Roope Stenhammar

Selecting Development Approach and Framework for Mobile Lodging Service Application

Master’s Thesis
Espoo, December 31, 2020

Supervisor: Prof. Mario Di Francesco, Aalto University
Advisor: Prof. Mario Di Francesco
In this thesis, a solution to aid developers in selecting a proper framework was researched from the perspective of mobile application development. Goal of the thesis was to overcome the difficulty of choice developers may encounter during software engineering. Objectives of the thesis were to discover a pattern and criteria regarding framework selection, to select the most suitable framework for a lodging service-related mobile application called My Hotel for Android and iOS, and to develop the application for evaluating suitability of the selected framework.

The following discoveries were made in this work: first, 9 different challenges were identified regarding mobile application development of which user experience (UX) and fragmentation of source code were most notable. Second, it was discovered that to avoid at least some of the challenges a specific development approach (DA) should be selected. A total of 7 DAs were discovered of which 5 can be considered practical for modern cross-platform applications. Third, a total of 27 selection criteria for framework selection in mobile application development were recognized. Fourth, another challenge regarding selection of the multiple-criterion decision-making (MCDM) method, such as the weighted sum method (WSM), was encountered. Fifth, for the case of My Hotel, hybrid DA and Apache Cordova framework were identified as the most suitable choices.

As a conclusion more research regarding the used selection process and tool selection methods in general was suggested. Additionally, creation of a guide or a web application for aiding in decision-making in similar situations was recommended.

Keywords: development approach, development approaches, mobile, mobile application development, mobile application framework, software engineering, software framework, software frameworks, web application, web applications

Language: English
Tekijä: Roope Stenhammar

Työn nimi: Kehityslähestymistavan ja ohjelmistokehyksen valinta mobiilille majoituspalvelusovellukselle

Päiväys: 31. joulukuuta 2020

Pääaine: Computer Science

Valvoja: Prof. Mario Di Francesco, Aalto-yliopisto

Koodi: SCI3042

Ohjaaja: Prof. Mario Di Francesco

Tässä diplomityössä etsittiin ratkaisua ohjelmistokehittäjien auttamiseksi ohjelmistokehyksen valitsemisessa mobiilisovelluskehityksen näkökulmasta. Tarkoituksena oli löytää ratkaisu valinnanvaikeuteen, minkä kehittäjät saattavat kohdanta ohjelmistokehitysprosessin aikana. Tavoitteena oli löytää kaava ja kriteerit ohjelmistokehyksen valinnalle, valita sopivin kehys majoituspalvelusovellukselle nimiä My Hotel Android- ja iOS-laitteille, sekä kehittää itse sovellus valitun ohjelmistokehyksen sopivuuden arvioimiseksi.


Johtopäätöksensä työssä ehdotettiin jatkokutkimusta työssä käytetylle valintaprosessille sekä työkalujen valintametodeille yleistä. Lisäksi päättytiin myös suosittelemmaan oppaan tai web-sovelluksen luomista päätöksenteon avustamisessa vastaavissa tilanteissa.

Asiasanat: kehityslähestymistapa, kehityslähestymistavat, mobiili, mobiilisovelluskehitys, mobiilisovelluskehys, ohjelmistokehitys, ohjelmistokehys, ohjelmistokehykset, web-sovellus, web-sovellukset

Kieli: Englanti
Acknowledgements

I wish to thank my advisor Mario Di Francesco for his patience and positivity during the long journey of writing this thesis. Additionally, I would like to thank all my friends and family for their support and understanding as well as my employers HKL and Veikkaus for their flexibility.

One thing I am also very grateful for is all the years spent at Aalto University which has allowed me to get introduced to new, inspiring people, in addition to allowing me to get to know the wonderful world of science!

Espoo, December 31, 2020

Roope Stenhammar
### Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABR</td>
<td>Application Boundary Resources; part of PBR</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AVD</td>
<td>Android Virtual Device</td>
</tr>
<tr>
<td>CLI</td>
<td>Command-Line Interface</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets; a language for describing styling of a web page</td>
</tr>
<tr>
<td>DA</td>
<td>Development Approach; an approach for implementing a mobile application</td>
</tr>
<tr>
<td>DBR</td>
<td>Development Boundary Resources; part of PBR</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>JS</td>
<td>JavaScript; a scripting language</td>
</tr>
<tr>
<td>LC</td>
<td>Lodging Company; a hotel, for example</td>
</tr>
<tr>
<td>MC</td>
<td>Mobile Client</td>
</tr>
<tr>
<td>MCDM</td>
<td>Multiple-Criterion Decision Making</td>
</tr>
<tr>
<td>MVP</td>
<td>Minimum Viable Product</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PBR</td>
<td>Platform Boundary Resources; consists of ABR, DBR and SBR</td>
</tr>
<tr>
<td>SBR</td>
<td>Social Boundary Resources; part of PBR</td>
</tr>
<tr>
<td>SPA</td>
<td>Single-Page Application</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>UX</td>
<td>User eXperience</td>
</tr>
<tr>
<td>WSM</td>
<td>Weighted Sum Method</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
Contents

Abbreviations and Acronyms 5

1 Introduction 8
   1.1 Motivation .............................................. 8
   1.2 Objectives of the thesis ................................. 9
   1.3 Structure of the thesis .................................. 9

2 Application overview 10
   2.1 Idea .................................................... 10
   2.2 Preliminary requirements of the minimum viable product (MVP) 11
   2.3 Implementation plan ..................................... 12

3 Background 14
   3.1 Mobile application development .......................... 14
      3.1.1 Challenges ........................................ 14
      3.1.2 Development approaches ............................ 15
          3.1.2.1 Browser-based development approaches (DAs) 17
          3.1.2.2 Compiler-based DAs ............................ 18
          3.1.2.3 Other DAs .................................... 20
   3.2 Technology selection research ............................ 22

4 Methods 26

5 Implementation 33
   5.1 Additions to client requirements and selection criteria ........ 33
   5.2 Definitions for WSM-based MCDM .......................... 35
   5.3 Development approach selection ............................ 36
   5.4 Framework selection .................................... 39
   5.5 Application development with Cordova ..................... 43
      5.5.1 Challenges of Cordova .............................. 43
   5.6 Development of My Hotel ................................. 44
6 Evaluation
  6.1 Success of My Hotel application ............... 51
  6.2 Apache Cordova as the selected framework .......... 52
  6.3 Used selection process .................. 55

7 Conclusions ........................................... 58

A Complete list of requirements .......................... 63

B Getting started with Apache Cordova .................. 65

C DA literature: pros and cons .......................... 71
Chapter 1

Introduction

This is the introductory chapter of this thesis. In this chapter, motivation, research objectives and the structure of the thesis are described.

1.1 Motivation

Since the beginning of time, mankind has been aching to complete tasks more and more efficiently. Especially tasks regarded the most mundane as well as those regarded the most critical have been streamlined as much as possible. This phenomenon can be regarded as a force of nature, acting like evolution. After thousands of years, this “force” has applied to IT industry as well. As a result, a vast amount of different tools, libraries and frameworks among all other technologies have emerged.

Mostly the trend of emerging tools has been positive. It has allowed programmers to develop software more efficiently as all components are not necessary to be re-invented. Additionally, the end result may be more secure and reliable. Furthermore, the amount of tools involves competition with other tools which encourages their maintainers to keep them updated, at least in those cases where their maintainers want their tools to be the most favored.

However, as this phenomenon has been unfolding for years, there is at least one difficulty which probably most of the software engineers and developers have already run into: the difficulty of choice. Due to the ever-increasing, vast amount of possibilities, and the limits of cognitive capabilities of humans, developers will be facing more and more challenges in selecting the right tools for the job. Whereas in the past tools could be chosen by bare vision, nowadays, at least in the field of computer science, one is obliged to search for information through the internet. Documentation, articles and
questions asked by other developers on sites such as Stack Overflow have to be read to achieve as clear comprehension as possible of the possibilities.

Considering the facts that the amount of software development-related technology continues to emerge with an increasing pace and the cognitive capabilities of the human brain are limited, some actions must be taken. Some guidelines and general heuristics should be formed to allow developers to be what they desire probably the most: as efficient as possible. After all, for software developers, focus should be on development instead of inspection and evaluation of every single piece of information of every single existing tool.

As there are limitless scenarios where developer could face this dilemma of choice, one thesis is clearly not sufficient to discuss them all and a specific focus is required. Since mankind has become, and is still becoming, more dependant of mobile devices, this thesis specifically discusses the issue of selecting the best possible framework for a mobile application that aims to act as an interface between a lodging company and their customers. However, despite the narrow scope of the application, the guidelines this thesis aims to formulate could probably help decision-making in other fields, such as web development, also.

1.2 Objectives of the thesis

This thesis has three objectives. First, the thesis aims to discover a pattern in framework selection: the criteria that should be used at the moment of selection. Second, it utilizes the discovered criteria to use them to select the most suitable framework for a mobile application. Third, to develop an application by using the framework and evaluate its suitability.

1.3 Structure of the thesis

In Chapter 2, overview and preliminary requirements of the application to be implemented are discussed. Chapter 3 addresses the background of mobile application development and technology selection. Chapter 4 describes the methods that will be used to achieve the goals of this thesis. In Chapter 5, the development process and the end result of the mobile application are discussed. Then, in Chapter 6 the application, framework and used selection process are evaluated. Finally, in Chapter 7 the conclusion of the thesis is described.
Chapter 2

Application overview

In this chapter, requirements of the application called My Hotel to be created are discussed on a general level to comprehend choices regarding framework selection in Section 5.4. Additionally, plan regarding the implementation of the application is examined briefly.

2.1 Idea

During the past couple of years a large amount of companies have emerged to offer services for booking hotel or motel rooms. These include companies such as Expedia Inc., The Priceline Group, The HRS Group, Momondo A/S and Nustay A/S. As a result, there are multiple different websites for finding the most suitable room.

However, at the moment of writing, there seems to be no real solution for acting as an interface between the hotel and its staying customers. Some hotels seem to offer a web-based portal which is accessed via a wireless local area network that is also offered by the hotel. The current widely used solution is not as efficient and usable as it could be, however. For example, the customer has to be in the vicinity of their hotel to access the portal which restricts the user. Additionally, the portal-based applications do not usually enhance the cost-efficiency or service quality of the hotel as well as they could. The applications could, for instance, allow the customer to set a "Do not disturb" flag for their room. This would reduce workload of the hotel cleaners as they would not be required to inspect the door knobs of each room.

As there seems to be a market gap for a solution that allows already staying customers to communicate with their host more efficiently, a mobile application named My Hotel will be developed in this thesis dedicated to this
CHAPTER 2. APPLICATION OVERVIEW

purpose. To match the current standards of software development, the application will be based on a framework that is one of the discussed frameworks in Section 5.4 and selected by criteria defined in Table 3.1. Additionally, the application should cover the most used devices and resolutions, as the device range is unrestricted due to the fact that customers and staff of lodging companies may use any devices. As devices based on Google Android and Apple iOS operating systems are currently the most used ones [14], devices utilizing these operating systems (OS) should be supported at least.

2.2 Preliminary requirements of the minimum viable product (MVP)

Before framework selection and application development, preliminary requirements for the application have to be determined. The definition of done for this MVP regarding customers of lodging companies (LC) will be as follows. Customers using the application should be able to do the following tasks: view general information regarding the LC, such as name and address; view room-specific information, such as surface area and maximum amount of guests; to create, read, update and delete their own service requests regarding their accommodation or other services offered by their LC; and enable to decline from room service temporarily by setting a "Do not disturb flag" on. These requirements are listed in Table 2.1 with IDs MC-2.1, MC-2.2, MC-2.3 and MC-2.4, respectively.

As most of the customer interaction with the application involves interacting with the LC staff as well, MVP of the application should also allow the staff members to use the application as follows: view their personal tasks (MC-1.1 of Table 2.1); and both view customer-created service requests and update them accordingly (MC-1.2 of Table 2.1). As there should be a separation between employees and customers of LCs, requirements MC-1.0 and MC-2.0 are needed respectively. Additionally, the user interface (UI) should be graphical (GUI), thus the requirement MC-3.0. Due to the fact that the application should work on Google Android and Apple iOS, it should have a rather good and developer-friendly cross-platform support (MC-4.0 of Table 2.1).

To enable the application to fulfill all the listed requirements, a back-end solution is needed as well. Due to the fact that this thesis focuses on mobile applications, requirements regarding the server are omitted. To summarize, MC allows users to view basic LC information, commence simple tasks, create simple service requests for the staff and hotel staff to execute tasks designated
to them. A more detailed description of the requirements for MVP regarding the client is illustrated in Table 2.1.

2.3 Implementation plan

The development will be done by a team of 2 members according to a plan discussed in this section. Members of the team have some experience of mobile application development, yet are not familiar with any specific frameworks in that field. Thus, the team aims to get familiar with the documentation of the framework at the time of its selection and creates an example application first.

The development itself will be commenced during a suitable time window, preferably during a weekend. The following tools will be used during the development: Android Studio for programming, Git for version control and GitHub as the remote repository of the project. Due to the fact that some of the tools are known only after the framework has been selected in Section 5.4, the list of tools shall be considered only preliminary.
<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC-1.0</td>
<td>View for employees</td>
<td>A view designated to employees of LC’s.</td>
</tr>
<tr>
<td>MC-1.1</td>
<td>Employee task list</td>
<td>A view that allows employees to read and update their tasks.</td>
</tr>
<tr>
<td>MC-1.1.1</td>
<td>Employee task list real-time CMS updates</td>
<td>The task list view for employees will be updated in real-time as set in the company CMS.</td>
</tr>
<tr>
<td>MC-1.2</td>
<td>Employee service requests</td>
<td>A view that allows company employees to read and update service requests created by customers.</td>
</tr>
<tr>
<td>MC-1.2.1</td>
<td>Real-time CMS updates for employee service requests</td>
<td>The service requests view for employees will be updated in real-time as per requests created by customers.</td>
</tr>
<tr>
<td>MC-2.0</td>
<td>View for customers</td>
<td>A view dedicated to staying customers.</td>
</tr>
<tr>
<td>MC-2.1</td>
<td>General LC information</td>
<td>A view for customers that allows them to see general information about the LC.</td>
</tr>
<tr>
<td>MC-2.1.1</td>
<td>Details of LC information</td>
<td>General information of the LC should include the following details: name, address, coordinates, phone number.</td>
</tr>
<tr>
<td>MC-2.2</td>
<td>Room-specific information</td>
<td>View for the customer for inspecting details about their room.</td>
</tr>
<tr>
<td>MC-2.2.1</td>
<td>Details for room specific information</td>
<td>The following details should be implemented for room-specific information: surface area, minibar, maximum amount of guests, floor.</td>
</tr>
<tr>
<td>MC-2.3</td>
<td>Service request view for customers</td>
<td>The customers should be able to create, read, update and delete (CRUD) service requests for employees of their LC.</td>
</tr>
<tr>
<td>MC-2.3.1</td>
<td>Easy extension to stay (service request)</td>
<td>The customers should be able to request an extension to their stay as intuitively and easily as possible.</td>
</tr>
<tr>
<td>MC-2.4</td>
<td>&quot;Do not disturb&quot; flag for room</td>
<td>Customers should be able to set a &quot;Do not disturb&quot; flag for their room.</td>
</tr>
<tr>
<td>MC-3.0</td>
<td>Graphical user interface (GUI)</td>
<td>The MC should implement as intuitive and modern-looking GUI as achievable.</td>
</tr>
<tr>
<td>MC-4.0</td>
<td>Cross-platform support</td>
<td>The MC should be operable on both Google Android and Apple iOS systems.</td>
</tr>
</tbody>
</table>

Table 2.1: Requirements for MVP of the mobile client (MC)
Chapter 3

Background

This chapter reviews mobile application development in general. Additionally, research and literature regarding framework selection are discussed for selecting a framework later in Section 5.4.

3.1 Mobile application development

In this section, mobile application development is discussed in general. First, the related challenges are reviewed. Then, as a partial solution to the challenges, different development approaches are examined.

3.1.1 Challenges

Before analysing the problem of selecting a framework for a mobile application, a review of different related challenges is needed. As mobile applications differ rather greatly from other types of applications, there are a multitude of factors to be taken into consideration. For example, there exists currently numerous different OS versions for both Google’s and Apple's platforms. For Google Android, there is version 9.0 Pie with a 38.57% market share, version 10.0 10 with the market share of 12.45% and 8.1 Oreo with the share of 11.83% [12]. For Apple iOS, current version shares are 74.24%, 8.55% and 2.57% for versions 13.3, 12.4 and 13.1, respectively [13]. As another example, there are a multitude of different mobile device resolutions. Currently, the distribution for top-3 resolutions is as follows: 18.7% for 360x640, 7.34% for 375x667 and 6.76% for 414x896 [15]. Consequently, the application, and the framework it is based on, should be as extensive and adjustable as possible to cover these possibilities.
CHAPTER 3. BACKGROUND

According to Ahmad et al., there are in fact nine different types of challenges regarding mobile application development [18]. These include, in the order of frequency, user experience (UX), fragmentation, compatibility, testing, change management, reuse of code, lack of tools support, lack of expertise and security [18]. The challenges are listed below with their descriptions and frequencies for further reference.

- UX: overall experience of an application for a user [21]. Frequency: 54%

- Fragmentation: fragmentation of the device market which obliges companies and developers to create the same application multiple times for different platforms [25]. Additionally, fragmentation may occur within a specific platform due to device variance [26]. Frequency: 46%

- Compatibility: compatibility of the application across different runtime systems such as browsers [25] or platforms [18]. Frequency: 46%

- Testing: lack of testing automation tools, for example [18]. Frequency: 44%

- Change management: developing new features, commencing bug fixes or reacting to changes in operating system versions, for example [18]. Frequency: 27%

- Reusability of code: challenge of using the same code for multiple platforms, for example [18]. Frequency: 15%

- Lack of tools support: insufficient tools or the need for multiple different tools for commencing the same task [18]. Frequency: 14%

- Lack of expertise: lack of proficiency of different platforms or developer tools [18]. Frequency: 14%

- Security: security of the application [18]. Frequency: 4%

3.1.2 Development approaches

To avoid some challenges, especially fragmentation, one can select a specific development approach (DA) for developing the mobile application. In their paper, Ahmad et al. mention three different DAs regarding the creation of mobile applications: native, web and hybrid [18]. The first one, native DA, is developed directly onto a specific platform such as Google Android [18].
CHAPTER 3. BACKGROUND

This implies, however, that the same application must be developed for multiple platforms if, for example, iOS users are also wanted to be targeted. Thus, fragmentation has been mentioned as the greatest challenge in native application development [18]. The second approach, web DA, can be utilized to develop an application only once by using web technologies [18] such as HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript (JS). Consequently, development time and cost are reduced as the development may be commenced for all platforms at once. Unfortunately, applications developed with web DA are unable to access hardware-specific features of the devices such as sensors or camera [18]. The third approach, hybrid DA, combines the advantages of web-based and native DAs. Browser engine of the device is utilized to embed the web page in a native web container [29]. This web container is referred to as WebView for Android and UIWebView for iOS [29]. To access the hardware-specific functionalities, an abstract JavaScript bridge is utilized [29]. This has a negative impact on performance, though, as execution is done in the browser engine [29].

Some studies suggest there are even more DAs for comprising a mobile application. Studies by both Latif et al. and Bjørn-Hansen et al. mention at least three new DAs: interpreted, cross-compiled and model-driven [29] [19]. Additionally, an approach called progressive web application (PWA) is also a feasible alternative [20] [19]. In interpreted DA, a common language is used for writing UI-related code and generate its native equivalent for each platform [29]. Native features are accessed by utilizing an abstract layer that interprets the code at runtime [29]. As with the interpreted DA, a common language is used in cross-compiled DA as well [29]. In this case, though, platform-specific compilers are utilized instead of run-time interpretation to create a native application for each platform [29]. Model-driven DA, on the other hand, is derived from model-driven architecture specified by Object Management Group (OMG) [29]. The objective is to avoid dealing with low-level technicalities by using a platform independent model (PIM) [29]. The PIM is then transformed into native code by platform-specific model and model-to-text transformation [29]. The fourth alternative DA, PWA, is practically an advanced version of the ordinary web-based DA [20], thus rendering it as obsolete. The use of PWA-based allows, for example, offline support and background synchronization for the application [20].

As some DAs are somewhat related to each other, the approaches are divided into four different categories: browser-based, compiler-based, interpreter-based and native DAs. However, due to the fact that the categories "native" and "interpreter-based" represent basically DAs of the same names, only three categories are used in this thesis: browser-based, compiler-based and other DAs. The following DAs are categorized as browser-based due to
the fact that they are based on the browser of the client device: web-based and PWA-based DAs. Compiler-based DAs consist of cross-compiled, hybrid and model-driven DAs as all of them are compiled or transformed to a native application eventually. Other DAs, on the other hand, consist of native and interpreted DAs as the first one can be considered to be the predecessor of all the DAs for mobile application development and the latter one uses an interpreter instead of a compiler for access to native functionalities. One could argue that the hybrid DA should be considered as browser-based as it resembles web-based and PWA-based DAs. Another possibility would be to have hybrid and interpreted DAs under the same category such as "Bridge-based" as both utilize a bridge to access native features. However, in this thesis, hybrid DA is classified as a compiler-based DA as the implementation will be wrapped into a native application and is deployable via app stores in the end [19], thus being a compiled application.

All of the known DAs are categorized and described in more detail in Sections 3.1.2.1, 3.1.2.2 and 3.1.2.3 with their respective advantages and disadvantages that have been mentioned in previous scientific literature. Most of the advantages and disadvantages have been referenced as word-by-word citations from their respective sources as Section 5.3 discusses them in more detail for choosing a proper DA for My Hotel application. Thus, some advantages and disadvantages may be redundant in the lists below. Regarding disadvantages mentioned by Ahmad et al. in [18] concerning web, hybrid and native DAs, only those with a relative frequency greater than 21% in Table 5 have been mentioned. More details regarding the interpretation of the aspects mentioned in literature can be seen in Appendix C for each DA.

3.1.2.1 Browser-based development approaches (DAs)

As two of the encountered DAs were identified to be based on the use of a browser application such as Google Chrome, are those DAs listed here. First of the approaches represent a more traditional web application whereas the second one can be considered its successor.

- **Web**: the use of web technologies to create an application [18]. The development process of this approach is rather simplistic as can be seen from Figure 3.1. Advantages of this approach include: cross-platform [29] [18]; no application updates [29]; and reduced development time and cost [18]. Disadvantages of this DA, on the other hand, are: partial access to native functionalities [29] [18]; low performance [29]; not publishable on application stores [29]; compatibility [18]; UX [18]; and testing [18].
Progressive web application (PWA): usage of modern web technologies that allow the use of some additional features such as support offline usage and background synchronization, for instance [20]. The development process is similar to that of the ordinary web approach which is shown in Figure 3.1. The advantages of this approach are: cross-platform [19]; advantages of web-based DA [19]; native look and feel achievable [19]; and offline use possible [19]. There is, however one disadvantage to this DA: UX on iOS [19].

3.1.2.2 Compiler-based DAs

Three of the discovered DAs were recognized to be based on code generation or compilation: cross-compiled, hybrid and model-driven. Consequently, all the DAs listed below are full-fledged native applications at the end of development process.

Cross-compiled: a DA in which the code for all targeted platforms is written once in a common language such as C# [29] [19] [22]. Afterwards, the written code will be compiled for all the desired platforms [29] [19] [22]. Thus, in the end, the developed application will be a native application for each targeted platform [29] [19] [22]. Consequently, the process resembles that of native DA as may be seen from Figure 3.2. However, in this approach, the application will be written only once whereas in native DA it would be written for each platform separately as mentioned in Section 3.1.2.3. Advantages of this approach are: cross-platform [29] [19]; native performance [29] [19]; access to native functionalities [29]; and native UI [29] [19]. In contrast of the mentioned advantages, there is one drawback as well: only partial access to native functionalities [29].

Hybrid: the use of web technologies to create an application. A JavaScript bridge is utilized to access hardware functionalities,
native web container, a WebView component, is used for rendering as is illustrated in Figure 3.3 [29] [19]. Advantages in the case of hybrid DA are quite abundant as follows: cross-platform [29] [19]; advantages of web-based DA [29]; advantages of native DA [29] [19]; publishable on application stores [29] [19] [18]; access to native functionalities [29] [19] [18]; popularity [19]; native UI achievable [19]; low effort [19]; and alternative frameworks [19]. Despite the listed advantages, there are a couple of disadvantages that one should be aware of, however: low performance [29]; native UI and UX not guaranteed [29] [19]; and requires additional tools [19].
• **Model-driven**: development done with the use of a platform independent model which will be converted into native code with the help of platform-specific models and transformations [29]. The development process of this approach is represented in Figure 3.4. Advantages that can be expected to achieve with this DA are as follows: cross-platform [29] [19]; high abstraction level [29] [19]; native performance [29] [19]; native UI [29]; and customizability [29]. Disadvantages, on the other hand, include: maintainability [29]; limited programmability [29]; requires expertise despite having a high abstraction level [29]; cross-platform development [29]; popularity [19]; and framework-lock [19].

![Diagram](image)

Figure 3.4: Model-driven DA [19] [29]

### 3.1.2.3 Other DAs

Rest of the DAs are listed below. These include native and hybrid DAs. Due to the fact that native DA does not have inherently cross-platform support quite logically it is considered as other DA in this thesis. As interpreted DA differs from other DAs by having the application based on an interpreter, it is deemed as other DA as well.

• **Native**: development of the application with native code platform-specifically [18]. As can be seen from Figure 3.5, this DA implies that to create an application for all platforms one is obliged to develop the software as many times as there are targeted platforms, thus multiplying the amount of work and causing fragmentation that was mentioned in Section 3.1.1. Advantages of this approach are, quite logically, the following: native functionalities [18]; native performance [18]; publishable on application stores [18]. However, disadvantages of this approach
are: increased development time and cost [18]; fragmentation [18]; testing [18]; and change management [18].

![Diagram of Native DA](image)

Figure 3.5: Native DA

- **Interpreted**: the use of a common language to write the UI-related code; native features are accessed via an abstract layer at run-time [29]. This approach is quite similar to that of hybrid DA, yet this one does not depend on a WebView component for rendering [19]. Figure 3.6 describes the development approach in more detail. Advantages of the interpreted DA are: cross-platform [29] [19]; native UI [29] [19]; and native functionalities [29] [19]. Disadvantages of this DA include: dependence on the development environment [29]; updateability [29]; lower performance [29] [19]; framework-lock [19]; and low popularity [19].

![Diagram of Interpreted DA](image)

Figure 3.6: Interpreted DA [19] [29]
The advantages and disadvantages mentioned in the lists above appear to be mostly concordant with the Tables IV-VI written by Delia et al. [23]. However, there are some notable differences. For example, their comparison considers a greater amount of features regarding each approach. Model-driven approach is unfortunately not discussed in their paper. Moreover, the DA comparison is done only regarding two tools of each approach except for web-based approach or, as they refer to it, mobile web applications [23].

In summary, there are seven different DAs, or six if the web-based one is considered to be superseded by PWA-based DA. The approach should be chosen carefully as each have their own advantages and disadvantages. Some challenges can be avoided by choosing a proper DA.

### 3.2 Technology selection research

For assisting with the decision of selecting the most suitable framework a review of existing literature is required. Although there seems to be rather few pieces of literature regarding the topic, fortunately Myllärniemi et al. have commenced some relevant research [31].

In their study, Myllärniemi et al. had two research questions: "How do platform boundary resources support or hinder the adoption and continuous use of a framework?" and "What factors support the customer loyalty of application developers after initial framework use?" [31]. In this context, platform boundary resources (PBR) means application boundary resources (ABR), development boundary resources (DBR) and social boundary resources (SBR) which affect developer experience of a framework [31]. For answers, 9 practitioners were interviewed and a complete longitudinal survey on 51 students was commenced [31]. Main focus was on a framework called Qt that is used for cross-platform mobile application development, embedded application development and graphical user interface development as it is widely accessible [31].

For the first research question, the results were as follows: regarding ABR, application programming interface (API) capabilities, programming language and being an open-source project are important factors [31]. Specifically, programming language should preferably be a language that is already known by the developer, and the framework should have a long enough lifespan (continuity) [31]. In addition to continuity, source code of the framework should be as visible as possible [31]. For good continuity and visibility, the framework should be preferably an open-source project [31]. Concerning DBR, development and deployment tools should be easy to install and use, and they should not include unnecessary libraries [31]. Additionally, if a sufficient editor and
documentation are provided by the platform owner, development efficiency is increased. For SBR, peer experience, active community and tutorial coverage are affecting factors [31]. Also, documentation regarding the framework should be as extensive and developer-focused as possible [31].

Regarding the second research question of Myllärniemi et al., there were five observations. First, enjoyable development with a framework increases loyalty towards it of the developer [31]. Second, the framework should have good usability to achieve developer loyalty [31]. Third, quality of the boundary resources should be as good as possible as well [31]. Fourth, searching for information regarding the framework and its installation should not be time-consuming [31]. Fifth, lower usability and enjoyment are caused by difficulties encountered during the use of a framework [31].

In addition to the research commenced by Myllärniemi et al., Gholamshahi and Hasheminejad have researched software component identification and selection for component-based software development [24]. Although their research regarding component selection concerns the choice between procuring a commercial off-the-shelf solution or developing a component in-house, the same principles apply to choosing a suitable framework. According to their paper, the top-5 criteria for component selection have been overall cost, reliability, number of components, delivery time and functionality, respectively [24]. Whereas the research commenced by Myllärniemi et al. was a developer-focused longitudinal survey [31], the research by Gholamshahi and Hasheminejad was a literature review aiming to discover a solution for the component selection problem [24]. According to the conclusion of their paper, multi-objective optimization methods seem to be best options for component selection [24].

Whereas Myllärniemi et al. concentrated on developer-experience regarding frameworks and Gholamshahi and Hasheminejad focused on general component selection, Masi et al. have constructed a framework for assisting in technology selection regarding mobile applications [30]. In their paper, there are two types of requirements for the decision making: developers' needs and device features [30]. Developers' needs consist of multiple requirements, such as access to native features, performance, cross-platform support, upgradability, deployability on application stores, development speed, complexity, aesthetics, and development cost [30]. Unfortunately, some criteria such as small footprint had to be excluded as they have not been described sufficiently enough. Finally, rather presumably, device features consist of sensors and other features such as network access and Bluetooth [30]. In this thesis, these device features are considered part of "access to native features" concerning developers' needs.

Rather fortunately, one study commenced by Delia et al. has already com-
mented DA and framework comparison by using three different categories of
to features: non-functional, developer-centric and software project management
related features [23]. These categories consist of a total of 23 features [23].
Non-functional features consist of the following feature comparisons: UX,
UI, performance, installation mode, battery use, disk usage/app size, image
rendering and initial boot time [23]. Developer-centric features, on the other
hand, consist of integrated development environment (IDE), programming
languages, open source/license and cost, GUI design, learning curve and ac-
cess to advanced device specific [23]. The last category, software project
management, contains the following features: target platforms, speed and
cost of development, maintainability, degree of maturity/long-term feasibil-
ity, mobile apps categories, offline usage, code reuse, distribution/access and
potential users [23]. As most of the aforementioned features are already dis-
cussed earlier, the most relevant ones still missing are UX, learning curve
and maintainability, at least according to the list of challenges in Section 3.1
that mentions UX, change management and lack of expertise.

Surprisingly, none of the selected pieces of literature state that testability
would be a significant factor regarding the selection process although testing
is in top-3 of the list of challenges in Section 3.1. Only Gholamshahi et al.
have testing time and testability mentioned in Table 14 as rather negligible
criteria [24]. Due to the fact that testing is a significant challenge and it has
been mentioned, testability will be considered an important aspect in this
thesis as well.

To summarize, the most relevant criteria for selecting a framework regard-
ing developer-friendliness are API capabilities, well-known programming lan-
guage, open-sourcedness, ease of installation of development and deployment
tools, ease of use of development and deployment tools, source code visibil-
ity, source code continuity, extensive documentation, editor support, peer
experience, community activity and tutorial coverage. Additionally, more
general criteria including overall cost, reliability, number of components, ef-
efct on delivery time and functionality should also be taken into account.
Regarding mobile application development specifically, at least following fac-
tors would be advisable to be taken into account as well: access to native
features, performance, cross-platform support, deployability on application
stores, complexity and aesthetics. In addition to the aforementioned selec-
tion criteria, UX, learning curve, maintainability and testability should be
taken into consideration as well. These selection criteria (SC), 26 in total,
are enumerated in Table 3.1 for later reference and used in Chapter 5 for
selecting the most suitable framework for the lodging company application.
<table>
<thead>
<tr>
<th>ID</th>
<th>Selection criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>API capabilities</td>
</tr>
<tr>
<td>SC2</td>
<td>Programming language is well-known</td>
</tr>
<tr>
<td>SC3</td>
<td>Open-sourcedness</td>
</tr>
<tr>
<td>SC4</td>
<td>Ease of installation of development and deployment tools</td>
</tr>
<tr>
<td>SC5</td>
<td>Ease of use of development and deployment tools</td>
</tr>
<tr>
<td>SC6</td>
<td>Source code visibility</td>
</tr>
<tr>
<td>SC7</td>
<td>Source code continuity</td>
</tr>
<tr>
<td>SC8</td>
<td>Documentation</td>
</tr>
<tr>
<td>SC9</td>
<td>Editor support</td>
</tr>
<tr>
<td>SC10</td>
<td>Peer experience</td>
</tr>
<tr>
<td>SC11</td>
<td>Community activity</td>
</tr>
<tr>
<td>SC12</td>
<td>Tutorial coverage</td>
</tr>
<tr>
<td>SC13</td>
<td>Overall cost</td>
</tr>
<tr>
<td>SC14</td>
<td>Framework reliability</td>
</tr>
<tr>
<td>SC15</td>
<td>Amount of components</td>
</tr>
<tr>
<td>SC16</td>
<td>Effect on delivery time</td>
</tr>
<tr>
<td>SC17</td>
<td>Functionality</td>
</tr>
<tr>
<td>SC18</td>
<td>Access to native features</td>
</tr>
<tr>
<td>SC19</td>
<td>Performance</td>
</tr>
<tr>
<td>SC20</td>
<td>Cross-platform support</td>
</tr>
<tr>
<td>SC21</td>
<td>Deployability on application stores</td>
</tr>
<tr>
<td>SC22</td>
<td>Complexity</td>
</tr>
<tr>
<td>SC23</td>
<td>Aesthetics</td>
</tr>
<tr>
<td>SC24</td>
<td>UX</td>
</tr>
<tr>
<td>SC25</td>
<td>Learning curve</td>
</tr>
<tr>
<td>SC26</td>
<td>Maintainability</td>
</tr>
<tr>
<td>SC27</td>
<td>Testability</td>
</tr>
</tbody>
</table>

Table 3.1: Criteria for technology selection used or mentioned in literature
Chapter 4

Methods

This Chapter discusses the methods that will be used to achieve the goals of this thesis. As the main goal of this thesis is to formulate a way to select the most suitable framework, first criteria for the framework selection has to be formulated. As choosing a proper DA seems to be a crucial aspect regarding the selection, the selected criteria will be used for DA selection first. Then, frameworks are compared against each other with respect to the defined criteria and the most suitable framework for the case of the mobile application is selected accordingly. Afterwards, the mobile application development will be tested with the specified framework. Finally, the selection will be reviewed for evaluating its real suitability.

As some aspects of DAs are not binary and there are multiple criteria to be considered in the decision-making, the selection process will be rather complex. Consequently, a suitable method for multiple-criterion selection should be chosen. However, as can be read from the works of Wallenius et al. [32], Kolios et al. [28] and Kaur and Singh [27], the decision of choosing a proper decision method is complex as well. This is due to the fact that there are multiple different methods for commencing multiple-criterion decision making (MCDM): analytic hierarchy process (AHP) [27, 28], goal programming [32], EMO [32], multiattribute utility theory (MAUT) [32], mathematical programming [32], vector optimization [32], weighted sum method (WSM) [27, 28], weighted product method (WPM) [28], technique for the order of preference by similarity to the ideal solution (TOPSIS) [28], elimination et choix traduisant la réalité (ELECTRE) [28, 32] and the preference ranking organization method for enrichment evaluation (PROMETHEE) [28, 32].

According to Kolios et al., as stated in Chapter 6. Conclusions, none of the methods seem to outperform other MCDM methods in their case study concerning wind turbines [28]. Additionally, the selection process for selecting a proper MCDM is quite paradoxical: for choosing the most suitable
alternative, one must choose the most suitable MCDM method.

Unfortunately there seems to be a rather low amount of recent scientific literature available regarding the utilization of MCDM methods in software component selection. Since none of the methods can be considered most suitable and there is a lack of scientific resources regarding the best alternative for software tool selection, the simplest MCDM method will be used in this thesis as the principle of Occam’s razor might suggest in a similar situation. As WSM seems to be the most simple method mentioned in sources [28] and [27], it will be used. In WSM, each alternative will have a calculated weighted sum score which is achieved by utilizing the following formula:

\[ A = \sum_{i} a_i w_i \]

where \( A \) is the weighted sum score for an alternative, \( m = \) amount of criteria, \( i = 1, \ldots, m \), \( a_i \) represents the value of \( i \)-th criterion for the alternative in question and \( w_i \) is the weight for the \( i \)-th criterion. After \( A \) has been calculated for each alternative, one with the highest value for \( A \) will be chosen. To utilize WSM, specific weights and the scale for \( a_i \) for the selection criteria for DA and framework are determined later in Section 5.2.

As there is a relatively extensive data regarding DAs in previous literature, a rating method for determining each \( a_i \) is required. The rating will be commenced as follows: if an advantage of a DA listed in Section 3.1.2 is a clear advantage, it will be interpreted as a 4. If the advantage can be considered as a minor advantage or slightly contradicted by a disadvantage regarding the same SC, \( a_i \) for the alternative will be a 3. If there is a major disadvantage that is slightly contradicted by another advantage regarding the same SC, the value for \( a_i \) will be 1 in this case. If there is a clear disadvantage for the alternative regarding the SC in question, 0 will be used. Value of 2 is used as a baseline as mentioned in Section 5.2. Each SC that have been named to represent it as a disadvantage must be considered the opposite for proper calculation. For example, SC22 will be interpreted in the calculations as "simplicity" instead of "complexity", meaning that the value of \( a_i \) can always be interpreted as "higher is better" in the table.

In contrast to the DA selection, however, the framework itself must be chosen with separate rating methods for each SC as there is not enough literature regarding frameworks themselves that state advantages and disadvantages clearly enough. Thus, the following logic will be applied to determine \( a_i \) for each SC:

- SC1: as the most significant API capabilities are, in the context of
this thesis, taken into consideration with SC18 and SC20, this SC will account for the following capabilities:

- Energy efficiency
- Extensibility with plug-ins, for example
- The difficulty of implementing client-server HTTP communication

As the application to be developed is rather simple, the role of this SC is rather insignificant. Thus, this will be evaluated subjectively as follows: if none of the listed aspects are discussed in the documentation and seem not to be taken into consideration to at least some extent, $a_i$ will have a value of 0. If all of the aspects are taken into consideration in the framework, the value will be 4. If only some of the aspects are taken into account, values of 1 or 3 three may be used. Otherwise, $a_i$ will be 2.

- SC2: the programming language is considered "well-known" if it is mentioned in the following list:
  - Python
  - JavaScript (and additionally HTML and CSS)
  - Java
  - C/C++
  - C#

As the development team is mostly experienced with Python, JavaScript, HTML and CSS, $a_i$ will be 4 if the framework requires expertise in those languages. If the required language is Java, then 3. Respectively, a value of 2 will be used in the case of C/C++ and a value of 1 for C#. Otherwise the value will be 0.

- SC3: $a_i$ will be 4 if the framework is open-sourced, otherwise 0.

- SC4: if the development and deployment tools can be installed with one script or installer, a value of 4 will be used. Otherwise, if the approximated time of installation will take less than 15 minutes, the value will be 3. If 30 minutes, then 2, and if 45 minutes, then 1. Otherwise $a_i$ will be 0.

- SC5: this SC will be omitted as it is rather subjective and the ease of use can not really be evaluated beforehand in a reasonable manner
• SC6: if the source code is freely available on a site such as GitHub, a value of 4 will be used. Otherwise 0 will be used.

• SC7: if the developer community is active in a way that there is a relatively constant flow of updates to the repository of the framework, higher value will be used. Otherwise lower. This is calculated in the following way:
  – At least 2 or more commits in the past 2 weeks: 4
  – At least 2 commits during the past 4 weeks: 3
  – At least 1 commit in the past 1 month: 2
  – At least 1 commit in the past 3 months: 1
  – All commits older than 3 months: 0

• SC8: if the documentation seems clear and easily accessible, a value of 4 will be given. If the documentation is easily available, but incomplete or ambiguous, a value of 3 will be selected for \( a_i \). If documentation is available, but clearly incomplete, a value of 1 will be used. If there is no documentation, a value of 0 will be used. If no other category is fitting enough, a value of 2 will be chosen.

• SC9: if there is sufficient editor support such as syntax highlighting and code suggestions for multiple main-stream editors, a value of 4 will be used. If no editor support is available, a value of 0 will be used. If the support is lacking, values of 1 or 3 are used. Otherwise \( a_i \) will be 2.

• SC10: \( a_i \) for peer experience will be defined as follows:
  – 4 if all of the developers of the team have been using the framework regularly
  – 3 if one or more developers of the team have been using the framework on a regular basis
  – 2 if at most half of the developers in the team have been using the framework frequently
  – 1 if at most half of the developers have only some experience of the framework
  – 0 if none of the developers have experience of the framework

• SC11: \( a_i \) for this will be calculated as follows:
– 4 if the amount of open issues is lower than 15% of all reported issues
– 3 if the amount of open issues is lower than 30% of all reported issues
– 2 if the amount of open issues is 30%-70% of all reported issues
– 1 if the amount of open issues is higher than 70% of all reported issues
– 0 if the amount of open issues is lower than 85% of all reported issues

• SC12: due to the fact that this aspect is difficult to evaluate beforehand, the following logic is used: \( a_i \) will be 4 if the tutorial material seems to be extensive and clear. If there seems to be no tutorial available or the tutorial is clearly incomplete or ambiguous, \( a_i \) will be 0. Otherwise, a value of 2 will be used.

• SC13: \( a_i \) for this aspect will be defined as follows from the perspective of development and commercial use of the application utilizing the selected framework:
  – 4 if the expected costs are 0€
  – 3 if the expected costs are single-payment only
  – 2 if the development may be done for free of charge yet commercial use obliges recurring payments of up to 10€ per month
  – 1 if the costs are recurring and up to 20€ per month
  – 0 if there are recurring costs that are higher than 20€ per month

• SC14: as this criterion is difficult to determine beforehand, it will be excluded as SC5.

• SC15: if the framework requires additional tools to achieve the desired end result, a value of 0 will be used. If everything that is required to compose the application is included within the framework, a value of 4 will be used. Otherwise, \( a_i \) will be 2.

• SC16: as this SC is rather difficult to evaluate in an objective manner beforehand, this will be evaluated qualitatively as follows: if the framework seems to be fast to install, learn and use, a value of 4 will be given for \( a_i \). If, however, it seems to be relatively complex and sluggish, a value of 0 will be given. If it seems to be fast or slow yet has some
aspects that might suggest the opposite expectations, a value of 1 or 3 may be used. Otherwise, \( a_i \) will be 2.

- **SC17**: if all requirements regarding the MVP of the application as listed in Tables 2.1 and 5.1 can be expected to be achieved, a value of 4 will be used for \( a_i \). Otherwise, \( a_i \) will be as follows:
  - 3 if at least 75% of the requirements can be expected to be achieved
  - 2 if at least 50% of the requirements can be expected to be achieved
  - 1 if at least 25% of the requirements can be expected to be achieved
  - 0 the framework can not fulfil the requirements at all

- **SC18**: as this SC is practically defined by the DA itself, this will be excluded as well.

- **SC19**: as with SC18 above, this is mostly determined by the DA. Thus, performance differences between different frameworks are assumed negligible enough to be excluded.

- **SC20**: the chosen DA determines whether or not the comparable frameworks are cross-platform or not. Consequently, this SC will be omitted as well.

- **SC21**: this SC is also dictated by the DA and is thus excluded.

- **SC22**: as complexity of a framework is somewhat determined by the DA, and a reasonable and clear objective meter is rather impossible to define, this SC will be omitted in addition to others.

- **SC23**: this SC will be omitted as well due to these reasons: aesthetics regarding as native-like UI as possible is dictated by DA as well and the developed UI itself is dependant of the developers themselves. Additionally, aesthetics of the final application are impossible to determine objectively.

- **SC24**: as with SC23, this criterion is mainly mandated by the selected DA and skills of the software developers. Thus, this is also excluded.

- **SC25**: due to the subjective nature of this SC, \( a_i \) for this criterion will be defined as follows:
  - 4 if the development team considers the framework as easy for them to work with
- 3 if the development team considers the framework as quite easy for them to work with
- 2 if the framework is such difficult that it can be considered to require a reasonable amount of learning
- 1 if the development team considers the framework as quite difficult for them to work with
- 0 if the development team considers the framework as difficult for them to work with

- SC26: as the maintainability of the client application is mostly dictated by the DA and can be considered to be affected by the skills of the developers as well, this will be omitted.

- SC27: if the framework has sufficient amount of testing tools and support, a value of 4 for $a_i$ will be used. If there seems to be no testing support, $a_i$ will be 0. If there is testing support and some relevant aspects are missing, a value of 1 or 3 may be used. Otherwise, $a_i$ will be given a value of 2.

As some of the selection criteria are too subjective to measure or mainly dictated by the selected DA, only a subset of the criteria will be used for framework comparison. Consequently, only the following criteria are used for comparing the frameworks: SC1, SC2, SC3, SC4, SC6, SC7, SC8, SC9, SC10, SC11, SC12, SC13, SC15, SC16, SC17, SC25 and SC27.

In conclusion, each SC will be given a specific weight with respect to the requirements of the MVP of the application. The DA and framework will be chosen with WSM. Alternatives with the highest WSM value A will be chosen.
Chapter 5

Implementation

This chapter discusses the implementation and selection process, and it consists of three sections. In the first section, a specific DA is selected to narrow down the most viable frameworks. Second section describes the process of framework selection. Finally, in the third section, development of the mobile application itself utilizing the selected approach and framework is represented.

5.1 Additions to client requirements and selection criteria

Due to the fact that the preliminary requirements for the MVP of the mobile application defined in Table 2.1 only accounted for the behaviour of the application rather than the development process and maintainability of it, the requirements should be reviewed and extended before selecting a proper DA. The object is to avoid challenges mentioned in Section 3.1 and take selection criteria listed in Table 3.1 into consideration.

Regarding the challenges listed in the beginning of Section 3.1, the following challenges should at least be taken into consideration in general as they are the most common ones: UX, fragmentation, compatibility, testing and change management. However, in the case of My Hotel, due to the rather low amount and probability of requests for complex features, the challenge of compatibility will be excluded from consideration. Considering the risk of low code reusability may be omitted as fragmentation is quite logically one of the primary reasons for it and thus it has already been mostly taken into account. The challenge of lacking tool support will, on the other hand, be considered as both members of the development team are rather inexperienced with modern mobile software development tools. Additionally, for
<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC-3.1</td>
<td>Decent UX</td>
<td>The application should have as native look and feel as possible and be performant enough in most situations.</td>
</tr>
<tr>
<td>MC-5.0</td>
<td>Efficient development</td>
<td>The application development should be as effortless as possible and minimize the risk of fragmentation. Additionally, the framework, other necessary tools and programming languages should be easy to assimilate as rapidly as possible.</td>
</tr>
<tr>
<td>MC-5.1</td>
<td>Efficient testing</td>
<td>The application should be testable as quickly as possible during development and have overall good testing support.</td>
</tr>
<tr>
<td>MC-5.2</td>
<td>Maintainability</td>
<td>Changes to the application in the future should be achievable as effortlessly as possible.</td>
</tr>
</tbody>
</table>

Table 5.1: Additional requirements for MVP of the mobile client (MC), as an extension to the requirements in Table 2.1

mostly the same reason, lack of expertise will be considered as well by utilizing as familiar tools and programming languages as possible. Finally, as the mobile application will be rather simple and the main goal is to evaluate the suitability of a specific framework regarding development and the end result, challenge of low security will be omitted as well.

To overcome the challenges at least partially, the following requirements are added for the application: MC-3.1, MC-5.0, MC-5.1 and MC-5.2. These are, respectively: decent UX, efficient development, efficient testing and maintainability. The first is for overcoming the challenge of bad UX as the title may suggest. The second one, MC-5.0, is to avoid multiple challenges at once: fragmentation, lack of tools support and lack of expertise. The third one, MC-5.1, is a requirement for overcoming testing difficulties and to complement the requirement MC-5.0. Finally, the challenge of having poor change management in the future is supposed to be avoided by requirement MC-5.2. These requirements are described in more detail in Table 5.1.

Complete requirements for the mobile client application can be viewed in Appendix A. In summary, some non-functional requirements were added to the list of requirements to enhance the UX and developer-friendliness of the application.
5.2 Definitions for WSM-based MCDM

As WSM is utilized for selecting the most suitable DA and framework, the weight $w_i$ has to be calculated for each SC as was mentioned in Chapter 4. Additionally, a scale for each SC in the comparison is required to define $a_i$ for each alternative. Consequently, these are discussed in this section briefly.

From the developer's perspective, the following requirements for the application can be considered the most important: MC-5.0, MC-5.1 and MC-5.2. From an user's perspective, the following can be regarded as the most crucial: MC-3.0, MC-3.1 and MC-2.0. The closest selection criteria to the developer-related requirements are as follows: SC2, SC4, SC5, SC8, SC9, SC12, SC15, SC16, SC17, SC22 and SC25 for MC-5.0; SC27 for MC-5.1; and SC26 for MC-5.2. Concerning the user's perspective, the following selection criteria can be considered significant: SC23 for MC-3.0, and both SC19 and SC24 for MC-3.1. MC-2.0 will be considered as implementable despite the chosen DA and framework.

Due to the fact that there are a total of 16 selection criteria that may be considered significant with respect to the requirements for the MVP of the application, there will be a total of 17 weights used in this thesis: from 17 to 1. Each SC will get weights as described in Table 5.2, prioritizing the requirements from the users perspective. The most suitable DA will be chosen by using these weights in Section 5.3.

<table>
<thead>
<tr>
<th>SC</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC24</td>
<td>17</td>
</tr>
<tr>
<td>SC19</td>
<td>16</td>
</tr>
<tr>
<td>SC23</td>
<td>15</td>
</tr>
<tr>
<td>SC25</td>
<td>14</td>
</tr>
<tr>
<td>SC2</td>
<td>13</td>
</tr>
<tr>
<td>SC16</td>
<td>12</td>
</tr>
<tr>
<td>SC22</td>
<td>11</td>
</tr>
<tr>
<td>SC17</td>
<td>10</td>
</tr>
<tr>
<td>SC8</td>
<td>9</td>
</tr>
<tr>
<td>SC12</td>
<td>8</td>
</tr>
<tr>
<td>SC5</td>
<td>7</td>
</tr>
<tr>
<td>SC4</td>
<td>6</td>
</tr>
<tr>
<td>SC9</td>
<td>5</td>
</tr>
<tr>
<td>SC15</td>
<td>4</td>
</tr>
<tr>
<td>SC26</td>
<td>3</td>
</tr>
<tr>
<td>SC27</td>
<td>2</td>
</tr>
</tbody>
</table>
Table 5.2: SC weights

Regarding the scale for $a_i$ for each alternative, a scale of $[0, 4]$ will be used for simplicity in this thesis. By default, the value used for $a_i$ will be 2.

5.3 Development approach selection

In this section, a specific development approach is selected for later selecting the framework itself in Section 5.4. Due to the requirement MC-4.0 which is described in Table 2.1, native DA is excluded. Additionally, the ordinary web-based DA is omitted as well as it is superseded by PWA-based approach. Thus, the remaining possible alternatives for the DA correspond to those mentioned in Table 1 of the survey conducted by Bjørn et al. in [19]: hybrid, interpreted, cross-compiled, model-driven and PWA.

For commencing the comparison, the advantages and disadvantages of DAs mentioned in List 3.1.2 must be mapped to selection criteria. This is done as follows:

- Cross-platform: SC20
- No application updates: SC16, SC26
- Reduced development time and cost: SC13, SC16
- Partial access to native functionalities: SC18
- Low performance: SC19, SC24
- Not publishable on application stores: SC21
- Compatibility: not used as the respective challenge is omitted in this thesis as stated in Section 5.1
- UX: SC24
- Testing: SC27
- Advantages of web-based DA: not applicable to any SC directly
- Native look and feel achievable: SC23, SC24
- Offline use possible: SC1, SC17
• UX on iOS: SC23
• Native performance: SC19, SC24
• Access to native functionalities: SC18
• Native UI: SC23
• Advantages of native DA: not applicable to any SC directly
• Publishable on application stores: SC21
• Popularity: SC7, SC8, SC9, SC10, SC11, SC12, SC14, SC26
• Native UI achievable: SC23
• Low effort: SC13, SC16, SC22, SC25
• Alternative frameworks: SC14
• Native UI and UX not guaranteed: SC23, SC24
• Requires additional tools: SC5, SC15, SC16, SC22, SC25, SC26
• High abstraction level: SC16, SC22, SC25, SC26
• Customizability: SC1, SC17
• Maintainability: SC26
• Limited programmability: SC1, SC17
• Requires expertise despite having a high abstraction level: SC2, SC5, SC22, SC25
• Cross-platform development: SC20
• Framework-lock: SC10, SC11, SC26
• Native functionalities: SC18
• Increased development time and cost: SC13, SC16
• Fragmentation: SC20, SC26
• Change management: SC26
• Native functionalities: SC18
CHAPTER 5. IMPLEMENTATION

- Dependence on the development environment: SC9, SC15, SC25, SC26, SC27
- Updateability: SC16, SC26
- Lower performance: SC19, SC24
- Low popularity: SC7, SC8, SC9, SC10, SC11, SC12, SC14, SC26

Considering the list above, a_i for each alternative DA can now be rated. A complete WSM-based comparison for each DA is detailed in Table 5.3 according to the plan as described in Chapter 4.

<table>
<thead>
<tr>
<th>SC</th>
<th>w</th>
<th>Hybrid</th>
<th>Interpreted</th>
<th>Cross-compiled</th>
<th>Model-driven</th>
<th>PWA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>SC2</td>
<td>13</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>SC3</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SC4</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SC5</td>
<td>7</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>SC6</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SC7</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>SC8</td>
<td>9</td>
<td>4</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>SC9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>SC10</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>SC11</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>SC12</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>SC13</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>SC14</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>SC15</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SC16</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>SC17</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>SC18</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>SC19</td>
<td>16</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SC20</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>SC21</td>
<td>1</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>SC22</td>
<td>11</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>SC23</td>
<td>15</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
Table 5.3: Development approach comparison, scale: [0, 4]. Value for A was calculated with Google Spreadsheets for each alternative DA

According to the comparison in Table 5.3, the results are as follows: hybrid DA received a WSM score A of 442 whereas the second most suitable DA, cross-compiled, has A of 408. Model-driven, PWA and interpreted DAs got WSM scores of 360, 329 and 262 respectively. This means that the most suitable DA should be hybrid DA and the least suitable DA would be interpreted DA. Thus, according to the table, hybrid DA will be the selected DA in the case of My Hotel.

5.4 Framework selection

This section discusses the selection process and results regarding the framework that the mobile application will be based on. As the chosen DA in Section 5.3 is hybrid DA, there are multiple available frameworks. According to Biørn et al., there are 17 possible frameworks under the category of hybrid DA in Table 1 [19]. It should be noted, though, that most of the hybrid DA frameworks listed in [19] are based on Apache Cordova, as is mentioned on page 4 [19]. Only three alternative frameworks are clearly based on other technologies: Capacitor, RhoMobile and Trigger.io [19].

One framework, Kony, might also be considered to be an alternative to Cordova [19]. However, for comparing the frameworks, access to issues and repositories of the frameworks are required and apparently does not seem to be open-sourced [11]. Additionally, according to their website, Kony seems to be a banking-oriented framework [11] and, thus, it might not be a sufficient tool for My Hotel. Consequently, Kony will be excluded from this thesis. As with Kony, Trigger.io is not open-sourced either, rendering it excluded as well in this thesis [17].

Assuming that Biørn et al. refer to RhoMobile Suite with the term "RhoMobile" in [19], the framework in question seems to be actually based on Rhodes framework, according to its documentation available at [16]. As Biørn et al. interpret Rhodes framework to be a framework of cross-compiled approach in Table 1 in [19], there seems to be a contradiction whether the
framework may be interpreted as a hybrid framework or not. As cross-compiled DA is conceptually significantly different than hybrid DA and RhoMobile Suite is dependant of such a framework, RhoMobile Suite will be interpreted as a cross-compiled framework in this thesis. Thus, RhoMobile Suite is omitted as well from the comparison.

Due to the fact that, regarding functionality and API capabilities, there are only two clear competing alternatives remaining and that a native-like UI/UX can be achieved by utilizing other libraries and tools, these frameworks will be compared in this section: Cordova and Capacitor. For defining \( a_i \) for both alternatives, some sources are needed first. Regarding Apache Cordova, the repositories may be found from their webpage [6]. Documentation for Cordova can be found from their pages as well at [1]. For Capacitor, the repository and documentation may be found from [10] and [8], respectively.

To describe Cordova shortly, the framework in question is created by Apache Software Foundation that can be reviewed at https://cordova.apache.org/ and installed with the following command: `npm install -g cordova` [3]. A simple Hello World -application can be generated with `cordova create hello com.example.hello HelloWorld` after installation [3]. As the DA dictates, Cordova is a framework that is based on rendering the application content within a WebView component and accessing native features via APIs as can be seen in Figure 5.1 [5]. Cordova is also extensible with additional plugins. The architecture and scale of Cordova is reflected in its codebase: it consists of 33 different repositories hosted on GitHub [6].

Capacitor, on the other hand, is created by Ionic and its home page can be found at https://capacitorjs.com/. As with Cordova, it can be installed with `npm install @capacitor/core @capacitor/cli` [9]. A capacitor-based application may be initialized with a simple `npx cap init` and, additionally, a fresh project may be initialized with `npx @capacitor/cli create` [9]. As Capacitor is practically a successor to Apache Cordova, it is also utilizable for creating a web application that is run natively while having access to native features [8]. Capacitor consists only of one repository that is hosted on GitHub [10].

Comparison between Cordova and Capacitor is commenced in Table 5.4 and it is based on the resources mentioned above. The following criteria are have equal value for \( a_i \) for both frameworks: SC1, SC2, SC3, SC4, SC6, SC7, SC8, SC9, SC10, SC13 and SC17. Regarding SC1, both frameworks are extensible with plug-ins according to the documentation. Energy efficiency has not been mentioned nor researched for determining it. Client-server communication for HTTP may be achieved with an external plug-in: cordova-plugin-
advanced-http for Cordova which is available at https://www.npmjs.com/package/cordova-plugin-advanced-http and @capacitor-community/http for Capacitor that is available at https://www.npmjs.com/package/@capacitor-community/http. Due to the fact that energy efficiency is not a major factor in My Hotel and that both frameworks are extensible with plug-ins, including a plug-in for HTTP communication, both options deserve a value of 3 for a. SC2 is dictated by DA and SC3 is 4 automatically as both options are open-sourced. Installation of both frameworks can be done with one command, and thus both receive 4 for a. Due to the fact that both of the frameworks are freely available on GitHub, SC6 will be 4 as well for both options. Both alternatives have also had at lest 2 commits during the past 2 weeks at the time of measurement on 11.11.2020. Consequently, SC7 will be 4 for each framework. SC8 and SC9 are 4 for both frameworks as well as each option has extensive and clear documentation and both are based on web technologies which are expected to be supported by multiple different editors and IDEs. SC10 is 0 as neither of the developers has experience regarding the frameworks. SC13 is evaluated as 4 for both options as the overall cost is expected to be zero as neither option entail any sort of payments. SC17 is evaluated 4 as well as it is highly likely that all requirements can be implemented with the either of the frameworks.
CHAPTER 5. IMPLEMENTATION

The remaining criteria, SC11, SC12, SC15, SC16, SC25 and SC27, have differences for the following reasons: concerning SC11, Capacitor had significantly less open issues than Cordova. After counting the amount of issues for both frameworks, including all of the Cordova repositories at [6], on 11.11.2020, Cordova had 2896 issues in total, of which 1223 were still in opened state. In contrast, Capacitor had only 152 open issues of 1900 total issues. Thus, Cordova had about 42% opened issues of all issues whereas Capacitor had 8%. As a consequence, Capacitor is granted a value of 4 and Cordova a value of 2 for a_. Regarding SC12, Cordova documentation has a "Create your first app" chapter and examples for emulating the app whereas Capacitor documentation does not. Thus, the tutorial of Cordova is considered more extensive than that of Capacitor. Documentation of both frameworks are rather clear. For these reasons, values of 4 and 2 are granted, respectively.

For SC15, Cordova got a greater value for a_ as it does not require such many components to be used although commencing tests, for example, seem to require additional tools. Conversely, the documentation of Capacitor states that additional tools are required. SC16 is considered to be advantageous for Cordova as it seems to have apparently less tools required, has a bit more extensive tutorial and has a simpler app build process than Capacitor. SC25 is also greater for Cordova as the framework seems to be quite easy whereas Capacitor requires additional tools and does not have as extensive documentation as Cordova. Finally, SC27 is also in Cordova's favour as it seems to be more testable than Capacitor due to the additional tools as discussed earlier regarding SC15.

<table>
<thead>
<tr>
<th>SC</th>
<th>w</th>
<th>Cordova</th>
<th>Capacitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>SC2</td>
<td>13</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>SC3</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SC4</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SC6</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SC7</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SC8</td>
<td>9</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SC9</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SC10</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SC11</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>SC12</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>SC13</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SC15</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 5.4: Framework comparison, scale: [0, 4]. Value for A was calculated with Google Spreadsheets for each alternative framework

As can be seen from Table 5.4, Cordova got a value of 302 for A whereas Capacitor got 249 respectively with WSM-based comparison. Due to the fact that A is greater for Cordova than Capacitor, Cordova will be the chosen framework for My Hotel.

5.5 Application development with Cordova

Development of the application with the selected framework is briefly discussed here. First, some of the challenges associated with the framework are briefly reviewed. Afterwards, implementation of the My Hotel application itself will be examined. As the main focus of this thesis is on the mobile client, implementation details regarding the web server are omitted.

5.5.1 Challenges of Cordova

According to the documentation of Cordova and observations in Appendix B, there are a plentitude of challenges associated with Cordova despite it being the most suitable framework according to Table 5.4. The following challenges are to be expected:

- Android-related
  - Debugging
  - OS can kill the application in the background for freeing memory, for example. Consequently, the WebView will be destroyed and the state will be lost as well. This can happen if the user taps home or back buttons, or when a specific native activity, such as camera activity, is engaged by the application [2]
  - State has to be maintained for pause and resume events to overcome the issue of OS killing the application [2]
• iOS-related
  
  – No macOS available or Xcode editor for Windows 10: difficulties with building, emulating and debugging the application with iOS

• Common
  
  – Highly recommended to use the single-page application (SPA) approach [4]: implies that a separate front-end framework practically has to be used or the JS code will bloat. This is rather reasonable, however, though it increases complexity

  – 300ms delay on click events: must use WKWebView for iOS or either touchstart or touchend events instead of click event [4]

  – Animations have to be implemented mainly with CSS instead of JS [4]

  – Cache has to be used as many times as possible to avoid network-related issues [4]

  – Preparations for offline use: separate XML HTTP requests (XHR) may be needed to detect the current network status effectively [4]

  – UI implementation must adhere to the guidelines of each platform: recommended to use a UI specific framework [4]

  – Platform-specific quirks [4]

5.6 Development of My Hotel

Due to the fact that the Hello World example created in Appendix B does not match the requirements of My Hotel that are defined in Tables 2.1 and 5.1, a fresh project has to be created. Additionally, as the documentation states that the UI design may be rather tricky and that SPA approach is highly recommended, a more suitable Cordova template will be used. As suggested by the documentation, templates can be searched with keyword cordova:template on https://www.npmjs.com/. For the reasons that cordova-react template package can be found on npm at https://www.npmjs.com/package/cordova-react, ReactJS is suggested by the documentation of Cordova [4] and developers are at least partially familiar with ReactJS to some extent, it will be used as the template.

For starting the development of My Hotel, the following commands were run:
> cordova create MyHotel --template cordova-react
> cd MyHotel
> npm install

While attempting to start the application for the first time as mentioned in the instructions at https://www.npmjs.com/package/cordova-react, the following issue was encountered:

> cordova run android --devmode --devserver
Current working directory is not a Cordova-based project.

Apparently, the issue mentioned above can be solved by adding a www-folder to the project according to a user called S.Yadav on StackOverflow at https://stackoverflow.com/a/42951603. Additionally, it was discovered that the command cordova platform add android was mandatory as well. However, after these changes one more error was encountered regarding directory permissions:

> cordova run android --devmode --devserver
Setting up Symbolic links...
Error: EPERM: operation not permitted, symlink 'C:\myhotel2\client\MyHotel\assets\client\css' -> 'C:\myhotel2\client\MyHotel\www\css'
at Object.symlinkSync (fs.js:1016:3)
at C:\myhotel2\client\MyHotel\assets\scripts\symlink.js:35:16
at Array.forEach (<anonymous>)
at Object.createSymlink [as create] (C:\myhotel2\client\MyHotel\assets\scripts\symlink.js:31:36)
at Object.start (C:\myhotel2\client\MyHotel\assets\scripts\dev-env.js:31:13)
at run (C:\myhotel2\client\MyHotel\assets\hooks\run-dev.js:32:26)
at module.exports (C:\myhotel2\client\MyHotel\assets\hooks\run-dev.js:40:5)
at runScriptViaModuleLoader (C:\Users\<USER>\AppData\Roaming\npm\node_modules\cordova\node_modules\cordova-lib\src\hooks\HooksRunner.js:157:32)
at runScript (C:\Users\<USER>\AppData\Roaming\npm\node_modules\cordova\node_modules\cordova-lib\src\hooks\HooksRunner.js:136:12)
CHAPTER 5. IMPLEMENTATION

at C:\Users\<USER>\AppData\Roaming\npm\node_modules\cordova\node_modules\cordova-lib\src\hooks\HooksRunner.js:108:40
    errno: -4048,
sysexit: 'symlink',
    code: 'EPERM',
    path: 'C:\myhotel2\client\MyHotel\assets\client\css',
    dest: 'C:\myhotel2\client\MyHotel\www\css'
}

Due to the fact that cordova-react template seems to entail more complexity for the project, as was expected, My Hotel application will be based on the Hello World example for the sake of the defined goals in Section 1.2. Thus, the application ended up being initialized and tested as with the Hello World example in Appendix B with the following commands:

> cordova create MyHotel2 com.myhotel.client "My Hotel"
> cd MyHotel2/
> cordova platform add android
> cordova build android
> cordova emulate android

After executing the commands above, the following error was encountered: PANIC: Missing emulator engine program for 'x86' CPU. To solve this, AVD was started manually first with AVD Manager of Android Studio. Apparently this error occurs every time if the AVD has not been started yet with the AVD Manager. Additionally, if cordova emulate android is called before starting the AVD manually first, the following error is encountered in Android Studio: Unable to locate adb. Despite the error, AVD will start after pressing OK.

The application development of My Hotel was started on 20.11.2020. The following design decisions were made during the process: as the web server development was omitted, authentication and authorization were decided to be implemented with a mock implementation, in other words, with a role selector that is shown in the left half of Figure 5.2. As can be seen from the red warning text in the figure, the role selector is supposed to be omitted in the final version of the application. Additionally, regarding platforms, Android v10 and browsers were decided to be targeted. Browsers were decided to be targeted as well instead of iOS as a consequence of the team not having a macOS-based system available which would be needed for building the application. As iOS-based devices have web browsers, customers
CHAPTER 5. IMPLEMENTATION

could access the application with this alternative method. Due to the fact
that the use of cache and state were recommended by the documentation of
Cordova, My Hotel implementation utilizes window.localStorage as its
cache. State management utilizes it as well for allowing the application to be
resumed correctly in case the Android OS kills it suddenly. For separating
application HTML into logical sections, jQuery was decided to be used due to
its method $("<ELEMENT>").load("<HTML_FILE_PATH>") that allows
HTML files to be loaded into the specified element. In addition to partition-
ing HTML, jQuery offered some utilities for commencing SPA-related element
management as selecting a specific element for content update, for example.
As Cordova is based on using a WebView for managing the application, utili-
zation of the back-button of the phone had to be taken into consideration
as well. A rather common approach of so-called "bread crumbs" was imple-
mented to accommodate intuitive navigation. In other words, state of the
application was pushed into an array every time the user navigated within
the application, and when the state had to be reverted or recovered, the state
was simply popped from the array.

The application was relatively complete on 24.11.2020, after only four
days of development, considering its requirements. The development took
approximately 24 hours. Some images of the application prototype are shown
in Figures 5.2, and 5.3. Regarding the browser-based alternative for iOS
users, the application was created and started with the following commands:
cordova platform add browser
cordova build browser
cordova run browser

Last one of the listed commands above allowed the same My Hotel application
to be served as a web application. An image of the browser-based solution
is illustrated in Figure 5.4.

During development, the following difficulties were encountered:

- No real-time testing for the application with AVD: every time a change
  was added, cordova emulate android had to be run manually
  afterwards. This could be solved, in most cases, by creating a plain
  web application first with cordova serve [port] command as is
  stated in the documentation of Cordova

- Debugging JS objects was difficult as the application output that was
  printed into the Logcat-view of Android Studio could not manage print-
  ing the whole object as an expandable object as ordinary web develop-
  ment tools of Google Chrome, for example. Calling console.log(’
CHAPTER 5. IMPLEMENTATION

Figure 5.2: Role selector mock implementation and customer main view of My Hotel

Figure 5.3: View for a customer for requesting an extension to their stay at a lodging company and the task list view for an employee
Figure 5.4: Customer’s main view in the browser-based solution as observed with Google Chrome v87.0.4280.88
object = ', object); would lead to a situation where only "object = " was printed. This could be partially solved by concatenating
the string with the object by using a plus-sign. This, however, caused
the output to contain only "[object Object]" at the place of the object-
variable

• Rotating the device by 90 degrees would not automatically cause the
application to render the content in portrait or landscape mode. To
solve this, a plug-in called cordova-plugin-screen-orientation was im-
ported and configured. Unfortunately, the application could not be ro-
tated even with the plug-in although <preference name="Orientation" value="fullUser" /> was added to the con-
fig.xml and window.screen.orientation.unlock() was called

• While deploying the application as a web app with cordova serve
command as mentioned in documentation, a couple of error messages
were encountered on the client-side every time when visiting the web
page. For example, first of the messages was titled "Message from
address localhost:8000" and had an input field containing "gap_init.3".
Fortunately, this could be overcome by utilizing browser-platform and
its respective build and run commands

Additionally, the following obscurities regarding development were ex-
perienced:

• Development of the application implies the use of two separate IDE
sessions: one for editing the source code in www folder and one for
building and debugging the application in platforms/android directory
with Android Studio

• Details and rationale of cordova.js that is imported in index.html by de-
fault raises the following questions: is it required? Where is it imported
from as it does not appear to have a path?

• <h1> tag could not be used for titles as it caused a slight unexpected
offset to the right side of the view without any CSS definitions affecting
it
Chapter 6

Evaluation

In this Chapter the suitability of the selected framework in the context of My Hotel development is evaluated. Additionally, the developed application itself and the chosen metrics and methods are reviewed for analysing the suitability of the selection process as well.

6.1 Success of My Hotel application

Considering the fact that the server-side part of the implementation was excluded, the application was rather successful. Most of the requirements listed in Tables 2.1 and 5.1 were able to be implemented during the time reserved for development.

Out of 19 defined requirements for the MVP, 9 were completed successfully. MC-3.1 is considered as successful as although the browser-based solution for iOS use contains unwanted scrollbars, those could be easily removed in the next iteration. 6 of the requirements were partially reached, including MC-1.1, MC-1.1.1, MC-2.3.1, MC-2.4, MC-4.0 and MC-5.0. Most significant ones of the aforementioned requirements are MC-4.0 and MC-5.0 as the former one was not met as the application could not be tested on iOS devices although the application should have cross-platform support as per the selected DA and framework. However, considering the fact that the application could be started as a web application, this requirement can be considered met to at least some extent. The latter one, MC-5.0, can be deemed as partially reached as the development was only relatively efficient due to the learning curve of the framework. The remaining requirements could be finalized relatively easy during the next iteration of development.

The following requirements were not reached: MC-1.2, MC-1.2.1, MC-5.1 and MC-5.2. These were mostly due to the relatively short development time.
However, at least MC-1.2 and MC-1.2.1 could be estimated to take only a couple of hours. MC-5.1, on the other hand, was not met as there was no real-time testing. This could probably have been achieved if the development was done with a plain browser first. However, in the end the application should be tested on emulators or real physical devices due to access to native features which does not seem to be possible to be commenced in real-time. Regarding MC-5.2, the value could not be determined as maintainability can really be evaluated after a relatively long maintenance period during which at least a reasonable amount of changes would be added to the software.

6.2 Apache Cordova as the selected framework

As a relatively complete prototype of My Hotel was able to be achieved with Cordova, it can be considered as quite suitable for My Hotel. A summary of reflection of the success of Cordova as the framework with respect to the WSM calculations commenced in Tables 5.3 and 5.4 is shown in Table 6.1.

<table>
<thead>
<tr>
<th>SC</th>
<th>Hybrid DA</th>
<th>Cordova</th>
<th>Match</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>-</td>
<td>3</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC2</td>
<td>-</td>
<td>3</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC3</td>
<td>-</td>
<td>4</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC4</td>
<td>-</td>
<td>4</td>
<td>No</td>
<td>A value of 3 would be more suitable due to the encountered difficulties</td>
</tr>
<tr>
<td>SC5</td>
<td>1</td>
<td>-</td>
<td>No</td>
<td>2 or 3 would be more accurate as all the tools were quite easy to use</td>
</tr>
<tr>
<td>SC6</td>
<td>-</td>
<td>4</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC7</td>
<td>4</td>
<td>4</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC8</td>
<td>4</td>
<td>4</td>
<td>No</td>
<td>3 would have been more precise as, for example, documentation of the screen orientation plug-in was incomplete</td>
</tr>
<tr>
<td>SC9</td>
<td>3</td>
<td>4</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC10</td>
<td>3</td>
<td>0</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC11</td>
<td>4</td>
<td>2</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC12</td>
<td>4</td>
<td>4</td>
<td>No</td>
<td>3 would be more preferable as the tutorial did not explain structure of the example project. Additionally, more help with AVD and cordova emulate android command would have been needed</td>
</tr>
<tr>
<td>------</td>
<td>---</td>
<td>---</td>
<td>----</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SC13</td>
<td>3</td>
<td>4</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC14</td>
<td>3</td>
<td>-</td>
<td>OK</td>
<td>Could not be verified yet</td>
</tr>
<tr>
<td>SC15</td>
<td>2</td>
<td>2</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC16</td>
<td>3</td>
<td>3</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC17</td>
<td>-</td>
<td>4</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC18</td>
<td>4</td>
<td>-</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC19</td>
<td>3</td>
<td>-</td>
<td>No</td>
<td>4 would probably be more accurate as no performance issues could be detected despite it being mentioned as a disadvantage of hybrid DA in Section 3.1</td>
</tr>
<tr>
<td>SC20</td>
<td>4</td>
<td>-</td>
<td>OK</td>
<td>Although iOS could not be tested, a web application could be created. Thus, this is considered as 4</td>
</tr>
<tr>
<td>SC21</td>
<td>4</td>
<td>-</td>
<td>OK</td>
<td>Could not be evaluated during the first iteration</td>
</tr>
<tr>
<td>SC22</td>
<td>3</td>
<td>-</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC23</td>
<td>3</td>
<td>-</td>
<td>No</td>
<td>A value of 2 would probably be more precise as the DA and framework do not truly affect the aesthetics of the end result</td>
</tr>
<tr>
<td>SC24</td>
<td>2</td>
<td>-</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>SC25</td>
<td>3</td>
<td>3</td>
<td>No</td>
<td>1 or 2 would be more suitable as there was quite a bit of new skills to learn due to the amount and complexity of the required tools. Affected also by some weaknesses of the tutorial as mentioned with SC12</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SC26</td>
<td>3</td>
<td>-</td>
<td>OK</td>
<td>Could not be tested as a longer evaluation period is required to see effects on maintainability</td>
</tr>
<tr>
<td>SC27</td>
<td>2</td>
<td>2</td>
<td>No</td>
<td>Better testability could be achieved by developing the application as an ordinary web application first with the help of cordova server [port] command as stated in the CLI reference part of the documentation. Thus, a more precise value for SC27 could be 3</td>
</tr>
</tbody>
</table>

Table 6.1: DA and framework evaluation

As 19 selection criteria out of 27 can be deemed accurate, and the remaining ones would have to be altered slightly only, the framework can be considered relatively suitable in the perspective of selection criteria as well. Criteria with the most significant changes would be SC5 and SC25. SC5 should have been evaluated higher as tools of the framework were relatively easy to handle after getting used to them, especially the CLI commands. SC25, on the other hand, would be downgraded as the learning curve was a bit steeper than anticipated.

Unfortunately, some of the aspects of the framework with respect to the selection criteria could not be evaluated due to the relatively short development time frame. However, considering the challenges of mobile application development listed in Section 3.1, most of them can be considered as solved thanks to the hybrid DA. Security and UX are the only challenges that can not be solved purely by using hybrid DA and Cordova as no benefits regard-
ing security were encountered in the documentation during the development, and the successfulness of UX depends on developer skills or additional tools. As stated before, UX may be taken into account by using another framework, such as Ionic, in addition to Cordova.

6.3 Used selection process

Although evaluation of the selection process was not stated in the goals of this thesis in Section 1.2, some evaluation of it is commenced in this section as it had a significant role regarding the achieved end result. Overall, the used criteria were mostly suitable for the process and assisted greatly in selecting the most fitting framework. However, as there was quite a lack of scientific research regarding the selection of a DA, the criteria were utilized for selecting the DA as well. This can be deemed somewhat controversial as the list of criteria was formed for frameworks. During the selection process it was also noted that there would be need for one more SC: licensing. This is due to the fact that, in some situations, developers might want to favour some other licensing model over another. Regarding the quality of the used criteria, there exists yet two more issues. First, most of the criteria can be considered developer-centric and non-functional. This might cause a bias towards favouring developer-friendliness instead of users which conflicts with the principle of developing an application for users, not developers. Second, some of the used criteria ended up being bit ambiguous and even partially overlapping. For example, the difference between SC1 (API capabilities) and SC17 (functionality) is not clear enough. Additionally, SC18 and SC20 may be considered to be a subset of SC1. In this thesis, SC1 was interpreted as "additional API capabilities" that do not directly affect the functionality or major aspects of the application from the developer’s perspective.

Additionally, albeit the selected criteria were quite sufficient and justified for My Hotel, the use of WSM as MCDM method may be reckoned as contentious as there exists quite a bit of other MCDM methods, such as Promethee, that may be more suitable for mobile application tool selection. WSM was used due to the fact that it is simple and also as there was a lack of scientific literature for this aspect as well. Although WSM usage can be deemed controversial, it was quite suitable for the core of the selection process as the comparison could effectively be commenced with simple spreadsheet calculation.

Assuming that the used method and criteria are considered rational and justified, the used weights may be considered controversial as well. This is due to the fact that the weights may be defined in countless other ways.
Thus, as multiple decision makers can be expected to be more rational than a small developer team, the weight definitions should be done with a large amount of designers and developers to achieve more realistic weights. Additionally, if the usable weights are not limited in any way, one criterion could be significantly favoured over others. For example, if SC13 was given a weight of 1 million, PWA would have clearly been most suitable DA with hybrid DA being second. Thus, using WSM may lead to a biased selection if weights are defined arbitrarily.

In addition to the issues and criticism mentioned above, the following issues should be noted as well with the used process:

- The WSM-based comparison for DAs could have been more thorough: some cells in the comparison were left blank due to the lack of literature. This does not automatically imply that the real value for \( a_i \) would be greater or lesser than the average value of 2. To overcome this, more references should have been used to cover all selection criteria for all DAs.

- The process favours frameworks that contain all imaginable features and is thus more suitable for that approach. For some developers, at least in some cases, finding as light-weight framework as possible is more preferable. Thus, some developers may consider the frameworks deemed suitable by the used process too heavy. This could, however, be taken into account by omitting some criteria or even by giving negative weights for undesired features.

- Some of the used metrics for framework comparison are subjective rather than objective. This could, though, be overcome by commencing a qualitative research with multiple expert opinions, for example, although that could not be considered satisfying in all use cases.

- Some of the used metrics depend on open-sourcedness of the framework as they could not be determined otherwise. Thus the used process is suitable for only open-sourced frameworks in practice. As a solution, other metrics should be used if possible.

- Due to the nature of some selection criteria, such as SC7, SC8, SC10, SC11 and SC14, the process tends to favour more matured frameworks over novel alternatives. In some situations, a more novel framework could be more feasible than a mature one.

Despite the issues and controversy regarding all aspects related to the selection process, it can be considered as at least quite feasible for the case
of My Hotel. This is due to the reasons that that it was quite simple and fast to utilize, and it ended up choosing probably the most mature DA and framework. Quite likely, the process, or at least the selection criteria, could be used in other situations as well.
Chapter 7

Conclusions

This thesis had three objectives: to discover a feasible pattern and criteria for selecting a framework, to utilize them in framework selection regarding a mobile application called My Hotel, and finally to develop the application with the selected framework. The goals were met as criteria and selection method were defined, and the application was developed relatively completely. Additionally, before selecting the framework, different development approaches were compared.

After determining the requirements for My Hotel, 9 challenges regarding mobile application development were discovered in literature. To overcome at least some of the challenges one of 7 different development approaches could be used: native, web, progressive web application, hybrid, model-driven, cross-compiled or interpreted. Concerning the most relevant criteria of mobile software development, a total of 27 different criteria were discovered thanks to previous literature. Method used for multiple-criterion decision-making regarding the competing development approaches and frameworks was weighted sum method due to the lack of consensus of a more viable selection process. As a result of the used method and criteria, the chosen development approach for My Hotel was hybrid, and Apache Cordova ended up being the selected framework. After evaluation of the application, framework and used process it was discovered that the used criteria and process were quite suitable in the case of My Hotel.

Although the thesis discovered quite a lot of aspects that one should take into account during mobile application development, some questions still remain. Most relevant conundrum of this thesis concerns the suitability of the chosen development approach and framework: were the selections really the most suitable out of all the options? At least some of the used selection criteria could not be determined during the thesis, such as maintainability. In addition to maintainability, cross-platform support of Apache Cordova could
CHAPTER 7. CONCLUSIONS

not be extensively evaluated as iOS could not be targeted with the available
development tools. The metrics regarding the criteria for framework selection
could also be a relevant research subject as some the used metrics could not
be applied to some frameworks and could be more generic and profound.

Regarding future research, suitability of the selection process should be
evaluated in other fields of software development as well as the amount of
technologies is not increasing only for mobile application development. Addition-
ally, it would be helpful to divide selection criteria in two parts: general
criteria, that would apply to all software development, and specific criteria
that would apply to different fields of software development such as web ap-
application development. One rather relevant point of research could also be to
evaluate all development approaches by developing one specific application
by utilizing all the possible approaches. This would allow developers to have
a practical comprehension of advantages and disadvantages of each approach.

From a more general perspective, more research concerning different multi-
ple-criterion decision-making methods, at least in the field of computer sci-
ence, would be recommended as the point of having MCDM methods is
to assist in making as solid long-term decisions as possible and not be an-
other source of complexity, causing even more difficulty of choice. Regarding
practical solutions to the challenge of increasing amount of tools from the
developer’s perspective, a guide or book regarding tool selection in the world
of increasing amount of choices would be suggested. Also a tool, such as
a web application containing all available tools, that would help developers
with decision-making itself would be useful.
Bibliography


Appendix A

Complete list of requirements

A complete list of requirements for the MVP of My Hotel are listed here. These were utilized for determining weights in Section 5.2:

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC-1.0</td>
<td>View for employees</td>
<td>A view designated to employees of LC’s.</td>
</tr>
<tr>
<td>MC-1.1</td>
<td>Employee task list</td>
<td>A view that allows employees to read and update their tasks.</td>
</tr>
<tr>
<td>MC-1.1.1</td>
<td>Employee task list real-time CMS updates</td>
<td>The task list view for employees will be updated in real-time as set in the company CMS.</td>
</tr>
<tr>
<td>MC-1.2</td>
<td>Employee service requests</td>
<td>A view that allows company employees to read and update service requests created by customers.</td>
</tr>
<tr>
<td>MC-1.2.1</td>
<td>Real-time CMS updates for employee service requests</td>
<td>The service requests view for employees will be updated in real-time as per requests created by customers.</td>
</tr>
<tr>
<td>MC-2.0</td>
<td>View for customers</td>
<td>A view dedicated to staying customers.</td>
</tr>
<tr>
<td>MC-2.1</td>
<td>General LC information</td>
<td>A view for customers that allows them to see general information about the LC.</td>
</tr>
<tr>
<td>MC-2.1.1</td>
<td>Details of LC information</td>
<td>General information of the LC should include the following details: name, address, coordinates, phone number.</td>
</tr>
<tr>
<td>MC-2.2</td>
<td>Room-specific information</td>
<td>View for the customer for inspecting details about their room.</td>
</tr>
<tr>
<td>MC-2.2.1</td>
<td>Details for room specific information</td>
<td>The following details should be implemented for room-specific information: surface area, minibar, maximum amount of guests, floor.</td>
</tr>
<tr>
<td>MC-2.3</td>
<td>Service request view for customers</td>
<td>The customers should be able to create, read, update and delete (CRUD) service requests for employees of their LC.</td>
</tr>
<tr>
<td>MC-2.3.1</td>
<td>Easy extension to stay (service request)</td>
<td>The customers should be able to request an extension to their stay as intuitively and easily as possible.</td>
</tr>
<tr>
<td>MC-2.4</td>
<td>&quot;Do not disturb&quot; flag for room</td>
<td>Customers should be able to set a &quot;Do not disturb&quot; flag for their room.</td>
</tr>
<tr>
<td>MC-3.0</td>
<td>Graphical user interface (GUI)</td>
<td>The MC should implement as intuitive and modern-looking GUI as achievable.</td>
</tr>
<tr>
<td>MC-3.1</td>
<td>Decent UX</td>
<td>The application should have as native look and feel as possible and be performant enough in most situations.</td>
</tr>
<tr>
<td>MC-4.0</td>
<td>Cross-platform support</td>
<td>The MC should be operable on both Google Android and Apple iOS systems.</td>
</tr>
<tr>
<td>MC-5.0</td>
<td>Efficient development</td>
<td>The application development should be as effortless as possible and minimize the risk of fragmentation. Additionally, the framework, other necessary tools and programming languages should be easy to assimilate as rapidly as possible.</td>
</tr>
<tr>
<td>MC-5.1</td>
<td>Efficient testing</td>
<td>The application should be testable as quickly as possible during development and have overall good testing support.</td>
</tr>
<tr>
<td>MC-5.2</td>
<td>Maintainability</td>
<td>Changes to the application in the future should be achievable as effortlessly as possible.</td>
</tr>
</tbody>
</table>
Appendix B

Getting started with Apache Cordova

For the sake of evaluating the ease of installation and initializing an example project, an example application is created in this section. Due to the fact that the development was not started on a fresh system, the environment before evaluation was as is stated in Table B.1.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Windows 10 Pro</td>
<td>19041</td>
</tr>
<tr>
<td>Interpreters</td>
<td>Node.js</td>
<td>13.3.0</td>
</tr>
<tr>
<td>VCS</td>
<td>Git Bash</td>
<td>2.21.0.windows.1</td>
</tr>
<tr>
<td>IDE</td>
<td>Android Studio</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Table B.1: System environment before evaluation

As suggested by the documentation of Cordova at [3], the following commands were run to initialize the project:

> npm install -g cordova
> cordova create hello com.example.hello HelloWorld
> cd hello/
> cordova platform add ios
> cordova platform add android

The command cordova requirements was run to check whether or not the system met the requirements for Cordova. At this point, the first issue was encountered:

> cordova requirements
Requirements check results for android:
Java JDK: installed 1.8.0
Android SDK: installed true
Android target: installed android-29
Gradle: not installed
Could not find an installed version of Gradle either in Android Studio, or on your system to install the gradle wrapper. Please include gradle in your path, or install Android Studio

Requirements check results for ios:
Apple macOS: not installed
Error: Cordova tooling for iOS requires Apple macOS
Some of requirements check failed

Despite the fact that Android Studio was installed beforehand and that it had Gradle already, an error was encountered regarding Gradle installation. Fortunately, however, Java installation was detected to be working properly. While attempting to overcome the issue, Android Studio was updated to version 4.1.1. After trying multiple solutions, only a manual installation of Gradle as discussed at https://gradle.org/install/ fixed the problem. Additionally, according to the console output above, macOS seems to be required for building, emulating and debugging the application which will cause issues as the development is done with a Windows-based system.

When building the application for Android for the first time with the command cordova build android, a second, rather minor, issue was encountered regarding the build path as it contained illegal characters. This was due to the use of Scandinavian characters in the path. As a solution, the project directory had to be moved elsewhere.

After managing to get the project built, a few more issues were encountered while attempting to emulate the application for the first time with the following command: cordova emulate android. First of the issues was as follows:

> cordova emulate android
Checking Java JDK and Android SDK versions
ANDROID_SDK_ROOT=undefined (recommended setting)
ANDROID_HOME=undefined (DEPRECATED)
Using Android SDK: C:\Users\<USER>\AppData\Local\Android\sdk
Subproject Path: CordovaLib
Subproject Path: app
Deprecation Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use `--warning-mode all` to show the individual deprecation warnings.
See https://docs.gradle.org/6.5/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 1s
40 actionable tasks: 40 up-to-date
Built the following apk(s):
C:\myhotel2\client\hello\platforms\android\app\build\outputs\apk\debug\app-debug.apk
Checking Java JDK and Android SDK versions
ANDROID_SDK_ROOT=C:\Users\<USER>\AppData\Local\Android\sdk (recommended setting)
ANDROID_HOME=undefined (DEPRECATED)
Using Android SDK: C:\Users\<USER>\AppData\Local\Android\sdk
No emulator images (avds) found.
1. Download desired System Image by running: "C:\Users\<USER>\AppData\Local\Android\sdk\tools\android.BAT" sdk
2. Create an AVD by running: "C:\Users\<USER>\AppData\Local\Android\sdk\tools\android.BAT" avd
HINT: For a faster emulator, use an Intel System Image and install the HAXM device driver

android.BAT file was attempted to be executed to create and Android Virtual Device (AVD) as suggested by the console output. However, as the second issue, using android.BAT did not work as expected as the android command had been deprecated. Consequently, the AVD was created by using the AVD manager of Android Studio to overcome the issue. The AVD was created by using the specifications mentioned in Table B.2. After creating the AVD manually and starting it once via Android Studio, the issue was solved.

<table>
<thead>
<tr>
<th>Type</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Pixel 2</td>
</tr>
<tr>
<td>Resolution</td>
<td>1080x1920</td>
</tr>
<tr>
<td>Dots per inch (DPI)</td>
<td>420</td>
</tr>
<tr>
<td>OS</td>
<td>Android Q (v10.0, API level: 29)</td>
</tr>
</tbody>
</table>

Table B.2: AVD specifications
Structure of the generated project is as follows:

```bash
> tree /a /F
Folder PATH listing
Volume serial number is 2877-3DD4
C:.
  .gitignore
  config.xml
  package-lock.json
  package.json

  ---node_modules
  <All NodeJS modules listed here>

  ---platforms
  <All Cordova platform-command generated files and folders here>

  ---plugins
  | android.json
  | fetch.json
  | ios.json

  \\
  \\
  ---cordova-plugin-whitelist
  | CONTRIBUTING.md
  | LICENSE
  | NOTICE
  | package.json
  | plugin.xml
  | README.md
  | RELEASENOTES.md

  \\
  ---src
  | \\
  | android
  | WhitelistPlugin.java

  \\
  ---tests
  | package.json
  | plugin.xml
  | README.md
  | tests.js

  \\
  ---scripts
```
In the project structure above it can be seen that the application is indeed based on web technologies as there are .html, .css and .js files in the www directory. It should be noted, though, that the following files and folders were generated by the cordova platform add commands: node_modules, package.json, package-lock.json, platforms and plugins. Consequently, these seem to be related to building and emulating the application on specific platforms. config.xml contains metadata related to the project such as name, description, author and platforms. Rather interestingly, as can be seen from the project structure, even the Hello World application utilizes a plug-in called cordova-plugin-whitelist which, according to the documentation, allows navigation in the WebView to any whitelisted uniform resource locators (URLs) [7].

At this point, finally, the Hello World application was able to be emulated with cordova emulate android. The main view of the application is shown in Figure B.1. It took approximately 2.5 hours to get the example application to run due to the encountered issues.
Figure B.1: Hello World application of Apache Cordova
Appendix C

DA literature: pros and cons

This appendix mentions which statements in reviewed literature regarding development approaches were considered advantages or disadvantages in Section 3.1.2. Aspects interpreted from statements in the used sources are only listed here.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Interpreted from statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>No application updates</td>
<td>”[Web-based applications] exist in a similar fashion across mobile web browsers on all platforms. Thus, no mobile application updates are required.” [29]</td>
</tr>
<tr>
<td>Reduced development time and cost</td>
<td>”[...] lessens development cost and time” [18]</td>
</tr>
<tr>
<td>Partial access to native functionalities</td>
<td>”[...] access to the device’s native functionalities (such notifications system, GPS, Contact list, etc) is limited” [29]</td>
</tr>
<tr>
<td>Partial access to native functionalities</td>
<td>”[...] no access to use device specific hardware features, i.e., camera or accelerometer” [18]</td>
</tr>
<tr>
<td>Low performance</td>
<td>”[...] the time it takes to render the web pages by loading them from the network is longer than that of the native mobile user interface” [29]</td>
</tr>
<tr>
<td>Not publishable on application stores</td>
<td>”[...] web applications are only accessible via a URL and cannot be made readily available on mobile app stores. This would have a diminishing impact on the approach’s attractiveness.” [29]</td>
</tr>
</tbody>
</table>

Table C.1: Advantages and disadvantages of web-based DA as stated in the literature
## APPENDIX C. DA LITERATURE: PROS AND CONS

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Interpreted from statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages of web-based DA</td>
<td>&quot;[...] a PWA is a web app with enhanced capabilities&quot; [19]</td>
</tr>
<tr>
<td>Native look and feel achievable</td>
<td>&quot;[PWA-based DAs] allow for web apps to look and feel like a regular Native or cross-platform built app&quot; [19]</td>
</tr>
<tr>
<td>Native look and feel achievable</td>
<td>&quot;Due to being web-based, the user interface of PWAs can be designed to look and feel similar to Native apps&quot; [19]</td>
</tr>
<tr>
<td>Offline use possible</td>
<td>&quot;[PWAs] allow for offline usage of the website&quot; [19]</td>
</tr>
<tr>
<td>Offline use possible</td>
<td>&quot;[...] app-like feeling of using a PWA compared to a regular website browsed in a traditional way [after being installed by visiting the relevant website]&quot; [19]</td>
</tr>
<tr>
<td>UX on iOS (disadvantage)</td>
<td>&quot;[On iOS-systems:] e.g. lack of state management between sessions, and white screens in the App Switcher&quot; [19]</td>
</tr>
</tbody>
</table>

Table C.2: Advantages and disadvantages of PWA-based DA as stated in the literature

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Interpreted from statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native performance</td>
<td>&quot;[...] the applications are able to attain native performance&quot; [29]</td>
</tr>
<tr>
<td>Native performance</td>
<td>&quot;Cross-compiled approach do not rely on [...] on-device (JavaScript) interpreters&quot; [19]</td>
</tr>
<tr>
<td>Native performance</td>
<td>&quot;Neither the use of- nor the access to Native device features is controlled by such a layer [as with interpreted or hybrid DAs]&quot; [19]</td>
</tr>
<tr>
<td>Access to native functionalities</td>
<td>&quot;[Applications developed with cross-compiled DA] deliver all the features of native applications&quot; [29]</td>
</tr>
<tr>
<td>Native UI</td>
<td>&quot;[...] native interface components&quot; [29]</td>
</tr>
<tr>
<td>Native UI</td>
<td>&quot;Cross-compiled approach do not rely on WebView components&quot; [19]</td>
</tr>
<tr>
<td>Native UI</td>
<td>&quot;[...] generated user interfaces, specifically how they are rendered as Native interface components&quot; [19]</td>
</tr>
</tbody>
</table>
### APPENDIX C. DA LITERATURE: PROS AND CONS

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Interpreted from statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages of web-based DA</td>
<td>&quot;[…] advantages of web technologies and those of native functionalities&quot; [29]</td>
</tr>
<tr>
<td>Advantages of native DA</td>
<td>&quot;[…] advantages of web technologies and those of native functionalities&quot; [29]</td>
</tr>
<tr>
<td>Advantages of native DA</td>
<td>&quot;[Cordova will] generate a new Native app including a WebView and two-way communication between the WebView and Native code&quot; [19]</td>
</tr>
<tr>
<td>Publishable on application stores</td>
<td>&quot;[…] distributable through application stores&quot; [29]</td>
</tr>
<tr>
<td>Publishable on application stores</td>
<td>&quot;[…] publishable, deployable Native app&quot; [19]</td>
</tr>
<tr>
<td>Publishable on application stores</td>
<td>&quot;[…] available for download through the platform’s application distribution store [1], [4]&quot; [18]</td>
</tr>
<tr>
<td>Access to native functionalities</td>
<td>&quot;[…] native features are available through the abstract layer&quot; [29]</td>
</tr>
<tr>
<td>Access to native functionalities</td>
<td>&quot;[…] bridging [1], allows developers to communicate with platform-specific Native code from within a non-native environment&quot; [19]</td>
</tr>
<tr>
<td>Access to native functionalities</td>
<td>&quot;[…] can access the device hardware features&quot; [18]</td>
</tr>
<tr>
<td>Popularity</td>
<td>&quot;In addition to providing easy Hybrid app initialization, Cordova also provides a plugin system with thousands of available plugins, including such as camera access, GPS access and contact list access, features requiring the aforementioned bridging system to function&quot; [19]</td>
</tr>
<tr>
<td>Popularity</td>
<td>&quot;[…] highly popular amongst cross-platform developers&quot; [19]</td>
</tr>
<tr>
<td>Native UI achievable</td>
<td>&quot;[…] the user interface may look as Native-app-like or non-Native-app-like as one may wish&quot; [19]</td>
</tr>
</tbody>
</table>

Table C.3: Advantages and disadvantages of cross-compiled DA as stated in the literature
## APPENDIX C. DA LITERATURE: PROS AND CONS

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Interpreted from statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low effort</td>
<td>“ [...] allow for re-use of existing knowledge for web developers” [19]</td>
</tr>
<tr>
<td>Alternative frameworks</td>
<td>“ [...] an alternative to Cordova, named Capacitor, is being developed by the Ionic Framework team” [19]</td>
</tr>
<tr>
<td>Low performance</td>
<td>“ [...] inferior in performance compared to the native interfaces since the execution happens in the browser engine” [29]</td>
</tr>
<tr>
<td>Native UI and UX not guaranteed</td>
<td>“ [...] the interface does not have access to the native look and feel” [29]</td>
</tr>
<tr>
<td>Native UI and UX not guaranteed</td>
<td>“ [...] developing Native-app-like user interfaces which adhere to the interface guidelines of all supported platforms [...] may be challenging and time-consuming to do from scratch” [19]</td>
</tr>
<tr>
<td>Native UI and UX not guaranteed</td>
<td>“ [...] additional tools and libraries should be used to develop Native-like and Native-feeling user interfaces and interactions” [19]</td>
</tr>
<tr>
<td>Requires additional tools</td>
<td>“Cordova library only provides the foundation” [19]</td>
</tr>
<tr>
<td>Requires additional tools</td>
<td>“ [...] additional tools and libraries should be used to develop Native-like and Native-feeling user interfaces and interactions” [19]</td>
</tr>
</tbody>
</table>

Table C.4: Advantages and disadvantages of hybrid DA as stated in the literature
<table>
<thead>
<tr>
<th>High abstraction level</th>
<th>&quot;[...] one of the philosophies behind the Model-Driven approach is enabling non-developers and non-technical users with domain expertise to model line-of-business apps based on a provided DSL – being textual or graphical [79]&quot; [19]</th>
</tr>
</thead>
<tbody>
<tr>
<td>High abstraction level</td>
<td>&quot;[...] developing apps across mobile platforms will require knowledge of the DSL rather than Objective-C and Java&quot; [19]</td>
</tr>
<tr>
<td>Native performance</td>
<td>&quot;[...] performance of native applications&quot; [29]</td>
</tr>
<tr>
<td>Native performance</td>
<td>&quot;Generators will then convert the models/code into Native source code for the targeted platforms&quot; [19]</td>
</tr>
<tr>
<td>Native UI</td>
<td>&quot;[...] user interface is totally implemented with the native component [sic]&quot; [29]</td>
</tr>
<tr>
<td>Customizability</td>
<td>&quot;[...] the language and the code generator can be improved&quot; [29]</td>
</tr>
<tr>
<td>Maintainability (disadvantage)</td>
<td>&quot;Developers have to maintain the model of the application in order to maintain the native code for each platform&quot; [29]</td>
</tr>
<tr>
<td>Limited programmability</td>
<td>&quot;[...] limited to the application domain of the model language. The only applications that can be modeled are those that fall into the category supported by the model&quot; [29]</td>
</tr>
<tr>
<td>Requires expertise despite having a high abstraction level</td>
<td>&quot;[...] generated code remains incomplete and should be manually completed with the use of the native language and SDK tools. Thus, this code should be written individually for each platform&quot; [29]</td>
</tr>
<tr>
<td>Cross-platform developement (disadvantage)</td>
<td>&quot;[...] generated code remains incomplete and should be manually completed with the use of the native language and SDK tools. Thus, this code should be written individually for each platform&quot; [29]</td>
</tr>
<tr>
<td>Popularity (disadvantage)</td>
<td>&quot;[...] technical implementations building on the MDD approach are rare among practitioners and in developer communities&quot; [19]</td>
</tr>
</tbody>
</table>
Table C.5: Advantages and disadvantages of model-driven DA as stated in the literature

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Interpreted from statement</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native functionalities</td>
<td>&quot;[...] enables developers to utilize full capabilities of a device&quot; [18]</td>
<td></td>
</tr>
<tr>
<td>Native performance</td>
<td>&quot;Native apps are known for their better performance&quot; [18]</td>
<td>Interpreted as &quot;native performance&quot; as the DA in question is native DA and the source states it to have better performance</td>
</tr>
<tr>
<td>Publishable on application stores</td>
<td>&quot;[...] available for download via platform’s dedicated application stores&quot; [18]</td>
<td></td>
</tr>
<tr>
<td>Increased development time and cost</td>
<td>&quot;[...] demands additional resources (skills, time and costs)&quot; [18]</td>
<td></td>
</tr>
</tbody>
</table>

Table C.6: Advantages and disadvantages of native DA as stated in the literature

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Interpreted from statement</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native UI</td>
<td>”Interpreted approach use [sic] common language (like JavaScript or others) to write the code of user interface and generate the equivalent for native component for each platform” [29]</td>
<td></td>
</tr>
<tr>
<td>Native UI</td>
<td>”[...] native user interfaces” [29]</td>
<td></td>
</tr>
<tr>
<td>Native UI</td>
<td>”Interpreted apps do not rely on a WebView component to render a bundled website [29]” [19]</td>
<td></td>
</tr>
<tr>
<td>Native UI</td>
<td>”[...] can render actual Native user interface components to the screen” [19]</td>
<td></td>
</tr>
<tr>
<td>Native UI</td>
<td>”[...] generate Native user interfaces” [19]</td>
<td></td>
</tr>
<tr>
<td>Native functionalities</td>
<td>”Native features are provided by an abstract layer that interprets the code on runtime” [29]</td>
<td></td>
</tr>
<tr>
<td>Native functionalities</td>
<td>”[...] has access to Native device features, the Interpreted approach” [19]</td>
<td></td>
</tr>
<tr>
<td>Dependence on the development environment</td>
<td>”[...] dependence on the development environment” [29]</td>
<td></td>
</tr>
<tr>
<td>Dependence on the development environment</td>
<td>”[...] new platform-specific features such as new user interface features would not be made available to applications unless they are supported by the development environment” [29]</td>
<td></td>
</tr>
<tr>
<td>Updateability</td>
<td>”[...] new platform-specific features such as new user interface features would not be made available to applications unless they are supported by the development environment” [29]</td>
<td></td>
</tr>
<tr>
<td>Lower performance</td>
<td>”[...] performance degradation that is caused by calling the abstract layer on runtime” [29]</td>
<td></td>
</tr>
<tr>
<td>Lower performance</td>
<td>”[...] does not compile, convert or transpile the codebase into Native byte code” [19]</td>
<td></td>
</tr>
<tr>
<td>Framework-lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Low popularity</td>
<td>”[Different framework APIs] fragments frameworks and developer communities of the Interpreted approach greatly” [19]</td>
<td>Interpreted as ”low popularity” due to the fact that the framework, that will be selected, will have smaller developer community as the community is fragmented</td>
</tr>
</tbody>
</table>

Table C.7: Advantages and disadvantages of interpreted DA as stated in the literature