

Master's Programme in Computer, Communication and Information Sciences

Selecting a Development Approach for Mobile App Development: A Framework and Case Study

Niclas Vauhkonen

Copyright © 2025 Niclas Vauhkonen

Author Niclas Vauhkonen

Title Selecting a Development Approach for Mobile App Development: A Framework and Case Study

Degree programme Computer, Communication and Information Sciences

Major Software and Service Engineering

Supervisor Prof. Casper Lassenius

Advisor Prof. Casper Lassenius

Collaborative partner Accountor HR Solutions Oy

Date 21.9.2025

Number of pages 83+5

Language English

Abstract

This thesis will present a requirement-based evaluation framework. The thesis will then use the frameworks to evaluate development approaches for an employee self-service app (case app). The thesis will answer the following research questions: (1) what are the requirements regarding the case app, (2) which of the case app requirements are met for each technical framework and the native approach, (3) based on the met case app requirements, how do the technical frameworks and the native approach compare, and (3.1) based on the technical framework and the native approach comparison, how do the development approaches compare?

To answer the research question, the thesis elicited requirements with documentation analysis, focus groups, and interviews. The data gained from the focus groups and the interviews were analyzed. Finally, the evaluation framework in the thesis would be used to filter the elicited requirements, choose technical frameworks each development approach, and compare the technical frameworks from which the development approaches could be compared.

The thesis identified fourteen requirements that could be compared and that were not met by all the technical frameworks. These requirements were categorized into hardware features as well as functional, quality, and other requirements.

Keywords Capacitor, Ionic, Mendix, NativeScript, PWA, Vue.js, Xamarin, app, application, approach, case study, cross-compiled, development, focus group, framework, functional, hardware feature, hybrid, interpreted, interview, mobile, model-driven, native, progressive, quality, requirement, web

Författare Niclas Vauhkonen

Titel Väljande av utvecklingsmetod för mobilappsutveckling: ett ramverk och en fallstudie

Utbildningsprogram Computer, Communication and Information Sciences

Huvudämne Software and Service Engineering

Övervakare Prof. Casper Lassenius

Handledare Prof. Casper Lassenius

Samarbetspartner Accountor HR Solutions Oy

Datum 21.9.2025

Sidantal 83+5

Språk engelska

Sammandrag

Detta examensarbete kommer att presentera ett kravbaserat utvärderingsramverk. Arbetet kommer sedan att använda ramverket för att utvärdera utvecklingsmetoder för en app avsett för anställda som självbetjäningssapp (appen). Arbetet kommer att ge svar på tre studiefrågor: (1) vad är kraven för appen, (2) vilka av appens krav är bemötta för varje tekniska ramverk och för den nativa metoden, (3) baserat på de bemötta kraven för appen, hur jämförbara är de tekniska ramverken och den nativa metoden med varandra, och (3.1) baserat på de tekniska ramverken och den nativa metoden, hur är utvecklingsmetoderna jämförbara med varandra?

För att ge svar på studiefrågorna, insamlade detta examensarbete krav med hjälp av dokumentanalys, fokusgrupper och intervjuer. Data som erhöles från fokusgrupperna och intervjuerna analyserades. Slutligen kommer utvärderingsramverket i arbetet att användas för att filtrera de insamlade kraven, att välja tekniska ramverk för varenda en utvecklingsmetod och att jämföra de tekniska ramverken med varandra från vilket utvecklingsmetoderna är möjliga att jämföras med varandra.

Detta examensarbete identifierade fjorton krav som kunde användas i jämförelsen och som inte bemötes av alla de tekniska ramverken. Dessa krav kategoriserades i hårdvaru-, funktionella, icke-funktionella och andra krav.

Nyckelord Capacitor, Ionic, Mendix, NativeScript, PWA, Vue.js, Xamarin, app, applikation, cross-plattformapp, cross-plattformapplikation, evalueringsramverk, fallstudie, fokusgrupp, funktionell, hybridapp, hybridapplikation, hårdvarukrav, icke-funktionell, intervju, krav, low code plattform, low-code-plattform, mobil, model-driven, nativ app, nativ applikation, nativ, native, plattformsoberoende, plattformsspecifik, program, progressiv, ramverk, teknisk ramverk, tolkad app, tolkad applikation, utvecklingsmetod, webbapp, webbapplikation

Contents

Abstract	3
Abstract (in Swedish)	4
Contents	5
Abbreviations	7
1 Introduction	8
1.1 Background and research questions	8
1.2 Scope	9
1.3 Structure of the thesis	9
2 Related work	10
2.1 Development approaches	10
2.2 Framework candidates	16
2.3 Selection and evaluation frameworks	17
3 Case description	19
4 Methods	21
4.1 Requirement elicitation	21
4.2 Development approach evaluation	24
5 Requirements	26
5.1 Hardware features	26
5.2 System requirements	28
5.3 Functional requirements	29
5.4 Regulative/Legislative requirements	31
5.5 Quality requirements	32
5.6 Other requirements	36
6 Results	43
6.1 Frameworks	43
6.2 Evaluated requirements overview	46
6.3 Hardware features	46
6.4 System requirements	49
6.5 Functional requirements	50
6.6 Regulative/Legislative requirements	52
6.7 Quality requirements	52
6.8 Other requirements	54
6.9 Requirement summary	61

7	Discussion	63
7.1	RQ1: What are the requirements regarding the case app?	63
7.2	RQ2: Which of the case app requirements are met for each technical framework and the native approach?	63
7.3	RQ3: Based on the met case app requirements, how do the technical frameworks and the native approach compare?	63
7.4	RQ3.1: Based on the technical framework and the native approach comparison, how do the development approaches compare?	64
7.5	Limitations of the study	64
8	Conclusion	66
8.1	Thesis contribution	66
8.2	Future work	67
	References	68
A	Architect interview questions	84
B	Director of product development interview questions	85
C	Lead UI/UX designer interview questions	86
D	Product owner interview questions	87
E	Sales director interview questions	88

Abbreviations

A2HS	Add to home screen
API	Application Program(ming) Interface
CIAM	customer identity and access management
COVID-19	coronavirus disease 2019
CSS	Cascading Style Sheets
dp	density-independent pixel
DSL	Domain-Specific Language
EEA	European Economic Area
EU	European Union
GDPR	General Data Protection Regulation
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDE	integrated development environment
JSON	JavaScript Object Notation
MDD	Model-Driven Development
MDSO	Model-Driven Software Development
OIDC	OpenID Connect
OS	operating system
PIN	personal identification number
PWA	Progressive Web App
REST	Representational State Transfer
RESTful	system that follows the REST software architecture
RPA	robotic process automation
RWD	responsive web design
SDK	Software Development Kit
SMS	Short Message Service
TWA	Trusted Web Activities
UI	user interface
UX	user experience
WebAuthn	Web Authentication API
WCAG	Web Content Accessibility Guidelines

1 Introduction

1.1 Background and research questions

When developing a mobile application (app), one essential design decision is to decide on a development approach for the app. A development approach can be referred to as specific programming languages, tools, and development environments for creating an app for a particular platform (Biørn-Hansen, Grønli, and Ghinea 2018). Because a development approach potentially affects many essential aspects of developing an app, such as programming languages used, it is crucial to choose the correct development approach for the app. Choosing the correct development approach requires that development approaches are carefully evaluated. Using an evaluation framework can enable the careful evaluation of development approaches.

There are several studies (Ahti, Hyrynsalmi, and Nevalainen 2016; Khachouch et al. 2020; Lachgar and Abdali 2017; Rieger and Majchrzak 2019; Sommer and Krusche 2013) which present frameworks for evaluating mobile app development approaches. However, this thesis will argue that these frameworks are not best suited for evaluating development approaches when the main requirements for the app are known. To address this issue, this thesis will present a requirement-based evaluation framework. The thesis will then use the framework to evaluate development approaches for an employee self-service app (henceforth also known as the case app). In order to evaluate the development approaches, the thesis will answer the following research questions:

RQ1: What are the requirements regarding the case app?

RQ2: Which of the case app requirements are met for each technical framework and the native approach?

RQ3: Based on the met case app requirements, how do the technical frameworks and the native approach compare?

RQ3.1: Based on the technical framework and the native approach comparison, how do the development approaches compare?

A technical framework—also known as a software framework or just as a framework—is a template for developing apps or software components (Butterfield, Ngondi, and Kerr 2016). A framework provides the basic structure for the app and is basically the foundation on which the app is implemented on (Butterfield, Ngondi, and Kerr 2016). A framework may also provide other facilities, such as a runtime environment (Butterfield, Ngondi, and Kerr 2016).

This thesis will refer to two types of frameworks: technical and evaluation frameworks. Because the thesis will mostly address technical frameworks, the word framework may be used as a shorthand for this type of framework.

1.2 Scope

Despite evaluating development approaches based on technical frameworks selected for each approach, this thesis is not meant to be interpreted as a technical framework evaluation.

The thesis will only evaluate the development approaches based on relevant and the most important requirements for the case app and any requirement deemed optional will be ignored. For instance, if an interviewee stated that the case app *should* have a feature, then the feature will not be evaluated because the word ‘should’ carries the possibility not including the feature. However, if an interviewee would instead state that the case app *has to have* a feature, then the feature is required by the interviewee.

When evaluating PWAs, the combination of the operating system and the browser determines which requirements may be supported. In order to make the evaluation manageable, this thesis will evaluate one browser for each operating system. For Android, the browser will be Google Chrome, whereas for iOS, the browser will be Safari.

During the writing of this thesis, the support for Xamarin ended and apparently with it the documentation needed for assessing whether a requirement is fulfilled by Xamarin was taken down. The requirements that weren’t assessed for Xamarin are marked with ‘—’ in Table 19.

1.3 Structure of the thesis

The next chapter will present the related work. Chapter 3 will present and describe the employee self-service app and the software systems that are required for the app to function. Chapter 4 will describe the research methods used in this thesis. Chapter 5 will present and describe the elicited requirements. Chapter 6 will present the results. Chapter 7 will discuss the results. Finally, Chapter 8 makes a summary of the findings and present topics for future research.

2 Related work

Section 2.1 presents and describes the mobile app development approaches used in the comparison. Section 2.2 presents the framework candidates. Section 2.3 presents decision and evaluation frameworks found in other studies.

2.1 Development approaches

This chapter describes the mobile app development approaches shown in Figure 1. Figure 1 depicts how mobile app development approaches can be arranged as a hierarchy. Approaches can be divided into two main categories: the native and cross-platform approach. According to Rieger and Kuchen (2019): “The terms cross-platform or multi-platform development typically denote the creation of apps for multiple platforms *within the same device class*, for example iOS and Android in the smartphone domain—potentially extended to technically similar tablets”. According to Biørn-Hansen, Grønli, and Ghinea (2018), cross-platform approaches can be divided into further categories: the hybrid, interpreted, cross-compiled, model-driven, and progressive web app. However, this list of cross-platform approaches is not exhaustive (Biørn-Hansen, Grønli, and Ghinea 2018). It has to be noted that development approaches are categorized in a variety of ways (Majchrzak, Ernsting, and Kuchen 2015; Nunkesser 2018).

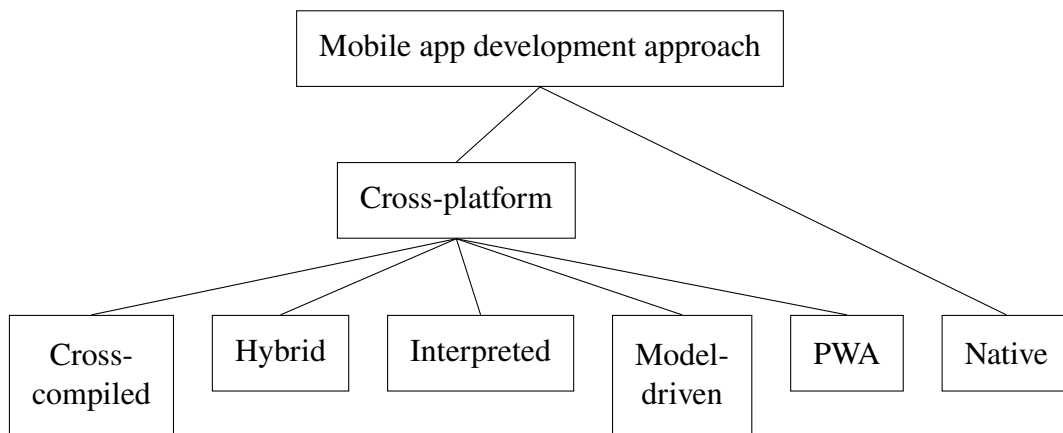


Figure 1: Hierarchy of mobile app development approaches.

Next, each section in this chapter describes an approach used in the comparison. The study by Biørn-Hansen, Grønli, and Ghinea (2018) has served as the main source of information about these different approaches.

2.1.1 Native approach

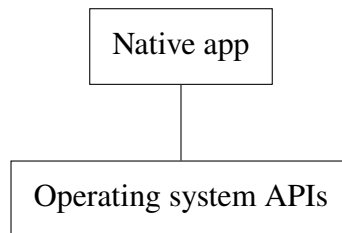


Figure 2: The architecture of a native app (The Apache Software Foundation 2021).

The native development technique refers to using a combination of programming languages and software specifically meant for producing apps for a specific platform. The software can include tools, development environments, and software development kits (SDK). For instance, with the Android Studio integrated development environment (IDE) one can develop Android apps.

The software used in the native approach tends to be developed, maintained, and even used by the platform owner. Because of this first-party support, the native approach is at least among the first that has access to the latest platform features.

The problematic part with the native approach is that app developed for one platform cannot be executed on another. Instead, in order for the app to support another platform it needs to be developed practically from the ground up to that platform. This means that the effort used for developing, testing, and maintaining an app is at worst roughly multiplied with the number of platforms that is supported. Besides this, the entity developing the app needs to have more platform-specific knowledge compared to the other development approaches.

2.1.2 Hybrid approach

The hybrid approach is a hybrid of a web app and a native app, hence the name of the approach. In this approach, the app is developed like a web app and deployed as a native app. This means that the app is developed with regular web technologies, such as HTML, CSS, and JavaScript, as well as that the resulting app can be served from a mobile app store and be used as a regular native app. Because hybrid apps are developed using regular web technologies, the hybrid approach is very popular among cross-platform developers (Biørn-Hansen, Grønli, and Ghinea 2018).

In the hybrid approach, the user interface and business logic code, written in regular web technology programming languages, is executed and rendered by basically an embeddable browser, a component named *WebView*. The component with the code are wrapped into an app with a so-called *native shell*.

Besides the hybrid approach enabling what is practically a web app to be used as a native app, the approach also enables the app to use native code through the *WebView* component. The communication between the *WebView* component and the native code is called *bridging*. With access to native code, the app can do anything that a native

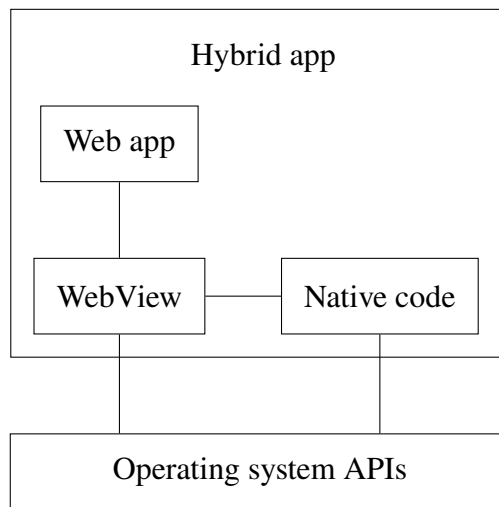


Figure 3: The architecture of a hybrid app (The Apache Software Foundation 2021).

app can; one instance of this is accessing device features. However, accessing device features requires that there is native code which provides the access for the app code inside the WebView component to the device feature. Fortunately, many frameworks already provide this code for most, if not all, device features. In Apache Cordova, which is the underlying framework for many other hybrid frameworks (Biørn-Hansen, Grønli, and Ghinea 2018), the access to device features are provided with so-called *plugins*.

Because the WebView component is used to render the content of the app, the user interface is web-based. This is different from the apps that are developed with the native approach, where the UI is constructed with native UI elements, resulting in a so-called native user interface. In a web-based user interface, the interface is theoretically identical for each platform, whereas with a native user interface, the appearance of the interface varies between mobile platforms.

Because the hybrid approach is basically a web app wrapped inside a native app, it is possible to have a web app and the hybrid app to share most of the app code. This ease of code transferability, however, depends on what frameworks are used in each instance.

2.1.3 Interpreted approach

In the interpreted approach, the app code is interpreted by an *interpreter* and in many frameworks the app code is written in JavaScript (Biørn-Hansen, Grønli, and Ghinea 2018). In iOS the default JavaScript interpreter is JavaScriptCore and in Android V8 is frequently used as the interpreter (Biørn-Hansen, Grønli, and Ghinea 2018).

Although JavaScript is used in both the interpreted and hybrid approaches, the interpreted approach does not use WebView. Instead, the app code is interpreted by an on-device interpreter and native code exposes functionalities to the app code through the interpreter by bridging. For instance, access to device features are provided with

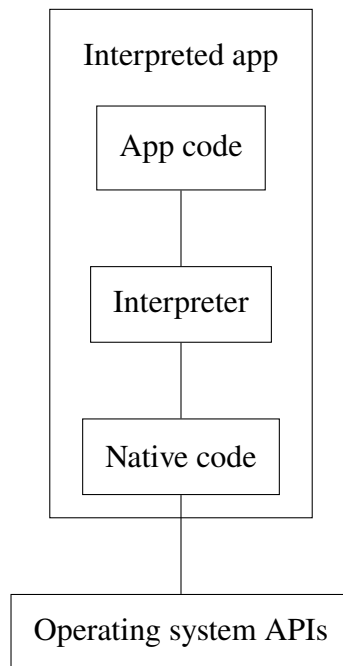


Figure 4: The architecture of an interpreted app (The Apache Software Foundation 2021).

native code that is provided by or extended by implementations into a framework.

The interpreted approach can render the user interface with native UI components. This may be seen as the main difference from the hybrid approach, where the user interface is web-based. Like the hybrid approach, an interpreted framework can enable to use the same user interface code for all supported platforms, in other words from an engineering perspective there is no need to write separate user interface code for each platform.

One downside with the interpreted approach is that the frameworks do not share a common base system, unlike the hybrid approach where Apache Cordova is used in most frameworks (Biørn-Hansen, Grønli, and Ghinea 2018) to provide the base to the framework. This means that device-feature plugins in the interpreted approach frameworks are not directly transferable, i.e., a plugin in one framework cannot be used as such in another.

Because the interpreted approach does not use WebView to render content, the user interface code is different from user interface code used in web and hybrid apps. This most likely means that at least the user interface code needs to be reimplemented for the interpreted approach when using a web or hybrid app code as a template. However, if the business logic code for the interpreted app is written in JavaScript, there may be the possibility to use the same code from the web and hybrid app.

2.1.4 Cross-compiled approach

In the cross-compiled approach, a framework compiles the app code, after which the compiled/object code is generated into a native app for each targeted platform. The resulting apps resemble closely their counterparts developed with the native approach. Unlike in the hybrid and interpreted approaches, the code in this approach is not interpreted. Device features are accessed through the framework's SDK.

Cross-compiled apps can be rendered with native UI components like in the native and interpreted approaches.

2.1.5 Model-driven approach

The model-driven approach is based on the model-driven development (MDD), also known as model-driven software development (MDSD), paradigm. The model-driven approach is commonly mentioned in the articles related to the field, but there are quite a small number of both academic and industry implementations (Biørn-Hansen, Grønli, and Ghinea 2018). One philosophy of the model-driven approach is to enable non-developers and non-technical users to develop the app.

According to Biørn-Hansen, Grønli, and Ghinea (2018), the MD² framework does not require any platform-specific programming language knowledge and that these kinds of abstractions are part of the methodology of MDD development.

The approach allows platform independent modeling of the app. Apps are commonly coded with a domain-specific language (DSL), which may be exclusive to the framework.

2.1.6 Progressive web app approach

Recently, progressive web apps (PWAs) have emerged as a compelling option for a native-app-like experience on mobile devices due to its light-weight setup.

In the PWA approach, supported features are determined differently than in other approaches. In other approaches, the feature support is determined by the underlying OS capabilities, where differences between different devices in the same platform is determined by the installed OS version. However, in PWAs the feature support is more platform specific.

The progressive web app development approach means using techniques that can be used to produce *progressive web apps* (PWAs), also referred to as *progressive apps* (Russell 2021a) or *HTML5 apps* (Firtman 2020). As the term progressive web app implies, it is basically a web app. This means that PWAs are developed using common web technologies, such as HTML, CSS, and JavaScript. The word progressive in PWA, according to Hakulinen (2021), means: “. . . that the user experience is enhanced gradually based on the browser's capabilities”. However, the capabilities that enhance the user experience are not standardized. Broadly speaking, PWAs are about combining the benefits offered by web apps and native apps: reachability offered by web apps and capabilities offered by native apps (Richard and LePage 2020). A more specific definition what a PWA is varies, this is partly due to that there different “qualities”

of PWA. In practice, web browsers define what qualifies as a PWA by providing the option to install them. Whereas apps produced from other approaches are usually installed from the app store, a PWA is installed from a PWA-enabled website. Also, unlike apps produced with other approaches, PWAs are not platform-specific. Instead, because a PWA is essentially a web app, a web browser takes care of executing the app, difference being that the device displays the PWA in a standalone window instead of a web browser tab.

If it is required that the PWA has to function in offline mode, the following technical features need to be present: the app needs to run in a *secure context*, as well as have a *web app manifest* file and a *service worker*.

Secure context means that the web app is served over a connection which is deemed secure. If a resource is non-local, then one of the requirements is that the resource is served over HTTPS or the WebSocket protocol. According to (Mozilla 2020a): “The primary goal of secure contexts is to prevent MITM attackers from accessing powerful APIs that could further compromise the victim of an attack”.

The web app manifest is a JSON formatted file which holds information about the web app. The file contains at least the name, short version of the name, start URL, display mode, and icon location(s) of the app (Russell 2021b). The manifest provides the necessary details to make a shortcut to the app on the mobile device.

Service worker is essentially a script running in the background of the web app. According to Mozilla (2020b): “Service workers essentially act as proxy servers that sit between web apps, the browser, and the network (when available)”. Service worker makes it possible to make the web app to work in offline mode. Another notable feature is that the worker enables the use of push notifications; however, currently this functionality is not supported in iOS.

The progressive web app (PWA) approach is the newest of the approaches. Since 2016 the approach has gained popularity among practitioners. Some time ago iOS increasingly started supporting PWAs.

If a PWA is installed with Chrome, it can be started like any other native app from the app drawer. The static assets are stored on the device and the app can be used in offline mode. Last when the author checked, iOS can delete data related to PWAs after some time.

Despite having a native-like feel, PWAs currently lack some platform and device features, which are available in the other approaches. These capabilities are determined by the platform and device features that are bridged to the WebView component, which are set and implemented by the platforms. The lack of platform and device features is probably to some extent due to that PWAs can be installed on the phone from outside the app store, which means that they don't undergo the same checks as the native apps.

2.2 Framework candidates

Table 1 shows the framework candidates for the hybrid, interpreted, cross-compiled, model-driven, and PWA approaches. Most of the hybrid frameworks are built on top of Cordova¹. Ionic² has two major versions: one built on top of Cordova and another built on top of Capacitor³ (Ionic 2020c).

Table 1: Framework candidates (all frameworks listed are from Biørn-Hansen, Grønli, and Ghinea (2018)).

Hybrid	Interpreted	Cross-compiled	Model-driven	PWA
AppGyver ^a	Adobe AIR	Apportable	Appian	Angular
Capacitor	Fusetools	Codename One	Applause	Ember.js
Cocoon ^a	Jasonette	Corona	Automobile	Glimmer.js
Cordova	Kony	Crosslight	AXIOM	Ionic
EvoThings ^a	LuaView	DragonRAD	MAML	Mithril
Framework7 ^a	MoSync	Flutter	MD ²	Moon.js
Intel App Framework ^a	NativeScript	Marmalade	Mendix	Polymer
Intel XDK ^a	React Native	MonoCross	MobDSL	Preact
Ionic ^{a,b}	Smartface Cloud	MoSync	Mobia Modeler	React.js
Kony ^{a,2}	Tabris.js	RAD Studio (Delphi)	Mobl	Stencil.js
NSB/App-Studio ^a	Titanium Appcelerator	Rhodes	ModML	Svelte
Onsen UI ^a	Weex	RoboVM	mdsl	viperHTML
PhoneGap ^a		Qt Mobile	MOPPET	Vue.js
RhoMobile		Xamarin	WebRatio	Zuix
Sencha Touch ^a		Xojo Mobile	XIS-Mobile	
Trigger.io			Xmob	
Quasar Framework ^a				

^a Built on Cordova.

^b Built on Capacitor.

1. <https://cordova.apache.org/>
2. <https://ionicframework.com/>
3. <https://capacitorjs.com/>

2.3 Selection and evaluation frameworks

This section provides an overview of selection and evaluation frameworks presented in other studies. This thesis focuses on studies which main goal is to provide a framework for selecting or evaluating development approaches or technical frameworks.

There are numerous studies presenting selection and evaluation frameworks for mobile app development approaches or technical frameworks. These selection and evaluation frameworks can consist of one or several selection or evaluation methods. One method is decision trees, which studies by Khachouch et al. (2020) as well as Lachgar and Abdali (2017) present for the selection a development approach. Another method is quantitative comparison. In quantitative comparisons, each metric is assigned with a value to indicate how well an evaluated option meets the metric. Metrics and/or groups of metrics can be weighted. After assigning a value for each metric, a sum or average is calculated. The evaluated options are compared by comparing the sums or averages. The study by Rieger and Majchrzak (2019) quantitatively compares technical frameworks. For each technical framework, several weighted criterion are rated between zero and five, after which an average is calculated for the framework. The framework can be tailored by adjusting the weights of the criteria. Another method is qualitative comparison with quantitative or qualitative metrics. For instance, the study by Ahti, Hyrynsalmi, and Nevalainen (2016) presents an evaluation framework that uses quantitative and qualitative metrics to qualitatively compare technical frameworks. One quantitative metric the evaluation framework includes is app starting time. One qualitative metric the evaluation framework includes is easiness of development.

As mentioned earlier, a selection or evaluation framework can consist of several selection or evaluation methods. The study by Lachgar and Abdali (2017) presents an evaluation framework which consists of one selection and one evaluation method. The first method is a decision tree, which is used to select a development approach. Each node in the decision tree is a question, which can be either answered with yes or no. By answering each traversed node, the leaf of the decision tree will tell which development approach to select. The second method is a quantitative comparison, which is used to evaluate technical frameworks belonging to the selected development approach. For each technical framework, different criteria are rated and summed together, where higher numbers are better.

The selection or evaluation frameworks can differ in other ways besides the type or count of selection or evaluation methods. There are frameworks which have been meant for certain domains. For instance, the framework by Sommer and Krusche (2013) focuses on business apps. Another way selection and evaluation frameworks differentiate is whether they are case dependent. For instance, the framework by Rieger and Majchrzak (2019) can be tailored to a particular case, but the framework can also be used without tailoring for a particular case. In case a selection and evaluation framework is case dependent, the framework can also be implementation dependent. This means that selection or evaluation framework requires an implemented app for the selection or evaluation process to be possible. However, what app is implemented and in what extent the app is implemented are factors that differentiate from one framework

to another. For instance, the framework by Ahti, Hyrynsalmi, and Nevalainen (2016) requires an implemented app for each evaluation option in order to provide the values for the quantitative metrics, such as the app starting time. The applicability of the selection and evaluation frameworks also differ. Some selection and evaluation frameworks are used to select or evaluate development approaches, whereas other selection and evaluation frameworks are used to select or evaluate technical frameworks. Some selection and evaluation frameworks can be for both development approaches and technical frameworks. For instance, the framework by Lachgar and Abdali (2017) can be used for both development approaches and technical frameworks in the manner that was mentioned earlier.

3 Case description

In 2021, Accountor HR Solutions had some time ago started to develop a self-serving web app for employees; more specifically, a web app that provides company employees the means to *view and manage company's human resource and payroll data related to themselves*, such as viewing ones pay stubs. Besides active development during that time, the web app was also under internal testing. The web app was designed from the beginning to be used on a wide variety of devices, however, for mobile devices there was a need to utilize capabilities that are inherent to these devices, such as device notifications, which lead to the need for a mobile-enhanced version of the app. This mobile-enhanced version of the app (henceforth referred to as the *case app*) is the case in this thesis. Depending on the chosen development approach, the mobile-enhanced app would either be the same app as the web app or a separate native mobile app. If the PWA approach would be chosen, then the web app and the mobile-enhanced app would be the same app. If another approach evaluated in this thesis would be chosen, then the mobile-enhanced app would be served as a separate native mobile app.

The web app was developed using Vue.js⁴ and ASP.NET Core⁵. Both Vue.js and ASP.NET Core are technical frameworks meant for developing web apps. Apps on Vue.js are developed using JavaScript, whereas apps on ASP.NET Core are developed using C#. Despite both frameworks are meant for developing web apps, most of the client side of the app, in other words the part of the app that runs on the end-user's browser, was implemented on Vue.js, whereas most of the server side of the app was implemented on ASP.NET Core. Most of the web app had been implemented on Vue.js using TypeScript⁶, a programming language that compiles to JavaScript. One of the major features that TypeScript provides compared to JavaScript is static typing.

The TypeScript in the web app was unit tested with Jest⁷ and the deployed app was tested automatically with tests written on the Robot Framework⁸ and Selenium⁹ toolchain. Robot Framework is a framework used for test automation and robotic process automation (RPA). According to Software Freedom Conservancy (2021), "Selenium is a suite of tools for automating web browsers".

Figure 5 shows the software system that surrounded the case app. Whether the case app would be the same app as the implemented web app or a separate native mobile app, the case app would communicate with an identity provider and an application programming interface (API). The case app would be communicating with the API using the HTTPS protocol. The API was a RESTful API, in other words it followed the representational state transfer (REST) architectural style. An API that follows the REST style has to come with certain features, such as being stateless and indicating whether a response can be cached (Fielding 2000). The identity provider authenticated

4. <https://vuejs.org/>

5. <https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core>

6. <https://www.typescriptlang.org/>

7. <https://jestjs.io/>

8. <https://robotframework.org/>

9. <https://www.selenium.dev/>

and authorized the end-user using the OpenID Connect¹⁰ (OIDC) protocol. OIDC is an authentication and authorization protocol, where the authorization is based on the OAuth¹¹ 2.0 protocol (OpenID 2021). According to (OpenID 2021), “(Identity, Authentication) + OAuth 2.0 = OpenID Connect[.]” OIDC can be used with web apps and native mobile apps (OpenID 2021). When the end-user would log into the app, they would receive an access token from the identity provider. This access token would be sent along any request to the API in order for the API to verify from the identity provider that the end-user would have the right to make the request. If the end-user would be authorized to make the request, the API would forward requests from the app to the back-end. The back-end would handle the requests forwarded from the API gateway.

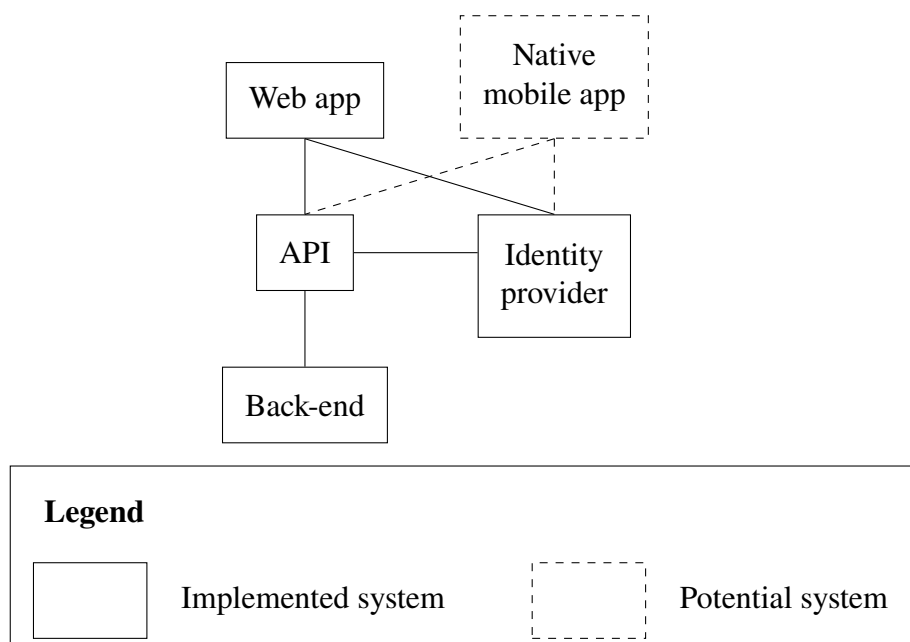


Figure 5: Architecture of the software system.

10. <https://openid.net/connect/>

11. <https://oauth.net/>

4 Methods

Section 4.1 presents and describes the methods used to elicit the requirements found in Chapter 5. Section 4.2 motivates and describes the requirement-based evaluation framework.

4.1 Requirement elicitation

In order to evaluate the development approaches, the evaluation criteria needed to be identified. These criteria were chosen to be the requirements for the case app. The first step for determining the requirements for the case app was to perform a content analysis on documentation regarding the app, which is described in Section 4.1.1. The content analysis yielded functional requirements in form of use cases, which were prioritized in focus groups, which are described in Section 4.1.3. A comprehensive list of requirements also includes non-functional requirements, so these were elicited with interviews, which are described in Section 4.1.4. Section 4.1.2 presents and describes the interview and workshop participants. Section 4.1.5 describes how the final list of requirements were acquired from the focus group and interview data.

4.1.1 Documentation analysis

The content analysis of documentation regarding the case app revealed planned use cases for the app. Use cases are brief, one-sentence descriptions of tasks, such as “View latest payslip information”. Due to the high-level nature of use cases, some of the use cases written in the documentation could be completed in multiple different ways, i.e., they had multiple possible execution paths. For instance, the use case “Call contact persons” could have one execution path where the user copies a contact person’s phone number from the app and then pastes it to the device’s dialing app from which the person makes the call. Another execution path for this use case could be that the user clicks on the contact persons’ phone number in the app, which directly starts the call to the person. Use cases which were identified with multiple execution paths were split up into multiple new use cases, so that each new use case represented only one execution path. Each use case associated with only one execution path was linked with requirements required by the use case and that possibly could only be met by part of the frameworks selected for the development approach comparison. Use cases that were linked with these kinds of requirements were selected to be prioritized in focus groups. Section 4.1.3 describes the focus groups.

4.1.2 Focus group and interview participants

Five persons from the company participated in focus group sessions and interviews: an architect, the director of product development, a lead designer, a product owner, and a sales director. Rest of this section provides information about each person.

The architect was one of the architects to one of the company’s products. Their technical knowledge in the product would be applicable to the surrounding systems of

the case app, especially regarding the backend that would be used for the case app.

The director of product development would in their role be able to provide knowledge related to the resources allocated for the case app.

The lead designer had around the time of the interviews and focus groups been working about two years for the company. The person was hired as a lead designer who would be responsible for all the new products' user experiences and interfaces, including the case app. As the lead designer for the case app, they would be able to provide knowledge related to the planned user experience of the case app.

The product owner was the product owner for the case app and another company product. As the product owner for the case app, they would be able to provide knowledge related to the current and planned features as well as the properties of the case app.

The sales director had been working around the time of the interviews and focus groups ten years for the company in a variety of positions. As the sales director, they would be able to provide knowledge related to commercial expectations of the case app.

4.1.3 Focus groups

The resulting seventeen use cases from the content analysis were prioritized in two focus group sessions. A time slot of 1,5 hours was reserved for each session. The sessions were held within a week's time-span. A majority of the invited person were able to attend each session. Because of the gathering restrictions due to the COVID-19 pandemic being in place at the time, the focus group sessions were held as audio calls. Prior to the first session, the participants received material which contained instructions for both sessions and the use cases to be prioritized in these sessions. During the focus group sessions, instructions and the use cases were presented to the participants by the thesis author both orally and visually.

In the focus group sessions, the participants were asked together to prioritize each use case with a number from 1 to 5:

- 1: the use case is unnecessary
- 2: the use case is somewhat useful
- 3: the use case is quite useful
- 4: the use case is very useful
- 5: the use case is mandatory

If the participants could not agree on a number, the number with the most votes was marked for the use case. After a priority was determined for a use case, the priority was recorded by the thesis author.

4.1.4 Interviews

The aim of the interviews was to elicit requirements, particularly non-functional requirements. These elicited requirements would complement the functional requirements identified in the documentation analysis. The format of the interviews was decided to be semi-structured.

The interview questions prepared for each interviewee are presented in Appendices A–E. Six to twelve questions were prepared for each interviewee. Most of the interview questions were tailored for each interviewee, whereas the other questions were prepared for several interviewees. The tailored questions were tailored based on the interviewee’s role in the company, whereas the shared questions could be answered by the interviewees to whom the questions were prepared. In addition to the interview questions some follow-up questions were also prepared. As the final question in all the interviews, the interviewees were asked if they had anything to add to the interviews’ research question, which was “What are the requirements for the case app”?

Each interviewee received their interview questions along with instructions prior to their interview. The interviewees were asked to prepare for the interview by examining the interview questions. One hour time slot was booked for each interview. Some interviews exceeded this time slot, however, consent from the interviewee was asked before exceeding the time slot. Table 2 shows the duration of the recording for each interview. These durations do not include pre- and post-interview activities, such as informing the interviewee why the interview is conducted. All the interviews were conducted during a one-week period. During one interview, the interviewer provided material to help the interviewee to answer an interview question. The interviewees were asked to contact the thesis author in case there were some information related to the research question that they recalled after their interview. One interviewee sent material, which they figured could help the author to answer the interviews’ research question. Some interviewees were asked to clarify statements they made during the interview after the interview had taken place.

Table 2: Duration of the recording for each interview.

Interviewee	Duration (minutes)
Architect	72
Development director	59
Lead designer	91
Product owner	60
Sales director	44

Due to the COVID-19 pandemic, the interviews were held as video calls, which were recorded by the interviewer. From the video recordings, the audio was extracted. Besides recording the interviews, the interviewer took notes during the interviews.

Next section describes how the audio recordings and the interviewer's notes were analyzed.

4.1.5 Focus group and interview data analysis

This section describes how requirements used as comparison criteria were gained from the focus group and interview data.

The data gained from focus group sessions was a list of prioritized use cases. The requirements that were associated with at least one use case that gained the highest priority in a focus group session would be selected as comparison criteria.

The interview data was analyzed by identifying requirements from the interview audio recordings and the notes taken during the interviews. More specifically, the audio recordings were analyzed by taking notes and by partly transcribing the recordings.

Any requirement identified from the recordings or the interview notes needed to meet three conditions in order to be selected as a comparison criterion. The first condition was that the interviewee had expressed that the requirement was mandatory. The condition would also be met if the priority for the requirement could not be determined. For instance, there was a possible requirement that the interviewee answered was probably mandatory. However, because the requirement was not certainly required, the requirement was left out of the comparison. The second condition was that the requirement somehow applied to the case app. For instance, if the requirement only applied to the web API, then the condition would not be met. The last condition was that the requirement would possibly only be met by some of the frameworks selected for the development approach comparison. In other words, if the requirement would clearly be met by no framework or all frameworks selected for the development approach comparison, then the condition would not be met.

4.2 Development approach evaluation

Section 2.3 described different types of selection and evaluation frameworks found in academic literature. Despite several studies presenting selection and evaluation frameworks, all of these frameworks are based on a defined set of specific metrics or criteria, such as supported platforms or hardware features. A criteria-based framework may be useful during the requirement engineering phase, when there is a need to evaluate development approaches or technical frameworks to determine what requirements to include in the requirement specification. However, if the most suitable development approach or technical framework is sought for a planned app, then a criteria-based evaluation framework is poorly suited, because of a limited set of criteria cannot take into account all the requirements set for an app. In these cases an evaluation framework based on concrete requirements is better suited. A requirement-based evaluation also ensures the validity of the results gained from the evaluation, whereas a criteria-based evaluation may for instance indicate that a particular technical framework is the most suitable for an app, but when closer inspecting the framework against the app it is found out that a mandatory requirement cannot be met, thus making the technical framework unsuitable for developing the

app. Because the thesis strives to rank the development approaches for an app that has been already defined, the thesis will not use any of the evaluation frameworks presented in the other studies, but instead make use of a requirement-based evaluation presented and defined in this chapter.

In order to perform a holistic evaluation with the requirement-based evaluation framework, it is required that the requirements for the app have been elicited. The requirements that do not depend on any development approach should be left out from the evaluation framework. The first step in the framework is to select a technical framework to represent each development approach. For each development approach, the selection is done by selecting the technical framework which can meet most of the app requirements (henceforth referred to as criteria). In case several technical frameworks share this position, then just arbitrary selecting one of these is enough. After the technical frameworks have been selected, they are qualitatively evaluated based on the criteria. To help the qualitative evaluation, the criteria can be assigned into groups, such as hardware features, or one or several sums of met criteria can be calculated for each technical framework. Based on the technical framework evaluation, the final step is to evaluate the development approaches.

5 Requirements

Requirement in Section 5.6.8 affects other requirements in that mobile operating systems are evolving rapidly, thus features that are now supported may have not been possible to implement a year back.

To protect the identity of the interviewees, the interviewees are identified with numbers instead of occupation. This is done because in some cases just the interviewee's occupation is enough to point to which person has provided what statements. The numbers attached of each interviewee are only to differentiate the persons in each requirement sections. The numbers are section specific, meaning that same number can be a different person in a different requirement section.

Table 3 shows an overview of the requirements.

5.1 Hardware features

In this thesis, four significant required hardware features were identified.

5.1.1 Camera

When asked interviewee 1 which platform features in their opinion must be included in the mobile app, the interviewee answered the camera. The interviewee also stated that in their opinion if the camera is not possible, that the case app will fail in the future. When asked about use cases for the camera, the interviewee gave examples of taking pictures of documents and then attaching or sending them. Because camera can be interpreted to mean taking pictures, capturing videos, or both, clarification on which alternative the interviewee meant was asked after the interview. The interviewee answered that in their opinion there is no need for capturing video. Based on these statements by the interviewee, the end-user has to be able to start the picture taking procedure from the app. The taken pictures can then be managed by the app, such as attaching it to a form.

5.1.2 Click to call

The case app has two use cases related to calling. One is calling the emergency contact and the other is calling contact persons. Calling contact persons is listed under the category "Employee's organization", which can be interpreted that the case app has to provide the end-user the ability to call their contact persons within their organization, which may be persons such as their supervisor. Based on the use cases, the user has to be able to call persons using the case app. This thesis uses the term "click to call", used by LePage (2014), for referring to this feature. Based on the use cases, the case app has to support click to call.

Table 3: Requirements overview.

Category	Requirement
Hardware features	Camera Click to call Notifications Passwordless authentication
System requirements	OpenID Connect RESTful HTTPS API
Functional requirements	Appearance configuration Calendar Dynamic UI Local storage
Regulative/Legislative requirements	Accessibility General Data Protection Regulation
Quality requirements	Availability Fault tolerance Maintenance Modularity Performance Portability Reliability & dependability Scalability Security
Other requirements	App store distribution Authentication methods Automatic logout Direct manipulation Integrate other systems Logging Multi-language support Operating system Responsive design Rollback Testing

5.1.3 Notifications

Numerous use cases written for the case app had notifications as an underlying requirement. The use cases deemed most important for the app were selected to be prioritized in the focus group sessions, in which all of these use cases were identified, thus making notifications a requirement for the case app. In these sessions, the participants were explicitly told to rate the use cases based on the premise that the notification would be native-like, i.e., other notifications such as email notifications wouldn't apply here. There are two types of device notifications: local and push notifications. Local notifications are ones coming from the app on the device, whereas push notifications are "pushed" from an online notification service to the device. Because the focus groups were asked to prioritize only device notifications, and not local and push notifications, the evaluation will be performed on both local and push notifications.

5.1.4 Passwordless authentication

Interviewee 1 stated that fingerprint-based authentication and similar authentication methods in their opinion has to be supported by the app. When asked interviewee 2 if fingerprint-based authentication could be used in the case app, the interviewee answered that it may be acceptable for the case app. However, when asked if fingerprint-based authentication is not enough for bank apps, the interviewee answered yes, which would mean that the interviewee 1's statement that the case app has to be as secure as a bank app (see Section 5.5.9 for further details) would not be met according to interviewee 2 if fingerprint-based authentication would be included. Interviewee 2 stated that for instance the OP bank app does only allow PIN for authentication. However, at least the S-mobiili app on Google Play allows authenticating and paying with fingerprint-based authentication instead of using online bank credentials (Google 2021), thus indicating that fingerprint-based authentication can be used at least with some online bank services. Based on this, it is assumed that fingerprint-based authentication is secure enough for the case app and thus the case app has to support fingerprint-based authentication.

After the interview, interviewee 1 was asked which authentication methods besides fingerprint-based authentication need to be supported, to which the interviewee answered PIN authentication. The interviewee elaborated that either device PIN or a PIN set for the case app will do. Based on the statements, the thesis will require that the case app will support device PIN authentication. Note that the app-specific PIN should probably be considered as a viable alternative in case device PIN authentication is not possible, however, the alternative will not be considered in the thesis.

5.2 System requirements

5.2.1 OpenID Connect

One of the requirements set by the software system (Chapter 3) is that the app has to communicate with the identity provider. The communication with the identity

provider is accomplished with the OpenID Connect (OIDC) protocol. This means that the app has to support OpenID Connect.

5.2.2 RESTful HTTPS API

A requirement set by the software system (Chapter 3) is that the case app has to be able to communicate with RESTful HTTPS APIs. More specifically, what this means is that the app has to be able to make HTTPS requests and handle the responses. The REST architecture style does not set any specific requirements on the app, however, when combined with HTTP(S) APIs, there are usually requirements set by convention, one of them which are supported HTTP request methods. An article concerning RESTful web services (Microsoft 2021b) mentions the following HTTP request methods:

- GET
- POST
- PUT
- PATCH
- DELETE.

Based on this, the app has to support the listed request methods.

5.3 Functional requirements

5.3.1 Appearance configuration

Interviewee 1 stated that the case app has to have some kind of modifiability and they seemed to indicate that at least the appearance has to be modifiable. Interviewee 2 stated that the app has to support customer branding to a pretty large degree. The documentation states that product branding is essential for the product and that branding is usually achieved with logos, fonts, and colors. More specifically regarding color, the color palette has to be configurable. Also, more specifically regarding fonts, based on the documentation, the font family has to be configurable. In the use cases, one use case mentions end-user specific UI preferences where dark and light mode has been either exemplified or specified. Because no other UI preferences in the use case has been mentioned, this thesis will only require that the case app supports end-user specific dark and light mode. It is also assumed that the preference can persist over several sessions in order to ensure an acceptable user experience regarding the feature. Also, in the implemented web app; logos, colors, and fonts can be changed in order to achieve the branding that the customer requires. Based on the statements, use case, documentation, and the web app; images, the color palette, and font families have to be configurable for the case app. Because the images can be configured in several different ways, this thesis assumes that the minimum requirement is that the source, of the images can be configured. It is also assumed that changing the appearance does

not require the app to be redeployed, otherwise each appearance configuration would essentially be a separate app for instance on the app stores.

5.3.2 Calendar

Under the category “Receive notification”, the use case “Receive and save calendar events” was listed. After the interview, when asking interviewee 1 clarification about calendar event saving to a calendar app on the mobile device, the interviewee answered that this is probably a mandatory requirement, if the feature can also be implemented in the web apps. Regardless whether the web app condition can be met, the feature is not stated as certainly mandatory and thus when only regarding the interviewee’s case will not be evaluated. However, from my understanding this is also applicable in the case of the use case, as my understanding is that the interviewee is responsible for the use case, and thus the use case of saving calendar events will not be evaluated. Based on the use case mentioned earlier, receive calendar events through notifications is left as a requirement for the mobile app. However, the thesis assumes that receiving calendar events as notifications is no different to receiving other types of notifications, thus if the approach fulfills the notification requirement stated in Section 5.1.3, then the calendar requirement is also fulfilled.

5.3.3 Dynamic UI

One use case for the case app states: “Show and hide sensitive data with step-up authentication”, which has been listed under “Application settings” and “End-user preferences”. This use case most likely means that the end-user has to be able to show and hide sensitive data in the case app.

Interviewee 1 seemed to state that we have to build settings that for instance hides or blurs data, so that the data does not accidentally show to other persons. The interviewee seemed to state that even if the end-user is logged in, there is some data that is not necessarily shown directly or which has to be separately chosen to be made visible. The interviewee also seemed to state that settings related to showing and hiding data can be configured by the end-user.

When asked if interviewee 2 had anything to add to the interview’s research question, the interviewee mentioned customization. The interviewee seemed to elaborate this with the possibility of changing the order and toggling the visibility of UI elements. The interviewee stated that this needs to be end-user specific. The interviewee also indicated that these settings have to survive between sessions and that the settings are preferably remembered globally, meaning that for instance the settings are same between the web app and the case app.

Based on the use case and the statements made by the interviewees, the requirement is that the case app is capable of hiding and rearranging elements. The use case and interviewee 1’s statements also seem to indicate that the toggling data visibility can be either persistent or non-persistent. In case the visibility of data needs to persist, then this can be achieved by storing the information in the system backend. The thesis assumes that is possible in case the case app is capable of HTTPS RESTful

API communication, which is described in Section 5.2.2. This requirement should also cover interviewee 2's possible requirement of global storage of UI settings. The persistent local storage requirement lined by interviewee 2 will be examined as a separate requirement in Section 5.3.4. The use case mentioned step-up authentication for displaying sensitive data. This requirement will be handled separately in Section 5.6.2, which handles authentication methods.

5.3.4 Local storage

When interviewee 1 mentioned possible requirements related to aspects of a dynamic UI (Section 5.3.3), the interviewee also mentioned the possibility of saving these settings locally. Based on this, the case app has to be capable of storage data locally.

5.4 Regulative/Legislative requirements

5.4.1 Accessibility

The Finnish law requires that Finnish authorities follow the accessibility law for digital services (Regional State Administrative Agency 2024b). Because the case app had been planned to be used by Finnish authorities, the app has to also comply with this law. According to Regional State Administrative Agency (2024a), these are “. . . 49 criteria for level A and level AA of the international Web Content Accessibility Guidelines 2.1”. In the case app's documentation, the requirement for the case app has been stated to be both A and AA levels of the WCAG 2.1. Interviewee 1 stated that we must take the laws and directives into account when regarding accessibility.

5.4.2 General Data Protection Regulation

Interviewee 1 stated that they would take data protection in consideration continuously when regarding the requirements and technologies. One data protection and privacy regulation that is required by the EU and Finnish law is the General Data Protection Regulation (GDPR) is followed. One requirement related to GDPR that will be evaluated is that the data stays inside the European Economic Area (EEA). Despite GDPR allowing the data to leave the EEA if certain conditions are met, the thesis will take a stricter stance and require that the data stays inside the EEA, because otherwise defining and evaluating the requirement will be considerably more laborious, which makes it unsuitable for the scope of this thesis. Otherwise, if it is assumed that the case app only manages personal data that can also be stored on external systems accessible to the company, then no other requirements related to GDPR will affect the evaluation and thus not added to the evaluation.

5.5 Quality requirements

5.5.1 Reliability and dependability

This section will address both reliability and dependability, due to the fact that the interviews were held in Finnish and that the Finnish word *luotettavuus* used in some interviews can be either interpreted to mean either reliability or dependability. According to Sommerville (2016), reliability and dependability have different meanings in the software industry. Quoting Sommerville (2016, p. 310), reliability is “[t]he probability of failure-free operation over a specified time, in a given environment, for a specific purpose”. Dependability is an umbrella term that contains besides reliability, also availability, safety, security, and resilience (Sommerville 2016).

The documentation for the case app states that the app must always provide accurate and correct data, which can be interpreted to mean that the case app must in this sense be reliable. The author of this thesis would have liked to go through more of case company’s documentation related to reliability, but due to end of author’s employment during the writing of this thesis this was not done. Interviewee 1 stated that accuracy and security/safety make an aspect of dependability, which is in a way the development principle of the app. The interviewee also seemed to state that this aspect has to be considered in a big way. The interviewee stated later that in a way flawlessness and reliability/dependability are the biggest non-functional requirements. After the interview, the interviewee was asked to elaborate what they meant with reliability/dependability, when they stated that this is a mobile app development principle and one of the biggest non-functional requirements. The interviewee answered that reliability/dependability means the app’s and service’s technical reliability/dependability, that they work in all situations correctly and that there is not too many outages or too much interference in the service. Based on this, this thesis interprets that interviewee 1 meant dependability, which they elaborated in this case with reliability and availability. Availability is considered in Section 5.5.2 and thus will not be analyzed here further. Nevertheless, based on the documentation and interviewee 1’s statements, the case app has to be reliable in the sense that the app works correctly in all situations as well as that the data the app presents must be correct and accurate. Assuming that the evaluated frameworks and the native SDKs work correctly, i.e., do not contain defects, then the aspects of reliability required by the documentation and interviewee 1 will not affect the evaluation and thus will be excluded from the evaluation.

5.5.2 Availability

In Section 5.5.1, interviewee 1 stated that the case app has to not have too many outages or too much interference, which this thesis interprets that the app has to have some degree of availability.

When asked about what should maybe be taken into account when making decisions about technologies, interviewee 2 mentioned accessibility when considering the whole platform. The interviewee elaborated that the service has to remain offline as little as possible. The elaboration does not seem to make sense with the word accessibility,

thus it is assumed that the interviewee meant availability instead of accessibility, which are very similar in Finnish (saatavuus and saavutettavuus respectively). When later asked what are the non-functional requirements for the app, the interviewee restated accessibility. Because the interviewee had only once earlier in the interview mentioned accessibility, which was interpreted to mean availability, it is assumed that the interviewee meant availability also in this case. It is difficult to exactly say what the interviewee meant by the word platform in the earlier statement, but if connected with the interviewee's elaboration that the service should remain offline as little as possible, it could be interpreted to mean the whole software system, possibly excluding the app.

Despite interviewee 2 seemingly listing availability as one non-functional requirement for the case app, the interviewee seemed to indicate that this requirement is more related to the surrounding software system. Despite interviewee 1 and possibly interviewee 2 requiring the case app to be available to some degree, the thesis will not examine the requirement, because no concrete metrics were provided which the case app has to fulfill regarding availability. Because of these reasons the requirement will not be included in the comparison.

5.5.3 Fault tolerance

When asked of what should maybe be taken into account when deciding technologies, interviewee 1 stated fault tolerance. The interviewee seemed to elaborate that even if some service goes offline, the app could still function. Interviewee 2 stated that the system should be very fault-tolerant, but did not seem to require it. Interviewee 3 stated that the app should be in a way fault-tolerant, but did not seem to require it. However, the interviewee seemed to also state that fault tolerance is in their opinion one of the most important attributes. Because no interviewee seem to have indicated that the fault tolerance of the case app is an essential non-functional requirement, the requirement will not be used in the evaluation.

5.5.4 Maintenance

Interviewee 1 stated that minimizing maintenance should definitely be prioritized over the case app's access to hardware and software features as well as the case app's performance. When asked what are the requirements regarding technologies used in the mobile app, the interviewee wrote same iOS and Android. This thesis interprets that the interviewee meant that the mobile app implementation has to be same for both Android and iOS. The interviewee had also seemed to have written that there is no need for native access and that if there is a trade-off between maintenance and performance, then absolutely favor maintenance. In the interview and in the written response to the interview questions, interviewee 1 seemed to state that maintenance of the case app has to be minimized. This seems to be supported by the interviewee seemingly stating in the interview that the less technology work or code the company has to maintain, the better. This thesis does not know what the interviewee exactly meant with technology work, but the thesis interprets that this part of work that goes to implementing and maintaining an app. In the written response to the interview questions, the interviewee

stated that ideally the mobile app would be part of web development. In the interview, the interviewee stated same code base, which is interpreted to mean that the case app has to or should use the same code base as the web app. The interviewee also seemed to indicate that mobile-specific technology has to be kept to a minimum, because this increases maintenance and technology work. Based on this, this thesis will evaluate which evaluated alternative will share most of the code and technology together with the implemented web app. However, this thesis will not consider code related to unit and automated testing (Section 5.6.11).

5.5.5 Modularity

Interviewee 1 stated that the mobile app must be modular. The interviewee specified that the app must be made of modules that the app requests for. All of these modules do not need to be available, but instead the app works with the ones that are available to it. The interviewee stated that the data for the modules can be located in different places. The interviewee also stated that the architectural solutions need to be modular.

Based on the statements of interviewee 1, the app has to have on-demand and location-agnostic modules. Despite the interviewee listing modularity in their opinion as the most important requirement and also stating that the requirement also applies to the case app, the interviewee's statements indicate that the requirement has more significant implications in the surrounding software systems than in the case app. Based on this, the minimum requirement related to modularity is that the app can handle the situation where the surrounding software system is unreachable or returns an exception. This kind of exception handling should be required from communication that is directed outside the app, which would be the CIAM and APIs. This thesis makes the assumption that if the approach supports OpenID Connect (see sections 5.2.1 and 6.4.1), the approach also supports this requirement regarding the CIAM, thus this requirement will only regard APIs. When considering more specifically exception handling for API communications and the case app, the approach is expected to handle cases where the response of the API:

- takes a significant time or never returns
- is abnormal.

In the case of the first item, the app is expected to not freeze, but remain usable despite not gotten the response. In the case of the abnormal response, the app is expected to be able to react to it differently than in the case that the response would be normal.

Interviewee 2 mentioned in the interviewee modular renewability related to the architecture. This modular renewability comes from the interviewee's thinking that even when technologies change, the mobile app should be adaptable to these changes. This is interpreted to mean that parts of the mobile app can be rewritten at a time when changes in technologies require that the app is adapted. This is probably to make the technological transition/evolution of the mobile app easier.

5.5.6 Performance

Interviewee 1 stated that the performance of the case app has to be on a level that it is nice to use and that there is not too much stalling in the app. When asked what maybe should be taken into when deciding technologies, interviewee 2 answered performance. The interviewee seemed to state that in their opinion most of the pages should definitely load under 200 milliseconds and that this would in their opinion be a good target. When later asked what are the non-functional requirements for the mobile app, the interviewee seemed to state that they already in the interview listed performance. Based on the statements made by interviewee 1, the case app is required to perform on a level that does not too much negatively affect the user experience. Because the interviewee didn't state a more concrete performance characteristics, the thesis will not examine this requirement. Because interviewee 2 did not seem to require the performance target that they stated in the interview, this thesis will not evaluate whether the case app would be able to meet the performance target.

5.5.7 Portability

When asked what are the architecturally significant requirements for the mobile app, interviewee 1 stated that it has to be platform independent. For the same question the interviewee also stated the word portable, which can be interpreted to mean that the mobile app has to be portable. The interviewee seemed to also state that the platform independence and portability are more related to the back-end system, which is not related to the app. When later asked what are the non-functional requirements for the mobile app, the interviewee seemed to state that they already in the interview listed portability. Because interviewee 1 seemed to state that platform independence and portability relate more to the back-end system, the requirement is deemed non-essential for the case app and thus left out from the evaluation.

5.5.8 Scalability

When asked what are the architecturally significant requirements for the mobile app, interviewee 1 answered that it has to be scalable. The interviewee seemed to also state that this requirement is more related to the back-end system, which is not related to the app. When later asked what are the non-functional requirements for the mobile app, the interviewee seemed to state that they already in the interview listed scalability. Because interviewee 1 seemed to state that scalability relates more to the back-end system, the requirement is deemed non-essential for the case app and thus left out from the evaluation.

5.5.9 Security

This section inspects documentation and interview statements that generally refers to the security of the case app. Documentation and interview statements related to specific security features, such as passwordless authentication, will be analyzed in

their own sections. There are also sections that can be associated with security, such as logging (Section 5.6.6) and rollback (Section 5.6.10).

The documentation mentions security as one of the key factors for the case app and elaborates that because of the nature of the data, it “is important to keep safe and secure all the time[.]” Interviewee 1 seemed to state that they see that the case app has the same information security requirements as an online bank application. Interviewee 2 seemed to indicate that security must be considered in a big way, because of the data that is managed. The interviewee also seemed to state that attention should be paid to that the system is secure.

The documentation and the interviewees statements seem to indicate that security is an important aspect of the case app. To define a more concrete level of security expected of the case app, interviewee 1’s statement that the case app has the same information security requirements as an online bank application is considered as the security requirement. However, to the best of the author’s knowledge, there is no Finnish or international standard on online bank application security, so this thesis will not evaluate the frameworks and development approaches based on interviewee 1’s security requirement. Thus, this thesis will not evaluate security in general.

5.6 Other requirements

5.6.1 App store distribution

In the interview, interviewee 1 seemed to state that if the case app is a native app, then it has to be distributed from platforms’ app stores. Based on this statement and on the operating systems that the app has to support (Section 5.6.8), the frameworks and the native approach have to produce apps that can be distributed from Google Play¹² and Apple’s App Store¹³. However, because interviewee 1 also seemed to indicate in the interview that app store distribution only applies to native apps, the app store distribution of PWAs will not be evaluated.

5.6.2 Authentication methods

This section does not cover passwordless authentication, these authentication methods are covered in Section 5.1.4.

One use case states: “Show and hide sensitive data with step-up authentication”, which means that the case app has to support step-up authentication. In the interview, interviewee 1 mentioned multifactor authentication (MFA), but did not state whether this is a requirement. In the interview, interviewee 2 seemed to state that there was one or several discussions before the interview that for instance changing the bank account number would not be possible without strong authentication. With this the interviewee possibly referred to a report, which seems to state that the implemented web app should require re-authentication, step-up authentication, adaptive authentication, or multi-factor authentication when performing sensitive operations, such as changing the bank

12. <https://developer.android.com/distribute/google-play>

13. <https://www.apple.com/app-store/>

account number. It is assumed that this is also valid for the case app. According to Terry (2025), “[a]daptive authentication, also called risk-based authentication, is a context-aware security approach that continuously evaluates authentication attempts and adjusts security measures dynamically based on real-time risk signals. Contextual factors—such as location, device, or time—related to a user’s login or access request feed into adaptive authentication as part of the continuous risk assessment and evaluation process.” Whether or not these authentication methods are required, my understanding is that taking these methods into use may require mostly, if not exclusively, changes in the identity provider, the API, or both (Chapter 3). Thus, this thesis will not evaluate these authentication methods.

Another use case related to authentication methods is “Select preferred authentication methods and situations”, which most likely means that the end-user and/or the client company has to be able to select the preferred authentication methods and which also possibly means that the entity has to be able to select situations which require authentication. This use case does not probably require anything particular from the case app other than that the app supports OpenID Connect (Section 5.2.1), RESTful HTTPS APIs (Section 5.2.2), or both, so that the app can communicate with the identity provider, the API, or both (Chapter 3) about the preferred authentication methods and which situations require authentication.

5.6.3 Automatic logout

When asked of interviewee 1 whether after unlocking the phone and before getting to look at the app the end-user has to authenticate with their fingerprint or PIN, the interviewee answered yes. After the interview, the interviewee was asked for additional information regarding in which situations the end-user needs to log into the case app, to which the interviewee answered after the app has been shut down and is started again. When asked when the interviewee considers the mobile app is shut down, the interviewee answered when the device is locked, shut down, started, or restarted. The interviewee also stated that if this feature impacts the usability too much, then another alternative is that login is required when sensitive data is being modified, such as a bank account. Assuming that the authentication requirement does not impact the case app’s usability too much, the case app has to require authentication after the device has been locked, shut down, or restarted. Based on the statements, it is also assumed that authentication is required if the app is started from a shut-down state. Based on this, the approach is assumed to be able to fulfill the requirement if the app can detect and react to resuming from the states described above.

5.6.4 Direct manipulation

In the interview, interviewee 1 mentioned direct manipulation. They elaborated that you can initiate the modification of data from the place that it is shown. The interviewee was asked after the interview whether direct manipulation is a requirement for the case app, to which the interviewee answered yes.

Because there are numerous ways in which direct manipulation can be implemented,

the thesis will not include this requirement in the evaluation. The more specific case of initiating data alternation mentioned by interviewee 1 will not either be included in the evaluation, as it also can be implemented in numerous ways.

5.6.5 Integrate other systems

One of the use cases that is defined for the case app is to integrate other systems. This means that software systems, such as Slack¹⁴, can be connected to the case app. An example could be that users from within the case app could use Slack to some extent; for instance to send messages using Slack.

The integration of other systems, such as Slack, could be considered to be achieved with (1) communicating with the other system through API(s) provided by the vendor or (2) opening a web browser inside the mobile app through which the web app of the other system can be used. Based on the clarification gotten from interviewee 1, the author got the feeling that the first option would be the one that would be required of the mobile app. Based on these statements, this requirement is fulfilled in case the approach is capable of using RESTful HTTPS APIs. The requirement is described in section 5.2.2 and the results are shown in section 6.4.2.

5.6.6 Logging

When asked if interviewee 1 had anything to add to the interview's research question, which the interviewer seemingly elaborated to the interviewee to be what are the requirements for the mobile app, the interviewee mentioned logging. The interviewee seemed to elaborate that the purpose of logging is to know what has been done. This can be interpreted that the case app has to be able to log actions taken by itself and by the end-user in the app. This requirement is deemed to be fulfilled in case the case app supports local storage and RESTful HTTPS APIs, which are described in Section 5.3.4 and Section 5.2.2 respectively. If the assumption is made that logging only occurs when the app can reach online services, then only the latter feature is required.

5.6.7 Multi-language support

Four use cases together seem to state that text in the case app has to be possible to be displayed in English, Finnish, Swedish, and other languages. One use case seems to state that the device's region and language has to be applied to the case app. Another use case seems to state that the end-user has to be able to choose a language for the case app that they prefer. The use case also seems to indicate that this choice should be persistent, in other words it should survive over one session. Based on these use cases, several language related requirements can be deduced. The first requirement is that the case app has to have the ability to display text in different languages. However, this thesis will not consider right-to-left languages. The second requirement is that the default language for the case app has to be the one that has been set for the device or browser, if the language is supported by the case app. Finally, the third requirement

14. <https://slack.com/>

is that the end-user has to be able to change the language for the case app. For the persistence for this setting, this thesis assumes the case app supports this in case local storage (Section 5.3.4) or RESTful HTTPS APIs (Section 5.2.2) is supported.

5.6.8 Operating systems

In this case arguably the most influential requirement when evaluating the frameworks is which operating systems the frameworks support. Not meeting the requirement means that the app cannot—at least by normal means—run on the operating system. Another aspect to consider besides the operating systems itself is which versions of the operating systems the frameworks have to support. This affects the range of devices that the app is able to run on.

This thesis will only analyze whether the case app is able to run on a particular operating system or its version and will not consider whether Accountor HR Solutions has to support the case app for a particular operating system, its version, or a particular device. Based on this, any statement of company supporting the case app for an operating system, its version, or a device will be directly interpreted, meaning without stating this separately in the thesis, as a statement that the case app works on an operating system, its version, or a device.

Interviewee 1 stated that the app has to be able to run on the Android and Apple operating systems. Strictly speaking there is no operating system named Apple, nevertheless, it can be assumed that the interviewee meant an operating system developed by Apple. Because the interviewee had stated earlier in the interview that the app has to run on smartphones, it can be deduced that this operating system would be iOS. Interviewee 2 stated that the app has to run on Android and iPhone devices, meaning that the app has to run on Android and iOS. Interviewee 3 mentioned Android and iOS when asked about technology requirements, so it is assumed that the interviewee meant that the app has to run on Android and iOS. In addition, when asked of interviewee 4 on what kind of devices the app is expected to be run on, the interviewee answered smartphones and tablets. The interviewee also indicated that the Android platform and Apple devices are the only ones worth considering. Making the conclusion on the basis of these statements by the interviewee that the app has to run on tablets as well as making the assumption from the statements that Apple devices are to be supported, then the app has to also run on iPadOS. Despite Apple splitting iOS into smartphone-specific iOS and iPadOS for tablets, mobile app development documentation, expect Apple's own documentation, seems to use iOS as an umbrella term for smartphone-specific iOS and iPadOS. This thesis will also use this convention and thus will from this point forward use iOS to indicate both iOS and iPadOS. Based on this, the case app has to run on Android and iOS, so that the statements related to which operating systems the case app has to be able to run on will be met for all the interviewees.

Besides which operating systems the app has to be able to run on, another criterion will be the operating systems' versions that the app has to be able to run on. Based on interviewee 1's statements, the interviewee required that the app has to run on up to five-year-old devices with the operating system versions that were installed on these

devices in the factory. Interviewee 4 stated that the app has to run on up to 4–5-year-old devices, but did not state whether the app has to run on the factory-installed operating system version or is it enough that the app runs on the latest operating system version that has been provided for the devices by the manufacturer. Based on these statements, the app has to be able to run on up to five-year-old version of the operating systems, so that all the statements related to operating system versions will be met. However, it has to be noted that Android devices can be released with an older version of the operating system than what is available when the device is released. To take this into account, the minimum operating system version requirement for Android will be set to the newest operating system version that was available five years ago and then lower the major version number by one. Based on this and considering only major operating system versions, the app has to be able to run on Android 5 and later as well as iOS 9 and later. This is based on that the decision of what OS versions to inspect in this thesis was to the author’s recollection made in 2021. In addition, interviewee 2 indicated after the interview that the app should be able to run on devices that are currently supported by the devices’ manufacturer. Assuming that these devices are running with an operating system version supported by the operating system vendor, then in this case the app has to be able to run on Android 8.1 (Android Open Source Project 2021b) and later as well as iOS 12.5 (Apple Inc. 2021a) and later. However, because the minimum operating system versions in this case are higher than the ones set by the 5-year-old operating system requirement, the operating system version requirement will remain at Android 5 and later as well as iOS 9 and later in order to meet the statements stated here for all the interviewees.

5.6.9 Responsive design

Interviewee 1 seemed to indicate that when the screen size increases, the amount of information or the size of user interface components has to increase. The interviewee seemed to state that responsive design is exactly the technology that enables this kind of scaling, changing the layout between portrait and landscape modes, as well as toggling data element presence according to the screen size. Responsive design, also known as responsive web design (RWD), is a design approach to design websites and apps which respond to attributes exposed by the environment, such as the screen size of the device. Although responsive design is strongly associated with websites and web apps, the design approach can also be applied to other types of apps, such as native mobile apps. Interviewee 1 possibly indicated that the components in the UI can change order. This thesis assumes that this is a requirement and takes this into consideration in the comparison.

Responsive design can be considered to consist of three techniques: flexible grids, responsive images, and media queries. A flexible (or fluid) grid adapts to its surroundings. One such adaptation could be that the grid’s overall width follows the width of the browser window. A responsive (or fluid) image scales in order to stay within its assigned space. A media query is used to query an environment attribute, such as the screen width. Based on the result from the query, the website or app can then be adapted.

This thesis assumes that the user interface in the case app has to adapt similarly than in the implemented web app. Based on this and on the statements by interviewee 1, all the responsive design techniques have to be supported. However, based on the same assumption and interviewee, only a subset of media queries defined in a media query specification by World Wide Web Consortium (2021) has to be supported, which are viewport height, width, and orientation querying. Note that in the specification these queries are called height, width, and orientation. A viewport is the area through which the content of the website or app is viewed. Interviewee 1 referred to that the user interface responsiveness is based on the screen size, however, this thesis will instead use the viewport height and width, because then the user interface will arguably also respond correctly to scenarios where the app does not encompass the whole screen. In this thesis, flexible grids will be considered supported when grids are capable of scaling their cells proportionally in width. In this thesis, responsive images will be considered supported when images are capable of scaling themselves in order to stay within its assigned space. It is also required that the images can be set to have a max size. Because the media queries in the specification by World Wide Web Consortium (2021) are meant for websites and web apps, this thesis will consider a query to be supported if the basic functionality of the query is met. For instance, if a query can be used to discern different levels of viewport heights, then the query is seen to be sufficient to serve a replacement for the viewport height query defined in the specification. In order for the viewport height and width queries to be useful, it is assumed that the results from these queries have to correlate to physical length. For instance, density-independent pixel (dp) count correlates to physical length, whereas physical pixel count does not suffice as it does not necessarily correlate to physical length. If the viewport orientation can be determined by other means, such as through calculation using viewport height and width, it will also be an acceptable solution. There may be other media queries which are required, but because they have not been explicitly stated or used in the implemented web app, they will not be evaluated.

Another feature which can be related to responsive design and which was required by interviewee 1 is toggling the presence of data elements, or more generally, user interface (UI) elements. This feature can also be used to seemingly adjust the location of elements, which is required by the case app because of the assumption that the user interface of the app has to adapt similarly than the implemented web app. This thesis stated earlier that the interviewee possibly required that UI components can be arranged in different order. These features are presented in the dynamic UI requirement (Section 5.3.3) and thus the results of these features will not be shown under this requirement.

5.6.10 Rollback

Interviewee 1 seemed to state that the publishing process has to provide the possibility to perform rollbacks. The interviewee also seemed to state that Accountor HR Solutions has to be able to roll back its every service. Based on these statements, the case app has to be possible to roll back. This means that older versions of the case app has to be possible to be redeployed; for instance by redeploying the deployable artifacts or by

restoring a backup of the app.

5.6.11 Testing

In the interview, interviewee 1 seemed to state that Accountor HR Solutions has to write automated tests and unit tests. The interviewee also seemed to state that there is no other alternative than automated testing. These are interpreted to mean that the case app has to support unit testing and automated testing. In Accountor HR Solutions, automated tests refer to tests that are intended to replace testing that is done by persons. These tests are end-to-end user interface (UI) tests. When asked what are the requirements regarding the testing of the mobile app, the interviewee seemed to indicate both in the interview and in the written response to the interview questions that testing has to have a low learning threshold and has to provide the possibility to advance step-by-step. This is interpreted that a testing solution has to have a low learning threshold and that the testing solution has to provide the possibility to expand testing step-by-step. However, this thesis will not inspect automated end-to-end UI testing, because to the author's knowledge there is no information on whether the testing has to be performed on physical and/or simulated devices. However, this thesis will not evaluate whether the testing solutions have a low learning threshold or provide the possibility to expand testing step-by-step. Based on this, the case app has to support unit testing. However, this thesis will only assess whether unit testing is supported in some capacity, not in what capacity unit testing supported.

6 Results

Section 6.1 will present which frameworks were selected to represent the development approaches and describe each of these frameworks. Sections 6.3–6.8 will tell whether the requirements can be met for each framework and the native approach.

6.1 Frameworks

Table 4 shows the frameworks and integrated development environments (IDE) chosen for the development approaches. Vue.js, Ionic, NativeScript, Xamarin, and Mendix are frameworks, whereas Android Studio and Xcode are IDEs.

Table 4: Frameworks and IDEs chosen for the development approaches.

Development approach	Framework
PWA	Vue.js ¹
Hybrid	Ionic ²
Interpreted	NativeScript ³
Cross-compiled	Xamarin ⁴
Model-driven	Mendix ⁵
Native	Android Studio ⁶ Xcode ⁷

¹ <https://vuejs.org/>

² <https://ionicframework.com/>

³ <https://nativescript.org/>

⁴ <https://dotnet.microsoft.com/apps/xamarin>

⁵ <https://www.mendix.com/>

⁶ <https://developer.android.com/studio>

⁷ <https://developer.apple.com/xcode/>

6.1.1 Android Studio

Android Studio is an IDE by Google to natively develop Android apps. Android apps can be developed natively with four programming languages: Kotlin, Groovy, Java, and C++, of which Kotlin is preferred¹⁵.

6.1.2 Xcode

Xcode is an IDE by Apple to natively develop iOS apps. More specifically, this thesis will evaluate Xcode version 13. IOS apps can be developed natively with two

15. https://en.wikipedia.org/wiki/Android_Studio

programming languages: Objective-C and Swift. There are also two ways to develop the UI, with UIKit or SwiftUI.

6.1.3 Vue.js

Vue.js is a framework for developing web apps, including PWAs. This thesis will evaluate same version of Vue.js which has been used in the implemented web app, which is version 2.

On Android, the thesis will only consider PWAs installed through the Google's Chrome browser, which means that they will be installed as a WebAPK. WebAPK is a PWA wrapped as a native app, thus will for the most part behave (LePage 2021) as any other native app on Android; for instance, a WebAPK is uninstalled in the same way as native apps. The PWA inside WebAPK will run on the version of Chrome it was installed with (LePage 2021). This thesis assumes that the devices that are running the case app are capable of installing the latest version of Chrome Android. Chrome 94 was selected as the version to be used in the thesis, as this was the latest version during some time of writing this thesis.

On iOS, the thesis will only consider PWAs installed through the Apple's Safari browser. Based on the results from the operating system requirement (Section 6.8.4 and Stack Exchange Inc (2025), minimum supported Safari on iOS has to be 9.0. However, Stack Exchange Inc (2025) shows this is Safari version is for iOS 9.2.1, but this thesis assumes that the same version of Safari was used with the launching of iOS 9.

For Vue.js it is assumed that a requirement is supported in case the browsers chosen for each operating system support the requirement.

6.1.4 Ionic

Ionic is a framework for developing web and hybrid apps. This thesis will evaluate Ionic for creating hybrid apps. Because there may be differences in how different versions of Ionic work, this thesis will evaluate Ionic version 5. Ionic can be integrated with other web app frameworks, such as Angular, React, or Vue. The combination of Ionic and Vue will be used in the evaluation. In Ionic, native functionality can be supported using Capacitor and Cordova plugins. However, when using Vue on top of Ionic only Capacitor plugins are supported, more specifically plugins made with Capacitor version 2. Thus, only Capacitor version 2 plugins will be considered when evaluating Ionic.

Capacitor is a cross-platform native runtime. Capacitor plugins can be created to access native APIs and run native code. Android plugins can be written with Java or Kotlin whereas iOS plugins can be written with Swift or Objective-C.

For Ionic, it is assumed that a requirement is supported in case the WebView in each operating system supports the requirement. According to Android Developers (2024a), starting from Android 7.0 and Chrome 51, Chrome seemingly provides and renders WebViews. According to Android Developers (2024b), Android 5.0 included a version of Chromium for WebView. The Chromium version could be updated from

Google Play Android Developers (2024b). Based on this, this thesis assumes that WebView on Android is based on Chrome 94, the same Chrome version used for comparing requirements with PWAs. This approach may not be entirely accurate, but this thesis assumes that this will approach will serve the needs for the comparison.

6.1.5 NativeScript

NativeScript is a framework for developing interpreted apps. This thesis will evaluate NativeScript version 7. NativeScript supports accessing the native APIs of Android and iOS. According to NativeScript (2025a), NativeScript apparently supports all native APIs with custom code, which this thesis interprets that every capability available in the native approach can be implemented for NativeScript apps. NativeScript supports using Objective-C and Swift code. NativeScript seems to support using Android source code as well.

6.1.6 Xamarin

During writing this thesis, the support for Xamarin ended and there is the possibility to migrate Xamarin apps to .NET (Microsoft 2025).

Xamarin is a framework for developing cross-compiled apps. This thesis evaluates Xamarin together with Xamarin.Forms, a cross-platform UI library for writing cross-platform UIs. Xamarin support accessing native APIs. Seemingly, Xamarin does not directly support running native code, instead the native code has to be packaged into libraries. Based on this, this thesis assumes that Xamarin apps, with custom code, support every capability available in the native approach.

6.1.7 Mendix

For the model-driven approach, this thesis uses Mendix Mobile App Development Platform¹⁶ as the chosen framework to compare feature support between approaches. The framework was chosen because it was deemed as the most capable commercially available framework of the model-driven approach. The framework can produce either hybrid or native apps. However, the support for hybrid apps has been deprecated and thus this thesis will evaluate the part of Mendix that produce native apps. Mendix can be extended with widgets and JavaScript actions. According to (Mendix Technology BV 2024e), “[w]ith JavaScript actions, you can extend your application’s functionality in ways nanoflows alone cannot. To use a JavaScript action, call it from a nanoflow using the JavaScript Action Call”. Both widgets and JavaScript actions can use native dependencies, which are libraries targeting React Native.

React Native¹⁷ is a framework for building native apps, such as Android and iOS apps. React Native seems to have the capability to access all the native APIs the native approach has, thus this thesis assumes that Mendix apps can have all the same capabilities as the native approach.

16. <https://www.mendix.com/mobile-architecture/>

17. <https://reactnative.dev/>

6.2 Evaluated requirements overview

Table 5 shows an overview of the evaluated requirements.

Table 5: Requirements overview.

Category	Requirement
Hardware features	Camera
	Click to call
	Notifications
	Passwordless authentication
System requirements	OpenID Connect
	RESTful HTTPS API
Functional requirements	Appearance configuration
	Dynamic UI
	Local storage
Regulative/Legislative requirements	Accessibility
	General Data Protection Regulation
Quality requirements	Maintenance
	Modularity
	Performance
Other requirements	App store distribution
	Automatic logout
	Multi-language support
	Operating systems
	Responsive design
	Rollback
	Testing
Total	21

6.3 Hardware features

6.3.1 Camera

Ionic, Mendix, NativeScript, Xamarin, and the native approach support taking pictures in app or from the camera app (Android Open Source Project 2021c; Apple Inc. 2021g; Capacitor 2021a; Ionic 2021; GitHub, Inc. 2021g, 2021h, 2021n; Microsoft 2021j). PWA also supports this on Android, however, for iOS, version 11 or later is required

(Mozilla 2021b).

6.3.2 Click to call

Web apps have support for click to call by having an anchor element (`<a>`) that contains the `href` attribute that contains the phone number that has been prepended with `tel:` (Mozilla 2025a). The anchor element and the `href` attribute are supported by Chrome Android 18 onward and Safari on iOS 1 onward (Mozilla 2025a), which means that the requirement is supported on required browser versions for the PWA. This thesis assumes that this functionality for the web app works in the same way for PWAs and thus the framework supports click to call. The anchor element and the `href` attribute are supported by WebView on seemingly Android 4.4 onward and WebView on iOS 1 onward (Mozilla 2025a), which means that the requirement is supported on required WebView versions for Ionic. Because this and the assumption that Ionic works in similar way as a PWA, this thesis concludes that Ionic also supports click to call. NativeScript apparently supports placing a call from the app with a plugin (GitHub, Inc. 2025b). However, this plugin is apparently quite old and no Android or iOS version is provided. This thesis assumes that if this plugin does not fulfill the requirement, then it is possible to implement a custom plugin that provides the click to call feature for the required operating systems. Mendix supports placing a call from the app (Mendix Technology BV 2024a, 2024b, 2024c, 2024d, 2025b). The native approach supports placing a call from the app (Android Developers 2025c; Apple Inc. 2025c, 2025f). However, according to Apple Inc. (2025c), the support for placing outgoing calls on iOS is only supported from version 10.0 onward, but other documentation seems to apply that iOS versions under 10.0 also support call to click. Thus, this thesis assumes that iOS supports on all required iOS versions.

6.3.3 Notifications

Table 6 shows the supported requirements related to notifications for the frameworks and the native approach. Ionic, Mendix, Xamarin, and the native approach support local and push notifications for both Android and iOS (Capacitor 2021c, 2021d; Mendix Technology BV 2021i, 2021k; Microsoft 2021n). NativeScript supports local notifications (GitHub, Inc. 2021m) and push notifications with Firebase¹⁸ (GitHub, Inc. 2021b). Despite the push notifications plugin being made to work with Firebase, the plugin can most likely be modified to work with other backends. Despite Ionic, Mendix, NativeScript, and Xamarin supporting local and push notifications, the documentation does not specify the Android and iOS version support. In web apps, Notifications API enables local notifications (What Web Can Do Today? 2021), whereas Notifications API and Push API enables push notifications (Google Developers 2019). Chrome on Android supports both Notifications API and Push API, and thus supports local and push notifications (Can I use 2021h, 2021i). Safari on iOS does not support either Notifications API or Push API, and thus lacks support for local and push notifications (Can I use 2021h, 2021i).

18. <https://firebase.google.com/>

Table 6: Supported requirements related to notifications for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Local notifications	✗ ¹	✓ ²	✓ ²	✓ ²	✓ ²	✓
Push notifications ³	✗ ¹	✓ ²	✓ ²	✓ ²	✓ ²	✓

¹ Android only.

² Android and iOS version support not specified. If a required version is not supported, there should be the possibility to enable support with a custom plugin.

³ Deprecated for iOS 9 (Apple Inc. 2021k; Google Developers 2021).

6.3.4 Passwordless authentication

Table 7 shows the supported requirements related to passwordless authentication for the frameworks and the native approach. The thesis will only consider native support for the passwordless authentication methods, which in case for the fingerprint-based authentication was added in Android 6.0 (Android Open Source Project 2021a) and iOS 8 (Hughes 2021). The native approach supports device PIN and fingerprint-based authentication (Android Open Source Project 2021e; Apple Inc. 2021c). Ionic, NativeScript, Xamarin, and Mendix support device PIN and fingerprint-based authentication for iOS (GitHub, Inc. 2021g, 2021j; Microsoft 2021f; NativeScript 2021a; Raj 2021). Ionic supports device PIN and fingerprint-based authentication for Android 6 and later by using a plugin (GitHub, Inc. 2021j). Device PIN authentication for Android 5 can be enabled by implementing a Capacitor plugin (Capacitor 2021b). NativeScript supports fingerprint-based and possibly device PIN authentication for Android by using a plugin (NativeScript 2021a). If device PIN authentication is not supported by the plugin, the authentication method for Android can be enabled by implementing a NativeScript plugin (NativeScript 2021e). Xamarin supports fingerprint-based authentication for Android by using a plugin (Microsoft 2021f; Raj 2021). Device PIN authentication for Android can be enabled by implementing an interface (Microsoft 2021k). Mendix seemingly supports fingerprint-based authentication for Android by using a nanoflow action (GitHub, Inc. 2021g). Mendix should support the device PIN authentication for Android by possibly implementing a bridge to the appropriate native API(s). Web Authentication API (WebAuthn) enables using passwordless authentication in Vue.js. According to a tutorial by Kitamura (2021), Android as well as iOS 14 and later support WebAuthn. However, this thesis could not establish which passwordless authentication methods are available on Android. According to Safari 14 release notes (Apple Inc. 2021e), iOS 14 ships with a Web Authentication platform authenticator, which uses Face ID or Touch ID. This should mean that iOS 14 and

later support device PIN and fingerprint-based authentication for Vue.js.

Table 7: Supported requirements related to passwordless authentication for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Fingerprint-based authentication ¹	✗ ²	✓	✓	✓	✓	✓
Device PIN authentication	✗ ²	✓ ³	✓ ⁴	✓ ⁵	✓ ⁶	✓

¹ Android 6.0 and later.

² iOS 14 and later. Android support unknown.

³ Android 5 support by implementing a plugin.

⁴ If nothing else, Android support for device PIN authentication by implementing a plugin.

⁵ Android support by implementing an interface.

⁶ Android support by implementing a bridge to appropriate native API(s).

6.4 System requirements

6.4.1 OpenID Connect

The implemented web app communicates with the identity provider using OpenID Connect (OIDC) (see Chapter 3 for further details). This thesis assumes that any browser provided by the required operating systems (Section 6.8.4) supports requirements set forth by OIDC and thus Vue.js supports it. Ionic, Mendix, Xamarin, and the native approach also support OIDC (GitHub, Inc. 2021c, 2021d, 2021k, 2021l, 2021o; Mendix Technology BV 2021g). NativeScript should support OIDC by implementing a plugin.

6.4.2 RESTful HTTPS API

Table 8 shows the supported requirements related to RESTful HTTPS APIs for the frameworks and the native approach. All the frameworks and the native approach support communication with RESTful HTTPS APIs (Android Open Source Project 2021d; Apple Inc. 2021l; Hathibelagal 2021; Mendix Technology BV 2021a; Microsoft 2021c, 2021i; Mozilla 2021d; NativeScript 2021b). For Ionic the thesis assumes that the support is the same as for Vue.js.

Table 8: Supported requirements related to RESTful HTTPS APIs for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
HTTPS requests	✓	✓	✓ ^{1,3}	✓	✓ ²	✓
GET	✓	✓	✓	✓	✓	✓
POST	✓	✓	✓	✓	✓	✓
PUT	✓	✓	✓ ³	✓	✓	✓ ⁴
PATCH	✓	✓	✓ ³	✓	✓	✓ ⁴
DELETE	✓	✓	✓ ³	✓	✓	✓ ⁴

¹ Support indicated in examples (NativeScript 2021b).

² Support indicated in an example figure (Mendix Technology BV 2021a).

³ If nothing else, can be supported with a plugin (NativeScript 2021e) in case the native approach supports it.

⁴ Not specifically mentioned in the documentation (Apple Inc. 2021i), but assumed that the method is valid.

6.5 Functional requirements

6.5.1 Appearance configuration

Table 9 shows the supported requirements related to appearance configuration for the frameworks and the native approach. Color or color palette configuration is supported by Vue.js and Ionic (Mozilla 2021c, 2022b; Vue.js 2022), as well as NativeScript (2022c), Xamarin (Microsoft 2022c), Mendix (Mendix Technology BV 2022b, 2022c, 2022d), and the native approach (Android Developers 2022b; Apple Inc. 2022b; Stack Exchange Inc 2022). Image source configuration is supported by Vue.js and Ionic, (Mozilla 2021c, 2022a; Vue.js 2022), as well as NativeScript (2022a, 2022b), Xamarin (Microsoft 2022b), Mendix (Mendix Technology BV 2021c, 2022a), and the native approach (Android Developers 2022a; Apple Inc. 2022c). Typeface configuration is supported by Vue.js and Ionic, (Mozilla 2021c, 2022c; Vue.js 2022), as well as NativeScript (2022a, 2022b), Xamarin (Microsoft 2022a), Mendix (Mendix Technology BV 2022b, 2022c, 2022d; Meta Platforms, Inc. 2022), and the native approach (Android Developers 2021b; Apple Inc. 2022a).

Table 9: Supported requirements related to appearance configuration for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Color configuration	✓	✓	✓	✓	✓	✓
Image source configuration	✓	✓	✓	✓	✓	✓
Typeface configuration	✓	✓	✓	✓	✓	✓

6.5.2 Dynamic UI

Table 10 shows the supported requirements related to dynamic UI for the frameworks and the native approach. Vue.js, Ionic, NativeScript, Mendix, and the native approach support hiding and rearranging of elements (Android Developers 2025f, 2025g; Apple Inc. 2025a, 2025d, 2025e, 2025g, 2025h; Mozilla 2024, 2025b, 2025f, 2025g) For PWA and Ionic support, this thesis assumes that the required CSS properties can be modified or overridden for UI changes to take effect. Additionally, this thesis assumes that `display: none` is used to hide elements, which means that from Android Chrome 65 onward and WebView Android 65 onward can have some abnormal behavior on elements inside `iframes`, when using `display: none` on the `iframe`. Finally, this thesis assumes that because PWA and Ionic support order CSS property, that the frameworks also support ordering elements. For NativeScript and Mendix support, this thesis assumes that the requirement can be fulfilled with, if nothing else, custom bridge implementations.

Table 10: Supported requirements related to dynamic UI for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Hidable elements	✓	✓	✓	—	✓	✓
Rearrangeable elements	✓	✓	✓	—	✓	✓

6.5.3 Local storage

Vue.js, Ionic, NativeScript, Mendix, and the native approach support local storage (Vue.js 2025; Ionic 2025b; NativeScript 2025c; Mendix Technology BV 2025e; Android Developers 2025b; Apple Inc. 2025i). This is shown in Table 10.

Table 12: Supported requirements related to local storage for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Local storage	✓	✓	✓	—	✓	✓

6.6 Regulative/Legislative requirements

6.6.1 Accessibility

There seems to be no information available regarding whether native Android apps can fulfill all the required WCAG 2.1 requirements. Because of the lack of this information regarding one of the development approaches, the thesis will not evaluate this requirement. See Chapter 8.2 for further discussion regarding this subject.

6.6.2 General Data Protection Regulation

All the frameworks and the native approach can meet the requirement that the personal data stays inside the EEA. Vue.js, Ionic, NativeScript, Xamarin, and the native approach can be implemented so that the personal data stays between the app and the API (see chapter 3 for more information), which means that the requirement can be met. Mendix may require that personal data moves through the Mendix Runtime server. According to Mendix Technology BV (2025c), if Mendix Cloud is used, the customer can opt that the servers that are running the app are located in the EU, thus fulfilling the EEA requirement.

6.7 Quality requirements

6.7.1 Maintenance

When researching the development approaches related to maintenance, the requirement crystallized to whether the approach can share its business and UI code between the web and mobile app.

Vue.js fulfills this requirement because of it's also the web app, thus no duplication of features between the web and mobile app. Ionic is also capable entirely sharing the code between the web and mobile app. Mendix is capable to produce both PWAs

and cross-compiled apps. Xamarin does not produce PWAs, but the app logic may be possible to share between Xamarin and ASP.NET. According to VanToll (2018), NativeScript needs separate UI implementations for PWA and native. Assuming the native approach is produced the “native” way, no sharing between this approach and the web app will occur.

Another aspect to take into account is that the PWA approach is the easiest to take into use with the current web app, other development approaches require adaptation or rewriting of the web app to be applicable.

6.7.2 Modularity

One of the modularity requirements is that the mobile app can handle exceptions originating from API communications. The case app can handle this requirement and this thesis assumes that the browsers and their versions required by the PWA are also supported by the case app, thus the thesis concludes that the PWA also fulfills this requirement. Ionic should not be any different due to being a hybrid development approach; the web app inside the hybrid app should by all accounts take care of the requirement. NativeScript can seemingly handle exceptions originating from APIs (NativeScript 2025d). However, this thesis could not determine the Android and iOS version support for this feature. Nevertheless, the thesis will assume that such as basic feature should be supported across the required operating system versions. The native approach can handle exceptions originating from APIs (Apple Inc. 2025k; Android Developers 2024c). However, this thesis could not determine Android version support for the feature, but the thesis will make the assumption that such a basic feature is supported across the required Android versions. Mendix seems to be able to handle API requests asynchronously (Mendix Technology BV 2025a) and handle error responses (Mendix Technology BV 2025h), thus supporting exception handling regarding APIs.

Interviewee 1 required that the mobile app has modular renewability, which I take to mean that parts of the mobile app can be rewritten at a time when changes in technologies require that the app is adapted. The difference between PWAs and the other development approaches is that the deliverable of the PWA is in source code (HTML, CSS, etc.), whereas the other approaches produces a native mobile app that can contain binary, if not being one large binary. Because of the source-code nature of PWAs, it's relatively easy to replace part of the PWA with newer technology; for instance, replacing code that is deprecated with new code. However, native mobile apps can make this harder if not impossible, due to the possible binary nature of the app. Even if this was possible, the vendor could possibly forbid such conduct; for instance, reverse engineering of the app's binary. For this reason I would claim that the PWA approach takes the crown of modular renewability.

Table 14 shows the supported requirements regarding modularity for the frameworks and the native approach.

Table 14: Supported requirements related to modularity for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
API exception handling	✓	✓	✓	—	✓	✓
Modular renewability	✓	✗	✗	—	✗	✗

6.7.3 Performance

According to the study by Biørn-Hansen, Grønli, and Ghinea (2018), most studies focusing on performance seem to reveal that cross-platform development is performance-wise a viable option to native development. However, Biørn-Hansen, Grønli, and Ghinea (2018) also state that no studies focusing on performance was found for the model-driven approach. Nevertheless, the study by Biørn-Hansen et al. (2020) investigates the performance of the model-driven approach using the MD² framework¹⁹, but because the model-driven framework is different from the one that is evaluated here, the conclusion of the Mendix framework’s performance can not be drawn. Because not all approaches do seem to have viable performance research available, this thesis will not evaluate this requirement.

6.8 Other requirements

6.8.1 App store distribution

It is assumed that apps developed with the Android and iOS SDK can be distributed from Google Play and Apple’s App Store, respectively. Documentation for NativeScript, Xamarin, and Mendix instruct how to deploy apps developed with their respective frameworks to Google Play and Apple’s App Store (Mendix Technology BV 2021b; Microsoft 2021a, 2021g, 2021h, 2021i, 2021q; NativeScript 2021f). Documentation for Ionic 8 with Capacitor 7 instructs how to deploy apps to Google Play and Apple’s App Store (Capacitor 2025; Ionic 2025a, 2025c). However, these version of Ionic and Capacitor are newer than the ones inspected in this thesis, but because this is an essential feature for a framework that produces mobile apps, this thesis assumes that older versions of Ionic and Capacitor can be deployed to the app stores. The deployment documentation for all the required frameworks seem to indicate that apps developed with these frameworks can be distributed from the app stores and thus this thesis deemes the app store distribution requirement fulfilled for these frameworks.

19. <https://github.com/wwu-pi/md2-framework>

6.8.2 Automatic logout

Web pages that contain sensitive information usually require that the user confirms they are still present after some time of inactivity on the page. PWAs nowadays provide foreground detection as a possible alternative to inactivity timeout. The foreground detection feature lets the web app to know when it's in the foreground or background. Based on testing of the feature on Bar (2025) on the author's mobile phone, the feature seems to detect when the app's off the screen in these situations:

- viewing different tab on the web browser
- switching to a different mobile app
- locking the device
- going to the home screen of the device, in other words minimizing the web browser.

These scenarios should be enough to fulfill the requirement for PWAs. Both Chrome for Android and Safari for iOS 7 onward support foreground detection (Can I use 2025), thus the PWA approach supports the requirement.

Ionic seems to be capable to detect and respond to when the app returns to the foreground (Ionic 2025d). NativeScript seems also be able to detect and react to return-to-foreground event, however, this is not certain as it's poorly described in NativeScript (2025b). Mendix has the Native App Event Listener widget (Mendix Technology BV 2025d) which claims to be capable of responding to transitions between active, inactive and background states. However, this widget requires Studio Pro 10.12.0 or higher, but either the widget should be possible to adapt or reimplement to Studio Pro 9, which this thesis requires. Thus, it seems like Mendix is able to fulfill the requirement. In the native approach, Android and iOS provide the capability to detect when the app starts or resumes (Android Developers 2025e; Apple Inc. 2025j), which can be used to fulfill the requirement. However, this thesis cannot confirm the minimum supported Android version for this, but assumes that required Android versions are supported.

6.8.3 Multi-language support

Table 15 shows the supported requirements related to multi-language support for the frameworks and the native approach. The implemented web app can display text in different languages, defaults to the language set for the browser, and the end-user can change the language. This thesis assumes that the web app supports the required browsers and their version required for the PWA, thus Vue.js supports these requirements. Because Vue.js supports the requirements, it is assumed that Ionic also supports them. However, this thesis assumes that the default language is set the same as the device's language instead of the browser's language on the device. NativeScript, Xamarin, Mendix, and the native approach support displaying text in different languages, defaulting to the language set for the device, and end-user

changing the language for the app (Agramonte 2021; Mendix Technology BV 2021d, 2021j; Microsoft 2021o; NativeScript 2021c). The native approach supports displaying text in different languages and defaulting to the language set for the device (Android Developers 2021c; Apple Inc. 2016, 2021b). However, Mendix will not apply the language change immediately (Mendix Technology BV 2021d). With the native approach, Android makes it possible to change the language of the app inside the app at runtime (Android Developers 2025d). However, iOS does not seem to encourage this (Apple Inc. 2025b) and instead the language of the app needs to be changed from iOS settings (Hoffman 2019). However, this changing the language of the app without changing the language of the operating system was seemingly made possible starting from iOS 13 (Hoffman 2019). Because changing the language only for the app seems to not be officially encouraged before iOS 13, this thesis assumes that the feature is not supported for iOS 12 and under. This is because there is a possibility that the app may not be accepted to the app store. However, Mendix documentation has a list of supported languages (Mendix Technology BV 2021e), which may mean that an app developed with Mendix can not display text in any language. Nevertheless, Mendix supports the English, Finnish, and Swedish languages (Mendix Technology BV 2021e). However, despite earlier claiming that NativeScript, Xamarin, and Mendix supports all the multi-language related requirements, this thesis makes the same assumption as for the native approach that per-app language change may not be possible below iOS 13, thus changing language for these approaches are marked as unsupported. Ionic gets a exemption to this, because the language is not changed with one or several native APIs.

Table 15: Supported requirements related to multi-language support for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Display text in different languages	✓	✓	✓	✓	✓ ¹	✓
Default to the device or browser language	✓	✓	✓	✓	✓	✓
Change language	✓	✓	✗ ³	✗ ³	✗ ^{2,3}	✗ ³

¹ May not support any language.

² The change is not applied immediately (Mendix Technology BV 2021d).

³ iOS 13 and above seemingly supported (Hoffman 2019).

6.8.4 Operating systems

Table 16 shows the supported requirements related to operating systems for the frameworks and the native approach. In this thesis, operating system versions supporting PWAs will be denoted by add to home screen (A2HS) support, in which case PWAs are supported by Android 5 and later as well as iOS 11.3 and later (Can I use 2021a). NativeScript supports iOS 9 and later (NativeScript 2021h), and seems to support Android 5 and later (NativeScript 2021d, 2021i). Xamarin.Forms apps can be developed for Android 5 and later as well as iOS 9 and later (Microsoft 2021p). Ionic and Mendix also support Android 5 and later (Ionic 2020a; Mendix Technology BV 2021h). However, Ionic only supports iOS 11 and later (Ionic 2020a), and Mendix supports the two latest major iOS versions (Mendix Technology BV 2021h), which seems to indicate that Mendix only supports iOS 14 and later. With the native approach, it seems that Android allows to develop apps that work on Android 5 and later. For iOS, it is assumed that the minimum iOS deployment version in Xcode²⁰ is the lowest iOS version that an iOS app can support, which is iOS 9 and later (Apple Inc. 2021d).

Table 16: Supported requirements related to operating systems for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Android 5–	✓	✓	✓	✓	✓	✓
iOS 9–	✗ ¹	✗ ²	✓	✓	✗ ³	✓

¹ iOS 11.3 and later.

² iOS 11 and later.

³ iOS 14 and later.

6.8.5 Responsive design

Table 17 shows the supported requirements related to responsive design for the frameworks and the native approach.

All the frameworks and the native approach support columns that scale proportionally in width (Android Developers 2020; Apple Inc. 2021i; Can I use 2021e; Fisher 2017; Ionic 2020b; Mendix Technology BV 2021f; Microsoft 2021m; Mozilla 2021a; NativeScript 2022b).

Vue.js supports responsively scaling images in the web app. This thesis assumes that the web app supports the required browsers and their versions that the PWA requires, thus the approach supports it. Because Vue.js supports it, the thesis also assumes that Ionic supports it. NativeScript, Xamarin, and the native approach support

20. <https://developer.apple.com/xcode/>

responsively scaling images (Android Developers 2022a; Apple Inc. 2022c; Microsoft 2021e; NativeScript 2022b). Vue.js and Ionic as well as native Android support assigning max height and width to images (Android Developers 2022a; Can I use 2021f, 2021g; Mozilla 2025c, 2025d, 2025e). NativeScript, Xamarin, and native iOS support assigning max height and width to an image programmatically (Microsoft 2021d; OpenJS Foundation 2021; Stack Exchange Inc 2021c). Mendix support for responsively scaling images as well as assigning max height and width to images could not be determined from the documentation. However, if these features are not supported, a custom widget for images should enable these features.

Vue.js and Ionic as well as NativeScript, Xamarin, and the native approach support querying viewport width and height (Android Developers 2021a; Apple Inc. 2021j; Can I use 2021b, 2021d; NativeScript 2021g; Stevens 2021). Vue.js and Ionic as well as native iOS support querying the viewport orientation (Apple Inc. 2021h; Can I use 2021c). Xamarin and native Android support querying the viewport orientation programmatically (Android Developers 2021a; Stevens 2021). NativeScript supports orientation querying, but the documentation does not specify whether this is device or viewport orientation (NativeScript 2021g). This thesis assumes that Mendix supports viewport width, height, and orientation queries, because the native approach supports them.

Table 17: Supported requirements related to responsive design for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Flexible grids	✓	✓	✓	✓	✓	✓
Responsively scaling images	✓	✓	✓	✓	✓ ³	✓
Image max size	✓	✓	✓	✓	✓ ³	✓
Viewport height	✓	✓	✓ ²	✓	✓	✓
Viewport width	✓	✓	✓ ²	✓	✓	✓
Viewport orientation	✓	✓	?	✓	✓	✓

¹ Possible with custom widgets.

² Not supported with Angular.

³ Custom widget.

6.8.6 Rollback

Building a web app developed with Vue.js produces a collection of artifacts, which can be for instance HTML, JavaScript, and CSS files. The web app can be deployed by transferring these artifacts to a server. A rollback of the web app can be performed

by removing the current artifacts from the server and adding the artifacts of the app version which the app is rolled back to.

For native apps, because neither Google Play nor App Store seem to provide the possibility to roll back the app, rollback seems to be possible by redeploying an earlier version of the app to the stores (Stack Exchange Inc 2021a, 2021b). However, at least iOS apps seem to require that the build string is incremented before the app is redeployed to App Store in order for the devices to roll back the app (Apple Inc. 2021f). Because incrementing the build string in iOS apps seems to need to be done before the app is built (Apple Inc. 2021f), an earlier version of an iOS app would need to be rebuilt before redeploying it to App Store. Based on this, it is assumed that native apps can be rolled back if the source code and assets of an earlier version of the app can be restored for rebuilding and redeployment. This thesis will only inspect this for the app-specific source code and assets and thus leave out any other content, such as libraries, that is required by the app. The source code and assets of Ionic (GitHub, Inc. 2021e), NativeScript (GitHub, Inc. 2021i), Xamarin (GitHub, Inc. 2021q), and the native approach (GitHub, Inc. 2021a, 2021p) apps can be found on GitHub²¹, which means that apps developed with these frameworks and the native approach can be version controlled, thus the app-specific source code and assets of the app can be rolled back for rebuilding and redeployment. Native mobile apps developed with Mendix seems to also be possible to store on GitHub (GitHub, Inc. 2021f). This thesis assumes that GitHub, Inc. (2021f) works for Mendix native apps instead of Mendix hybrid apps, because Mendix hybrid apps are no longer supported. This would mean that the app-specific source code and assets of Mendix native apps can be rolled back for rebuilding and redeployment.

6.8.7 Testing

Table 18 shows the supported requirements related to testing for the frameworks and the native approach. The implemented web app supports unit testing. This thesis assumes that the web app can test on the required browsers and their versions, which means that Vue.js also supports the unit testing requirement. Mendix provides a unit test module enables unit testing within Mendix applications (Mendix Technology BV 2025f). According to Mendix Technology BV (2025f): “You can test your microflows, Java actions, and other logic by implementing test microflows and JUnit (Java) tests.” There is also a module for Mendix which claims to enabling unit testing for nanoflows (Mendix Technology BV 2025g). Based on this, this thesis assumes that it is possible to test at least microflows, nanoflows, and Java actions as well as assumes that these are adequate for the unit testing requirement. The native approach supports unit testing (Android Developers 2025a; Apple Inc. 2025i). Ionic seems to support unit testing (GitHub, Inc. 2025a; Lynch 2017). NativeScript 8.9 seems to support unit testing (NativeScript 2025e). According to NativeScript (2025e): “With the NativeScript CLI, you can extensively test **all JavaScript-related functionality**. You cannot test styling and UI which are not applied or created via JavaScript.” This thesis assumes

21. <https://github.com/>

that NativeScript 7 also supports unit testing, thus fulfilling the requirement.

Table 18: Supported requirements related to testing for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Unit testing	✓	✓	✓	—	✓	✓

6.9 Requirement summary

Table 19 shows the supported requirements for the frameworks and the native approach. The table does not include requirements that are fulfilled by all frameworks and the native approach, but instead they are listed here:

- System requirements:
 - OpenID Connect
 - RESTful HTTPS API
- Functional requirements:
 - Appearance configuration
- Regulatory/Legislative requirements:
 - GDPR
- Other requirements:
 - Rollback

Requirements not assessed:

- Regulatory/Legislative requirements:
 - Accessibility
- Non-functional requirements:
 - Performance

Table 19: Supported requirement for each framework and the native approach.

Requirement	Vue.js	Ionic	Native-Script	Xamarin	Mendix	Native approach
Hardware requirements						
Camera	✗ ¹	✓	✓	✓	✓	✓
Click to call	✓	✓	✓	—	✓	✓
Notifications	✗ ²	✓	✓	✓	✓	✓
Passwordless authentication ³	✗ ⁴	✓	✓	✓	✓	✓
Functional requirements						
Dynamic UI	✓	✓	✓	—	✓	✓
Local storage	✓	✓	✓	—	✓	✓
Non-functional requirements						
Maintenance	✓	✓ ⁵	✗	✗	✓ ⁵	✗
Modularity	✓	✗	✗	—	✗	✗
Other requirements						
App store distribution	—	✓	✓	✓	✓	✓
Automatic logout	✓	✓	✓	—	✓	✓
Multi-language support	✓	✓	✗ ⁶	✗ ⁶	✗ ⁶	✗ ⁶
Operating systems	(✗)	(✗)	(✓)	(✓)	(✗)	(✓)
Android 5–	✓	✓	✓	✓	✓	✓
iOS 9–	✗ ⁷	✗ ¹	✓	✓	✗ ⁸	✓
Responsive design	✓	✓	?	✓	✓	✓
Testing	✓	✓	✓	—	✓	✓
Sum	10/14 (71%)	13/15 (87%)	11/15 (73%)	7/9 (78%)	12/15 (80%)	12/15 (80%)

¹ iOS 11 and later.

² No iOS support.

³ Fingerprint-based authentication requires Android 6.0 or later.

⁴ iOS 14 and later. Android support unknown.

⁵ Requires the web app to be adapted or rewritten to some extent.

⁶ iOS 13 and later.

⁷ iOS 11.3 and later.

⁸ iOS 14 and later.

7 Discussion

Sections 7.1–7.4 in this chapter answer the research questions described in Section 1.1 based on the results gotten in Chapter 6. Section 7.5 discusses the limitations of the study.

7.1 RQ1: What are the requirements regarding the case app?

This thesis identified 21 requirements regarding the case app, excluding requirements that depend on other requirements. These requirements are listed in Section 6.9. Some of these requirements were further divided into sub-requirements which can be identified in their respective sections. The thesis grouped the requirements into six categories: hardware features as well as system, functional, regulative/legislative, quality, and other requirements. Table 3 lists the requirements categorized.

7.2 RQ2: Which of the case app requirements are met for each technical framework and the native approach?

Table 19 in the previous chapter (Chapter 5) shows the requirements met for each framework. Based on the results, no development framework meets all the requirements set for the case app. Ionic has the highest relatively met requirements. Mendix and the native approach has second highest relatively met requirements. The native approach meets nearly all the functional requirements, but fails to meet the maintenance and modularity requirement. Ionic again fulfills the maintenance requirement, but fails to work on all required operating systems. Vue.js has the least relatively met requirements. Nevertheless, Vue.js fulfills both the maintenance and modularity requirements, which no other development approach can meet at the same time.

7.3 RQ3: Based on the met case app requirements, how do the technical frameworks and the native approach compare?

The native approach and possibly Xamarin meet nearly all the functional requirements listed in Section 6.9. However, both fail in one or both quality requirements, which are maintenance and modularity. Vue.js is the only development approach able to meet both of these requirements, but has the most unmet functional requirements. Ionic meets the multi-language requirement where Xamarin and the native approach fail, but Ionic fails support the required operating system versions. This thesis treats maintenance and modularity as absolute requirements, but there is an argument to made that these requirements should be treated as relative requirement and thus evaluate the requirements in relation to each development approach. The development approaches could be put on a scale, where on one end is development and maintenance effort and on the other end is app capability. Based on the results, Vue.js would be on the former side of the scale and the native approach on the later side of the scale. The other approaches would fall somewhere in between the ends of the scale.

7.4 RQ3.1: Based on the technical framework and the native approach comparison, how do the development approaches compare?

Based on the results for the technical frameworks and the native approach, there are differences that can be generalized to the development approaches. When inspecting the capabilities of each approach, the PWA approach has in this case limited capability, whereas the other approaches can theoretically have the same capabilities. The difference between the development approaches, excluding the PWA approach, can be seen in the maintenance requirement. Of these approaches, the hybrid and the model-driven approaches fare best. This is due to the possible capability of producing both a PWA and mobile apps for each targeted operating system. Next comes the interpreted and the cross-compiled approaches. These approaches can produce mobile apps for each targeted operating system, but are not capable to produce a web app. On last place is the native approach which needs separate implementations for each supported operating system and that cannot produce a web app. If the PWA approach is taken into account, it can be used as a web app and can function as a mobile app. However, the definition of mobile app in this case is loose, because it differs from the native mobile apps the other approaches produce.

7.5 Limitations of the study

The requirements engineering process in this thesis has been only performed partly. A complete requirements engineering process consists of requirements elicitation, specification, validation, and change, which usually are undergone multiple times in order to get a comprehensive, correct, and current requirements specification. This thesis has only undergone the requirements elicitation and specification phases in a short manner, thus the requirements specification (Chapter 5) for the app presents only a preliminary specification of the app. Nevertheless, the author believes that the requirements gathered through the means presented in the methods chapter (Chapter 4) has resulted in the elicitation of the most significant requirements, which should be sufficient in answering the research questions to a satisfactory level. Having a comprehensive requirements specification probably not result in a major change in the research problem, but instead would lead to the study falling out of scope for a master's thesis. Another possible issue with partial requirements engineering process is that requirements could be incorrect. This is due to the lack of verification and validation process, which should be normally be included in the requirements engineering process.

Due to the limited scope of a master's thesis, analyzing which requirements were met for each framework was done in a superficial manner, which means that some proportion of which requirements are met and not met could be incorrect. Regarding the native Android approach, the Android for Developers website²² provided in the thesis author's opinion the best documentation. Because almost all the results for

22. <https://developer.android.com/>

the native Android approach was retrieved from this site, the author's confidence for the result validity for the native Android approach is the highest. Probably on the second most accurate results are for the iOS native approach, which were mostly gotten from the Apple Developer website²³. After these two native approaches, the resources for the PWA approach were the most encompassing. Because PWA is essentially a web app, there is a vast amount of resources concerning different functionalities and other resources. I would say the most lacking in documentation were related to Ionic, NativeScript, and Mendix, which made finding relevant information difficult. Nevertheless, open source frameworks enables inspecting the code for which requirements are met, but due to the vast amount of time which it would take to determine which requirements are met, the source code was left out as a source for analyzing the results. Another way to determine which requirements are met would be to implement prototypes to determine the support. However, this is also a very laborious way to determine which requirements are met.

Due to the vast amount of discovered requirements, this thesis only compared a subset of these requirements. Firstly, the thesis did only include requirements that were deemed mandatory. Requirements that are optional would be left out from the thesis. To even further decrease the amount of requirements inspected in the thesis, there were some additional criteria for these mandatory requirements to fulfill in order to be included. Requirements that are imperative to a use case would be included.

23. <https://developer.apple.com/>

8 Conclusion

All the development approaches, excluding the PWA approach, could theoretically have all the capabilities for a mobile app that Android and iOS provide.

The native approach, comes at the highest cost of all the approaches. Instead of implementing one web app, the company would have to develop three apps: the web app, the Android app, and the iOS app. This would, besides additional resources, require the company to possibly hire and/or educate staff that are capable of implementing, testing, deploying, and maintaining Android and iOS apps.

The PWA approach is the least capable in terms of functional requirements. Arguably the most significant lacking functionality is notifications on iOS. Whereas two other functional requirements, usage of the device's camera and passwordless authentication, were not either supported, at least on iOS. IOS does not support notifications at all, no matter the version assessed in this thesis. Whereas the "functional gap" in most cases could be closed in time, the future of notifications remain uncertain. For the case app this is made worse, because notification support was seemingly deemed as one of the most important requirements for the mobile app version of the case app. The arguably most significant upside with the PWA approach is the lowest implementation and maintenance cost when regarding the case app. Converting the web app to a PWA would be trivial compared to implementing a mobile app with the other approaches. The implementation of the web app would continue as a PWA, with arguably a very small addition to the development costs compared to developing a web app alone.

The third option would be to choose a cross-platform development approach. Here arguably the differentiating factor for the case app is what products the approach is capable of producing. In this case, the hybrid and the model-driven approaches could be theoretically capable of producing with one implementation all the required apps for the Accountor HR Solutions. Arguably one of the main drawbacks compared to the PWA approach is that the web app would have to be partly or fully reimplemented and then extended to work with the required native APIs. Arguably another main drawback would be that maintaining the code related to the native APIs would most likely be more costly than the maintaining the PWA-exclusive capabilities for a web app.

Depending on the weights of the different assessed requirement, this thesis finds the PWA approach, the hybrid approach, or the model-driven approach as the most suitable approach to develop the case app.

8.1 Thesis contribution

Originally, the intended thesis contribution was to hopefully serve as a deciding factor for the approach that would be selected to start developing the mobile app. However, due to factors this ended not to be the case. Instead, this thesis hopes that the collaborative partner can instead gain useful insights for their future endeavors.

8.2 Future work

After choosing the most appropriate development approach for the app, the next step would be to determine the most appropriate framework under that approach. The selection of framework could be seen as an item for future research or work.

One requirement that this thesis could not compare was information security, which was concluded to be one of the most critical requirements for the case app, due to the app handling very sensitive information, such as users' bank accounts. This thesis suggests that further study in framework and development approach should be conducted that includes information security. These studies could provide essential information for the industry when deciding on the framework for developing their app in. The studies in framework information security should be continuous, because even a study in this which is a few years old will not be useful due to the rapid development of these frameworks.

This thesis also suggests that accessibility compliance and performance metrics should be studied for different technical frameworks in order to provide arguably important data for comparing frameworks.

References

- Agramonte, Charlin. 2021. “Mastering Multilingual in Xamarin.Forms.” *Xamarin Blog* (blog). Accessed October 24. <https://devblogs.microsoft.com/xamarin/mastering-multilingual-in-xamarin-forms/>.
- Ahti, Ville, Sami Hyrynsalmi, and Olli Nevalainen. 2016. “An Evaluation Framework for Cross-Platform Mobile App Development Tools: A Case Analysis of Adobe PhoneGap Framework.” In *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*, 41–48. CompSysTech ’16, Palermo, Italy. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450341820. <https://doi.org/10.1145/2983468.2983484>.
- Android Developers. 2020. “Linear Layout.” Last modified November 18. <https://developer.android.com/guide/topics/ui/layout/linear>.
- . 2021a. “App resources overview.” Last modified October 27. <https://developer.android.com/guide/topics/resources/providing-resources>.
- . 2021b. “Fonts in XML.” Last modified October 27. <https://developer.android.com/guide/topics/ui/look-and-feel/fonts-in-xml>.
- . 2021c. “Support different languages and cultures.” Last modified September 17. <https://developer.android.com/training/basics/supporting-devices/languages>.
- . 2022a. “ImageView.” Last modified February 10. <https://developer.android.com/reference/android/widget/ImageView>.
- . 2022b. “Styles and Themes.” Last modified March 7. <https://developer.android.com/guide/topics/ui/look-and-feel/themes>.
- . 2024a. “Android 7.0 for Developers.” Last modified May 20. <https://developer.android.com/about/versions/nougat/android-7.0.html>.
- . 2024b. “Android Lollipop.” Last modified May 20. <https://developer.android.com/about/versions/lollipop>.
- . 2024c. “Send a simple request.” Last modified January 3. <https://developer.android.com/develop/connectivity/cronet/start>.
- . 2025a. “Build local unit tests.” Last modified February 10. <https://developer.android.com/training/testing/local-tests>.
- . 2025b. “Data and file storage overview.” Last modified June 24. <https://developer.android.com/training/data-storage/>.
- . 2025c. “Intent.” Last modified March 13. <https://developer.android.com/reference/android/content/Intent>.
- . 2025d. “Per-app language preferences.” Last modified February 10. <https://developer.android.com/guide/topics/resources/app-languages>.

- Android Developers. 2025e. “ProcessLifecycleOwner.” Last modified May 15. <https://developer.android.com/reference/androidx/lifecycle/ProcessLifecycleOwner.html>.
- . 2025f. “View.” Last modified April 17. <https://developer.android.com/reference/android/view/View>.
- . 2025g. “ViewGroup.” Android Developers. Last modified April 17. <https://developer.android.com/reference/android/view/ViewGroup>.
- Android Open Source Project. 2021a. “Android 6.0 APIs.” Android Developers. Last modified March 11. <https://developer.android.com/about/versions/marshmallow/android-6.0>.
- . 2021b. “Android Security Bulletin—May 2021.” Android Open Source Project. Last modified May 4. <https://source.android.com/security/bulletin/2021-05-01>.
- . 2021c. “Camera API.” Android Developers. Last modified May 18. <https://developer.android.com/guide/topics/media/camera>.
- . 2021d. “URLConnection.” Android Developers. Last modified February 18. <https://developer.android.com/reference/java/net/URLConnection.html>.
- . 2021e. “Show a biometric authentication dialog.” Android Developers. Last modified July 27. <https://developer.android.com/training/sign-in/biometric-auth>.
- Apple Inc. 2016. “Technical Q&A QA1828: How iOS Determines the Language For Your App.” Apple Developer. Last modified March 23. https://developer.apple.com/library/archive/qa/qa1828/_index.html.
- . 2021a. “Apple security updates.” Apple Support. Accessed May 31. <https://support.apple.com/en-us/HT201222>.
- . 2021b. “Localization.” Apple Developer. Accessed October 24. <https://developer.apple.com/localization/>.
- . 2021c. “Logging a User into Your App with Face ID or Touch ID.” Apple Developer Documentation. Accessed October 12. https://developer.apple.com/documentation/localauthentication/logging_a_user_into_your_app_with_face_id_or_touch_id.
- . 2021d. “Minimum iOS Deployment Target in Xcode 13.” Accessed October 17. <https://developer.apple.com/forums/thread/691201>.
- . 2021e. “Safari 14 Release Notes.” Apple Developer Documentation. Accessed August 5. <https://developer.apple.com/documentation/safari-release-notes/safari-14-release-notes>.

- Apple Inc. 2021f. “Set the version number and build string.” Xcode Help. Accessed October 28. <https://developer.apple.com/documentation/xcode/preparing-your-app-for-distribution>.
- . 2021g. “UIImagePickerController.” Apple Developer Documentation. Accessed June 14. <https://developer.apple.com/documentation/uikit/uiimagepickercontroller?language=objc>.
- . 2021h. “UIInterfaceOrientation.” Apple Developer. Accessed November 24. <https://developer.apple.com/documentation/uikit/uiinterfaceorientation>.
- . 2021i. “UIStackView.” Apple Developer. Accessed November 22. <https://developer.apple.com/documentation/uikit/uistackview>.
- . 2021j. “UITraitCollection.” Apple Developer. Accessed November 24. <https://developer.apple.com/documentation/uikit/uitraitcollection>.
- . 2021k. “UIUserNotificationSettings.” Apple Developer Documentation. Accessed October 11. <https://developer.apple.com/documentation/uikit/uiusernotificationsettings>.
- . 2021l. “URLRequest.” Apple Developer Documentation. Accessed October 5. <https://developer.apple.com/documentation/foundation/urlrequest>.
- . 2022a. “Adding a Custom Font to Your App.” Apple Developer. Accessed March 9. https://developer.apple.com/documentation/uikit/text_display_and_fonts/adding_a_custom_font_to_your_app?language=objc.
- . 2022b. “UIAppearance.” Apple Developer. Accessed March 9. <https://developer.apple.com/documentation/uikit/uiappearance?language=objc>.
- . 2022c. “UIImageView.” Apple Developer. Accessed March 9. <https://developer.apple.com/documentation/uikit/uiimageView?language=objc>.
- . 2025a. “authorizationViewDidHide:” Apple Developer. Accessed July 8. [https://developer.apple.com/documentation/objectivec/nsobject-swift.class/authorizationviewdidhide\(_\)?language=objc](https://developer.apple.com/documentation/objectivec/nsobject-swift.class/authorizationviewdidhide(_)?language=objc).
- . 2025b. “Change the language of iOS 13.” Apple Developer Forums. Accessed July 14. <https://developer.apple.com/forums/thread/129946>.
- . 2025c. “CXStartCallAction.” Apple Developer. Accessed July 10. <https://developer.apple.com/documentation/callkit/cxstartcallaction>.
- . 2025d. “hidden.” Apple Developer. Accessed July 8. <https://developer.apple.com/documentation/AppKit/NSView/isHidden?language=objc>.
- . 2025e. “Layouts.” Apple Developer. Accessed July 8. <https://developer.apple.com/documentation/uikit/layouts?language=objc>.

- Apple Inc. 2025f. “Making and receiving VoIP calls.” Apple Developer. Accessed July 10. <https://developer.apple.com/documentation/callkit/making-and-receiving-voip-calls>.
- . 2025g. “NSCollectionLayoutItem.” Apple Developer. Accessed July 8. <https://developer.apple.com/documentation/uikit/nscollectionlayoutitem?language=objc>.
- . 2025h. “NSObject.” Apple Developer. Accessed July 8. <https://developer.apple.com/documentation/ObjectiveC/NSObject-swift.class?language=objc>.
- . 2025i. “Persistent storage.” Apple Developer. Accessed July 8. <https://developer.apple.com/documentation/swiftui/persistent-storage>.
- . 2025j. “UIApplicationDelegate.” Apple Developer. Accessed June 18. <https://developer.apple.com/documentation/uikit/uiapplicationdelegate>.
- . 2025k. “URLSession.” Apple Developer. Accessed June 18. <https://developer.apple.com/documentation/foundation/urlsession>.
- . 2025l. “XCTest.” Apple Developer. Accessed July 13. <https://developer.apple.com/documentation/xctest>.
- Bar, Adam. 2025. “Foreground Detection.” What Web Can Do Today? Accessed June 18. <https://whatwebcando.today/foreground-detection.html>.
- Biørn-Hansen, Andreas, Tor-Morten Grønli, and Gheorghita Ghinea. 2018. “A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development.” *ACM Computing Surveys* (New York, NY, USA) 51, no. 5 (November). ISSN: 0360-0300. <https://doi.org/10.1145/3241739>.
- Biørn-Hansen, Andreas, Christoph Rieger, Tor-Morten Grønli, Tim A. Majchrzak, and Gheorghita Ghinea. 2020. “An empirical investigation of performance overhead in cross-platform mobile development frameworks.” *Empirical Software Engineering* 25, no. 4 (June 9, 2020): 2997–3040. <https://doi.org/10.1007/s10664-020-09827-6>.
- Butterfield, Andrew, Gerard Ekembe Ngondi, and Anne Kerr, eds. 2016. *A Dictionary of Computer Science*. Seventh. Oxford University Press. <https://doi.org/10.1093/acref/9780199688975.001.0001>.
- Can I use. 2021a. “Add to home screen (A2HS).” Can I use. Accessed October 17. <https://caniuse.com/web-app-manifest>.
- . 2021b. “CSS at-rule: @media: height media feature.” Accessed November 24. https://caniuse.com/mdn-css_at-rules_media_height.
- . 2021c. “CSS at-rule: @media: orientation media feature.” Accessed November 24. https://caniuse.com/mdn-css_at-rules_media_orientation.
- . 2021d. “CSS at-rule: @media: width media feature.” Accessed November 24. https://caniuse.com/mdn-css_at-rules_media_width.

- Can I use. 2021e. “CSS Flexible Box Layout Module.” Accessed November 24. <https://caniuse.com/flexbox>.
- . 2021f. “CSS property: max-height.” Accessed November 24. https://caniuse.com/mdn-css_properties_max-height.
- . 2021g. “CSS property: max-width.” Accessed November 24. https://caniuse.com/mdn-css_properties_max-width.
- . 2021h. “Push API.” Can I use. Accessed October 11. <https://caniuse.com/push-api>.
- . 2021i. “Web Notifications.” Can I use. Accessed October 11. <https://caniuse.com/notifications>.
- . 2025. “Page Visibility.” Can I use. Accessed June 18. <https://caniuse.com/pagevisibility>.
- Capacitor. 2021a. “Camera.” Capacitor. Accessed June 11. <https://capacitorjs.com/docs/v2/apis/camera>.
- . 2021b. “Creating Capacitor Plugins.” Capacitor. Accessed October 12. <https://capacitorjs.com/docs/v2/plugins/creating-plugins>.
- . 2021c. “Local Notifications Capacitor Plugin API.” Capacitor. Accessed October 8. <https://capacitorjs.com/docs/apis/local-notifications>.
- . 2021d. “Push Notifications Capacitor Plugin API.” Capacitor. Accessed October 8. <https://capacitorjs.com/docs/apis/push-notifications>.
- . 2025. “Deploying your Capacitor Android App to the Google Play Store.” Capacitor Documentation. Accessed July 14. <https://capacitorjs.com/docs/android/deploying-to-google-play>.
- Fielding, Roy Thomas. 2000. “Architectural Styles and the Design of Network-based Software Architectures.” PhD diss., University of California. <https://www.ics.uci.edu/~fielding/pubs/dissertation/faq.htm>.
- Firtman, Maximiliano. 2020. “Safari on iOS 14 and iPadOS 14 for PWA and Web Developers.” *firt.dev*, September 16, 2020. Accessed January 25, 2021. <https://firt.dev/ios-14>.
- Fisher, John. 2017. “UIStackView Tricks: Proportional Custom UIViews with ‘Fill Proportionally’.” *Atomic Object* (blog), February 7, 2017. <https://spin.atomicobject.com/2017/02/07/uistackview-proportional-custom-uiviews/>.
- GitHub, Inc. 2021a. “android/sunflower: A gardening app illustrating Android development best practices with Android Jetpack.” GitHub. Accessed October 28. <https://github.com/android/sunflower>.

- GitHub, Inc. 2021b. “EddyVerbruggen/nativescript-plugin-firebase: NativeScript plugin for Firebase.” GitHub. Accessed October 8. <https://github.com/EddyVerbruggen/nativescript-plugin-firebase>.
- . 2021c. “IdentityModel/IdentityModel.OidcClient: Certified C#/NetStandard OpenID Connect Client Library for native mobile/desktop Applications (RFC 8252).” GitHub. Accessed October 22. <https://github.com/IdentityModel/IdentityModel.OidcClient>.
- . 2021d. “IdentityModel/IdentityModel.OidcClient.Samples.” GitHub. Accessed October 22. <https://github.com/IdentityModel/IdentityModel.OidcClient.Samples>.
- . 2021e. “ionic-team/ionic-vue-conference-app: Ionic Conference app ported to Vue.” GitHub. Accessed October 28. <https://github.com/ionic-team/ionic-vue-conference-app>.
- . 2021f. “mendix/StarterApp_NativeMobile.” GitHub. Accessed October 28. https://github.com/mendix/StarterApp_NativeMobile.
- . 2021g. “mendix/widgets-resources: Mono repository of Mendix Platform Supported Widgets.” GitHub. Accessed October 13. <https://github.com/mendix/widgets-resources>.
- . 2021h. “NativeScript/nativescript-camera: NativeScript plugin to empower using device camera.” GitHub. Accessed March 15. <https://github.com/NativeScript/nativescript-camera>.
- . 2021i. “NativeScript/nativescript-marketplace-demo: NativeScript kitchen sink demo. All of NativeScript’s functionality in one app.” GitHub. Accessed October 28. <https://github.com/NativeScript/nativescript-marketplace-demo>.
- . 2021j. “NiklasMerz/cordova-plugin-fingerprint-aio: Cordova Plugin for fingerprint sensors (and FaceID) with Android and iOS support.” GitHub. Accessed October 12. <https://github.com/NiklasMerz/cordova-plugin-fingerprint-aio>.
- . 2021k. “openid/AppAuth-Android: Android client SDK for communicating with OAuth 2.0 and OpenID Connect providers.” GitHub. Accessed October 22. <https://github.com/openid/AppAuth-Android>.
- . 2021l. “openid/AppAuth-iOS: iOS and macOS SDK for communicating with OAuth 2.0 and OpenID Connect providers.” GitHub. Accessed October 22. <https://github.com/openid/AppAuth-iOS>.
- . 2021m. “plugins/packages/local-notifications at main · NativeScript/plugins.” GitHub. Accessed October 8. <https://github.com/NativeScript/plugins/tree/main/packages/local-notifications>.

- GitHub, Inc. 2021n. “triniwiz/nativescript-videorecorder: NativeScript plugin for Video Recording.” GitHub. Accessed April 30. <https://github.com/triniwiz/nativescript-videorecorder>.
- . 2021o. “wi3land/ionic-appauth: Intergration for OpenId/AppAuth-JS into Ionic V3/4/5.” GitHub. Accessed October 22. <https://github.com/wi3land/ionic-appauth>.
- . 2021p. “wikimedia/wikipedia-ios: The official Wikipedia iOS app.” GitHub. Accessed October 28. <https://github.com/wikimedia/wikipedia-ios>.
- . 2021q. “xamarin-forms-samples/ToDo at main · xamarin/xamarin-forms-samples.” GitHub. Accessed October 28. <https://github.com/xamarin/xamarin-forms-samples/tree/main/ToDo>.
- . 2025a. “Ionic Unit Testing Example.” GitHub. Accessed July 13. <https://github.com/ionic-team/ionic-unit-testing-example>.
- . 2025b. “msywensky/nativescript-phone.” GitHub. Accessed July 10. <https://github.com/msywensky/nativescript-phone>.
- Google. 2021. “S-mobiili – Google Play -sovellukset.” Google Play. Accessed October 7. <https://play.google.com/store/apps/details?id=fi.spankki>.
- Google Developers. 2019. “Introduction to Push Notifications | Web.” Last modified May 1. <https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications>.
- . 2021. “Set up a Firebase Cloud Messaging client app on iOS.” Firebase. Last modified October 6. <https://firebase.google.com/docs/cloud-messaging/ios/client>.
- Hakulinen, Roope. 2021. “What is a Progressive Web App (PWA) and why should I care?” *Gofore* (blog). Accessed January 25. <https://gofore.com/progressive-web-app-pwa-i-care-2/>.
- Hathibelagal, Ashraff. 2021. “Android From Scratch: Using REST APIs.” Envato Tuts+, April 18, 2021. Accessed June 17, 2021. <https://code.tutsplus.com/tutorials/android-from-scratch-using-rest-apis--cms-27117>.
- Hoffman, Chris. 2019. “How to Change the Language of an App on Your iPhone or iPad.” *How-To Geek*. Last modified September 24. <https://www.howtogeek.com/441716/how-to-change-the-language-of-an-app-on-your-iphone-or-ipad/>.
- Hughes, Neil. 2021. “Apple opens Touch ID to third-party applications with iOS 8.” *AppleInsider*. Accessed August 3.
- Ionic. 2020a. “Browser Support.” Last modified December 7. <https://ionicframework.com/docs/reference/browser-support>.
- . 2020b. “Responsive Grid.” Last modified December 7. <https://ionicframework.com/docs/layout/grid>.

- Ionic. 2020c. “Support Policy.” Ionic Documentation. Last modified December 7. <https://ionicframework.com/docs/reference/support>.
- . 2021. “Install Camera | Cordova Plugin Cameras for Ionic Applications.” Ionic Framework. Accessed April 30. <https://ionicframework.com/docs/native/camera>.
- . 2025a. “Android Play Store Deployment.” Ionic DOCS. Accessed July 14. <https://ionicframework.com/docs/deployment/play-store>.
- . 2025b. “Data Storage.” Ionic DOCS. Accessed July 8. <https://ionicframework.com/docs/react/storage>.
- . 2025c. “iOS App Store Deployment.” Ionic DOCS. Accessed July 14. <https://ionicframework.com/docs/deployment/app-store>.
- . 2025d. “Platform.” Ionic DOCS. Accessed June 19. <https://ionicframework.com/docs/angular/platform>.
- Khachouch, Mohamed Karim, Ayoub Korchi, Younes Lakhrissi, and Anis Moumen. 2020. “Framework Choice Criteria for Mobile Application Development.” In *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, 1–5. <https://doi.org/10.1109/ICECCE49384.2020.9179434>.
- Kitamura, Eiji. 2021. “Build your first WebAuthn app.” Accessed October 13. <https://codelabs.developers.google.com/codelabs/webauthn-reauth>.
- Lachgar, Mohamed, and Abdelmounaïm Abdali. 2017. “Decision Framework for Mobile Development Methods,” vol. 8. 2. The Science / Information Organization. <https://doi.org/10.14569/IJACSA.2017.080215>.
- LePage, Pete. 2014. “Click to Call.” *web.dev*. Last modified June 17. <https://web.dev/articles/click-to-call>.
- . 2021. “WebAPKs on Android | Web Fundamentals.” Google Developers. Last modified May 18. <https://developers.google.com/web/fundamentals/integration/webapks>.
- Lynch, Max. 2017. “Basic Unit Testing in Ionic.” *Ionic Blog* (blog). Accessed March 23. <https://ionic.io/blog/basic-unit-testing-in-ionic>.
- Majchrzak, Tim A., Jan Ernsting, and Herbert Kuchen. 2015. “Achieving Business Practicability of Model-Driven Cross-Platform Apps.” *Open Journal of Information Systems (OJIS)* 2 (2). <http://hdl.handle.net/11250/2392249>.
- Mendix Technology BV. 2021a. “Call REST Service - Studio Pro 9 Guide.” Mendix Documentation. Last modified August 25. <https://docs.mendix.com/refguide/call-rest-action>.
- . 2021b. “Deploy Your First Mendix Native Mobile App.” Mendix Documentation. Last modified August 30. <https://docs.mendix.com/howto/mobile/deploying-native-app>.

- Mendix Technology BV. 2021c. “Images.” Mendix Docs. Last modified September 23. <https://docs.mendix.com/refguide/images>.
- . 2021d. “Language Menu.” Mendix Docs. Last modified July 19. <https://docs.mendix.com/refguide/translatable-texts>.
- . 2021e. “Language Settings.” Mendix Docs. Last modified May 31. <https://docs.mendix.com/refguide/language-settings>.
- . 2021f. “Layout Grid.” Mendix Docs. Last modified September 23. <https://docs.mendix.com/refguide/layout-grid>.
- . 2021g. “OpenIDConnect (OIDC) Single Sign-on (SSO).” Accessed October 22. <https://marketplace.mendix.com/link/component/117529>.
- . 2021h. “Properties Common in the Page Editor.” Mendix Docs. Last modified July 6. <https://docs.mendix.com/refguide/common-widget-properties>.
- . 2021i. “Set Up Push Notifications - Studio Pro 9 How-to’s.” Mendix Documentation. Last modified June 10. <https://docs.mendix.com/howto/mobile/setting-up-native-push-notifications>.
- . 2021j. “Translate Your App Content.” Mendix Docs. Last modified June 18. <https://docs.mendix.com/howto/collaboration-requirements-management/translate-your-app-content>.
- . 2021k. “Use Local Notifications - Studio Pro 9 How-to’s.” Mendix Documentation. Last modified June 10. <https://docs.mendix.com/howto/mobile/local-notif-parent>.
- . 2022a. “Build a Pluggable Native Widget.” Mendix Docs. Last modified January 11. <https://docs.mendix.com/howto/extensibility/build-native-widget>.
- . 2022b. “Native Mobile Styling.” Mendix Docs. Last modified February 24. <https://docs.mendix.com/refguide/native-styling-refguide>.
- . 2022c. “Native: how to change styles dynamically.” Mendix Forum. Accessed March 9. <https://forum.mendix.com/link/questions/104274>.
- . 2022d. “Properties Common in the Page Editor.” Mendix Docs. Last modified January 5. <https://docs.mendix.com/refguide/common-widget-properties>.
- . 2024a. “2.6 Items Section - Button Properties.” Mendix Documentation. Last modified August 20. <https://docs.mendix.com/refguide9/button-properties/#items>.
- . 2024b. “3.5.1 Link Types - On Click Event and Events Section.” Mendix Documentation. Last modified September 1. <https://docs.mendix.com/refguide9/on-click-event/#on-click-link-type>.
- . 2024c. “Buttons.” Mendix Documentation. Last modified August 20. <https://docs.mendix.com/refguide9/button-widgets/>.

- Mendix Technology BV. 2024d. “Control Bar.” Mendix Documentation. Last modified August 20. <https://docs.mendix.com/refguide9/control-bar/>.
- . 2024e. “JavaScript Actions.” Mendix Documentation. Accessed August 20. <https://docs.mendix.com/refguide9/javascript-actions/>.
- . 2025a. “Async calls to REST API.” Mendix Community. Accessed June 18. <https://community.mendix.com/link/space/microflows/questions/114808>.
- . 2025b. “Make a telephone call from mobile app.” Mendix Community. Accessed July 10. <https://community.mendix.com/link/space/mobile/questions/92315>.
- . 2025c. “Mendix and data privacy.” Accessed June 19. <https://www.mendix.com/trust/mendix-and-data-privacy/>.
- . 2025d. “Native App Event Listener.” Mendix marketplace. Accessed June 19. <https://marketplace.mendix.com/link/component/240679>.
- . 2025e. “Offline-First Apps.” Accessed July 8. <https://www.mendix.com/evaluation-guide/app-lifecycle/develop/ux-multi-channel-apps/offline-first-apps/>.
- . 2025f. “Unit Testing.” Mendix Documentation. Last modified June 18. <https://docs.mendix.com/appstore/modules/unit-testing/>.
- . 2025g. “Unittesting for nanoflows (also native!)” Mendix Marketplace. Accessed July 13. <https://marketplace.mendix.com/link/component/228534>.
- . 2025h. “Will throw others HTTP responses than 200 OK or 304 an exception?” Mendix Community. Accessed June 18. <https://community.mendix.com/link/space/integrations/questions/117465>.
- Meta Platforms, Inc. 2022. “Text Style Props.” React Native. Last modified January 19. <https://reactnative.dev/docs/text-style-props>.
- Microsoft. 2021a. “App Store Distribution.” Microsoft Docs. Accessed October 16. <https://docs.microsoft.com/en-us/xamarin/ios/deploy-test/app-distribution/app-store-distribution/>.
- . 2021b. “Consume a RESTful web service - Xamarin.” Microsoft Docs. Accessed October 5. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/web-services/rest>.
- . 2021c. “HttpClient Class (System.Net.Http).” Microsoft Docs. Accessed October 5. <https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-5.0>.
- . 2021d. “Image Class.” Microsoft Docs. Accessed November 24. <https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.image?view=xamarin-forms>.

- Microsoft. 2021e. “Images in Xamarin.Forms.” Microsoft Docs. Accessed November 24. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/images>.
- . 2021f. “Plugin.Fingerprint 2.1.4.” NuGet Gallery. Accessed October 13. <https://www.nuget.org/packages/Plugin.Fingerprint/>.
- . 2021g. “Preparing an Application for Release.” Microsoft Docs. Accessed October 27. <https://docs.microsoft.com/en-us/xamarin/android/develop/test/release-prep/>.
- . 2021h. “Publishing an Application.” Microsoft Docs. Accessed October 16. <https://docs.microsoft.com/en-us/xamarin/android/develop/test/publishing/>.
- . 2021i. “Transport Layer Security (TLS) 1.2 - Xamarin.” Microsoft Docs. Accessed October 6. <https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport-layer-security>.
- . 2021j. “Xamarin.Essentials: Media Picker - Xamarin.” Microsoft Docs. Accessed March 15. https://docs.microsoft.com/en-us/xamarin/essentials/media-picker?WT.mc_id=friends-0000-jamont&tabs=android.
- . 2021k. “Xamarin.Forms DependencyService Introduction - Xamarin.” Microsoft Docs. Accessed October 12. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction>.
- . 2021l. “Xamarin.Forms deployment and testing.” Microsoft Docs. Accessed October 16. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/develop/test/>.
- . 2021m. “Xamarin.Forms Grid.” Microsoft Docs. Accessed November 22. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/layouts/grid>.
- . 2021n. “Xamarin.Forms local notifications - Xamarin.” Microsoft Docs. Accessed October 8. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/local-notifications>.
- . 2021o. “Xamarin.Forms String and Image Localization.” Microsoft Docs. Accessed October 24. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/localization/text?pivots=macos>.
- . 2021p. “Xamarin.Forms supported platforms.” Microsoft Docs. Accessed October 17. <https://docs.microsoft.com/en-us/xamarin/get-started/supported-platforms>.
- . 2021q. “Xamarin.iOS app distribution overview.” Microsoft Docs. Accessed October 16. <https://docs.microsoft.com/en-us/xamarin/ios/develop/test/app-distribution/>.

- Microsoft. 2022a. “Dynamic Styles in Xamarin.Forms.” Microsoft Docs. Accessed March 9. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/styles/xaml/dynamic>.
- . 2022b. “Image.Source Property.” Microsoft Docs. Accessed March 9. <https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.image.source?view=xamarin-forms>.
- . 2022c. “Theme a Xamarin.Forms Application.” Microsoft Docs. Accessed March 9. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/theming/theming>.
- . 2025. “Xamarin.” .NET. Accessed July 13. <https://dotnet.microsoft.com/en-us/apps/xamarin>.
- Mozilla. 2020a. “Secure contexts.” MDN. Last modified June 3. https://developer.mozilla.org/en-US/docs/Web/Security/Secure_Contexts.
- . 2020b. “Service Worker API.” MDN. Last modified December 15. https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.
- . 2021a. “Flexbox.” MDN Web Docs. Last modified October 13. https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox.
- . 2021b. “MediaDevices.getUserMedia() - Web APIs.” MDN. Last modified February 19. <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia>.
- . 2021c. “style.” MDN Web Docs. Last modified October 3. https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/style.
- . 2021d. “Using XMLHttpRequest - Web APIs.” MDN. Last modified June 1. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest.
- . 2022a. “: The Image Embed element.” MDN Web Docs. Last modified February 18. <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img>.
- . 2022b. “Applying color to HTML elements using CSS.” MDN Web Docs. Last modified February 2. https://developer.mozilla.org/en-US/docs/Web/HTML/Applying_color.
- . 2022c. “Fundamental text and font styling.” MDN Web Docs. Last modified February 18. https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling_text/Fundamentals.
- . 2024. “Ordering flex items.” Mdn web docs. Last modified November 7. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Ordering_Flex_Items.

- Mozilla. 2025a. “<a>: The Anchor element.” MDN. Last modified July 10. <https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/a>.
- . 2025b. “display.” Mdn web docs. Last modified June 26. <https://developer.mozilla.org/en-US/docs/Web/CSS/display>.
- . 2025c. “HTMLImageElement.” Mdn web docs. Last modified April 10. <https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement>.
- . 2025d. “max-height.” Mdn web docs. Last modified June 23. <https://developer.mozilla.org/en-US/docs/Web/CSS/max-height>.
- . 2025e. “max-width.” Mdn web docs. Last modified June 23. <https://developer.mozilla.org/en-US/docs/Web/CSS/max-width>.
- . 2025f. “order.” Mdn web docs. Last modified July 1. <https://developer.mozilla.org/en-US/docs/Web/CSS/order>.
- . 2025g. “visibility.” Mdn web docs. Last modified July 1. <https://developer.mozilla.org/en-US/docs/Web/CSS/visibility>.
- NativeScript. 2021a. “Fingerprint Auth.” NativeScript. Accessed August 5. <https://docs.nativescript.org/plugins/fingerprint-auth.html>.
- . 2021b. “Http.” NativeScript. Accessed June 16. <https://docs.nativescript.org/http.html>.
- . 2021c. “Localize.” NativeScript. Accessed October 24. <https://docs.nativescript.org/plugins/localize.html>.
- . 2021d. “Metadata Overview.” Accessed October 17. <https://v7.docs.nativescript.org/core-concepts/android-runtime/requirements>.
- . 2021e. “Plugin Reference.” NativeScript Docs. Accessed October 12. <https://v7.docs.nativescript.org/plugins/plugin-reference>.
- . 2021f. “Releasing.” NativeScript. Accessed October 16. <https://docs.nativescript.org/releasing.html>.
- . 2021g. “Supporting Multiple Screens.” Accessed November 24. <https://v7.docs.nativescript.org/ui/supporting-multiple-screens>.
- . 2021h. “System Requirements.” Accessed October 17. <https://v7.docs.nativescript.org/core-concepts/ios-runtime/requirements>.
- . 2021i. “System Requirements.” Accessed October 17. <https://v7.docs.nativescript.org/core-concepts/android-runtime/metadata/overview.html>.
- . 2022a. “Data binding.” Accessed March 9. <https://v7.docs.nativescript.org/core-concepts/data-binding>.
- . 2022b. “UI & Styling.” Accessed March 9. <https://docs.nativescript.org/ui-and-styling.html>.

- NativeScript. 2022c. “User Interface Styling.” Accessed March 9. <https://v7.docs.nativescript.org/ui/styling>.
- . 2025a. “Adding custom native code to a project.” Accessed July 13. <https://docs.nativescript.org/guide/adding-native-code>.
- . 2025b. “Application.” Accessed June 19. <https://docs.nativescript.org/core/application>.
- . 2025c. “ApplicationSettings.” Accessed July 8. <https://docs.nativescript.org/core/application-settings>.
- . 2025d. “Http.” Accessed July 12. <https://docs.nativescript.org/core/http>.
- . 2025e. “Unit Testing.” Accessed July 13. <https://docs.nativescript.org/guide/testing>.
- Nunkesser, Robin. 2018. “Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development.” In *2018 ACM/IEEE 5th International Conference on Mobile Software Engineering and Systems: MOBILESoft 2018*, edited by Juan E. Guerrero, 214–218. ICSE ’18: 40th International Conference on Software Engineering, Gothenburg, Sweden. New York, NY, USA: Association for Computing Machinery. ISBN: 978-1-4503-5712-8. <https://doi.org/10.1145/3197231.3197260>.
- OpenID. 2021. “OpenID Connect FAQ and Q&As.” OpenID. Accessed October 4. <https://openid.net/connect/faq/>.
- OpenJS Foundation. 2021. “Class Image.” NativeScript. Accessed November 24. <https://docs.nativescript.org/api-reference/classes/image.html>.
- Raj, Delpin Susai. 2021. “Xamarin.Forms - Fingerprint Authentication In your App.” *C# Corner*. Last modified February 17. <https://www.c-sharpcorner.com/article/xamarin-forms-fingerprint-authentication-in-your-app/>.
- Regional State Administrative Agency. 2024a. “Requirements of the Act on the Provision of Digital Services,” January 30, 2024. Accessed January 30, 2024. <https://www.webaccessibility.fi/requirements-of-the-act-on-the-provision-of-digital-services/>.
- . 2024b. “Scope: does the Act apply to us?,” January 29, 2024. Accessed January 29, 2024. <https://www.webaccessibility.fi/requirements-of-the-act-on-the-provision-of-digital-services/scope-does-the-act-apply-to-us/>.
- Richard, Sam, and Pete LePage. 2020. “What are Progressive Web Apps?” *web.dev*. Last modified February 24. <https://web.dev/what-are-pwas/>.

- Rieger, Christoph, and Herbert Kuchen. 2019. "Towards Pluri-Platform Development: Evaluating a Graphical Model-Driven Approach to App Development Across Device Classes." In *Towards Integrated Web, Mobile, and IoT Technology*, edited by Tim A. Majchrzak, Cristian Mateos, Francesco Poggi, and Tor-Morten Grønli, 36–66. Cham, Switzerland: Springer International Publishing. ISBN: 978-3-030-28430-5. https://doi.org/10.1007/978-3-030-28430-5_3.
- Rieger, Christoph, and Tim A. Majchrzak. 2019. "Towards the definitive evaluation framework for cross-platform app development approaches." *Journal of Systems and Software* 153:175–199. ISSN: 0164-1212. <https://doi.org/10.1016/j.jss.2019.04.001>.
- Russell, Alex. 2021a. "Progressive Web Apps: Escaping Tabs Without Losing Our Soul." *Infrequently Noted* (blog). Accessed January 21. <https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/>.
- . 2021b. "What, Exactly, Makes Something A Progressive Web App?" *Infrequently Noted* (blog). Accessed January 21. <https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/>.
- Software Freedom Conservancy. 2021. "About Selenium." Selenium. Accessed October 4. <https://www.selenium.dev/about/>.
- Sommer, Andreas, and Stephan Krusche. 2013. "Evaluation of cross-platform frameworks for mobile applications." In *Software Engineering 2013 - Workshopband*, edited by Stefan Wagner and Horst Lichter, 363–376. Bonn, Germany: Gesellschaft für Informatik e.V. Accessed August 7, 2020. <https://dl.gi.de/handle/20.500.12116/17386>.
- Sommerville, Ian. 2016. *Software engineering*. Tenth edition. Always learning. Boston, MA, USA: Pearson. ISBN: 978-1-292-09613-1.
- Stack Exchange Inc. 2021a. "Google Play Console - How remove an update of published application." Stack Overflow. Accessed October 28. <https://stackoverflow.com/questions/48191631/google-play-console-how-remove-an-update-of-published-application>.
- . 2021b. "Roll back version." Stack Overflow. Accessed October 28. <https://stackoverflow.com/questions/54322337/roll-back-version>.
- . 2021c. "UIImageView with auto-layout and max size keeping aspect ratio." Stack Overflow. Accessed November 24. <https://stackoverflow.com/questions/29287993/uiimageView-with-auto-layout-and-max-size-keeping-aspect-ratio/29290149>.
- . 2022. "Change Activity's theme programmatically." Stack Overflow. Accessed March 9. <https://stackoverflow.com/questions/11562051/change-activitys-theme-programmatically>.

- Stack Exchange Inc. 2025. “What version of mobile safari comes with each version of iOS?” Stack Overflow. Accessed July 12. <https://stackoverflow.com/questions/8627968/what-version-of-mobile-safari-comes-with-each-version-of-ios>.
- Stevens, Adrian. 2021. “Adaptive UI with Xamarin.Forms.” *Xamarin Blog* (blog). Accessed November 24. <https://devblogs.microsoft.com/xamarin/adaptive-ui-xamarin-forms/>.
- Terry, Ryan. 2025. “Adaptive Authentication Explained.” *CrowdStrike*. Last modified March 11. <https://www.crowdstrike.com/en-us/cybersecurity-101/identity-protection/adaptive-authentication/>.
- The Apache Software Foundation. 2021. “Architectural overview of Cordova platform.” Apache Cordova. Accessed January 26. <https://cordova.apache.org/docs/en/latest/guide/overview/index.html>.
- VanToll, TJ. 2018. “How to Build a PWA, an iOS App, and an Android App - From One Codebase.” *Xamarin Blog* (blog). Accessed October 30. <https://blog.nativescript.org/how-to-build-a-pwa-an-ios-app-and-an-android-app-from-one-codebase/>.
- Vue.js. 2022. “Template Syntax.” Accessed March 9. <https://v2.vuejs.org/v2/guide/syntax.html>.
- . 2025. “Client-Side Storage.” Accessed July 8. <https://v2.vuejs.org/v2/cookbook/client-side-storage.html?redirect=true>.
- What Web Can Do Today? 2021. “Local Notifications.” What Web Can Do Today. Accessed October 11. <https://whatwebcando.today/local-notifications.html>.
- World Wide Web Consortium. 2021. “Media Queries.” Accessed November 18. <https://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619/>.

A Architect interview questions

1. What are the architecturally significant requirements for the mobile application?
2. What are the quality requirements for the mobile application (performance, etc.)?
3. How will the architecture of the mobile application be different from the architecture of the web application?
4. What systems will the mobile application communicate with?
5. How will the architecture for the web and mobile application evolve?
6. How will the mobile application be deployed/installed?
7. What are the requirements regarding the deployment process of the mobile application?
8. What are the changes that need to be done so that the architecture supports the mobile application with all its planned features?
9. What planned features for the mobile application are potentially CPU-intensive (calculating statistics, etc.)?

B Director of product development interview questions

1. What are the reasons for creating the mobile application?
2. What are the reasons for creating the web application?
3. What is the planned launch date for the mobile application?
4. What is the planned life-time for the mobile application?
5. What resources are planned to be allocated to the mobile application?
6. What resources are planned to be allocated to the web application?
7. What competences does the company have in regards to software development?
8. How much resources is the company ready to spend on educating its employees that will work on the web and mobile application?
9. What are the requirements concerning the technology used in the mobile application?
10. What are the requirements related to testing of the mobile application?
11. How are the employees of the company taken into account during planning and designing of the mobile application?

C Lead UI/UX designer interview questions

1. What are the reasons for creating the mobile application?
2. What are the reasons for creating the web application?
3. What are the requirements regarding the user experience (UX) for the mobile application?
4. What are the requirements regarding the user interface (UI) for the mobile application?
5. Which of the planned features for the mobile application are potentially graphically demanding (for instance, animations and videos)?
6. What are the possible benefits the mobile application will have over the web application (for instance, using the mobile application in offline mode)?

D Product owner interview questions

1. What are the reasons for creating the mobile application?
2. What are the reasons for creating the web application?
3. What is the planned launch date for the mobile application?
4. What is the planned life-time for the mobile application?
5. What are the planned features for the mobile application for launch?
6. What are the planned features for the mobile application after launch?
7. How will the mobile application differ feature-wise from the web application?
8. What are the quality requirements for the mobile application (performance, etc.)?
9. What are the possible benefits the mobile application will have over the web application (for instance, using the mobile application in offline mode)?
10. How business critical will the mobile application be for the clients (for instance, clients' reactions to downtime)?
11. What devices are the target end users expected to use the mobile application with (device platform, type, size, age, and performance)?
12. How will accessibility be considered in the mobile application (for instance, screen readers)?

E Sales director interview questions

1. What are the reasons for creating the mobile application?
2. What are the reasons for creating the web application?
3. How will the mobile application be marketed?
4. What is expected to be the importance of the mobile application in marketing and sales when comparing to the other products by the company?
5. Who is the target audience for the mobile application?
6. Who is the target audience for the web application?
7. What devices are the target end users expected to use the mobile application with (device platform, type, size, age, and performance)?
8. What values should the mobile application reflect to the customers?
9. What values should the web application reflect to the customers?
10. What values do the existing applications by the company reflect to the customers?
11. How many customers are expected to deploy the mobile application?
12. How many end users are expected to be using the mobile application?