Sauli Rajala

TAKING THE WEB BEYOND THE WEB: BUILDING CROSS-PLATFORM SERVICES USING WEB CMS AS A BACKEND

UNIVERSITY OF JYVÄSKYLÄ
DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS
2019

ABSTRACT

Rajala, Sauli
Taking the web beyond the web: building cross-platform services using web CMS as a backend
Jyväskylä: University of Jyväskylä, 2019, 89 p.
Information Systems, Master's Thesis
Supervisor: Semenov, Alexander

This paper proposed a UI-framework, which can be used to create cross-platform service from existing web service. Special attention was paid to the user experience in the context of whole cross-platform service. This paper also proposed a new taxonomy for classifying native mobile applications and approaches to cross-platform development. The study was done using design science as research method, and the developed artifact was demonstrated with WordPress and React Native. Existing WordPress website was analyzed, and a part of the website was converted into mobile user interface. Then mobile application was developed with React Native. This led to cross-platform service with two separate platforms (website and native mobile application) using the same data and having the same functionalities. UI-framework was noticed to be functional but insufficient. For example, the user experience in the platform itself should be elaborated on in more details. The new taxonomies for mobile applications and cross-platform development approaches was noticed to be true in the demonstration application. However, when examined in broader scale, taxonomies were found to be insufficient. As conclusion it was noted that this kind of configuration of cross-platform service sets some limitations for both the backend and the mobile application itself. For example, the data should be in structured form in the website.

Keywords: React Native, WordPress, mobile development, user experience, cross-platform service, mobile application, design science

TIIVISTELMÄ

Rajala, Sauli
Webin vienti webin ulkopuolelle: useamman alustan palvelun rakentaminen käyttäen web CMS:ää backendinä
Jyväskylä: Jyväskylän yliopisto, 2019, 89 p.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Semenov, Alexander

Tässä tutkimuksessa esiteltiin viitekehys, jota voidaan käyttää useamman alustan palvelun luomiseen olemassa olevan verkkosivuston varaan. Erityistä huomiota kiinnitettiin useamman alustan palvelun käyttökokemukseen. Tässä tutkimuksessa esiteltiin myös taksonomiat, joilla voidaan luokitella erityisesti natiivi mobiilisovelluksien teknistä luonnetta sekä lähestymistapoja useamman alustan mobiilisovellusten kehittämiseen. Käytetty tutkimusmenetelmä oli suunnittelutiede (englanniksi design science) ja kehitetty artefakti demonstroitiin WordPressillä ja React Nativella. Osa olemassa olevan WordPress-sivuston käyttöliittymästä muutettiin mobiili käyttöliittymäksi ja lopuksi sovellus ohjelmoitiin käyttäen React Nativea. Näin saatiin aikaan useamman alustan palvelu, jossa hyödynnettiin verkkosivuston dataa ja toiminnallisuuksia. Kehitetty artefakti todettiin toimivaksi, mutta riittämättömäksi. Esimerkiksi enemmän huomiota olisi tullut kiinnittää käyttökokemukseen mobiiliialustassa itsessään. Tutkimuksessa todettiin myös, että tällainen useamman alustan palvelun rakenne asettaa tiettyjä vaatimuksia sekä backendiin että mobiilisovellukseen itseensä. Yksi tällainen rajoitus on, että verkkosivuston data tulee olla strukturaalisessa muodossa.

Avainsanat: React Native, WordPress, mobiilikehittäminen, käyttökokemus, useamman alustan palvelu, mobiilisovellus, suunnittelutiede

FIGURES

TABLES

TABLE OF CONTENT

INDEX OF CONCEPTS

Ajax	Asynchronous JavaScript And XML
Backend	Part of computer system that is responsible for storing data
CLI	Command-line interface
GraphQL	Query language for APIs
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MySQL	Relational database management system
URL	Uniform Resource Locator
PHP	Programming language
Rest Api	Representational State Transfer Application programming interface

1 INTRODUCTION

Nowadays more and more people have mobile devices and with increasingly speed they expect to be able to find the same information and do the same things with mobile devices that they previously did with desktop computers. This demand of mobile services places the service providers like cities, news companies, etc. into difficult situation. At the same time, it also gives huge possibilities to provide services in a new and more effective way. One of the biggest problems here is the money. Building native mobile applications is hard and time-consuming work. You need people with special coding skills in different programming languages and you need to develop different applications to different mobile platforms. Because of this many service providers are hesitating to step into the mobile world. At the same time, they need to develop and to update their websites to answer the growing requirements for websites. This makes the step into the mobile world even more frightening to take.

These practical problems lead to a few practical questions: Could it be possible to combine mobile and web worlds together in a cost-effective and reliable way? To update the content of the website and of the mobile application with the same familiar content management system (CMS) that you use to update website? Would it be possible to do this natively and with cross-platform support? These very practical questions lead to more theoretical questions about the nature of that kind of service. What kind of system would we actually be building? How to maintain the consistency between website and mobile application? What is the theory or theoretical framework behind this?

1.1 Research question

My research question is "Under which conditions can you build native mobile applications using cross-platform technologies and Web CMS as a backend and how to do it using React Native and WordPress?"

This research question can be divided into three sub question, that specifies the research question. For the sake of clarity these sub questions are referred in this thesis as sub question 1, sub question 2 and sub question 3. Sub question 1 is, "What is the framework to use when creating mobile user interface from web user interface in order to produce cross-platform service so that cross-platform user experience is taken into account?" Sub question 2 is, "What is the framework to classify different applications and cross-platform development approaches?" Sub question 3 is, "How to do it using React Native and WordPress?".

The research method of this study is design science and chapter 2 introduces the research method in detail.

1.2 Objectives of a solutions

Existing definitions for mobile applications and for cross-platform development approaches are somewhat misty and does not take into account advanced modern technologies such as React Native. As a result of this thesis, we will have a rigor framework for classifying different mobile applications and approaches to cross-platform development. Another goal of this study is to develop a framework, which web developers around the world can use to create cross-platform services from their existing website cost-effectively and with high quality.

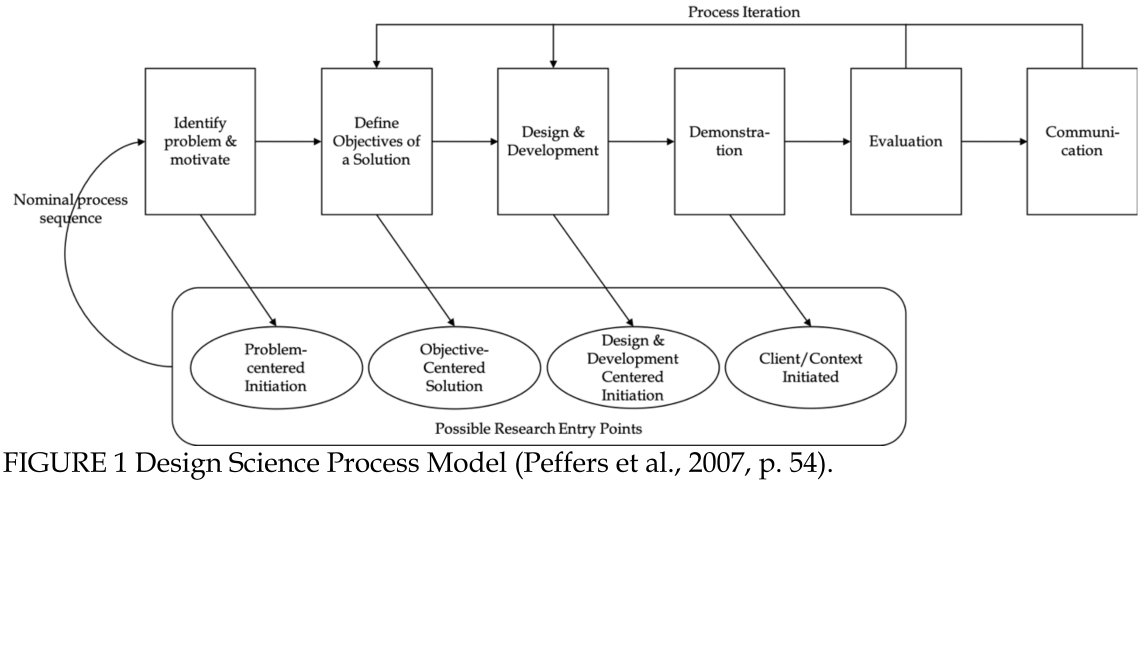
WordPress is the most popular web CMS with 60% market share of all the websites whose content management system is known and React Native is trending as a new technology to build cross-platform mobile application ("Usage statistics", 2019). Both of these technologies are open source and have a big and committed community of developers. As a result of this study, we would get a good knowledge of how to use these two technologies to build cross-platform services and also one demonstration of the frameworks described above.

The rest of the thesis is structured as follow. In chapter 2 the design science is explained in more details. The background theories for this thesis is covered in chapters 3-6. In chapter 3, the concept of cross-platform service is explained and after that, in chapter 4, the concepts relating to human-computer interaction are explained and applied in the context of cross-platform service. After that the existing classifications for mobile applications is examined and lastly common definitions for cross-platform development is given in chapter 6. After that the artifact is designed and developed in chapters 7-8 and in chapters 9-10 this artifact is demonstrated with React Native and WordPress. Lastly the artifact is evaluated in chapter 11. Short discussion about the findings of this study is done in chapter 12, where the answers to research question is also provided.

2 DESIGN SCIENCE

The research method in this thesis is design science. According to Hevner, March, Park and Ram (2004) the design science paradigm creates knowledge of a problem and its solution by building and implementing new and innovative artifact. It is a problem-solving paradigm, where focus is laid in creating and evaluating artifacts, which are meant to solve different kinds of problems. The creation of these artifacts is dependable from existing kernel theories, which the researcher applies, test and extend in order to solve the problem. (Hevner et al., 2004.) Markus, Majchrzak and Gasser (2002) defines the kernel theory to be fundamental for design science and it can be either academic or practical theory.

In this thesis, I will use the design science process model (figure 1) introduced by Peffers, Tuunanen, Rothenberger and Chatterjee (2007). This model is created as a consensus from influential prior research on design science. It serves as a commonly accepted framework for phasing studies with design science research method. Process model has two key concepts: activities and entry points. Activities is introduced in next chapter in more detail and entry points is discussed shortly in chapter 2.2.


Process Iteration
Identify problem & motivate
Define Objectives of a Solution
Design & Development
Demonstra-tion
Evaluation
Communi-cation
Nominal process sequence
Problem-centered Initiation
Objective-Centered Solution
Design & Development Centered Initiation
Client/Context Initiated
Possible Research Entry Points
FIGURE 1 Design Science Process Model (Peffers et al., 2007, p. 54).

2.1 Activities

Peffers et al. (2007) divides the design science into six activities: problem identification and motivation, define the objectives for a solution, design and development, demonstration, evaluation and communication. In this chapter each activity is defined and also specified the corresponding chapter of this thesis, since the structure of thesis follows the activities.

First activity is to identify the research problem and to motivate why it is important to seek solution to this problem. Hevner et al. (p. 81, 2004) call these problems as wicked problems, which are characterized for example by unstable requirements and complex interactions among subcomponents of the problem and its solution. The research question should be introduced in this activity and broken into smaller concepts. This helps to make sure that the solution is really the solution for all the aspects of the research question. Another important phase in this activity is to define why it is important to find solution to the problem. This will motivate both the researcher and the audience to seek and to accept the solution. (Peffers et al., 2007.) In this thesis, the problem is identified and motivated in chapter 1.

Second activity is to define objectives of a solution. According to Hevner et al. (p. 84, 2004) general objective of design science research is to enable development and implementation of solutions to unsolved and important business problems. In design science process model the objectives of solution can be derived from the previous activity i.e. by identifying the problems. These objectives can be either qualitative or quantitative. (Peffers et al., 2007.) Second activity is implemented in chapter 1.2 in this thesis.

Third activity is to design and develop the artifact. Artifact can be for example a construction, model, method or instantiation. They are rarely full-grown information systems, but more likely innovations that define practices and technical capabilities so that use of information systems can be effectively and efficiently accomplished. (Hevner et al. 2004.) The research contribution should be embedded in designing the artifact. In this activity researcher need to specify functionalities and architecture of the artifact. After that, he or she can develop the artifact. (Peffers et al., 2007.) The artifact is designed and developed in the chapters 7 and 8 in this thesis.

Fourth activity is the demonstration part, where researcher shows how the artifact works at least in one instance of the problem. Demonstration can be for example experimentation, simulation or case study. (Peffers et al., 2007.) This activity is applied in chapters 9 and 10, where the artifact will be demonstrated in real life situation.

Fifth activity is the evaluation of the artifact. Here the objectives of solution from activity two are compared to the results received from demonstration of the artifact. In order to evaluate the artifact rigorously researcher needs to consider what the relevant metrics or analysis methods are. Evaluation can be a comparison between expected results and received results or quantitative or qualitative performance measurements. Evaluation can also happen in terms of completeness, consistency or usability of the artifact. Researcher can decide whether to iterate back to third activity or not before moving to sixth activity. (Hevner et al., p. 85, 2004; Peffers et al., 2007.) The evaluation of the artifact takes place in chapter 11 in this thesis.

Sixth and last activity is to communicate the study to other researchers and to other relevant groups. In this activity, the researcher reflects previous activities and writes down the important points from each activity. What is the problem and why is it relevant? What is the artifact that was developed? Is it new and feasible? Was the design process rigor? (Peffers et al., 2007.) Research must be presented in manners that will benefit both technology- and management-oriented audiences. For technology-oriented audience research needs to include adequate detail for implementing the artifact. Research needs to also offer sufficient data for management-oriented audience to determine if the artifact is fit for their specific organizational context. (Hevner et al., 2004.) This study is communicated by publishing the thesis at the Jyväskylä University Digital Repository.

2.2 The entry points

Even though the above activities are numbered and presented in sequential order, there is no need for the researcher to always start from activity one. As is seen in figure 1 there are four entry points for researchers to start their research. *Problem centered initiation* can be triggered for example from suggested future research ideas from prior research literature. Researcher can also start the research by defining the objectives of a solution, if the idea for research comes from the need of industry that can be addressed by developing an artifact. The entry point for these kinds of researches would be *objective-centered solutions*. Third option is to start with the artifact and move outward. *Design and Development centered initiation* happens for example from already existing artifact that has come from another research domain but has not yet been formally studied. Last entry point is called *client or context initiated*. Here the idea for research can be a result of observing a practical and working solution. (Peffers et al. 2007.)

The idea for this research comes from already existing solution to the research problem offered by Gould (2015). He introduces the idea and practical steps on creating simple mobile application with React Native and WordPress as backend. This observation of practical solution serves as a starting point to the research in order to find out what the artifact for the demonstration is and how to evaluate that artifact.

3 CROSS-PLATFORM SERVICE

This chapter consists of discussion on cross-platform service. First it provides the definition of cross-platform service. Second part of this chapter presents the possible configurations of cross-platform service. Last part of this chapter presents the somewhat similar concept than cross-platform service – multiple user interfaces.

Even though the emphasis in this thesis is mostly in web and mobile, cross-platform services enables you to create whole ecosystem around the web CMS. However, in this thesis, the attention is mostly drawn to taking the web beyond the web into native mobile application

3.1 The definition

Cross-platform service can be divided into two concepts. First concept is *service*, which can mean many things depending on the context. In the field of economy and business it is defined as a process, which consists of a series of intangible activities so that the solutions to customer's problems are provided. (Lusch & Vargo, 2006; Grönroos, 2007, p. 52; Kotler & Armstrong, 2012, p. 248.)

In this thesis the service-concept is applied digitally. Williams, Chatterjee and Rossi (2008) define digital service as a benefit that one can give to another through a digital transaction over Internet Protocol. There are two key roles in this process: service provider and service user. Service provider is the one responsible for providing the service to the service user who uses the service. (Williams et al., 2008.)

Cross-platform services are generally web-based services and the configurations of cross-platform services will be explained in detail in chapter 3.2. Another characteristic of cross-platform service is that users can interact with it through multiple devices. These interactions take place through user interfaces and in the context of cross-platform services these user interfaces can also be called multiple user interfaces. The concept of multiple user interfaces is explained in more detail in chapter 3.3. (Wäljas, Segerståhl, Väänänen-Vainio-Mattila & Oinas-Kukkonen, 2010.)

3.2 Configurations of cross-platform service

There are two main points of view in the configurations of cross-platform service: device composition and service delivery. In this chapter I will explain these standpoints in more detail.

Device composition can be divided to two: device redundancy and the synergistic specificity of a service. *Device redundancy* describes the roles of devices in the cross-platform service. The roles can be redundant, complementary or exclusive depending on how much data and functions each device share with each other. The *synergistic specificity of a service* describes the dependency between devices in a cross-platform service: the higher the synergistic specificity, the greater the dependency. Device composition is one of the important aspects in a user experience of cross-platform services, because it helps to limit the use cases to only those use cases, where the service is built to support by its configuration. (Wäljas et al., 2010.)

The service delivery can be either multichannel or crossmedia. *Multichannel* means that content and functionality is equal or almost equal in multiple devices. The role of devices here is mostly redundant or complementary, so each device offers the same or almost the same content and functionalities. On the contrary, in *crossmedia systems* the role of devices is mostly complementary or exclusive. Unlike in multichanneled systems, the synergistic specificity is high in crossmedia systems. They become fully functional only when each component is used together with other components. (Wäljas et al., 2010.)

3.3 Multiple user interfaces

3.3.1 User interface

When people talk about softwares and applications they usually are talking about user interfaces. User interface (UI) is the part of the software that is visible to people. UI has two main components: input and output. Person communicates with the UI through input mechanism, which can be for example keyboard, mouse or person’s finger. Application interprets the users input and expresses the results of its computations to output. The most common output mechanism today is the display screen. (Galitz, 2007, p. 4; Olsen, 1998, p. 11.)

The main purpose of the UI is to view and edit information – to interact with a set of visible objects. Through these objects user can perform operations, in other words actions, to complete the tasks, which the user had in mind when starting to use the UI and the application in the first place. (Galitz, 2007, p. 16.)

3.3.2 Multiple user interfaces

Multiple user interfaces (MUI) was first introduced by Seffah and Javahery (2004). It is an interactive system, that provides access and views to the same information and services through different platforms and also coordinate these services. Each of these views should be developed with the conditions of a platform, but at the same time maintain cross-platform consistency and usability. Information and services can locate on a single server or in a distributed system and different views can access them using client/server protocol or direct peer-to-peer connection. These views can be considered as a platform specific UIs, which could differ in interaction styles, features or even in the displayed information. (Seffah & Javahery, 2004, pp. 11-13.)

There are four main aspects that characterize MUI-systems. First aspect is its ability to enable user to interact with server-side system by different interaction styles like gestures or mouse. The second aspect is, that it is possible for user of MUI to achieve their single objective, consisting of multiple interrelated tasks, using multiple platforms and devices. For example, user can book an appointment with the mobile phone and email information about the appointment with desktop computer. Third aspect is, that features, functions and information behave the same across all the platforms, in spite of the fact that every platform has its own unique look and feel. Last aspect of the MUI is that it feels like a set of platform specific versions of the single interface with the same capabilities. (Seffah & Javahery, 2004, pp. 11-13; Majrashi, Hamilton & Uitdenbogerd, 2015.)

Majrashi et al. (2015) categorize different MUIs into three model: on-demand, independent and hybrid. *On-demand model* means that the service and the information is located in a single repository and user access this repository by request in web browser. This model can be seen to correlate with multichannel system in cross-platform service. *Independent model* refers to distributed service model, where each view of the MUI can be seen as an independent UI for the platform and thus the function and information can be different between the views of the MUI. This model correlates with crossmedia systems in cross-platform service. *Hybrid model* is a combination of these two models meaning, that it includes both the services, that need to be accessed through web browser and the services, that are installed on device. (Majrashi et al., 2015.)

4 USER EXPERIENCE IN A CROSS-PLATFORM SERVICE

Most definitions of user experience (UX) only consider the user experience of one UI and not the wider UX of the cross-platform service. In this chapter I will first define the concept of UX and apply the definition to the context of MUI and cross-platform services. After that I will look more closely the characteristics of UX in a cross-platform service.

4.1 Usability and user experience

In a common language user experience and usability is in many times used as a synonym for each other. Although there is no common and rigor understanding of these two concepts, there is a common understanding that they mean different things. In this chapter I will first define the concept of usability in general and in the context of MUI and cross-platform service. After that I will define the concept of UX and UX in MUI and cross-platform services. Lastly, I will reflect the differences between usability and UX.

4.1.1 Usability

Usability is a difficult concept to define, because there is no common definition for it. In its simplest form, usability can be defined as a variable that tells how easy it is to use the UI. (Majrashi et al., 2015.) More complex definitions combine multiple attributes such as effectiveness, efficiency and satisfaction. For example, according to Jokela, Iivari, Matero and Karukka (2003) ISO 9241-11 standard defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". Nielsen (2012) divides usability into five components: learnability, efficiency, memorability, errors and satisfaction. Learnability takes the perspective of the first-time user and describes the easiness for this user to accomplish basic tasks with the design. Efficiency, on the other hand, takes the perspective of a more experienced user and describes how quickly this user can perform tasks. Memorability takes the perspective of the returning user and describes the easiness for this user to restore his/her skills relating to the design. Errors-component defines the quality and quantity of errors made by user. Lastly satisfaction-component describes the pleasantness of use of the design. (Nielsen, 2012).

Traditional usability definitions do not consider the usability in the context of cross-platform service. The usability in the MUI-systems can be divided into two dimensions: horizontal and vertical. Horizontal usability refers to the cross-platform usability requirements and vertical usability refers to the platform specific usability requirements. (Seffah & Javahery, 2004, p. 16.) According to Angulo and Ferre (2014) respecting the platform conventions (vertical usability) is more important in mobile applications than it is in desktop applications.

Wäljas et al. (2010) find three categories in usability in cross-platform services: inter-device consistency, transparency and adaptability. Inter-device consistency should be high enough, but not too high as in some cases there should be heterogeneity of functionalities. Transparency means how transparent the system is about its own structure and functionalities. The more transparent structure the clearer it is for user to understand the limitations of the system. Here one should especially think about how to increase knowledge about combining web and different kind of mobile devices in useful ways. Adaptability consists of two aspects: how well the system helps the user to learn and to use different components of the system in different use situations and how well the system adapts to the user's environment and to the properties of the device. (Wäljas et al. 2010.)

4.1.2 User experience

The concept of user experience can be divided into two term: user and experience. There is two meaning for the term *experience*. The saying "learning by experience" represent the first one. In this kind of understanding of the term, knowledge and skill is gained by doing and/or seeing things. The second meaning refers to the memorable activity that provided an experience, which affected in positive or negative way to people who was involved in the activity. *User* is someone who uses something and together these terms refers to memorable usage activity, that remains in the mind of the user as a reference of usage situation and context. The first meaning of experience – learning by experience – also annotates with user experience. If the user experience is good, the usage event will be most likely repeated, but if the usage event leads to negative outcome, user will avoid it in the future. He/she had learned by experience whether or not the design was worth the time. (Pallot & Pawar, 2012.)

Another way to think about user experience is to think the following five elements: usability, utility, availability, aesthetics and offline issues. What the user expects from these five elements and how the design response to these elements, are the key issues to produce a good user experience. (Hiltunen, Laukka & Luomala, 2002, p. 13). Majrashi et al. (2015) also think user experience (UX) as a parent concept for more specific variables like usability, emotional, aesthetic and values. The first requirement for UX is to meet the needs of the customer so that it is joyful to use the service, but the UX goes far beyond that. UX can be considered as something that includes all the aspects of user's interaction with the service. (Norman & Nielsen, 2016.) According to Law, Roto, Hassenzahl, Vermeeren and Kort (2009) ISO standard define user experience as "a person's perceptions and responses that result from the use or anticipated use of a product, system or service". Law et al. (2009) agree in general with this ISO definition and they use the following attributes to describe user experience: dynamic, context-dependent, subjective and individual (instead of social). User experience can be thought to be something that take shape when user interact with a product, system, service or an object (Law et al., 2009.)

According to Wäljas et al. (2010) the traditional aspect to UX has been device-based, but recent studies have taken more holistic view to UX in relation to service user experience, UX life cycles and distributed or crossmedial UX. Hassenzahl and Tractinsky (2006) follows this more holistic view and define UX as an combination of a user's internal state (e.g. expectations and needs), the context of use (e.g. organizational setting and meaningfulness of the activity) and the characteristics of the designed system (e.g. complexity and usability). When considered MUI-systems the last aspect can be divided into two categories: system wide characteristics and platform specific characteristics. (Majrashi et al., 2015.) Wäljas et al. (2010) remind that previous and expected experiences with other components in the system remarkably affects the UX of distinct system component.

4.1.3 The differences between usability and UX

As can be seen from the previous definitions, usability and UX sounds quite the same. Nevertheless there are a few differences, which are explained in more detail in this chapter.

First difference is in the wideness of the definition. UX is thought to be broader than the usability meaning that in the UX the usability is not the only aspect to consider. The more emotional aspect of system quality is associated with UX although users' internal state effects also the usability. The second difference is the point of view. Usability concentrates more on a design of a system, but in UX the point of view is more of the users. Third difference is the measurement meter. Usability can be measured by both objective (for example task execute time) and subjective meters (for example satisfaction rate), but the meter for UX measured is based on subjective meter. (Majrashi et al., 2015; Wäljas et al. 2010.)

It is also interesting to notice that usability and user experience does not always correlate with each other: bad usability does not necessarily result to poor user experience. For example, in the study done by Tulimäki (2015) with 24 participants the result was that even though the usability of iPhone phones was worse than in the Nokia phones, the overall user experience in iPhone was considered better than in the Nokia. One reason for this might be that according to Quinn and Tran (2010) the phones attractiveness compensates for ineffective and inefficient performance or as Tulimäki (2015) put it: users value more hedonistic attributes than the objective usability. Another reason might be, as Raita and Oulasvirta (2011) argue, that users bring their expectations and previous experiences with the phone to the test situation. Raita and Oulasvirta (2011) noticed in their study that if user had read positive product review, he was more likely to give positive ratings after the usability test even if user had failed in most of the tasks in usability test.

4.2 User experience of cross-platform service

Wäljas et al. (2010) introduce a framework for cross-platform service UX. Framework presents three themes of UX and each theme is broken down into three characteristics of cross-platform system, which influences into one element of cross-platform service UX. These elements of cross-platform service UX are fit for cross-platform tasks, flow of interactions and content and lastly perceived service coherence. (Wäljas et al., 2010.) Figure 2 summaries the idea of the framework.

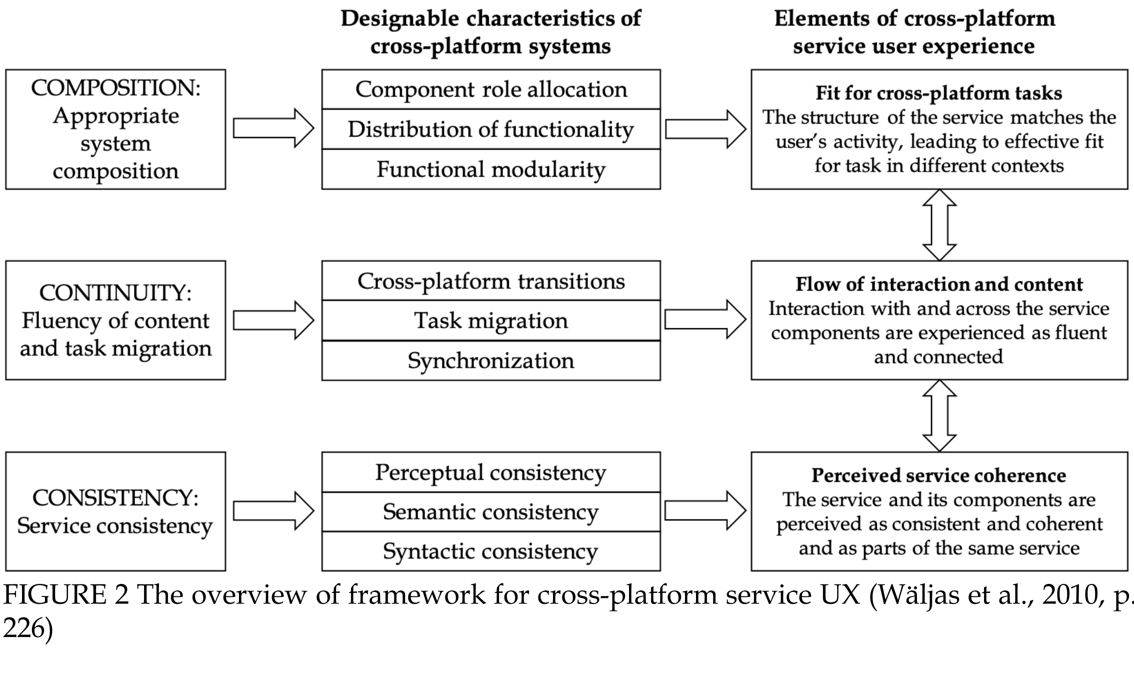

Designable characteristics of cross-platform systems
Elements of cross-platform service user experience
COMPOSITION: Appropriate system composition
Component role allocation
Distribution of functionality
Functional modularity
Fit for cross-platform tasks
The structure of the service matches the user's activity, leading to effective fit for task in different contexts
CONTINUITY: Fluency of content and task migration
Cross-platform transitions
Task migration
Synchronization
Flow of interaction and content
Interaction with and across the service components are experienced as fluent and connected
CONSISTENCY: Service consistency
Perceptual consistency
Semantic consistency
Syntactic consistency
Perceived service coherence
The service and its components are perceived as consistent and coherent and as parts of the same service

FIGURE 2 The overview of framework for cross-platform service UX (Wäljas et al., 2010, p. 226)

User experience will be better both for the whole cross-platform service and its distinct platforms, if the system is composed in line with the user's primary activity and expectations, e.g. the service fits for cross-platform tasks.

Composition-theme specifies the relationships between different platforms within a system and component role allocation, distribution of functionality and functional modularity are parts of composition-theme. The role that the user gives to component is significant, because that determines what the user thinks about the purpose and functionalities of component. *Role allocation* can be either task or situation-based. Task-based means that different platforms is used to do different task. Situation-based allocation, on the other hand, means that the same task is done by different devices depending on the situation. Understanding these situations helps to optimize role of devices in a cross-platform service system. Second aspect that belongs to the composition-theme is *distribution of functionality*, which can base on, for example, platforms' individual strengths or assumed use situation. If the functionality of cross-platform service is distributed to the distinct devices, the complexity of an individual device may decrease. The other side of the coin is that limitations to the functionality in certain platforms may be in contradictions with the user's expectations. To prevent these contradictions certain amount of *functional modularity* should be taken into account. This means that at least the core functionalities should not be situated on only one platform, because if that platform is not available, the whole system can become dysfunctional. (Wäljas et al., 2010.)

User experience will be better when the users are able to smoothly and intuitively carry out the tasks and activities across various platforms. This requires fluent flow of interaction and content. *Continuity-theme* determines how seamless is the synchronization of data and content within a system and how good is the support for users to shift to different platform in the middle of a task. Continuity between devices comes through cross-platform transitions, task migration and synchronization. In *cross-platform transitions* the focus is set on to the explicit and intuitive connection between devices and applications. The mere possibility to interconnect devices is not enough. Interconnection should be natural and do-nothing experience. *Task migration* means, that the user is able to continue the incomplete task on another device. If cross-platform service is composed in a crossmedial way, the focus should be on the logical chain of tasks. On the other hand, in multichanneled configurations more emphasis should be put on repetition of tasks: the task should be repeatable on different device. To make this possible, you need *synchronization of actions and content*. The state of actions and content should be the same even when switching the device in the middle of task. (Wäljas et al., 2010.)

The way the system and the distinct platforms present the same information and the functionalities is also important to user experience. Perceived service needs to be coherent. This is the area of the *consistency-theme*, which determines how coherent the symbols, terminology, interaction logic, etc. are within the system. The focus here is set on a perceptual, semantic and syntactic consistency. *Perceptual consistency* means that the look and feel should be similar across all platforms in cross-platform service: the same action or content should be presented in a consistent way. *Semantic consistency* means that the same terminology and symbols are used to represent the same functionalities. Lastly, *syntactic consistency* means that the interaction logic (for example navigations) is coherent through the devices. (Wäljas et al., 2010.)

5 WHAT IS MOBILE APPLICATION?

According to Zibula and Majchrzak (2013) mobile applications are technologically just ordinary computer programs. Mobile application runs in a mobile operating system (OS) in a mobile device such as smartphone or tablet. In the prior research mobile applications are classified into three categories. These categories are explained in more detail in this chapter.

5.1 Native application

First category of mobile applications is native application. Joorabchi, Mesbah and Kruchten (2013) define native application as an application that runs on a device's operating system and is system specific – meaning that application for iOS platform need to be developed separately from application for Android platform. Charland and Leroux (2011) argue that native applications are usually compiled, and they build user interface straight through devices' own functionalities and views.

Le Goaer and Waltham (2013) gives the following specifications to native mobile applications:

- User interface is recognizable and associated to certain OS meaning that the application "has the look and feel of the platform" (Heitkötter, Hanschke and Majchrzak, 2013).
- Application can access to other applications already installed on the systems (address book, photo gallery, etc.) and listen to system events like resuming from sleep and receiving an SMS.
- Hardware access: sensors (light, acceleration, etc.) and short-range wireless interactions (NFC, Bluetooth).
- Performance (speed and stability) is as good as it can be, because there are no second level components between UI and mobile device.

The source code for native application is written in Objective-C (iOS), Java (Android) or C# (Windows Phone).

5.2 Web application

Web applications are, with a little bit simplified, just web sites that is optimized for mobile devices. They are written in common web technologies like HTML5, CSS and JavaScript and runs in a browser of the device. Unlike native applications, web applications cannot be installed to devices, but can be accessed via URL in browser. (Zibula & Majchrzak, 2013; Heitkötter et al., 2013.)

The benefit of web applications is that they function and feels almost the same in different platforms, so the support for multiple platforms is easy to achieve. The disadvantage of web applications is that they cannot access to platform-specific features such as GPS sensor. In addition, it is hard or even impossible to achieve native look and feel in web applications. (Heitkötter et al., 2013.)

Smutny (2012) divides web applications into two categories: Dedicated and generic web applications. Dedicated web application is tailored to specific platform and generic web application is meant to work properly in any mobile devices (Smutny, 2012). In this thesis the term web application is used to have both of these meanings, since web application is not the focus point in the thesis.

5.3 Hybrid application

Hybrid applications are considered to be combination of web and native applications. Like web applications, they are written with the same web-technologies, but unlike web applications hybrid applications can access to platform-specific features. Like in web applications, there is also browser involved, but this time it is embedded inside the applications. These browsers without user controls i.e. Web views interprets the source code during runtime to show the web page. (Heitkötter et al., 2013.) Charland and Leroux (2011) argue that hybrid applications are able to call native code from JavaScript within Web view.

Like is the case with the native applications, users need to install hybrid applications on the device and developers can upload hybrid applications to App Store or Google Play. To put it shortly in hybrid applications, the ability to use platform-specific features is similar to native applications, but the look and feel of the application is similar to web applications. (Heitkötter et al., 2013.)

6 CROSS-PLATFORM DEVELOPMENT

This chapter consists of discussion on cross-platform mobile application development. The definition of platform is given in chapter 6.1. After that the focus is shifted to mobile application development and specially to cross-platform development. Lastly the five major approaches to cross-platform development mentioned in prior research is introduced.

6.1 What is platform?

Software-based platform can be defined as extendible codebase of software-based system, that offers core functionalities and interfaces to access these functionalities. This system can be extended with modules, which connects to the platform through interfaces. In the case of mobile platform, these modules are mobile applications, which are defined in more details in the previous chapter. (Tiwana, Konsynski & Bush, 2010.)

Another view to the definition of platform is that platform is a combination of operating system (OS), language, source development kit (SDK), hardware, user interface (UI) toolkit and device-specific features. (Seffah & Javahery, 2004; Heitkötter et al., 2013; Bishop & Horspool, 2006.) This definition is almost the same as given by Tiwana et al. (2010) except that it considers the hardware also as something that belongs to platform.

In this thesis the term *platform* is used to refer to the software and hardware-based system, that is specified by OS, language, SDK, UI toolkit and core functionalities. The term *mobile platform* is used, when speaking of platform applied to the mobile phones.

In this thesis only Android and iOS OS are taken into consideration, since they had the biggest market share in year 2018 (“IDC Research”, 2019). Because the OS can be considered as the most determining part of the platform, each mobile platform is referred according to it OS in this thesis. For example, Android-based platform is software- and hardware-based system, that is specified by Android as OS, Java as programming language, Android SDK as SDK, Android specific UI and Android specific core functionalities (like back-button).

6.2 What is mobile application development?

Mobile application development is, in general, similar to software engineering of any application. Mobile application developers need to consider of normal software engineering issues like integration with hardware and storage limitations of the device. There are also additional requirements in mobile application development. For example, mobile application can access to the sensors of mobile device or can be developed on top of another mobile application like is the case in hybrid or web applications. Developer need to keep in mind also the limitations of the device, such as limited access to power or limited screen size. One major characteristic for mobile application development is that developer must follow the UI guidelines, which are developed externally and mostly implemented in the SDKs. (Wasserman, 2010.)

Mobile application development can be considered as a third-party development, where the developer develops applications on behalf of the platform owner in order to satisfy the end-users of the platform. (Ghazawneh & Henfridsson, 2013.) According to Boudreau and Lakhani (2009) platform owners has rules and regulations, that third-party developers need to obey in order to get their application approved.

6.3 What is cross-platform development?

6.3.1 Definition

According to Bishop and Horspool (2006) *cross-platform software* means that there is a different version of the same software for different platforms. When applied this definition to mobile applications, it means that same application is available in multiple mobile platforms – for example in iOS and Android-based platforms. This means that developers need to develop application separately to different platforms and because of the significant differences between platforms, very little code can be reused. This increases the cost, time and the resources that are needed to develop mobile application. (Gaouar, Benamar & Bendimerad, 2015.)

Cross-platform development has emerged to solve this problem and it tries to help developers to avoid repetition and to increase productivity when developing cross-platform applications. A common phrase associated to cross-platform development is "write once, run everywhere", which means that developer programs the application once and that same codebase can be executed on multiple platforms. Development approaches can be divided into two groups: approaches that use runtime environment and approaches that generate platform-specific application at compile time. (Heitkötter et al., 2013; Gaouar et al., 2015.) These two groups and their subgroups are discussed more closely in chapter 6.4.

6.3.2 Requirements and constraints

One constraint to cross-platform development is the balance between generality and specificity. There has to be reasonable level of abstraction or otherwise the application cannot be implemented to differing platforms. On the other hand, developers need to be able to take advantage of platform specific functionalities like back-button in Android-based platforms or mute-button in iOS-based platforms. (Heitkötter et al., 2013.) Gaouar et al. (2015) list three constraints for cross-platform development: use of abstractions, problem decomposition and separation of concerns.

Dalmasso, Datta, Bonnet and Nikaein (2013) give eight requirements for frameworks in cross-platform mobile application development: multiple mobile platform support, rich user interface, backend communication, security, support for app-extensions, power consumption, accessing built-in features and open source. Next, these requirements will be explicated in more details.

Obvious requirement for cross-platform framework is that it has to support multiple mobile platforms – at least Android and iOS. Rich user interface means that there needs to be support for sophisticated graphics (2D, 3D), animation and multimedia. Backend communication protocols and data formats need to be supported, because mobile devices elevate an "always-connected" model. Security is one of the biggest weakness in mobile applications that are developed in a cross-platform way and this area needs more efforts both in the research and in the practice. App-extensions can be used for example to add the in-app purchase/billing capability to application. As stated in chapter 6.2 developers need to take into account the power limitations and to optimize the power consumption of application. Accessing built-in features means that application is able to use camera, sensors and other functionalities of platform. Lastly, if framework is open source, more applications will mostly likely be developed with the framework and developer community will also help with bug fixes and further development of framework. (Dalmasso et al., 2013.)

6.4 What are the approaches to cross-platform development?

6.4.1 Runtime environment

There is a browser in all modern-day smartphones and one way to accomplish the “write once, run everywhere” goal is to build mobile applications using *web* *approach*. This approach produces web applications that will be executed in the web browser of mobile device. Examples of tools of this approach are jQuery Mobile and Appsbuilder. (Gaouar et al., 2015.)

Hybrid approach produces hybrid applications. This approach is similar to web approach, because the implemented technologies are same: HTML, Javascript and CSS running in a browser. However, this time the browser is full screen Webview. Examples of tools of this approach are PhoneGap and Vaadin TouchKit. (Gaouar et al., 2015.)

Interpreted approach uses runtime environment that interprets the source code across different platforms. The main advantage of this approach is that it produces native application in the sense that all UI-components are native components, but on the other hand, because of the runtime environment they differentiate from the native applications that are built using cross-compiled or model-driven approach. Runtime environment can slow down the application a little bit and another disadvantage is the limitations to only to those features that the framework provider set. Examples of this approach are Appcelerator Titanium Mobile, React Native and NativeScript. (Gaouar et al., 2015.)

6.4.2 Generator-based

There are two approaches in the generator-based category of cross-platform mobile development. First one is *cross-compiled approach*, where the source code is generated into native code at the compile-phase of an application. For each platform its own version of the application is generated. Because source code is native, the application can access specific features of device. Weakness of this approach is that it is difficult to identify and fix the cross-compilation phase issues. Examples of this approach are Xamarin and QTMobile. (Gaouar et al., 2015.)

Model driven approach is second approach in the generator-based category. In this approach mobile applications are developed by describing models, which are abstraction of objects and observable facts from the real world. Models describe the functionality of an application in modeling language and there are two core categories of languages: Domain Specific Languages (DSLs) and General Purpose Modeling Languages (GPMLs). DSLs are specific to certain domain or context whereas GPMLs can be used in any domain or context. Unified Modeling Language (UML) is the most usual GPML. From these models the source code is automatically generated into the platform dependent native code. (Gaouar et al., 2015.)

7 ARTIFACT: THE UI-FRAMEWORK

In this chapter I will develop the first part of the artifact – a framework that will specify the rules and regulations for shifting from single-UI to MUI-system with the UX and usability requirements in mind. This framework is referred as the UI-framework later on.

I'm going to apply the framework introduced by Garret (2011) in the context of cross-platform service and mobile application to find answers to my research questions, especially to the second sub question. Garrett (2011, pp. 20-21) describes the five planes of user experience, that are meant to help in building a good web UX. These five planes are strategy, scope, structure, skeleton and surface planes (see figure 3). The idea of Garretts model is that you start from strategy plane and build good UX by making decisions at each plane. Each of these decisions' influences on upper planes until you have a ready UI at the surface plane. This decision making should be reiterating by nature, meaning that one can always go back to lower planes, if the situation demands so. The key aspect is not to finish whole plane, before you can proceed to next one, but rather not to finish work on one plane, before work on lower planes is finished.

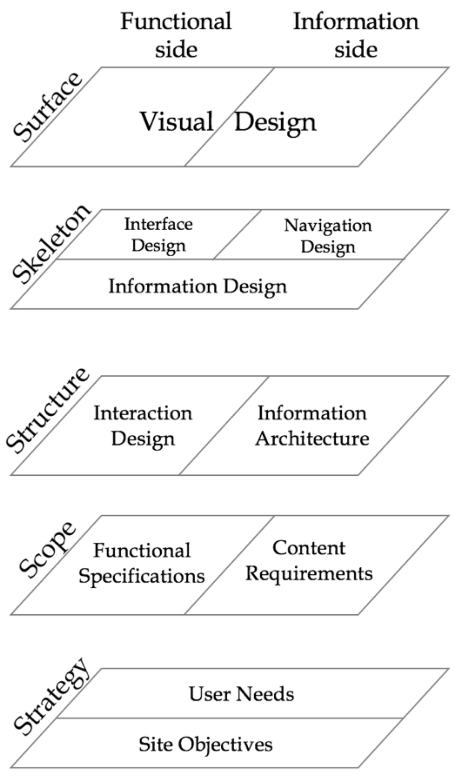


FIGURE 3 The planes of Web UX (Garrett, 2011, p. 33)

The nature of web service can be functional application, information resource or, as normally is the case, hybrid consisting some parts of the both sides. This influences on most of the planes and that is why Garrett (2011, p. 27) splits each layer at the middle. On the *functional side* the focus is put mostly on tasks. Web service is thought to be a tool or set of tools that user uses to do one or more tasks – for example to order shirts from ecommerce-store. On the *information side* the focus point is mostly the information that the online product offers and the meaning of that information for users. Here the most important aspect is to make it as easy as possible for the user to find, absorb and make sense of the information. Garrett (2011, pp. 20-28.). Next, these planes are explained in more detail while considering the duality nature of website at the same time.

7.1 The five planes of UX explained

Strategy plane is the most abstract level of UX, yet it is the most fundamental level. Every other plane build on top of strategy plane. The duality nature of web doesn't affect to this plane, so concerns are the same regardless of whether the online product is more functional application or information resource. There are two main focus point in this plane: user needs and product objectives. Key aspect of good UX is to understand what the user wants from the product. Another aspect is what the product provider wants from the product. Does he/she want to make more money or spread information or what? What are the objectives for the product? Why are we building this product? (Garrett, 2011, pp. 21, 28, 61.)

The main questions at the *scope plane* are following. What are we going to make? Do the requirements implement the strategy set on previous plane? Are the requirements feasible to implement? The strategy determined at previous plane (user needs and product objectives) is translated into specific requirements for the functionalities and the content, i.e. into the scope of the product. These requirements can be for the whole product or just to one single feature or concerning the hardware or the platform (do we need for example the camera or gyroscopic position sensors?). On the functional side of the plane, you create the feature set of the product: the functional requirements and specifications for the product. Content requirements are the scope on the information side of the plane. Here you describe the required content elements, frequency of updates and rough size of your content. It is also important to define the type, format and purpose of the information. For example, the purpose for frequently asked questions (FAQ) is to offer rapid way to commonly accessed information. One possible format for FAQ can be answer-question list and the type of the content could text, images and videos. (Garrett, 2011, pp. 21, 29, 61-74.)

At the *structure plane* one defines the conceptual structure for the webpage. Here one is concerned about the intuitive and logical navigation through the information and content specified at the scope plane. Interaction design is on focus at the functional side of the plane. Interaction design defines how the system reacts to the users' action and how the users' mistakes are handled. At the informational side the focus is on information architecture, which determines the arrangement of content elements in order to enhance human understanding. The structure of websites is mostly visualized using architecture diagrams. (Garrett, 2011, pp. 20, 30, 86-88, 104.)

At the *skeleton plane* the main concern is the placement of buttons, controls, photos and texts. There is a common component for both the functional and the informational side of the plane: *information design*. At this component the emphasis is put on the presentation of information. *Interface design* is the focus at the functional side of the plane. Here one arrange interface elements so that users are able to interact with the functionality of the system. *Navigation design*, on the other hand, is the focus at the informational side of the plane. Here one determines the set of screen elements that enables user to move within the information architecture. (Garrett, 2011, pp. 20, 30, 108-131.)

On the *surface plane* one sees a series of Web pages, that consists of for example texts, images and links. Concern on this plane is the same on both side of the plane: to produce a finished design that pleases the senses and fulfill the goals of other planes. This can be called as sensory design or visual design. At this plane one thinks about matters like contrast, grid, color palettes and typography. (Garrett, 2011, pp. 133-151.)

7.2 The UI-framework

The starting hypothesis for this research is that one has a web service up and running with web UI already implemented. The main concern is the steps to be taken in order to transform the existing web UI to native mobile UI and to produce a cross-platform service consisting of web UI and native mobile UI. In this chapter these steps will be introduced and explained in more detail.

The UI-framework will loosely follow Garrets five planes of UX, but instead of starting from strategy plane I'll start at the surface plane of existing website. The main idea is to find strategy of web service and then build on top of that the strategy of cross-platform service and especially strategy of mobile application. Then one can climb the planes of UX to produce the surface of mobile application. In order to separate web and mobile planes from each other, I have added web or mobile suffix to the plane. For example, scope of web means the scope plane of web service and correspondingly scope of mobile means the scope plane of mobile application (figure 4).

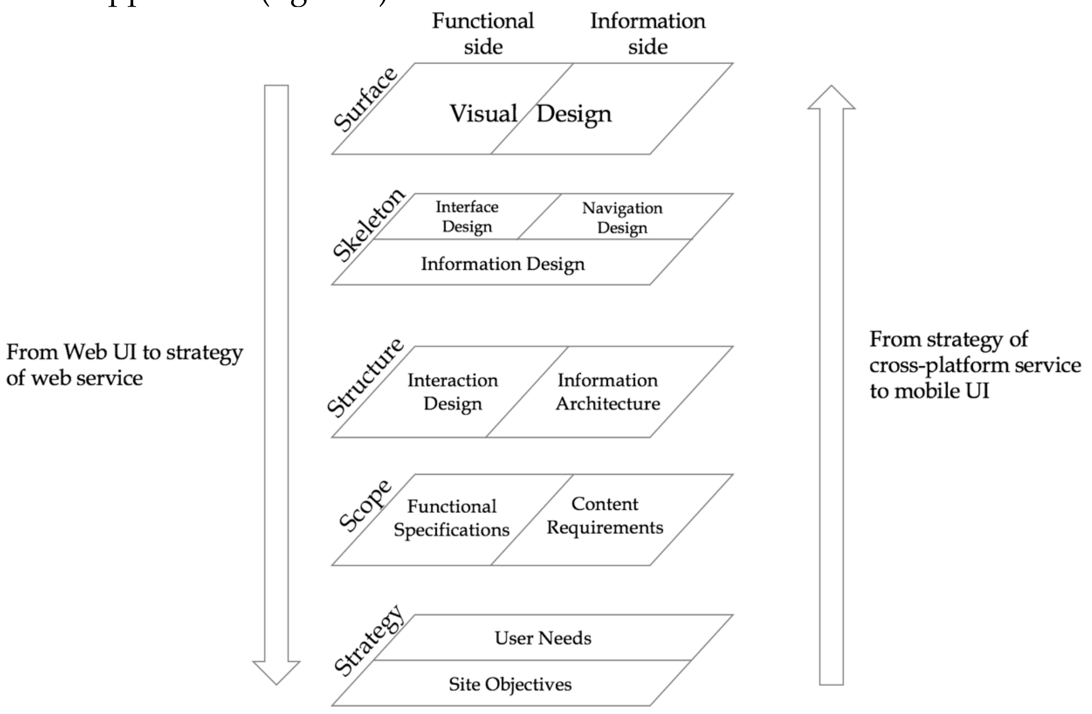


FIGURE 4 Initial UI-framework

One should notice, that the UI-framework merely concerns the user interface. Questions like who will update the data or what kind of architectural approaches should be taken falls mostly out of the context of the UI-framework. It is assumed, that these questions are already answered, because all data is retrieved from existing web service.

7.3 Steps of the UI-framework

The UI-framework consists of 10 steps (table 1). As is the case in Garrets five planes of user experience, each step in UI-framework also depends on the steps taken before it. This is especially true in between web and mobile steps. Mobile steps (6-10) will heavily lean on the documentation created in the corresponding web steps. However, this does not mean that everything should be ready before moving on to the next step. Moving back between steps should be rare, but natural move when the situation demands so.

TABLE 1 The steps of UI-framework

Step	The main goal	Documents
1. Surface of web	To familiarize the web service	Sitemaps and graphical guidelines
2. Skeleton of web	To identify content elements	Wireframe
3. Structure of web	To find conceptual models, vocabulary and information architecture	Visual Vocabulary -diagram
4. Scope of web	To identify the requirements for content and functionalities	List of functionalities and JSON-objects modelling content elements
5. Strategy of web	To identify the reasons for the service to exist from the aspect of users and service providers	Written strategy
6. Strategy of mobile	To define the objectives for the cross-platform service and for the application in special	Written strategy
7. Scope of mobile	To identify the requirements for content and functionalities	List of functionalities with priorities and JSON-objects modelling content elements
8. Structure of mobile	To define the user flow and data structure in mobile	Visual Vocabulary -diagram
9. Skeleton of mobile	To composite the templates from content elements	Wireframes
10. Surface of mobile	To create the visual design of layouts	Layouts

Surface of web is the first step to be taken and the main goal is to get to known with the web service. One should identify the different page layouts, colors, fonts, etc. that are used in the web service. Questions like what kind of content there are, how images are used, how many different colors there are, etc. are asked at this step. First documentation after this step is two lists: list of primary navigation up to second level and list of different templates. Second documentation is simple and general graphical guidelines. Lists gives you basic understanding of the extent of the site and the content. Graphical guidelines answers to questions related to graphic and are used when one creates the surface of mobile.

At the *skeleton of web,* one distinguishes the content elements of each layouts identified in the previous step. When one starts the process of converting web UI to cross-platform UI, one usually has some kind of idea of what part of web service should be transformed to mobile application. Thus, it is justified to select only some of the layouts for more detailed scrutiny, if website is very large and has many layouts. In the first step it was necessary to go through the whole web service, so that one can be sure not to leave any important functionality or content outside of the application. Primary documentation of this step is wireframes of selected layouts, where one has identified the content elements. According to Garrett (2011, p. 128) wireframe describes all the components of a page and their approximate position on the page. In this thesis it is assumed that these individual components will compose multiple autonomous logical entities. These entities are named as *content elements*. In most cases content elements are full width sections of layout, but sometimes it is also possible that content element is part of another content element. Nevertheless, from the wireframe one should easily identify the content elements and also the components that compose the content element itself.

At the *structure of web,* the conceptual model of web service is identified and documented. It's important to find the conceptual model of web functionalities, so that we can preserve the same conceptual model in mobile world. For example, in case of online store user adds and remove products from cart, not edit catalog order form. The same must be true in the context of mobile application, so that the consistent UX prevails and confusions are prevented. The primary documentation of this step is diagrams drawn using Visual Vocabular created by Garret. Visual Vocabulary was created specifically to describe the structure plane of a website. (Garret, 2002.)

At the *scope of web,* the functional specifications and content requirements are identified. With the help of Visual Vocabulary, one should be able to identify needed functional specifications i.e. what the system actually does. For content requirements, the UI-framework will take a slightly difference approach than Garret describes in his framework. Because it is meant to answer research questions of this thesis, framework is more concerned in content elements and what kind of data each content element holds rather than who will update the data and how often. For content requirements one should took the content elements identified in skeleton of web and to write a description of each content elements in JSON-format. JSON is used because it is quite largely in use in web development and it should be quite easy to build some automation tools, that take JSON-object and converts it for example to React component. One content element is one JSON object with three keys: type, props and children. Type is the name of the content element written in camel case and is obligatory for each content element. Props are the properties and their types that content element uses and children are the child content elements.

At the *strategy of web*, one should ask the same questions than in Garrets strategy plane: What user and product owner wants? Why this web service exists? From these answers one starts to draft the *strategy of mobile*. Here one should ask and answer to questions like these: What do the service providers and users want from mobile application? Why one wants to extend the service from web into native mobile? Is the mobile UI going to be equal with the web UI or will the mobile UI serve as a supplementary to web UI? How the data and functionalities differ from web service in mobile application? Especially the last two questions will help to define the configuration of cross-platform service as described in chapter 3.2. Documentation from this step is the written strategies of web and mobile. If strategy of web is the same as strategy of mobile, one documentation should be enough. However, there might be cases, where it is reasonable to write even three strategies: one for web, one for mobile and one for cross-platform service as a whole.

At the *scope of mobile*, one should start with the scope of web and while reflecting the strategy of mobile to add and remove functional specifications and content requirements. Each functionality is given a priority: high, medium or low. High priority means that functionality is essential in order to achieve product objectives and users' needs. Functionalities with medium priority are also important in light of strategy of mobile but are harder to implement. Functionalities with low priority are the least important in light of strategy. One should start with high priority functionalities and iterate one's way to low priority functionalities in the boundaries of available resources and budget. Same kind of supplementing and reducing should be done to content elements specified in scope of web. In light of strategy of mobile, should this content element be part of mobile application? Primary documentations from this step are list of functionalities with priorities and JSON-format representation of content elements included in mobile application.

Supplementing and reducing continues at the *structure of mobile*. The visual vocabulary created in the structure of web will serve as starting point at this step. With the strategy and scope in mind, one should transform visual vocabulary to describe the structure of mobile. After this step, one should have a good understanding for example about how user can navigate in mobile application or how content is structured in mobile application.

At the *skeleton of mobile*, the content elements described in scope of mobile is used to compose the actual layouts. At this step one should take into consideration of the physical limitations of mobile screens. In practice this might mean that one needs to revisit the previous steps – namely scope and/or structure of mobile. For example, it might be hard to implement all the functionalities specified in scope of mobile to narrow screen. One way to resolve this is to add or modify content elements. Another solution might be to give a second though to functional specifications and maybe drop some functionality out of the scope of mobile. In either case, the final documentation of this step is also a wireframe as in skeleton of web.

At the *surface of mobile*, one creates the visual design of layouts based on the wireframes. For consisting user experience, one should follow the general graphical guidelines created in surface of web. One should also take into considerations the design guidelines of each mobile platform in this step. This way the final results will remind the UI of web, but at the same time has the look and feel of mobile platform.

8 ARTIFACT: MOBILE-FRAMEWORK

In this chapter I will design and develop the second part of the artifact – classification of mobile applications and approaches to cross-platform development. This framework is referred as mobile-framework later on in this thesis and it consists of two parts: classification of mobile applications and classification of cross-platform mobile application development.

The structure of this chapter is as follows. First a few issues are raised from current classifications as described in chapter 5 and 6. After that initial framework for classifying mobile applications is offered. Lastly, the differences between approaches to cross-platform development is discussed.

8.1 Would the rigor framework please stand up?

First problem seems to be that in the literature the terms approach and applications are used interchangeably. For example, when Heitkötter et al. (2013) write about hybrid approach, they actually are describing mostly how hybrid applications works. Also, there is not so much emphasis in the development process between the approaches. This raises few questions. Firstly, how does the approach differentiate from application in these classifications? Secondly, are there other major distinctions between approaches than the type of application they produce? Thirdly, how the development process differentiates for example between hybrid and interpreted approaches or between interpreted and cross-compiled approaches? Are there some technologies involved that are specific to some approaches?

Second problem is that there are no studies about the differences between the native application built using the interpreted approach and the native application built using either cross-compiled or model-driven approach. In chapter 5, the different groups for mobile applications were described and there were only three groups. However, in chapter 6.4. when the different approaches to cross-platform development were introduced, it comes clear that there is a need for fourth group of mobile application – namely those applications, that involve runtime environment other than browser or Webview. Delia, Galdamez, Thomas, Corbalan and Pesado (2015) call these applications as interpreted applications.

8.2 Four categories of mobile applications

In the classification for mobile applications I am solely interested in the outcome i.e. the application itself regardless of how the application is made. The essential question here is "What distinguish the applications from each other?". I have developed three questions that defines the nature of mobile application (table 1). First question is how user can access to application. Does user access the application via URL in a browser or can user download the application from App Store or Google Play? One should also notice here, that in order to get access to platform specific store, application needs to fulfill certain requirements, because of the policies of Apple and Google. Second question relates to runtime environment. Does the application run in platform specific runtime or is there more abstract layer involved? Third question relates to the views of application. Are they fully native UI-components?

TABLE 1 Questions for distinguishing mobile applications

Question	Answers
1. How does user get access to application?	Via url or via app store
2. What is the runtime environment?	Webview, Javascript Core, Platform specific
3. What is the nature of components?	Native or web

Answering to these questions will help you to distinguish the nature of mobile application (table 2). If user can access application via url or if all the components are web components, application clearly falls into web application category. On the other hand, if all components are native components and there is no runtime environment involved, application clearly falls into the category of native application. If there is runtime environment involved and it is Webview, application falls into the category of hybrid application. This also means that components are not native components. So far, all this sound like the old and familiar classification described in chapter 5. However, there is a new kid in the town, namely applications that has 100% native UI components, but still includes some kind of runtime environment other than Webview. These applications will fall into a new category called interpreted applications.

TABLE 2 Categories of mobile applications

Question	Web	Hybrid	Interpreted	Native
1. How does user get access to application?	Url	App store	App store	App store
2. What is the runtime environment?	Webview	Webview	Javascript VM	Platform specific runtime
3. What is the nature of components?	Web	Web	Native	Native

The main difference between hybrid, interpreted and native application is the runtime environment. In purely native applications runtime environment is platform specific. In Android it is Android Runtime (ART) and in iOS platforms runtime is called Objective-C Runtime. On the other hand, in hybrid and web applications runtime environment is Webview i.e. the browser. Interpreted applications uses something called JavaScript Virtual Machine.

What is JavaScript Virtual Machine and how differs it from Webview or from platform specific runtimes? JavaScript Virtual Machine or JavaScript Engine is the interpreter that interprets and executes JavaScript code. JavaScript engines are usually associated with browsers, but JavaScript engines can also be used to run mobile applications that has nothing to do with browsers. In fact, that is the case with interpreted applications. For example, NativeScript uses V8 JavaScript engine to run JavaScript code in Android devices and JavaScriptCore to run JavaScript code in iOS devices. React Native completely rely on JavaScriptCore to communicate with native APIs regardless the device. Technical details about the architecture of React Native applications is described in chapter 10.2. ("JavaScript Environment", 2018; Looper, 2015; VanToll 2015.)

Comparing to hybrid applications, that runs on Webview, interpreted applications are much faster and smoother. Thus, I think it's essential to distinguish hybrid applications and interpreted applications from each other. However, as Eskola's (2018) study shows, the performance of React Native applications at least in Android is poorer than in purely native Android application. It should be noted that React Native ships three years old version of JavaScriptCore, which is definitely one reason for slowness of Android React Native applications ("Upgrading JSC", 2018). However, I think because of the interpreting nature of React Native applications, it is natural to be slower than fully native applications. That is why, I think interpreted applications should not be placed in the same group as native applications.

8.3 Approaches to cross-platform development

As I write in chapter 8.1. current literature uses the concepts of approaches to cross-platform mobile development and the nature of mobile applications interchangeably. I think one should clearly separate them from each other. In this chapter, I'll introduce my initial approaches to cross-platform development.

Because the point of interest in this thesis is native application, web and hybrid approaches are excluded from discussion.

The main difference between the interpreted, cross-compiled and model driven approaches are the technologies used to create the application. It is true, that they also might produce applications of different nature as an end product, but I don't think that is always necessity. I see no reason why model driven approach could not produce interpreted application. In some cases, it even might be more justifiable to generate for example React Native code from models instead of native code, because of competence of the developer team. Thus, the nature of application cannot work as distinguishing element between approaches.

Technologies that each approach embrace, is on the other hand very much distinguishing element. In interpreted approach, applications are written with web technologies or at least with technologies that closely reminds of web technologies. For example, styles in React Native are written in JavaScript objects, but the syntax and properties are almost the same as in CSS. On the other hand, cross-compiled approach uses more mainstream programming languages. For example, Xamarin applications are written in C#. Finally, model driven approach relies greatly on modeling languages such as UML.

One difference especially between interpreted and cross-compiled approaches is the development speed. Because of compiling process involved in cross-compiled approach, development speed is naturally slower than in interpreted approach. For example, React Native has introduced hot reloading, which helps the developer to see almost instantly the impact of code change in application (Bigio, 2016).

9 DEMONSTRATION: THE UI-FRAMEWORK

Sub question 3, introduced in chapter 1.2., was how to build native mobile application using WordPress as backend and React Native as chosen cross-platform technology. This sub question is answered in the next two chapters.

Firstly, in this chapter the UI-framework is demonstrated with the website of the Martha Organization (Martat in Finnish). The URL of the website is www.martat.fi. Primary goal of the organization is to promote well-being and quality of life in the homes by providing education about things such as food, nutrition, home gardening and household economics ("In English", 2019).

9.1 Surface of web

The website of Martha Organization is large-scale website both in terms of volume and diversity. There are total of 36 pages in main navigation in two levels and additionally four more page in helper navigation. These pages are listed in Appendix 1 in more detail. Since the website is mainly in Finnish, I have added short description in English after each menu item.

There are total of eight different content types and total of 13 different layouts visible in public. I have listed the layouts in table 2 and for each layout I have also specified what content types the layout serves.

TABLE 2 Different layouts and corresponding content types

Layout	Content type(s)
Frontpage	Page
Landing page template	Page
Single page	Page
Post Archive	News, blogpost
Single post	News, blogpost
Event Archive	Event
Single event	Event
Store	Product
Single product	Product
Recipe Archive	Recipe
Single recipe	Recipe
Contact Archive	Contact
Single contact	Contact
-	Media files

The website uses two fonts: Felt Tip Woman and Brandon Grotesque. The first is mainly used in titles and the latter is used elsewhere. The hex code of primary color is #006bb. Font color is either #2a2a2a, #333, #313131 or white, if there is image in the background. Images serve important role in creating the visual look and feel of the website. They are used as background for many content elements and there is also sitewide background texture in use.

9.2 Skeleton of web

Out of all layouts listed in the previous step, I take a closer look to the layouts of recipe archive and single recipe. One can access the recipe layouts via https://www.martat.fi/reseptit. In appendix 2 you can see the wireframes as whole. Below I have summarized my findings by each layout (table 3).

TABLE 3 Content elements by layout

Archive recipe	Single recipe
Header	Header
Archive hero	Single hero
Filters	Utilities
Search results	Recipe
Recipe item	Ingredients
Footer	Nutrients
	Footer

I identified six content elements that composes the archive recipe layout: header, hero, filters, search results, recipe item and footer. From these header and footer content elements are global content elements throughout the site. Recipe item seems to be also an autonomous content element even though it is part of search results content element.

I divided the single recipe layout into seven content elements: header, hero, utilities, recipe, ingredients, nutrients and footer. From these header and footer are the same as in archive recipe view. Hero content elements looks different in archive and single views, so I distinguished the two with prefix: archive-hero and single-hero content elements.

9.3 Structure of web

Selected views clearly include only one post type mentioned in step 1 – recipe. In this step I modeled the structure of the content and functionality of layouts by using Visual Vocabulary. I included the whole visual vocabulary document as appendix 3.

These diagrams illustrate how user moves from archive view to single recipe view and what kind of taxonomies and metadata is associated with recipe post type.

9.4 Scope of web

Looking at the diagrams from previous step, I was able to write down following functional specifications of website. First of all, user can search recipes by keywords and/or by categories. So there needs to be some kind of search input and checkboxes available. Secondly user can either add or remove recipe from his/her personal list of favorite recipes. If user is not logged in, system will redirect user to log in page. This leads to third functional requirements i.e. log in functionality. User can navigate between archive view and single recipe view, so system needs to have two view. User can also view comments and leave a comment, if commenting is made available in the admin of WordPress. I have summarized functional specification in table 4.

TABLE 4 Functional specification of web

Functionality of system	NB
Log in and log out	
Search by keyword	
Search by categories	
Add recipe to favorites	Directs to log in page, if user is not logged in.
Remove recipe to favorites	
Two views: archive and single recipe	
List comments of single recipe	
Leave a comment about single recipe	

For content requirements I wrote documentation in JSON-format (appendix 4). Each content element was analyzed in order to find properties of content element and possible child components. For each property I defined what kind of data should it hold – string, number, function or something else.

9.5 Strategy of web

This and the next chapter serve as examples of what they could look like, because of the limits of this thesis. In real life thigs like User research and personas should be done in this step to find out who the users of the website are and what they really want. Also, cautious study of business goals and brand identity should be done together with Martha organization to find out product objectives.

For the sake of example, I answer to two questions in this step: What users want to achieve when they use recipe section in website? What Martha organization wants to achieve with recipe section?

Users wants to find fast and easily recipes. They want to sort recipes by season or by popularity. Users wants to see nutrients and ingredients of single recipe and the different phases of recipe.

The website as whole was renovated to serve people of Martha organization and citizens. It is important that the diversity of Martha organization and its services are visible and easy to find from website. (Heikkilä, 2017.) For Martha organization recipe section can be seen as one way to serve the people both inside and outside of their organization. Recipes are great ways to introduce more people to Martha organization and the services they provide. In the end, it can be seen as a way to achieve one aspect of their primary goal i.e. promote well-being and quality of life in the homes.

9.6 Strategy of mobile

User needs can be seen quite similar whether the platform is mobile or desktop. However, it should be noticed here that user expect service work similarly regardless of the platform in hand. For example, user expect to see the same information about each recipe in mobile as the see in desktop. Because of this mobile UI is going to contain the same or almost the same functionalities and content than web UI.

One point in the strategy of Martha organization for year 2019 is to make Martha activities visible and encourage people to be part of local Marthaclubs (Heikkilä, 2018). Expanding their single-platform service to cross-platform service can be seen to answer this aspect of strategy. Mobile application opens more possibilities for people to be part of local Marthaclubs and thus Martha organization can draw attention more effectively and widely.

9.7 Scope of mobile

In this step I looked at the functional requirements of web and decided whether or not they should fit in the mobile application. Additionally, I will give a priority for each specification in light of the strategy plane. Summary of functional specifications with their priorities can be seen in table 5.

Considering the limitations of this thesis, I decided to continue only with the specifications of high priority: search by keyword, search by categories and two views. These functionalities are essentials in order to achieve users' needs and product objectives. Functionalities with medium priority are also important, but harder to implement, so they will form the basis of next iteration of development.

TABLE 5 Functional specification of mobile

Functionality of system	Priority
Log in and log out	Medium
Search by keyword	High
Search by categories	High
Add recipe to favorites	Medium
Remove recipe to favorites	Medium
Two views: archive and single recipe	High
List comments of single recipe	Low
Leave a comment about single recipe	Low

I excluded Header and Footer content elements from the scope of mobile application, because I am building an application dedicated to recipe section of website. I also needed to revisit this step, when I was creating wireframes of mobile views. I realized that I need to divide Filters content element into Filters row and Filters modal -content elements, because of screen size limitations. Table 6 summarize selected and added content elements. They are described in more detail in appendix 5.

TABLE 6 Content elements by layout in mobile

Archive recipe	Single recipe
Archive hero	Single hero
Filters row	Recipe
Filters modal	Ingredients
Taxonomy column	Nutrients
Search results	
Recipe item	

9.8 Structure of mobile

The metadata and taxonomies associated with recipe remains the same in structure of mobile than in structure of web. I also reserve the same conceptual model and vocabulary than in web, so that the user experience is possible whether user is using web or mobile service.

Flow chart in structure of web needed some adjustment, because of the decisions I made in the last step. Final user flow chart is simpler than equivalent in web (figure 5). See appendix 6 for the whole visual vocabulary of mobile.

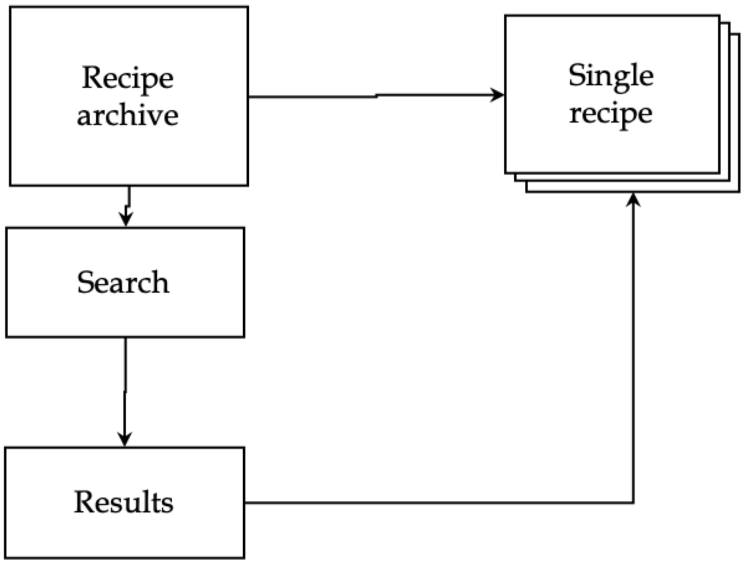


FIGURE 5 Flow of mobile application

9.9 Skeleton of mobile

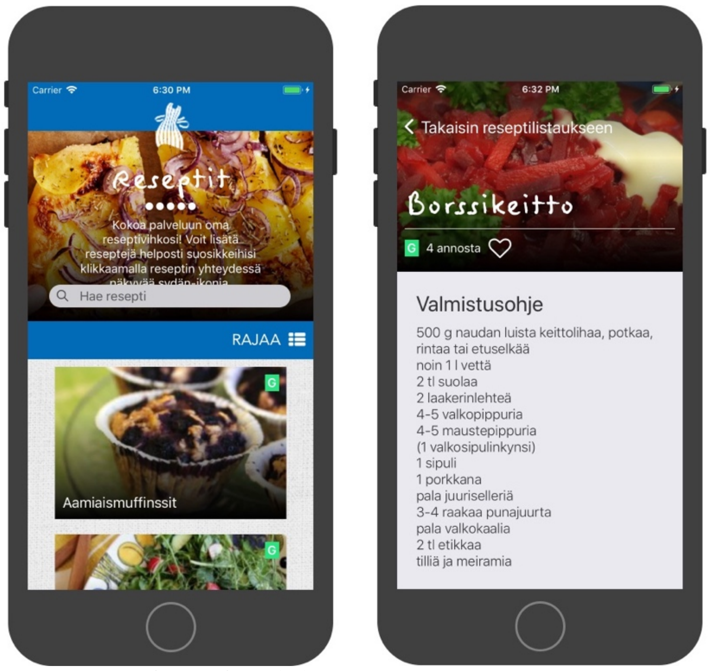
In this step I took the mobile content elements decided in scope of mobile and roughly composed archive and single recipe views from those content elements (appendix 7). Because of physical limitations of mobile screen, I decided to revisit scope of mobile plane and rethink content elements. I added Filter button, which user can press to open modal with taxonomy filters.

In addition, I excluded the pagination component from search results content element in favor of infinite scroll. This functionality was not implemented in the final product though, because of limitations of this thesis. I also excluded sorting column content element at this step to keep things simple enough.

9.10 Surface of mobile

I added blue header with the logo of Martha Organization to archive recipe view to keep the brand look and feel. Apple Human Interface Guidelines (Apple, 2018) suggest that one should not display logo throughout the app, so archive recipe view was the only place where I included the logo.

To create consistent user experience, I used the same colors that were defined in surface of web plane. I used the same font family in layout titles as is used in website. Background images for hero components is also the same as in website. Final layouts from actual application is seen in figure 6.


Carrier
6:30 PM
Reseptit
Kokoa palveluun oma reseptivihkosi! Voit lisätä reseptejä helposti suosikkeihisi klikkaamalla reseptin yhteydessä näkyvää sydän-ikonia
Hae resepti
RAJAA
G
Aamiaismuffinssit
G
Carrier
6:32 PM
Takaisin reseptilistaukseen
Borssikeitto
G
4 annosta

Valmistusohje

500 g naudan luista keittolihaa, potkaa, rintaa tai etuselkää
noin 1 l vettä
2 tl suolaa
2 laakerinlehteä
4-5 valkopippuria
4-5 maustepippuria
(1 valkosipulinkynsi)
1 sipuli
1 porkkana
pala juuriselleriä
3-4 raakaa punajuurta
pala valkokaalia
2 tl etikkaa
tilliä ja meiramia

FIGURE 6 Surface of mobile

10 DEMONSTRATION: MOBILE-FRAMEWORK

The technical details about the demonstration application is reported in this chapter. First subchapter contains the information about WordPress as backend in general and specifically relating to website of Martha organization. Second subchapter takes closer look on the architecture of React Native application and specifically the demonstration application.

10.1 WordPress as backend for the application

10.1.1 In general

WordPress is open source content management system, that is built on PHP and MySQL (WordPress, 2018a). WordPress websites can be divided into three parts: WordPress Core, plugins and themes. WordPress Core is the official WordPress itself. Plugins can be used to modify the functionality of WordPress Core and theme is the public UI of the website i.e. how the website looks like. The line between a theme and a plugin can be sometimes shady since WordPress doesn't force to distinguish functionalities with look and feel. That is why there can be some functionalities in the theme or plugins can alter how site looks like.

Developers can flexibly modify WordPress Core via hooks, which are ways for programmers to hook into the execution of WordPress at a specific time in execution process. This way programmers can either execute own custom code or to alter the values of variables that WordPress will use later on the execution.

Content architecture of WordPress sites can be quite simple even though plugins can make it more complex. All the content (for example pages and images) is saved as posts. Post has always a post type, title and post content and all posts can be found in wp_post table in the database. Posts can have metadata attached to it, and metadata is saved to wp_postmeta table as one metadata per row. Each metadata has post id for connecting the metadata with the post. Posts can also have taxonomies and programmer can create his/her own custom taxonomies also. Taxonomies and terms are saved in four different tables. Table 7 will summarize the tables of WordPress database, which are important in the scope of this thesis. (WordPress, 2018b.)

TABLE 7 The main database tables of WordPress

Table	Description
wp_post	All content (pages, images, etc.)
wp_postmeta	Metadata of single post
wp_term	All terms of categories, tags and custom taxonomies
wp_termmeta	Metadata of single term
wp_term_taxonomy	Attach each term in wp_term to single taxonomy
wp_term_relationships	Attach each term in wp_term to posts in wp_post

By default, all the content in the edit view of post is saved in single column in wp_post -table. With plugins, like Advanced Custom Field, it is possible to easily create rich UI for users to save the content data in wp_postmeta. For example, in figure 7, the values of title and background color of hero is saved individually to database as two separate metadata.

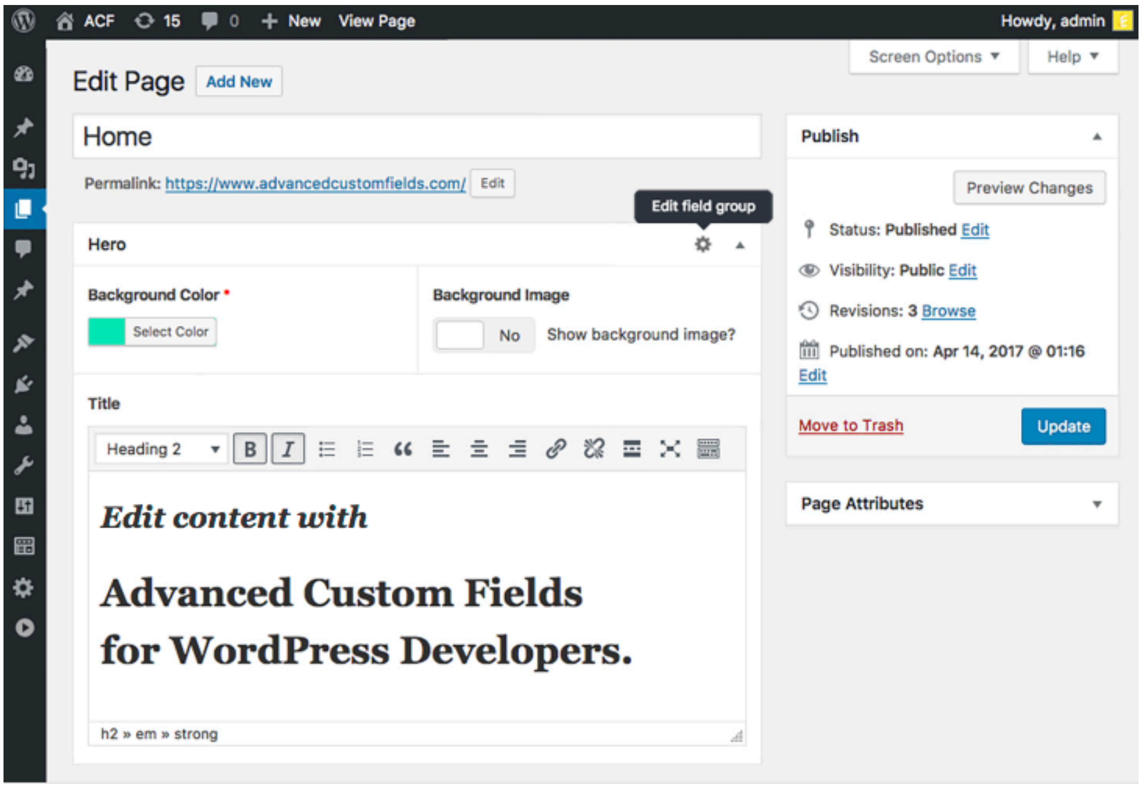


FIGURE 7 Example view of post edit view with Advanced Custom Fields (Advanced Custom Fields, 2018).

Starting from version 4.7. WordPress Core has come with the built-in Rest API. This API provides endpoints for developers to easily interact with sites even outside of the context of WordPress. This communication happens via JSON objects. WordPress has a few built-in endpoints for fetching for example pages or some individual post. Developers can create own custom endpoints, if the built-in endpoints are not enough. (WordPress, 2018c.)

10.1.2 Martat website as backend

The content of Martat website is saved to WordPress in a custom post type called recipe. Taxonomies of this post type are named in appendix 3, which also lists metadata of single recipe. However, appendix 3 was made by observing the front-end of the website. In database level only ingredients, nutrients, portion size and attached image are actually metadata of recipe. Has gluten -value is generated from whether or not recipe has certain term in a custom taxonomy. Is favorite -value is generated from metadata relating to current logged in user, but that is out of the scope of this thesis.

Filtering the recipes in web UI is done with FacetWP -plugin. Filtered recipes are fetched from WordPress database via AJAX-call. FacetWP also includes own Rest API Endpoint that can be used to access the raw data of FacetWP. (FacetWP, 2018.)

In my demonstration app the communication between the app and WordPress happens via two custom Rest API endpoint (figure 8). First endpoint is for fetching all recipes or filtered recipes. Second endpoint is for fetching the necessary data of a single recipe.

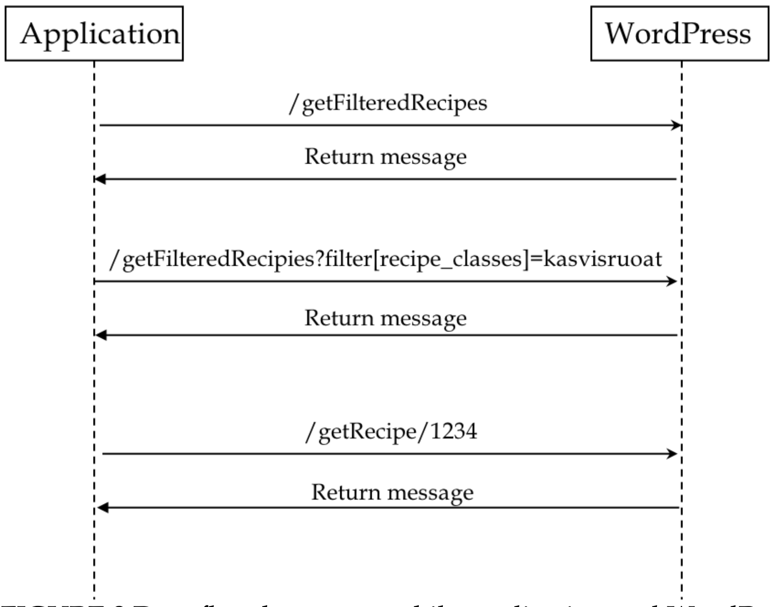


FIGURE 8 Data flow between mobile application and WordPress.

The first endpoint (appendix 8a) is a sort of wrapper for API endpoint of FacetWP -plugin, since I wanted to use the same filtering engine in mobile that is used in the web. This way the user experience would be congruent regardless of the platform. Unfortunately, FacetWP only returns the post ids of results and my application needs titles and URL of recipe images besides the post ids. Because of this I wrote own Rest API endpoint that returns the data, that the application needed, in a format, that the application is ready to consume.

What is happening under the hood, is that at application start or when user filters recipes, application does single HTTP GET -request to /wp-json/app/v1/getFilterdRecipes -endpoint with filtering terms as optional query parameters. After calling the Rest API endpoint of FacetWP, the missing data is fetched by using functions of WordPress Core like get_the_title() or has_term().

The final response message is JSON object containing three items: results, facets and pager. Results is an array of object each containing post id, title, url of recipe image and gluten information. Facets is object containing every selected filtering option and is empty, if user has not done any selection in the app. Pager is object containing pagination information. Because of the scope of this thesis, I didn't built pagination to my application.

Appendix 8b demonstrates the example answer from getFilterdRecipes endpoint in a situation where user wants to view all vegetarian cuisines that doesn't include milk. For simplicity sakes, I have included only a few recipe results.

The second custom endpoint is a lot simpler. It is accessible via "/wp-json/app/v1/getRecipe/ID" -path, where ID is the post id of recipe. This endpoint returns the necessary data from single recipe like ingredients or portion size. The code and the return messages of this endpoint are described in detail in Appendix 9.

10.2 React Native

10.2.1 Architecture and development of React Native application

As I stated in chapter 8.2 React Native applications are not native in a sense that they run inside of JavaScript Virtual Machine, namely JavaScriptCore ("JavaScript Environment", 2018). On the other hand, React Native applications are native applications in a sense that all views can be purely native. How is this accomplished under the hood?

React Native applications take benefit of at least three separate threads: Main thread, JavaScript thread and Shadow thread (figure 9). Main thread is responsible for rendering the UI and listening user interaction like touch events and scrolling. JavaScript thread is the place where all the business logic of application happens i.e. where all the JavaScript code runs. Schrock (2016) calls shadow thread as layout thread, but regardless of the name, its responsibility is to calculate the layout and the position of each component in the screen. Besides these threads there can be multiple native module threads for custom native modules, but these are out of the scope of this thesis. (Parashuram, 2018; Schrock, 2016; Zagallo, 2016; Facebook Inc., 2018a).

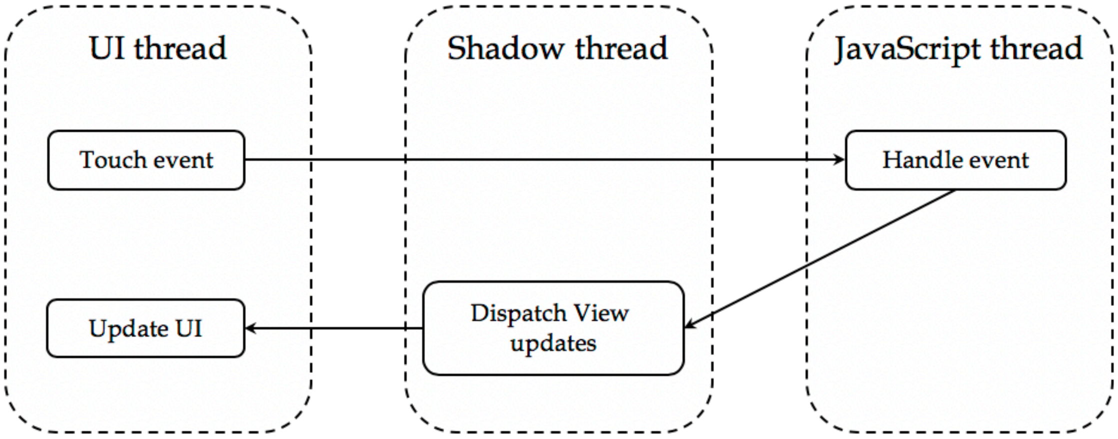
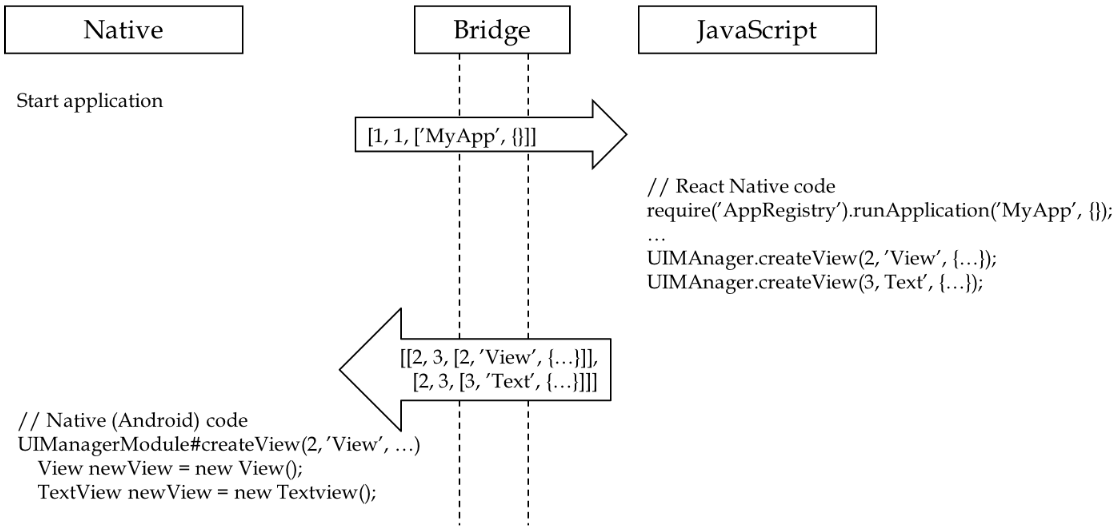


FIGURE 9 Main threads in React Native applications (Konicek, 2015)

The communication between native and JavaScript happens via something called bridges, which are asynchronous, batched and serialized (figure 10). Asyncronous means that neither JavaScript thread nor the main thread wait for the answer, i.e. the thread is not never blocked. This means for example, that user can continue scrolling the screen, after he/she has pressed a button and JavaScript thread is still doing work. Batched and serialized bridge means that all messages between native and JavaScript is bind together and serialized. (Konicek, 2015; Kotliarskyi, 2015).



```
// React Native code
require('AppRegistry').runApplication('MyApp', {});
...
UIMAnager.createView(2, 'View', {...});
UIMAnager.createView(3, Text', {...});
```

```
// Native (Android) code
UIManagerModule#createView(2, 'View', ...)
  View newView = new View();
  TextView newView = new Textview();
```

FIGURE 10 Communication between native and JavaScript in React Native applications (Konicek, 2015)

React Native applications are composed of multiple components, which can be written using syntax extension to JavaScript invited by Facebook. This syntax extension is called JSX and it is transpiled to standard JavaScript during the bundling process. ("Draft: JSX Specification", 2014; "Introducing JSX", 2018). There are multiple built-in JSX-components, which will render to native components in the end. React Native uses View Managers to map JSX-component straight to platform specific equivalent. For example, one of the most fundamental components in any mobile application is view-component and table 8 summarizes how it will be mapped in each platform. (Facebook Inc., 2018b).

TABLE 8 JSX-component mapping to platform equivalent (Facebook Inc., 2018b)

JSX	iOS	Android
<View />	UIView	android.view

Most of the React Native code can be shared between platforms, but there are ways to write platform specific components. One can even write own custom native component in Objective-C or Java, if the built-in components are not enough. (Facebook Inc., 2018c; Facebook Inc., 2018d).

React Native component can be styled with special JavaScript object called StyleSheet. This is an abstraction bundled with React Native package and enables developers to style components with CSS-like syntax. For example, font size can be changed in CSS with font-size and in React Native with fontSize. (Facebook Inc., 2018e).

One of the biggest advantages in React Native development compared to native development is how fast the code change is reflected in the application. This happens because of features like live reload and hot reload. Hot reload is especially useful, since it keeps the application state through reloads. This means that all the user interactions like clicking a button or changing a view are still there even though the code has changed. (Facebook Inc., 2018f).

The developing with React Native is very similar to developing with React. Concepts like state, component and props works the same way in both frameworks. State is the place to store data, that is going to change. State can live in a level of components, but in a bigger application it is recommend that state lives in the application level. Props are properties of components that remains the same throughout the lifecycle of a component. Props are also set in a parent component. (Facebook Inc., 2018g). Many JavaScript libraries that can be used in React works also in React Native.

10.2.2 Martat application

I used third party JavaScript library called Ignite CLI as starting point for my application. Ignite offers an easy and fast way to start writing the application code with the current best practices and solutions built-in (Infinite Red, 2018).

Ignite boilerplate includes multiple JavaScript libraries for helping and accelerating the development process. It is not in the scope of this thesis to go through all these libraries, but I will introduce the main ones, that can be divided into two groups.

In the first group are the JavaScript libraries that helps to create the UI of application (table 9). I used React Native Linear Gradient library to create dark linear gradient on top of images so that text is more visible regardless of how light the image is (see figure 6). React Native Modal library was used to create Modal view which contains all the filtering options. React Native Vector Icons library was used to easily include high quality icons, like heart icon. Native Base library was used to include some ready cross-platform components like search bar of archive view.

TABLE 9 Main JavaScript third party libraries: UI-components

Folder	Description
React Native Linear Gradient	Linear gradient on top of images
React Native Modal	Modal for filtering options
React Native Vector Icons	High quality icons
Native Base	Ready to use cross-platform components

In the second group are the JavaScript libraries that helps to manage the data and the state of application (table 10). Redux is obviously the main JavaScript library in this group. It is designed to manage the state of complex applications. React Navigation is also essential JavaScript library, since it is used to change the screen i.e. moving between different views of application. All other libraries in second group are helper libraries for these two. Since the navigation data (i.e. what screen user is viewing) is also stored in the state of application, React Navigation Redux Helpers is needed, because otherwise navigation will not work with Redux. I used Redux Saga to help with asynchronous things like fetching data from WordPress Rest API. Finally, Redux Persist library helps to persist the current state of application between instance of use. For example, if user has opened the details of single recipe and leaves the application, Redux Persist makes it possible that the same view is opened when application is next time started.

TABLE 10 Main JavaScript third party libraries: managing the data and the state

Folder	Description
Redux	Most of the components
Redux Saga	Full screen components
React Navigation	Image files
Redux Persist	Components relating preserving application current navigation and data
React Navigation Redux Helpers	Integrates React Navigation with Redux

Ignite boilerplate also includes thoughtful structure of files and folders (table 11). Each component is written in a single file and put inside of Components-folder.

Containers-folder includes the main views of application like view for all recipes and view for single recipe. The files relating specially to navigating between these two views are located in Navigations-folder. All the images used in application, is put inside of Images-folder. Redux-folder includes all files that relates to handling the Redux Store. Files that fetch the recipes from WordPress Rest API is put inside of Sagas-folder.

TABLE 11 Simplified version of folder structure of Martat-application

Folder	Description
Component	Most of the components
Containers	Full screen components
Images	Image files
Navigation	Components relating to navigation
Redux	Configuration files for Redux
Sagas	Configuration files for Sagas

11 EVALUATION

The sub questions 1 and 2, introduced in chapter 1.2, are answered in this chapter in the form of the fifth activity (evaluation of artifact) of design science process model.

Evaluation is divided into two section, which answers to different sub questions. First the sub question 1 is answered in the chapter 11.1 by evaluating the UI-framework. Sub question 2 is answered in chapter 11.2, where the evaluation of the mobile application framework takes place.

11.1 Evaluation of the UI-framework

The UI-framework is evaluated in this chapter. The framework introduced by Wäljas et al. (2010) and explained in chapter 4.2. is used as evaluation criteria. The UI-framework is evaluated by looking the three elements of cross-platform service UX. The point of interest here is how well the UI-framework directs to build a system where these elements are fulfilled. The following questions are asked:

1. Does the UI-framework lead to a service that is fit for cross-platform tasks?
2. Does the UI-framework lead to a service where interactions and content flows?
3. Does the UI-framework lead to a service where users perceive coherency?

These questions are answered next.

11.1.1 Fit for cross-platform tasks?

The first element of good cross-platform service UX is that system is fit for cross-platform tasks. In ideal situation individual platforms serve the functionalities and data which user expects to get from that platform.

In the UI-framework strategy of mobile and scope of mobile -steps are important in order to produce a service that is fit for cross-platform tasks. Decisions that are made in these steps will affect the general structure of the cross-platform service. Especially the strategy of mobile is essential since scope of mobile will lean on the decisions made in this step. Functionalities will make their way to final application based on the priorities that are given according to strategy of mobile. That been said, I don't think the strategy of mobile will be enough, since the questions asked in this step will not offer clear answers to questions related to structure of whole cross-platform service. For examples questions like how users allocate roles between system platforms needs to be addressed more explicitly. If user allocate roles more situation-based that means that almost every functionality identified in scope of web needs to be included in scope of mobile. Or at least we should be really thoughtful of excluding functionalities from the mobile application.

For example, the demonstration application will break too much the principle of distribution of functionality. Comments are not very wide used in the website, so it makes now harm to leave them outside of the scope of the application. On the other hand, the functionalities with medium priority are quite essential. For example, if user cannot see his/her favorite recipes, he/she will automatically think that the user experience in cross-platform service is bad. This is why functionalities with medium priority should be included in mobile application.

Another critical issue that should be mentioned here, is that the functionality of mobile application is quite dependent on the availability of website. If website is down or mobile application cannot access it for some other reason, application is quite useless. All data is fetched from the website and very little is saved to the own memory of the application. This is an issue that the UI-framework doesn't actually address at all. Libraries like Redux Persist can be used to help in these kinds of situations, but the responsibility relies solely on the shoulders of developers. It would be beneficial if these kinds of questions are asked and answered in the UI-framework.

11.1.2 Does interactions and content flows?

The second element of good cross-platform service UX is that interaction with and across UIs are considered as fluent and connected. In ideal system user can start one task on one UI and continue it with another UI without needing explicitly think about the UI in hand.

In the demonstration application this is not so significant since there are only a few functionalities included in the application. With the functionalities of medium priorities this will come more significant. For example, system needs to handle the use case where user adds one recipe as favorite with web UI, goes to kitchen to cook and wants to see the information of that favorite recipe on the mobile UI. How the information of favorite recipes is then fetched from WordPress? When application starts? When user clicks some button? The UI-framework leaves us empty with these kinds of questions.

One should notice, that the UI-framework will direct developer to recognize different tasks in structure of web and scope of web -steps. However, the UI-framework does not explicitly address how these tasks are handled in the context of cross-platform service.

11.1.3 Do users perceive coherency?

The third element of good cross-platform service UX is that users see consistency and coherency in the whole cross-platform service. For example, same icons are used to indicate the same data or functionality.

The first step in the UI-framework (surface of web) will embrace issues relating to perceptual consistency. The design guidelines are created by observing the website and identifying font families and main colors. This will be crucial when one forms the surface of mobile. It is easy to pick up the colors and font families from design guidelines and thus outcome the same look and feel than in web UI.

The UI-framework does not take into account very explicitly the semantic consistency. From the wireframes and content elements in skeleton of web one can find the icons and terminology to use in mobile application. One of the main goals in structure of web is to find vocabulary, but Visual Vocabulary -diagram seems not to be the best way to represent this vocabulary.

Syntactic consistency is also something where the UI-framework is not so clear. For example, in the demonstration application the navigation between archive and single recipe views happens the same way both in web UI and mobile UI i.e. clicking the recipe image. One can ask was this because of the explicit decision that I made in some step or was it more because of coincidence? Framework does not actually enforce to keep the same navigation logic that is used in web. On the other hand, listing content elements and their attributes will help to replicate the same navigation patterns. For example, in demonstration I have identified RecipeItem content element that has onClick property, which will do the navigation (appendix 4). The same RecipeItem content element is also included in scope of mobile and has onPress property, which will do the navigation (appendix 5).

11.1.4 Conclusions

After critical evaluation, it is clear that the UI-framework needs some refactoring and clarifications.

Biggest change would be to add another step between strategy of web and strategy of mobile, namely strategy of cross-platform service. This would emphasize more that we are actually building a cross-platform service and not just distinct Uis for web and mobile. Three questions should be asked in this step. First question relates to how user allocate roles between Uis. Does users do task-based or situation-based role allocation? Understanding this will have a huge impact on later steps. For example, it will affect to which functionalities needs to be included in mobile application. Second question would relate to functional modularity. How should the cross-platform service react when website is unreachable? Should, for example, some data be saved to mobile application? Third question relates to fluency of content and task migration. Should system enable moving between different platforms and how will it do it? When answering to these questions in strategy of cross-platform service, it will be easier to include or to add necessary functionalities in scope of mobile.

I also noted that Visual Vocabulary does not enable to write the needed documentation for structure of web -step. One should add some kind of written vocabulary as document for structure of web. This way one can make sure, that the same terminology is used both in web UI and mobile UI.

Reflection part should also be added to surface of mobile -step. The following questions should be asked: Is the design of mobile in line with strategy of cross-platform service and strategy of mobile? Is the design of mobile coherent with surface of web in terms of icons, style and functionality? It would help if in surface of web -step one would have described the visual design of website in more elaborately. For example, using some sort of design language system.

One thing that the UI-framework does not take into account is the user experience in the platform itself. Is the application in line with the design guidelines defined by the platform owner? Failing to follow these guidelines will results in poor UX even though the cross-platform service UX is considered good. For example, if in Android devices the back-button works differently in the application than in rest of the platform, UX is perceived to be poor. In worst case scenario application never make it to the App Store, because application fails to follow design guidelines.

11.2 Evaluation of mobile-framework

The mobile-framework is evaluated in this chapter. First the classification of mobile applications is evaluated. Secondly the classification of cross-platform mobile application development is evaluated in light of Nunkessers (2018) recent study on mobile application development.

11.2.1 Classification of mobile applications

Based on the demonstration application, the questions that was introduced in chapter 8.2 for classifying mobile applications seems to work quite well. As stated in chapter 10.2.1 the code written in React Native is interpreted in JavaScript Virtual Machine during runtime, but every UI-element is rendered as native UI element.

In React Native applications there are certain performance issues compared to native applications. These issues emerge especially with long lists and animations and are caused mainly because of asynchronous bridge. The coming refactoring of React Native architecture might fix these issues, but it would be a good idea to somehow recognize the differences in performance in mobile-framework. (Parashuram, 2018).

Frameworks like Fuse will question the reliability of classifications in mobile-framework for both applications and cross-platform development approaches. Unlike React Native, Fuse uses JavaScript only for application logic and UI components are rendered from a purely native code. Fuse applications would belong to native category, if you only look at the UI components. However, there is still JavaScript Virtual Machine and runtime interpretation involved, which would make it to fall into interpreted category of mobile-framework. Also, the UX markup, the language used for creating UI components when developing with Fuse, is not web technology and does not even look like a web technology as is the case with JSX in React Native. (Pedersen, 2016).

In Mobile-framework it is defined that Native applications have Platform specific runtime. This classification would need adjustments especially concerning native applications in Android. According Nunkesser (2018) strict definition of native Android application is, that is it written in C/C++ with Android Native Development Kit (NDK). Also, as Ohrt and Turau (2012) address, XCode for iOS is compiled to binary code while Android SDK creates Dalvik bytecode from Java code. This would lead to distich meaning for native application depending on which platform is in question.

11.2.2 Classification of cross-platform development approaches

In the second part of mobile-framework there were two main concepts to concern: technologies used in development and the speed of development.

Nunkesser (2018) follows this same line of thought that technology used in development process is the most distinguishing factor between approaches. However, Nunkesser offers completely different taxonomy for classification. He divides development approaches into three main categories: endemic, pandemic and ecdemic. These three Greek words could be translated in this context as "within platform", "all platforms" and "out platform". Next these three categories are explained in detail and reflected to the categories of mobile-framework.

Endemic applications are created with platform specific SDKs. In mobile-framework, this kind of development approach is not taken into consideration, since this would mean two different codebases: Java for Android and Objective-C for iOS. *Pandemic* applications are ones created with technologies, that are supported by every major mobile platform. Pandemic applications can be divided into four subcategories: web app, hybrid web app, hybrid bridged app and system language app. *Web app* and *hybrid web app* would correspond with web and hybrid approaches in more traditional classification are thus excluded from mobile-framework, since they do not produce native applications. Applications written in React Native are *hybrid bridged applications* since they use a bridge from JavaScript to endemic (platform specific) language. Last subcategory of pandemic applications is *system language app*, which are written in C/C++. *Ecdemic* applications would be something that is written in a language that is not natively supported by the platform. As seen in table 12, mobile-framework does not take into account the case with System language app. (Nunkesser, 2018.)

TABLE 12 How Nunkessers taxonomy correspond with the taxonomy of mobile-framework?

Nunkesser	**Mobile-framework**
Hybrid Bridged App	Interpreted approach
System Language App	Doesn't take into account
Foreign language app	Cross-compiled and model drive approaches

More research needs to be done concerning the approaches of cross-platform mobile development. One interesting field would be to create more precise meter for distinguishing approaches in terms of developer experience and technical possibilities. For example, things like code sharing, hardware accessibility, ability to extend with native code, debugging experience, compiling experience, code completion and maintenance should be considered here. Partly this work has been done by Nunkersser (2018) and Ohrt and Turau (2012), but complete meter is yet to be done.

12 DISCUSSION

This chapter answers to the research question of this thesis, which was "Under which conditions can you build native mobile applications using cross-platform technologies and Web CMS as a backend and how to do it using React Native and WordPress".

12.1 Limitations of backend

The conditions and limitations that arises from WordPress are specified in this chapter. Observations are mostly specific to WordPress but can be generalized to similar content management systems.

First requirement is that data needs to be saved in structural format in database. In WordPress this means metadata instead of post content. This way data can be easily accessed via Rest API. Given the popularity of different visual page builder plugins in WordPress ecosystem, this will be problematic in many WordPress sites. This is because most page builder plugins compromise the separation of data and presentation. They save the content to one post_content column in database and thus data will be muddled with visual presentation markup.

Second requirement concerns the structure of content in backend. Whether or not content is structured logically has a huge impact on how easy it is to use the content outside of the context of CMS. For example, in WordPress each logical unit of content should be represented in corresponding post type or applicable taxonomy or metadata. This will also help in surface of web -step, since the extend and the structure of website will be more easily recognized when content is served in structured form.

Third requirement is quite obvious. There must be some kind of way to use the data outside of CMS. Mostly this means possibilities to write your own custom endpoints to Rest API. This is also the case with WordPress. Even though the Rest API is part of WordPress Core, the built-in endpoints are not enough for serious and complex applications. Handling the HTTP requests and responses in mobile application are much easier if you get only the data you need from the endpoints.

12.2 Limitations of mobile application

In this chapter the conditions that arises from React Native are specified. Since the demonstration application is small both in size and features, this chapter mostly relies on the experiences of Airbnb and Udacity. Both companies used React Native seriously for two years, but then decided to remove it from their codebase. (Peal, 2018a, Ebel, 2018). Even though arisen issues are React Native specific, they can, at least in some extend, be generalized to other cross-platform mobile development framework like NativeScript.

First limitations might sound too obvious, but it's worth to mention. Cross-platform development solutions like React Native has more benefit, when developing a real cross-platform mobile application. Meaning that application should have only few platform specific features and desired behavior should be somewhat the same in all platforms. (Ebel, 2018.)

Second limitations relate to how easy it is to ensure native feel in each platform. According to Majchrzak, Biørn-Hansen and Grønli (2017) React Native components are mostly unstyled and leaves the responsiblity to developer to deliver consistent user experience within mobile platform. As Ebel (2018) notices it is not hard to deliver native feel, but that is something that developer needs to be aware. Another issue would be that even though there are thousands of NPM packages available and meant to ease developing process, many of them are missing the cross-platform support (Peal, 2018a).

Third limitations is more Android specific issue than actually React Native specific issue. Ebel (2018) notices that support for wide variety of Android device can be overwhelming. Animations that run smoothly on one Android device, can be rough in another Android device.

React Native is still quite immature and is changing rapidly. This results to issues like inaccuracies in documentation or breaking updates on React Native itself. Since React Native applications uses different JavaScriptCore depending on the platform where code is executed, recognizing and debugging issues are sometimes harder. (Peal, 2018a, Ebel, 2018.)

Last condition relates to the organizational structure of company and abilities of team of developers to adopt React Native. React Native suits best for situation where the team is building a new application from scratch. The troubles that both Airbnb and Udacity encountered originated mostly from the fact that they already had a mobile application written in Objective-C and Java. As VanToll (2018) states, integrating frameworks like React Native or NativeScript to existing applications can be a laborious and complex. Already set workflows and developer teams can raise organizational challenges when converting platform specific teams into more cross-platform. Things like team collaboration, testing and debugging needs to be addressed. For example, one change in Java code for Android application, might mean that also React Native code needs to be changed. This means, that iOS application needs to be tested in order to make sure that nothing is broken. Also, if codebase is much fragmented to Java, Objective-C and React Native code, one misses the benefit of cross-platform development. (Peal, 2018b, Ebel, 2018).

12.3 Conclusion and future research options

One of the biggest benefit of technologies like React Native is the code share between mobile platforms. Because of the limitations of this thesis, only the iOS-version of application was delivered. Nevertheless, the experiences of Airbnb are promising. They were able to achieve 95-100% code share between Android and iOS in most features that were written in React Native. (Peal, 2018a). Because of that, I would say that cross-platform mobile application development is best suit for situation where you are actually going to build almost similar application in both mobile platforms. On the other hand, if you need different functionalities or property sets in Android than in iOS, it might be a good idea to look for Java or Objective-C.

Since most cross-platform technologies relies so heavily on web technologies, they are especially useful in a situation where you already have a group of developers that are familiar for example with JavaScript. Even better, if you could use the same team that build the website in the first place, since they are already familiar with the content and technical details of the website.

The new query languages for APIs like GraphQL are interesting alternatives for Rest APIs and are coming to CMS like WordPress ("GraphQL", 2019; "GraphQL API for WordPress", 2019). GraphQL enables developers to fetch only the data that is needed and nothing more and is gaining attention and popularity especially in the community of React developers. How much this will affect the developers in WordPress community is yet to be seen but is something that definitely should be studied in the coming years. It would be interesting to write the same demonstration application using WP GraphQL and compare the differences.

Peal (2018a) raises the performance issues in React Native application with long lists. This issue did not emerge in my demonstration application since there are only a few recipes. It would have emerged, if there would be hundreds of recipes to be shown in one list. That is why I would suggest that at least React Native is best suited for applications that doesn't contain long lists with hundreds of items. According to VanToll (2018) the case would be better in NativeScript. This performance issue will most likely be solved in the upcoming rewrite of React Native architecture (Parashuram, 2018).

As conclusion I would suggest that cross-platform development solutions are beneficial for small and middle size businesses. One should also seek synergistic advantages by using the same kind of technology in both mobile and web platforms. If website is created with React, it would be beneficial to use React Native for mobile development. However more study needs to be done relating to this kind of cross-platform development. It would be interesting to see how great percent of the code could be shared between web, iOS and Android applications, if web UI is developed with React.

BIBLIOGRAPHY

Advanced Custom Fields. Advanced Custom Fields for WordPress Developers. Accessed December 2, 2018 https://www.advancedcustomfields.com/

Angulo, E. & Ferre, X. (2014). A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX. In *Proceedings of the XV International Conference on Human Computer Interaction* (pp. 27:1-27:8). New York, NY, USA: ACM.

Apple. Branding - Visual Design - iOS - Human Interface Guidelines - Apple Developer. Accessed September 26, 2018 https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/branding/

Bigio, M. (2016). Introducing Hot Reload. Accessed May 18, 2019 https://facebook.github.io/react-native/blog/2016/03/24/introducing-hot-reloading

Bishop, J. & Horspool, N. (2006). Cross-Platform Development: Software that Lasts. *Computer,* 39(10), 26-35.

Boudreau, K. & Lakhani, K. (2009). How to Manage Outside Innovation. *MIT Sloan Management Review,* 50(4), 69-76. http://search.proquest.com/docview/224962173?accountid=11774

Charland, A. & Leroux, B. (2011). Mobile Application Development: Web vs. Native. *Commun ACM,* 54(5), 49-53. http://doi.acm.org/10.1145/1941487.1941504

Dalmasso, I., Datta, S. K., Bonnet, C. & Nikaein, N. (2013). Survey, comparison and evaluation of cross platform mobile application development tools. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International* (pp. 323-328).

Delia, L., Galdamez, N., Thomas, P., Corbalan, L. & Pesado, P. (2015). Multi-platform mobile application development analysis. In *Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on* (pp. 181-186).

Draft: JSX Specification. (2014). Accessed November 30, 2018 https://facebook.github.io/jsx/.

Ebel, N. (July 3, 2018). React Native: A retrospective from the mobile-engineering team at Udacity. Accessed February 22, 2019 https://engineering.udacity.com/react-native-a-retrospective-from-the-mobile-engineering-team-at-udacity-89975d6a8102

Eskola, R. (2018). *React Native Performance Evaluation*. Master thesis. Aalto University.

Facebook Inc. (2018a). Performance. Accessed November 30, 2018 https://facebook.github.io/react-native/docs/performance.html

Facebook Inc. (2018b). View. Accessed November 30, 2018 https://facebook.github.io/react-native/docs/view.

Facebook Inc. (2018c). Platform Specific Code. Accessed December 1, 2018 http://facebook.github.io/react-native/docs/platform-specific-code

Facebook Inc. (2018d). Native Modules. Accessed December 1, 2018 http://facebook.github.io/react-native/docs/native-modules-ios

Facebook Inc. (2018e). StyleSheet. Accessed December 1, 2018 https://facebook.github.io/react-native/docs/stylesheet

Facebook Inc. (2018f). Debugging. Accessed December 1, 2018 https://facebook.github.io/react-native/docs/debugging

Facebook Inc. (2018g). State. Accessed December 2, 2018 https://facebook.github.io/react-native/docs/state

FacetWP. Introducing the FacetWP REST API. Accessed December 2, 2018 https://facetwp.com/introducing-the-facetwp-rest-api/

Galitz, W. O. (2007). The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques. Indianapolis, IN: Wiley.

Gaouar, L., Benamar, A. & Bendimerad, F. T. (2015). Model Driven Approaches to Cross Platform Mobile Development. In *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication* (pp. 19:1-19:5). New York, NY, USA: ACM.

Garret, J. J. (2002). A visual vocabulary for describing information architecture and interaction design. Accessed September 26, 2018 http://www.jjg.net/ia/visvocab/

Garret, J. J. (2011). The Elements of User Experience: User-Centered Design for the Web and Beyond, Second Edition. (2nd ed). Berkely: New Riders.

Ghazawneh, A. & Henfridsson, O. (2013). Balancing platform control and external contribution in third-party development: the boundary resources model. *Information Systems Journal,* 23(2), 173-192.

Gould, J. (2015). WP REST API Part 1: Creating a Mobile App with WP-API and React Native. Accessed March 16, 2016 https://deliciousbrains.com/creating-mobile-app-wp-api-react-native/

GraphQL. Accessed May 20, 2019 https://graphql.org/

GraphQL API for WordPress. Accessed May 20, 2019 https://www.wpgraphql.com/

Grönroos, C. (2007). Service management and marketing: customer management in service competition. (3rd ed). Chichester: Wiley.

Hassenzahl, M. & Tractinsky, N. (2006). User experience - a research agenda. *Behaviour & Information Technology,* 25(2), 91-97. http://dx.doi.org/10.1080/01449290500330331

Heikkilä, M. (2017). Vaikuttamisen kärkenä kestävä arki. Accessed May 18, 2019 https://www.martat.fi/wp-content/uploads/2018/04/Martat-vuosikertomus-2017.pdf

Heikkilä, M. (2018). Marttaliiton toimintasuunnitelma 2019. Accessed May 18, 2019 https://www.martat.fi/wp-content/uploads/2018/04/Marttaliiton-toimintasuunnitelma-2019-diat.pdf

Heitkötter, H., Hanschke, S. & Majchrzak, T. (2013). Evaluating Cross-Platform Development Approaches for Mobile Applications. 140120-138. http://dx.doi.org/10.1007/978-3-642-36608-6_8

Hevner, A. R., March, S. T., Park, J. & Ram, S. (2004). Design Science in Information Systems Research. *MIS Q.,* 28(1), 75-105. http://dl.acm.org/citation.cfm?id=2017212.2017217

Hiltunen, M., Laukka, M. & Luomala, J. (2002). *Mobile user experience.* Helsinki: Edita, IT Press.

IDC Research. Smartphone OS Market Share, 2015 Q2. Accessed May 19, 2019 http://www.idc.com/prodserv/smartphone-os-market-share.jsp

In English. Accessed May 18, 2019 https://www.martat.fi/in-english/

Infinite Red. (2018). Ignite CLI is Here. Accessed December 2, 2018 https://infinite.red/ignite

Introducing JSX. Accessed November 30, 2018 https://reactjs.org/docs/introducing-jsx.html

JavaScript Environment. Accessed September 23, 2018 https://facebook.github.io/react-native/docs/javascript-environment.

Jokela, T., Iivari, N., Matero, J. & Karukka, M. (2003). The Standard of User-centered Design and the Standard Definition of Usability: Analyzing ISO 13407 Against ISO 9241-11. In *Proceedings of the Latin American Conference on Human-computer Interaction* (pp. 53-60). New York, NY, USA: ACM.

Joorabchi, M. E., Mesbah, A. & Kruchten, P. (2013). Real Challenges in Mobile App Development. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on* (pp. 15-24).

Konicek, M. (November 25, 2015). *Under The Hood of React Native | Martin Konicek | Reactive 2015* [video]. Accessed November 30, 2018 https://www.youtube.com/watch?v=8N4f4h6SThc.

Kotler, P. & Armstrong, G. (2012). *Principles of marketing.* (14th ed). Boston: Pearson Prentice Hall.

Kotliarskyi, A. (August 12, 2015). *Alexander Kotliarskyi - React Native: Under the Hood | YGLF2015* [video]. Accessed November 30, 2018 https://www.youtube.com/watch?v=hDviGU-57lU.

Law, E. L., Roto, V., Hassenzahl, M., Vermeeren, A. P. O. S. & Kort, J. (2009). Understanding, Scoping and Defining User Experience: A Survey Approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 719-728). New York, NY, USA: ACM.

Le Goaer, O. & Waltham, S. (2013). Yet Another DSL for Cross-platforms Mobile Development. In *Proceedings of the First Workshop on the Globalization of Domain Specific Languages* (pp. 28-33). New York, NY, USA: ACM.

Looper, J. (2015). A Guide to JavaScript Engines for Idiots. Accessed September 23, 2018 https://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/.

Lusch, R. F. & Vargo, S. L. (2006). Service-dominant logic: reactions, reflections and refinements. *Marketing Theory,* 6(3), 281-288.

Majchrzak, T. A., Biørn-Hansen A. & Grønli, T. (2017). Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks. *HICSS.*

Majrashi, K., Hamilton, M. & Uitdenbogerd, R. L. (2015). Multiple user interfaces and cross-platform user experience: theoretical foundations. In *CCSEA,* 5(2), 43-57. http://airccse.org/V5N33.html

Markus, M. L., Majchrzak, A. & Gasser, L. (2002). A Design Theory for Systems that Support Emergent Knowledge Processes. *MIS Quarterly,* 26(3), 179-212. http://search.ebscohost.com/login.aspx?direct=true&db=bsh&AN=7277732&site=ehost-live

Nielsen, J. (2012). Usability 101: Introduction to Usability. Accessed May 1, 2016 https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Norman, D. & Nielsen, J. The Definition of User Experience. Accessed March 28, 2016 https://www.nngroup.com/articles/definition-user-experience/

Nunkesser, R. (2018). Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development. In *Mobile Software Engineering and Systems, 2018 5th International Conference on* (pp. 1-15).

Ohrt, J. & Turau, V. (2012). Cross-Platform Development Tools for Smartphone Applications. *Computer, 45*(9), pp. 72-79.

Olsen, D. R.,Jr (1998). *Developing user interfaces.* San Francisco (CA): Morgan Kaufmann.

Pallot & Pawar. (2012). A holistic model of user experience for living lab experiential design. In *Engineering, Technology and Innovation (ICE), 2012 18th International ICE Conference on* (pp. 1-15).

Parashuram, N. (October 27, 2018). *React Native's New Architecture - Parashuram N - React Conf 2018* [video]. Accessed November 30, 2018 https://youtu.be/UcqRXTriUVI.

Peal, G. (June 19, 2018a). React Native at Airbnb: The Technology. Accessed February 22, 2019 https://medium.com/airbnb-engineering/react-native-at-airbnb-the-technology-dafd0b43838

Peal, G. (June 19, 2018b). Building a Cross-Platform Mobile Team. Accessed February 22, 2019 https://medium.com/airbnb-engineering/building-a-cross-platform-mobile-team-3e1837b40a88

Pedersen, R. (March 9, 2016). How Fuse differs from React Native and NativeScript. Accessed February 20, 2019 https://blog.fusetools.com/how-fuse-differs-from-react-native-and-nativescript-525344f02aaf

Peffers, K., Tuunanen, T., Rothenberger, M. & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *J.Manage.Inf.Syst.,* 24(3), 45-77. http://dx.doi.org/10.2753/MIS0742-1222240302

Quinn, J. M. & Tran, T. Q. (2010). Attractive Phones Don'T Have to Work Better: Independent Effects of Attractiveness, Effectiveness, and Efficiency on Perceived Usability. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 353-362). New York, NY, USA: ACM.

Raita, E. & Oulasvirta, A. (2011). Too good to be bad: Favorable product expectations boost subjective usability ratings. *Interact Comput,* 23(4), 363-371.

Schrock, N. (February 24, 2016). *React.js Conf 2016 - Nick Schrock – Keynote* [video]. Accessed November 30, 2018 https://www.youtube.com/watch?v=MGuKhcnrqGA.

Seffah, A. & Javahery, H. (2004). Multiple user interfaces : cross-platform applications and context-aware interfaces. Hoboken NJ: J. Wiley,.

Smutny, P. (2012). Mobile development tools and cross-platform solutions. In *Carpathian Control Conference (ICCC), 2012 13th International* (pp. 653-656).

Tiwana, A., Konsynski, B. & Bush, A. A. (2010). Platform Evolution: Coevolution of Platform Architecture, Governance, and Environmental Dynamics. *Information Systems Research,* 21(4), 675-687. http://search.ebscohost.com/login.aspx?direct=true&db=bsh&AN=57155470&site=ehost-live

Tulimäki, J. (2015). Käytettävyyden ja käyttäjäkokemuksen suhde matkapuhelimia vertailevassa käyttäjätutkimuksessa. Master thesis. University of Jyväskylä

Upgrading JSC. Accessed September 23, 2018 https://github.com/facebook/react-native/issues/19737

Usage statistics and market share of WordPress for websites. Accessed May 19, 2019 https://w3techs.com/technologies/details/cm-wordpress/all/all

VanToll, TJ. (February 16, 2015). How NativeScript Works. Accessed September 23, 2018 https://developer.telerik.com/featured/nativescript-works/

VanToll, TJ. (June 27, 2018). Would Airbnb Have Fared Better With NativeScript Instead of React Native?. Accessed February 22, 2019 https://www.nativescript.org/blog/would-airbnb-have-fared-better-with-nativescript-instead-of-react-native

Wäljas, M., Segerståhl, K., Väänänen-Vainio-Mattila, K. & Oinas-Kukkonen, H. (2010). Cross-platform Service User Experience: A Field Study and an Initial Framework. In *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services* (pp. 219-228). New York, NY, USA: ACM.

Wasserman, A. I. (2010). Software Engineering Issues for Mobile Application Development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research* (pp. 397-400). New York, NY, USA: ACM.

Williams, K., Chatterjee, S. & Rossi, M. (2008). Design of emerging digital services: a taxonomy. *European Journal of Information Systems,* 17(5), 505-517. http://search.proquest.com.ezproxy.jyu.fi/docview/218760075?accountid=11774

WordPress. (2018a). Democratize Publishing. Accessed December 2, 2018 https://wordpress.org/about/

WordPress. (2018b). Database Description. Accessed December 2, 2018 https://codex.wordpress.org/Database_Description

WordPress. (2018c). REST API Handbook. Accessed December 2, 2018 https://developer.wordpress.org/rest-api/

Zagallo, T. (February 25, 2016). *React.js Conf 2016 - Tadeu Zagallo - Optimising React Native: Tools and Tips* [video]. Accessed November 30, 2018 https://www.youtube.com/watch?v=0MlT74erp60.

Zibula, A. & Majchrzak, T. (2013). Cross-Platform Development Using HTML5, jQuery Mobile, and PhoneGap: Realizing a Smart Meter Application. 14016-33. http://dx.doi.org/10.1007/978-3-642-36608-6_2

APPENDIX 1 NAVIGATIONS OF WWW.MARTAT.FI WEBSITE

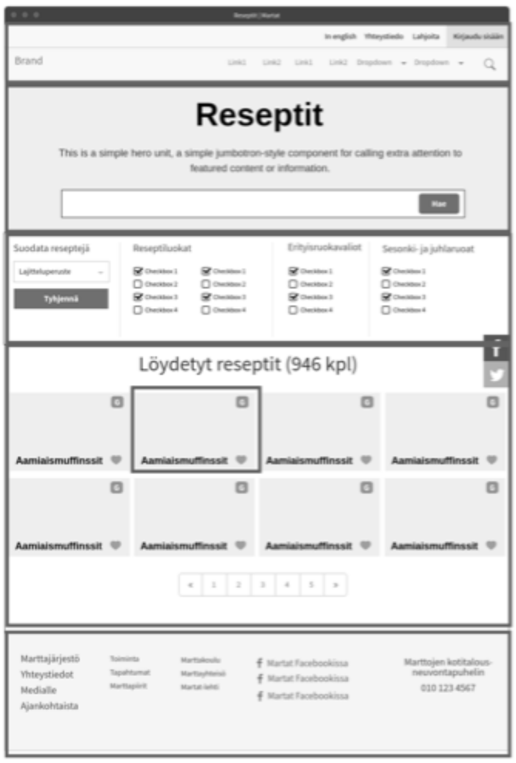
(a) Primary navigation

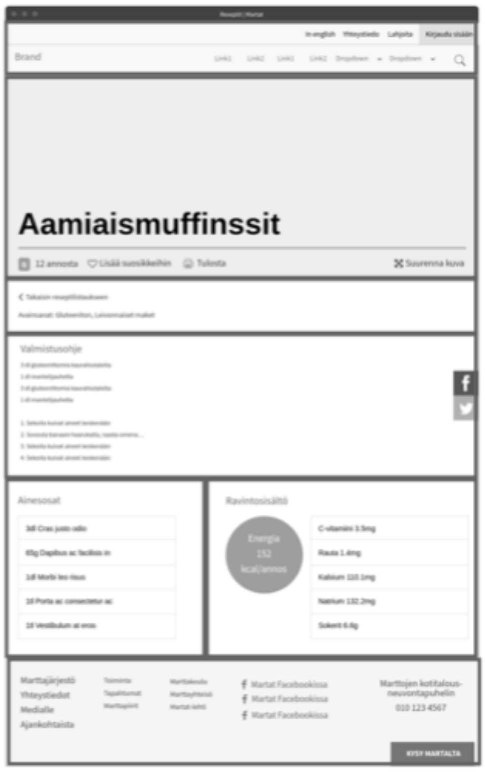
Etusivu - Fronpage
Ajankohtaista - News
  Marttaliiton uutiset - News from Martha Organization
  Marttapiirien uutiset - News from Martha District Associations
  Marttayhdistysten uutiset - News from local Marthaclubs
  Martat-lehti - Martat -magazine
  Uutiskirjeet - Newsletters
  Blogit - Blogs
  Martat sosiaalisessa mediassa - Martha organization in social media
  MartatTV - MarthaTV
Kurssit & tapahtumat - Courses and events
  Marttapiirien tapahtumat - Events from Martha District Associations
  Marttayhdistysten tapahtumat - Events from local Marthaclubs
  Juhlavuosi - Campaign page dedicated for celebration of 120 years old Martha organization
  Marttailuviikko - Campaign page dedicated for theme week
  Marttatori - Campaign page dedicated for single Martha event
  Puurokuu - Campaign page dedicated for single Martha event
  Ässäkokki-kurssit - Campaign page dedicated for single Martha event
Marttajärjestö - About the Martha Organization
  Rakenne ja organisaatio - Structure of Martha Organization
  Marttaperinne - History of Martha Organization
  Kohdennettu neuvonta - Advices for different target groups
  Marttapalvelut - Services offered by Martha Organization
  Marttapiirit - Martha District Associations
Marttailu - Being Martha
  Tule mukaan - Join us
  Yhdistyskanava - Information for local Marthaclubs
  Vapaaehtoistoiminta - Voluntary work
  Marttaopinnot - Martha studies
Marttakoulu - Martha school
  Ruoka - Food
  Ravitsemus - Nutrition
  Kodinhoito - Housekeeping
  Puutarha - Garden
  Teemat - Themes
Martanpuoti - Online store

(b) Additional navigation

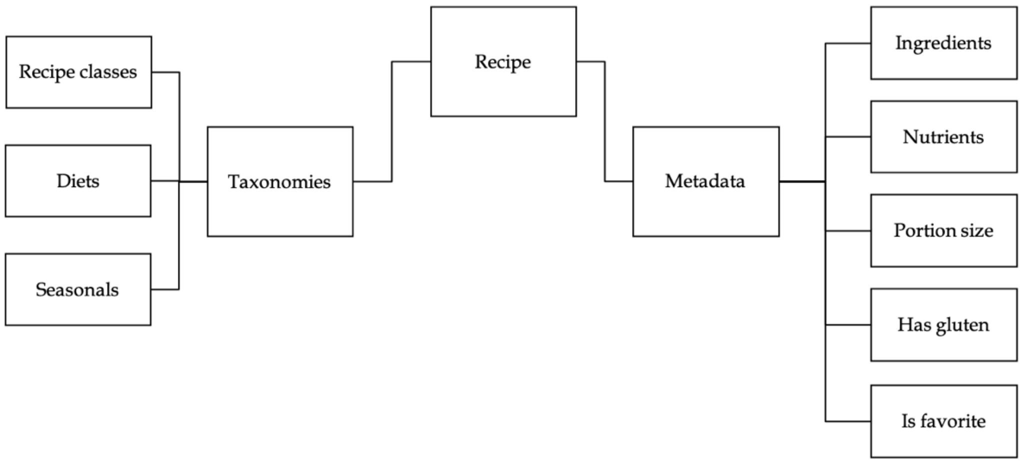
In English
Yhteystiedot - Contacts
Medialle - For media
Palaute - Feedback

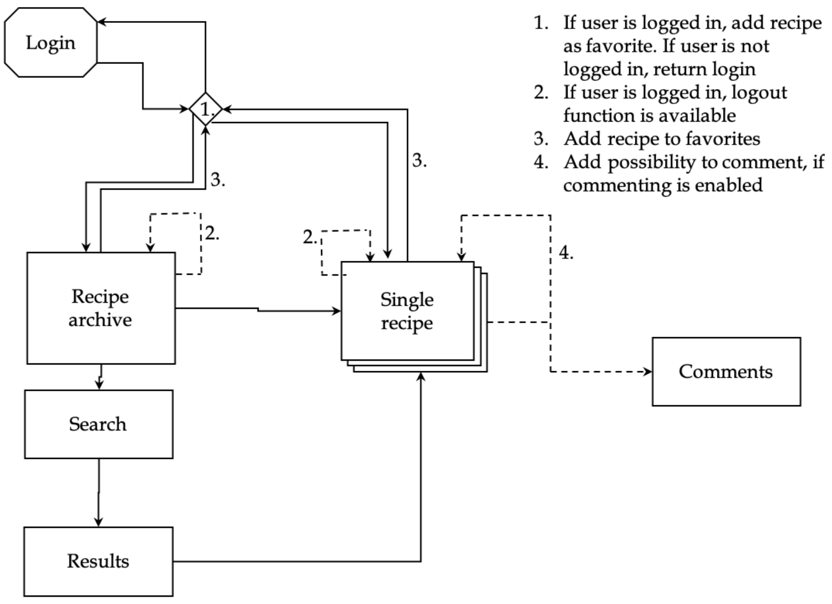
APPENDIX 2 WIREFRAMES OF SKELETON OF WEB


Reseptit
This is a simple hero unit, a simple jumbotron-style component for calling extra attention to featured content or information.
Suodata reseptejä
Reseptiluokat
Erityisruokavaliot
Sesonki- ja juhlaruoat
Tyhjennä
Löydetyt reseptit (946 kpl)
Aamiaismuffinssit
Marttajärjestö
Yhteystiedot
Medialle
Ajankohtaista
Marttojen kotitalous-neuvontapuhelin
010 123 4567

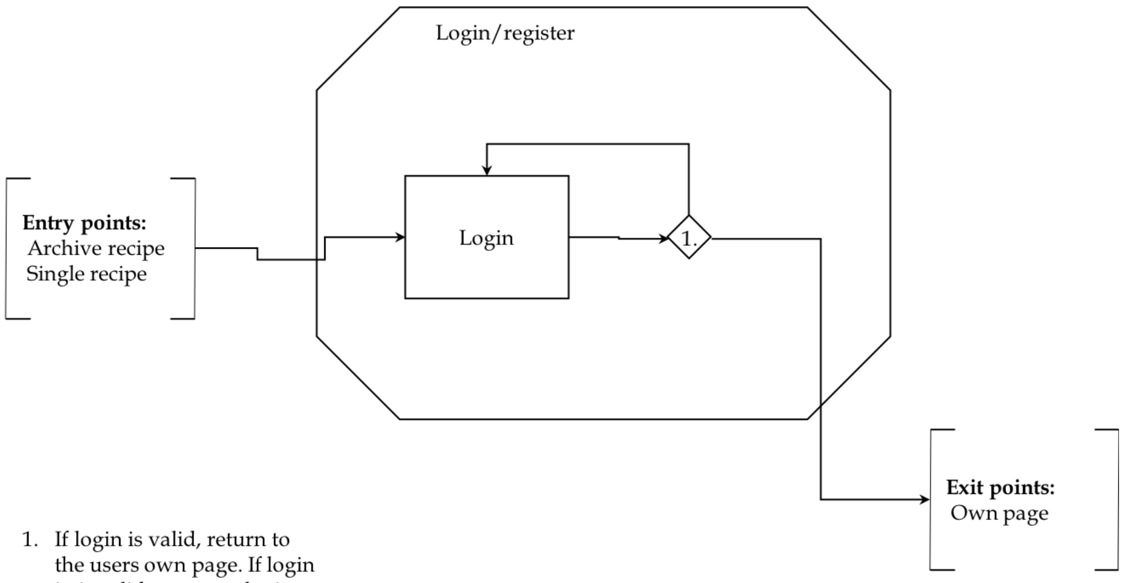


APPENDIX 3 VISUAL VOCABULARY OF STRUCTURE OF WEB





1. If user is logged in, add recipe as favorite. If user is not logged in, return login
2. If user is logged in, logout function is available
3. Add recipe to favorites
4. Add possibility to comment, if commenting is enabled



1. If login is valid, return to the users own page. If login in invalid, return to login.

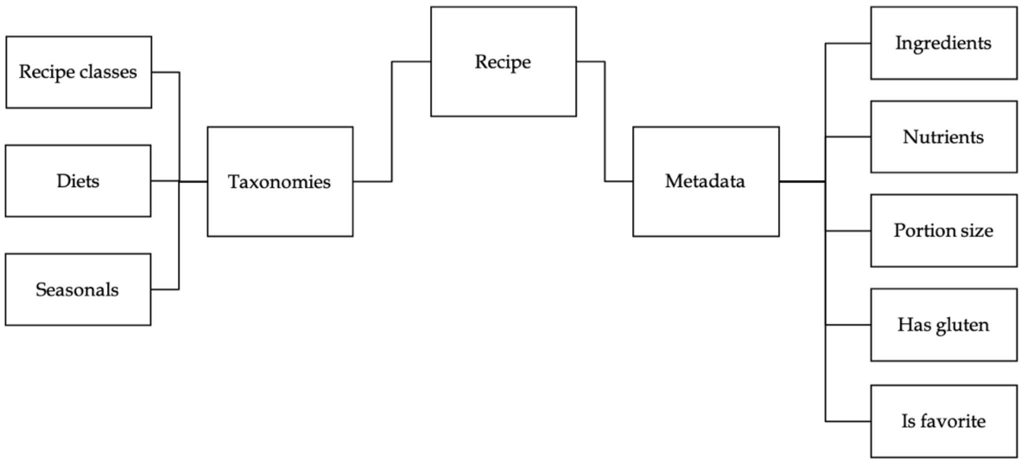
APPENDIX 4 CONTENT ELEMENTS OF SCOPE OF WEB

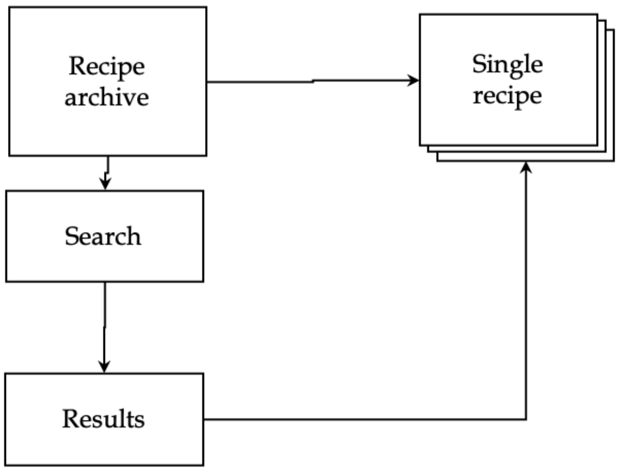
```
{
  "type": "RecipeArchive",
  "children": [
    {
      "type": "Header"
    },
    {
      "type": "ArchiveHero",
      "props": {
        "title": "String",
        "subtitle": "String",
        "backgroundImageUrl": "String",
      },
      "children": {
        "type": "SearchBar",
        "props": {
          "placeholder": "String",
          "onSubmit": "Function"
        }
      }
    },
    {
      "type": "Filters",
      "children": [
        {
          "type": "SortingColumn",
          "props": {
            "title": "String",
            "sortingOptions": "Dropdown",
            "resetButton": "Button",
            "onReset": "Function"
          }
        },
        {
          "type": "TaxonomyColumn",
          "props": {
            "title": "String",
            "options": "Checkbox",
            "onClick": "Function"
          }
        }
      ]
    },
    {
            "type": "SearchResults",
            "props": {
               "title": "String"
            },
            "children": [
               {
                  "type": "RecipeItem",
                  "props": {
                     "title": "String",
                     "backgroundImageUrl": "String",
                     "isFavorite": "Boolean",
                     "isGlutenFree": "Boolean",
                     "onClick": "Function",
                     "onAddToFavorite": "Function"
                  }
               },
               {
                  "type": "Pagination",
                  "props": {
                     "currentPage": "Number",
                     "numberOfPages": "Number"
                  }
               }
            ]
         },
         {
            "type": "Footer"
         }
      ]
   }

   {
      "type": "RecipeSingle",
      "children": [
         {
            "type": "Header"
         },
         {
            "type": "SingleHero",
            "props": {
               "title": "String",
               "backgroundImageUrl": "String",
               "portionSize": "String",
               "isGlutenFree": "Boolean",
               "addToFavorite": "Function",
               "print": "Function",
               "zoomImage": "Function"
      }
    },
    {
      "type": "Utilities",
      "props": {
        "onClick": "Function",
        "terms": "String"
      }
    },
    {
      "type": "Recipe",
      "props": {
        "title": "String",
        "content": "String"
      }
    },
    {
      "type": "Ingredients",
      "props": {
        "title": "String",
        "content": "List"
      }
    },
    {
      "type": "Nutrients",
      "props": {
        "title": "String",
        "energy": "Number",
        "nutrients": "List"
      }
    },
    {
      "type": "Footer"
    }
  ]
}
```

APPENDIX 5 CONTENT ELEMENTS OF SCOPE OF MOBILE

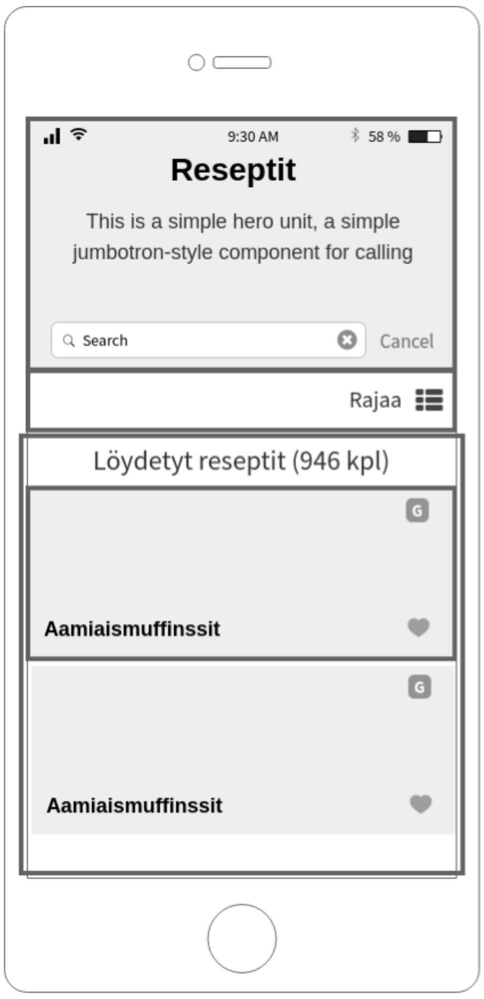
```
{
  "type": "RecipeArchive",
  "children": [
    {
      "type": "ArchiveHero",
      "props": {
        "title": "String",
        "subtitle": "String",
        "backgroundImageUrl": "String"
      },
      "children": {
        "type": "SearchBar",
        "props": {
          "placeholder": "String",
          "onSubmit": "Function",
        }
      }
    },
    {
      "type": "FiltersRow",
      "props": {
        "label": "String",
        "icon": "Icon",
        "onPress": "Function"
      }
    },
    {
      "type": "FiltersModal",
      "props": {
        "onClose": "Function"
      },
      "children": [
        {
          "type": "TaxonomyColumn",
          "props": {
            "title": "String",
            "options": "Checkbox",
            "onPress": "Function"
          }
        }
      ]
    },
    {
      "type": "SearchResults",
            "props": {
               "title": "String"
            },
            "children": [
               {
                  "type": "RecipeItem",
                  "props": {
                     "title": "String",
                     "backgroundImageUrl": "String",
                     "isFavorite": "Boolean",
                     "isGlutenFree": "Boolean",
                     "onPress": "Function",
                     "onAddToFavorite": "Function"
                  }
               }
            ]
         }
      ]
   }

   {
      "type": "RecipeSingle",
      "children": [
         {
            "type": "SingleHero",
            "props": {
               "title": "String",
               "backgroundImageUrl": "String",
               "portionSize": "String",
               "isGlutenFree": "Boolean",
               "addToFavorite": "Function",
               "backButton": "String",
               "onPressBackButton": "Function"
            }
         },
         {
            "type": "Recipe",
            "props": {
               "title": "String",
               "content": "String"
            }
         },
         {
            "type": "Ingredients",
            "props": {
               "title": "String",
               "content": "List"
            }
        },
        {
            "type": "Nutrients",
            "props": {
                "title": "String",
                "energy": "Number",
                "nutrients": "List"
            }
        }
    ]
}
```

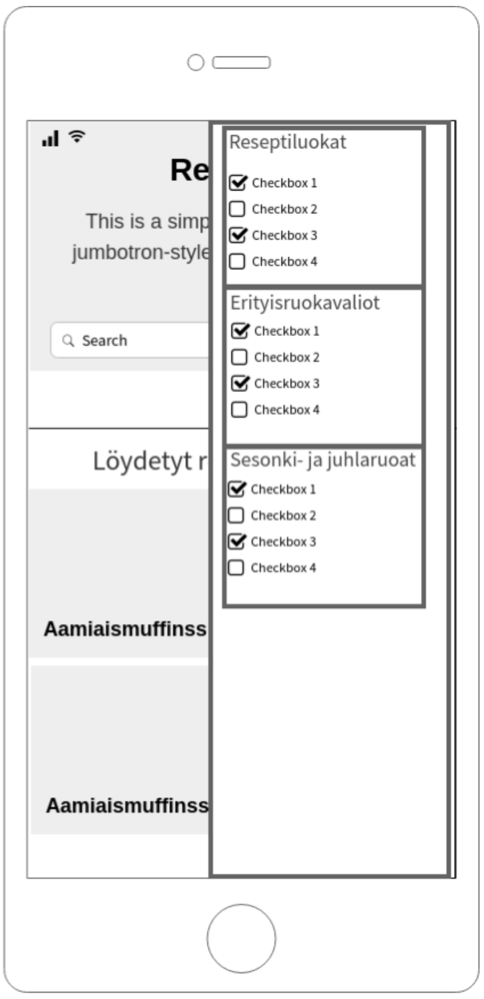
APPENDIX 6 VISUAL VOCABULARY OF STRUCTURE OF MOBILE

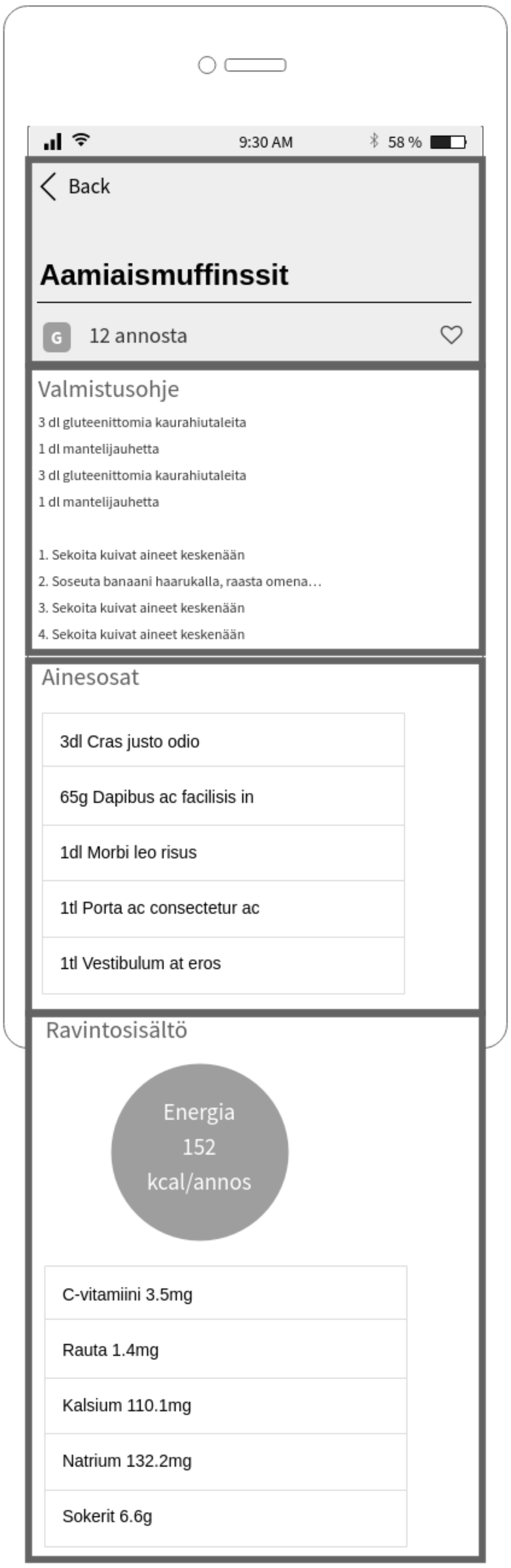




APPENDIX 7 WIREFRAMES OF SKELETON OF MOBILE


9:30 AM
58 %
Reseptit
This is a simple hero unit, a simple jumbotron-style component for calling
Search
Cancel
Rajaa
Löydetyt reseptit (946 kpl)
G
Aamiaismuffinssit
G
Aamiaismuffinssit





Aamiaismuffinssit

G 12 annosta

Valmistusohje

3 dl gluteenittomia kaurahiutaleita
1 dl mantelijauhetta
3 dl gluteenittomia kaurahiutaleita
1 dl mantelijauhetta

1. Sekoita kuivat aineet keskenään
2. Soseuta banaani haarukalla, raasta omena…
3. Sekoita kuivat aineet keskenään
4. Sekoita kuivat aineet keskenään

Ainesosat

3dl Cras justo odio
65g Dapibus ac facilisis in
1dl Morbi leo risus
1tl Porta ac consectetur ac
1tl Vestibulum at eros

Ravintosisältö

Energia
152
kcal/annos

C-vitamiini 3.5mg
Rauta 1.4mg
Kalsium 110.1mg
Natrium 132.2mg
Sokerit 6.6g

APPENDIX 8 API ENDPOINT FOR ALL RECIPES

(a) Code for creating the endpoint

```
add_action( 'rest_api_init', function () {
  register_rest_route( 'app/v1', '/getFilteredRecipes', [
    'methods'  => 'GET',
    'callback' => 'get_filtered_recipes',
  ] );
} );

function get_filtered_recipes( WP_REST_Request $request ) {
  $filters = $request->get_query_params();
  $filters = $filters['filter'];
  foreach ( $filters as $key => $filter ) {
    $filters[ $key ] = ( $filter !== '' ) ? explode( ',', $filter ) : [];
  }

  $data = [
    'facets'     => $filters,
    'query_args' => [
      'post_type'      => 'recipe',
      'posts_per_page' => 16,
      'post_status'    => 'publish',
      'orderby'        => 'title',
      'order'          => 'asc',
    ],
  ];

  $url = site_url() . '/wp-json/facetwp/v1/fetch';

  $response = wp_remote_post( $url, [
    'body' => [ 'data' => json_encode( $data ) ]
  ] );

  // If answer from FacetWP Rest API is not valid, throw new WP_Error.
  if ( 200 > $response['response']['code'] || $response['response']['code'] > 299 )
{
    $error_code = $response['response']['code'];
    $message    = $response['response']['message'];

    return new WP_Error( $error_code, $message, [ 'body' =>
$response['body'] ] );
  }

  $decoded_respose_body = convert_results_from_ids( $response['body'] );
  return $decoded_respose_body;
}

function convert_results_from_ids( $response_body = '' ) {

  // Decode string to json, so that we can handle it.
  $decoded_respose_body = json_decode( $response_body );
  $results_array        = $decoded_respose_body->results;

  // Change id to array that contains id, title, url, and isGlutenFree.
  $count = count( $results_array );
  for ( $i = 0; $i < $count; $i ++ ) {
    $recipe_id          = $results_array[ $i ];
    $results_array[ $i ] = (object) [
      'postID'       => $recipe_id,
      'title'        => get_the_title( $recipe_id ),
      'url'          => get_the_post_thumbnail_url( $recipe_id ),
      'isGlutenFree' => has_term( 'gluteeniton', 'diets', $recipe_id )
    ];
  }

  // Add new array as results
  $decoded_respose_body->results = $results_array;

  return $decoded_respose_body;
}
```

(b) Example response message
This is the example answer for HTTP GET -request to /getFilteredRecipes?filter[recipe_classes]=kasvisruoat&filter[recipe_diet]=maidoton. Some results are omitted for simplicity sake.

```
{
   "results": [
      {
         "postID": 132721,
         "title": "Aarnin inkiväärishotti",
         "url": "https://www.martat.fi/wp-content/uploads/2018/09/aarni2-
250x250.jpg",
         "isGlutenFree": true
      },
      {
         "postID": 1254,
         "title": "Carbanzokeitto",
        "url": " https://www.martat.fi /wp-
content/uploads/2016/10/carbanzokeittonetti-250x250.jpg",
        "isGlutenFree": true
      }
    ],
    "facets": {
      "recipe_classes": {
        "name": "recipe_classes",
        "label": "Reseptiluokat",
        "type": "checkboxes",
        "selected": [
          "kasvisruoat"
        ],
        "choices": [
          {
            "value": "kasvisruoat",
            "label": "Kasvisruoat",
            "depth": 0,
            "count": 50
          },
          {
            "value": "sailonta",
            "label": "Säilöntä",
            "depth": 0,
            "count": 1
          }
        ],
        "settings": {
          "show_expanded": "no"
        }
      },
      "recipe_diet": {
        "name": "recipe_diet",
        "label": "Recipe diets",
        "type": "checkboxes",
        "selected": [
          "maidoton"
        ],
        "choices": [
          {
            "value": "maidoton",
            "label": "Maidoton",
            "depth": 0,
            "count": 50
          }
        ],
        "settings": {
                "show_expanded": "no"
            }
        }
    },
    "pager": {
        "page": 1,
        "per_page": 16,
        "total_rows": 50,
        "total_pages": 4
    }
}
```

APPENDIX 9 API ENDPOINT FOR SINGLE RECIPE

(a) Code for creating the endpoint

```
add_action( 'rest_api_init', function () {
                register_rest_route( 'app/v1', '/getRecipe/(?P<id>\d+)', [
                                'methods'  => 'GET',
                                'callback' => 'get_recipe_by_id',
                ] );
} );
function get_recipe_by_id( WP_REST_Request $request ) {

  $post_id = $request['id'];

  // Bail early, if post type is not recipe
  if ( 'recipe' !== get_post_type( $post_id ) ) {
    return (object) [];
  }

  $ret_val = (object) [
    'post_id'       => $post_id,
    'title'         => get_the_title( $post_id ),
    'content'       => apply_filters( 'the_content', get_post_field( 'post_content',
$post_id ) ),
    'url'           => get_the_post_thumbnail_url( $post_id, 'page-header' ),
    'is_gluten_free' => has_term( 'gluteeniton', 'diets', $post_id ),
    'portion_size'   => get_field( 'portion_size', $post_id ),
    'right_column'   => get_field( 'right_column', $post_id ), //HTML
    'video'          => get_field( 'video', $post_id ), //url
    'keywords'       => wp_get_post_terms( $post_id,
get_post_taxonomies( $post_id ), [ 'fields' => 'names' ] ),
    'ingredients'    => get_field( 'ingredients', $post_id ),
    'nutrients'      => format_nutrients_to_array( $post_id ),
  ];

  return $ret_val;
}

function format_nutrients_to_array( $post_id ) {
  $nutrients = get_field( 'nutrients', $post_id );
  $ret_val   = [ 'energy' => [], 'nutrients' => [] ];
  foreach ( $nutrients as $key => $nutri ) {

    if ( 'Energia' === $nutri['NutrDesc'] ) {
      $ret_val['energy'] = (object) [ 'Nutr_Val' => round( $nutri['Nutr_Val'] ),
'Units' => $nutri['Units'], 'NutrDesc' => $nutri['NutrDesc'] ];
      continue;
    }

    $ret_val['nutrients'] [] = (object) [ 'Nutr_Val' => round( $nutri['Nutr_Val'],
1 ), 'Units' => $nutri['Units'], 'NutrDesc' => $nutri['NutrDesc'] ];
  }

  return $ret_val;
}
```

(b) Example response message

This is the example answer for HTTP GET -request to /getRecipe/1254 where 1254 is the post id of this recipe. Some results are omitted for simplicity sake.

```
{
  "post_id": "1254",
  "title": "Carbanzokeitto",
  "content": "<p>3-4 rkl öljyä<br />\n1 sipuli pilkottuna<br
/>\n…</p><p>Tämä on miedohko ja mukavan mausteinen keitto kylmään
talvipäivään.</p>\n",
  "url": " https://www.martat.fi /wp-
content/uploads/2016/10/carbanzokeittonetti-1920x420.jpg",
  "is_gluten_free": true,
  "portion_size": "4",
  "right_column": "",
  "video": "",
  "keywords": [
    "Kasvisruoat",
    "Maidoton",
  ],
  "ingredients": [
    {
      "amount": "3",
      "name": "Kasvisliemikuutio",
      "unit": "kpl"
    },
    {
      "amount": "400",
      "name": "Tomaattimurska",
      "unit": "g"
    }
  ],
  "nutrients": {
    "energy": {
      "Nutr_Val": 336,
      "Units": "kcal",
      "NutrDesc": "Energia"
    },
    "nutrients": [
      {
        "Nutr_Val": 15.9,
        "Units": "g",
        "NutrDesc": "Rasva"
      },
      {
        "Nutr_Val": 1.8,
        "Units": "g",
        "NutrDesc": "Rasvahapot tyydyttyneet"
      }
    ]
  }
}
```