Aalto University
School of Science
Master's Programme in ICT Innovation

Alessio Spallino

Native versus hybrid mobile application development for professional membership services

Master's Thesis
Espoo, July 30, 2018

Supervisors:	Professor Mario Di Francesco, Aalto University
	Professor Anna Perini, University of Trento
Advisor:	Tapio Rantanen M.Sc. (Tech.)

Aalto University
School of Science
Master's Programme in ICT Innovation

ABSTRACT OF
MASTER'S THESIS

Author:	Alessio Spallino		
Title: Native versus hybrid mobile application development for professional membership services			
Date:	July 30, 2018	**Pages:**	70
Major:	Software and Service Architectures	**Code:**	SCI3082
Supervisors:	Professor Mario Di Francesco Professor Anna Perini		
Advisor:	Tapio Rantanen M.Sc. (Tech.)		

Mobile development is a necessity for every modern business. Nowadays, many solutions to implement mobile applications are available in the market. The largest amount of applications are purely native, meaning that native development is still the preferred development solution. However, the advent of cross-platform mobile development and hybrid mobile development technologies is starting to change the software development process in companies. In fact, several major mobile applications including Facebook, Instagram, Skype, Bloomberg, Uber, Tesla, and Soundcloud are not native; these mobile applications are implemented using frameworks that combine native code with platform-independent code. These frameworks are part of the Hybrid mobile development solutions.

In many cases, businesses are trying to offer their services through mobile applications, but most of the times these applications are not complicated and do not require very innovative development. For this reason, software companies are keen to explore new mobile application development solutions that are cost-efficient.

This thesis presents the complete renovation process of a mobile application for professional membership services currently available in the Apple app store and Google play store. It starts showing the requirements engineering process that helped to define the software requirements. After that, this thesis details the implementation process of two native mobile applications and one in React Native. The three applications follow the same design constraints required by the customer. Finally, this thesis presents a performance comparison between the purely Native mobile applications and the React Native mobile application.

Keywords:	Mobile application development, Native applications, Hybrid applications, React Native, Performance, requirements engineering, applications, iOS, Android
Language:	English

Acknowledgements

I would first like to thank my thesis supervisor, Dr. Mario Di Francesco, who was always available to help me whenever I ran into a trouble spot or had a question about my research or writing. He never made me feel under pressure, without him I would never have managed to finish on time.

I would also like to thank my second thesis supervisor Prof. Anna Perini, research scientist at Fondazione Bruno Kessler and Professor at University of Trento. I am gratefully indebted to her for helping me with the requirements engineering chapter.

I would also like to thank the Insinööriliitto engineers association for giving me the freedom to write my MSc thesis using material from the implementation of their mobile application.

I am also grateful to Tapio Rantanen, Software leader at Eficode Oy. I am extremely thankful for the help he gave me to conclude this thesis with excellent results.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Espoo, July 30, 2018

Alessio Spallino

Abbreviations and Acronyms

Virtual DOM	Virtual Document Object Model
SR Model	Strategic Rationale Model
SD Model	Strategic Dependency Model
UML	Unified Modeling Language
SRS	Software Requirement pecification
RSS	Really Simple Syndication
AHP	Analytic Hierarchy Process
CPU	Central Processing Unit
IoT	Internet of Things
SDK	Software Development Kit
IDE	Integrated Development Environment
JS	Javascript
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
CLI	Command Line Interface
XML	eXtensible Markup Language
GPU	Graphics Processing Unit
MiB	Mebibyte

Contents

List of Figures

Chapter 1

Introduction

Despite the expansion of new and diverse technology niches, mobile applications and smartphones have still an important role in people's life. According to Ericsson, in five years' time, 70 percent of the world's population will use smartphones; an unimaginable measure that explains how central these devices are becoming to how we communicate with each other and perform daily activities [12].

Technology trends are continually changing; the Internet of Things (IoT), Artificial Intelligence, Augmented and Virtual Reality, and Chatbots are prevalent, but in most cases the platform that incorporates these technologies is always a mobile device. Nowadays, people use smartphones for more complex operations, due to the reduced difference between a computer and a smartphone. This is justified by the rapid increase in the performance, maturation of operating systems, and improvements in connectivity and storage [11]. For all these reasons, users expect all businesses to offer their services through mobile applications. Many small businesses we interact in our everyday life have their dedicated mobile app. For marketing reasons, having a mobile application is the fastest way to create a direct channel with users. Services like push notifications give the possibility to get even closer to direct interaction with the users, easily reminding about their services whenever they need to. Moreover, people do not expect just apps, but they pay attention to the mobile experience, the functionalities, and the design. Due to the high demand for mobile applications, the approaches to develop mobile applications are constantly growing. Indeed, developers and companies need to implement applications in a short time being able to propose new and compelling solutions that impress customers.

1.1 Research topic

Insinööriliitto is the Finnish Association of Engineers. This association offers the membership of the engineering union that provides several benefits. With this membership, Insinööriliitto aims to help engineers on their professional development. Members are free to ask questions about employment issues, career planning, or paycheck. Moreover, the membership offers a range of educational supplies and personal counseling, whenever the engineer is still a student or has already a working career. Insinööriliitto has a website where the user can quickly register to the union and check all their information. Moreover, after the registration, engineers get a physical membership card that guarantees several discounts for different services such as fuel and legal services.

The idea of Insinööriliitto is to expand the use of their services throughout their mobile application. The association already has a mobile application, but their idea is to renovate it with new features. Insinööriliitto does not want to create a new application from scratch; instead they intend to develop the new features on top of the old application. The mobile application is available for both Apple and Google platforms, and the structure is identical. Since the applications look the same, there would be several options on the market to develop the same application using less time and resources. These options offer a mobile development environment to implement just one application that works on different platforms, with small platform fixes. These options are called hybrid mobile development solutions[1].

The company that has agreed with Insinööriliitto for the implementation of the new app wants to extend the range of mobile development solutions that they offer to clients. Nowadays, they develop an application for the Apple application store and another one for the Google application store. If cross-platform mobile development solutions and hybrid mobile development solutions prove their reliability, the company might take seriously in consideration to add them in the technology stack they offer to customers.

1.2 Research questions and methodology

Based on the research topic presented before, this thesis aims at answering the following research questions.

[1]Hybrid applications combine the functionality of native apps with the cross-platform compatibility of web apps. The hybrid apps are then hosted inside a native Android or iOS container so that the same code can be deployed on multiple mobile platforms

1. Do hybrid mobile application development solutions have the potential to replace native mobile development solutions?
2. What are the differences in performance between hybrid mobile development solutions and native mobile development solutions?

This research goes through all the development process of a native mobile application. Moreover, it presents the implementation of the Insinööriliitto mobile application with a hybrid mobile development framework called React Native. After that, the thesis details the performance comparison between the two mobile applications implemented with the two different solutions.

1.3 Structure of the Thesis

The rest of the thesis is organized as follows. Chapter 2 introduces the background of mobile application programming, explaining several technologies used during the implementation and performance comparison. Chapter 3 presents the requirements engineering process, namely, how to find the software requirements starting from a set of goals of the stakeholders. Chapter 4 provides a detailed explanation of the implementation process for the Native mobile applications and the React Native mobile application. Chapter 5 presents the performance comparison and discusses the results obtained during the implementation and performance comparison process, trying to analyze the results and provide the answers to the research questions.

Chapter 2

Mobile application programming

This chapter presents different mobile application development solutions currently available in the market. Furthermore, It introduces to a few helpful technologies necessary to understand the next chapters. The last section of the chapter presents which mobile application solution is used by this research to perform the comparison between native and hybrid mobile applications.

With the rapid increase in the number of platforms available to develop mobile applications, the whole process of deciding which mobile development solution to use has become challenging. Companies and organizations in need of developing a mobile application must take into consideration important aspects, such as time and cost. Nowadays, the standard practice is still to develop different mobile applications from scratch for every targeted platform (e.g., iOS and Android)[35]. However, the mobile application development world is continuously evolving, with very frequently new mobile development technologies joining the market. E.g., The most recently released technology is Flutter, another free and open-source solution presented by Google that is now fast growing up.

The goal of this chapter is to introduce the four mobile development application solutions available on the market, by focusing on the pros and cons of each technology. The following are the four alternatives.

- Platform-specific native applications
- Progressive web applications
- Hybrid mobile applications
- React Native

2.1 Platform-specific native applications

A native application is a software program developed for use on a particular platform (iOS, Android, Windows). Building completely native mobile applications represent the best way to ensure an in-depth integration of the app with the mobile operating system. However, building platform-specific mobile applications requires a different application for each native platform. Moreover, each target platform requires different expertise with the programming languages and Software Development Kits (SDKs) [23]. The code is different for each target platform; this means that there is no way to share the same code. The UI entirely fit the specific platforms, a key point that guarantees everything to work fluidly. The following are the two most widely used software for writing native mobile applications for the leading mobile platforms.

- **Android Studio.** Integrated development environment (IDE) based on IntelliJ IDEA of JetBrains[1]. The goal of Android Studio is to help developers to be more productive and assist during the development process. Android Studio has built-in support for Google services, such as Google app engine and Firebase Cloud Messaging. Android Studio also provides extended support to develop for different devices like smartphones, wearables, and Android TV. Uploading an Android application on the Google play store requires only a few steps: Android Studio gives the possibility to download the .apk file. The user then needs to upload the file on the personal Google play store account.

- **Xcode.** Apple's official Integrated development environment (IDE) for Mac and iOS developers. It is the only software that allows developers to write code that runs on Apple products. Similar to Android Studio, Xcode offer the support to develop for different Apple's devices, although Xcode works only on MacOS operating systems; it cannot run on any other OS. Furthermore, Xcode holds a vital role in the publication of the applications. Apple has a meticulous way of checking developers' code, with different steps that start directly on Xcode before actually uploading the app on Apple's Servers.

Platform-specific mobile applications have a high cost since every platform needs a different mobile application, but also support and maintenance for different types of apps result in higher product price. However, the platform-specific solution of developing mobile applications is the most widely use and tested approach.

[1]Java integrated development environment (IDE) for developing computer software

2.2 Progressive web applications

Progressive web applications strongly rely on the web standards (JS, HTML, and CSS); for this reason, they can be installed via mobile web browsers and are highly portable across multiple mobile platforms. This method enables the app to work across platforms thereby reducing development cost and time. Nevertheless, Progressive web apps also have some significant limitations. E.g., the final user of the applications might use a browser or operating system that does not support some features that developers need to implement; in this case, the web standards represent a negative aspect that might cause some differences in the user experience. Moreover, Progressive web apps offer access just to a limited number of native device features.

On the other hand, a key characteristic of a progressive web application is the capacity for working and loading the content offline. The offline content loading relies on a service worker[2] that allows content caching.

It is enough to add the application to the device's home screen. After that, the app connects to the Internet to install the necessary data locally on the device. When the application finally appears on the home screen, it no longer carries the appearance of a site, giving the impression of being a native application [13]. More precisely, content caching represents a key feature that only progressive web applications offer, and it is in accordance with the requirements of the RAIL model (Response, Animation, Idle, Load) introduced by Google. Specifically, the RAIL model requires the application to always respond to user requests, showing an animation while waiting for content to load [27]. During inactivity, the so-called "Idle moment" of the RAIL model stores as much content as possible in the cache; this helps to load all the content in less time the next time the user launches the app.

Furthermore, progressive web applications must also be adaptive and responsive to guarantee the user with an optimal version of the app for different screen sizes.

2.3 Hybrid mobile applications

The definition of Hybrid mobile applications is still an open debate on the mobile development field. This section introduces to two different mobile application frameworks: Ionic framework and React native. In this case, the

[2]A service worker is a type of web worker. It is a JavaScript file that runs separately from the main browser thread, intercepting network requests, caching or retrieving resources from the cache, and delivering push messages.

Hybrid interpretation varies between the two solutions. The Ionic framework is part of the Hybrid mobile world where the applications are websites embedded in a mobile-native container. Contrarily, it is not entirely correct to define react native applications as native applications; although they use native components, the logic and interaction do not change; it remains written in Javascript. For this reason, the react native definition is open to interpretations. This research defines react native as a hybrid solution.

2.3.1 Ionic framework (Cordova)

The Ionic Framework is an open source and free software development kit (SDK) that is used to develop hybrid mobile applications with JavaScript, CSS, and HTML5. It only requires a computer with an Internet connection to start building applications. The Ionic framework has an active ever-growing community that makes it easier for new developers to start building their first applications.

As we have seen in the introduction of this section, the applications developed with the Ionic framework are Hybrid Mobile Applications; the main reason is that they use browser windows to display their interfaces. These browser windows are called WebViews[3], and together with Apache Cordova[4] they create the Architecture of Hybrid Mobile applications; everything relies on these two concepts, where the WebView communicates with Cordova's APIs which then further communicates with the mobile device's operating system. Cordova mainly represents an interface between the WebViews and the native functionalities of the phone, such as GPS or camera. Moreover, Cordova converts the application code written in Ionic for other specific platforms; the code is written only once, and then adapted to different platforms, such as Android and iOS. Another important aspect of the Ionic framework is that it provides a Command Line Interface (CLI) to develop and deploy applications.

AngularJS is the core technology behind the Ionic framework. More precisely, AngularJS represents the engine that defines the components of the graphical interface. To create native looking designs, developers must rely on the CSS portion of the framework together with AngularJS [28].

[3]Browser pages that run inside the scope of a mobile application using Ionic. This browser implements code written in HTML, CSS, and JavaScript.

[4]A platform that provides APIs written in JavaScript to interact with Native features of the mobile device such as access to the camera or a microphone.

2.3.2 React Native

React Native is a framework to develop native mobile applications. As seen with the Ionic framework, also React Native gives the possibility to deploy the same code for Android and iOS platforms. React native is based on ReactJS, a JavaScript framework used to develop web applications[5]. The JavaScript code is the logic of the application, while all UI compiles into native code. The application does not run into some wrapper, but a real native app is available after the development process. React Native uses JavaScript XML (JSX), a syntax extension to JavaScript that is used to write the UI of the application. JSX uses the full power of JavaScript; it produces React elements that can then be compiled into native code.

React Native follows a "component concept". A component consists of a state with some properties and a lifecycle. During the development of the application, the developer can use the components multiple times, and this usually really helps to speed up the development of the applications. Furthermore, React Native also introduces to a new system called **Virtual document object model** (Virtual DOM). This system is a virtual representation, and simplified copy of the HTML DOM[6]. The Virtual DOM allows React Native to do all the computational work within the virtual representation, to skip DOM operations, often slow and browser-specific. More precisely, when an event occurs, the page reacts to it modifying the Virtual DOM elements of the page.

The Figure 2.1 below helps to better understand the concepts introduced earlier. The figure compares some UI components of ReactJS for the web, Native components, and React Native components. React Native knows the translation of every Virtual DOM component for Android and iOS platforms. The "translation" happen at compile-time.

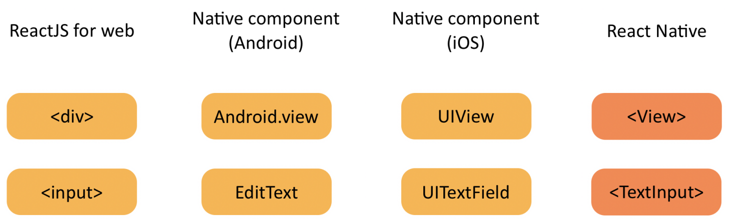


Figure 2.1: The standard components for the UI of different DOMs

[5]https://reactjs.org/

[6]The Document Object Model (DOM) defines the logical structure of documents and the way a document is accessed and manipulated.

Unlike the Ionic framework, React Native does not use WebViews, and this is a positive aspect because the WebView usually requires a substantial amount resources that lower the applications performances compared to the ones developed with native UI. React Native maps every component into the native interface of the Android and iOS platforms. Thanks to this mapping, the mobile applications developed with React native provide performances that are close to platform-specific mobile applications performances.

React Native applications also consist of some challenges that developers must take into consideration:

- **Limited cross-platform styling of components.** Developers must style components on their own or use third-party libraries. This styling process can take a considerable amount of time.
- **Only a basic set of pre-built components.** The solution to this problem is to build components from scratch or use a third party library to speed up the building process. Usually, it is common to use the second solution due to the number of open sources third-party libraries available on the Internet.
- **No responsiveness out of the box.** Developers must create responsive designs on their own, defining sizes in the stylesheets or using external packages (e.g., react-native-responsive-ui)

Besides having some restrictions, React Native is a framework that is constantly growing, and a new version is available practically every month; therefore, bugs sometimes still appear, and this issue is also due to the high dependency on third-party libraries React Native also includes powerful tools that help to overcome the challenges presented above. The development process might become hard, but the final results consist of applications with deep integration into the operating system and excellent performance close to native apps.

State management with Redux

After introducing React Native, it is important to mention and discuss Redux, a state management framework. The state of a system (e.g., an application) is the set of information that determines the output at a given input at a specific time. For instance, a booking system of a restaurant that shows that ten tables are free for the next day. During the day someone decides to book a table; after that, the booking system is updated and shows that only nine tables remain free. The internal state of the application changed, and the component for reserving a table updated the state to show the number nine instead of the number ten.

Redux helps to respond to user actions and manage the data displayed in the application. The UI does not directly change the state of the application, but instead, it sends a message to interpret the state.

Redux is built on top of three pillars.

- **Actions.** Actions are payloads of information that send data from your application to your store. They are the only source of information for the store.
- **Reducers.** Reducers specify how the application's state changes in response to actions sent to the store. Actions only describe what happened and not how the application's state changes.
- **Store.** Object that brings actions and reducers together. More precisely, the store holds the application state and allows to access and update it. Furthermore, the store allows listeners to register and unregister.

Figure 2.2 clearly shows how Redux handles all the state management. The following are the three key steps of the process.

1. The way to manipulate the state is to dispatch an action. As seen before, actions are messages that we send to the store; e.g., if the application wants to add a new row to a table, a component dispatches an action to complete that request.
2. The reducers receive the action, and it updates the state. The way it performs that is running a sync function. Reducers can be multiple and combined.
3. Whenever the central store receives an update, it triggers the information to all the subscribed components. The component can hook into this subscription, and it receives the updates. After that, the application might change some data displayed on the screen.

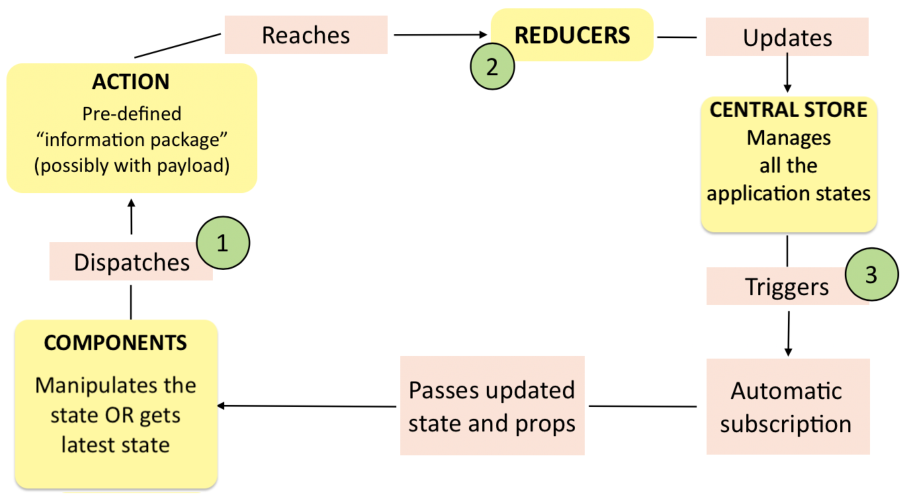

Reaches
REDUCERS
2
Updates
ACTION
Pre-defined "information package" (possibly with payload)
CENTRAL STORE
Manages all the application states
Dispatches
1
Triggers
3
COMPONENTS
Manipulates the state OR gets latest state
Passes updated state and props
Automatic subscription

Figure 2.2: Redux state management overview

2.4 Agile software development

The company that has agreed with Insinööriliitto for the implementation of the new application uses agile development methods for the development of their software projects. Agile software development is a widely used process, especially in the mobile landscape; it represents a set of different approaches to implement software, with more collaboration between the software developer and the user (or stakeholders) during the software development process [20]. Agile is now the mainstream software development method of choice worldwide. The most popular agile method is called SCRUM [19]. There are three principal roles in SCRUM [21] that includes product owner, scrum master and team members. The product owner is responsible for maintaining the list of requirements that are gathered by end users, teams, and stakeholders. Scrum teams are the people that work on a project, including designers, developers, and testers. Teams use to complete the tasks in all their iterations. Team members complete their work within a specific period, by arranging daily meetings where they broadly discuss the work. The Scrum master is the leader of the teams; it usually protects them from external or internal disturbance, such as the product owner that asks directly a developer to add a new feature. The Scrum master have to train the product owner and Scrum team to follow the principles of agile methodology [24].

Figure 2.3 overviews the SCRUM process. It is important to mention that the Scrum master, together with the product owner, has to decide when to schedule the sprint reviews. The figure shows that sprint reviews times usually take place every one to four weeks. During these Sprint reviews, the team has to show the work to the product owner.

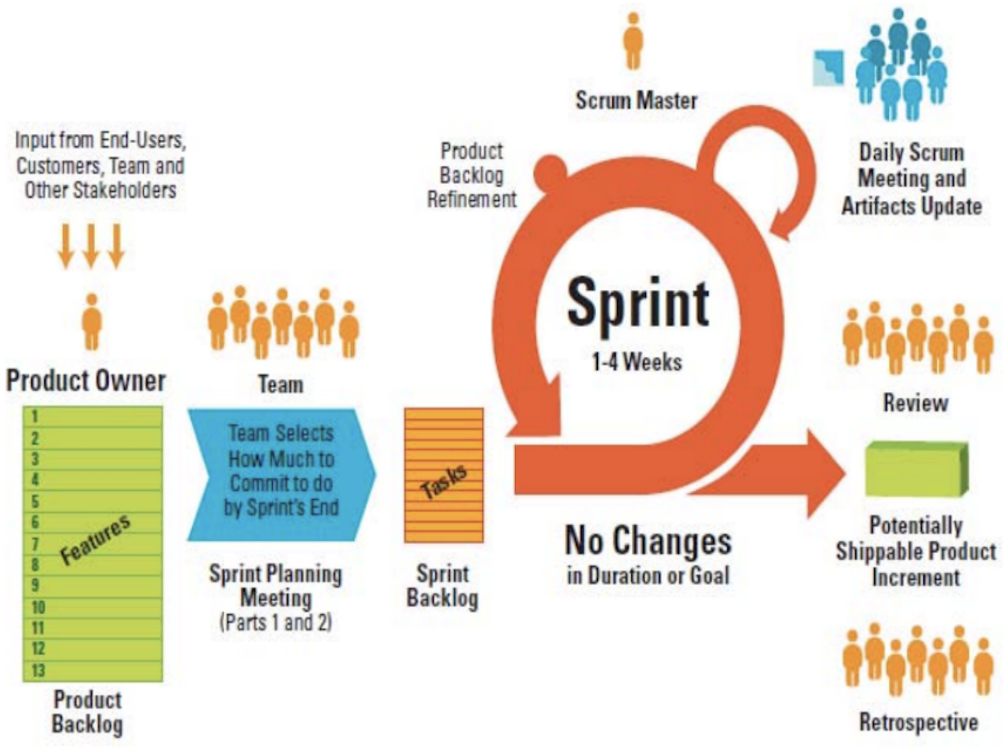

Input from End-Users, Customers, Team and Other Stakeholders
Product Owner
Features
Product Backlog
Team
Team Selects How Much to Commit to do by Sprint's End
Sprint Planning Meeting (Parts 1 and 2)
Tasks
Sprint Backlog
Scrum Master
Product Backlog Refinement
Sprint
1-4 Weeks
No Changes in Duration or Goal
Daily Scrum Meeting and Artifacts Update
Review
Potentially Shippable Product Increment
Retrospective

Figure 2.3: Scrum process overview [24]

The SCRUM process consists of the following five steps.

1. All the stakeholders start listing the requirements of the new system together with the product owner. This list is called **product backlog**.
2. The scrum team defines the different **sprints** and their length. After that, the team assigns the implementation of the requirements to the different sprints.
3. The first sprint starts. The focus is the implementation of all the requirements within the sprints created in the previous step. For every sprint that ends there is a **sprint review** with the product owner. During the development process or sprint reviews there might be the need to create other sprints, or change the related allocation of the time.

4. After that sprints are completed, there is a potential **final product** ready for the final users.

5. After the release, the entire SCRUM team gathers together to have a final meeting called **retrospective**. During the retrospective, team members discuss about the all process of development, trying to find if something could be improved.

Chapter 3

Requirements engineering

Requirements Engineering is a process about discover, understand, formulate, analyze and agree on what problem the software should solve, why such a problem needs a solution and who should take the responsibility of solving that problem. More precisely, *"Requirements Engineering is the process of identifying, communicating and documenting the requirements for a system"* [25]. Requirements represent the purpose of a system, a set of statements about the new system that the stakeholders agree must be right to solve the customer's problem adequately. The following are examples of requirements sources: high-level goals expressed by various stakeholders, pre-existing systems, pre-existing documentation, competing systems, documentation about interfacing systems, policies and collective agreements.

The Standish Group International, Inc.[1] has presented a report where they analyze the major sources of failure for new systems. The results show that in 50% of cases, projects fail because of incomplete requirements, last minute change of requirements, unrealistic expectations, and unclear goals [10].

The results of the study presented above explain why it is so important to consider requirements engineering during the development process; for this reason, software companies are now aware of the importance of this set of techniques. During the development of the new features for the Insinööriliitto native mobile application, the company that has agreed with Insinööriliitto for the implementation of the new app required to include the requirements engineering studies in the development process.

This chapter describes the entire process of requirements engineering introduced for the development of the new mobile application features.

[1]The Standish Group is a primary research advisory organization that focuses on software project performance.

3.1 Goal-oriented requirements engineering

Goal Oriented analysis is an element of Requirements Engineering that aims to define the goals of the stakeholders and the new system. A goal is an objective that the new system should achieve, they refer to properties that the system has to ensure. The system which a goal refers to may be the system-as-is (the current one, without modifications) or the system-to-be (The system after the development); both of them are into the Requirements Engineering process. The system-to-be includes both the software and the environment; it consists of active components such as humans, devices, and software. The paper called "Goal-Oriented Requirements Engineering: A Guided Tour" written by Axel van Lamsweerde [34] introduces the whole process of Goal-Oriented requirements engineering, focusing on the theory and specification of the most important elements in a complete manner.

Several languages available in the market offer the possibility to build goal-oriented models. The most famous one is the language called i* introduced by Eric Yu in the first half of the 90s [36]. The i* framework introduces to two type of representation: the graphical and the formal representation. This research focuses on the graphical representation, that has two modeling components: The Strategic Rationale (SR) and the Strategic Dependency (SD) model. The SD model describes mainly the dependency relationship between actors within the organization, and as a result, it helps in understanding how a specific goal fits inside the organization. On the other hand, the SR model describes stakeholders' interests and stakeholders' evaluation of various alternatives respecting their interests [31].

Figure 3.1 shows the i* basic elements for graphical representation.

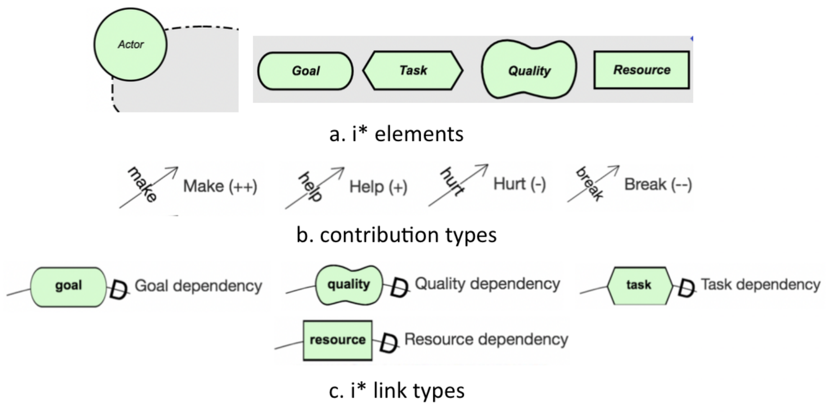


Figure 3.1: The i* basic elements

The following are the three main categories [31].

- **The intentional elements.** Goals, soft goals, tasks, resources of an actor boundary.
- **Links.** Connect the i* model elements together. **Contribution links.** positive (Make/Help links), negative (Hurt/ Break links).

This section presents four different i* diagrams related to the Insinööriliitto mobile application renovation: two Strategic Dependencies models (SD) and two Strategic Rationale (SR).
Figure 3.2 shows one i* SD diagram with the stakeholders and they strategic dependencies showing the current situation (as-is) considering early requirements. Figure 3.3 shows one i* SR diagram describing the goals of the user of the software application considering the early requirements, from which the need of the system to be emerge;

Figure 3.4 shows one i* SD diagram where the actor "system-to-be" (=software application) is introduced in to the "as-is" diagram, to satisfy some goals of the stakeholders (to-be situation), and show the new/changed dependencies. Figure 3.5 shows one i* SR diagrams describing the goals of actors and how the users goals get satisfied through the system-to-be;

Figure 3.6 categorise the main dependencies between system-to-be and users. In this case, the engineers.

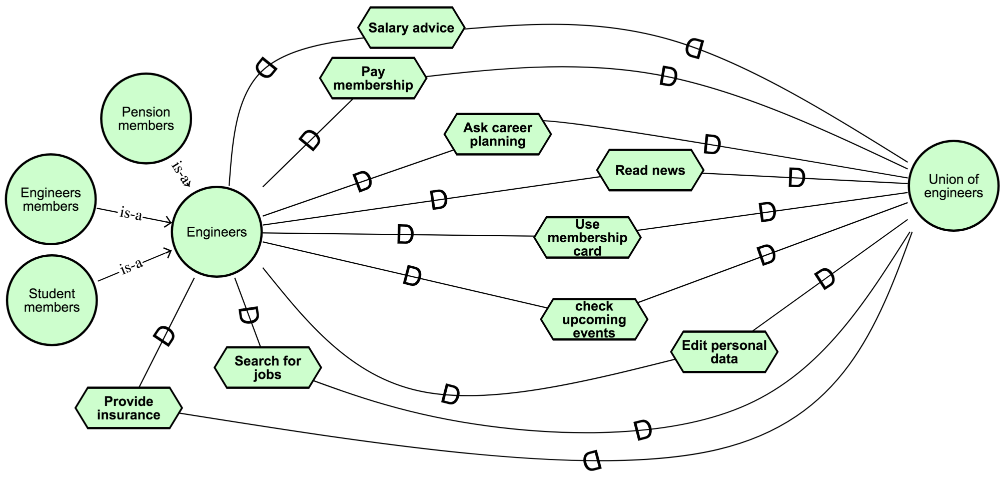


Figure 3.2: Strategic Dependencies model of the system as-is

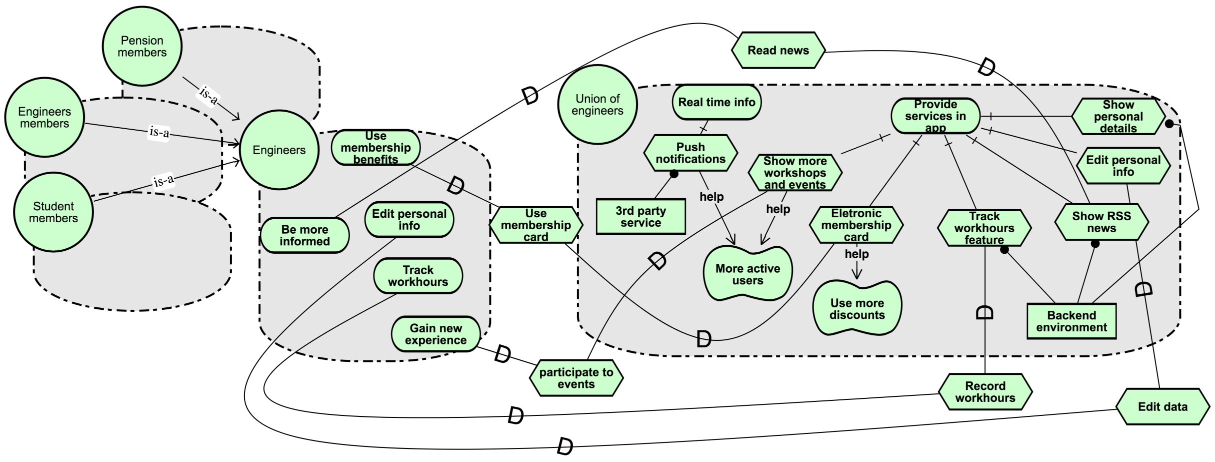


Figure 3.3: Strategic Rationale model of the system to-be

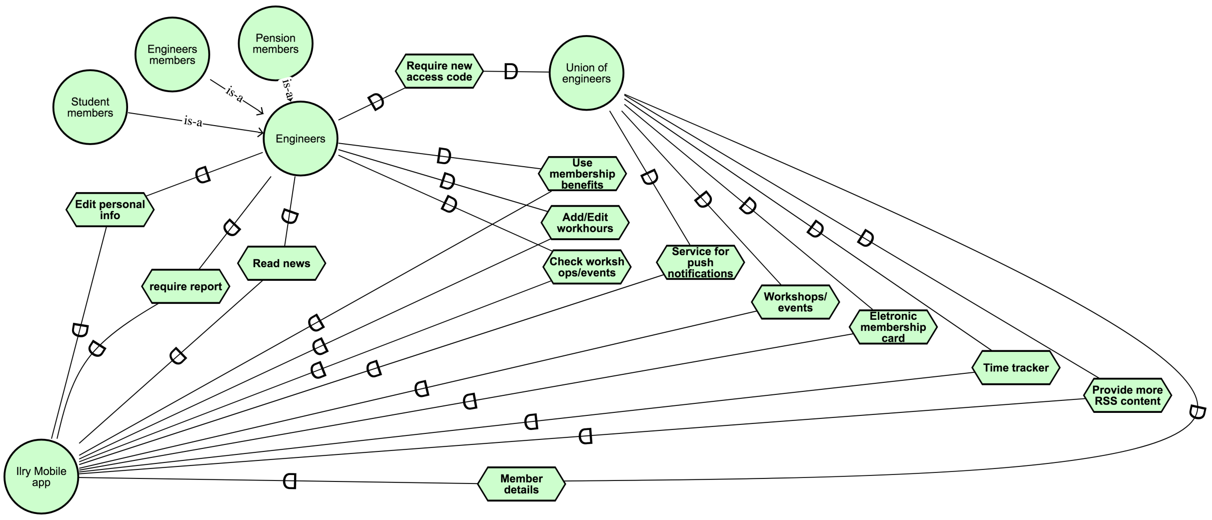


Figure 3.4: Strategic Dependencies model where system-to-be is introduced into the system-as-is

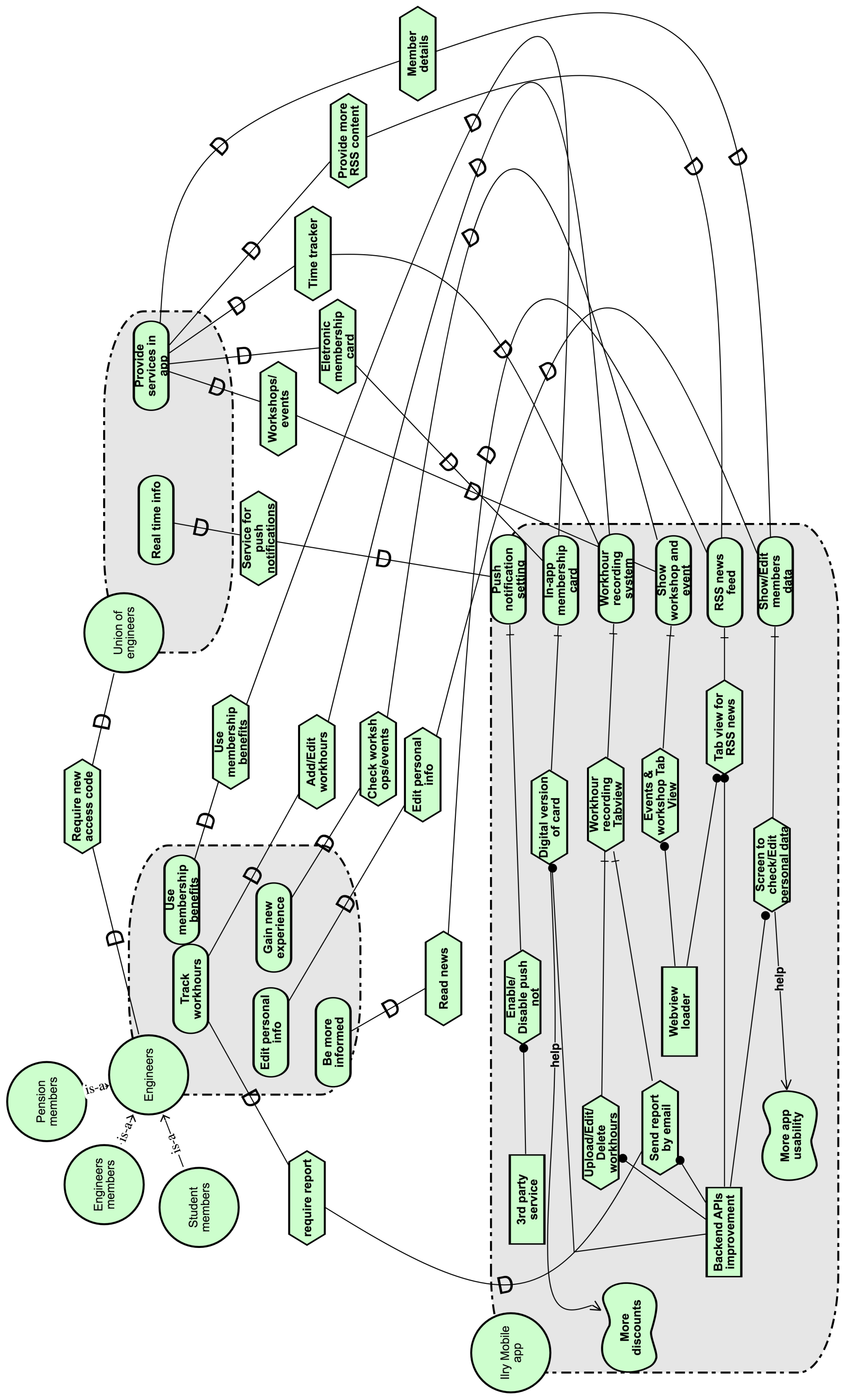
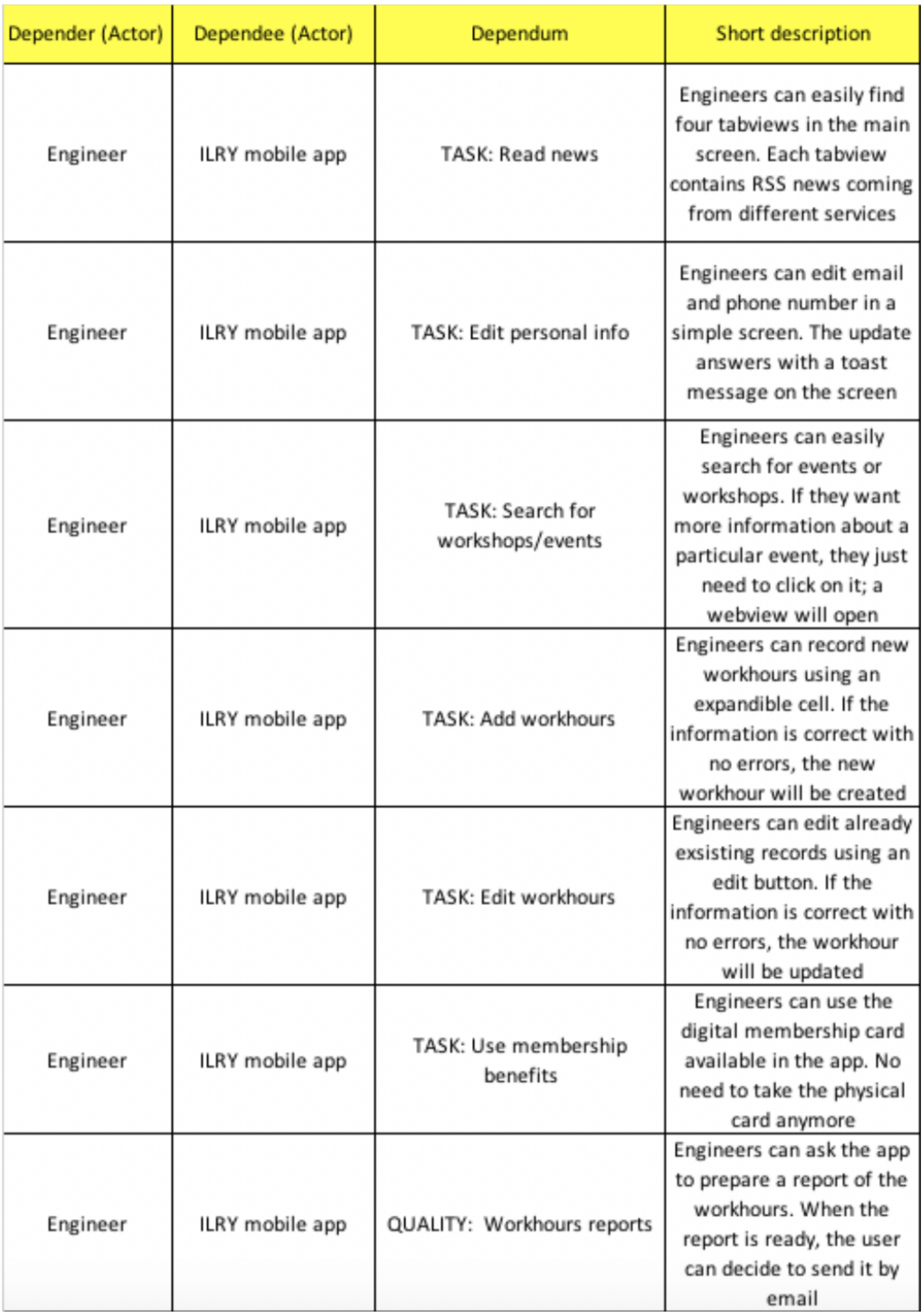


Figure 3.5: Strategic Rationale of the final system-to-be



Depender (Actor)	Dependee (Actor)	Dependum	Short description
Engineer	ILRY mobile app	TASK: Read news	Engineers can easily find four tabviews in the main screen. Each tabview contains RSS news coming from different services
Engineer	ILRY mobile app	TASK: Edit personal info	Engineers can edit email and phone number in a simple screen. The update answers with a toast message on the screen
Engineer	ILRY mobile app	TASK: Search for workshops/events	Engineers can easily search for events or workshops. If they want more information about a particular event, they just need to click on it; a webview will open
Engineer	ILRY mobile app	TASK: Add workhours	Engineers can record new workhours using an expandible cell. If the information is correct with no errors, the new workhour will be created
Engineer	ILRY mobile app	TASK: Edit workhours	Engineers can edit already exsisting records using an edit button. If the information is correct with no errors, the workhour will be updated
Engineer	ILRY mobile app	TASK: Use membership benefits	Engineers can use the digital membership card available in the app. No need to take the physical card anymore
Engineer	ILRY mobile app	QUALITY: Workhours reports	Engineers can ask the app to prepare a report of the workhours. When the report is ready, the user can decide to send it by email

Figure 3.6: Main dependencies between system-to-be and engineers

3.2 UML models

Another important element of Requirements Engineering is the Unified Modeling Language (UML). The UML was developed by three software engineers in 1994. These models help to model and document a software using diagrams representations. By using these diagrams it is possible to identify some errors in the software process. The Unified Modeling Language consists in a set of diagrams that serves for different purposes, before the implementation of the software or after. Some UML diagrams describe the behavior of the system, while others try to analyze the structure of a process. This research focuses on one UML category, the use cases. More precisely, this diagram helps to understand in which way the different actors can interact with the software. Figure 3.7 shows the different ways the actors (engineers and union of engineers) can interact with the new mobile application. This diagram takes in consideration only the new features.

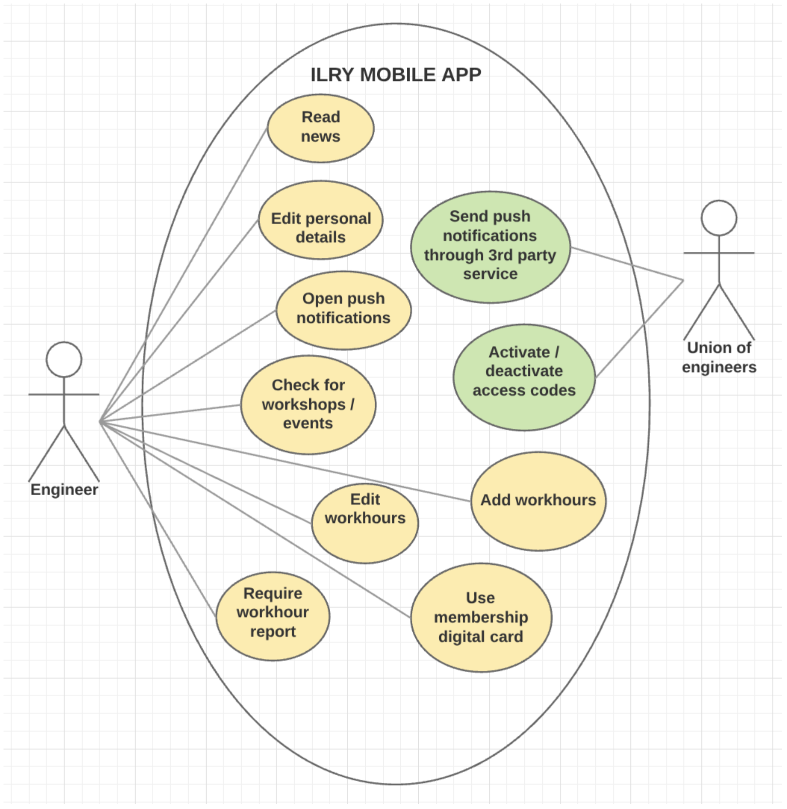
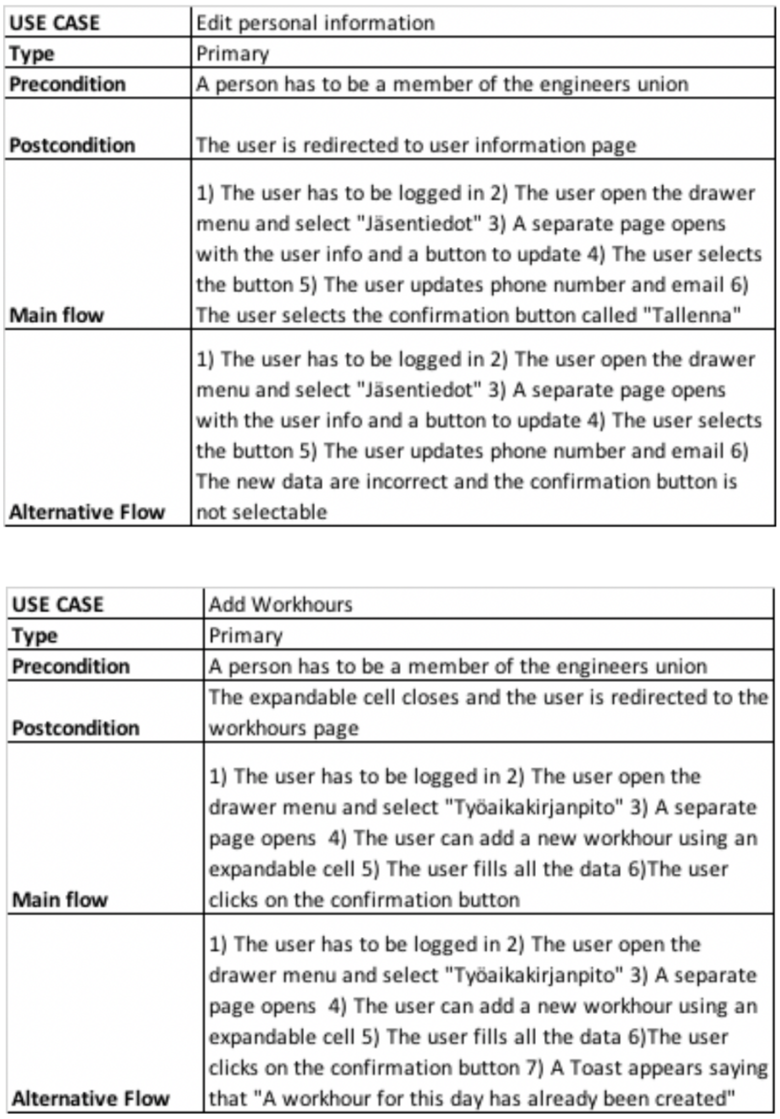


Figure 3.7: Use case diagram of the Insinööriliitto mobile app with the two actors

Figure 3.8 shows two examples of use case specification: The use case specification about the "Edit personal detail", and the use case specification about the "Add work-hours".



USE CASE	Edit personal information
Type	Primary
Precondition	A person has to be a member of the engineers union
Postcondition	The user is redirected to user information page
Main flow	1) The user has to be logged in 2) The user open the drawer menu and select "Jäsentiedot" 3) A separate page opens with the user info and a button to update 4) The user selects the button 5) The user updates phone number and email 6) The user selects the confirmation button called "Tallenna"
Alternative Flow	1) The user has to be logged in 2) The user open the drawer menu and select "Jäsentiedot" 3) A separate page opens with the user info and a button to update 4) The user selects the button 5) The user updates phone number and email 6) The new data are incorrect and the confirmation button is not selectable

USE CASE	Add Workhours
Type	Primary
Precondition	A person has to be a member of the engineers union
Postcondition	The expandable cell closes and the user is redirected to the workhours page
Main flow	1) The user has to be logged in 2) The user open the drawer menu and select "Työaikakirjanpito" 3) A separate page opens 4) The user can add a new workhour using an expandable cell 5) The user fills all the data 6)The user clicks on the confirmation button
Alternative Flow	1) The user has to be logged in 2) The user open the drawer menu and select "Työaikakirjanpito" 3) A separate page opens 4) The user can add a new workhour using an expandable cell 5) The user fills all the data 6)The user clicks on the confirmation button 7) A Toast appears saying that "A workhour for this day has already been created"

Figure 3.8: Two examples of use case specification

3.3 Software requirements

In the previous section, the paper analyzes just one type of diagram of the UML models: the use cases diagram. The UML models consist of a large set of diagrams; e.g., the UML class diagram, the sequence diagram, and the state machines represents other important UML models that where not required for the ilry mobile application[2].

This section presents the last part of the study of requirements for the Insinööriliitto new mobile application: The software requirements.

The previous sections of the requirements engineering studies have delivered all the possible interactions that the various actors can have with the new application. Now it is time to analyze in details each requirement, focusing on the description and the type of requirement.

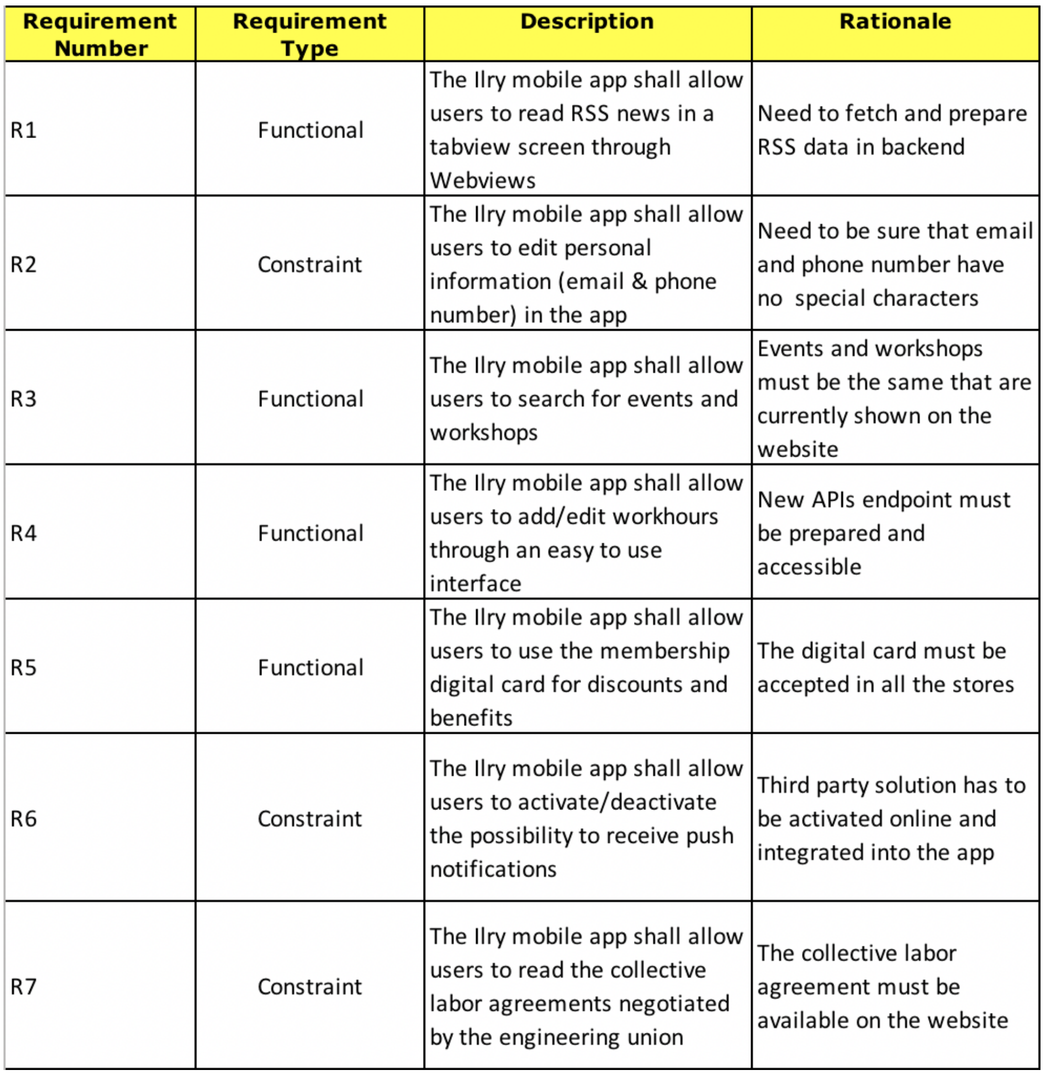
3.3.1 Requirements specification

A software requirements specification (SRS) is a relevant part of the Requirements Engineering document that captures complete description about how the system is expected to perform.

In software requirements specification, the requirements need to respect the following qualities: correctness, unambiguous, consistency, completeness, ranked for importance, verifiability, modifiability, and traceability [26]. These qualities follow the standard ISO-IEEE 830 [6]; this standard introduces to a set of recommended practice for Software Requirements Specifications. Moreover, every requirement belongs to a specific category, such as functional, performance, interface, maintainability, reliability, safety, quality, operational, resource, and constraint.

Figure 3.9 shows the requirements for the development of the new Insinööriliitto mobile application. Every requirement consist of a number, a type, and description. The description of every requirement must follow the rules of "Good requirements"; it needs to specify the system under discussion and the desired result that is wanted within a specified time that is measurable (if possible). The table in the figure shows the seven requirements for the development of the new application. These requirements do not represent the whole features of the application, but just the new ones. Four of these requirements are functional requirements, while three are constraint requirements. The rational column shows the condition that the system must respect to have a valid requirement. E.g., the R1 does not work if the backend is not fetching any RSS data.

[2]Official name of the application on the Android and iOS app stores



Requirement Number	Requirement Type	Description	Rationale
R1	Functional	The Ilry mobile app shall allow users to read RSS news in a tabview screen through Webviews	Need to fetch and prepare RSS data in backend
R2	Constraint	The Ilry mobile app shall allow users to edit personal information (email & phone number) in the app	Need to be sure that email and phone number have no special characters
R3	Functional	The Ilry mobile app shall allow users to search for events and workshops	Events and workshops must be the same that are currently shown on the website
R4	Functional	The Ilry mobile app shall allow users to add/edit workhours through an easy to use interface	New APIs endpoint must be prepared and accessible
R5	Functional	The Ilry mobile app shall allow users to use the membership digital card for discounts and benefits	The digital card must be accepted in all the stores
R6	Constraint	The Ilry mobile app shall allow users to activate/deactivate the possibility to receive push notifications	Third party solution has to be activated online and integrated into the app
R7	Constraint	The Ilry mobile app shall allow users to read the collective labor agreements negotiated by the engineering union	The collective labor agreement must be available on the website

Figure 3.9: Requirements specification table for the Insinööriliitto mobile application.

3.3.2 Prioritization of the requirements

With the whole group of requirements in place, the paper presents the last step of the Requirements Engineering study for the Insinööriliitto mobile application.

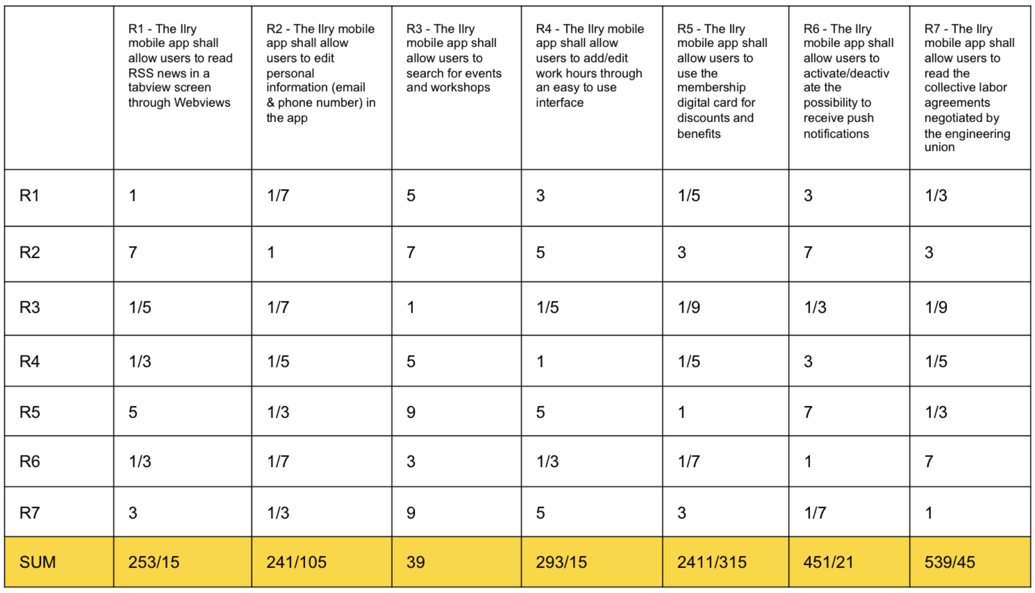
The prioritization of the requirements is essential before starting the implementation process; it helps to find the best ordering of requirements to ensure the quality and value of the new system. Through the prioritization of the requirements, developers receive the list of requirements with their priority. The following are the two general ways to give priority to a set of requirements.

- **Completely automated.** Given the set of objects to order, these can be rank on the bases of their properties.
- **Human assisted.** Human has to give a contextual assessment of the object to rank (considering their properties).

This project uses the human-assisted way to rank the requirements. This research uses two techniques that can work together to create a prioritization matrix: The pair-wise prioritization and the Analytic hierarchy process (AHP) matrix. The pair-wise technique compares every pair of two requirements giving a preference of which of the two is more important than the other. The comparison uses a ranking criterion that the user decides before. The following is the ranking chosen for the Insinööriliitto mobile application.

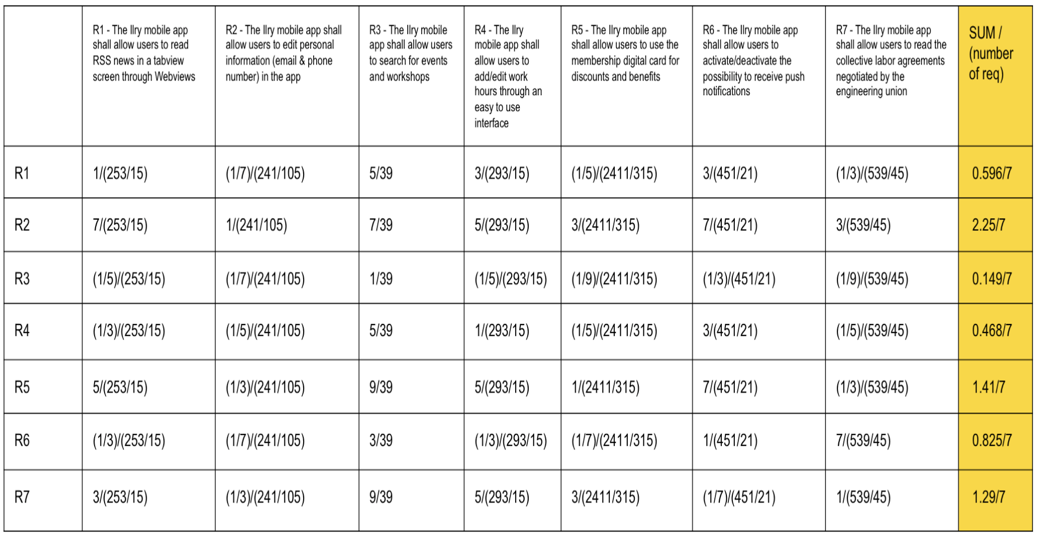
- **1 Neutral - 3 Slightly more important - 5 Moderately more important - 7 significantly more important - 9 Completely more important**

This ranking togheter with the Analytic hierarchy process (AHP), creates a matrix of comparison. Figure 3.10 shows the AHP matrix that compares the requirements with each other. Moreover, the AHP matrix normalizes all the column with the sum of the whole values. Figure 3.10 shows the same AHP matrix of the previous figure, but here the matrix is performing the second step, where the values in the cells have to divide the sum of the previous step. After that, the matrix creates the sum of the rows.



	R1 - The Ilry mobile app shall allow users to read RSS news in a tabview screen through Webviews	R2 - The Ilry mobile app shall allow users to edit personal information (email & phone number) in the app	R3 - The Ilry mobile app shall allow users to search for events and workshops	R4 - The Ilry mobile app shall allow users to add/edit work hours through an easy to use interface	R5 - The Ilry mobile app shall allow users to use the membership digital card for discounts and benefits	R6 - The Ilry mobile app shall allow users to activate/deactivate the possibility to receive push notifications	R7 - The Ilry mobile app shall allow users to read the collective labor agreements negotiated by the engineering union
R1	1	1/7	5	3	1/5	3	1/3
R2	7	1	7	5	3	7	3
R3	1/5	1/7	1	1/5	1/9	1/3	1/9
R4	1/3	1/5	5	1	1/5	3	1/5
R5	5	1/3	9	5	1	7	1/3
R6	1/3	1/7	3	1/3	1/7	1	7
R7	3	1/3	9	5	3	1/7	1
SUM	253/15	241/105	39	293/15	2411/315	451/21	539/45

Figure 3.10: First step of the Analytic hierarchy process (AHP)



	R1 - The Ilry mobile app shall allow users to read RSS news in a tabview screen through Webviews	R2 - The Ilry mobile app shall allow users to edit personal information (email & phone number) in the app	R3 - The Ilry mobile app shall allow users to search for events and workshops	R4 - The Ilry mobile app shall allow users to add/edit work hours through an easy to use interface	R5 - The Ilry mobile app shall allow users to use the membership digital card for discounts and benefits	R6 - The Ilry mobile app shall allow users to activate/deactivate the possibility to receive push notifications	R7 - The Ilry mobile app shall allow users to read the collective labor agreements negotiated by the engineering union	SUM / (number of req)
R1	1/(253/15)	(1/7)/(241/105)	5/39	3/(293/15)	(1/5)/(2411/315)	3/(451/21)	(1/3)/(539/45)	0.596/7
R2	7/(253/15)	1/(241/105)	7/39	5/(293/15)	3/(2411/315)	7/(451/21)	3/(539/45)	2.25/7
R3	(1/5)/(253/15)	(1/7)/(241/105)	1/39	(1/5)/(293/15)	(1/9)/(2411/315)	(1/3)/(451/21)	(1/9)/(539/45)	0.149/7
R4	(1/3)/(253/15)	(1/5)/(241/105)	5/39	1/(293/15)	(1/5)/(2411/315)	3/(451/21)	(1/5)/(539/45)	0.468/7
R5	5/(253/15)	(1/3)/(241/105)	9/39	5/(293/15)	1/(2411/315)	7/(451/21)	(1/3)/(539/45)	1.41/7
R6	(1/3)/(253/15)	(1/7)/(241/105)	3/39	(1/3)/(293/15)	(1/7)/(2411/315)	1/(451/21)	7/(539/45)	0.825/7
R7	3/(253/15)	(1/3)/(241/105)	9/39	5/(293/15)	3/(2411/315)	(1/7)/(451/21)	1/(539/45)	1.29/7

Figure 3.11: Second step of the Analytic hierarchy process (AHP)

The matrix is now complete. Now it is time to order the requirements based on their value. The third step of the Analytic hierarchy process requires to normalize the results, dividing each value for the number of requirements presents in the matrix (In this case seven). The following are the results of the normalization in order of the requirements priorities.

- **R2: 0,32** The ilry mobile app shall allow the user to edit personal information (email and phone number) in the app
- **R5: 0,20** The ilry mobile app shall allow the user to use the digital membership card for discounts and benefits
- **R7: 0,18** The ilry mobile app shall allow the user to read the collective labor agreements negotiated by the engineering union
- **R6: 0,11** The ilry mobile app shall allow the user to activate/deactivate the possibility to receive push notifications
- **R1: 0,085** The ilry mobile app shall allow users to read RSS news in a tab view screen through Web views
- **R4: 0,067** The ilry mobile app shall allow the user to add/edit work-hours through an easy to use interface
- **R3: 0,021** The ilry mobile app shall allow users to search for events and workshops

The prioritization is now complete, together with the Requirements Engineering process.

Chapter 4

Implementation

This section goes back to the fundamental requirements of the research: the update of the Insinööriliitto mobile application and the implementation of a hybrid mobile application with the same functionalities of the native one. As explained in chapter 1, the goal of the research is to perform a complete comparison between the two solutions. Section 1.1 presented the general expectation of the company, where the idea is to develop an entirely new update of the already existing mobile application.

Currently, Insinööriliitto has two native applications in the market, one for iOS platforms and the other one for the Android platform. The company is requiring developers to update and rewrite the existing code. After concluding the update of the existing applications, the research focuses on the comparison between the apps. Section 2.1 presents a complete overview of all the mobile application development solutions currently available in the market. The research requires the development of a third application to perform the comparison.

The first thing to clarify is the reason why this research does not focus on a Progressive Web Application as a possible solution of comparison. This research aims to propose a comparison with a solution that can replace the current one; therefore a fundamental constraint of Insinööriliitto needs to remain clear: the user must find the applications on the Apple store or Google play store. Unfortunately, it is not possible to upload Progressive Web Application to the official app store of iOS and Google. Furthermore, as we have seen in Section 2.1.2, Progressive Web Apps are entirely based on web standards (HTML, CSS, JavaScript). The web standards have a positive aspect, the cross-platform availability without the need for platform-specific adjustments. However, Progressive Web Apps are not close to native solutions; therefore it would be too challenging to perform a real comparison with the two complete native applications.

The remained solutions are the two hybrid ones: The Ionic framework and React Native framework. The positive aspect is that both solutions offer the cross-platform development. In case of React Native, Developers also can write platform-specific design, while Ionic automatically adapts some characteristics to the specific platform. This section uses the following four important aspects to help understand which hybrid solution is better for the research purpose.

- **Programming language.** The programming language of the two solutions is different. Although both solutions use JavaScript, React Native is based on ReactJS while Ionic on AngularJS.
- **Testing.** Another important feature offered by both solutions is the real-time testing. During the development process, both solution refresh instantly every time developers change the code; this is convenient and useful.
- **Developer experience.** React Native does not use Webviews while they are a key widget for the Ionic framework. With React Native the browser compatibility issues disappear, no time is spent checking the different styles for every browser. It is important to remember that it is not possible to force the user to use a specific browser, so it is difficult to avoid the styles issue with the Ionic framework.
- **User experience.** Ionic applications have a real problem with performance. The web was not though to implement the complex applications available today. On high-performance phones, the applications run fine, but with low-performance phones, things can get worse. Differently, React Native utilize native components with the smooth animations that people see in platform-specific mobile applications.

React Native includes all the favorable characteristics of the Ionic framework, but it also uses native components that help to offer a better developer and user experience. Therefore, this research focuses on the comparison between the native applications and a third application implemented with the React Native framework.

Now that the choice of the framework is clear and the requirements for the Insinööriliitto mobile application have been specified, it is time to discuss the implementation of the three different app versions:

- **Native iOS mobile application** using the xCode integrated development environment (IDE);

- **Native Android mobile application** using the Android Studio (IDE);
- **React Native mobile application** using a text editor called Visual studio code[1].

The company that has agreed with Insinööriliitto for the implementation of the new app uses the SCRUM agile method for the development of the Insinööriliitto native mobile applications (Section 2.5). Sprint reviews with Insinööriliitto are set every two weeks. The total time for the development of the native applications was two months and a half, with five sprint reviews. The total time for the development of the React native application was one month. The team consist of two developers and one designer. One developer worked on expanding the back-end side of the application while the author of this thesis worked on the front-end side of the applications. The product owner uses JIRA[2] for a better management of the project. The language of the applications is Finnish.

[1]https://code.visualstudio.com/
[2]JIRA is a complete tool that supports with Agile Project Management.

4.1 Native mobile applications

This section details the implementation of the two most important features of the mobile application: Reading RSS news with Web Views and work-hours creation, editing and reporting. Appendix A shows the figures of the other features' images.

Figure 4.1 shows the primary navigation tool of the apps, the drawer navigator. The navigation of the application uses a drawer navigator. The drawer opens from the left side of the app, and allows the user to access all the functionalities of the application. This type of Navigation is available natively for both Android and iOS platforms.

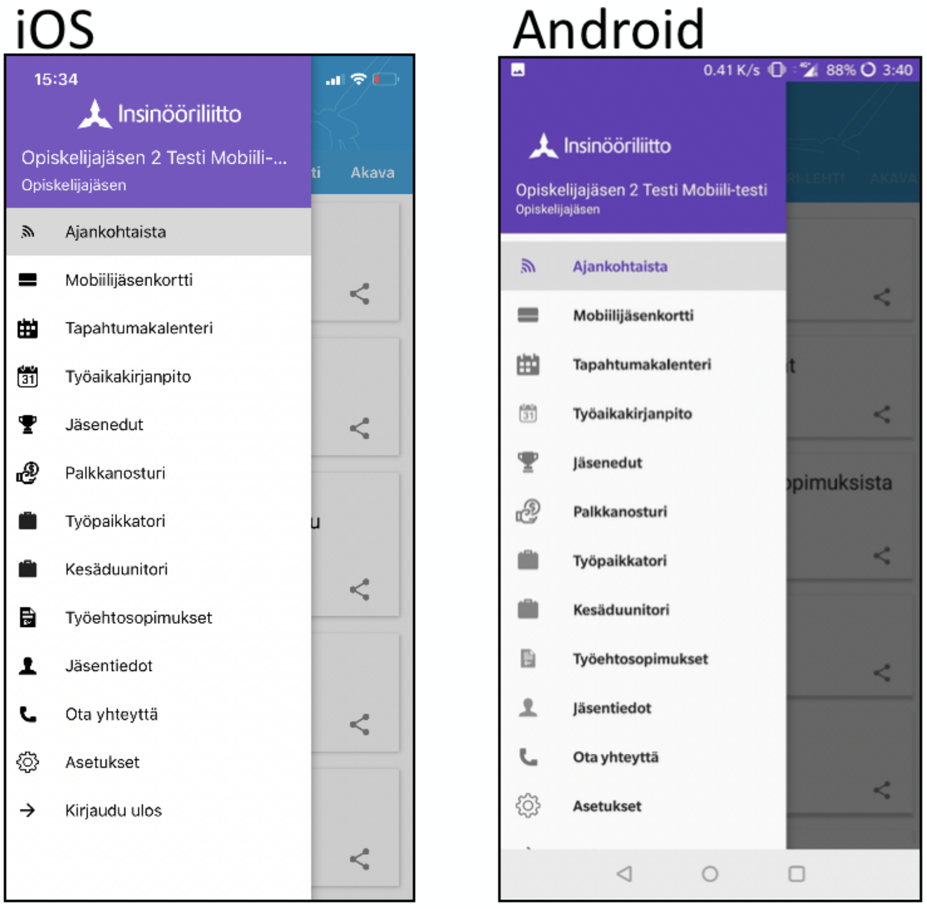


Figure 4.1: Drawer navigation of the native mobile applications

Read RSS news with WebView

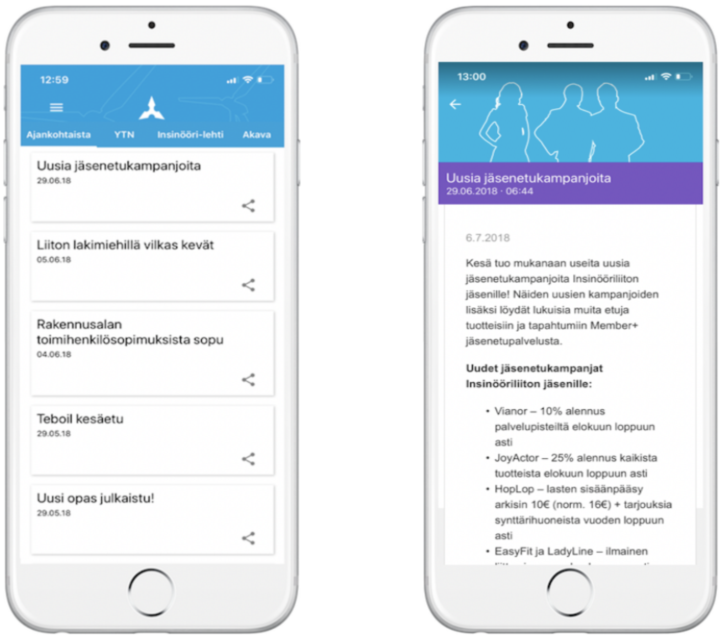
One of the requirements of Insinööriliitto is to bring RSS-feeds into the main view of the application. The user has to see the RSS news coming from four different sources. The sources are Ajankohtaista, YTN, Insinööri-lehti, and Akava[3]. The user must see a preview of the title of the news, with the possibility to open it and have a complete overview of the news. Moreover, the user must have the possibility to share the news in social networks. The RSS news page is the first one shown right after the login of the user. The view is set as follows: the team decided to create a tabbed view, with four tabs that the user can swipe. Every tab represents a different source of news; every view shows a list of cells where every cell corresponds to an article.

Figure 4.2 and Figure 4.3 show the look of the view for both iOS and Android platforms, where the tabs are on top of the view. The top position of the tabs represents the first challenge to overcome. The Android platform supports the top bars natively, while iOS supports the tabs only at the bottom of the view. During the design of the app, the product owner asked about having the same look on both platforms; for this reason, the iOS app has to install a third party library that allows the implementation of the top bars also in iOS. Specifically, the iOS application uses the xmartlabs library: **XLPagerTabStrip by xmartlabs**[4]. A parent View Controller needs to incorporate four children views. The xmartlabs library provides four different implementations of the top bars, however, the RSS news view focuses on the most popular one, the "Button Bar" (Similar to Instagram, Youtube, and Skype).

Every cell has a button for sharing the news on social networks. Moreover, if the user clicks on one particular news item, a new page opens. The new page incorporates a Web view that loads the RSS link of the article. These links are not ready to show only the relevant text, but they show also banners and other elements. Before loading the news on the Web views, the developers of the team have to replace the banners and other elements' code from the HTML page of the news (Setting the style parameter "display" to "none" is the faster solution).

[3]Namely, four different associations or federation which provide information for employees in the engineering sector.

[4]https://github.com/xmartlabs/XLPagerTabStrip


12:59

Ajankohtaista YTN Insinööri-lehti Akava

Uusia jäsenetukampanjoita
29.06.18

Liiton lakimiehillä vilkas kevät
05.06.18

Rakennusalan toimihenkilösopimuksista sopu
04.06.18

Teboil kesäetu
29.05.18

Uusi opas julkaistu!
29.05.18

13:00

Uusia jäsenetukampanjoita
29.06.2018 · 06:44

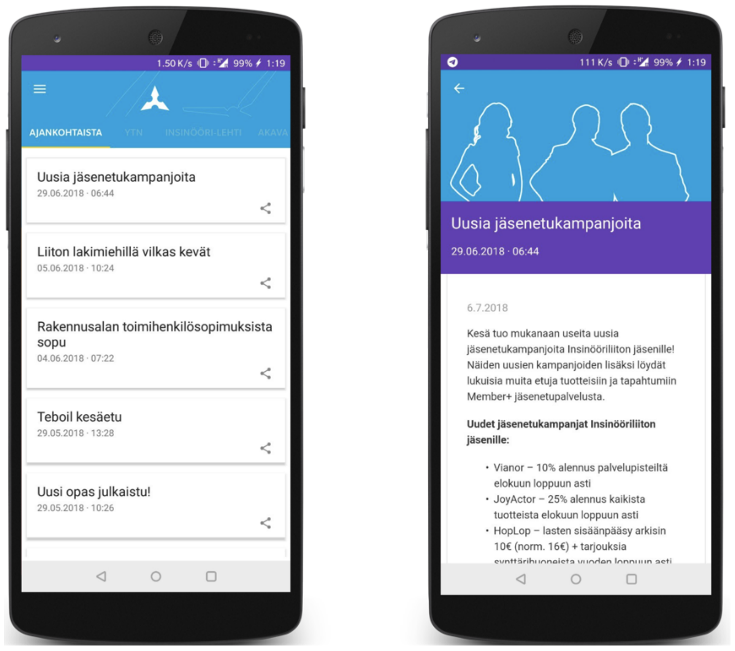
6.7.2018

Kesä tuo mukanaan useita uusia jäsenetukampanjoita Insinööriliiton jäsenille! Näiden uusien kampanjoiden lisäksi löydät lukuisia muita etuja tuotteisiin ja tapahtumiin Member+ jäsenetupalvelusta.

Uudet jäsenetukampanjat Insinööriliiton jäsenille:

- Vianor – 10% alennus palvelupisteiltä elokuun loppuun asti
- JoyActor – 25% alennus kaikista tuotteista elokuun loppuun asti
- HopLop – lasten sisäänpääsy arkisin 10€ (norm. 16€) + tarjouksia synttärihuoneista vuoden loppuun asti
- EasyFit ja LadyLine – ilmainen

Figure 4.2: Overview of the news feed under iOS


1.50 K/s 99% 1:19

AJANKOHTAISTA YTN INSINÖÖRI-LEHTI AKAVA

Uusia jäsenetukampanjoita
29.06.2018 · 06:44

Liiton lakimiehillä vilkas kevät
05.06.2018 · 10:24

Rakennusalan toimihenkilösopimuksista sopu
04.06.2018 · 07:22

Teboil kesäetu
29.05.2018 · 13:28

Uusi opas julkaistu!
29.05.2018 · 10:26

111 K/s 99% 1:19

Uusia jäsenetukampanjoita
29.06.2018 · 06:44

6.7.2018

Kesä tuo mukanaan useita uusia jäsenetukampanjoita Insinööriliiton jäsenille! Näiden uusien kampanjoiden lisäksi löydät lukuisia muita etuja tuotteisiin ja tapahtumiin Member+ jäsenetupalvelusta.

Uudet jäsenetukampanjat Insinööriliiton jäsenille:

- Vianor – 10% alennus palvelupisteiltä elokuun loppuun asti
- JoyActor – 25% alennus kaikista tuotteista elokuun loppuun asti
- HopLop – lasten sisäänpääsy arkisin 10€ (norm. 16€) + tarjouksia synttärihuoneista vuoden loppuun asti

Figure 4.3: Overview of the news feed under Android

Add/edit work-hours and send the report

An important new feature for the Insinööriliitto mobile applications is the work-hours tracking feature. It allows users to register the amount of time they work every day. Furthermore, Insinööriliitto required the option for users to edit the work-hours and receive a report by email.

Figure 4.4 shows the final implementation of the feature for the iOS platform and the Android platform. The structure of the view is similar to the news feed view; however, in this case, there are only two tab views. Figure 4.4a presents the first tab view with two different type of expandable cells. The first expandable cell (Kirjaa työtunnit) gives the possibility to register a workhour element, while the second type of cell represents pre-registered work-hours. If the user clicks on a registered workhour element, the cell expands by showing additional information. If the user clicks on the pen, an edit window opens.

The first cell asks the user to insert the following necessary information (Figure 4.4b): start date, start time, end date, end time, lunch break, and description. Once the user finishes inserting the necessary data, the application starts to calculate the bank-hours[5] in the following way:

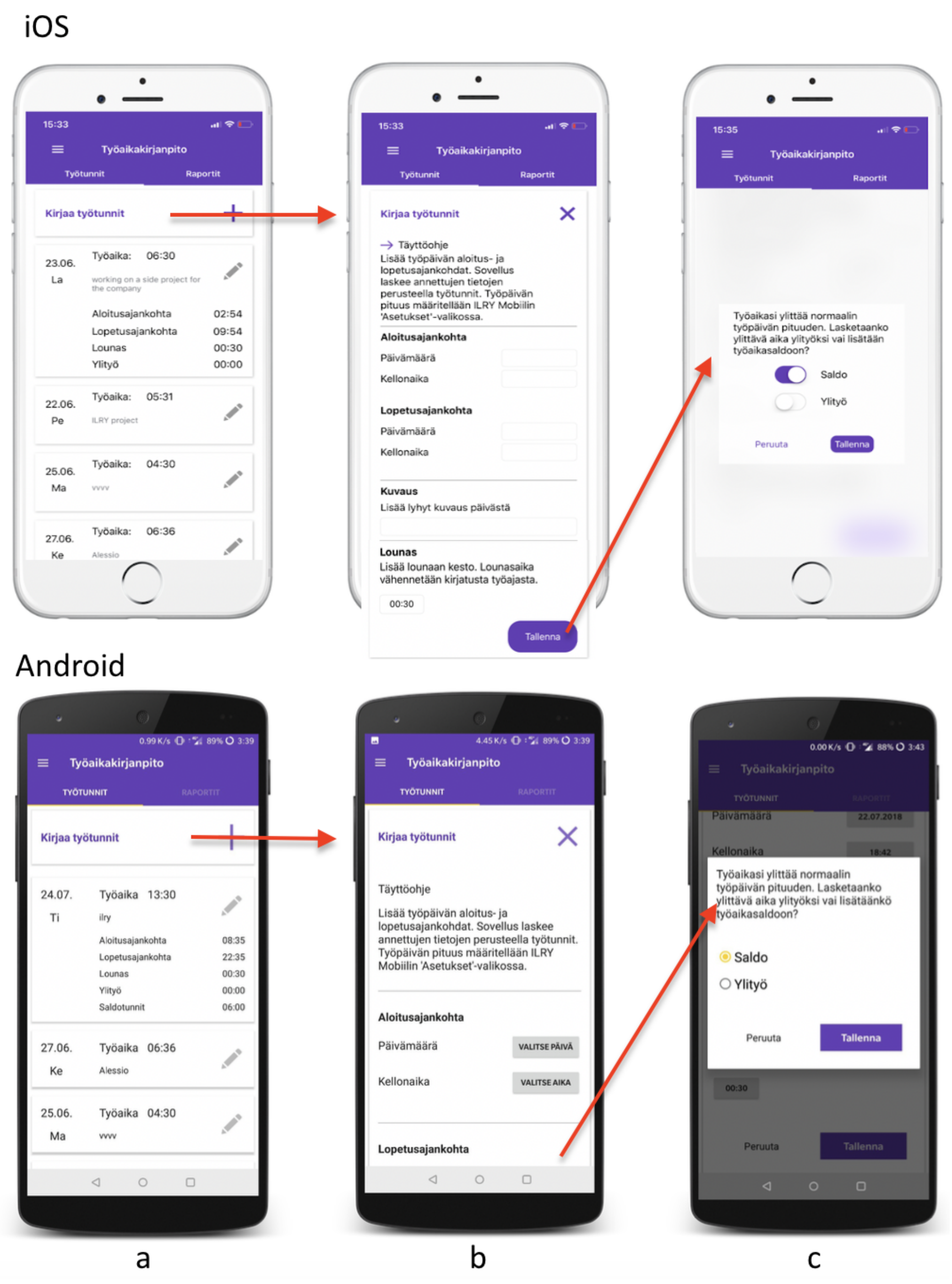
1. Convert start time, end time, lunch break, and default hours from "00:00" format to "hours" format. The default hours is a parameter set to "07:30"; the user can change this value through another view if the company requires to work more or less than 7.5 hours per day;

2. Calculate the balance of hours spent at work: [(end time - start time) - lunch break] - default hours;

3. If the result is more than 0, this means that the user has some extra hours of work. In this case, the application opens a dialog window to ask the user where to save these extra hours. The user can decide to save them as bank-hours, or as overtime hours (Figure 4.4c);

4. When this process finishes, the application calls the backend through an API to register a new workhour element.

[5]Total amount of hours spent at work by the user on a specific day of work



23.06. La Työaika: 06:30 working on a side project for the company

Aloitusajankohta 02:54
Lopetusajankohta 09:54
Lounas 00:30
Ylityö 00:00

22.06. Pe Työaika: 05:31 ILRY project

25.06. Ma Työaika: 04:30 vvvv

27.06. Ke Työaika: 06:36 Alessio

Täyttöohje
Lisää työpäivän aloitus- ja lopetusajankohdat. Sovellus laskee annettujen tietojen perusteella työtunnit. Työpäivän pituus määritellään ILRY Mobiilin 'Asetukset'-valikossa.

Aloitusajankohta
Päivämäärä
Kellonaika

Lopetusajankohta
Päivämäärä
Kellonaika

Kuvaus
Lisää lyhyt kuvaus päivästä

Lounas
Lisää lounaan kesto. Lounasaika vähennetään kirjatusta työajasta.
00:30

Työaikasi ylittää normaalin työpäivän pituuden. Lasketaanko ylittävä aika ylityöksi vai lisätäänkö työaikasaldoon?

24.07. Ti Työaika 13:30 ilry

Aloitusajankohta 08:35
Lopetusajankohta 22:35
Lounas 00:30
Ylityö 00:00
Saldotunnit 06:00

27.06. Ke Työaika 06:36 Alessio

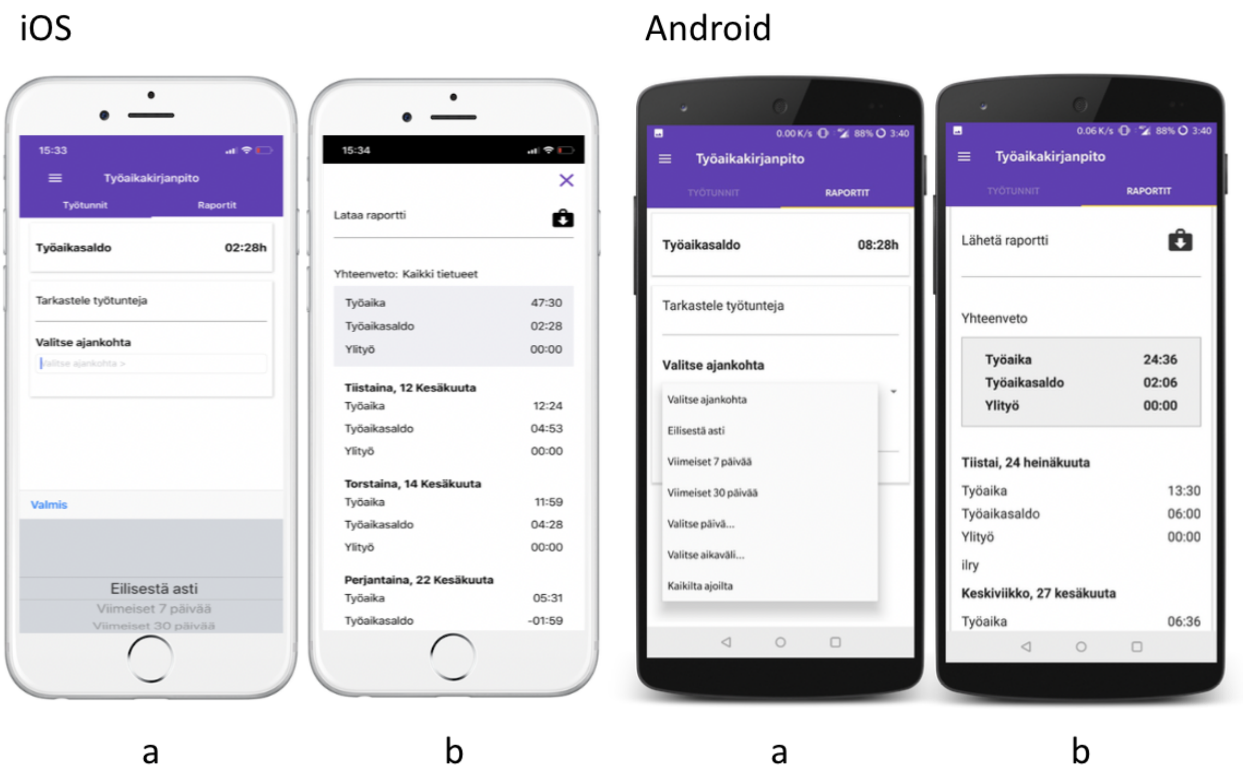
25.06. Ma Työaika 04:30 vvvv

Päivämäärä VALITSE PÄIVÄ
Kellonaika VALITSE AIKA

Päivämäärä 22.07.2018
Kellonaika 18:42

Figure 4.4: iOS and Android overview of the ”work-hours” feature implementation: first tab view

Figure 4.5 shows the second tab view (Raportit), which has two different cells; the first cell is not expandable, and it only shows the total amount of bank-hours for the specific user. With the second cell, the user requires a report of the registered work-hours; different options available for reports, such as a seven-day report, a monthly report, previous day report, a report based on specific dates and a report that takes in consideration all the registered work-hours. When the report opens, a button gives the possibility to send the report directly to the email of the user (Figure 4.5b).



Työaikasaldo 02:28h

Yhteenveto: Kaikki tietueet

Työaika	47:30
Työaikasaldo	02:28
Ylityö	00:00

Tiistaina, 12 Kesäkuuta

Työaika	12:24
Työaikasaldo	04:53
Ylityö	00:00

Torstaina, 14 Kesäkuuta

Työaika	11:59
Työaikasaldo	04:28
Ylityö	00:00

Perjantaina, 22 Kesäkuuta

Työaika	05:31
Työaikasaldo	-01:59

Työaikasaldo 08:28h

Yhteenveto

Työaika	24:36
Työaikasaldo	02:06
Ylityö	00:00

Tiistai, 24 heinäkuuta

Työaika	13:30
Työaikasaldo	06:00
Ylityö	00:00

ilry

Keskiviikko, 27 kesäkuuta

Työaika	06:36

Figure 4.5: iOS and Android overview of the “work-hours” feature implementation: second tab view

4.2 React Native mobile application

This section focuses on the implementation of the React Native application. As seen in Chapter 1, the main reason behind the implementation of the React Native application is the technical comparison, the final goal of this thesis. Therefore, the React Native solution has to be as much similar as possible to the native applications.

Similar to the previous section, we focus on the implementation of the two most important new features of the mobile application: Read RSS news with WebView and Add/edit work-hours and send the report. Appendix A shows a complete overview of all the other functionalities and views. As seen in Section 2.3.2, React Native has a key feature: it allows to write the code once but it allows to run it on both iOS and Android platforms, therefore, the look and feel of the apps is similar. Moreover, React Native applications strongly rely on third-party libraries. React Native uses node package manager modules[6] [5] to install them. For instance, to install a new package called "react-navigatio", the user has to open the command line inside the project main folder and write the following code: "npm install react-navigation".

The first significant concept of React Native is navigation. Several navigation packages are on the market, and developers must decide which solution suits better for their project. The following are the four major available navigation solutions.

- **React navigation.** The official React Native navigation solution. Therefore, this is the most widely used [4].

- **React Native Router.** React Native Router (or react-native-router-flux) was the first React navigation library. The library is still popular but not fully supported and improved any more [2].

- **React Native Navigation.** The most supported and maintained React Native navigation library. It has a native implementation, and it exposes the native APIs usable in JavaScript. React Native navigation is currently under development for the second version with additional features [1].

- **Native Navigation.** Library developed by Airbnb. As seen with React Native Navigation, also this library has a native implementation.

[6]The node package manager (npm) installs the packages the developer intends to use and provides a useful interface to work with them. It uses Node.js [29], a JavaScript runtime framework to write applications in JavaScript on the server.

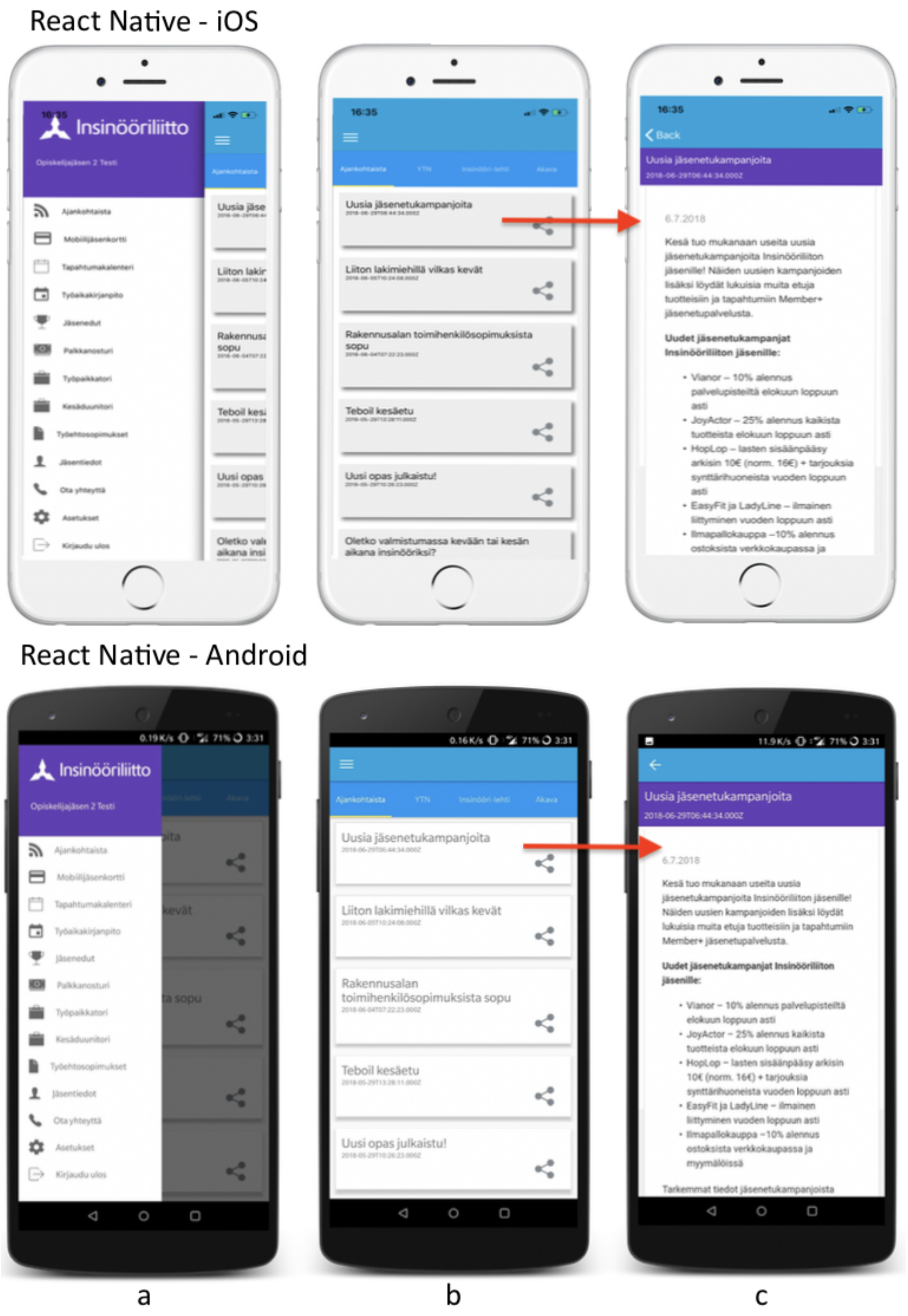
Native Navigation is still in its early stages [7].

The solutions listed before cover the entire navigation of the application; with the same library, developers can implement the drawer navigator and the tab view without the need of other libraries.

We used the React Native navigation library: since the goal is to implement a possible replacement for the native apps, a fully supported and stable library is essential. The only problem with React Native navigation is Tabs. Due to the native implementation of the library, React Native navigation respects the platforms' rules. Therefore, the TabView on Android and iOS are entirely different. The Android platform shows the tabs on top, while the iOS platform shows them on the bottom. The solution to this problem relies on a cross-platform tab view library called “React Native Tab View” [3].

Figure 4.6 shows the implementation of the drawer menu and the RSS news feed. Figure 4.6b shows that the React Native Tab View library gives the possibility to implement the tabs on top of the views of both the Android and iOS platforms. In contrary, the navigator bar is not the same as the native apps. Indeed, one challenge of working with third-party libraries in React Native is customization: it strongly depends on the implementation of the library. The React Native navigation library has different customization methods for the navigation bar, but these methods are not straightforward. Figure 4.6a shows the drawer navigator, while Figure 4.6c shows an article loaded inside a web view; the results of these implementations are equals to the native implementations. Likewise the native implementation, developers of the team have to replace the banners and other elements' code from the HTML page of the news (again, set the style parameter “display” to “none” is the faster solution).

Figure 4.7 shows the implementation of the work-hours feature. Here the result is close to the one of the native implementation.



Uusia jäsenetukampanjoita
2018-06-29T06:44:34.000Z

6.7.2018

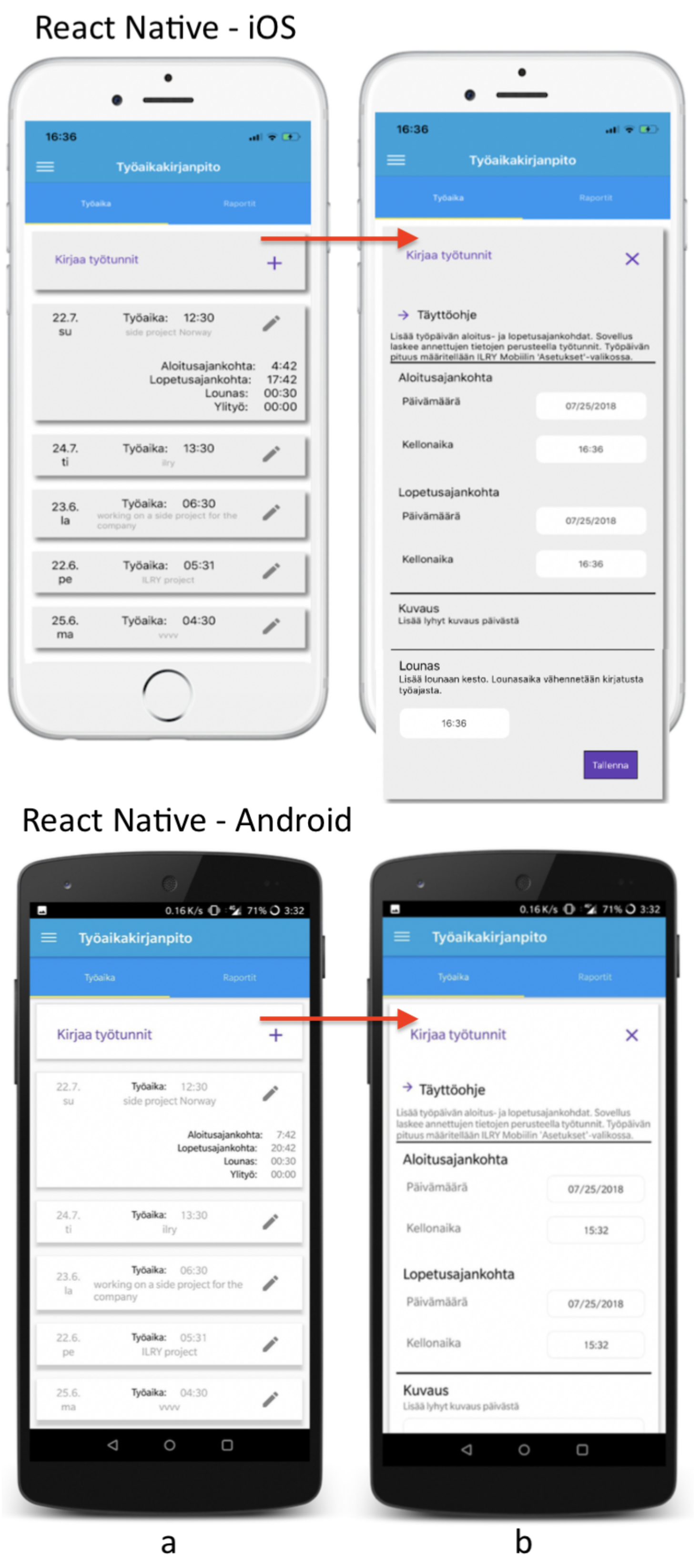
Kesä tuo mukanaan useita uusia jäsenetukampanjoita Insinööriliiton jäsenille! Näiden uusien kampanjoiden lisäksi löydät lukuisia muita etuja tuotteisiin ja tapahtumiin Member+ jäsenetupalvelusta.

Uudet jäsenetukampanjat Insinööriliiton jäsenille:

- Vianor – 10% alennus palvelupisteiltä elokuun loppuun asti
- JoyActor – 25% alennus kaikista tuotteista elokuun loppuun asti
- HopLop – lasten sisäänpääsy arkisin 10€ (norm. 16€) + tarjouksia synttärihuoneista vuoden loppuun asti
- EasyFit ja LadyLine – ilmainen liittyminen vuoden loppuun asti
- Ilmapallokauppa –10% alennus ostoksista verkkokaupassa ja myymälöissä

Tarkemmat tiedot jäsenetukampanjoista

Figure 4.6: React Native apps overview of the drawer and read RSS news feature



16:36

Työaikakirjanpito

Työaika Raportit

Kirjaa työtunnit +

22.7. su Työaika: 12:30
side project Norway
Aloitusajankohta: 4:42
Lopetusajankohta: 17:42
Lounas: 00:30
Ylityö: 00:00

24.7. ti Työaika: 13:30
ilry

23.6. la Työaika: 06:30
working on a side project for the company

22.6. pe Työaika: 05:31
ILRY project

25.6. ma Työaika: 04:30
vvvv

Kirjaa työtunnit ×

→ Täyttöohje

Lisää työpäivän aloitus- ja lopetusajankohdat. Sovellus laskee annettujen tietojen perusteella työtunnit. Työpäivän pituus määritellään ILRY Mobiilin 'Asetukset'-valikossa.

Aloitusajankohta
Päivämäärä 07/25/2018
Kellonaika 16:36

Lopetusajankohta
Päivämäärä 07/25/2018
Kellonaika 16:36

Kuvaus
Lisää lyhyt kuvaus päivästä

Lounas
Lisää lounaan kesto. Lounasaika vähennetään kirjatusta työajasta.
16:36

Tallenna

0.16 K/s 71% 3:32

Työaikakirjanpito

Työaika Raportit

Kirjaa työtunnit +

22.7. su Työaika: 12:30
side project Norway
Aloitusajankohta: 7:42
Lopetusajankohta: 20:42
Lounas: 00:30
Ylityö: 00:00

24.7. ti Työaika: 13:30
ilry

23.6. la Työaika: 06:30
working on a side project for the company

22.6. pe Työaika: 05:31
ILRY project

25.6. ma Työaika: 04:30
vvvv

Kirjaa työtunnit ×

→ Täyttöohje

Lisää työpäivän aloitus- ja lopetusajankohdat. Sovellus laskee annettujen tietojen perusteella työtunnit. Työpäivän pituus määritellään ILRY Mobiilin 'Asetukset'-valikossa.

Aloitusajankohta
Päivämäärä 07/25/2018
Kellonaika 15:32

Lopetusajankohta
Päivämäärä 07/25/2018
Kellonaika 15:32

Kuvaus
Lisää lyhyt kuvaus päivästä

Figure 4.7: React Native apps overview of the add/edit work-hours feature

Chapter 5

Performance comparison

This chapter presents a performance comparison between the two Native mobile applications (Android and iOS platforms) and the application developed with the React Native framework. As discussed in Chapter 4, the three applications looks almost the same; they slightly differ in some elements or text dimensions though. However, it is possible to solve this differences by working more accurately with the styling.

The idea of the comparison relies on the fact that Insinööriliitto explicitly asked to develop two Native applications that must look the same. For this reason, developers must follow the same design for both the iOS and Android platform; this led them to have some issues. Therefore, developers started thinking about a better solution that could improve all the development process of the Insinööriliitto mobile application.

The goal of this chapter is to find out if the React Native application can deliver performance results similar to the Native applications.

The comparison focuses on three resources: the Central Processing Unit (CPU), the Graphics Processing Unit (GPU), and the memory. The research includes tests for both platforms. This chapter first analyzes the Native iOS application (swift) against the React Native app, then the Native Android application against the React Native Android app. Tests run on real devices, which ensures the valid of the results. The device used for the iOS comparison is an iPhone X [9], while the device used for the Android comparison is a One Plus 5T [30]. The data connection used on the real devices during the tests is 4G, with no WiFi; this helps to simulate a realistic use of the applications.

The applications execute different types of functions. Between these functions, the comparison focuses only on the most relevant and challenging ones. In particular, the following are the four different functions considered for the performance comparison.

- **News list loading.** This test start directly after the login of the user.

The application has to load all the news inside the cells. Moreover, the application divides the news for each category/tabs.

- **Webview loading (Open news).** When a user clicks on the news, a new page is open, and a web view loads an article through a link.
- **Add workhour process.** As seen in Section 4.1, the user can register a new workhour element by compiling all the information correctly. This test runs from when the user clicks on the confirmation button to the moment when the list of workhours reloads.
- **Scrollview and tabs.** This test checks the performance of the apps when the user swipes through all the news tabs and scrolls all the lists of news.

5.1 iOS applications

For the iOS performance comparison, we used a software called "Apple Instruments", part of xCode, Apple's IDE [8]. Apple Instruments is a set of tools to check the performance of iOS applications. The following are the tools used for the Insinööriliitto iOS mobile application comparison: CPU ("Time Profiler Tool"), GPU ("Core Animation Tool"), and Memory ("Allocations Tool").

Apple Instruments is easy to use; it allows the user to plug in an iOS phone, pick any running app to test and choose a specific measurement tool. After that, the tool starts recording the performance.

5.1.1 CPU

Figure 5.1 shows the CPU measurements of the Native iOS mobile application (Swift) compared to the React Native mobile application. For both the applications, we tested the four functions described before. For each function, we run four experiments; each of these produced one value. For every function, the chart in the figure reports the average of the four experiments. This type of average is called "mean"[1].

[1]In arithmetic, the mean is the sum of all values divided by the total number of values.

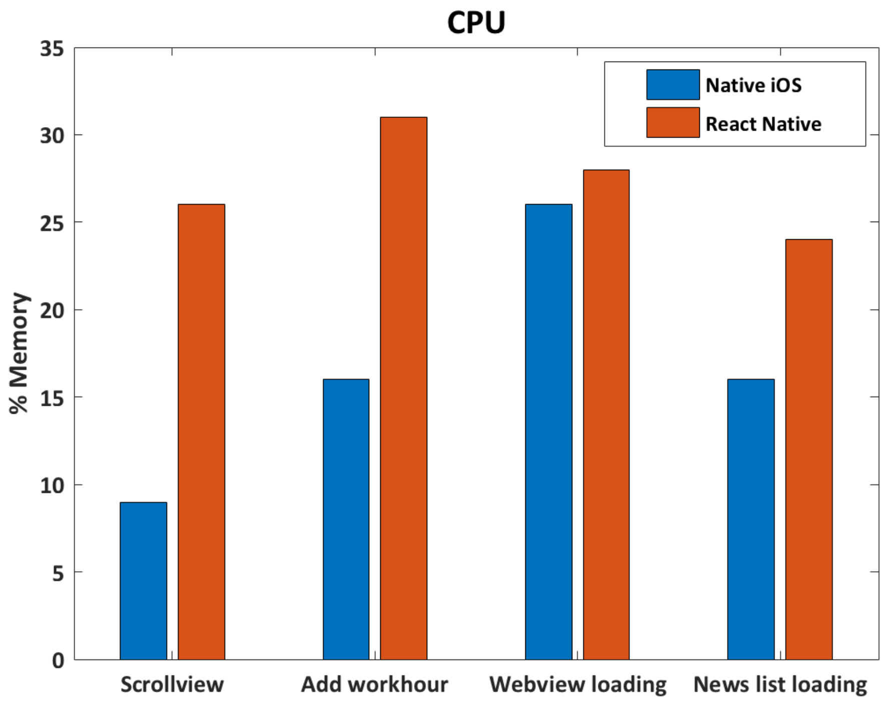


Figure 5.1: CPU Usage chart of the Native iOS mobile application versus the React-Native application

- **Scrollview and tabs test.** The Native iOS app uses 65% less CPU than the React Native application. The process of swiping and scrolling through the tabs and lists is the same for both applications.
- **Add workhour process test.** The Native iOS app uses 49% less CPU than the React Native application.
- **Webview loading test.** The Native iOS app uses 8% less CPU than the React Native application. This time the results difference is minimal. Loading a link inside a WebView has almost the same CPU usage for the Native application and the React Native one.
- **News list loading test.** The Native iOS app uses 34% less CPU than the React Native application.

The React Native application and the Native iOS application never exceeds 30% of CPU utilization. However, if we consider the proposed tests, the

Native iOS CPU values are always lower than the CPU values of the React Native app.

5.1.2 GPU

Figure 5.2 shows the GPU measurements of the Native iOS mobile application (Swift) compared to the React Native mobile application. Similar to the CPU measurements, each test was repeated four times. The value on the plot is bounded by 60 frames/second. Each second, over the course of time the task is performing, the "Core Animation" tool records a value. The chart in the figure shows the average higher value recorded during the tests.

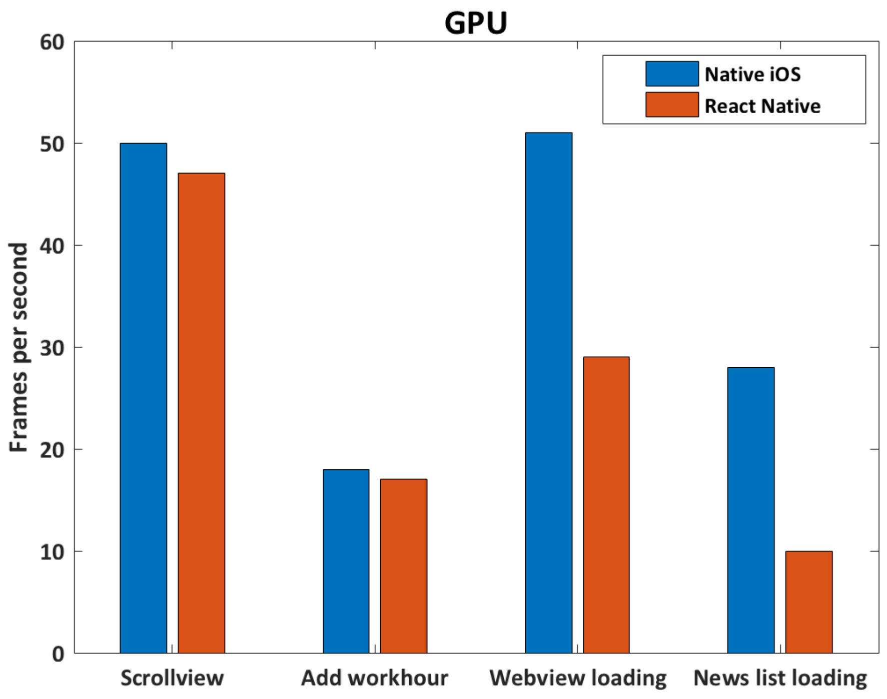


Figure 5.2: GPU Usage chart of the Native iOS application versus the React-Native application

The results of the GPU test are the following:

- **Scrollview and tabs test.** The Native iOS app has more success in this test slightly by running at 3 frames/second higher than React Native. This means that the performance of this test is almost the same for both applications.

- **Add workhour process test.** The Native iOS app has more success in this test slightly by running at 1 frames/second higher than React Native. Considering that the result is an average, the performance of both apps is the same.

- **Webview loading test.** The Native iOS app has more success in this test by running at 22 frames/second higher than React Native.

- **News list loading test.** The Native iOS app has more success in this test by running at 18 frames/second higher than React Native; again a high difference with the React Native application.

5.1.3 Memory

Figure 5.3 shows the memory usage measurements of the Native iOS mobile application (Swift) compared to the React Native mobile application. As usual, each experiment was repeated four times. While the user is performing the task, the “Allocations” tool records different measurements; the result in the chart is again an average of the higher peak value for each of the four tests.

The results of the memory usage test are the following:

- **Scrollview and tabs test.** The Native iOS app achieved better performance in this test by using 26 MiB less memory, a considerable difference.

- **Add workhour process test.** The React Native app achieved better performance in this test by using 25 MiB less memory. It is the first time that React Native is considerably better than the Native iOS application. The reason behind this result might be the state management framework called Redux (See Section 2.3.2). In fact, during the “add workhour” test of the Native application, the view and workhours list reload at the end of the adding process. Thanks to Redux, the React Native app reloads only the list and not the entire view.

- **Webview loading test.** The native iOS app achieved better performance in this test by using 33 MiB less memory, a considerable difference.

- **News list loading test.** The native iOS app achieved better performance in this test by using 23 MiB less memory.

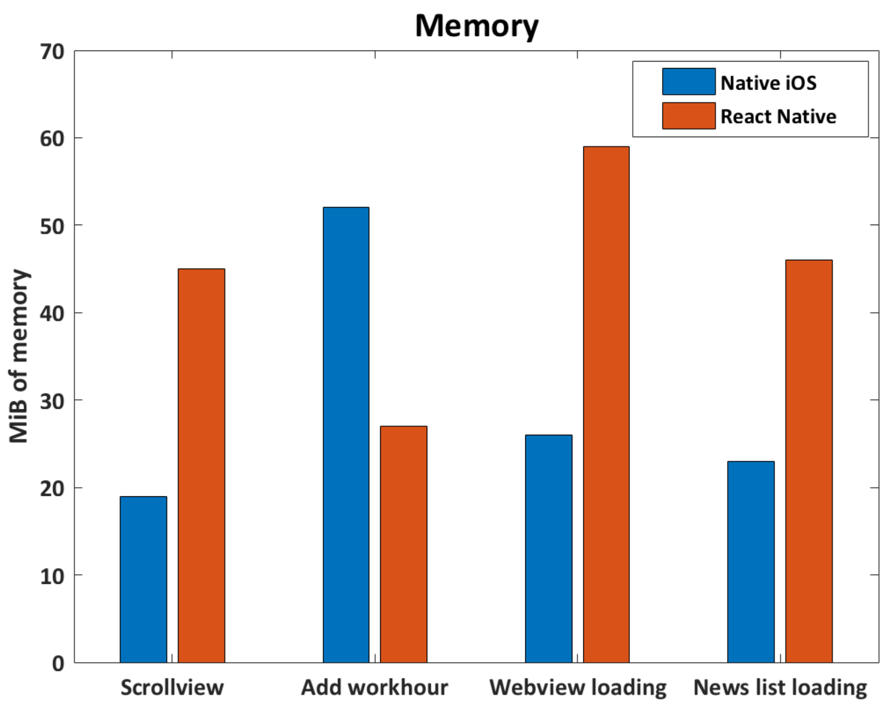


Figure 5.3: Memory usage chart of the Native iOS mobile application versus the React-Native application

5.2 Android applications

For the Android performance comparison, we used a software called "Android Profiler" [18]. Android Profiler is a set of advanced tools available on Android Studio that help developers in diagnosing app performance. The primary window available in Android Studio gives information on CPU, Memory and Network profilers. To see real-time information, the user has to connect a device with enabled USB debugging and have the app process running. Unfortunately, the Android Profiler tool does not offer the possibility to monitor the GPU usage of the app; therefore, we had to select another tool. The two main options to check GPU usage are the following: search for a third party application on the Play Store market, or use a tool called Profile GPU Rendering.

The "Profile GPU rendering tool" [16] is a built-in tool available in any Android smartphone with active Developer options [15]. The goal of this tool is to display the performance of applications. It indicates the relative time that each stage of the rendering pipeline takes to render the previous frame. This may help developers to understand which area of the application can be further improved and why. The rendering tool works in a simple way, it shows vertical bars with a different height that represents the amount of time the frame took to render (in milliseconds). The horizontal green line represents 16 milliseconds. An application has a good performance when it can render 60 frames per second or more, and this means that each bar has to stay under the green line. Each bar has colored components that map to a stage in the rendering pipeline. Figure 5.5 shows how the graph is displayed. Figure 5.4 indicates the different color values of the pipeline.

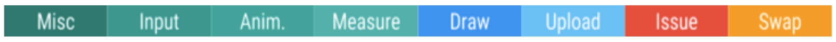


Figure 5.4: Profile GPU Rendering Graph Legend

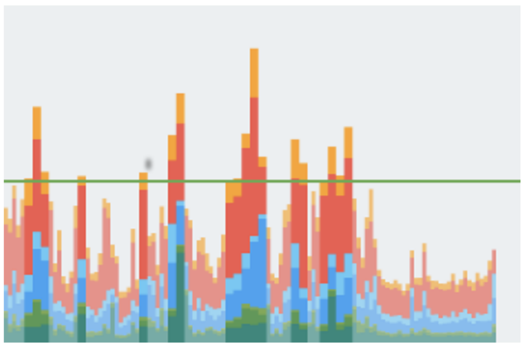


Figure 5.5: Enlarged Profile GPU Rendering graph

5.2.1 CPU

Figure 5.6 shows the CPU measurements of the Native Android mobile application (Java) compared to the React Native mobile application. We used the “mean” average like we did in the CPU measurements for the iOS applications.

The results of the CPU usage test are the following.

- **Scrollview and tabs test.** The Native Android app uses 25% less CPU than the React Native application.

- **Add workhour process test.** The Native Android app uses 62% less CPU than the React Native application.
- **Webview loading test.** The React Native app uses 10% less CPU than the Native Android application.
- **News list loading test.** The Native Android app uses 37% less CPU than the React Native application.

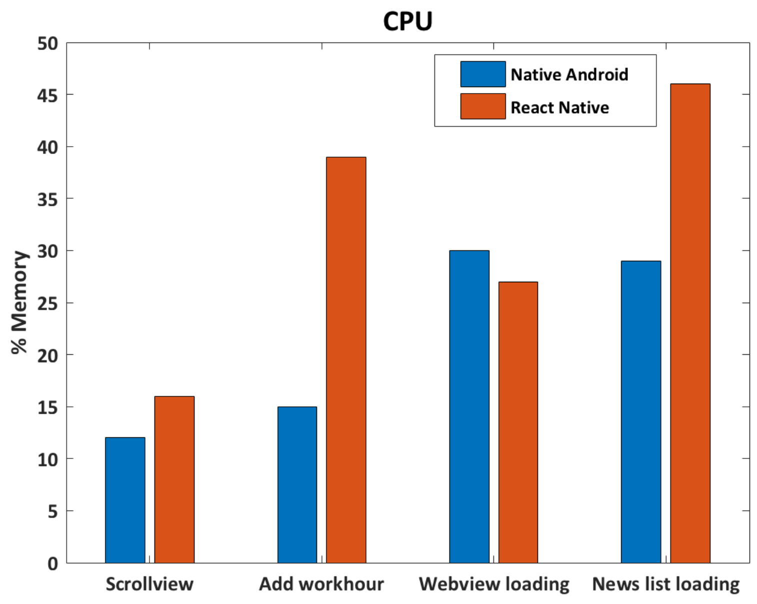


Figure 5.6: CPU Usage chart of the Native Android mobile application versus the React-Native application

5.2.2 Memory

Figure 5.7 shows the memory usage measurements of the Native Android mobile application (Java) compared to the React Native mobile application. The Android Profiler shows different data for the memory measurements; this section focuses on the "Native parameter". The Native parameter represents the amount of native memory used from the Android framework to allocate objects [17]. Each experiment was run four times, then creating a final average value using the "mean" method. The four values are the peaks registered during each of the four process (the highest peak). Unlike the iOS memory measurements, the unit of measure for the Android memory usage test is MegaByte (MB) and not Mebibyte (MiB).

The results of the memory usage test are the following.

- **Scrollview and tabs test.** The native Android app achieved better performance in this test by using 4 MB less memory during the test.
- **Add workhour process test.** The React Native app achieved better performance in this test by using 5 MB less memory during the add workhour process.
- **Webview loading test.** The React Native app achieved better performance in this test by using 24 MB less memory, a considerable difference.
- **News list loading test.** The React Native app achieved better performance in this test by using 4 MB less memory during the list loading test.

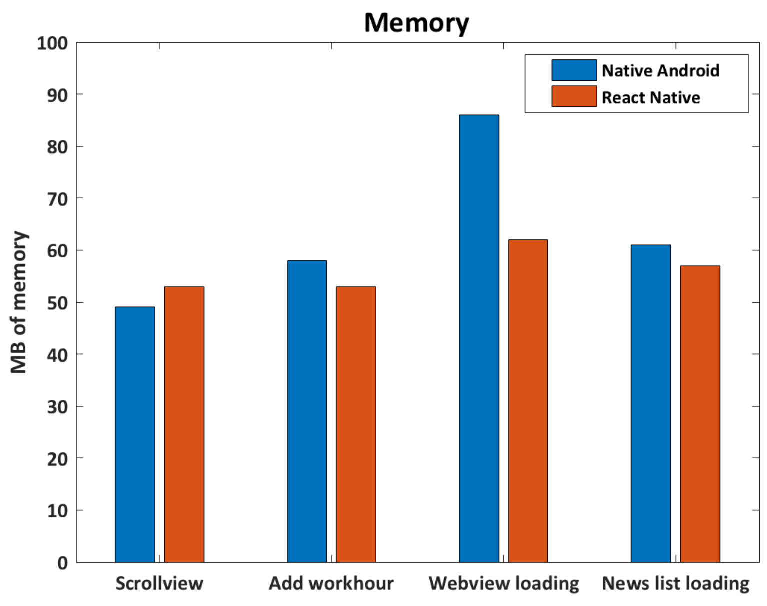


Figure 5.7: Memory usage chart of the Native Android mobile application versus the React-Native application

5.2.3 GPU

As explained before, The Android tool to measure the GPU usage of the two applications works differently from the tool used for the iOS apps comparison. The tool used for the Android GPU measurements is called profile GPU rendering tool. Developers activate it through the phone settings. After that, the bars show up on the screen of the phone. As a reminder, the green horizontal line represents the 60 frames per second limit. If the bars are higher than the green line, the application is rendering less than 60 frames per second; if the density of high lines is considerable, the application is probably not performing well. In case of the Android applications' comparison, the profile GPU rendering tool quickly shows which application is performing each task better, executing a higher number of frames per second.

Figure 5.8a shows the scrollview and tabs test. During the test, the majority of the vertical bars remain under the green line of 60 frames per second. Therefore, both applications are performing well. The Native Android application presents a few bars over the green line; however, the bars are not too high, so no significant difference in performance between the two apps.

Figure 5.8b shows the add workhour process test. During the test, the vertical bars of the React Native app remain under the green line, while for the Native Android application the trend is different. Three bars are over the 60 frames per second line; this means that the React Native app is performing slightly better. The three bars have a light green color; Figure 5.4 shows the parameter for every color of the bar. In this case, the parameters are the "Measure / Layout", "Animation", "Input Handling" and the "Misc Time"; the documentation [16] details the meaning of all the different colors.

Figure 5.9a shows the webview loading test. Both the application show a few bars that are very high. These bars appear right after the webview starts the loading process. The only difference between the bars of the two applications is the color. The Native Android application's bars have a large concentration of red color, while the React Native application's bars presents green color; this means that at that moment both applications where rendering slowly, but for different reasons. The documentation shows that the red color means "command issue", while the green colors have different meanings as seen before in the workhour process test.

Figure 5.9b shows the news list loading test. The results are similar; both applications present a few bars over the green line right after the lists start loading. However, the bars are not too high, no issues for the performance of both the apps.

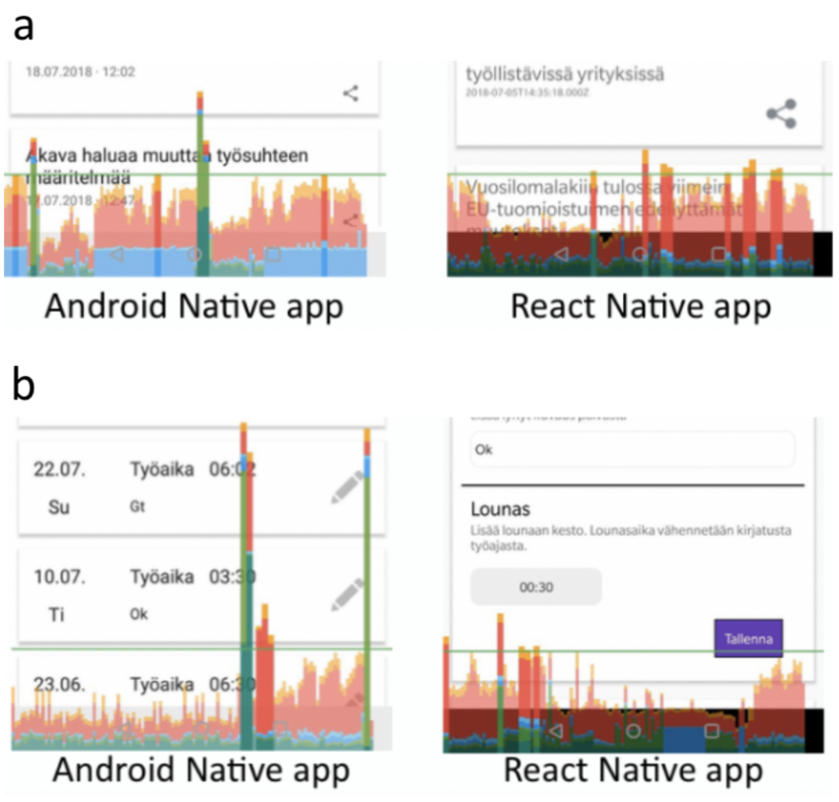


Figure 5.8: GPU Usage comparison for the scrollview test and add workhour process test

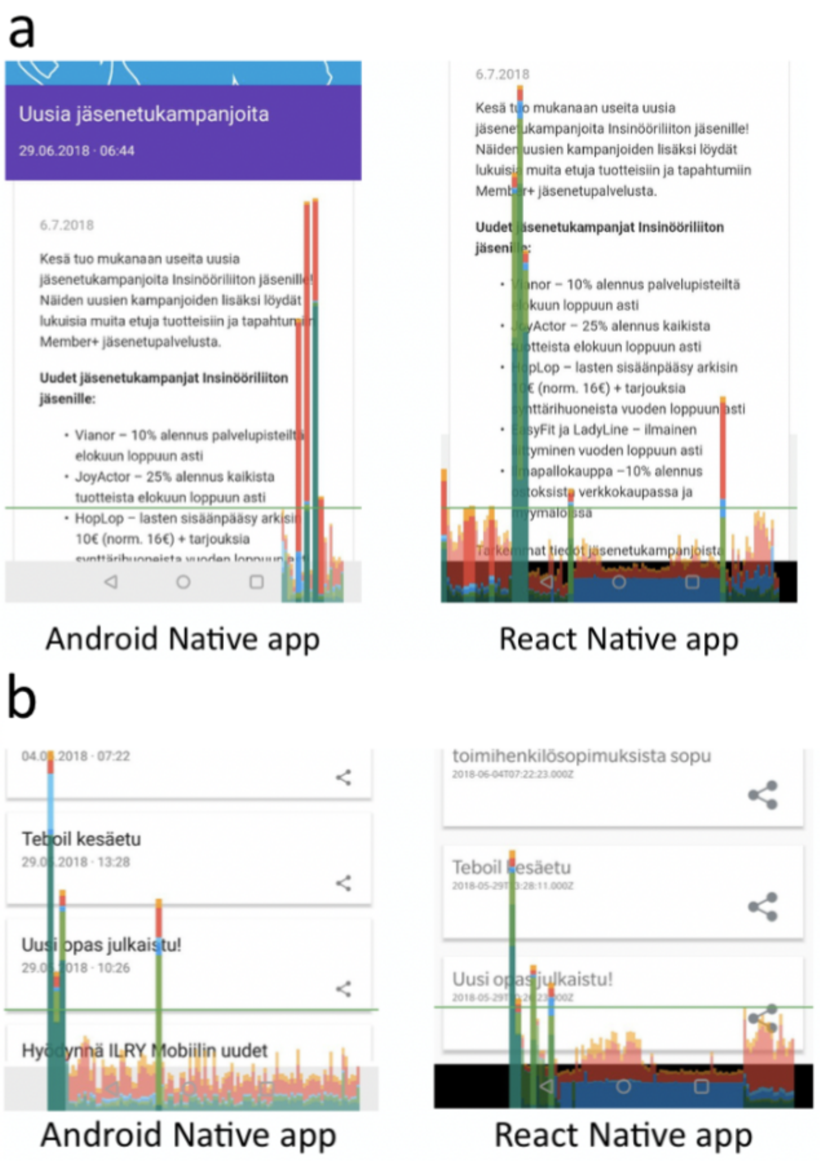


Figure 5.9: GPU Usage comparison for the webview loading test and News list loading test

5.3 Summary of results

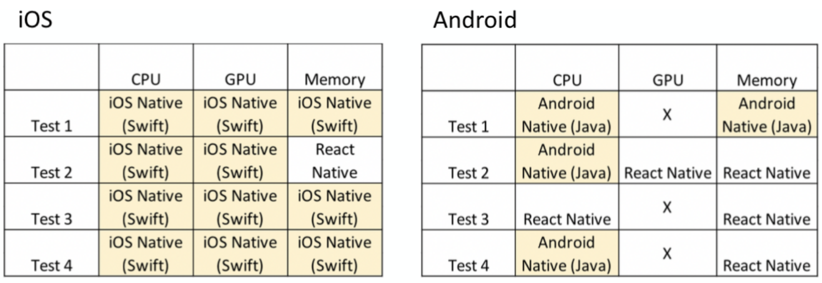
The previous sections presented extensive performance tests. The iOS and Android platform have several differences, not only in the implementation but also in the way the user can check the performance. The tools used to check the performance were different for each platform.

This section discuss the related findings and the reason why an application performed better than the other one. The goal is to discuss every measurement in details and come out with ideas that might help for future application projects. Moreover, we analyze the styling challenge found during the implementation process of the React Native applications.

The performance comparison between the Native iOS app and the React Native app is simple thanks to the Apple Instruments suite. The first tool we used is called “Time Profiler tool”; as explained in the previous chapter, this tool helps with the CPU measurements. The results of the CPU usage are very consistent; the React Native is always using a higher percentage of CPU during all the four tests. In this case, we can certainly say that the Native iOS solution is using the CPU better. The GPU usage test presents a different situation. The first two tests (the scrollview and the add workhour process), have similar results between the two applications. It is important to remember that the final value is an average of four different registration; a small difference is not enough to say that one application is performing better. Differently, in the last two test (Webview and News list loading) the Native iOS application has a significant advantage. Therefore, we can say that overall, the Native solution is using the GPU better. The last test is about memory usage. Likewise the previous two tests, the Native iOS application has an overall better performance. However, the second test (add workhour process) gave surprising results; the React Native application used less memory than the Native app. As seen in Section 5.1.3, Redux with the management of the state might have contributed to the result. In the React Native app, only the list of workhours reloads, while in the Native iOS application reloads all the view.

For the performance comparison between the Native Android app and the React Native app, we used two different tools (See Section 5.2). In the CPU measurements, the Native Android application is performing better in three tests. In the Webview loading test, the React Native application uses slightly less percentage of memory. Overall, the Native Android app uses the CPU more efficiently. The memory usage test gives interesting results. In general, the React Native app manages the memory better than the Native Android application. The results are similar, with the Native app that performs better only in the webview loading test. For the Android GPU comparison, we can only provide a visual idea thanks to the Profile GPU rendering tool. If we focus on the second test (add workhour process), React Native always renders more than 60 frames per second while the Native app has some edge cases. Apart from this test, the other results are relatively linear.

Figure 5.10 shows a summary of the performance comparison tests. The first table shows the results of the iOS comparison, while the second table presents the results of the Android comparison.



iOS

	CPU	GPU	Memory
Test 1	iOS Native (Swift)	iOS Native (Swift)	iOS Native (Swift)
Test 2	iOS Native (Swift)	iOS Native (Swift)	React Native
Test 3	iOS Native (Swift)	iOS Native (Swift)	iOS Native (Swift)
Test 4	iOS Native (Swift)	iOS Native (Swift)	iOS Native (Swift)

Android

	CPU	GPU	Memory
Test 1	Android Native (Java)	X	Android Native (Java)
Test 2	Android Native (Java)	React Native	React Native
Test 3	React Native	X	React Native
Test 4	Android Native (Java)	X	React Native

Figure 5.10: Results of the performance comparison tests

As we can see from the previous figure, the iOS development ecosystem with Swift is better than React Native in all three parameters we checked. However, both applications performed well; the React Native application has no issues during regular use of the application. The author of “Comparing the Performance between Native iOS (Swift) and React-Native” [22]implemented and compared a few basic features in both Native iOS and React Native. The implemented features are a facebook login, a to-do list, a tabbed interface and a map view. The considered parameters are the same we checked for this research (CPU, GPU, Memory usage). The difference lies in the results, where he was able to obtain leveled results with Swift that performed better in the CPU category, React-Native performed better in the GPU category (barely), and React-Native that performed better in the Memory category. The results of our iOS comparison are entirely different, React Native was never performing as good as the Native iOS application, but the tasks we performed are also more complicated.

Differently, the Android comparison presents some interesting findings. The Native Android application better manages the CPU usage, while the React Native application uses less memory to perform the tasks. The GPU measurements are mostly neutral, even though React Native has a better performance during the add workour process. We can eventually say that the Android development ecosystem and the React Native framework showed similar performance throughout all the tests. If we think that React Native is still a beta framework, these results are promising.

Besides pure performance, it is worth considering styling issues in React Native. React is advertised as "learn once, run everywhere", but developers and customers are still confused about this sentence. For this reason, the official React Native website has to state that "write once, run anywhere" is not what the framework offers [33]. Different platforms require style fixes; sometimes, during the development, a portion of code renders properly on one platform but is completely off on another one. Related to that, Figure A.2 shows an example of different styling result obtained during the implementation of the Insinööriliitto mobile application. The code for the styling of the cells is the same, but it looks as intended only for the Android platform.

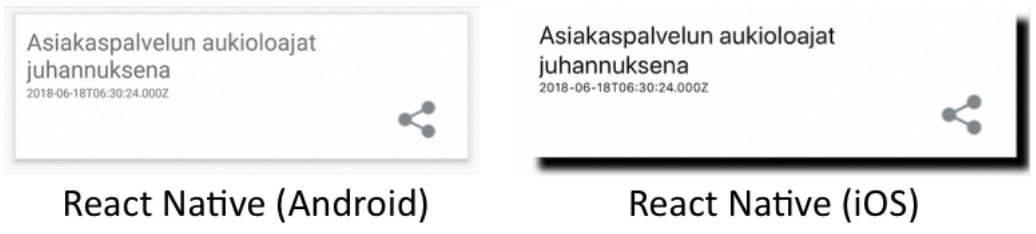


Figure 5.11: Styling issue during cells implementation

To fix the styling issues, React Native provides a library called "Platform". This library enables the developers to write different code specifying the target platform [32]. With this library we specified two different styling for the iOS and Android platforms implementation of the cells.

Chapter 6

Conclusion

In this thesis, we have seen the complete process of implementing the Insinööriliitto mobile application. We started from scratch, we went through the requirement engineering process and then the implementation. Both the Android and iOS application are available on the platforms application store, and the customer is happy with the general outcome of the project. With the performance evaluation, the goal was to analyze the React Native framework and answer the research questions presented in Chapter 1. We had interesting results; now the company has enough data to understand if they can use React Native for the development of future mobile applications for professional services. Based on the experimental results, a possible future approach would be to start testing both the applications with real users; before that, the React Native application needs some fixes on the design.

From the research point of you, we can say that the advantages of the development with the React Native framework do exist. The fact that React Native depends only on JavaScript is a fundamental aspect; developers do not need any other language to implement applications, this saves time and resources. For a company that focuses on web development, the React Native framework could result in a smooth start for implementing mobile apps without hiring any mobile developer.

Moreover, several mobile applications available in the market come from the React Native framework. The following are examples of popular applications developed with React Native: Instagram, Facebook, Bloomberg, SoundCloud, Gyroscope, and Delivery. The previous statement does not mean that the development of mobile applications is moving into the hybrid development world. Indeed, Gabriel Peal, an Android developer at Airbnb, released an article where he explained that due to a variety of technical and organizational issues, they were unable to meet their original goals, and they moved back from a React Native application to Native applications [14]. To conclude and answer the research questions, during our experiments React Native proved to perform well during all the tests. Therefore, the React Native framework demonstrated reliability and software companies should definitely take it into consideration for the development of professional membership services applications. However, there are pros and cons to both native development and hybrid development solutions, and the choice depends on the application to implement. The iOS development environment with Swift is still the best solution, but we are confident that React Native can only fill this gap in the years to come.

Bibliography

[1] React Native Navigation. `https://github.com/wix/react-native-navigation`.

[2] React Native Router. `https://github.com/aksonov/react-native-router-flux`.

[3] React Native tab-view: A cross-platform Tab View component for React Native. `https://github.com/react-native-community/react-native-tab-view`.

[4] React navigation. `https://reactnavigation.org/`.

[5] The node package manager (npm). `https://www.npmjs.com/`. Accessed 25.7.2018.

[6] Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998* (Oct 1998), 1–40.

[7] Airbnb. Native Navigation, 2017. `http://airbnb.io/native-navigation/`.

[8] Apple. Apple's performance-analysis and testing tool. `https://help.apple.com/instruments/mac/current/#/dev7b09c84f5`.

[9] Apple. iPhone X - Technical Specifications. `https://www.apple.com/lae/iphone-x/specs/`.

[10] Clancy, T. The Standish Group Chaos Report, 2014. `https://www.projectsmart.co.uk/white-papers/chaos-report.pdf`.

[11] Computer Hope. Computer vs. smartphone. webpage, January 24 2018. `https://www.computerhope.com/issues/ch001398.htm`.

[12] Ericsson. Ericsson mobility report june 2018. article, 2018. `https://www.ericsson.com/en/mobility-report/reports/june-2018`.

[13] Fortunato, D., and Bernardino, J. Progressive web apps: An alternative to the native mobile apps. In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)* (June 2018), pp. 1–6.

[14] Gabriel Peal, Android developer at Airbnb. Sunsetting React Native, 2018. `https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a`.

[15] Google - Android Developers. Activate Profile GPU rendering tool. `https://google-developer-training.gitbooks.io/android-developer-advanced-course-practicals/unit-2-make-your-apps-fast-and-small/lesson-4-performance/4-1a-p-profile-gpu-rendering/4-1a-p-profile-gpu-rendering.html`.

[16] Google - Android Developers. Inspect gpu rendering speed and overdraw. `https://developer.android.com/studio/profile/inspect-gpu-rendering.html`.

[17] Google - Android Studio. Memory allocations with Memory Profiler. `https://developer.android.com/studio/profile/memory-profiler`.

[18] Google - Android Studio. Real-time data of the application performance. `https://developer.android.com/studio/profile/android-profiler`.

[19] Hoda, R., Salleh, N., and Grundy, J. The rise and evolution of agile software development. *IEEE Software* (2018), 1–1.

[20] Hossain, E., Babar, M. A., and y. Paik, H. Using scrum in global software development: A systematic literature review. In *2009 Fourth IEEE International Conference on Global Software Engineering* (July 2009), pp. 175–184.

[21] Joey Cho, J. An exploratory study on issues and challenges of agile software development with scrum.

[22] John A. Calderaio. Comparing the Performance between Native iOS (Swift) and React-Native. `https://bit.ly/2fzpxls`.

[23] Joorabchi, M. E., Mesbah, A., and Kruchten, P. Real challenges in mobile app development. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement* (Oct 2013), pp. 15–24.

[24] KHALIL, M. A., AND KOTAIAH, B. Implementation of agile methodology based on scrum tool. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)* (Aug 2017), pp. 2351–2357.

[25] MATEEN, A., ABBAS, K., AND AKBAR, M. A. Robust approaches, techniques and tools for requirement engineering in agile development. In *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)* (Sept 2017), pp. 100–103.

[26] MEDEIROS, J., GOULÃO, M., VASCONCELOS, A., AND SILVA, C. Towards a model about quality of software requirements specification in agile projects. In *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)* (Sept 2016), pp. 236–241.

[27] MEGGIN KEARNEY (TECH WRITER), ADDY OSMANI (WEB DEVELOPER RELATIONS), KAYCE BASQUES (TECHNICAL WRITER FOR CHROME DEVTOOLS), JASON MILLER(WEB DEVREL)). Measure Performance with the RAIL Model, 2018. `https://developers.google.com/web/fundamentals/performance/rail`.

[28] MINH Q. HUYNH, PRASHANT GHIMIRE, D. T. Hybrid app approach: Could it mark the end of native app domination? vol. 14, pp. 049–065.

[29] NODE.JS FOUNDATION. JavaScript runtime framework. `https://nodejs.org/`. Accessed 25.7.2018.

[30] ONEPLUS. OnePlus 5T - Technical Specifications. `https://www.oneplus.com/it/5t`.

[31] RAMADAN, N., AND ZOHDY, B. Goal modeling techniques for requirements engineering. 739–746.

[32] REACT NATIVE, FACEBOOK. Platform Specific Code. `https://facebook.github.io/react-native/docs/platform-specific-code`.

[33] SOPHIE ALPERT. Introducing React Native. `https://reactjs.org/blog/2015/03/26/introducing-react-native.html`.

[34] VAN LAMSWEERDE, A. Goal-oriented requirements engineering: a guided tour. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering* (2001), pp. 249–262.

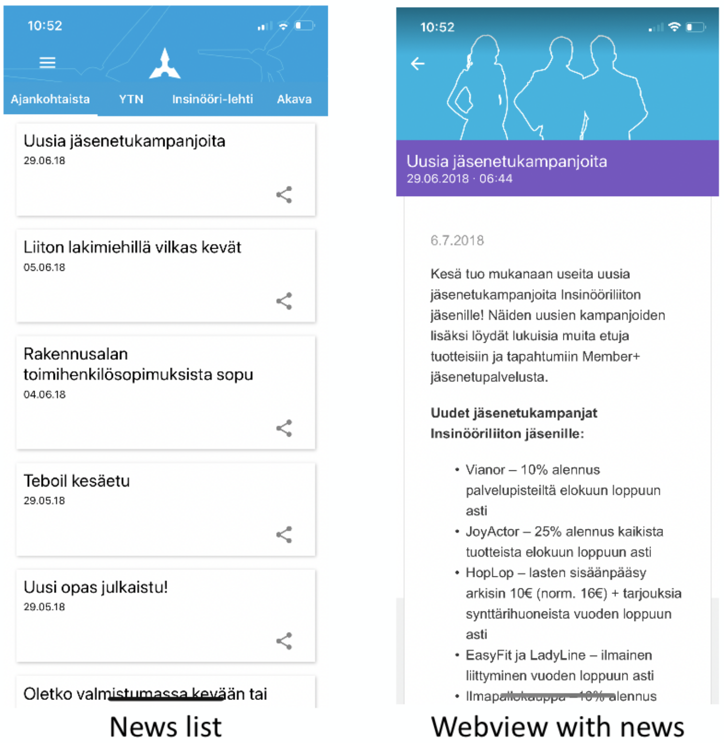
[35] Xanthopoulos, S., and Xinogalos, S. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics* (New York, NY, USA, 2013), BCI '13, ACM, pp. 213–220.

[36] Yu, E. S. K. Towards modelling and reasoning support for early-phase requirements engineering. In *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on* (Jan 1997), pp. 226–235.

Appendix A

First appendix

Appendix A shows all the implemented functionalities of the application. The views presented are taken from the Native iOS application, but the same functionalities have been developed for the Native Android application and the React Native application.


10:52

Ajankohtaista YTN Insinööri-lehti Akava

Uusia jäsenetukampanjoita
29.06.18

Liiton lakimiehillä vilkas kevät
05.06.18

Rakennusalan toimihenkilösopimuksista sopu
04.06.18

Teboil kesäetu
29.05.18

Uusi opas julkaistu!
29.05.18

Oletko valmistumassa kevään tai

News list

10:52

Uusia jäsenetukampanjoita
29.06.2018 · 06:44

6.7.2018

Kesä tuo mukanaan useita uusia jäsenetukampanjoita Insinööriliiton jäsenille! Näiden uusien kampanjoiden lisäksi löydät lukuisia muita etuja tuotteisiin ja tapahtumiin Member+ jäsenetupalvelusta.

Uudet jäsenetukampanjat Insinööriliiton jäsenille:

- Vianor – 10% alennus palvelupisteiltä elokuun loppuun asti
- JoyActor – 25% alennus kaikista tuotteista elokuun loppuun asti
- HopLop – lasten sisäänpääsy arkisin 10€ (norm. 16€) + tarjouksia synttärihuoneista vuoden loppuun asti
- EasyFit ja LadyLine – ilmainen liittyminen vuoden loppuun asti
- Ilmapallokauppa – 10% alennus

Webview with news

Figure A.1: Applications functionalities

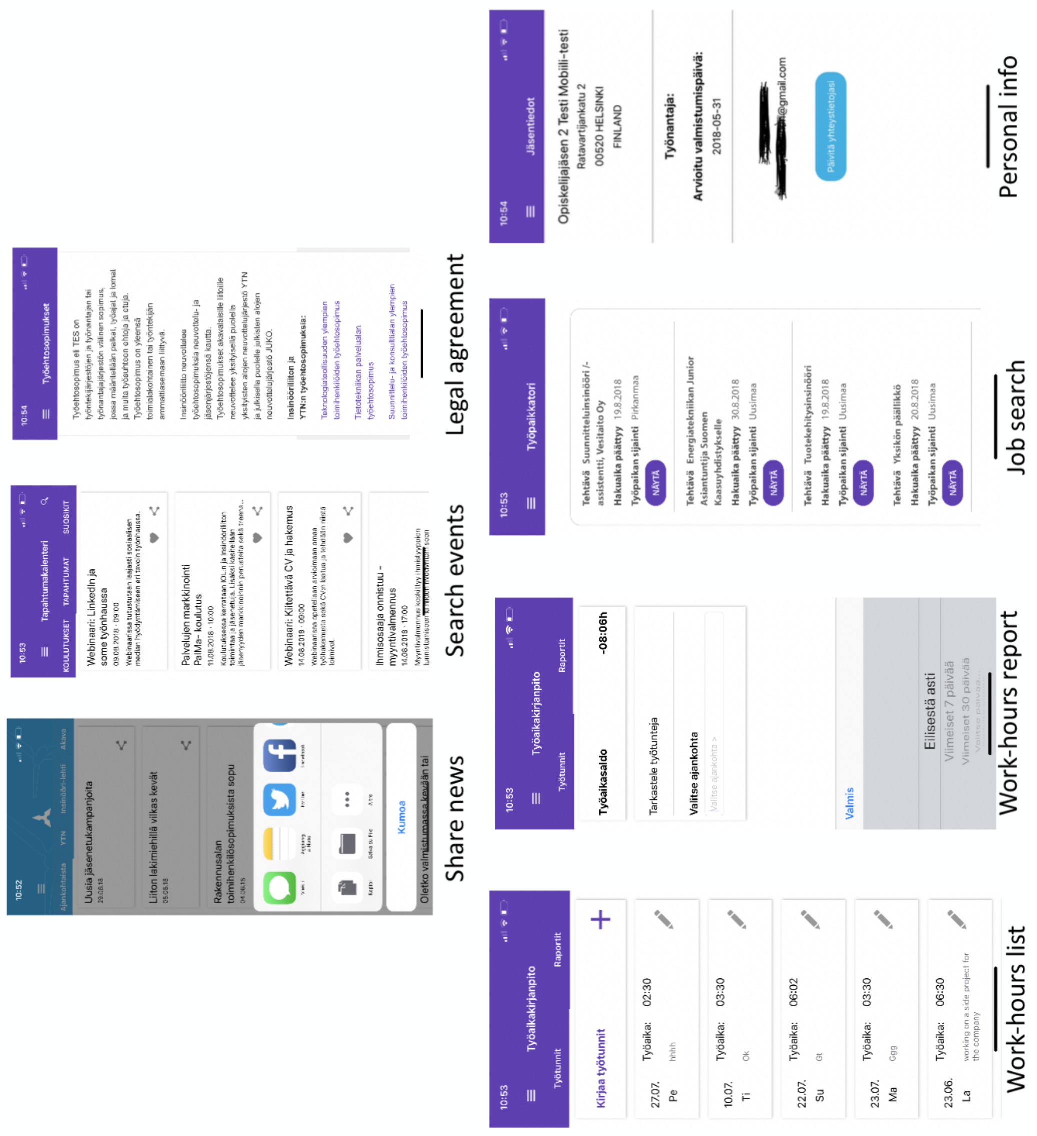


Figure A.2: Applications functionalities