

Aalto University  
School of Science  
Master's Programme in Information Networks

Harri Sarsa

# **Critical Requirements of Internal Enterprise Mobile Applications**

Master's Thesis  
Helsinki, November 27, 2017

Supervisor: Casper Lassenius, Professor  
Thesis advisor: Matti Paksula, M.Sc.

<b>Author</b>	Harri Sarsa	
<b>Title of Thesis</b>	Critical Requirements of Internal Enterprise Mobile Apps	
<b>Master's Programme</b>	Information Networks	
<b>Major / Code</b>	New Media / T310-3	
<b>Department</b>	School of Science	
<b>Supervisor</b>	Casper Lassenius, Professor	
<b>Thesis Advisor</b>	Matti Paksula (M.Sc.)	
<b>Date</b>	<b>Number of pages</b>	<b>Language</b>
27.11.2017	79 + 15	English

**Abstract**

This thesis constructs a framework for evaluating the functional requirements of internal enterprise mobile apps, called Sanity Checklist. Due to the multitude of different functional requirements that an internal enterprise mobile app might have and the difficulty of the requirements engineering process, it is hard for enterprises to make informed decisions on what development approaches and technologies would help them get on top of their app backlog (i.e. list of apps to be made).

In a literature review, we uncover internal enterprise mobile app requirements that have been identified by the academia. This data is then used as a basis for an interview with industry experts on the requirements of the internal mobile apps in their enterprises. Based on our methodology results, we construct a list of questions called the Sanity Checklist. Our hypothesis is that by answering these questions in the context of their planned app backlog, enterprise IT decision makers will be better equipped to understand if a certain development approach, technology or platform will be a fitting choice for their organization's needs.

We evaluate the Sanity Checklist against two already-completed internal enterprise mobile app projects, to see if applying the framework before the project was started would have provided value. Our observations indicate that the Sanity Checklist is a useful tool that will be valuable as part of a holistic requirements gathering process, especially in order to ensure the chosen development approach and toolset can tackle all the nuances of the app's requirements.

---

<b>Keywords</b>	enterprise, digitalization, internal mobile applications, mobile application development platforms, requirements engineering, low-code, visual application development
-----------------	--

---

<b>Tekijä</b>	Harri Sarsa	
<b>Työn nimi</b>	Critical Requirements in Internal Enterprise Mobile Apps	
<b>Koulutusohjelma</b>	Informaatioverkostot	
<b>Pääaine / koodi</b>	New Media / T310-3	
<b>Valvoja</b>	Casper Lassenius, professori	
<b>Työn ohjaaja</b>	Matti Paksula (M.Sc.)	
<b>Päivämäärä</b>	<b>Sivumäärä</b>	<b>Kieli</b>
27.11.2017	79 + 15	englanti

### Tiivistelmä

Tässä diplomityössä luodaan Sanity Checklist -niminen viitekehys, jolla yritysten sisäisten mobiilisovellusten toiminnallisia vaatimuksia voidaan arvioida. Koska yritysten sisäisillä mobiilisovelluksilla on merkittävän moninaisia toiminnallisia vaatimuksia, ja koska ylipäätään ohjelmistojen vaatimusmäärittely on haastavaa, yritysten on vaikea tehdä valistuneita päätöksiä sen suhteen, mitkä mobiilisovelluskehityksen lähestymistavat ja teknologiat auttaisivat heitä pääsemään niskan päälle sovellustyölistojensa (eng. mobile app backlog, eli mitä mobiilisovelluksia yrityksellä on suunnitteilla) kanssa.

Kirjallisuuskatsausosiossa listaamme, mitä yritysten sisäisten mobiilisovellusten vaatimuksia on akateemisessa kirjallisuudessa tunnistettu. Tätä tietoa käytetään asiantuntijahaastatteluiden pohjana, joissa selvitetään edelleen, mitä vaatimuksia haastateltujen organisaatioissa toteutettuihin sisäisiin mobiilisovelluksiin on kohdistunut. Metodologiamme tulosten pohjalta rakennamme Sanity Checklist -nimisen listan kysymyksiä. Hypotesimme on, että vastaamalla näihin kysymyksiin yrityksen suunnitteilla olevien sovellusten kontekstissa, IT-päätäjillä on paremmat valmiudet ymmärtää, onko jokin tietty sovelluskehityksen lähestymistapa, teknologia tai alusta sopiva valinta heidän organisaationsa tarpeisiin.

Lopuksi sovellamme Sanity Checklist -viitekehystä kahteen yrityksen sisäiseen mobiilisovellusprojektiin, tarkoituksenamme selvittää, olisiko viitekehityksen hyödyntämisestä ennen projektien alkamista ollut hyötyä. Havaintomme osoittavat, että Sanity Checklist on hyödyllinen työkalu osana holistisista vaatimusmäärittelyprosessia, etenkin varmistettaessa, että valitut kehityksen lähestymistapa ja työkalut todella onnistuvat ratkomaan kaikki sovelluksen vaatimusten nyanssit.

<b>Avainsanat</b>	yritykset, digitalisaatio, sisäiset mobiilisovellukset, mobiilisovelluskehitysalustat, vaatimusmäärittely, low-code, visuaalinen sovelluskehitys
-------------------	--

<b>1</b>	<b>INTRODUCTION</b> .....	<b>6</b>
1.1	The Ubiquitous Smartphone.....	6
1.2	Potential of the Internal Mobile App Revolution in Enterprise.....	7
1.3	Motivation for the Work.....	8
1.4	Scope of the Work.....	12
1.5	Structure.....	16
<b>2</b>	<b>BACKGROUND</b> .....	<b>18</b>
2.1	Different Development Approaches.....	18
<b>3</b>	<b>METHOD</b> .....	<b>24</b>
3.1	Literature Review.....	24
3.2	Qualitative Interviews.....	24
3.3	Results Analysis.....	26
<b>4</b>	<b>MAPPING OUT THE INTERNAL ENTERPRISE MOBILE APP REQUIREMENTS</b> .....	<b>27</b>
4.1	App Types.....	27
4.2	Platform, Device Model and Operating System Version Fragmentation.....	30
4.3	Battery Power Consumption.....	32
4.4	User Interface and Logic.....	33
4.5	Screen Size and Resolution.....	36
4.6	Interacting with Data.....	38
4.7	Native Device Capabilities.....	42
4.8	User Authentication.....	43
4.9	Offline Data.....	46
<b>5</b>	<b>METHODOLOGY RESULTS</b> .....	<b>49</b>
5.1	Overview of the App Projects and Mobile App Strategy.....	49
5.2	Technologies Used in Projects.....	52
5.3	Team Composition.....	53
5.4	Project Timeframes.....	54
5.5	Project Requirements Gathering.....	55
5.6	Functional Requirements.....	58
5.7	Blockers to More Rapid App Development.....	62

5.8	Observations on the Projects' App Requirements.....	63
<b>6</b>	<b>CONSTRUCTING THE FRAMEWORK.....</b>	<b>67</b>
6.1	Properties of Internal Enterprise Mobile App Requirements .....	67
6.2	The Sanity Checklist.....	68
<b>7</b>	<b>VALIDATION OF THE FRAMEWORK.....</b>	<b>70</b>
7.1	App Business Cases.....	70
7.2	Development Approach and Technology Used .....	71
7.3	Evolution of the Requirements and Difficulties Faced .....	71
7.4	Applying the Sanity Checklist .....	74
7.5	Observations on the Sanity Checklist.....	75
<b>8</b>	<b>DISCUSSION AND CONCLUSIONS.....</b>	<b>79</b>
8.1	Avenues for Future Research .....	79

# 1 Introduction

The digital transformation or digitalization of an enterprise can have tremendous positive business impact; at the same time, it is very challenging to execute successfully (Andal-Ancion, Cartwright and Yip, 2003; Zhu *et al.*, 2006; R. Agarwal *et al.*, 2010; Matt, Hess and Benlian, 2015).

Gartner (2017a) defines **digitalization** as

“[...] the use of digital technologies to change a business model and provide new revenue and value-producing opportunities; it is the process of moving to a digital business.”

For a digital technology to be useful for an organization, it must offer some new affordances and improvements over the previous way of working; otherwise there's no business driver for the change. Getting value from a new system also doesn't happen overnight: there's a process of information technology adoption that needs to be gone through (Karahanna *et al.*, 1999). Indeed, successful, impactful and increasingly rapid digitalization is becoming a practical necessity for companies to remain competitive and relevant (Plummer and Hill, 2014).

There are many technologies that can aid an enterprise in its digitalization efforts. In this work, we will focus on one set of technologies that has seen rapid proliferation over the past decade and whose impact on enterprise digitalization cannot be ignored: smartphones and the mobile apps running on them.

## 1.1 The Ubiquitous Smartphone

After the introduction of the Apple iPhone in 2007, global annual smartphone sales have increased over 1222%, from 122.32 million units in 2007 to 1495.36 million units in 2016 (Gartner, 2017d). The trend continues in 2017: global smartphone sales in the first quarter of 2017 increased 9.1 percent over the first quarter of 2016, up to 380 million handsets (Gartner, 2017b). Combined with the increased adoption of policies like Bring-Your-Own-Device or BYOD (Sobers, 2014), in most companies it is more than likely that an employee will have with them a smartphone that can be used to carry out work-related tasks throughout the day, regardless of the physical location of the employee.

The physical device itself is hardly enough to enhance business processes in meaningful ways. An employee needs to have the right kind of software running on her smartphone. In modern smartphones, this usually takes the form of mobile apps. Furthermore, and especially in a work

context, the value-producing apps are likely ones that do not come preinstalled with the mobile operating system. Email access from the field only goes so far.

“There’s an app for that” was a popular catchphrase in Apple’s 2009 commercials that the Cupertino company even went on to trademark (Chen, 2010). It is today truer than ever, as the three largest app stores in the world – Apple App Store, Google Play and Amazon AppStore – hold almost 6 million apps combined (Dogtiev, 2017), with many more distributed outside the public app stores.

The main brunt of innovation in the mobile app space has happened in the consumer space, and one will be hard pressed to find a large company without one or more consumer-facing apps. For example, a quick Google search reveals that every single one of the top 20 Fortune 500 companies (Fortune, 2017) has multiple apps on both the Apple App Store and Google Play<sup>1</sup>.

## 1.2 Potential of the Internal Mobile App Revolution in Enterprise

In contrast to the consumer-facing apps, enterprises are digitalizing their core processes by building **internal mobile apps**. This category is sometimes called Business-to-Employee (B2E) apps (Leow and Baker, 2017), even if the users/stakeholders are not necessarily legal employees of the company providing the app, but also e.g. business partners, suppliers, students, resellers, contractors and so on. It refers to apps whose main purpose is to provide a new or more efficient way to tackle a business problem, e.g. collecting and collating data more effectively by digitalizing a paper form, facilitating a workflow or providing mobile, real-time access to information such as sales figures or equipment locations.

The potential transformative impact of internal enterprise mobile apps has been amply covered in literature (Sørensen *et al.*, 2008; Unhelkar and Murugesan, 2010; Hoos *et al.*, 2014; Chung, Lee and Choi, 2015), but it seems that most companies are still building and deploying very few of them.

A Gartner study (Leow and Baker, 2017) found that the companies surveyed – over half of which were 2,500+ employees, \$500+ million revenue enterprises – had developed on average 8 mobile apps, with 2.6 in development and 6.2 planned for the next 12 months but not yet in development. 27% of surveyed companies had not built any mobile apps in the previous 12 months. Spending on mobile apps had been stagnant and even decreased between fiscal year 2015 and 2016.

---

<sup>1</sup> Google Search performed 19<sup>th</sup> October 2017, using a mix of search terms: company name or company brand name, “app”, “itunes”, “google play”, “app store”, “ios” and “android”.

Leow and Baker (2017) further note that when the low number of apps produced is combined with the fact that employees are increasingly choosing the devices, software and processes by which they complete their tasks, there is huge pressure in IT to deliver a larger variety of mobile apps in a short timeframe. Else, they face very fragmented and hard-to-manage IT structures, which in turn will significantly hinder company efficiency. Other Gartner reports have the same tone, with Leow (2017b) noting that there is a definite gap between mobile app demand and supply, and that enterprises need to adopt tools that produce apps more rapidly to bridge it.

It should be noted here that the vast majority of extant academic literature covering enterprise mobile apps is at least several years old. At the same time, the speed of change is increasingly fast in both the mobile app technologies themselves, as well as the development tools and methodologies used to create them (Leow, 2017a).

To provide a current viewpoint, we supplement our literature findings throughout the work with an expert interview with Matti Paksula (2017), CTO of mobile app development platform company AppGyver.

Paksula (2017) echoes the tones of the Gartner analysts:

“We see enterprises regularly describe tens of processes that could be digitalized with a purpose-built internal mobile apps. For larger companies, the number of potential internal apps on the backlog can be over a hundred. When you count in all the app ideas that haven’t been identified yet, it’s clear that there’s huge untapped potential. Organizations are generally struggling to build even the few mobile apps they’ve selected as champion projects.”

From all this, we can conclude that enterprises would benefit from a greater number of internal mobile apps, but they are not producing as many of them as they should. How to tackle this, if simply hiring more and more engineers is not a viable option due to the cost and difficulty of finding good talent?

### **1.3 Motivation for the Work**

As Brooks (1987) famously states, there is no single “silver bullet” solution or approach to software development that would by itself produce even an order of magnitude improvement in delivery speed. This is mainly due to the fact that a large part of the difficulties of building software stem from factors that are inherent in the essence of software, and thus cannot easily be solved. One

fundamental fact is that a piece software must do everything required of it, and all the necessary functionalities have to be somehow implemented.

According to Paksula (2017), this design complexity is comparatively high in the context of a modern enterprise mobile app, due to a multitude of requirements that even a simple app must fulfil:

“First of all, the app needs to solve the unique business problem at hand, so its design must be sound – and this can be a kind of low complexity thing. For example, consider an expense reporting app: you essentially need to take pictures of receipts, input some additional data and send the expense for approval. Not very complex, even though there are obviously nuances and additional requirements behind that simple user story.

However, on top of that, the app must also deliver many baseline requirements. It must work on all required mobile operating systems, screen sizes and device types. It must present a well-designed, performant UI for the task at hand. Corporate branding would be nice. It must allow employees to authenticate against a user database via the company’s preferred sign-on method. It must allow business data, stored in a backend, to be viewed and interacted with: in this case, submitted expenses and their statuses. Offline support, language localizations, push notifications and so on might be required. And this is just in the app itself; layers and layers of complexity arise from considerations like data integrations, backend services, IT governance, deployment pipelines, lifecycle management, testing, logging, analytics and so forth.”

Understanding what the specific requirements of a specific mobile app are is not an easy task: the complexity of the software requirements engineering process is enough to fill books (Leffingwell and Widrig, 2000; Pohl, 2010), and the academic literature on the subject agrees that it is a very difficult problem both overall (Nuseibeh and Easterbrook, 2000; Achimugu *et al.*, 2014; Inayat *et al.*, 2015) and in the context of enterprise applications (Shen *et al.*, 2004; Niu *et al.*, 2014).

Gartner (Baker, 2016) also notes that feature requirements of enterprise apps evolve over time, mentioning e.g. that “many enterprises do not initially believe that that they need offline capability”, implying that this “hidden” feature requirement can come up at some point after full steam mobile digitalization has begun. This is logical, as enterprises cannot possibly map out beforehand all the requirements of all of the apps they are ever going to build. As just one example, new technologies constantly emerge while old ones mature, and this evolution needs to be taken into account (Leow, 2017a).

Given how the requirements of internal enterprise mobile apps are clearly myriad and difficult to articulate fully beforehand, the image begins to emerge why companies are struggling to increase the speed at which they develop and deliver internal mobile apps. Even if the requirements engineering process is successful and complete, the resulting specification must be implemented as a functional mobile app – and software development is anything but trivial (Brooks, 1987), especially in the context of mobile apps (Sørensen *et al.*, 2008; Joorabchi, Mesbah and Kruchten, 2013).

### 1.3.1 Implementing the Requirements

However, the multitude of requirements can be a blessing in disguise, as each requirement is one dimension along which to improve the velocity of the development effort. We do not have to significantly better all aspects of enterprise app development to reach notable speed gains. Making it faster to implement even a part of these requirements could very well be enough to enable enterprises to get on top of their mobile digitalization efforts. Perhaps a silver bullet can be constructed, after all, by combining many small victories.

Recent years have seen many new technological solutions and development approaches that make many these requirements easier to handle: for example, numerous cross-platform technologies seek to make it more straightforward to develop apps that run on multiple mobile platforms, by removing the requirement to maintain separate codebases for each target mobile platform (El-Kassas *et al.*, 2017).

In a similar fashion, full-fledged Mobile App Development Platforms (MADPs) tackle many common app requirements like user authentication and data integrations in an implement-once, use-anywhere fashion, while also addressing the cross-platform issue (Wong *et al.*, 2017). Furthermore, numerous low-code development tools provide highly visual ways of crafting the mobile app, with little or no coding required (Rymer, 2017; Vincent *et al.*, 2017).

The different development approaches have benefits and drawbacks in relation to their ability to fulfil the critical requirements, including the development speed at which various features can be implemented (Joorabchi, Mesbah and Kruchten, 2013; El-Kassas *et al.*, 2017).

### 1.3.2 Different Development Approaches

Developing everything by hand naturally gives the highest level of control over the end product, as everything is fully customizable, at least within the limitations of the mobile operating systems and hardware targeted. However, this comes at the expense of development speed; creating things from

scratch is slow and error-prone, and e.g. separate codebases will need to be maintained for each target mobile platform (Joorabchi, Mesbah and Kruchten, 2013).

Conversely, a MADP solution could make it possible to e.g. simply switch on offline data support for an app, without any extra coding needed (Wong *et al.*, 2017). Given the complexity of offline data, such as gracefully handling changes in network status and resolving any data synchronization conflicts that might emerge (Giguère, 2001), not having to code your own solution would provide significant time savings, especially when the feature needs to be implemented into tens of different apps.

However, each development approach also has inherent limitations, and the development speed gains could very well be outweighed by the potential compromises and issues raised by these limitations. If the critical functional requirements of an app cannot be met, or the apps produced are e.g. very slow performance-wise or crash constantly, the underlying business process might end up being hindered instead of improved.

The negative impact of a chosen development approach can be an especially hard-hitting if the cause of the problems is in closed-source components. As an example, most MADPs are at least partly closed-source (Rymer, 2017; Vincent *et al.*, 2017; Wong *et al.*, 2017). At worst, this can mean that the developer has no way of overcoming the limitations and improving the situation, apart from switching to a completely different development approach.

In addition, many development approaches, especially MADPs, provide more benefits as the number of apps built and deployed with the platform increases (Leow, 2017b). A custom authentication solution might be as difficult to implement in a MADP as with a hand-crafted approach when doing it for the first time, but all future apps developed with the same MADP will benefit from the initial work.

This means that as more apps are developed and deployed with the same development approach, it will be increasingly painful and costly to switch to another approach or toolset, if a critical limitation inherent to the approach and toolset emerges.

### 1.3.3 Key Problem

From the above, we can articulate our key problem:

**It is not easy to understand what features and functionalities will be required by internal enterprise mobile apps, especially before a development approach is selected and any problems become apparent.**

To address this, we will be constructing a framework for IT decision makers to better understand the feature requirements of internal enterprise mobile apps. At the time of writing, no such framework was found to exist.

After analysing their app backlog via this framework, IT decision makers will then be better equipped to evaluate how different development approaches and toolset vendors could help their organization deliver more internal mobile apps.

## **1.4 Scope of the Work**

In this work, we will scope our examination to the requirements of the mobile client app; in other words, we will seek to answer the question: “What must the client-side code of the app installed on the user’s mobile device be able to do?”

The mobile app is the first and often only touchpoint the end user has to the whole digital solution used to enhance the underlying business process. Thus, it makes sense to first understand the requirements of the client mobile app in more detail, before considering any other aspects of the whole system. If the app itself doesn’t do what is required, it doesn’t matter if the backend data integrations are efficiently laid out and have great caching logic, or if user permissions can be managed with an intuitive interface, or if push notifications can be easily triggered based on system events.

For some digital solutions, the mobile app is of course just an additional touchpoint into the whole system. That said, the unique nature of the mobile app – its inherent mobility, i.e. the property of being always available regardless of employee location (Hoos *et al.*, 2014) – merits focusing just on solutions where the mobile app plays the key role.

Furthermore, we want to focus on the functional requirements of the app, instead of the development and managerial affordances provided by a development approach or toolset. It would certainly be interesting to consider how some approach makes it easier or more difficult for distributed teams to reuse created code, or how a platform makes it more efficient to deploy and test the apps developed

with it. However, these considerations would again be worthless if the mobile app produced cannot be used for its intended purpose.

Finally, extant literature that covers mobile apps in general can be applied within this scope, giving us a strong theoretical basis, whereas very few works consider the enterprise mobile app ecosystem as a whole. Also, answering even focused questions like “how should my mobile app backend be deployed” would require a huge additional literature review, and likely a different empirical approach. Limiting our scope in this way will produce a more focused work.

Next, we will go through some other limitations on the scope of the work that we impose in order to keep our research to the point. We will also dismiss several other features and affordances that a development approach might provide that are not critical to fulfilling the app use case itself, but rather help the holistic process of developing, testing, deploying, maintaining and iterating the app.

#### **1.4.1 Stable and Bug-Free Functionality**

We will assume that the features discussed are implemented without bugs that cause the app to crash or otherwise work in unintended ways. In real life, this might not be the case, and thus the overall stability and bugginess of both the platform and the apps produced should be considered and evaluated.

#### **1.4.2 Target Device and Operating System**

We will limit ourselves to just apps running on Apple iOS and Google Android mobile phones and tablets, as they make up 99,8% of mobile devices currently sold worldwide (Gartner, 2017c).

To keep our work focused, we will not consider desktop native apps, desktop web apps, large display apps, apps built for wearable devices, augmented/virtual reality apps, virtual assistants/chatbots, or other such mediums, as they would all expand the possible functionalities and thus requirements of the apps running on them, diluting our focus.

#### **1.4.3 App Type**

We will consider all iOS and Android mobile apps that are distributed as a native binary, i.e. an .ipa or .apk file, regardless of the underlying technology used to produce the app.

We will not consider web-based mobile apps in this work, i.e. mobile-optimized websites that are accessed via the browser, without a native binary container. Web-based apps have different

capabilities compared to mobile apps. For example, they cannot utilize many native device APIs, but can be accessed more easily, as they are available via a web browser, without the user having to download a native binary (El-Kassas *et al.*, 2017). These differences mean that the potential requirements of web apps would be significantly different from native mobile apps, causing a loss of focus for our work.

#### 1.4.4 App Stakeholders

We will only examine the feature requirements of **internal enterprise mobile apps**, as defined in Chapter 1.

This is due to the unique opportunity that they provide to increase the efficiency of existing business processes in a measurably way. Also important is the fact that a typical organization is likely to produce a significant number of them, making it more important to understand their common requirements before committing to a development approach.

We will leave out **Business-to-Consumer (B2C) mobile apps**, i.e. apps that are available for download via the public app stores and whose users are consumers of the company's products/services. While they can be used to improve existing business processes, they also have value in building brand identity and luring in new customers, and as such have divergent requirements.

Similarly, we will leave out **Business-to-Business (B2B) mobile apps** where the users of the app are not yet in a business relationship with the company providing the app (or one of its partners).

Compared to internal mobile apps, where users are working towards realizing the same common business goals, in both B2C and B2B apps users need to be "won over" in a much more pronounced way. In B2E apps, the users can of course solve the problem with some another way, but a B2B or B2C app user can discard the enterprise providing the app altogether and conduct his business with some other company.

This, again, would lead to different requirements and thus a less focused framework: for example, a B2B app seeking to capture new business would likely need some way for companies to register as app users, whereas most internal apps would utilize an existing corporate directory for user authentication.

#### **1.4.5 Backend Requirements**

We will not consider backend requirements of the mobile app, such as the server-side part of user authentication, permissions management, data integrations, and backend services such as sending push notifications, generating PDF reports or performing intensive calculations. This includes implications of the backend solution on app security.

#### **1.4.6 Lifecycle Management**

We will not consider how a platform solution might help app lifecycle management, e.g. moving the app from development to testing to staging to production, rolling out app updates, and managing different backend systems for each environment. These factors become important only after the app's functional requirements are fulfilled.

#### **1.4.7 Development Process Requirements**

We will not consider the potential requirements applied on the mobile app development process itself, such as the ability of visual tools to give non-code-savvy business users a hands-on role in the app development or the kind of tools are available for debugging the app as it is being developed. This includes implications on IT governance, such as the ability of IT administrators to limit developer access to certain production environments.

#### **1.4.8 Security**

We will not consider mobile application security beyond discussing user authentication and permission. That said, the potential security requirements are numerous, from the inherent vulnerability of over-the-air data connections to different encryption methods for data traffic to addressing stolen devices to encrypting app data stored on the device, and so on.

Covering mobile app security in sufficient detail would require a significant amount of additional literature review and likely a different empirical method. Also, security is a more ephemeral, app-wide quality: many functional requirements can be implemented in a secure or non-secure way. As such, understanding the security requirements of an internal enterprise mobile app is an additional track alongside the actual functional requirements.

### 1.4.9 Other Platform Features

We will not consider the following additional features and activities that a mobile app development platform might make it easier to implement or perform, as they are again ancillary to realizing the actual functional requirements.

- App-specific administrative duties, such as managing app users and permissions
- App distribution, including support for various Mobile Application Management/Mobile Device Management platforms
- App usage analytics, e.g. session data, screen views, event tracking
- Audit logs on user activity, i.e. who did what
- Crash logging
- Feedback collection
- Automated and manual testing
- Systems health monitoring

There are naturally more ways in which a MADP might assist the holistic development and management of an internal enterprise mobile app; the above list should not be considered complete.

## 1.5 Structure

In *Introduction*, we introduce the root problem of enterprises not increasing their speed of mobile app delivery, outline the potential cause in complex requirements and lack of a “silver bullet” solution to tackling these requirements, and present the motivation and scope for our work. The next chapter, *Background*, examines the different development approaches in more detail and expands on the motivation for the work.

The chapter *Methodology* presents how the study was constructed and conducted. In *Mapping Out the Internal Enterprise Mobile App Requirements*, we review the extant literature, uncover applicable internal enterprise mobile app requirements, and translate them into interview questions. In *Methodology Results*, we present the findings of our study and our observations on the results.

We then move on to *Constructing the Framework*, where we utilize our results to build our framework for better understanding internal enterprise mobile app requirements. This framework is then validated against real-life app projects in *Validation of the Framework*. Finally, in *Discussion and Conclusions*, we present our conclusions and discuss further avenues of research.



## 2 Background

Before continuing on to discuss the mobile app requirements in more detail, it will be beneficial for us to better understand the other side of the equation: how are the requirements then fulfilled, i.e. how are the apps developed in practice?

In this chapter, we will examine in more detail the different approaches available for developing enterprise mobile apps, and see what their main strengths and weaknesses are. This will allow us to uncover additional motivation for our work.

### 2.1 Different Development Approaches

Paksula (2017) categorizes three approaches to internal enterprise mobile application development: hand-crafted, and two types of mobile application development platform (MADP) approaches: high-code and low-code.

#### 2.1.1 Hand-Crafted

The hand-crafted method is sometimes called “traditional coding”, in comparison to modern mobile app development platforms (Leow, 2017b). According to Paksula (2017), the hand-crafted approach means:

“[...] creating your enterprise apps by picking and choosing from the plethora of programming languages, frameworks, libraries, SDKs, backend deployment environments and so forth available. The end result will be a mix of app-specific new code and app-specific configurations of existing services, frameworks, SDKs, libraries and other components, all of which can be either closed-source or open-source.

This approach gives the greatest freedom, as there’s the fewest possible limits on what technologies or approaches can be chosen to implement each requirement of the internal mobile app. At the same time, this approach places immense burden on the capacity and skill level of the development team to make the right choices.

If the team is skilled, the app product vision is clear and the requirements necessitate full control over all aspects of the app, this approach can be the best one to take. The app will be as good as the team can build it.

However, especially when multiple apps are developed with the hand-crafted approach, the maintenance overhead starts to build up. Even if the same libraries and SDKs are used, there will be different configurations, and ensuring each app uses e.g. the same versions of third-party packages requires a lot of manual work. If proper care is not taken when creating e.g. an authentication solution, the resulting code might not be feasible to reuse across different apps, much less maintain in the long run. Backend systems introduce a whole new layer of complexity. When an OS version or other critical component is updated, every app needs to be tested, fixed and re-deployed separately. This can be further complicated if the original development team is not present anymore to do this, but instead the codebase needs to be read and understood by a completely new set of developers, who then implement the necessary fixes.”

### 2.1.2 Mobile Application Development Platforms

It has been known for years that there is a need for an unified, platform approach to building enterprise mobile apps by both the academia (Unhelkar and Murugesan, 2010) and industry analysts: Gartner noted already 14 years ago that “mobile solutions not built on an extensible multichannel architecture will cost twice as much over their deployment lifetime” (Clark, 2003).

As a result of this, more and more companies are trying to create consolidated offerings for building enterprise mobile applications. The first Gartner Magic Quadrant for Mobile Enterprise Application Platforms report in 2008 included 11 vendors (Clark and King, 2008). By comparison, the 2017 Magic Quadrant for Mobile App Development Platforms report included 17 vendors (Wong *et al.*, 2017), and the 2016 Market Guide for Rapid Mobile App Development Tools included 34 vendors (Wong *et al.*, 2016).

There were no academic definitions or relevant research found on enterprise mobile application development platforms. As such, to get an idea of what an enterprise MADP should be able to do, we'll turn to Gartner (Wong *et al.*, 2017) and note a few key features:

- A MADP should provide tools, technologies, components and services that together constitute the critical elements of a platform for creating custom mobile apps (.ipa and .apk binary files for iOS and Android, respectively)
- A MADP must be able to address the requirements of diverse enterprise use cases, as well as connect to different enterprise systems that may run on-premises or in the cloud

- A MADP should be as broad as possible in addressing both frontend<sup>2</sup> and backend development needs
- A MADP serves to centralize the life cycle activities (such as design, develop, test, distribute, manage and analyze) for a portfolio of mobile apps running on a range of operating systems and devices

Gartner places further criteria for inclusion into the MADP category, but for our purposes, the above pickings function as a good baseline.

Paksula (2017) states:

“A MADP essentially makes it faster and easier to create and publish enterprise mobile apps by taking away a lot of the heavy lifting that would be required when building a hand-crafted solution. They allow many critical feature requirements – authentication, data integrations, offline support, creating UI for interacting with data – to be implemented by either simple configuration or with less coding.

A lot of the power is in component reuse, governance and lifecycle management, so while these tools definitely have an upfront cost in learning curve and license fees, the right platform will definitely pay back the investment when the organization’s app fleet grows to tens of apps.”

Even though we make a distinction between high-code and low-code MADP approaches, it should be noted that a single vendor can offer both solutions: for example, MADP vendor Mendix offers both a high-code SDK<sup>3</sup> and a low-code desktop IDE called Modeler<sup>4</sup>.

### 2.1.3 High-Code MADP Approach

We define the **high-code MADP approach** as one where the apps are essentially put together by coding. The development environment can be fully native or leverage a proprietary or third-party cross-platform framework. MADP-specific SDKs and libraries are used to access platform features from the app code, e.g. backend data connectivity or push notifications.

---

<sup>2</sup> I.e. having to deal with the app user interface and logic

<sup>3</sup> Mendix Platform SDK: <https://docs.mendix.com/apidocs-mxsdk/mxsdk>. Accessed: 1 November 2017.

<sup>4</sup> Mendix Desktop Modeler Overview: <https://docs.mendix.com/refguide/desktop-modeler-overview>. Accessed: 1 November 2017.

With the high-code approach, there's very few limitations to what can be accomplished, as the app source code can be edited directly. The required skillset for developers is still essentially as high as with a non-MADP, hand-coded approach, even if some advanced features like offline support are provided as packaged features. Productivity gains come mainly from re-using once-implemented features across apps: once user authentication is set up for one app, the platform makes it easy to re-use in all other projects.

#### 2.1.4 Low-Code MADP approach

The Gartner term for this category is Rapid Mobile Application Development or RMAD tools (Wong *et al.*, 2016), while Forrester talks about Low-Code Development Platforms (Rymer, 2017).

For our purposes, we define the **low-code MADP approach** as one where the main characteristic of the development is that a large chunk of the required features can be implemented without coding, e.g. via a visual builder or wizard-like configuration tools. Extension points that allow custom code to be written can still be available.

This approach seems to be effective. A 2017 survey by Forrester (Rymer, 2017) asked respondents if adopting a low-code MADP approach had addressed issues encountered when building apps with hand-crafted coding methods; 97,5% reported “significant” or “notable” improvements, with a single respondent reporting “marginal” improvements and no-one reporting no improvements.

It should be noted that some low-code/visual app development tools only generate e.g. frontend code or otherwise lack critical enterprise features, and thus should not be included in this category.

#### 2.1.5 Choosing the Best Approach

We can see that adopting a MADP solution could be highly beneficial for an enterprise's app development efforts. The speed gains from visual development and reusable features are clear, and the app lifecycle management and governance functionalities are likely to be invaluable once the app fleet grows to tens of apps.

At the same time, as we noted in Chapter 1, MADPs might have inherent limitations that only become apparent after the commitment has been made and thus cause trouble. Also, the strengths of a MADP toolset depend on environmental factors: for example, while the SAP Fiori<sup>5</sup> platform can be used with data sources other than SAP, it is clear that an enterprise will get the most benefits if the

---

<sup>5</sup> SAP Fiori. <https://www.sap.com/products/fiori.html>. Accessed on: 23 November 2017.

apps built revolve around SAP data and processes. Similarly, Microsoft PowerApps<sup>6</sup> is built to play best with data and services in Microsoft's Azure cloud.

Paksula (2017) states in his interview:

“Choosing the right mobile application development approach requires a very deep understanding of the requirements that each internal enterprise app has, the surrounding IT environment, and the challenges that a developer typically faces while implementing these solutions.

On paper, it seems great that a MADP provides features like “offline support” on a platform level, all thought out and ready to just plug in and use. However, there are numerous sub-requirements that go into “offline support”, and if the platform implementation doesn't match what is required, and if there are no extension points, the enterprise is in trouble.

Similarly, even if a visual UI builder can produce mostly satisfactory user experience, it can impose hard limitations on how far the user experience can be improved. Furthermore, even if something is technically possible, it could be that implementing the feature requires a significant amount of extra coding or hacky solutions. This can greatly undermine the benefits of the chosen MADP in comparison to a hand-crafted approach.

Thus, an IT decision maker should seek to understand both the feature requirements of the apps on the backlog as well as the strengths and weaknesses of different development approaches. This is very hard, and the difficulty of requirements gathering is something that plagues all software projects, not just enterprise mobile apps.

Regardless, this evaluation process is especially important when adopting a MADP, as there's high initial costs in getting the platform up and running, training the developers and implementing the basic shared functionalities like user authentication and key data integrations. Switching away from the wrong MADP choice after a few apps have been developed on it will be very costly.”

From this, we find additional motivation for our work. An enterprise would do well to adopt a MADP to tackle their internal mobile apps backlog, but making the wrong choice can be costly. Thus, understanding and articulating the functional requirements of the entire upcoming app backlog in as

---

<sup>6</sup> Microsoft PowerApps. <https://powerapps.microsoft.com/>. Accessed on: 23 November 2017.

much detail as possible will help the enterprise avoid vendor lock-in with a MADP solution that doesn't fit their needs and won't produce the apps they want.

## 3 Method

This chapter presents the methodology for our study and describes how it was conducted.

### 3.1 Literature Review

We first examine the extant literature on the common feature requirements and implementation challenges found in mobile apps. The literature study is conducted using Google Scholar, with a variety of keywords related to mobile apps and enterprise app development. Since many of the challenges are related to software development in general, specific keywords are used for things like “offline data synchronization”, “mobile cloud computing” and “enterprise user authentication”. Furthermore, as suggested by Webster and Watson (2002), we go backward and forward by both reviewing the citations in the articles found in the primary search to uncover relevant prior articles, as well as utilizing Google Scholar’s “Cited By” functionality to discover new works that reference key articles discovered in the primary search.

The findings are codified under nine main categories: App Type; Platform, Device Model and Operating System Version Fragmentation; Battery Power Consumption; User Interface and Logic; Interacting with Data; Native Device Capabilities; User Authentication and Offline Data.

This information is further supplanted by an expert interview with mobile app company AppGyver’s CTO Matti Paksula (2017), as well as analyst reports from Gartner and Forrester, in order to bring in a present-day viewpoint, as well as cover areas where extant academic research doesn’t exist.

### 3.2 Qualitative Interviews

From this data, an interview framework is constructed for a qualitative interview (Patton, 2005) with industry experts.

We start by asking some general questions about the role of internal mobile apps in the corporate strategy, past mobile app projects, the requirements gathering process and so on.

We then move on to the questions defined in Chapter 4, *Mapping Out the Internal Enterprise Mobile App Requirements*.

We conclude with some reflections on learnings from past projects, as well as seeking to understand what factors currently limit building more apps in general, as well as the development speed of individual apps.

The interview frame used can be found in Appendix A.

### 3.2.1 Interview Participant Demographics

We interviewed 4 experts from 4 different companies. Each interview session took on average around 60 minutes. We recorded audio in the interview sessions, which was then replayed to collect participant responses in text form for further analysis.

Table I presents the participants' role in the app projects, the company industry, and number of company employees.

The four interviewees were pre-selected as people who have been involved in building internal mobile apps for an enterprise.

<b>ID</b>	<b>Role in App Projects</b>	<b>Industry</b>	<b>No. Employees</b>
<b>App-Maintenance</b>	Integration architect, implementation coordinator	Maintenance	1,000-2,500
<b>MADP-Education</b>	Team lead for an IT team responsible for apps, integrations, identity access management and business intelligence	Education	500-1,000, (with 5,000-10,000 students)
<b>MADP-Manufacturing</b>	Technical platform architect, responsible also for internal mobile apps portfolio	Manufacturing	2,500-5,000
<b>App-Engineering</b>	Line of Business representative in app project	Engineering and Service	50,000-100,000

Table I: Interview Participants.

### 3.3 Results Analysis

To analyze the interview results, we code the qualitative answers under the themes established in the interview frame and examine them for additional insights. We seek to uncover common themes, shared feature requirements and differences between the participants' app projects. In addition to the functional requirements themselves, the requirements gathering and definition process is examined.

The results of the literature review and interview data are then analyzed and combined to form a framework that will allow enterprise IT decisions makers to better understand what are the real and holistic requirements of the internal app projects planned or under development.

We then validate our framework by applying it to two existing app projects: what benefits or drawbacks would there have been from applying this framework to the app projects before they were started?

## 4 Mapping Out the Internal Enterprise Mobile App Requirements

In this Chapter, we will identify a number of requirements and potential challenges for constructing the client-side functionalities of an internal enterprise mobile app, based on a review of extant literature, as well as an expert interview with mobile app development platform company CTO Matti Paksula (2017).

For each requirements category, we will define relevant interview questions.

### 4.1 App Types

There are many different categorizations of mobile app types in the literature, but one clear distinction is between native and hybrid apps (Heitkötter, Hanschke and Majchrzak, 2012; Dalmasso *et al.*, 2013; Joorabchi, Mesbah and Kruchten, 2013; Xanthopoulos and Xinogalos, 2013; El-Kassas *et al.*, 2017).

#### 4.1.1 Basic App Types

We provide the following definitions:

**Native mobile apps** are coded in the platform's programming language of choice: for example, Objective-C or Swift for iOS and Java for Android. Native apps have access to the full range of APIs provided by the OS vendors, and thus are generally superior in user experience and speed compared to web-based and hybrid apps (Palmieri, et al., 2012). They can also be distributed via the Apple App Store, Google Play or the various internal enterprise app delivery mechanisms available. Their main disadvantage is that a developer seeking to target both platforms must know both programming languages, as well as maintain two separate code bases for the project (Joorabchi, Mesbah and Kruchten, 2013). Also, developers should know the native SDKs, platform standards and best practices of all platforms targeted (Holzinger, Treitler and Slany, 2012).

**Hybrid mobile apps** implement an embedded web browser to present a HTML5-based web app within a native binary container. The native container allows the app to be distributed via the same channels as a native app, as well as utilize native device capabilities beyond what would be possible in a web-based app (Charland and LeRoux, 2011). This enables the same code to be used for all targeted platforms, with only minimal modifications needed for different targets, but comes at the potential cost of decreased performance and inferior user experience (Joorabchi, Mesbah and

Kruchten, 2013). Also, new native capabilities require OS-specific code that bridges the native API over to the HTML5 side, so writing native code cannot necessarily be completely avoided.

#### 4.1.2 Additional App Types

Xanthopoulos and Xinogalos (2013) additionally define the categories of interpreted and generated app.

**Interpreted mobile apps** allow developers to write their cross-platform app using a single programming language such as HTML5 (Facebook's React Native<sup>7</sup>, Progress's NativeScript<sup>8</sup>, Axway's Appcelerator<sup>9</sup>), C# (Microsoft Xamarin<sup>10</sup>), Ruby (RubyMotion<sup>11</sup>) or others. The single codebase is then transpiled into a fully native app for the supported target platforms (Xanthopoulos and Xinogalos, 2013), or in the case of some newer frameworks like React Native, evaluated during runtime (Occhino, 2015). This approach gives the benefit of having a common codebase for the app without compromises in app quality. However, the available APIs and UI components are still limited only to what is provided by the framework used, with some platforms providing a way for developers to extend the functionalities via native coding.

**Generated mobile apps** produce native code based on some formal definition of the application; Xanthopoulos and Xinogalos (2013) give the example of Applause, which allows apps to be defined in a domain-specific language (DSL). The app definition is then used to automatically generate native code for multiple platforms. While Applause has been abandoned for years<sup>12</sup>, the apps produced by visual low-code tools could be seen to fall into this category, with the distinction that some MADPs generate hybrid apps, and often the app code cannot be exported for manual editing.

There are other, more granular categorizations found in literature (El-Kassas *et al.*, 2017), but we will limit ourselves to these four app types.

#### 4.1.3 Choosing the Right App Type

The literature agrees that a general recommendation on which approach is best cannot be made: native apps generally have superior performance and UI that matches both user expectations and

---

<sup>7</sup> React Native. <https://facebook.github.io/react-native/>. Accessed on: 13 November 2017.

<sup>8</sup> NativeScript. <https://www.nativescript.org/>. Accessed on: 13 November 2017.

<sup>9</sup> Appcelerator. <https://www.appcelerator.com/>. Accessed on: 13 November 2017.

<sup>10</sup> Xamarin. <https://www.xamarin.com/>. Accessed on: 13 November 2017.

<sup>11</sup> Ruby Motion. <http://www.rubymotion.com/>. Accessed on: 13 November 2017.

<sup>12</sup> Applause GitHub repository. <https://github.com/applause/applause>. Accessed on: 26 October 2017.

platform guidelines exactly, but they are also harder to build (Charland and LeRoux, 2011; Heitkötter, Hanschke and Majchrzak, 2012; Holzinger, Treitler and Slany, 2012; Xanthopoulos and Xinogalos, 2013; El-Kassas *et al.*, 2017).

Gartner places cross-platform frameworks in the “Peak of Inflated Expectations” segment in their 2017 Hype Cycle for Mobile Applications and Development report, noting that day-one support for new vendor SDK features is often not available, and many frameworks rely very heavily on their user communities to build support for new features (Leow, 2017a).

Paksula (2017) states in his interview:

“If you have great native coders and you are building a hero app that has to be absolutely top notch in all regards, then native coding is probably the way to go. Otherwise, native coding is just slow and difficult, and talent is very hard to come by. Maintaining different codebases for iOS and Android will definitely slow down development speed, even if the MADP provides native SDKs for common platform features.

Most MADPs that provide interpreted or generated native apps do get rid of the two-codebase handicap, but the native feature set of the end apps is limited to only those APIs, UI components and other functionalities that have been made available by the toolset vendor. Extendibility via custom native code is often possible, though.

In the interpreted native app category, React Native stands out from the crowd by being coded with JavaScript, which is something way more people know how to write adequately than e.g. C# or Ruby, and being based on React, which is a modern, popular web framework by Facebook. However, it is just a framework, not a platform: it needs to be manually combined with a high-code MADP approach to be useful for wide-scale enterprise development.

Hybrid apps are probably adequate for most enterprise app needs. Additional native capabilities can be usually added via plugins or other custom code if required, and HTML5 runs quite fast on modern phones. However, especially low-end Android devices can have unbearable performance issues.

Generated apps are as good as their generators, and should be evaluated mostly in terms of the parent app type, i.e. hybrid or native. That said, they might suffer from additional performance issues caused by a bloated and otherwise suboptimal nature of the generated code.”

Thus, app type is not really a feature requirement but a consequence of the chosen development approach and toolset. As such, the suitability of different app types should be considered in terms of the benefits and compromises it brings about in relation to the feature requirements of the app. Similarly, it is important to understand what app types a MADP supports.

We define the following interview questions:

- **What app types have you developed?**
- **What influenced the decision?**

## **4.2 Platform, Device Model and Operating System Version Fragmentation**

Most apps need to run on multiple platforms, device models and operating system versions (Unhelkar and Murugesan, 2010; Charland and LeRoux, 2011; Heitkötter, Hanschke and Majchrzak, 2012; Joorabchi, Mesbah and Kruchten, 2013). The problem is twofold: an enterprise first needs to figure out which platforms, devices and OS versions it wants the app to support, and then ensure the app works on all of them.

### **4.2.1 Importance of Testing**

As each combination of platform, device model and operating system version can potentially produce its own quirks, an app should be tested on all possible combinations to be 100% sure that no issues abound - which is obviously a monumental task (Joorabchi, Mesbah and Kruchten, 2013).

As Paksula (2017) notes in his interview:

“Ensuring the app works on all devices it is deployed to is obviously important. If a critical app suddenly starts crashing in the middle of a fast-paced business process, the implications can be huge.

At the same time, very few companies do anything close to 100% device testing coverage for their apps, and that’s rarely a wise investment. Generally, a well-coded app will work on all similar devices. There are very few differences between an iPhone 6 Plus running iOS 11.0 and an iPhone 7 running iOS 11.3, apart of course from the bugs that were fixed in the newer iOS version.

Android is a bit more of a wild jungle, since vendors often tweak the core Android OS implementation in small ways. For example, some native SDKs might work differently on two

devices from different manufacturers, even if both are running the same Android OS version number.

Bugs can also affect entire device categories: a mobile device might have a 32-bit or a 64-bit processor architecture, which require different things in certain parts of the native code. Thus, it might be that a bug affects all 64-bit devices.

It's very rare to see a bug that really only occurs on a single device model, but we've seen that happen, too – so the only way to really know your app works 100% on a given device is to test it end-to-end in a real-life situation.”

#### **4.2.2 Platform Fragmentation**

Given the market share of different mobile operating systems, today the platform requirement virtually always means iOS and/or Android. For example, in Q1 2017, only 0.2% of all shipped mobile devices had some other operating system (Gartner, 2017c).

Similarly, for a Mobile Application Development Platform to be included in Gartner's Magic Quadrant for 2017 report, only iOS and Android support is required; other platforms are not even mentioned by name (Wong *et al.*, 2017).

#### **4.2.3 Device Model Fragmentation**

At the time of writing, there are 22 different Apple devices that currently support the latest iOS version (Apple, 2017g), while different Android devices in use around the world number at minimum in the thousands (Piejko, 2017), if not tens of thousands (OpenSignal, 2015).

#### **4.2.4 Operating System Version Fragmentation**

There are significant differences between mobile application operating systems versions. While Google and Apple strive to provide a similar experience across different version numbers, the reality is that OS version fragmentation needs to be taken into account.

For example, it wasn't until Android 5.0 that Google made it possible for the rendering engine running WebViews (the rendering component of HTML5-based hybrid apps; essentially a full-screen web browser) to be updated independent of the OS version (Google, 2017d). Before this, the WebView rendering engine version was tied to the Android OS version, meaning different versions would implement different APIs and thus render app content in wildly different ways.

Similarly, some features are simply not available on older OS versions. Apple opened up their Near-Field Communication (NFC) APIs only in iOS 11 (Apple, 2017e). While device support goes all the way back to iPhone 5S, this means that if an enterprise develops an iOS app that supports reading NFC tags, all devices running it need to be updated to iOS 11.

For iOS, new OS version adoption is very fast. On 30<sup>th</sup> October 2017, just 6 weeks after the iOS 11 release, Mixpanel (2017) reported that the adoption rate for iOS 11 was 58.31%, with only 9.77% of devices running an OS version older than iOS 10.

Compare this with Android: Google's (2017b) statistics on the devices that visited the Google Play app store in the 7-day period ending on October 2, 2017 show that 50% of devices run a version of Android that was released in 2015 or earlier; the Android OS 5.1 launch date was March 25<sup>th</sup>, 2015 (Google, 2015).

It should be noted that these statistics are collected from visits to the public Google Play store. This means a potentially significant number of devices in internal enterprise use that have their apps distributed solely via private channels would not show up here, as the users would not visit the Play Store on their devices.

We define the following interview questions:

- **What platforms, device models and mobile OS versions was the app required to run on?**
- **How were the device compatibility requirements decided?**
- **What kind of testing did you do to ensure the app works on all the targeted combinations?**

### **4.3 Battery Power Consumption**

Mobile devices rely on limited battery life for operation in the field. Indeed, mobile battery life and energy consumption of mobile apps is a potential issue brought up consistently in literature (Unhelkar and Murugesan, 2010; Wasserman, 2010; Dalmaso *et al.*, 2013; Joorabchi, Mesbah and Kruchten, 2013; Yadav and Kanchan, 2016).

In the case of MADPs, it is especially important to understand if the closed-source, unmodifiable platform features cause significant power consumption in apps produced by the platform, as optimizing these parts of the app might be impossible.

We define the following interview questions:

- Did you face any issues with high battery power consumption?
- How were these issues tackled?

## 4.4 User Interface and Logic

A major part of any mobile app is its user interface. This includes the contents of individual pages or views within the app, the navigation structure to move between them, and the user-facing logic, e.g. what happens when the user clicks on a button.

The goal of the app user interface designer is to create the best user interface (UI) for the problem at hand, so that using the app is as simple and intuitive as possible (Perchat, Desertot and Lecomte, 2013).

An internal app will work in the context of a specific business process: for example, submitting travel expenses or viewing sales leads on the go. In today's world, these business processes are very likely to be unique to the given organization and its business partner network, with factors like industry and the organization's existing technologies affecting the optimal solution (van der Aalst, 2013).

Kuusinen and Mikkonen (2014) note that the amount of UI and user experience (UX) design required also varies between different enterprise mobile apps. If the idea of the app is not novel, user involvement in the design process might not be needed, and the app can be built with standard UI elements. In other cases, a more heavyweight UX design process is warranted.

As such, it is beyond the scope of this work to provide a comprehensive list of requirements for the various UI components or functionalities that an app might have: not only would we need to research and categorize all different UI components and navigation structures, but also define requirements for what the application client logic needs to be able to do.

Instead, we will cover any applicable UI/logic requirements as they come up in each of the following categories. However, we will discuss three UI requirements common to many internal enterprise apps: performance, corporate branding and language localizations.

### 4.4.1 UI Performance

Native apps set the gold standard for mobile UI performance; the literature consistently considers hybrid mobile app UIs to be inferior (Heitkötter, Hanschke and Majchrzak, 2012; Holzinger, Treitler

and Slany, 2012; Palmieri, Singh and Cicchetti, 2012; Dalmasso *et al.*, 2013; Francese *et al.*, 2013; Joorabchi, Mesbah and Kruchten, 2013).

Similarly, Gartner mentions “high-fidelity UI with native look and feel” and “native app performance” as evaluation criteria in their Magic Quadrant for Mobile Application Development Platforms for 2017 report (Wong *et al.*, 2017).

However, it should be noted that all of the cited academic research is 4-5 years old. As Paksula (2017) notes:

“Hybrid app UIs have really taken a leap forward since the early days of PhoneGap and jQuery Mobile. Device processing power has increased, the HTML5 standard has matured, and modern mobile-first UI frameworks like Ionic or Angular Material have been developed. Hybrid apps are a significantly more viable option for creating beautiful and performant UIs today than even two years ago.

At the same time, hybrid app performance still lags behind that of native apps, especially on lower-end Android phones. Long lists, maps and video players are examples of UI elements where a hybrid app can perform very badly.

The amount of user interactivity is a crucial factor, though: if a map view is required to simply show the location of a static asset on a map alongside other data, there’s hardly a reason why a HTML5 component would not be enough. Conversely, if the app’s main user interaction happens via a map, a native solution will likely provide a much smoother user experience.

In any case, great care should be taken to ensure the UI is highly performant. People have become used to extremely well-crafted consumer mobile apps, and internal tools that do not match those expectations risk user abandonment.”

We define the following interview question:

- **What type of UI performance issues have you faced?**
- **How did you solve them?**

#### **4.4.2 Company Branding**

Company branding is generally considered to be important primarily in the consumer market, but Bendixen et al. (2004) demonstrate that a strong company brand can be a prime factor in the B2B

market, too. Furthermore, work by e.g. Burmann and Zeplin (2005; 2009) on internal brand management highlight the role of consistent and effective internal brand communications.

Paksula (2017) notes in his interview:

“In practice, it is often enough for companies to have their own logo in the app, plus a colour scheme that follows their general graphical guidelines. Very few companies have formal design guidelines for internal mobile apps that would explicitly determine e.g. allowed fonts, but this baseline customization is very important: it makes the app feel theirs.”

We define the following interview question:

- **In what ways was the app made look like your brand?**
- **Are there any specific graphical or design guidelines for internal mobile apps?**

#### 4.4.3 Internationalization and Localization

As enterprises today are more and more global, internal apps must also cater to different locales.

He *et al.* (2002) define internationalization and localization as follows:

- **Internationalization** is “the process of developing or re-engineering a program by separating the program into culturally dependent and independent elements so that it can be easily adapted, without engineering changes, to various other locales”.
- **Localization** is “the process of adapting a program version, which may be the original or internationalised, for one or more other locales”.

In other words, a properly internationalized program will make the localization efforts easier.

Translating the user interface to multiple languages is the most obvious aspect of localization, but it can involve adding or deleting culture- and locale-specific features, serving localized versions of assets like sounds and images, ensuring dates and times are shown in the correct format for the locale, and so on (He, Bustard and Liu, 2002).

Paksula (2017) notes in his interview:

“Comprehensive internationalization of an enterprise’s mobile app fleet can be a big effort, especially if it is done from scratch for each app. For most companies, language translations are enough, but the specific requirements should be well mapped out beforehand to ensure that the chosen MADP’s internationalization implementation covers current and future needs, and if there are missing features, that adding the required support will not be prohibitively complex and hard.”

We define the following interview questions:

- **Was the app localized for different languages/locales? Which ones?**
- **What aspects of the app were localized?**
- **How was the localization implemented?**

## **4.5 Screen Size and Resolution**

Mobile devices come with different physical screen sizes, aspect ratios and pixel resolutions (Holzinger, Treitler and Slany, 2012); and the user interface of apps needs to be adapted to fit these different form factors (Unhelkar and Murugesan, 2010).

### **4.5.1 Physical Screen Size**

Apple divides the device types an app can target strictly into two categories, iPhone (includes iPod Touch) and iPad; apps can even be explicitly configured to work only on either target device type (Apple, 2017d). This marks a clear separation into phone-size UIs and tablet-size UIs, with some features like Split Views (i.e. rendering two native view containers side-by-side) being only available for iPad apps (Apple, 2017c).

For Android, Google instead provides four rough categories for physical screen size, from small to extra-large (Google, 2017c) – exact categorizations would be practically impossible due to the tens of thousands of different Android device models in existence (OpenSignal, 2015).

### **4.5.2 Screen Resolution**

For iOS, there are 12 different screen resolutions, each of which can be viewed in either portrait or landscape orientation (Apple, 2017b).

Depending on the combination of physical screen size and pixel resolution, an iOS device can have a 2x or 3x pixel density: dimensions of UI elements are defined in points, with each point representing either 2 or 3 physical pixels (Apple, 2017f).

For Android, Google defines six categories of screen pixel density from ldpi to xxhdpi (Google, 2017b); the combination of screen size and pixel density maps to a specific range of pixel resolutions.

### 4.5.3 Handling Different Screen Sizes and Resolutions

Jain (2014) defines the goal of responsive design to be to “detect the visitor’s screen size and orientation and change the layout accordingly”. Jain is talking about web pages, but the same concept can be extended for mobile apps as well.

On both iOS and Android, phones and tablets have their different design guidelines and approaches (Apple, 2017b; Google, 2017c). Often, the best UI for a mobile app will require separate layouts to be built for phone and tablet size screens (Holzinger, Treitler and Slany, 2012). Nayebi *et al.* (2012) specifically state that the implications of different screen sizes, resolutions and device orientations on mobile usability would require further research.

Both iOS and Android provide built-in native features for handling the different screen sizes. For example, Android has configuration qualifiers to serve different layouts/UI assets based on device screen size (Google, 2017c), while iOS has concepts like TextKit and Auto Layout (Apple, 2017a). These obviously require platform-specific knowledge to leverage to their full effect.

Paksula (2017) states:

“In the native environments, a developer obviously has the finest level of control over how the UI looks and behaves on different screen sizes and device orientations, provided they know how the platform and available APIs work.

For interpreted apps, it really depends on the framework or MADP in question on what’s easily possible. React Native, for example, doesn’t provide a built-in best practice for this, but the underlying APIs are powerful enough that e.g. iPhone and iPad layouts for the same app can be created with a bit of hacking around.

For hybrid apps, virtually all MADP vendors provide some responsive CSS framework, and if not, you can always tweak it to your liking. However, complex design patterns like the iPad Split View, where the app is essentially rendering both the index and the details page at the

same time can be difficult if not impossible to achieve. If the feature is not inherently supported by the framework, constructing the CSS queries and HTML structures can be quite complex, or in the case of visual tools that produce generated apps, even completely impossible.”

All the different UI design patterns that might come up as requirements are too numerous to go through in this work, but there are many little caveats and things to note.

To give an example, since bitmap images cannot be scaled without loss of quality, it is important for non-vector images to be supplied in different resolutions (Apple, 2017f). Android further supports 9-Patch images, which allow a bitmap image to stretch itself to different sizes, by defining regions in the image that can be safely repeated (e.g. the uniform sides of a button, in contrast to its rounded corners) (Google, 2017a). Support for serving multiple resolutions of the same image assets is thus a requirement for top-of-the-line UI.

We define the following interview questions:

- **What different screen sizes did the app target?**
- **How was the UI adapted to work on all supported screen sizes?**

## 4.6 Interacting with Data

Dinh et al. (2013) define mobile cloud computing as “infrastructure where both the data storage and data processing happen outside the mobile device”. For the most part, this holds true for many internal enterprise mobile apps. As Paksula (2017) states:

“Most business processes revolve around data. This means interacting with digital business data is a key requirement for most internal apps. The data used by apps is mostly stored in existing ERPs, CRMs, databases and other backend systems, or maybe a new database is set up for the app. Similarly, large parts of an app’s business logic, such as report generation, task allocation and notifications scheduling are performed in the backend. After such a backend service has run, the result can be displayed back in the app UI, for example in the way of an updated data model (e.g. task has a new assignee) or by presenting the resultant artefact (e.g. PDF report) to the user.”

Similarly, Gartner lists connectors and API mediation as one of the evaluation criteria in their Magic Quadrant for Mobile Application Development Platforms for 2017 report (Wong *et al.*, 2017), and

Forrester mentions data and integrations as features that should be configurable declaratively by low-code platforms in their *Low-Code Development Platforms* report (Rymer, 2017).

The surveyed academic literature on mobile cloud computing does not really cover real-life scenarios applicable to how internal enterprise mobile applications should handle working with data integrations and backend services. Rather, they are focused on research on subjects like image processing and natural language processing; battery energy consumption; and mobile privacy and security in general, to name a few (Dinh *et al.*, 2013; Fernando, Loke and Rahayu, 2013; Sanaei *et al.*, 2014).

Many of these considerations do apply on some level to internal enterprise mobile apps, too: if the battery runs out very fast, the app can become quite unusable in the field, and e.g. cloud-based optical character recognition (OCR) could be very useful for some situations, such as reading device serial numbers if barcodes are not available.

Mapping out the all the different approaches and detailed requirements that the mobile client app might have in relation to working with data and services in the backend is outside the scope of this work. Instead, we will go through some common use scenarios and see what general requirements can be extracted.

#### **4.6.1 Common Scenarios for Interacting with Data**

Since most business processes involve multiple people (van der Aalst, 2013), a vast majority of internal mobile apps require the apps to share common data, in order to effectively execute the process as designed.

As each mobile app is running on a mobile device isolated from the other app instances, the apps must share a common backend system or multiple systems where the shared data is stored. A key requirement for an app, then, is that it must be able to communicate with the right backend systems over a mobile network, in order to view and interact with the relevant business data.

Of course, some data like app settings does not need to be stored in a backend system, but instead can be saved locally on the device itself.

In his interview, Paksula (2017) gives four general categories for how apps interact with data:

“The first category consists of apps that deal with just local data: for example, a calculator tool to determine the correct welding temperature for different metals. The result of the calculation is never sent anywhere, and any data storage happens locally on the device.

The second category is collecting new digital data, e.g. replace a paper form with a mobile app or gather employee feedback with a completely new process.

The third category is turning existing digital data into a more accessible and maintainable format, e.g. replacing an Excel sheet on a shared drive with a database and an UI for interacting with the data.

The fourth category is to provide a mobile interface for existing business data that is in the right kind of system or database, but currently hard or impossible access from the field. This could be e.g. leads data from a CRM or equipment data from an ERP.

In all but the first category, the real requirement is that the app has some way of talking with a backend. After that, the question becomes more about backend design: where is the shared data stored, how is the connection secured, how are conflicts handled if two users edit the same record, and so on.

It should also be noted that for most applications, the requirement is for simple CRUD (Create, Read, Update, Delete) operations on text-format data, often supplemented by e.g. binary photo attachments. Streaming data is rarely required, except maybe in the case of high-volume Internet of Things sensory readings and similar use cases.

Naturally, these four categories can also mix and match: perhaps the formula used by the local calculator is fetched from the backend, or new service call data is enhanced by customer data from a CRM.”

To keep with the scope of our work, we will not consider any of the backend requirements of an internal enterprise mobile application. Instead, we simply state that a certain category of apps will need a way to talk with the backend, fetching data and syncing local changes back to the server.

If the way of interfacing with the backend is uniform across an enterprise’s apps and backend systems, it is clear that development speed gains can be achieved, as a different approach doesn’t need to be created and tested for each app project. Dinh *et al.* (2013) share this notion, calling for a standard

interface between mobile apps and their cloud services, although their discussion is more on a protocol level rather than API/client implementation level.

#### 4.6.2 User Interface for Data

Once the data has been received by the mobile app, it naturally needs to be presented to the user over a suitable UI to be useful. This could be e.g. a list of recent leads or a map of all vehicles in the company's fleet. If the data is not read-only, an UI needs to be provided for the user to create new or edit existing data.

As Paksula (2017) states:

“The basic functionality that a MADP should fulfil, then, is an easy and uniform way to create UI that allows users to list data, view details of individual data records, create new records and edit existing ones. Additional features like visualizing equipment locations on a map, showing graphs or collecting photos alongside text data are likewise very important.

Good user experience is key here, to: inputting the data should happen with the right kind of controls, and data loading times should be optimized by e.g. paginating long lists as multiple pages, and with smart caching.

Of course, some data sources have different requirements for the UI. If the app is streaming Internet of Things sensor data at tens of readings per second, the developer would not want to treat each data point as an individual record, but probably rather show the data stream e.g. in an animated graph.”

Gartner (Vincent *et al.*, 2017) specifically mentions that many low-code MADP vendors specifically focus on providing CRUD interfaces for interacting with data, for “easy entering of information and ease of access to information”. If companies focus on these aspects of their platforms, it is logical to assume that this is due to corresponding business demand.

We define the following questions:

- **What kind of business data does the app utilize?**
- **How is data synchronized between the app and the backend?**
- **What kind of UI is there for the user to view and interact with the data?**

## 4.7 Native Device Capabilities

We define “native device capabilities” as those functionalities that cannot be accessed from a web app, or that have limitations when accessed via a web app; as such, they are one of the main reasons why

Based on a review of academic literature and analyst reports on cross-platform apps, we find the following native device capabilities that can be crucial:

- **Barcode/QR code reading:** being able to read different kinds of 2D codes and access the information within (Hartmann, Stead and DeGani, 2011; Palmieri, Singh and Cicchetti, 2012; Wong *et al.*, 2016).
- **Bluetooth support:** being able to interface with Bluetooth devices, including BLE (Bluetooth Low-Energy) beacons (Hartmann, Stead and DeGani, 2011; Palmieri, Singh and Cicchetti, 2012; Wong *et al.*, 2016)
- **Camera:** being able to capture photos and videos, as well as access existing ones stored on the device (Hartmann, Stead and DeGani, 2011; Heitkötter, Hanschke and Majchrzak, 2012; Palmieri, Singh and Cicchetti, 2012; Ribeiro and da Silva, 2012; Dalmasso *et al.*, 2013; Joorabchi, Mesbah and Kruchten, 2013; Wong *et al.*, 2016; El-Kassas *et al.*, 2017)
- **Contacts:** access to the device contacts information (Hartmann, Stead and DeGani, 2011; Palmieri, Singh and Cicchetti, 2012; Ribeiro and da Silva, 2012)
- **Device sensors:** being able to access to data from device sensors like geolocation, accelerometer, compass, luxometer, temperature sensor, and so on (Charland and LeRoux, 2011; Hartmann, Stead and DeGani, 2011; Palmieri, Singh and Cicchetti, 2012; Ribeiro and da Silva, 2012; Dalmasso *et al.*, 2013; Joorabchi, Mesbah and Kruchten, 2013; Wong *et al.*, 2016, 2017; El-Kassas *et al.*, 2017; Latif *et al.*, 2017)
- **File system access:** being able to store files on the device internal file system and access those files (Hartmann, Stead and DeGani, 2011; Palmieri, Singh and Cicchetti, 2012; El-Kassas *et al.*, 2017; Latif *et al.*, 2017)
- **Maps:** being able to use the native maps component to display highly performant map-based UIs (Hartmann, Stead and DeGani, 2011; Latif *et al.*, 2017)
- **Push notifications:** being able to receive and display push notifications sent from the backend (Hartmann, Stead and DeGani, 2011; Palmieri, Singh and Cicchetti, 2012; Wong *et al.*, 2016, 2017; Latif *et al.*, 2017)

The literature mentions other features like Apple Touch ID (Wong *et al.*, 2016), NFC support (Palmieri, Singh and Cicchetti, 2012), calendar access (Hartmann, Stead and DeGani, 2011) and so on. Since new native device capabilities are constantly being developed, trying to reach an exhaustive list of all potential native capability requirements is beyond the scope of this work.

Instead, the requirements will be apparent from a well-defined app business case. Thus, it is relatively simple to see if the MADP under consideration gives access to the required features either as a packaged platform feature, via third-party extensions such as PhoneGap plugins (Heitkötter, Hanschke and Majchrzak, 2012), or by allowing developers to build the required capability with native code.

We define the following interview questions:

- **What kind of native device capabilities does the app utilize?**

## 4.8 User Authentication

Enterprises usually want employees to have a single username/password combination with which to log in to all corporate systems. This approach is called Single-Sign On (SSO), and it has significant benefits from a security and management point of view (Pashalidis and Mitchell, 2003).

### 4.8.1 Difficulty of Implementation

It would make sense for employees to authenticate with the same SSO user credentials for all internal mobile apps, which in turn requires the same SSO solution to be implemented for each app.

This can be quite difficult. For example, Sun and Beznosov (2012) describe various security vulnerabilities that can typically be found in implementations of OAuth 2.0 SSO<sup>13</sup>.

Since the implementation is not trivial to get right, it follows that if the SSO implementation is done from the ground up for each app, the probability of a bug with serious security implications finding its way to production increases.

---

<sup>13</sup> OAuth 2.0 is a widely used authentication protocol, used by e.g. Microsoft Azure (Microsoft, 2017) whose Active Directory service is a popular choice for corporate user management.

## 4.8.2 Protection Against Vulnerabilities

Vulnerabilities in login protocol implementations are constantly discovered: for example, Armando *et al.* (2008) describe finding a new flaw in Google’s SAML-based SSO. The bug was promptly patched up by Google, but such a patch is useless unless it can be rolled out to all affected systems.

## 4.8.3 Different Authentication Types

Of course, one SSO solution is probably not enough to cover all authentication needs for the enterprise. As Paksula (2017) states:

“While authentication against Active Directory or LDAP is the most commonly seen auth type, internal apps often involve stakeholders that do not have an account in the existing corporate directory. In such cases, it might not be viable to create new accounts, but instead a separate user management system will need to be implemented for the mobile app.

Similarly, some apps require users to be able to register for new accounts in a self-serve manner, sometimes with an approval process to e.g. link CRM data with the registering customer.”

## 4.8.4 Multi-Factor Authentication Support

Authenticating the users by multiple factors (e.g. both password and biometrics, or password and SMS-delivered security code) provides additional security (Huang *et al.*, 2014). Thus, support for different kinds of Multi-Factor Authentication (MFA) scenarios might be a requirement.

## 4.8.5 Session Persistence Across Multiple Apps

If the same SSO solution is used to authenticate with all internal apps, it would be ideal if a separate login process wasn’t needed for each app. Instead, employees should log in once and be authenticated with all the internal apps they use.

Mobile Application Management (MAM) platforms allow centralized management of mobile apps on employee devices (Harris and Patten, 2014), and sharing the SSO session between multiple apps is one of their key selling points (Pawar, 2016). A MADP should ideally either provide this functionality on its own, or allow some external MAM solution to be used.

## 4.8.6 User Permissions

Since business processes have different kinds of stakeholders with different responsibilities and rights, it is natural that different users need different kinds of permissions in an app.

As Paksula (2017) states:

“The first step is to get the attach the correct roles for the authenticated user. This can be done by looking at permissions, roles or user groups metadata for the user from the corporate directory that the user authenticates with.

Alternatively, the mobile app backend can define its own app-specific user roles, which then requires manual work to assign each user to the correct roles/user groups.

Typically, an app has at least regular users and admin users, but each business process determines the kind of roles required, and as such, separate UI

Then, permissions attached to each role need to be set up. On a general level, there’s data permissions and UI customization. Different users and user groups need to be able to perform different kinds of operations on data, e.g. regular users can list data but only admins can edit it. In some cases, access to individual records or fields should be limited.

Similarly, many apps require UI customization based on permissions: admins might need to have access to more fields when creating a new record, or have a completely separate view that is not shown for regular users.”

In this work, we do not consider what kind of backend and admin interfaces are needed for robust user and permissions management. Instead, we look at only the requirements for the client app once the user has his or her assigned roles.

We define the following interview questions:

- **How was user authentication handled in the app?**
- **What kind of different user roles were there?**
- **How did the different roles affect the functionalities of the end app?**

## 4.9 Offline Data

Giguère (2001) defines three categorizes for mobile data connectivity models:

- **Always connected**, where the application is constantly connected to the backend, and loss of connectivity results in a fatal error or otherwise unusable program.
- **Occasionally connected**, where access to the backend is only occasionally available, and a local data server is required to keep the application running.
- **Always available**, a blend of the two previous models, where the expectation is that a connection to the backend is available, but that the application can handle situations where the connection is lost. This can be achieved via a local data cache or a local data server.

A key factor to consider is that mobile device network connectivity cannot be guaranteed, especially over cellular data networks: the device can always suddenly go offline. That said, enterprise apps can function under all of the three data connectivity models. Paksula (2017) states in his interview:

“For some apps, going offline is not a problem, as the app is designed from the ground up to work without any network connectivity. This could be the case if the app is used in areas without any cellular coverage, such as remote locations or underground mines. That said, even these apps might have features that require Internet connectivity, such as when initially authenticating with the backend, or when downloading fresh data to the local database.

If the app is used in areas where WiFi or cellular data can be virtually guaranteed, offline support might not be a requirement at all. If the data connection is momentarily lost, the app can simply cease to function for the duration. It should optimally recover gracefully when connectivity resumes, but not caring about offline can greatly simplify the architecture of the mobile app.

The third option is to provide graceful and unobtrusive transitions between offline and online modes, with perhaps some features disabled when network connectivity is down, while still allowing currently cached data to be browsed and new data transactions to be queued up for when the connectivity resumes.”

Academic literature notes that offline support might be a requirement (Hartmann, Stead and DeGani, 2011; Smutný, 2012; Dalmaso *et al.*, 2013) and that offline caching can be hard to get right

(Joorabchi, Mesbah and Kruchten, 2013), but does not really address the intricacies of real offline support requirements.

Gartner reports, on the other hand, highlight the importance of robust offline support for enterprise mobile app development (Baker, 2016; Wong *et al.*, 2016, 2017; Vincent *et al.*, 2017).

Next, we'll examine how reading and writing data while offline pose two very different challenges.

#### **4.9.1 Reading Data While Offline**

Reading data while offline is relatively simple: the data can simply be cached to a local database or other storage method, and then refreshed to the latest version when connectivity resumes.

However, in some cases the remote data set can be too large to download into the local cache, or at least specific strategies have to be developed for data syncing, so that the whole data set doesn't need to be re-downloaded every time (Joorabchi, Mesbah and Kruchten, 2013).

Paksula (2017) furthermore notes:

“If multiple data sources are consumed by the application, developing a local caching solution that reflects the structure of all the outside data sources might not be straightforward to implement, especially if the different data resources have relations between each other that need to be resolved offline.

Another aspect to note is user permissions: if different users have access to different data resources or records, it could be a security issue to download the whole database onto the local device. Thus, permissions should be resolved before caching the data. If the permissions are defined separately from the data sources themselves, this process can become quite complex.”

A more thorough review of the different strategies available for offline caching is beyond the scope of this work.

#### **4.9.2 Writing Data While Offline**

Writing data while offline is a more complex operation, mainly due to the potential for conflicts when synchronizing the local data with the backend after connectivity resumes (S. Agarwal, Starobinski and Trachtenberg, 2002).

Gartner (Baker, 2016) mentions two modern approaches to mobile app offline support: the data can be stored in a local database that can then be synced with the backend database when connectivity resumes, or then transactions can be cached while offline and replayed individually against the backend once back online.

The first option might be viable when there is an explicit database that can be synced with, but if the app is integrated with multiple data sources, any transactions need to be played back to all affected systems, which is a lot more complex than syncing with a single database.

Both approaches require some sort of strategy for conflict resolution if the transactions affect existing data, i.e. data is edited or deleted, instead of just new data being created: if two users perform an edit operation on the same field while offline, some strategy needs to define which edit is actually saved to the master data in the backend.

A more thorough review of the different strategies and nuances of data synchronization is outside the scope of this work.

### **4.9.3 Requirements for Offline Data**

The requirements for offline support are highly dependent on the business process implemented by the app, so a viable course of action is to examine what app features and workflows need to be available offline, and then see what that means in terms of data caching, synchronization and conflict resolution.

We define the following interview questions:

- **Did the app have offline support? To what extent?**

## 5 Methodology Results

This chapter provides an overview of the results of our qualitative interviews in relation to the literature-based findings uncovered in Chapter 4.

### 5.1 Overview of the App Projects and Mobile App Strategy

The companies' internal app projects and the corporate strategy and business drivers behind them varied greatly. Table II provides an overview of the main business cases and strategic drivers for the apps developed, as well as when the app development started and the number of apps created.

<b>ID</b>	<b>App Main Business Case(s)</b>	<b>Strategic Drivers</b>	<b>App Dev Started</b>	<b>No. Apps</b>
<b>App-Maintenance</b>	Mobile reporting and management of work orders for mechanics	Digitalizing core part of business for increased efficiency	Over 4 years ago	1
<b>MADP-Education</b>	Contacts directory for external stakeholders; giving students machine-learning-driven suggestions on how to better spend their time	Become more competitive by improving key metrics like student happiness and relevance of learnt skills upon graduation	1 year ago	2
<b>MADP-Manufacturing</b>	Mobile access to business data; approval processes; digital forms; employee education; asset management	Boost internal functions and processes; increase level of digitalization in organization	2 years ago	15
<b>App-Engineering</b>	Interface with an embedded computer of a proprietary machine developed by the company, in order to change and read parameters	Replace old proprietary hardware and accompanying paper manual required to decode the outputted data	Over 2 years ago	1

Table II: Interview Participant App Projects.

### 5.1.1 App-Maintenance's App Projects

App-Maintenance's company had worked on just a single app, with no intention of building more custom apps. The driver for the project, which began 4 years ago, was to digitalize a core aspect of their business with a mobile app, in order to get more billable hours in and get a better grip of the business data related to the work done by the mechanics. An important driver was also to be a forerunner and get the app working before competitors would get a similar mobile solution for free from their enterprise system providers (such as IBM Maximo); a prediction that has recently come true.

The app project was done as part of a bigger digitalization effort, which includes e.g. backend integrations to some 20+ systems via a custom-built integration bus, reporting tools and a desktop app for management purposes.

The app's functionality is straightforward. A back-office desktop app allows work orders to be created and assigned to mechanics, who then acknowledge them, update the work order status throughout the job, and finally turn in a report of the work. Additional features include hour reporting and an upcoming feature for mileage logging.

### 5.1.2 MADP-Education's App Projects

MADP-Education's company had started working on internal mobile apps around a year ago, when the company made a significant shift in strategy towards embracing internal app development - previously, there had been a hard line against developing anything internally. The main business driver is to become more competitive by improving the quality of teaching, student wellbeing, relevance of student skills upon graduation and similar metrics. Improving employee processes was seen to take a significantly lower priority. MADP-Education noted that mobile apps do not have inherent value for them: first, the processes that might require improvement are identified, and then developing an internal mobile app is simply one of the possible solutions.

There had been two native apps developed to date (with an additional two responsive mobile web apps not covered in the interview). The first one was a simple app for stakeholders to better find the right people and their contact information. The second one was work in progress, with the aim of analyzing big data gathered from the student learning processes, in order to present students with

automated guidance on how to best spend their time. The first version of the student app that was due to be released soon simply presented some data about available courses and campus restaurant menus, the plan being to develop and roll out the big vision gradually.

### **5.1.3 MADP-Manufacturing's App Projects**

MADP-Manufacturing had the biggest velocity in internal app development, with some 15 internal apps developed and deployed over the past two years for various purposes. Their mobile app strategy stemmed from general corporate strategy that outlined the importance of boosting internal functions and processes, as well as digitalization throughout the organizational levels. After making the observation that a mobile interface would be a nice improvement for many processes, the internal app development process was ramped up from an initial single pilot project to its current state.

The apps created ranged from improved mobile interfaces to SAP and other backend systems, to different approval and invoicing processes, to a QR-code-based device registration app, to even a small game to teach employees the company code of conduct. Most of the apps had been targeted for office workers, but some had been deployed all the way to the factory floor.

### **5.1.4 App-Engineering's App Projects**

App-Engineering's company had developed a single native internal app (with an additional responsive web app not covered in the interview), in an ongoing project that started over two years ago. The app's purpose is to let service mechanics interface with the embedded control computer of the machines produced by the company. This then allows control parameters to be manipulated and error codes to be interpreted.

The app replaced an older process, where a custom-built device displayed the raw error codes and gave access to parameters by ID number only, requiring additional paper manuals to figure out what the data means or what a specific parameter does. The app utilizes a custom-built USB dongle that handled the interface between the smartphone and the machine.

The main business driver behind App-Engineering's company's app development effort is to increase the efficiency of business processes; in this app's case, to get rid of the paper manuals and to improve the productivity of work. App-Engineering mentioned that mobile apps will have a larger role in the company strategy in the future.

## 5.2 Technologies Used in Projects

App-Maintenance’s single app was built on top of an industry-specific framework, with many custom-coded features and functionalities. The framework used is not a MADP in that it does not e.g. provide any packaged authentication methods or any integration capabilities apart from a simple proxy middleware.

MADP-Education and MADP-Manufacturing used a single, full-featured Mobile App Development Platform for all their internal app development, with MADP-Manufacturing additionally having used the low-code mobile app development tools of their ERP system to create a few apps.

App-Engineering’s app was hand-coded with a cross-platform framework that produces native apps. A possible future requirement for iOS support influenced the decision, as well as wanting to share the code base with a desktop app with similar functionalities.

Table III presents the technologies and development approaches used in the app projects

<b>ID</b>	<b>Platform Technologies Used</b>	<b>Development Approach</b>
<b>App-Maintenance</b>	Highly customized industry-specific app framework	High-code
<b>MADP-Education</b>	Full-fledged Mobile App Development Platform	Low-code with high-code components
<b>MADP-Manufacturing</b>	Full-fledged Mobile App Development Platform; ERP system provider’s low-code app development tool	Low-code with high-code components
<b>App-Engineering</b>	Cross-platform framework	High-code

Table III: Technologies Used.

## 5.3 Team Composition

App-Maintenance and App-Engineering utilized fully outsourced developer teams with internal employees handling product ownership and project coordination. App-Maintenance had around 20+ coders from different companies working on the app project, with 4 internal project coordinators; though to be fair, the mobile app under work is just one aspect of the whole digitalization effort.

App-Engineering had a single project coordinator and three outsourced developers working on-site. MADP-Education had 1,5 developers working on building the mobile apps, plus a project coordinator. MADP-Manufacturing had 2 full-time app developers and one project coordinator.

In addition, all companies utilized non-technical team members in the requirements gathering process. App-Engineering also had an internal UX team working on the app project, and further mentioned non-developers being used to write up language localizations.

### 5.3.1 Role of Citizen Developers

None of the companies had citizen developers participating in actual app development. For App-Maintenance and App-Engineering, the hand-crafted approach meant that coding skills were a requirement for participating in the development process. The MADPs used by both MADP-Education and MADP-Manufacturing included powerful low-code/no-code tools, but both participants strongly felt that the hassle and required oversight brought on by citizen developers would outweigh any benefits. MADP-Manufacturing further noted that “coming from a traditional industry, it’s already difficult enough to get users to use the apps in the first place”.

App-Engineering noted that it would be wonderful to be able to edit e.g. the app UI without having to go through the development process. App-Maintenance, on the other hand, felt that since the main brunt of the app requires such complex business logic and data integrations that they essentially require coding to implement, and as such it seems unlikely that a visual tool or the involvement of citizen developers would speed things up.

Table IV summarizes the team compositions of the interview participants.

<b>ID</b>	<b>Development Team Size and Composition</b>	<b>Role of Non-Developers</b>	<b>Dedicated UX Team</b>
<b>App-Maintenance</b>	20+ outsourced developers; 4 internal project coordinators	Testing, requirements definition, content production	No
<b>MADP-Education</b>	1,5 in-house developers; 1 in-house project coordinator	Testing, requirements definition	No
<b>MADP-Manufacturing</b>	2 in-house developers; 1 in-house project coordinator	Testing, requirements definition	On some projects
<b>App-Engineering</b>	3 outsourced developers (on-site); 1 in-house project coordinator	Testing, requirements definition, UX design input, translations	Yes

Table IV: Team Composition.

## 5.4 Project Timeframes

The time required by the different app projects varied greatly. App-Maintenance’s single app had been in the works for 4 years, while App-Engineering’s app had been developed for over 2 years. Granted, both had undergone significant evolution throughout that time. App-Maintenance noted that the first version of the app took 6 months to complete.

The contrast is stark with the development times reported by the MADP users: MADP-Education said the phone book app took 3 days to create, while the first version of the student app took around one month’s worth of man-hours. MADP-Manufacturing had even greater velocity, saying that the first version of a new app is usually created in days, and in some cases even just a few hours.

App-Maintenance and App-Engineering both mentioned that implementing the actual features took most of the time: for App-Maintenance, it was data integrations with the various backend systems and different kinds of work orders; for App-Engineering, it was coding the communications channel between the app and the proprietary control system on the machines.

For both MADP users, a significant amount of time was spent in requirements gathering and communicating with the business owners, with the app development itself taking less time. While exact figures were not discussed, the feeling was that getting line-of-business product owners to take time out of their regular schedules to e.g. plan the next development sprint or weigh in on a design decision took considerably more effort than actually implementing the features. MADP-Manufacturing also mentioned testing and a UX-focused planning process as things that take time.

## 5.5 Project Requirements Gathering

All of the respondents talked with the end users of the app to some extent as part of the requirements gathering process; everyone also mentioned that it would be good to do it more.

App-Maintenance felt that gathering the initial requirements for the app was quite straightforward, as it was an existing core process that was to be digitalized. In the current process, representatives of the main user groups and the product owner gather to collate feedback and bug reports from the field, and prioritize them for further development. There's a steady influx of concrete ideas from the field, and figuring out how to implement them hasn't posed any problems.

For MADP-Education, student questionnaires have been an important source of data on what sort of apps should be developed. New app ideas go to an idea portfolio, where they are then moved to a project portfolio when it's clear what's to be done. As the apps evolve, the final word on requirements is given by the product owners of the app. Integrations especially have been easy to define, as the data already exists.

MADP-Manufacturing was the only participant that had a formal process defined for requirements gathering. New ideas are refined into a list of ideas as "opportunity cards". They are prioritized, and the most promising ones are moved to a mobile reference group, who then develop and prioritize them further. Finally, a product owner is found for each app. When the app project is started, external UX consultants go through a service design process, with e.g. wireframes used to ensure the UI design is sound. Feature-level sprint planning takes place for a 2-week Scrum sprint at a time.

MADP-Manufacturing also mentioned that they hadn't mapped out the requirements of upcoming mobile apps in any comprehensive capability before selecting the platform, but rather had been fortunate in that all the app ideas could be realized with their chosen MADP and the skills of the development team without any real technical issues or blockers.

For App-Engineering, the initial requirements were clear: the app had to replicate the functionality of the older proprietary interfacing device. New requirements have been gathered by collating feedback from the field and then refined into features to be implemented by the UX team, the end-users (i.e. the mechanics) and the developers. The basic functionality has remained the same.

### **5.5.1 Difficulties in the Requirements Gathering Process**

All participants mentioned various difficulties in the requirements gathering process.

MADP-Education and App-Engineering both mentioned the difficulty of UX design. For MADP-Education, it took effort to reach a login design for the student app that's both secure and usable. For App-Engineering, the hardest part had been in figuring out how to reimagine the old physical device interface to utilize the smartphone touchscreen in the best way.

App-Maintenance and MADP-Manufacturing both noted that even if the business process is defined similarly on a global level, different sites in different countries will have different practical implementations of it, stemming from local workarounds to process and enterprise system limitations. Since App-Maintenance's company had formed through numerous corporate acquisitions, the particulars of local agreements also posed difficulties for implementing uniform processes.

App-Maintenance also mentioned the inherent complexity of the underlying processes as a challenge: there are two legal corporations, with cross-billing and generally convoluted financial processes that need to be taken into account.

MADP-Education highlighted the importance of product owners, while mentioning that it was hard to find them for new apps, as everyone is busy with existing work. For MADP-Manufacturing, on the other hand, the difficulty was not in finding product owners, but rather getting them to spend enough time with the apps, as all of them have many other responsibilities, too.

MADP-Manufacturing further brought up the difficulty of getting product owners to think in terms of user stories and the general core business problem. Instead, they have a tendency of jumping straight

into implementation details and UI design; this was attributed to their background in working with SAP and other legacy systems with a more waterfall-type specification and development process.

## 5.5.2 Evolvement of the Requirements

All of the participants mentioned that app requirements had evolved throughout the app projects.

The basic functionality App-Maintenance's app has evolved organically as new feature requests have been brought in and implemented, often responding to changes in the underlying business processes. Compliance with the EU GDPR regulation was specifically mentioned as a requirement that was simply unknown when the project was started.

MADP-Education and MADP-Manufacturing both note a very rapid cycle in requirements re-definition. MADP-Education says that requirements evolve weekly, while MADP-Manufacturing has two-week Scrum sprints. MADP-Manufacturing did mention that the change in requirements can be even more rapid at times: even during a 2-week sprint a product owner can get some new insight from a site visit abroad and come back with new requirements that can, at worst, cause recent work to be essentially worthless.

By contracts, the broad strokes of App-Maintenance's app backlog are decided on for a full year every Spring, although there's natural evolution and re-prioritization happening as the year progresses.

For App-Engineering, the requirements have developed very gradually and mostly on the UI level; the communications channel between the mobile device and the machine's embedded systems has stayed the same. The biggest change has been in understanding that the old physical device doesn't need to be copied over one-to-one, but that e.g. informative names can be shown for errors instead of number codes.

App-Engineering also brought up that for a non-developer, it can be difficult to know what is easy to code and what is not. If the participant could start the project over, more feedback would be gathered to specify certain things in more detail before deciding on the implementation, as it is harder to modify things afterwards. By contrast, App-Maintenance noted that it would be great if business understood the realities of software development better, i.e. what is easy to code and what is not, and how does taking technical debt potentially affect future development.

Both App-Maintenance and MADP-Manufacturing mentioned that as the mobile app projects have been rolled out and old-fashioned end users have witnessed the benefits first-hand, the speed of idea

generation has increased. It becomes easier for people to consider if a mobile app feature would help with a particular problem once there's some hands-on experience.

None of the participants mentioned any fundamental technological requirements changing, such as suddenly needing to implement offline support or support for a new mobile platform.

App-Engineering specifically brought up that lightweight interactive prototypes would be helpful in validating the ROI of an app project before committing to full development.

## **5.6 Functional Requirements**

We then move on to cover how the requirements of the participant's apps map to our categories identified in chapter 6.

### **5.6.1 App Types**

MADP-Manufacturing knew the apps produced by their MADP to be fully native. MADP-Education was unsure, but guessed the apps to be hybrid due to the hot-reload functionality (i.e. app updates can be pushed to users over-the-air, without the need to publish a new app binary). Analysis of the websites of the MADP vendors in question reveals that the low-code tool in both cases produces fully native apps.

App-Maintenance and App-Engineering were both unsure about the app type, with both coming to the conclusion that they had to be fully native. It should be noted that neither was involved directly with the coding the app, which explains the uncertainty at least to some extent.

MADP-Manufacturing further mentioned that they had previously built apps with simple UIs as responsive websites (i.e. web apps whose UI automatically adapts to both desktop and mobile screen sizes), but since the MADP makes native app development as fast as building HTML5 apps, even simple apps are now built natively.

### **5.6.2 Platform, Device Model and Operating System Version Fragmentation**

App-Maintenance wanted each mechanic to have the same Android device, whose requirements included durability and dirt resistance to account for the conditions in which the mechanics do their work. The company decided on Samsung Xcover 2, which was the only device on the market to fulfil the requirements. The target model has since been upgraded twice, with only a single officially

supported device in use at any time. iOS is not supported at all, and while the app does run on other devices, no official support is given.

App-Engineering's app targeted the official list of Android devices supported by the company, which is some 20 devices from three manufacturers.

Neither App-Maintenance or App-Engineering remember the exact Android OS version requirement; App-Engineering guessed 5.0 to be the minimum.

MADP-Education and MADP-Manufacturing support both iOS and Android. MADP-Education effectively supports all the device types supported by the MADP platform, with specific OS version requirements written into their mobile strategy: 4.4+ for Android OS and 9+ for iOS. MADP-Manufacturing uses Microsoft Intune as their mobile device management solution, and effectively supports any device supported by Intune. MADP-Manufacturing's employees can bring their own device, or choose from a list of company-provided phones.

App-Maintenance and App-Engineering test on all the supported device models. For MADP-Education and MADP-Manufacturing, testing the functional requirements on whatever devices have been available for the development team has been enough, with MADP-Education further utilizing closed beta testing to weed out bugs. Both MADP-Education and MADP-Manufacturing trust the MADP vendor to have ensured that the apps run similarly on all supported devices.

### **5.6.3 Battery Power Consumption**

None of the respondents noted any issues with battery power consumption. App-Engineering mentioned that because of the physical USB dongle, phone actually charges while the app is being used.

### **5.6.4 User Interface**

The UI requirements for all apps were very straightforward: everyone reported the need for basic lists, details views, input fields and buttons. MADP-Education further reported using modals, swipeable controls and map views, all of which were provided by their MADP.

App-Engineering also noted that it would be extremely helpful if the app UI could be tweaked without having to go through the actual development team. App-Maintenance noted the benefits of having an XML-based data structure, which allows new data fields to be added very fast, with the UI

adapting automatically. At the same time, App-Maintenance noted that the heavy XML structure can cause some performance issues, as processing it is not effortless.

App-Maintenance was the only participant to report issues with app UI performance, though that was attributed to the initial Samsung Xcover 2 device being generally underpowered, instead of real issues with the app code itself. The issue was tackled with tweaks like data pagination, but ultimately solved by upgrading the phone model to a more powerful one.

All participants had customized the app UI to look like their company brand. MADP-Education and MADP-Manufacturing had configured a company template with logo, icons, colors, font etc. into the MADP tools, App-Maintenance had done minimal customization on the framework base app in terms of colors and logo, whereas App-Engineering had developed a fully branded UI from scratch. App-Engineering was the only participant to have specified formal design guidelines for mobile apps.

All participants mentioned the need for language localizations, with MADP-Manufacturing having the first localized app under development, and everyone else having already implemented localizations. The only localization type required had been text translations. Supported languages ranged from just Finnish and English for MADP-Education to “most European languages and Chinese” for App-Engineering. All participants used some localization tool that came with the framework or MADP that was used to build the app.

MADP-Education noted that language localizations were harder to implement for native apps compared to web with the MADP tooling used, but that it wasn't a big nuisance.

### **5.6.5 Screen Size and Resolution**

App-Maintenance, MADP-Education and App-Engineering have only targeted phone-size screens with their apps, with MADP-Manufacturing having additionally built apps for tablet use, too.

All of the respondents mentioned that the app UI scales to tablet resolutions without glitches. None of the respondents had developed separate UI layouts for tablets for any apps.

### **5.6.6 Data**

MADP-Education and MADP-Manufacturing noted that sometimes getting the data on the phone from multiple backends was slow, but once the data had been fetched, everything worked very well. MADP-Education mentioned using the caching solution provided by the MADP to solve slow backend response times.

The framework used by App-Maintenance provided a basic middleware proxy server, i.e. a dedicated server sitting between the mobile app and the rest of the backend systems being connected to. The proxy server then connected to an integration bus custom-built by App-Maintenance's team to reach the rest of the integrated systems.

For MADP-Education and MADP-Manufacturing, all data syncing and backend integration needs were covered by the MADP. For App-Engineering, the only business data used is the information flowing through the USB dongle. Syncing collected data to backend systems happens via a separate Windows-based desktop program.

App-Maintenance mentioned some Cross-Origin Resource Sharing (CORS) problems with data, as well as the difficulty of getting data to flow from third-party systems with suboptimal or missing APIs.

MADP-Education noted that when data was made available via the mobile app, it became apparent that the data in previously closed legacy systems was not always high quality: since the data hadn't been used anywhere else, people had skipped corners when inputting it.

### **5.6.7 Native Device Capabilities**

App-Maintenance mentioned camera, push notifications with deep linking and Google speech-to-text, with an upcoming mileage reporting feature including geolocation.

MADP-Education mentioned camera and geolocation, with push notifications and Bluetooth beacons coming up on the backlog.

MADP-Manufacturing mentioned camera and QR scanning.

Both MADP-Education and MADP-Manufacturing had been able to use built-in native features of their MADP solution, with no need for custom native extensions so far.

App-Engineering mentioned the USB communications channel as the only native capability utilized. Different USB standards for different Android devices had caused difficulties.

### **5.6.8 User Authentication**

App-Maintenance, MADP-Education and MADP-Manufacturing all authenticated against corporate directories: App-Maintenance with Windows AD, MADP-Education with LDAP and MADP-Manufacturing with AD FS (Active Directory Federation Services, a single-sign on system by

Microsoft). App-Engineering authenticated users via the USB dongle: each mechanic had a personal one that identified the user.

App-Maintenance specifically brought up authentication as a feature that was difficult to implement.

MADP-Education and MADP-Manufacturing had implemented Multi-Factor Authentication. For MADP-Education, users log in via their LDAP credentials. When logging in for the first time, a mobile PIN code is set up. The PIN code needs to be entered every time the app opens, and changed every 6 months.

MADP-Manufacturing was the only one to have implemented different user roles: for the most apps, a user was either allowed to use the app or not, with a few apps having a data admin type role. The roles were determined via ADFS data. MADP-Education wasn't sure yet if a separate teacher role would be required for the student app, or if teachers simply couldn't use the app, but that role distinction would be picked up from LDAP data, too.

No participants mentioned the requirement for user permissions and roles apart from those defined in the corporate directory.

### **5.6.9 Offline Data**

Both App-Maintenance and MADP-Education's apps had no real offline support and effectively required a network connection to function.

MADP-Manufacturing had built several apps with offline data functionality. The requirements of the offline capability varied between apps. For some, simply reading the data offline was enough. For others, it was a requirement to sync offline operations with the backend once connectivity resumes, which then required conflict resolution rules to be set up. Everything was achievable with the tools provided by the MADP.

App-Engineering's app doesn't utilize the network connection at all, so it naturally works offline.

## **5.7 Blockers to More Rapid App Development**

App-Maintenance, MADP-Education and App-Engineering mentioned budget and thus limited developer resources as the main blocker on app development speed. MADP-Manufacturing noted that two full-time MADP developers is a very optimal team size: if more developers were hired, it would be an even harder task to get the product owners to sit down enough to guide the app projects.

MADP-Education also felt that the developer count couldn't be just brought up e.g. tenfold, as then gathering business requirements and collecting feedback from product owners would become the bottleneck.

All participants had more or less well-formed, prioritized backlogs of app ideas – or in the case of App-Maintenance and App-Engineering, feature ideas for the individual app. The specifications and requirements of an app and its features became clearer as it approached entering development, with many ideas existing in a less refined stage due to prioritization. Lack of raw ideas or business needs was clearly not a blocker for development throughput. Rather, the process of gathering requirements and designing the optimal solution was the slow part.

App-Maintenance and App-Engineering mentioned slowness of coding as a limiting factor, while MADP users MADP-Education and MADP-Manufacturing didn't. MADP-Manufacturing in particular was very happy with the speed, “everything happens at the snap of fingers”.

MADP-Education and MADP-Manufacturing further noted that while their MADPs definitely provided significant speed gains over traditional coding, it still required a competent developer to operate them. MADP-Education mentioned that it came as a bit of a surprise that a user pretty much needs to know how to code to really use even the visual parts of the platform. MADP-Manufacturing talked about some student interns who had built apps with the MADP that more or less performed their intended functionality, but were quite horribly crafted under the hood; highlighting need for developer skills. Still, MADP-Manufacturing mentioned that it's great that even relatively incompetent developers can get a usable app up and running quite fast.

App-Engineering also mentioned that it is not straightforward to determine the business impact of a new app to justify the return on investment, while MADP-Manufacturing noted that with the rapid development speeds, it has become viable to develop apps whose target audience can be as small as 5 employees.

## **5.8 Observations on the Projects' App Requirements**

The app cases and business environments of the four interview participants are quite varied, so very generalized, overarching conclusions cannot be drawn from the interview data. Some relevant, interesting observations can still be made:

### 5.8.1 Shared and App-Specific Requirements

The interviews reinforce our initial hypothesis that internal enterprise mobile apps have both shared and highly app-specific requirements.

For example, the required UI components of most discussed apps are very basic ones – lists, details views, data input fields and so on.

At the same time, while App-Engineering’s app has a very basic UI, one whose components could likely be re-used from other apps, the native USB messaging channel is a highly app-specific component that could only be re-used in other apps that interface with the proprietary machines produced by App-Engineering’s company.

Some specific requirements are fulfilled on a company level by the platform solution: both MADP-Education and MADP-Manufacturing were happy with the general MADP theme they had constructed, with no need to tweak the basic UI elements for different apps once the brand-specific customization had been once done.

Similarly, MADP-Education and MADP-Manufacturing utilized the same authentication mechanism across all their apps.

### 5.8.2 Benefits of Low-Code Tools

While the interviews didn’t explicitly explore the strengths and weaknesses of the platforms approach, it became clear that when the required features were available in the MADP used by MADP-Education and MADP-Manufacturing, significant development speed gains were realized, with few if any compromises to end-app quality or other requirements.

Furthermore, both App-Maintenance and App-Engineering brought up some low-code/no-code parts of their development process in positive light. For App-Maintenance, being able to modify data models by editing XML instead of having to change the underlying database schema was a welcome speed boost. For App-Engineering, being able to edit language translations and notification messages as a non-coder was beneficial.

Finally, both App-Maintenance and App-Engineering noted the inherent slowness of the traditional coding method and the challenges it brings. For example, changing things back and forth is costly, and developers are required even for the smallest changes.

### **5.8.3 Not All Potential Features Are Required**

Not all of the feature requirements identified in our literature review were relevant for the apps discussed. For example, offline data was required only by some of MADP-Manufacturing's apps, and none of the participants had built tablet-specific UI layouts. Many of the native device capabilities identified as potential requirements, such as calendar access or sensor data apart from geolocation were not required at all.

At the same time, it is quite impossible to determine all potential app requirements beforehand, as both the business and technological environments evolve. Neither MADP-Education nor MADP-Manufacturing had faced a situation where they would've had to code some significant feature by hand because the MADP toolset didn't support what they had to do, so we have no data on the potential implications of that.

### **5.8.4 Defining the App Requirements Is Hard**

All participants noted difficulties in defining the exact business problem to solve. Even for a dedicated team, getting things right from the get-go can be difficult, and both App-Maintenance and App-Engineering mentioned problems arising from insufficient understanding of end-user requirements. With a traditional coding approach, wrong design choices can be costly to fix. App-Engineering also mentioned problems arising from having to take on technical debt due to business pressure to deliver new features faster.

Making it easier and faster to develop apps doesn't remove the difficulty of the requirements gathering process. Both MADP-Education and MADP-Manufacturing felt that getting the organization fully on board to define and refine the app requirements was hard.

Since the requirements of an app start from a vague idea or business need and evolve throughout the app's lifecycle, and given that all app ideas that will be developed over the coming years have not even been thought up yet, it is impossible to define all the requirements of the complete internal mobile app fleet of an enterprise beforehand. This impossibility is further accentuated by the rapid change in the technological landscape and the need to adopt emerging technologies as they become relevant.

### **5.8.5 Custom Code Is Required**

It's clear from MADP-Education and MADP-Manufacturing's answers that the benefits of adopting a MADP grow with the number of apps developed and deployed with it. Thus, it is very important that platform shortcomings can be overcome with a high-code approach and custom extensions.

Otherwise, one or more apps would need to be developed with a hand-crafted approach separate from the MADP. In addition to the extra development necessary to implement by hand any requirements already available in the MADP environment (such as authentication or offline support), the resultant app would require further integration to be brought under the governance and lifecycle management functionalities of the platform.

App-Engineering's app case underlines the fact that as proprietary technologies exist, interfacing with them will absolutely require custom code: no MADP vendor can create platform components beforehand to integrate with systems they have no access to.

Thus, a well-rounded MADP will provide both extension points to plug in custom code to the low-code parts of their platform, as well as SDKs for hand-crafted native and hybrid development; something Gartner also highlights (Wong *et al.*, 2017). That way, the maximum level of flexibility can be achieved for the fringe cases that need it, while sacrificing as little of the platform's benefits as possible.

As neither MADP-Education or MADP-Manufacturing had run into any fundamental blockers or limitations that would've required a fully hand-crafted approach, the interview data gives no insight into the exact requirements of these SDKs.

## 6 Constructing the Framework

From our literature review and interview data, we see that the requirements for different internal enterprise apps are very diverse. Factors that affect them include corporate decisions on mobile and digitalization strategy, personal opinions of IT managers and product owners, existing systems and business processes, variations in ways of working across the company, evolving technological environments, user expectations, target user groups themselves, the business problem to be solved by the app, platform UX guidelines and likely other factors that have not come up in our research.

Furthermore, app requirements evolve throughout the projects, both as the app design works its way towards the optimal solution to the problem at hand, and as the business and technological environment around the app changes. Also, new app ideas bring with them new requirements that cannot be fully known when first starting to build mobile apps.

Digitalization, including mobile digitalization, is a process without an end, and as such constructing a comprehensive list of all potentially required features is not possible, even in our scope of just the mobile client app binary.

### 6.1 Properties of Internal Enterprise Mobile App Requirements

At the same time, it is clear that a single app will have both critical and non-critical requirements. For example, if the company mobile strategy says that employees can use both iOS and Android devices for mobile work, and that apps developed must work on all employee devices, the platform support requirements are effectively set in stone. By contrast, having a separate UI layout for tablets is something that will likely be an improvement to user experience if done correctly, but it's by no means a critical requirement that will prevent the app from fulfilling its business case.

Another property of app requirements is that very few of them are binary. Even a simple feature like “camera access” is much more than an answer to the question “can a picture be taken with the device camera or not”. The requirements of Instagram’s “camera access” are quite different from a simple form app that needs to attach pictures to a report.

Requirements also cause other requirements: if the app supports multiple screen sizes, it logically follows that the UI must scale accordingly. If the app utilizes data from multiple backends, an offline sync functionality must work against all those backends.

Finally, we can consider that certain requirements can be enterprise-wide, app-specific, or something in between. Authentication against the corporate directory is likely required by all of the internal apps developed, with perhaps a few exceptions, such as App-Engineering's USB-dongle-based user authentication. On the other hand, offline support will be a strict requirement only for apps used in areas where network connectivity is not ensured.

## 6.2 The Sanity Checklist

From our observations, we see that everything boils down to the requirements of the individual apps: even enterprise-wide requirements are such because each individual app case requires them. The only expectations arise from corporate mobile strategy: e.g. supported platforms can be decided on separately from the individual app projects and their requirements.

Constructing a framework to help with the first part of the app requirements gathering process – specifying the business problem and figuring out the first guess at a viable solution – is beyond the scope of this work. It is deep in the domain of service design, user experience research and similar fields.

Instead, we consider the moment when there is some sort of more-or-less well-defined app backlog at hand, even if it consists of just a single app that needs to be built. To aid IT decision makers in evaluating if the development approach they are considering is viable, we propose the **Sanity Checklist**.

By drawing on what we've discovered from our literature review and interview data, we have created a list of questions that should be answered in detail in the context of each app case at hand. The answers will then allow the IT decision maker to gain a better understanding of what's really required of their apps, and hopefully avoid running into hidden requirements, surprising sub-requirements and other pitfalls.

When multiple apps are analyzed, the answers can then be cross-compared to see which requirements are shared and which are unique. Furthermore, this holistic analysis of the app portfolio is likely to uncover requirements that are similar but not identical across apps. Tackled individually, they could result in slightly different implementations and thus higher maintenance overhead, but with small, non-critical tweaks to the requirements, they could be fulfilled with the same generalized solution.

Completing the feature requirements evaluation in this way will allow IT decision makers to also better evaluate if a MADP or other development approach under consideration will be a good fit for their app backlog. It can also be utilized even when a development approach or MADP has been selected, to scope out what potential challenges might be posed by new and upcoming app projects.

Due to its length, the Sanity Checklist is presented in Appendix B.

## 7 Validation of the Framework

In order to validate our Sanity Checklist framework, we examined two internal enterprise mobile app projects completed by an IT consultancy focused on enterprise digitalization.

Together with a developer who worked on the project, we answered the questions in the Sanity Checklist with regard to what was known about the app requirements in the beginning of the development.

By comparing these answers with how the development process unfolded, we can see if applying the framework before the development was started and the development approach was selected would have produced any additional value.

### 7.1 App Business Cases

We examined two app business cases, which we'll call Inspection-App and Time-Tracking-App.

#### 7.1.1 Inspection-App Business Case

Inspection-App was an app for conducting the weekly safety inspections of a category of physical assets. Previously, these inspections had been done via paper forms. If the inspection failed, workers would be prohibited from using the asset until the faults had been fixed and a new inspection had been successfully completed. Before the app, the safe-to-use status was marked physically on the asset.

There were several business drivers behind the app, with the main ones being:

1. Get rid of paper forms and create a digital archive of past inspections
2. Allow photos to be added to the inspection reports
3. Allow PDF reports of inspections to be generated and emailed to desired people
4. Allow inspection history to be accessed via the app
5. Allow a broad-level view of the status of all assets at a specific location

Users of the app were company employees, subcontractors and customers.

#### 7.1.2 Time-Tracking-App Business Case

Time-Tracking-App was an app for tracking employee working hours via Bluetooth-beacon based geofences. Additionally, the app would enforce employees to read and sign certain safety documents

when entering a geofenced location. Also, an employee should receive a push notification when entering a geofence, with different content depending on if he's signed the safety documents or not. Finally, tracked hours should be viewable with several permission levels, from an admin who sees everyone to foremen who see their team's hours to individual users, who only see their own hours.

The business driver behind the app was to enforce more accurate hours tracking with a technological solution, as well as get rid of paper forms for the safety documents. Users of the app were company employees, subcontractor foremen and subcontractor workers.

## **7.2 Development Approach and Technology Used**

Both apps were developed with a MADP's low-code toolset, with some high-code custom components created to navigate around areas where the low-code tooling could not get the job done.

Even though the MADP in question provided numerous extension points where custom code could be implemented, some aspects of the platform were not open for code-level modification. Other app functionalities could be modified if the developer knew what she was doing. However, implementing these hacks usually meant that the features in question could no longer be configured visually, but instead relied on highly specific custom code.

Additionally, for Time-Tracking-App, a third-party native SDK and its accompanying backend service were used to implement the beacon-based geofences, as the closed-source solution provided the required functionalities out of the box.

## **7.3 Evolution of the Requirements and Difficulties Faced**

We reviewed with the MADP vendor developer how these projects had unfolded, and discussed how the requirements evolved as the app progressed. We also looked at what difficulties were faced in the development process.

### **7.3.1 Inspection-App Requirements Evolution**

For Inspection-App, the requirements stayed fairly same throughout the app development process. The biggest changes came from clarifications to user roles: certain parts of the UI had to be accessible to one user group but not the other.

Another requirement that evolved was the search functionality for locating an asset, as the criteria for which user roles can see which locations, projects and assets was not well clarified in the beginning. For example, it was not clear if a subcontractor working on project A in location Y should be able to see inspection history of assets that belong to project B in the same location, or just the current situation.

However, all of the issues that emerged were more questions of configuration rather than completely new requirements; in essence, the app was completed without any significant hidden requirements emerging.

### **7.3.2 Inspection-App Difficulties Faced**

The main difficulty with Inspection-App arose from implementing certain kinds of complex app logic. In some instances, such as customizing the app UI based on logged-in user role, the drag-and-drop data binding features of the visual tool could not be used. Instead, the data had to be populated by manipulating the hybrid app's HTML5 code directly.

Luckily, the platform enabled this level of down-and-dirty manipulation, so the app could be completed without any hard blockers.

### **7.3.3 Time-Tracking-App Requirements Evolution**

The initial requirements of Time-Tracking-App were quite broad, and technological limitations brought up quite many hidden requirements as the app development progressed.

For example, the third-party geofence technology automatically calculated a random “visitor ID” based on the unique device and app binary version combo, whereas the MADP platform identified logged-in users by an ID number that was tied to the particular user account, regardless of where the login happened. Since the third-party technology provided no way to override the random visitor ID, a separate database table had to be set up to track which visitor IDs matched the actual platform user IDs.

The push notification requirements also evolved during the project, with the user requesting a “nagging notification” for unsigned safety documents: the worker should be reminded e.g. every 5 minutes to sign the safety documents when inside the geofence.

Once the feature for viewing the PDFs and signing them was completed, the customer requested a more forceful flow that would've required each PDF to be scrolled down to the bottom before it

could be signed. This was not possible with the MADP's PDF viewer, and was left out. Eventually, the decision to use PDFs for the safety documents was switched out in favor of using HTML content for each location's documents, as that could be updated more easily on the fly.

Since no wireframes were produced before the app project was started, the UI requirements were nebulous. Requirements like real-time visibility of which user is present in which location emerged, or the ability to include different safety documents for different locations.

#### **7.3.4 Time-Tracking-App Difficulties Faced**

Navigating the mismatching design philosophies of the third-party location technology vendor and the MADP proved difficult. For example, the hours data coming in from the backend was so complex in structure, that it proved impossible to utilize the visual data mapping tools of the MADP vendor. It also required a significant amount of custom code to transform the individual events data into actual tracked hours.

Each of the different user roles had the same UI, with some differences in visible data and logic. For example, when an admin clicked on a location with tracked hours, the app opened a new page showing a table view with all the workers and their hours data. For an individual worker, this needed to be scoped down to just the user herself. Likewise, admins (but not foremen and workers) needed to have access to edit the metadata for individual locations. Managing all this proved to require a bit of custom coding with the MADP utilized.

Getting the third-party location system to trigger the push notifications on geofence enters was difficult: they had their own push notification system, which couldn't be utilized with the MADP vendor's native binary wrapper, so instead the third-party system had to be made to reliably trigger notifications using the MADP's own system. There was also no ready-made implementation for the "nagging notification" requested by the customer, so it was left out.

Further issues arose from the third-party native SDK: even though a Cordova plugin was provided by the vendor, the MADP vendor's native wrapper proved incompatible with it due to e.g. mismatching Android SDK versions. Significant monkey-patching had to be done in order to get the beacon-based native geolocation functionalities working.

## 7.4 Applying the Sanity Checklist

By applying the Sanity Checklist to what was known about the projects when development started, we uncover the following.

### 7.4.1 Inspection-App Sanity Checklist Results

Hard limits in regard to supported device models, OS versions, screen sizes and so on were not explicitly mapped out beforehand; rather it was trusted that the MADP vendor supports reasonably modern iOS and Android devices.

Required UI elements were not mapped out beforehand, as it was known that the MADP provided a set of ready-made UI components plus full HTML5-based customization. UI performance was likewise trusted to be satisfactory. However, it did become as a surprise at how much code-level hacking was required to implement simple-seeming features like dynamically populated dropdown menus or multiple image attachments to reports.

The rest of the topics – company branding, data requirements, native device capabilities, offline support and user authentication – were generally covered beforehand, but the details were not discussed. For example, the requirement for the image attachments was simply “multiple images need to be attached to each report”, with the implementation details left otherwise open.

While no completely new major requirements emerged, going through the Sanity Checklist highlighted the fact that many of the requirements were brushed over with a general “I’m sure the MADP vendor has a good baseline solution for this” mind-set. Understanding the requirements of the detailed implementations beforehand could’ve enabled the developer to better evaluate if this particular MADP is the right fit for this project, and aided in setting realistic time estimates for the development. Luckily in this case, the MADP vendor’s feature set was adequate, and no critical shortcomings emerged.

### 7.4.2 Time-Tracking-App Sanity Checklist Results

For Time-Tracking-App, we find the Sanity Checklist uncovering many more hidden requirements.

Hard limits in regard to supported device models, OS versions, screen sizes and so on were not explicitly mapped out beforehand. There were no real wireframe designs, so required UI elements were not articulated. Corporate branding was not discussed in detail. User authentication was discussed, but its full effect on e.g. data visibility was not talked through beforehand.

One significant source of difficulties during the development process was the incoming data from the backend. Answering the questions related to data would've enabled the third-party location service vendor to be evaluated more thoroughly, and perhaps switched out for another solution altogether.

Similarly, the native code questions would have helped uncover the difficulties in implementing the third-party native SDK for beacon tracking.

Offline support was talked about, but left out of for the first version. Localization was not important for the customer. Other native device capabilities apart from the beacon tracking were not required.

For Time-Tracking-App, we also find that many of the evolving requirements were caused more by an unclear specification of the business case itself and its functional requirements. For example, as it was first assumed that the safety documents could be included in the app binary as static files, the initial answer to the Sanity Checklist question "Does the user need to upload binary files?" would have been "no", and no additional insight would've been gained. Similarly, the "nagging push notification" requirement might not have emerged in an initial go-through. Thus, while the Sanity Checklist appears to hit many blind spots in the initial requirements definition, it is not certain that it alone would have produced a major difference in this regard.

However, we can speculate that going through the Sanity Checklist together with the customer would have had the effect of bringing about more thorough discussion on the requirements, perhaps sparking a "you know what, actually yes, sorry I didn't realize to bring it up before" type of answer.

We also find verification for our speculation that no feature requirements list can capture the full complexity of software engineering. On the Sanity Checklist, there is no question for: "Can the platform user account ID be used to override local user IDs generated by third-party libraries?"

## **7.5 Observations on the Sanity Checklist**

From these two projects, we note that the Sanity Checklist does seem to provide additional insight into the mobile app requirements and can serve as an important tool in a holistic requirements engineering process. At the same time, the importance of understanding the user requirements, the underlying business case, the surrounding technological environment, corporate mobile strategy and other such factors cannot be overemphasized. Ideally, the Sanity Checklist would be utilized throughout the design process, including after UI wireframes have been drawn up and possible prototypes created.

### 7.5.1 Enterprise Mobile Apps Have Complex Requirements

We notice that as expected of internal enterprise mobile apps, both Inspection-App and Time-Tracking-App have very complex and large sets of requirements. However, since the development work was done by an external IT consultancy, and a full-fledged MADP was used to implement the apps, a vast majority of the feature requirements covered by the Sanity Checklist were brushed over and discussed only superficially, both in design and implementation wise.

For the customer, this is natural: they are buying a made-to-order, turn-key solution from an expert company; they don't want to understand the minutiae of every requirement. They know the main functionalities the app needs to perform, and everything else is just supposed to work as it should.

For the developer, this brushing-over can be more dangerous. In Time-Tracking-App especially, several hard limitations of the MADP in question were hit. While hacking around them with custom code would've likely been possible, it would've completely wrecked the initial time estimate. Especially when the customer is buying ultra-rapid, visual app development, it is very disheartening to suddenly learn that a seemingly simple thing like being able to queue up push notifications every five minutes requires a day or two of work. Had these limitations been laid out beforehand, the discussion would at least have been a lot easier. Also, since these limitations were discovered only after development was well underway, the cost of switching to a completely new solution was also significantly higher than at the beginning of the project.

### 7.5.2 MADPs Do a Lot of Heavy Lifting

At the same time, using a MADP clearly dodged several bullets that could've caused serious damage with a hand-crafted approach. In these cases, it didn't matter that the specific functional requirements of the device and version support or user authentication were not mapped out beforehand, as the default feature sets provided by the MADP worked very well. Implementing the multi-role, specific-to-the-app user authentication would've easily taken more time than rest of the app combined if it had to be hand-coded from scratch (not to mention the costs of backend hosting and all the difficulties that come from that).

In Time-Tracking-App's case, it can be debated if the MADP proved more harmful, though: as the third-party location service vendor had its own, closed-source, opinionated ways of doing things (like tracking users by device-specific visitor ID) that were incompatible with the MADP's philosophy, the benefit of the MADP's packaged features was diminished.

In any case, the Sanity Checklist clearly serves as a reminder of the complexity of building a modern internal enterprise mobile app. Most MADPs seem to tackle the potential requirements at least on paper, but as the app cases demonstrate, the “I’m sure the MADP vendor has got this figured out” mind-set can be quite damaging when things go wrong.

### **7.5.3 MADPs Can Provide Incomplete Feature Implementations**

Also noteworthy is that a customer might not be aware of all the latest features available through Apple and Google’s native APIs, and thus might reach a specification that leads to suboptimal user experience, compared to what it could be if all the native capabilities were leveraged.

For example, Apple provides very extensive push notification features, such as snoozing an alarm-type notification or displaying a set of custom controls when the push notification is swiped open from the lock screen (Apple, 2017h). At the same time, a MADP vendor might not provide all of these features via its visually configurable push notification admin interface. Furthermore, it will very likely not advertise features it is lacking. Indeed, Gartner (Leow, 2017a) notes that many cross-platform frameworks often do not provide day-one support for new native SDKs made available by new OS version releases, and the same applies to MADPs too.

This means that if the person defining the app requirements doesn’t know what is possible on the native code level, some functionalities that would prove useful down the road might be glossed over. If the evaluation is done on the level of ticking the box of “platform has push notification support”, these advanced requirements are not captured.

### **7.5.4 Deficiencies of the Sanity Checklist**

Some of the functionalities found in the Sanity Checklist were not found in either app, so we cannot evaluate if how well the questions under categories like offline support or localization would have worked in uncovering additional hidden requirements.

We also know from the IT consultancy that there was no big platform buy-in for either customer, but rather both apps were standalone projects and such the consequences of a suboptimal platform choice would have been limited. It will be future work to verify if the Sanity Checklist would be more useful when applied to an enterprise’s entire upcoming app backlog rather than a single app.

Also, it should be noted that checklists in general can give a skewed view of the subject: if everything checks green on the checklist, it can lull the respondent into a false sense of security that all aspects of the problem are covered, when in reality any checklist is naturally just a subset of the whole problem

space. However, this only serves to highlight the fact that the Sanity Checklist should not be used alone.

## 8 Discussion and Conclusions

Throughout this work, we have explored the challenges of internal enterprise mobile app development. We have uncovered some inherent features of these apps, as well as identified some key requirements that are likely to come across in typical projects.

We've come to the conclusion that it would be practically impossible to construct any formal framework that would capture and collate app feature requirements into categories and assign them generalized values such as "likely to be a critical requirement" or "unlikely to be a critical requirement", due to the complexity of both the apps themselves and the business and technological environment surrounding them.

We also see that evaluating if certain internal enterprise mobile app requirements are relevant or not needs to happen holistically, in the context of the enterprise in question, the business problem the app is meant to solve, the app design and any environmental factors at play.

That said, a holistic list of potential feature requirements can be created, and this is what we've done with our Sanity Checklist. By applying the Sanity Checklist to two real-life internal enterprise mobile app projects, we've observed that it provides additional insights to the requirements gathering process.

It seems especially useful when evaluating if a development approach and toolset under consideration is viable for the project, as it can uncover sub-requirements and other nuances that might be otherwise brushed over in the initial design phase of "the app needs to do X and Y".

In conclusion, the Sanity Checklist is not a be-all-end-all solution, but used as part of a holistic requirements gathering and engineering process, can be a very useful tool.

### 8.1 Avenues for Future Research

The usefulness of the Sanity Checklist should be further validated with a sufficiently large sample of internal enterprise mobile app projects, to reinforce its value and ensure the requirements listed apply outside the research done in this work. Additionally, the value of the Sanity Checklist in a situation where different MADP vendors and development approaches are evaluated by an enterprise should be investigated.

Further research should be conducted to also expand the Sanity Checklist, both to capture more fully the requirements on the mobile client app binary, as well as to cover the other requirement categories

scoped out of this work, such as backend, governance, lifecycle management and so on. As the mobile app development landscape changes so rapidly, interviewing enterprise app development professionals would be a good course of action to gather relevant and current data.

The Sanity Checklist should also be expanded to cover the B2C and B2B app categories scoped out of this work, to see how their requirements differ.

Mobile Application Development Platforms in themselves would also merit additional research. In particular, it would be interesting to research and categorize their strengths and weaknesses in detail, so that enterprise IT decision makers would have a systematic approach to evaluating them.

Another avenue of research would be to consider the development process affordances provided by both the high-code and low-code MADP approaches. Both the critical functional capabilities of these platforms as well as the components of their development experience should be studied. An additional avenue of research is the role of “citizen developers” in enterprise mobile app development: how to enable employees without real coding skills to create viable apps?

The entire process of mobile app requirements gathering in the enterprise context would merit from focused research: how does the process differ from other software requirements gathering and engineering?

All in all, the whole subject of internal enterprise mobile app development is not at all a focused line of research at the moment, and as the technologies are maturing and the speed of change somewhat slowing down, there’s ample space for foundational academic work to be done.

## Bibliography

van der Aalst, W. M. P. (2013) 'Business Process Management : A Comprehensive Survey', *ISRN Software Engineering*, 2013, pp. 1-37.

Achimugu, P. *et al.* (2014) 'A systematic literature review of software requirements prioritization research', *Information and Software Technology*. Elsevier B.V., 56(6), pp. 568-585. doi: 10.1016/j.infsof.2014.02.001.

Agarwal, R. *et al.* (2010) 'The digital transformation of healthcare: Current status and the road ahead', *Information Systems Research*, 21(4), pp. 796-809.

Agarwal, S., Starobinski, D. and Trachtenberg, A. (2002) 'On the scalability of data synchronization protocols for PDAs and mobile devices', *IEEE Network*, 16(4), pp. 22-28. doi: 10.1109/MNET.2002.1020232.

Andal-Ancion, A., Cartwright, P. A. and Yip, G. S. (2003) 'The digital transformation of traditional business', *MIT Sloan Management Review*, 44(4), pp. 34-41.

Apple (2017a) *Adaptive User Interfaces*. Available at: <https://developer.apple.com/design/adaptivity/> (Accessed: 30 October 2017).

Apple (2017b) *Adaptivity and Layout, Human Interface Guidelines*. Available at: <https://developer.apple.com/ios/human-interface-guidelines/visual-design/adaptivity-and-layout/> (Accessed: 23 October 2017).

Apple (2017c) *Adopting Multitasking Enhancements on iPad*. Available at: <https://developer.apple.com/library/content/documentation/WindowsViews/Conceptual/AdoptingMultitaskingOniPad/> (Accessed: 23 October 2017).

Apple (2017d) *Configuring Your Xcode Project for Distribution, App Distribution Guide*. Available at: <https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/ConfiguringYourApp/ConfiguringYourApp.html> (Accessed: 30 October 2017).

Apple (2017e) *Core NFC*. Available at: <https://developer.apple.com/documentation/corenfc> (Accessed: 30 October 2017).

Apple (2017f) *Image Size and Resolution, Human Interface Guidelines*. Available at: <https://developer.apple.com/ios/human-interface-guidelines/icons-and-images/image-size-and-resolution/> (Accessed: 30 October 2017).

Apple (2017g) *iOS 11*. Available at: <https://developer.apple.com/ios/> (Accessed: 30 October 2017).

Apple (2017h) *Local and Remote Notifications Overview*. Available at: [https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/index.html#//apple\\_ref/doc/uid/TApp-Engineering0008194-CH3-SW1](https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/index.html#//apple_ref/doc/uid/TApp-Engineering0008194-CH3-SW1) (Accessed: 27 November 2017).

Armando, A. *et al.* (2008) 'Formal Analysis of SAML 2.0 Web Browser Single Sign-on: Breaking the SAML-based Single Sign-on for Google Apps', *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering*, pp. 1-10. doi: 10.1145/1456396.1456397.

Baker, V. L. (2016) 'Top Considerations and Concerns When Developing a Mobile App Architecture', *Gartner*.

Bendixen, M., Bukasa, K. A. and Abratt, R. (2004) 'Brand equity in the business-to-business market', *Industrial Marketing Management*, 33(5), pp. 371-380. doi: 10.1016/j.indmarman.2003.10.001.

Brooks, F. P. (1987) 'No Silver Bullet – Essence and Accidents of Software Engineering', *IEEE Computer*, 20(4), pp. 10-19.

Burmann, C. and Zeplin, S. (2005) 'Building brand commitment: A behavioural approach to internal brand management', *Journal of Brand Management*, 12(4), pp. 279-300. doi: 10.1057/palgrave.bm.2540223.

Burmann, C., Zeplin, S. and Riley, N. (2009) 'Key determinants of internal brand management success: An exploratory empirical analysis', *Journal of Brand Management*, 16(4), pp. 264-284. doi: 10.1057/bm.2008.6.

Charland, A. and LeRoux, B. (2011) 'Mobile Application Development: Web vs . Native', *Communications of the ACM*, 54, pp. 0-5. doi: 10.1145/1941487.

Chung, S., Lee, K. Y. and Choi, J. (2015) 'Exploring digital creativity in the workspace: The role of enterprise mobile applications on perceived job performance and creativity', *Computers in Human Behavior*. Elsevier Ltd, 42, pp. 93-109. doi: 10.1016/j.chb.2014.03.055.

Clark, W. (2003) 'Two Choices Prevail for Mobile Application Platforms', *Gartner*.

Clark, W. and King, M. J. (2008) 'Magic Quadrant for Mobile Enterprise Application Platforms', *Gartner*.

Dalmasso, I. *et al.* (2013) 'Survey, comparison and evaluation of cross platform mobile application development tools', *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, (April 2015), pp. 323–328. doi: 10.1109/IWCMC.2013.6583580.

Dinh, H. T. *et al.* (2013) 'A survey of mobile cloud computing: architecture, applications, and approaches', *Wireless Communications and Mobile Computing*, 13(18), pp. 1587–1611. doi: 10.1002/wcm.1203.

Dogtiev, A. (2017) *App Download and Usage Statistics 2017, Business of Apps*. Available at: <http://www.businessofapps.com/data/app-statistics/> (Accessed: 13 November 2017).

El-Kassas, W. S. *et al.* (2017) 'Taxonomy of Cross-Platform Mobile Applications Development Approaches', *Ain Shams Engineering Journal*. Ain Shams University, 8(2), pp. 163–190. doi: 10.1016/j.asej.2015.08.004.

Fernando, N., Loke, S. W. and Rahayu, W. (2013) 'Mobile cloud computing: A survey', *Future Generation Computer Systems*. Elsevier B.V., 29(1), pp. 84–106. doi: 10.1016/j.future.2012.05.023.

Fortune (2017) *Fortune 500 Companies 2017*. Available at: <http://fortune.com/fortune500/list/> (Accessed: 9 November 2017).

Francese, R. *et al.* (2013) 'Supporting the development of multi-platform mobile applications', *Proceedings of IEEE International Symposium on Web Systems Evolution, WSE*, pp. 87–90. doi: 10.1109/WSE.2013.6642422.

Gartner (2017a) *Digitalization, Gartner IT Glossary*. Available at: <https://www.gartner.com/it-glossary/digitalization/> (Accessed: 13 November 2017).

Gartner (2017b) *Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017, Gartner Newsroom*. Available at: <https://www.gartner.com/newsroom/id/3725117> (Accessed: 1 November 2017).

Gartner (2017c) *Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2017, Statista*. Available at: <https://www.statista.com/statistics/266136/global-market-share->

held-by-smartphone-operating-systems/ (Accessed: 26 October 2017).

Gartner (2017d) *Number of smartphones sold to end users worldwide from 2007 to 2016 (in million units)*, *Statista*. Available at: <https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/> (Accessed: 1 November 2017).

Giguère, E. (2001) 'Mobile data management: challenges of wireless and offline data access', *Proceedings 17th International Conference on Data Engineering*, pp. 227–228. doi: 10.1109/ICDE.2001.914831.

Google (2015) *Android 5.1: Unwrapping a new Lollipop update*, *Official Android Blog*. Available at: <https://android.googleblog.com/2015/03/android-51-unwrapping-new-lollipop.html> (Accessed: 30 October 2017).

Google (2017a) *Create Resizable Bitmaps (9-Patch files)*, *Android Studio User Guide*. Available at: <https://developer.android.com/studio/write/draw9patch.html> (Accessed: 30 October 2017).

Google (2017b) *Dashboards*. Available at: <https://developer.android.com/about/dashboards/index.html> (Accessed: 30 October 2017).

Google (2017c) *Supporting Multiple Screens*. Available at: [https://developer.android.com/guide/practices/screens\\_support.html](https://developer.android.com/guide/practices/screens_support.html) (Accessed: 23 October 2017).

Google (2017d) *WebView for Android*. Available at: <https://developer.chrome.com/multidevice/webview/overview> (Accessed: 30 October 2017).

Harris, M. A. and Patten, K. P. (2014) 'Mobile device security considerations for small- and medium-sized enterprise business mobility', *Information Management & Computer Security*, 22(1), pp. 97–114. doi: 10.1108/IMCS-03-2013-0019.

Hartmann, G., Stead, G. and DeGani, A. (2011) 'Cross-platform mobile development', *Mobile Learning Environment, Cambridge*, 16(9), pp. 158–171.

He, Z., Bustard, D. W. and Liu, X. (2002) 'Software internationalisation and localisation: practice and evolution', *Proc. PPPJ*, pp. 89–94.

Heitkötter, H., Hanschke, S. and Majchrzak, T. A. (2012) 'Comparing Cross-Platform Development Approaches for Mobile Applications', *Proceedings of the 8th International Conference on Web Information Systems and Technologies*, 140, pp. 299–311. doi: 10.5220/0003904502990311.

- Holzinger, A., Treitler, P. and Slany, W. (2012) 'Making apps useable on multiple different mobile platforms: On interoperability for business application development on smartphones', *Multidisciplinary research and practice for information systems*. Springer, pp. 176-189.
- Hoos, E. *et al.* (2014) 'Improving Business Processes through Mobile Apps an Analysis Framework to Identify Value-Added App Usage Scenarios', *Proceedings of the 16th International Conference on Enterprise Information Systems (ICEIS)*, pp. 71-82.
- Huang, X. *et al.* (2014) 'Robust multi-factor authentication for fragile communications', *IEEE Transactions on Dependable and Secure Computing*, 11(6), pp. 568-581. doi: 10.1109/TDSC.2013.2297110.
- Inayat, I. *et al.* (2015) 'A systematic literature review on agile requirements engineering practices and challenges', *Computers in Human Behavior*. Elsevier Ltd, 51, pp. 915-929. doi: 10.1016/j.chb.2014.10.046.
- Jain, N. (2014) 'Review of Different Responsive CSS Front-End Frameworks', *Journal of Global Research in Computer Science*, 5(11), pp. 5-10.
- Joorabchi, M. E., Mesbah, A. and Kruchten, P. (2013) 'Real Challenges in Mobile App Development', *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 15-24. doi: 10.1109/ESEM.2013.9.
- Karahanna, E. *et al.* (1999) 'Information Technology Adoption Across Time : A Cross-Sectional Comparison of Pre- Adoption and Post-Adoption Beliefs Stable URL ', *MIS Quarterly*, 23(2), pp. 183-213. doi: 10.2307/249751.
- Kuusinen, K. and Mikkonen, T. (2014) 'On Designing UX for mobile enterprise apps', *Proceedings - 40th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2014*, (August), pp. 221-228. doi: 10.1109/SEAA.2014.17.
- Latif, M. *et al.* (2017) 'Review of mobile cross platform and research orientations', *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems, WITS 2017*, pp. 0-3. doi: 10.1109/WITS.2017.7934674.
- Leffingwell, D. and Widrig, D. (2000) *Managing software requirements: a unified approach*. Addison-Wesley Professional.

- Leow, A. (2017a) 'Hype Cycle for Mobile Applications and Development, 2017', *Gartner*.
- Leow, A. (2017b) 'The Key Fundamentals Required to Scale Mobile App Development', *Gartner*.
- Leow, A. and Baker, V. L. (2017) 'Survey Analysis: The Mobile App Development Trends That Will Impact Your Enterprise in 2017', *Gartner*.
- Matt, C., Hess, T. and Benlian, A. (2015) 'Digital Transformation Strategies', *Business and Information Systems Engineering*, 57(5), pp. 339-343. doi: 10.1007/s12599-015-0401-5.
- Microsoft (2017) *Authorize access to web applications using OAuth 2.0 and Azure Active Directory*. Available at: <https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-protocols-oauth-code> (Accessed: 22 October 2017).
- Mixpanel (2017) *iOS 11 adoption*. Available at: [https://mixpanel.com/trends/#report/ios\\_11](https://mixpanel.com/trends/#report/ios_11) (Accessed: 30 October 2017).
- Nayebi, F., Desharnais, J.-M. and Abran, A. (2012) 'The state of the art of mobile application usability evaluation', *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, (May), pp. 1-4. doi: 10.1109/CCECE.2012.6334930.
- Niu, N. *et al.* (2014) 'Analysis of Architecturally Significant Requirements for Enterprise Systems', *Ieee Systems Journal*, 8(3), pp. 850-857.
- Nuseibeh, B. and Easterbrook, S. (2000) 'Requirements engineering', *Proceedings of the conference on The future of Software engineering - ICSE '00*, pp. 35-46. doi: 10.1145/336512.336523.
- Occhino, T. (2015) *React Native: Bringing modern web techniques to mobile, Facebook Code*. Available at: <https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/> (Accessed: 26 October 2017).
- OpenSignal (2015) *Android Fragmentation Report August 2015*. Available at: <https://opensignal.com/reports/2015/08/android-fragmentation/> (Accessed: 26 October 2017).
- Paksula, M. (2017) 'Interview on Challenges in Enterprise App Development'. Helsinki. Interviewed by H. Sarsa on 06 October 2017.
- Palmieri, M., Singh, I. and Cicchetti, A. (2012) 'Comparison of cross-platform mobile development tools', *2012 16th International Conference on Intelligence in Next Generation Networks, ICIN 2012*,

pp. 179–186. doi: 10.1109/ICIN.2012.6376023.

Pashalidis, A. and Mitchell, C. J. (2003) ‘A taxonomy of single sign-on systems’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2727 LNCS, pp. 249–264. doi: 10.1007/3-540-45067-X\_22.

Patton, M. Q. (2005) *Qualitative research*. John Wiley & Sons, Ltd.

Pawar, V. (2016) *Mobile Authentication Done Right: Secure Single Sign-On for Mobile Apps*, *MobileIron Blog*.

Perchat, J., Desertot, M. and Lecomte, S. (2013) ‘Component based framework to create mobile cross-platform applications’, *Procedia Computer Science*. Elsevier B.V., 19, pp. 1004–1011. doi: 10.1016/j.procs.2013.06.140.

Piejko, P. (2017) *Device fragmentation still an Issue*, *Device Atlas*. Available at: <https://deviceatlas.com/blog/device-fragmentation-still-issue> (Accessed: 29 October 2017).

Plummer, D. and Hill, J. (2014) ‘Digital Business Technologies Are Changing the Nature of Change’, *Gartner*.

Pohl, K. (2010) *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer Publishing Company.

Ribeiro, A. and da Silva, A. R. (2012) ‘Survey on cross-platforms and languages for mobile apps’, *Proceedings - 2012 8th International Conference on the Quality of Information and Communications Technology, QUATIC 2012*, pp. 255–260. doi: 10.1109/QUATIC.2012.56.

Rymer, J. R. (2017) *The Forrester Wave™ : Low-Code Development Platforms For AD & D Pros , Q4 2017*.

Sanaei, Z. *et al.* (2014) ‘Heterogeneity in mobile cloud computing: Taxonomy and open challenges’, *IEEE Communications Surveys and Tutorials*, 16(1), pp. 369–392. doi: 10.1109/SURV.2013.050113.00090.

Shen, H. *et al.* (2004) ‘Integration of business modelling methods for enterprise information system analysis and user requirements gathering’, *Computers in Industry*, 54(3), pp. 307–323. doi: 10.1016/j.compind.2003.07.009.

- Smutný, P. (2012) 'Mobile development tools and cross-platform solutions', *Proceedings of the 2012 13th International Carpathian Control Conference, ICC 2012*, (February), pp. 653–656. doi: 10.1109/CarpathianCC.2012.6228727.
- Sobers, A. (2014) 'BYOD and the Mobile Enterprise'.
- Sørensen, C. *et al.* (2008) 'Exploring enterprise mobility: lessons from the field', *Information Knowledge Systems Management*, 7, pp. 243–271.
- Sun, S.-T. and Beznosov, K. (2012) 'The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems', *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, pp. 378–390. doi: 10.1145/2382196.2382238.
- Unhelkar, B. and Murugesan, S. (2010) 'The enterprise mobile applications development framework', *IT Professional*, 12(3), pp. 33–39. doi: 10.1109/MITP.2010.45.
- Vincent, P. *et al.* (2017) 'Magic Quadrant for Enterprise High-Productivity Application Platform as a Service', *Gartner*.
- Wasserman, A. I. (2010) 'Software engineering issues for mobile application development', *ACM Transactions on Information Systems*, pp. 1–4. doi: 10.1145/1882362.1882443.
- Webster, J. and Watson, R. T. (2002) 'Analyzing the Past to Prepare for the Future : Writing a Literature Review', *MIS Quarterly*, 26(2).
- Wong, J. *et al.* (2016) 'Market Guide for Rapid Mobile App Development Tools', *Gartner*.
- Wong, J. *et al.* (2017) 'Magic Quadrant for Mobile Application Development Platforms', *Gartner*.
- Xanthopoulos, S. and Xinogalos, S. (2013) 'A comparative analysis of cross-platform development approaches for mobile applications', *Proceedings of the 6th Balkan Conference in Informatics on - BCI '13*, p. 213. doi: 10.1145/2490257.2490292.
- Yadav, D. S. and Kanchan, D. (2016) 'Mobile Cloud Computing Issues and Solution Framework', *International Research Journal of Engineering and Technology*, 11, pp. 2395–56.
- Zhu, K. *et al.* (2006) 'Innovation diffusion in global contexts: determinants of post-adoption digital transformation of European companies', *European Journal of Information Systems*, 15(6), pp. 601–616. doi: 10.1057/palgrave.ejis.3000650.

## Appendix A: Interview Questions

We start by reiterating our definition of internal mobile apps, to ensure we are talking about the same subject. Then, we move on to some general questions.

- What has your role been in the internal mobile app projects?
- What kind of role do internal mobile apps have in your corporate strategy?
- What is the main motivation to build the apps?
- What kind of custom internal mobile apps have you built yourselves or outsourced as a company?
- How many mobile apps?

Next, we ask about the app projects themselves. If the interviewee has been involved in multiple app projects, the answers will be collected separately for each app.

- What does the app you've created do?
- Who have been the app users?
- What has been the business case?
- What technologies or platforms have been used to build the mobile app?
- Has the app been a separate project, or built as part of a larger digitalization undertaking?
- What kind of team built the app?
  - Team size
  - Internal developers vs. consultants
  - Role of non-technical team members
- How long did the project take?
  - What took the most time?
- How have the app requirements been gathered?
- What has been easy about the requirements gathering process?
- What has been hard about the requirements gathering process?
- Have there been any hidden requirements that weren't initially considered?
- How have the requirements evolved through the app project?
- In general, what went well in the project?
- What kind of problems did you run into?
- What features were especially easy to implement?

- What features were especially hard to implement?

We then move on to the feature questions, again addressing each app project under examination separately.

Under each section, we will further ask if any of the required features were particularly hard or easy to implement, and what specific technologies were used in the implementation.

- **App Types**
  - What app types have you developed?
  - What influenced the decision?
- **Platform, Device Model and Operating System Version Fragmentation**
  - What platforms, device models and mobile OS versions was the app required to run on?
  - How were the device compatibility requirements decided?
  - What kind of testing did you do to ensure the app works on all the targeted combinations?
- **Battery Power Consumption**
  - Did you face any issues with high battery power consumption?
  - How were these issues tackled?
- **User Interface and Logic**
  - UI Performance
    - What type of UI performance issues have you faced?
    - How did you solve them?
  - Company Branding
    - In what ways was the app made look like your brand?
    - Are there any specific graphical or design guidelines for internal mobile apps?
  - Localization
    - Was the app localized for different languages/locales? Which ones?
    - What aspects of the app were localized?
    - How was the localization implemented?
- **Screen Size and Resolution**
  - What different screen sizes did the app target?
  - How was the UI adapted to work on all supported screen sizes?
- **Interacting with Data**

- What kind of business data does the app utilize?
- How is data synchronized between the app and the backend?
- What kind of UI is there for the user to view and interact with the data?
- **Native Device Capabilities**
  - What kind of native device capabilities does the app utilize?
- **User Authentication**
  - How was user authentication handled in the app?
  - What kind of different user roles are there?
  - How are they determined?
  - How did the different roles affect the functionalities of the end app?
- **Offline Data**
  - Did the app have offline support? To what extent?
- **Any other critical features?**

We then discuss the future roadmap of the companies.

- What new features or other apps are on the roadmap?
- How clear is the roadmap?

Finally, we do some reflection based on the past projects.

- What have you learned about building internal mobile apps from these projects?
- What would you do differently, if you could start the projects again?
- What prevents you from building more apps?
- What do you consider to be the limiting factors for app development speed?

## Appendix B: The Sanity Checklist

### **What mobile app platforms, device models and OS versions must the app support?**

Consider the device policies of your organization and the app target users. What devices do they use? Is it viable to purchase dedicated devices to run this app? Are there any core features that function differently across the targeted devices?

### **What screen sizes and device orientations must the app support?**

Is it enough that the UI simply scales, or are differently structured layouts required?

### **What UI elements are required?**

Based on your app design wireframes or the basic functionalities such as CRUD operations on data or viewing shipments on a map, what are the individual UI components required by this app? Consider things like lists, details views, input fields, buttons, checkboxes, toggle switches, dropdowns, maps, charts and so forth.

### **What kind of UI performance is required?**

Can the app afford to be a little slow, or is fully-native, blazing-fast UI performance a must? Do the UI elements specified in the app design require e.g. native UI animations to be realized in a satisfactory way?

### **What kind of company branding is required?**

Are fonts, colors and a logo enough, or are there more detailed guidelines that should be followed?

### **What kind of localization support is required?**

What languages need to be supported? Are non-Latin alphabets required, or right-to-left writing? Does each localized version need to fetch different data from the backend? Are there images, audio and video that need to be localized, too? Does the app need to account for different timezones and handle different date and time formats? Are there any other locale-specific considerations apart from language?

**What kind of business data does the app need to interact with?**

Is data just viewed or also created, updated or deleted? Does it come from an existing backend?

What kind of authentication exists for the backend? How often must the data be refreshed? Does some database need to be bundled with the app binary itself? What sort of local caching is required?

What parts of app data (such as static texts or dropdown selections) should be changeable without having to publish a new app binary?

**What kind of binary data files does the app need to interact with?**

Does the user need to download and view binary files? What kind? Audio, video, photo, PDF, other? Do they need to be stored on the device after downloading? Does the user need to upload binary files? Do some of these files need to be bundled with the app binary itself?

**What kind of other data does the app need to interact with?**

E.g. data from Bluetooth, NFC, USB peripherals, phone sensors, streaming data from the backend.

**Is camera access is required? What kind?**

Do photos need to be edited after taking? Is video capture required?

**Is Barcode/QR code scanning required?**

What barcode/QR code standards?

**Is Bluetooth required?**

What kind of devices does the app need to interface with? What is the Bluetooth protocol used?

**Is NFC required?**

Is it enough to read tags, or must information be transmitted both ways? What kind of tags must be supported?

**Is geolocation required?**

Is background location required, or is it enough for the position to update just when the app is running?

### **Are push notifications required?**

Is deep linking required, i.e. that the app opens to a specific view from the notification? Are local notifications required, i.e. scheduled by the app itself instead of the backend? Does the app need to provide custom controls when accessing the push notifications from the lock screen?

### **Are other native capabilities required?**

E.g. device contacts and calendar, Apple Touch ID, sensors like accelerometer and compass, and so on.

### **What kind of user authentication is required?**

Is it always against the same corporate directory? What system? How are external users handled? Is multi-factor authentication required? Does the session need to be shared across different apps? How are non-authorized logins handled? How are sessions revoked when users are removed from the corporate directory?

### **What kind of user roles are required?**

Are these roles available in the corporate directory? Are there app-specific roles that do not map directly to existing roles?

### **How do the user roles affect app functionality?**

Are different data access permissions required for different user roles? How granular do the permissions need to be? Does the UI need to adapt based on the user roles?

### **Is offline support required?**

Which of the CRUD operations are required for offline data? What kind of conflict resolution is required? Do binary files like photos need to be cached offline and then uploaded when connectivity resumes? Does the online sync need to trigger automatically, manually or both? Are there other functionalities than data that need to work offline?

**What fully custom native code is required by the app?**

Are there some functionalities where existing frameworks, libraries or components do not exist, such as interfacing with proprietary control systems? Do the native APIs and capabilities leveraged function the same across all targeted devices?