool because each of them has different purposes.

3.5.4 App My Site

App My Site is another popular no-code app builder, but its difference is that it mainly focuses on making mobile apps by converting them from websites. Using this tool one can still create a custom app even if he or she does not own a website, but the tool's most powerful solutions come from collaborating with platforms such as WordPress, WooCommerce and Shopify. Although App My Site supports the creation of various app types, the last two examples clearly show that e-commerce especially emphasizes attention.

Converting an existing website into an iOS or Android app takes several minutes which is extremely fast compared to other app builders not to mention traditional app development. As a result, customers get a website and a mobile app in sync. A privilege that is worth mentioning is that users do not have to pay in advance to use this tool. Instead, they can pay in the end after creating the app and testing it on different operating system emulators or even real devices. Other advantages include an automated deployment process and monitoring important data based on provided analytics.

App builders like App My Site are mostly convenient for small or start-up businesses since the solution does not require a lot of resources and is ready in a short period of time. Nevertheless, still, there are some disadvantages that make people avoid it. Firstly, being dependent on third-party services leads to risks in the form of unpredictable price changes or losing support because of the vendor's cancelled business. Secondly, data security may not be guaranteed to the right extent.

3.6 Distribution to app stores

Publishing an application to iOS and Android platforms involves going through several steps that are described below:

1. Preparation of assets and documentation.

   Assets like app icons, screenshots and promotional materials are created.

   An app description is written, and any required documentation is prepared for submission.

2. App Store Registration.

   For iOS:

   Registration as an Apple Developer on the Apple Developer website.

   Configuration of App IDs and provisioning profiles specific to the app.

   Payment of the fee for the Apple Developer Program.

   Getting ready for App Store Review.

   For Android:

   Registration as a Google Play Developer using the Google Play Console.

   Payment of a one-time registration fee.

   Configuration of the app listing details including pricing information.

3. App Store Listing.

   For iOS:

   An app listing is created on App Store Connect.

   App information such as name, description, keywords and screenshots are provided.

   In-app purchases are set up if applicable.

   Submission of the app for review.

   For Android:

   An app listing is created on the Google Play Console.

   Details including title, description, graphics and pricing information are provided.

   If applicable, in-app purchases are set up.

   The app is published on the Play Store.

4. App Store Review.

   For iOS:

   The app will undergo a review process conducted by Apple. It should adhere to Apple's guidelines.

Developers should be prepared to handle any issues or rejections from Apple's review team.

For Android:

The Google Play Store does not have a publishing review process, but they may review apps afterwards to ensure compliance, with their policies.

5. Distribution.

   Once the app is approved (for iOS) or published (for Android) it will be available for users to download.

Using marketplaces like App Store and Google Play is considered to be the most optimal way for public app distribution, although developers have a variety of other options, especially when it comes to Android apps. For example, Android apps can also be released by publishing them on websites or sending them directly to users. On the other hand, Apple forbids downloading and installing any iOS binary files directly. Also, the approval process for publishing an app to the App Store is much stricter than in Google Play (buildfire.com, 2023). Considering these factors and that Android is dominating the global market share, it is estimated by Statista that by 2026 the number of app downloads from the Google Play Store will be almost four times more than app downloads from the App Store, making up 143 billion (www.statista.com).

3.7 Revenue models

Mobile app revenue models represent the means of monetizing the applications. App developers believe that selecting a revenue model is one of the hardest and most significant choices they must make in order to thrive in the market (Roma, Ragaglia, 2016, p. 174). Some of the most common revenue models include a free revenue model, a paid revenue model and a freemium revenue model.

Some developers use a free app approach, making the app available without any charge. In this case, the revenue is targeted to be generated from in-app advertisements or from selling non-personally identifiable user data (Roma, Ragaglia, 2016, p. 175). In-app advertisements are now used almost in every free downloadable app and usually, it's a rather easy procedure that requires the developers to use the SDK that the ad provider provides (Devi, Thomas, 2021). However, the main prerequisite for this strategy to succeed is having a sizable user base. On the other hand, the information that the app could be

gathering could be very beneficial to organizations that do marketing or research. Perhaps in the future, the data gathered will be utilized to give consumers a better user experience. Typically, data that is collected include the device the app is operated on, location, network and so on. It is required to disclose this to the user in a public manner through the Privacy Policy and the Terms of Service.

While most applications available on the market may be downloaded for free, others require payment upfront before installation. These apps allow users full access to all functions without any additional adverts or interruptions. Developers must ensure that the cost they incur to obtain a certain feature is justified, although this is an incredibly easy and practical method of making money. Based on Roma's and Ragaglia's findings, compared to free applications in the Apple App Store, paid apps are more likely to be linked to higher daily revenue levels, even though there is no noticeable difference between them on Google Play. This is because customers of the Apple App Store are less concerned about spending their money if they receive superior service since they are more eager to pay than users of Android apps (Roma, Ragaglia, 2016, p. 188).

Combining the terms "free" and "premium," "freemium" refers to a business strategy that depends only on in-app purchases. Users pay for more features, the upgrading of an app to access all content and services, or the elimination of ads that appear in the free version of the app. Freemium business models are popular in the gaming industry. In fact, among all app categories, gaming is by far the most downloaded and offers the highest income (Tang, 2016, p. 225).

Choosing the right revenue model is critical and depends on various factors, but it is also possible and even beneficial to combine them to maximize profitability.

4 Practical Part

4.1 Research questions and scenario

The practical part of this thesis will include two steps: first is to analyze the development methods for building e-commerce mobile applications to choose the optimal method and afterward build the application prototype using the chosen developing tool. Going through these two steps answers to the following research questions will be found:

- Which development method is best to implement a practical solution for online shopping applications?
- Will the final prototype meet the expectations that are based on the literature review and MCDA criteria assessment?

The first part of this practical work will utilize multi-criteria decision analysis (MCDA). The MCDA will be done according to one given scenario. Describing this scenario ahead is crucial because points like which criteria to choose and what weights to assign to them are dependent on certain constraints and that is why the optimal alternative may change in different scenarios.

In this work, the solution to the problem should be found for the business whose main business activities are selling women's clothing in stores and through their website. This is a developing brand that first conducted its trade online because of the pandemic and only after two years a store in Prague was opened. Since most sales come from online shopping, the brand owners decided that launching a mobile application would be beneficial for them to raise the customers' satisfaction by offering a more pleasant and fast shopping. The brand has no strict time limits needed for the development since it already has a working website. However, the budget is limited because it is only a developing business that was opened a few years ago. The brand owners came to a development company to consult their situation with professionals. Now the company has to offer their clients an optimal solution.

4.2 MCDA of mobile application development tools

MCDA is a chosen technique in this thesis to compare the ways of developing a mobile e-commerce application and calculate the optimal solution. The process is based on evaluating each criterion for each alternative and giving the weights to them.

An alternative is a variant that is considered and one of several alternatives will be chosen in the end as an optimal one. In this case, there are four alternatives for building a mobile application for shopping: Swift programming language, React Native framework, Flutter Flow and App My Site.

Weights are assigned according to the importance of each criterion, and this is done subjectively. For example, the cost of development may be the most important to consider for one and therefore, its weight will be the highest. However, if the cost is equally important with other criteria, then each criterion will be given the same amount of weight. In the end, the weights are added together to investigate the ranking of alternatives. The higher the sum is, the higher the ranking will be.

The concept of MCDA has more nuances such as normalization that will be described throughout the whole process of calculation.

4.2.1 Criteria assessment

For MCDA of development tools for e-commerce mobile applications, five criteria were chosen. These are development cost, functionality range, maintenance and support, performance and security.

1. Development cost evaluates the budget needed from the beginning till the end of the development cycle. The evaluation will be written in a verbal form because it might be challenging to provide concrete numbers.

The highest development cost is covered by using **Swift** programming language due to its native development nature which also assumes the potential development of the application for Android users. This means that the cost of development could double in amount. Thus, for this criterion, the Swift alternative gets the evaluation of "very high" cost.

Regarding **React Native**, although it is a cross-platform technology that allows building applications for both iOS and Android, it still requires the assistance of developers or software development firms as well as Swift. Compared to these two alternatives, **Flutter Flow** and **App My Site** are app builders that do not require special programming skills, and

that in turn excludes the need for specialists. That is why the cost will be lower. So, the estimation is as follows: cost of development in React Native – "high", in Flutter Flow – "average" and in App My Site – "average".

The services of the last two alternatives are provided through monthly subscriptions. According to the official websites, it is 30$ for the standard and 70$ for the Pro package in Flutter Flow, 29$ for the Pro and 49$ for the Premium package in App My Site. The Standard package in Flutter Flow and the Pro package in App My Site contain similar services and the same situation is for the Pro package and the Premium package. It is worth mentioning because the possibility of having different naming conventions may impact the comparison of the pricing systems. Also, both platforms allow building a demo app for free which is a great opportunity to test if the solution is suitable. In summary, due to the similar pricing of both no-code platforms, the cost of development was set as "average" for both Flutter Flow and App My Site alternatives. In addition, one may argue that paying a monthly fee for years will result in a high cost but in comparison with the first two alternatives it is lower. That is why the cost was not set as "low", but it is "average" instead because the long-term running of the app is considered.

2. Functionality range assesses the possibility of adding and customizing various features of the application. **Swift** and **React Native** apps are not restricted in this spectrum because the solution in both cases happens to be custom solutions. This means that developers have the freedom to build any features they want to implement, consequently, the results of this criterion for both Swift and React Native is "versatile".

In **Flutter Flow**, many given templates can be used to build an application. At the same time, this platform also offers an opportunity to start the project from scratch without using any templates. The drag-and-drop interface allows users to add the most needed functionalities but because of this very interface, the range of these functionalities is limited compared to code-based development. However, when Flutter Flow is put together with **App My Site**, its variety of functionalities appears to be higher due to the conversion of the website to the mobile application in App My Site. So, in App My Site, what will be included in a mobile app will directly depend on the website itself. Thus, App My Site gets "limited" for functionality range and Flutter Flow gets "average".

3. Performance indicates how fast and smooth the application runs. Native apps show the best-optimized performance since they are built for a single platform, leading to efficient system resource utilization. Performance results of cross-platform apps are not that high because of the additional layer between the code and the underlying platform. Also, these types of apps may be influenced by updates introduced by the framework developers. Therefore, it is fair enough to state that **Swift** apps have "very high" performance and **React Native** apps have it as just "high".

Another "high" performance can be observed in **Flutter Flow** applications. Flutter framework uses Dart, a compiled language, and as an outcome, it might have a better runtime performance compared to interpreted JavaScript in React Native. In addition, Flutter's widget-based architecture grants a high degree of control over the UI which can lead to smooth animations and responsiveness.

The performance of apps built with **App My Site** may depend on the underlying technologies and frameworks used by the platform. The app can be generated from any website that was created using any tool, for example, WordPress. That is why it is a little bit complicated to give a certain answer when it comes to its performance, therefore, it is indicated as "average".

4. Security is a criterion that measures how the application is secured from vulnerabilities like hacker attacks, data leakage and third-party dependencies.

Swift and **React Native** win in this category because custom apps are developed with full control over the codebase. Developers can implement security best practices, conduct thorough security testing, and address specific security concerns based on the application's requirements. Custom security features, encryption algorithms, and authentication mechanisms tailored to the specific needs of the application may be implemented with ease. Moreover, developers have the power of control over the selection and usage of third-party libraries, ensuring that only reputable and secure libraries are integrated into the app. In the case of no-code app builders, limited control over code makes everything dependent on the vendor's predefined security measures. Also, dependency on third parties automatically affects the application's vulnerability. This is because third-party companies may change their policies or pricing systems at any time, and it might have a negative impact. Such limitations provide **Flutter Flow** and **App My Site** with a security level of "low" while in contrast, Swift and React Native obtain it "high".

5. Maintenance and support evaluate how specialists in the chosen method help maintain the application by upgrading versions of the app and resolving potential bugs in it. This criterion also evaluates if those specialists are ready to assist in case of other issues or questions.

Swift and **React Native** have a big advantage in this case because again using these tools custom applications are developed. Working with developers or software companies, you meet with real people, and that increases the likelihood of being heard. On the other hand, building the app on your own may complicate things when it comes to issues with the app. For example, types of support in **App My Site** are only community and basic email support. It will take a long time before it will be your turn after thousands of other customers like you to get help. **Flutter Flow**, however, has official access to experts who can take care of the app and meet the requirements that you need. So, in terms of maintenance and support Swift and React have it as "efficient", Flutter Flow – "average" and App My Site – "inefficient".

To summarize, the assessment of MCDA criteria can be recapped in Table 2.

Table 2: Assessment of MCDA criteria

		Criteria				
		Development cost	Functionality range	Performance	Security	Maintenance and support
Alternatives	Swift	Very high	Versatile	Very high	High	Efficient
	React Native	High	Versatile	High	High	Efficient
	Flutter Flow	Average	Average	High	Low	Average
	App My Site	Average	Limited	Average	Low	Inefficient

Source: own processing

4.2.2 Calculations

Step 1. Scale conversion

There are certain issues with selecting the best alternative based on Table 2. Firstly, all criteria are evaluated using different terms. For instance, the functionality range has a versatility scale, while the maintenance and support have an efficiency level. Secondly, all criteria are not expressed in some units, instead, linguistic terms are used. Therefore, to

resolve the issues words should be converted into numbers before initiating the calculations.

For this case, a scale of a maximum of 5 points will be used as shown in Table 3.

Table 3: 5 points scale

1	Very low	Limited	Inefficient
2	Low	-	-
3	Average	Average	Average
4	High	-	-
5	Very high	Versatile	Efficient

Source: own processing

Step 2. Decision matrix

After substitution with numerical values, a decision matrix is created. In Table 4, each value in each cell is known as a performance value x_{ij} of i^{th} alternative over j^{th} criteria.

Table 4: Decision matrix

	Development cost	Functionality range	Performance	Security	Maintenance and support
Swift	5	5	5	4	5
React Native	4	5	4	4	5
Flutter Flow	3	3	4	2	3
App My Site	3	1	3	2	1

Source: own processing

Step 3. Non-beneficial and beneficial criteria categorization

Next, criteria are categorized into non-beneficial and beneficial criteria. Non-beneficial criteria are those criteria whose lowest value is desired and with beneficial criteria it is vice-versa, meaning that their highest value is wanted. In our scenario, only the development cost is the non-beneficial criterion because customers would like to have a product with the lowest cost. The remaining four criteria are classified as beneficial based on the same logic.

Step 4. Normalization

To make all criteria comparable normalization is done. In beneficial criteria, the performance value of each cell is divided into the maximum value of all alternatives in the column. Similarly, for non-beneficial criteria, the minimum value is divided into the performance value.

Table 5: Normalized decision matrix

	Non-beneficial	Beneficial	Beneficial	Beneficial	Beneficial
	Development cost	Functionality range	Performance	Security	Maintenance and support
Swift	0,60	1,00	1,00	1,00	1,00
React Native	0,75	1,00	0,80	1,00	1,00
Flutter Flow	1,00	0,60	0,80	0,50	0,60
App My Site	1,00	0,20	0,60	0,50	0,20

Source: own processing

Step 5. Assigning the weightage

According to the given scenario, the store owners have a limited budget and no other constraints. Consequently, the highest weightage of 40% is assigned to the development cost criterion. Since the sum of all criteria weights must be 100% and because all remaining four criteria are equally important, they are granted the same weight of 15%.

Step 6. Weighted normalized decision matrix and performance scope

The weightage percent is converted into a number. Then the weight is multiplied by each criterion with its normalized performance value. On solving the weighted normalized decision matrix is created.

As the last calculation all weighted normalized performance values of each alternative are added to get a performance scope.

Table 6: Weighted normalized decision matrix

	Non-beneficial	Beneficial	Beneficial	Beneficial	Beneficial	
	Development cost	Functionality range	Performance	Security	Maintenance and support	Sum
Swift	0,24	0,15	0,15	0,15	0,15	0,84
React Native	0,30	0,15	0,12	0,15	0,15	**0,87**
Flutter Flow	0,40	0,09	0,12	0,08	0,09	0,78
App My Site	0,40	0,03	0,09	0,08	0,03	0,63
Weights	0,40	0,15	0,15	0,15	0,15	

Source: own processing

Step 7. Ranking

The performance score helps allocate ranks to mobile development methods. The full order of alternatives in descending order is shown in Table 7.

Table 7: Ranking of alternatives

Rank	Alternative	Performance score
1	React Native	0,87
2	Swift	0,84
3	Flutter Flow	0,78
4	App My Site	0,63

Source: own processing

Based on the weightage assigned to each criterion React Native is the best alternative among all other alternatives as it has rank 1. Therefore, a mobile application prototype for online shopping will be built using this development method.

4.3 Development of mobile application prototype

Since only a prototype is developed, the application lifecycle will not be completed. Nevertheless, it is possible to have the first three steps which include planning, design and development itself.

'Prototype' in this work means a mobile app with a completed user interface but at the same time with limited functionalities. It will be possible for the user to perform the very basic steps such as navigating between pages, choosing between several options and clicking on some buttons. The system on its side will also do some calculations according to the user's input. In this way, the application will enable some real-life user experience.

4.3.1 Planning

Despite being short, the scenario still allows to figure out generalized requirements and timelines for this project. The main objective is to develop a mobile application for a women's clothing brand. Thus, the prototype should cover the key features and functionalities of the average e-commerce app:

1. Homepage should be displayed to the user showing:
    - clothes cards classified into categories
    - search bar
    - brand name
2. Details page should be displayed for each clothing and include:
    - clothing title
    - price

- user rating
- description
- possibility of choosing the size
- possibility to add an item to the shopping cart

3. Cart page should be displayed to the user and include:
    - clothing cards
    - possibility to increase and decrease the quantity of the item
    - total price
    - possibility to checkout
4. Payment page should contain several payment methods.

In addition, it was written in the scenario that the timelines are not heavily restricted due to the presence of a running website. It leads to an estimated design period of 1 day and a development period of one month.

4.3.2 Design

According to the listed requirements, a low-fidelity wireframe was created. This wireframe helps properly locate the design figures on each page. Most importantly, it allows to think about the user experience and structure functionality flows. The wireframe (Image 1) for this project was done on paper and drawn by hand.

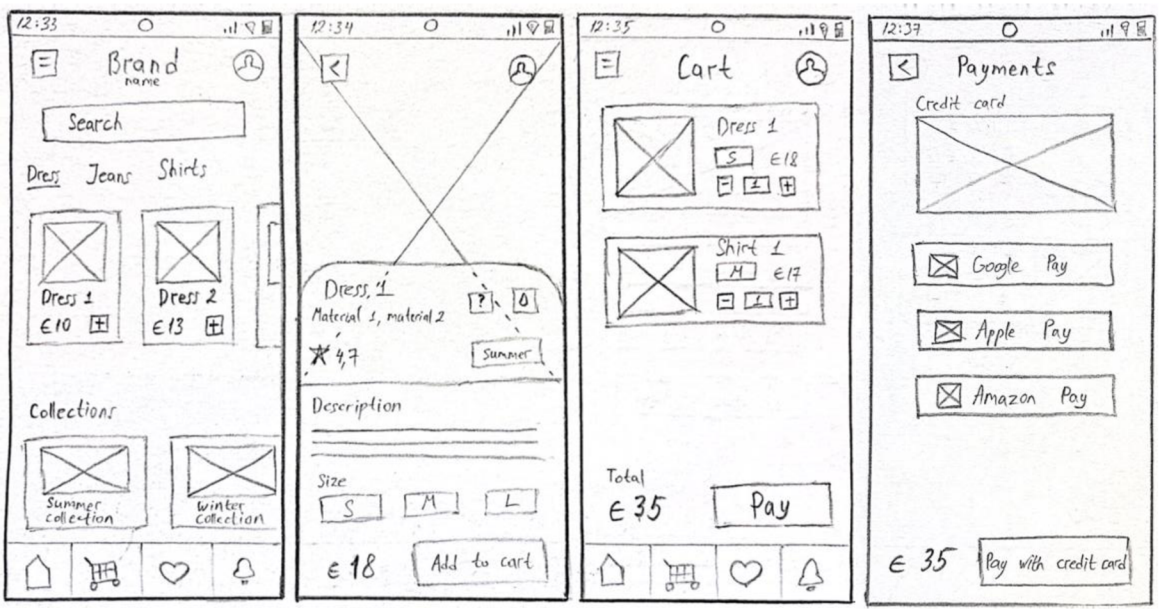

Image 1: Wireframe, Source: own processing

In total, there will be four working pages: homepage, item details page, cart page and payment page. The navigation will also be in this order.

4.3.3 Development

The first step in starting any development process is the setup of the development environment. The prototype was decided to be built for an Android OS since it is the most shared in the market. Moreover, it is important to mention that the used machine runs on MacOS because the development tools may also vary according to the OS. Here is a list of tools that were installed to develop an Android application on MacOS:

- iTerm 2 is a terminal for MacOS
- Homebrew is a package manager for MacOS
- Node.js is a JavaScript runtime environment that comes with its package manager (npm)
- Watchman is a tool developed by Facebook that automatically reruns the project whenever there are changes in files
- React Native and React Native CLI
- Android Studio is an IDE that runs Android emulators
- VS Code is a source code editor

After setting up the environment successfully the assets were prepared as a prerequisite for beginning coding. The assets contain pictures and fonts that were also downloaded from external resources.

Then the usual steps of developing the project included opening the project in VS Code, running the React Native in the terminal and another terminal was initiating the Android emulator. The main dependency of the source code is represented by the relationship between the files in the Components and Screens folders: each Screen utilizes a few Components that represent some UI elements. Calculations are described in store.tsx file. The final solution is shown in Images 2 and 3.

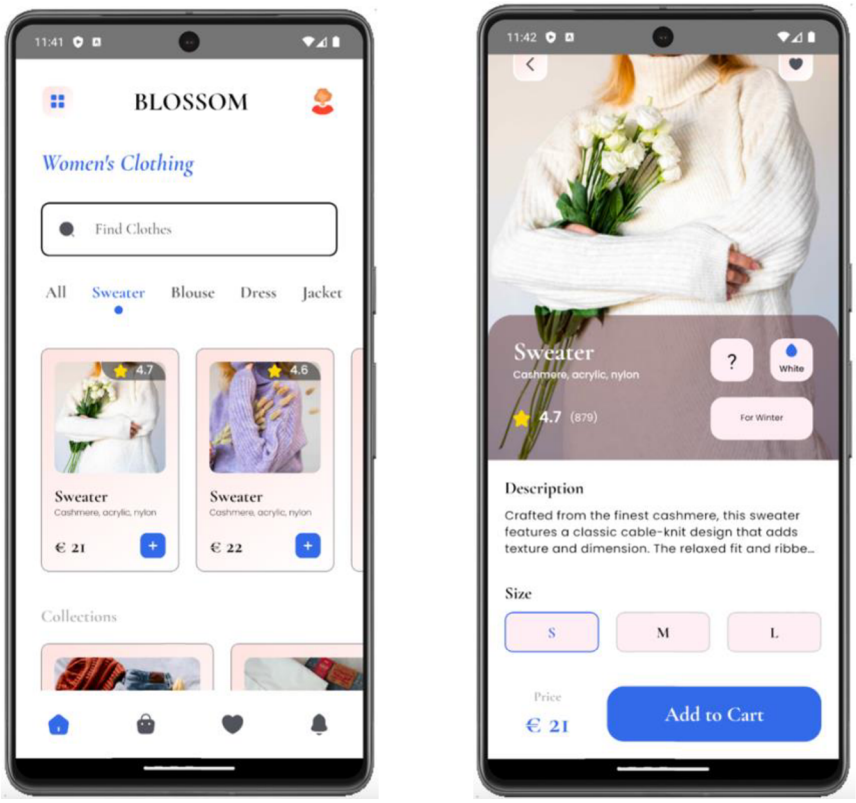

Image 2: Homepage and Details page, Source: own processing

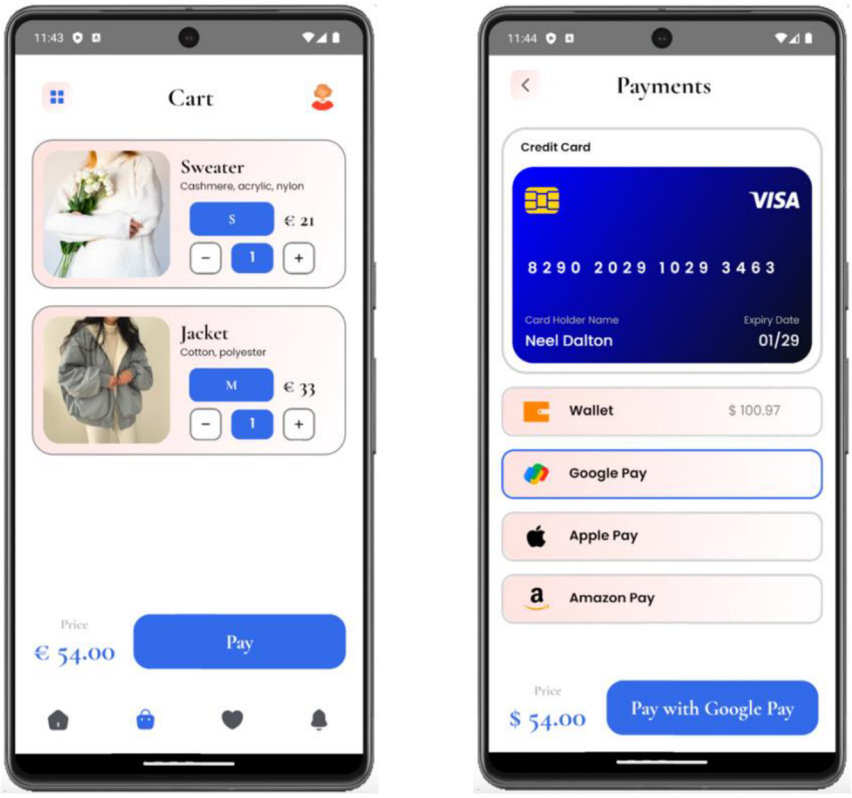

Image 3: Cart page and Payments page, Source: own processing

5 Results and Discussion

As it was estimated in the planning phase the mobile application prototype was ready in one month. The only unexpected part of the process was regarding the period of the development environment setup. It took more time than expected because of a variety of dependencies that should have been installed correctly on the new system. Although there were a lot of minor mistakes in writing the source code including both syntax and semantic errors, the most serious errors throughout the whole project occurred specifically because of the wrong configuration of packages.

On the other hand, because of facing such difficulties, it was possible to evaluate the support level of the React Native developers' community. All the errors that appeared during the development were feasible to solve by reading the web forums and even by watching explanatory videos on YouTube. The fact that many people were having similar problems and finding answers to their questions means that the React Native community is big enough and willing to assist each other. In addition, at the beginning of the development and while getting some error messages the official React Native documentation was mainly used. The explanation is understandable even for beginners and in general, it was very useful till the end of the development.

Regarding the programming language utilized in the solution, it is mentioned in the theoretical part that React Native applications can be written in JavaScript. However, during the development process itself it was found out that besides JavaScript, the source code can also be written in TypeScript which is a newer and enhanced version of JavaScript which offers more language features. That is why it was decided to use the object-oriented language, TypeScript.

Furthermore, the literature review contains information about how knowing JavaScript is an advantage for starting React Native development since many web developers already use it. From my personal experience with the solution for this thesis, I can say that having some knowledge of HTML and CSS without JavaScript can also be counted as an advantage. Dealing with the code that looked familiar gave me some sort of strength and helped me to feel more confident.

To summarize, the final app prototype corresponds to the requirements of the project. The solution was completed on time and did not bring any cost because all

development tools were open source. The range of functionalities that could be added was not limited and the performance of the app is stable.

6 Conclusion

Finding the best solution for the practical implementation of mobile e-commerce applications have required evaluating and contrasting several development approaches such as Swift, React Native, Flutter Flow and App My Site. This main goal of the thesis was achieved because as the result of MCDA that was based on the literature review, the mobile application prototype was delivered using React Native Framework.

In the theoretical part, several sub-topics of e-commerce and mobile application development were described. The practical part was divided into two parts: the analysis of the chosen development tools using the MCDA approach and the development of the final solution utilizing the method that was calculated as the best alternative in MCDA.

As a result, considering the scenario, an Android app prototype was built having four pages with the necessary functionalities used in all e-commerce solutions. Despite attaining the project goals, next time it would be beneficial to add the testing phase. In that way, it would be possible to get some feedback from users who would get almost a real-life experience with the app.

In conclusion, although there is a huge variety of means of developing mobile applications for online shopping, React Native is a trustworthy tool that offers many useful features and makes the development process easier.

7 References

Adetunji, O. *et al.* (2020) 'Dawning of Progressive Web Applications (PWA): Edging Out the Pitfalls of Traditional Mobile Development', 68(1).

Ain, Q.U., Missen, M.M.S. and Prasath, S. (2023) 'LRAP: Layered Ring Based Adaptive and Personalized Usability Model for Mobile Commerce Apps', *International Journal of Interactive Mobile Technologies (iJIM)*, 17(12), pp. 74–93. Available at: https://doi.org/10.3991/ijim.v17i12.37995.

Alsaadi, H.A. *et al.* (2021) 'Factors that affect the utilization of low-code development platforms: survey study', *Revista Română de Informatică și Automatică*, 31(3), pp. 123–140. Available at: https://doi.org/10.33436/v31i3y202110.

Annual mobile app downloads worldwide by store 2026 (no date) *Statista*. Available at: https://www.statista.com/statistics/1010716/apple-app-store-google-play-app-downloads-forecast/ (Accessed: 2 March 2024).

C G, T. and Devi, A. (2021) 'A Study and Overview of the Mobile App Development Industry', *International Journal of Applied Engineering and Management Letters*, pp. 115–130. Available at: https://doi.org/10.47992/IJAEML.2581.7000.0097.

Chavan, M., Bhatkar, M. and Muley, K. (2022) 'Progressive Web Apps vs Responsive Web Apps', *International Journal of Advanced Research in Science, Communication and Technology*, pp. 211–214. Available at: https://doi.org/10.48175/IJARSCT-5668.

Cross-platform mobile frameworks used by global developers 2022 (no date) *Statista*. Available at: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (Accessed: 2 March 2024).

'Difference between Swift and Objective C' (2020) *GeeksforGeeks*, 25 December. Available at: https://www.geeksforgeeks.org/difference-between-swift-vs-objective-c/ (Accessed: 2 March 2024).

Fentaw, A.E. (2020) 'Cross platform mobile application development: a comparison study of React Native Vs Flutter'.

Hjort, E. (2020) 'Evaluation of React Native and Flutter for cross-platform mobile application development'.

How to Distribute an iOS App Without Using App Stores (2023) *BuildFire*. Available at: https://buildfire.com/ios-app-distribution-without-app-store/ (Accessed: 2 March 2024).

Inc, A. (no date) *Swift - Apple Developer*. Available at: https://developer.apple.com/swift/ (Accessed: 2 March 2024).

Inc, A. (no date) *Swift.org*, *Swift.org*. Available at: https://swift.org (Accessed: 2 March 2024).

Introduction to Mobile Application Development | IBM (no date). Available at: https://www.ibm.com/topics/mobile-application-development (Accessed: 2 March 2024).

Jayvir, S. (2023) 'Cross Platform Mobile App Development 2023: The Ultimate Guide'.

jmango 360 (no date) 'Mobile App versus Mobile Website Statistics: 2020 and beyond', *JMango360*. Available at: https://jmango360.com/mobile-app-vs-mobile-website-statistics/ (Accessed: 2 March 2024).

Low-Code/No-Code: The Future of Development (no date) *SAP*. Available at: https://www.sap.com/products/technology-platform/low-code/what-is-low-code-no-code.html (Accessed: 2 March 2024).

Majchrzak, T.A., Biørn-Hansen, A. and Grønli, T.-M. (2018) 'Progressive Web Apps: the Definite Approach to Cross-Platform Development?'

Mobile eCommerce Statistics (2024): User & Revenue Growth (no date) *Capital One Shopping*. Available at: https://capitaloneshopping.com/research/mobile-ecommerce-statistics/ (Accessed: 2 March 2024).

Nathan, B. (2023) 'React Native: A native code integration perspective'.

Oltrogge, M. *et al.* (2018) 'The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators', in *2018 IEEE Symposium on Security and Privacy (SP)*. *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 634–647. Available at: https://doi.org/10.1109/SP.2018.00005.

Roma, P. and Ragaglia, D. (2016) 'Revenue models, in-app purchase, and the app performance: Evidence from Apple's App Store and Google Play', *Electronic Commerce Research and Applications*, 17, pp. 173–190. Available at: https://doi.org/10.1016/j.elerap.2016.04.007.

Salz, P.A. and Moranz, J. (2013) *The Everything Guide to Mobile Apps: A Practical Guide to Affordable Mobile App Development for Your Business*. Simon and Schuster.

Shevtsiv, N.A. and Striuk, A.M. (2020) 'Cross platform development vs native development'.

Swilley, E. (2015) *Mobile Commerce: How It Contrasts, Challenges, and Enhances Electronic Commerce*. New York, UNITED STATES: Business Expert Press. Available at: http://ebookcentral.proquest.com/lib/czup/detail.action?docID=4388929 (Accessed: 25 July 2023).

Tang, A.K.Y. (2016) 'Mobile App Monetization: App Business Models in the Digital Era', *International Journal of Innovation, Management and Technology*, pp. 224–227. Available at: https://doi.org/10.18178/ijimt.2016.7.5.677.

Tarasewich, P., Nickerson, R.C. and Warkentin, M. (2002) 'Issues in Mobile E-Commerce', *Communications of the Association for Information Systems*, 8. Available at: https://doi.org/10.17705/1CAIS.00803.

The Basics | Documentation (no date). Available at: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/thebasics/ (Accessed: 2 March 2024).

Use Progressive Web Apps - iPhone & iPad - Google Chrome Help (no date). Available at: https://support.google.com/chrome/answer/9658361?hl=en&co=GENIE.Platform%3DiOS (Accessed: 2 March 2024).

What is Mobile App Development? | Microsoft Azure (no date). Available at: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-mobile-app-development (Accessed: 2 March 2024).

8 List of figures, tables and images

8.1 List of figures

Figure 1: Mobile share of global e-commerce revenue, Source: capitaloneshopping.com, 2024 12
Figure 2: Most used cross-platform frameworks, Source: www.statista.com 15

8.2 List of tables

Table 1: Comparison of mobile app types 18
Table 2: Assessment of MCDA criteria 32
Table 3: 5 points scale 33
Table 4: Decision matrix 33
Table 5: Normalized decision matrix 34
Table 6: Weighted normalized decision matrix 34
Table 7: Ranking of alternatives 35

8.3 List of images

Image 1: Wireframe, Source: own processing 36
Image 2: Homepage and Details page, Source: own processing 38
Image 3: Cart page and Payments page, Source: own processing 38

9 Appendix

Link to GitHub repository: https://github.com/arumy191/Blossom_ecommerce_app