

Developing web application and hybrid application;
How developing for different platforms
differentiated and how users experienced these

Teemu Tenkanen

University Of Turku

Department of Computing

May 2022

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin Originality Check service.

UNIVERSITY OF TURKU

Department of Computing

TEEMU TENKANEN: Developing web application and hybrid application; How developing for different platforms differentiated and how users experienced these

Masters Thesis, 64 pages, 4 appendix pages

Interaction Design

May 2022

The mobile application development process is evolving rapidly, and new frameworks are created every month. When choosing how to develop mobile application, there are three main development platforms: web application, native application, and hybrid application. These all have their positives and negatives, and they all have their own strengths when chosen for right kind of project.

Previous research has been conducted but they have mixed results when comparing. Thus, this research was made to investigate more closely how the development process for web application and hybrid application differentiated and how the end product applications differentiated, when analysing the experiences of the users. Two applications were developed during this research: React.js application for web development and React Native for hybrid development.

To analyse users experience, web and hybrid versions of the same application was given to the participants. First the participants were asked to test the main functionalities of the application, and after testing answering the survey. The survey tried to ask how positive or negative some of the features of the applications were.

In conclusion, the study shows that the hybrid application version was superior. In the survey hybrid application had slightly more positive answers and was clearly faster version of the two. The development process for the web application was easier and had better options for choosing the best suitable tools and libraries.

Keywords: React, React Native, Mobile application, Development platforms, Hybrid application, Web application

TURUN YLIOPISTO

Tietotekniikan laitos

TEEMU TENKANEN: Developing web application and hybrid application; How developing for different platforms differentiated and how users experienced these

Pro gradu -tutkielma, 64 sivua, 4 liitesivua

Vuorovaikutusmuotoilu

Toukokuu 2022

Mobiilisovellusten ohjelmointi prosessit kehittyvät nopeasti, ja uusia ohjelmointityökaluja julkaistaan joka kuukausi. Mobiilisovelluksen kehittämistä valittaessa on kolme pääkehitysalustaa: verkko-, natiivi- ja hybridisovellus. Näillä kaikilla on positiiviset ja negatiiviset puolensa, sekä omat vahvuutensa, kun ne valitaan oikeanlaiseen projektiin.

Aikaisempia tutkimuksia on tehty, mutta näiden tulokset ovat ristiriitaisia. Näin ollen tämä tutkimus tehtiin tarkentamaan, miten verkko- ja hybridisovelluksen kehitysprosessi eroavat ja miten lopputuotteet erottuivat käyttäjien kokemuksia analysoitaessa. Tämän tutkimuksen aikana kehitettiin kaksi sovellusta: React.js -sovellus verkkokehitykseen ja React Native hybridikehitykseen.

Käyttäjäkokemuksen analysoimiseksi osallistujille annettiin samasta sovelluksesta verkko- ja hybridiversiot. Ensin osallistujia pyydettiin testaamaan sovelluksen päätoimintoja ja testauksen jälkeen vastaamaan kyselyyn. Kyselyn tavoite oli mitata, kuinka positiivisia tai negatiivisia jotkin sovellusten ominaisuudet olivat.

Tutkimus osoitti, että hybridisovellusversio oli parempi kuin verkkosovellusversio. Kyselyssä hybridisovellus sai hieman enemmän myönteisiä vastauksia ja oli selvästi nopeampi versio näistä kahdesta. Verkkosovelluksen kehitysprosessi oli helpompaa ja siinä oli paremmat mahdollisuudet valita sopivimmat työkalut ja kirjastot.

Contents

1.	Introduction	1
2.	Development platforms	2
2.1	Web application	2
2.2	Native application	3
2.3	Hybrid application	4
3.	Development technologies	6
3.1	React.js	6
3.1.1	JSX	6
3.1.2	Class and functional components	7
3.1.3	Lifecycle methods	7
3.1.4	Hooks	8
3.1.5	State and props	8
3.1.6	Virtual DOM	10
3.2	React Native	11
3.2.1	Similarities with React.js	11
3.2.2	Differences with React.js	12
3.2.3	Native and core components	12
3.2.4	Custom components	13
3.2.5	Platform specific code and platform-specific extensions	14
3.3	Node.js	15
3.4	Firebase Realtime Database	15
4.	Pizza corner applications	17
4.1	Planning	18
4.2	Views	19
4.3	Version control	22
5.	React application	24
5.1	Setting up the environment	24
5.2	Projects folder and file structure	26
5.2.1	Application.js	28
5.2.2	MainPage.js	31
5.2.3	RestaurantPage.js	32
5.2.4	CreateNewReview.js	33
5.2.5	RequestNewRestaurant.js	33
6.	React Native application	35
6.1	Setting up the environment	35
6.2	Running the application on android phone	37

6.3	Projects folders and file structure	39
6.3.1	Application.js	40
6.3.2	MainPage.js	43
6.3.3	RestaurantPage.js	45
6.3.4	CreateNewReview.js	46
6.3.5	RequestNewRestaurant.js	46
7.	Research and Testing the Applications	47
7.1	Methodology	47
7.2	Research Design	48
7.3	Description of the case study	48
8.	Results and analysis	50
8.1	Expectations and Hypotheses	50
8.2	Development experience	51
8.3	Data presentation	53
8.4	Data analysis	59
9.	Conclusions	61
9.1	Answers to the Research Questions	61
9.1.1	How did developing for two different platforms differentiated?	61
9.1.2	Was either one of the application versions better?	62
9.1.3	What factors did affect the experience of the applications?	62
9.2	Discussion	62
9.3	Future works	63
	References	65

1. Introduction

This study examines how the development process for web and hybrid applications differs, and how users experience these two versions. The motivation behind this study is to examine is developing applications more beneficial with web or hybrid technologies.

In this study two applications were developed with web and hybrid application technologies. These two applications were tested, and survey was organized, which tried to measure the experience of the users with both versions. Survey data was first collected and then analysed, and applications data were compared to each other, to found out which of the two versions the participants experience was better.

In the second section three main development platforms are explained, and examples of using these different platforms are presented. Third section will introduce the main technologies that were used in the develop process of the two applications. In the fourth section the developed application's different features are explained. Fifth section will examine the main components and platform specific features of the web application. Same is done for hybrid application in the sixth section. Seventh section will examine the research's methodology, the design of the research and summary of the case study that was held. In the eight section results of the develop process, that researcher had, and the survey data is analysed. In the final section conclusions are made and future work improvements are discussed.

This research aimed to investigate how developing application for two different platforms differed from each other and does the end users experience these two application versions differently. To investigate preceding topics, an experiment was made where participants tried these applications and after that opinion and experience were examined under the consideration of the following research questions:

RQ1. How did the development for two platforms differed from each other?

RQ2. Was either one of the application versions better?

RQ2. What factors did affect the user experience of the applications?

2. Development platforms

When developing mobile application there are multiple different approaches and platforms to choose. The most used and common approaches are *web, native, and hybrid application*. In the following subsections, these approaches are explained and briefly describe in what projects and cases they are best suitable.

2.1 Web application

A web application (or web app) is an application that does not run on locally on OS (Operating System) of the device, such as computer or mobile device, but it runs on a web server. User can access the application via a web browser with active internet connection. Web applications are supported now days both in desktop and mobile devices.

Technologies that are mostly used in web applications are HTML5, JavaScript, and CSS. HTML (HyperText Markup Language) is the standard markup language for documents to be displayed in a browser. HTML is the building block which creates containers for the content to show. JavaScript (often shortened JS) is a programming language that is used to handle the logic of the web application. CSS (Cascading Style Sheets) is a style sheet language that is used for adding styles and visual effects. To the content these effects can be used, for example, font modifications, sizes, shapes, colors, and layouts changes.

Web applications are considered as the cheapest option when developing applications (Martin, S 2020). This is because developers can develop one project that can be used on any OS, such as iOS, Android or Windows. Maintaining only one project instead of two or three is cheaper and faster. There are also reasons why not to use web applications. Good internet connection is needed for the application to work, and it is executed in the browsers. Web applications have often worst user experience and can have more performance issues compared to the native applications (Martin, S 2020).

In Sopegno's case study (Sopegno, 2016) web application called AMAC (Agricultural Machine App Cost Analysis) was developed for determining the machinery cost in field operations. The mobile application platform was chosen because the application needed

to be easy to use, to be available free of charges and does not require any installations on the end user's device. AMAC was developed to be cross-platform application, meaning that it can be operated in any device. The web application was ideal choice for these reasons and performed great in this kind of use.

2.2 Native application

Native application is an application that needs to be installed on a device before it can be used. These applications do not necessarily need internet connection to work and can be operated when device is offline. These applications have been developed for specific OS by using platform specific programming languages and IDE (Integrated Development Environment). For Android the programming language is Java or Kotlin, and for iOS it is Swift or Objective-C.

Native applications have often better performance than web applications. This can be seen, for example, when using applications with heavy graphics or complex calculations. Native applications have access to platform specific features such as GPS, camera, microphone, and push notifications. Web applications have some of these features and before they can be used user needs to give consent for the application to access those. There are also big drawbacks why not to use native applications. A least two different projects are needed to implement the applications, because, for example, Android and iOS are using different programming languages. It also takes more time and resources to create a native application. This is expensive when considering the development and the maintenance of the project (Martin, 2020).

In Ajayi's research (Ajayi, 2018) performance of native and hybrid Android applications were evaluated. The goal of the research was to compare the performance of both platforms and to explore the limitation of the mobile development frameworks. The research concluded that the native application was superior to hybrid application in all conducted tests. The bad performance of the hybrid application was the most likely reason why user's satisfaction of the application was lower. This research supports the Martin's hypothesis for native applications to have overall better performance, compared to the other platforms (Martin, 2020).

2.3 Hybrid application

The separation in operating systems and programming languages of the Android and iOS has pushed people to think development solutions that would be suitable for both. Web application technologies are often used as a base for hybrid application frameworks.

Hybrid application is an application that combines elements of both native applications and web applications. They are essentially web applications that have been put in shell of a native application. Hybrid application needs to be installed locally to work. The same codebase can be used on both operating systems, and this makes it affordable option to consider. Hybrid applications utilize same kinds of technologies that web applications are using, HTML, CSS, and JavaScript. These are then encapsulated within a native application, and using plugins these applications have same access, as native applications, to the device's platform specific features. Instead of the application being shown within the user's browser, it is run from within a native application and its own embedded browser, which is essentially invisible to the user. For example, an iOS application would use the WKWebView to display our application, while on Android it would use the WebView element to do the same function. The code is then embedded into a native application wrapper using solutions like Cordova or React Native. These solutions create native container in which it will load the web application.

Combining web technologies with native building block of both Android and iOS is another way to implement hybrid application. Instead of using web application technologies, such HTML and CSS, these applications are written in JavaScript. JavaScript is then translated into native application specific user interface elements.

Hybrid applications are quick to develop and with one codebase it makes distribution for multiple platform easy. But hybrid applications are often low performing compared to native applications. The performance of the application depends how powerful the device is. If the device is fast, then the performance will be higher. Developing hybrid applications can be slow for beginners because one needs to learn third party service like React Native or Cordova and these frameworks have usually a large learning curve (Martin, S 2020).

In Willocx' (Willocx, 2015) research the benchmarks reveal that Cordova applications use more device's CPU (central processing unit) and memory compared to the native applications. This affects the performance of the application, and the application might feel slow. This is not the case when high-end devices are used, as these devices have better components for processing, for example faster CPU and memory units. Mobile devices are improving fast, and the limitation of the device might not affect as much in the future.

3. Development technologies

In this chapter two development technologies for web and hybrid applications are presented. The basic development process and the most important features of these technologies are presented and explained.

3.1 React.js

React.js (also known as React or ReactJS), developed by Facebook, is a free open-source JavaScript library that is designed for building user interfaces. It is one of the most popular web development JavaScript libraries in the world (Liu, 2021), and the reason for that is perhaps the component focused approach and that it is quick to learn. The strength of the React.js is its size and simplicity because it is not a full framework but only a JavaScript library. However, for creating fully functional React application developers will need use of additional libraries for routing and client-side functionalities. React.js can be used as a base for single-page application (SPA) or mobile application development. Core features of React.js are explained in the following subsections.

3.1.1 JSX

JSX (JavaScript XML) has similar syntax with HTML (Hypertext Markup Language) and XML (Extensible Markup Language). It has all the same functionalities that JavaScript has and the same opening and closing `<>` `</>` - symbols that XML has. It is a syntax extension to JavaScript, and it describes what the UI (user interface) should look like. JSX allows developers to write HTML elements or code similar syntax in JavaScript and place them in the DOM (document object model) without any extra methods. JSX converts everything to pure JavaScript code (see Figure 1).

```
src > components > JS HelloWorld.js > HelloWorld
1  class HelloWorld extends React.Component{
2    |   render(){
3    |     |   return (<p>HelloWorld</p>);
4    |     |   }
5    |   }
```

Figure 1. Hello world-class component

Figure 1, presents a simple React.js component that displays the text “HelloWorld” to the screen. The `render ()` function handles the rendering to the users screen. We can see that JSX uses HTML-tags `<p></p>` inside the JavaScript code. Other custom JavaScript or lifecycle methods and hooks can be written top or under the `render` method. JSX will throw error if the HTML is not correct or if the HTML misses parent element.

3.1.2 Class and functional components

React.js components can be written two different ways. Using classes (see Figure 1) or as a functional component which are basically normal JavaScript functions (see Figure 2).

```
src > components > JS HelloWorld.js > HelloWorld
1  function HelloWorld (props){
2    return (<p>HelloWorld</p>);
3  }
```

Figure 2. Hello World- functional component

When writing function components, one gets more shorter and simpler approach (see Figure 2). The method `render ()` is not used in function components. Function components are usually used when nested components are required or the component itself does not include much logic. Class components are used when there is a need for more logic. For example, when one needs to use lifecycle methods (explained in chapter 3.1.3) class component should be used, because the function components do not support them.

3.1.3 Lifecycle methods

In applications with many components, it is important to free up resources taken by the components when they are destroyed. Each component in React has a lifecycle which can be monitored and manipulated during its three main phases. These phases are mounting, updating, and unmounting. In class components, we can declare special methods called lifecycle methods such `componentDidMount ()` and `componentWillUnmount ()` when component mounts and unmounts. For example,

`componentDidMount()` runs after a component output has been rendered to the DOM. This method can be used for doing requests to the server-side or update the state object (explained in 3.1.5).

3.1.4 Hooks

Hooks are functional component specific feature that lets the developer to use state and other React features without writing a class. There are three rules for hook.

1. Hooks can only be called inside React function components
2. Hooks can only be called at the top level of a component
3. Hooks cannot be conditional.

```
src > components > JS HelloWorld.js > HelloWorld
1  import React, { useState } from "react";
2  import ReactDOM from "react-dom";
3
4  function HelloWorld (props){
5      const [text, setMessage] = useState("HelloWorld");
6
7      return (
8          <>
9              <h1>{text}</h1>
10             <button
11                 type="button"
12                 onClick={() => setMessage("HelloStudent")}
13             >CLICK ME</button>
14         </>
15     );
16 }
```

Figure 3. Example use of Hooks

Figure 3 shows how to use the `useState()` hook to keep track of the application state. Without this hook there is no other way to update the state, when using functional components.

3.1.5 State and props

React components have a built-in state object which can store property values. The state object is initialized in the constructor and when the state value changes, the component re-renders itself. State object can contain multiple properties, but every property needs to have a unique name. If one wants to refer to a state object, it is required to use `this.state.propertyname` syntax (see Figure 4). If the state needs updating,

`setState()` method is required. In functional components, we can use `useEffect()` hook to update the state. This method is also used for initialization and clean-up functions, so whenever changes are made to the state `useEffect()` hook is needed.

```
src > components > JS HelloWorld.js > HelloWorld > constructor
1
2 class HelloWorld extends React.Component{
3   constructor(props) {
4     super(props);
5     this.state = {
6       name: "Teemu Tenkanen",
7       phone: "1234567890",
8       year: 1994,
9     };
10  }
11  render(){
12    return (
13      <>
14        <h1>My name is {this.state.name}, born {this.state.year}</h1>
15        <p>
16          Phone number {this.state.phone}
17        </p>
18      </>
19    );
20  }
21 }
```

Figure 4. Examples of state and props

Props are arguments that are passed into the React components. If the developer wants to send props into some other component, they must use the same syntax as HTML attributes (see Figure 5).

```
const myElement = <Hello text="Hello"/>;
```

Figure 5. Props example (attribute in element)

In the component itself, the props values work in a similarly to the state syntax, `this.props.attributename`. One can send string or variables (between curly brackets) to other components this way.

3.1.6 Virtual DOM

The virtual document model (VDOM) is a programming concept where a “virtual” representation of the UI is kept in memory and synchronized with the “real” DOM by using a library such as ReactDOM. This is implemented because VDOM is much faster and efficient than the “real” DOM. When new elements are added to the UI, VDOM is created. This is represented as a tree, and each element is a node on this tree. If any of the element in the tree is changes or updated, a new tree is created. This new tree is compared to the previous tree and then see what is different. After this VDOM calculates the best solution method to make these changes to real DOM. This reduces the performance costs and makes things faster.

In React, every user interface element is a component, and each component has its own state. React listens for state changes and when the state of a component changes, React updates the VDOM tree. After VDOM has been updated, it compares itself to the previous version of the tree. This process is called diffing. When the comparison is done, React updates only those objects/components that changed to the real DOM (see Figure 6).

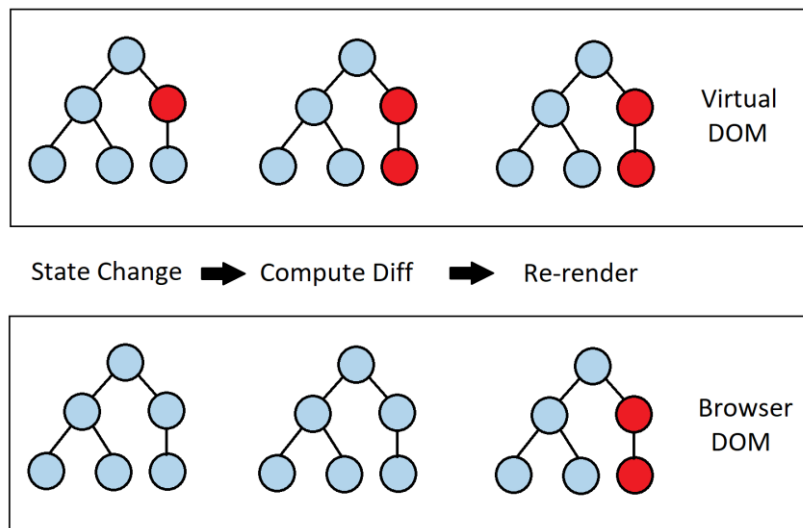


Figure 6. VDOM tree and the diffing process (Paypal, 2020)

All of these details are abstracted away from developers, and the developer only needs to know how to update the state of react components. React takes care of the rest. This makes learning to develop with React.js fast and easy.

3.2 React Native

React Native, developed by Facebook, is a free open-source UI software framework designed to use for developing applications for Android, Android TV, iOS, macOS, tvOS, Web, Windows and UWP. It allows developers to use React framework with native platform capabilities. React Native is built on React.js, so most of the React.js features are available also in React Native. Developing and the code syntax for React Native is hence very similar with React.js. The following sections elaborate the basic React Native fundamentals.

3.2.1 Similarities with React.js

Because React Native is built on React.js it has most of the same basic React concepts. State and props are handled exactly the same way. One can use hooks to update the state in functional components and use lifecycle methods like `componentDidMount()` in class components. JSX is used in React Native same way without the need to use, for instance, HTML tags.

```
1 // React Native Counter Example using Hooks!
2
3 import React, {useState} from 'react';
4 import {View, Text, Button, StyleSheet} from 'react-native';
5
6 const App = () => {
7   const [count, setCount] = useState(0);
8
9   return (
10    <View style={styles.container}>
11      <Text>You clicked {count} times</Text>
12      <Button onPress={() => setCount(count + 1)} title="Click me!" />
13    </View>
14  );
15 };
16
17 // React Native Styles
18 const styles = StyleSheet.create({
19   container: {
20     flex: 1,
21     justifyContent: 'center',
22     alignItems: 'center',
23   },
24 });
25
```

```
1 // ReactJS Counter Example using Hooks!
2
3 import React, { useState } from 'react';
4 import { useState } from 'react';
5
6 const App = () => {
7   const [count, setCount] = useState(0);
8
9   return (
10    <div className="container">
11      <p>You clicked {count} times</p>
12      <button
13        onClick={() => setCount(count + 1)}> 1) title="Click me!" />
14      Click me!
15    </button>
16    </div>
17  );
18
```

Figure 7. React Native and React.js functional components

Figure 7 presents two simple functional components, implemented with React Native and React.js. There is no difference in handling the state with hooks. We can also see that there are some differences how JSX is written and how styles are handled.

3.2.2 Differences with React.js

The main difference between React Native and React.js is that React.js can be used on every platform where web browsers is accessible, while React Native is only usable on Android and iOS. Another key difference is that React.js offers more animation than React Native. On web applications animation is done primarily on CSS, whereas React Native needs separate API or library to produce animation. Components between these two differ as well. In the next section, we compare the differences between React.js and React Native components.

3.2.3 Native and core components

When developing for Android, developers are writing views in Kotlin or Java, and in iOS development Swift or Objective-C is used. When developers are developing with React Native they invoke these views with JavaScript and React components. When the application is running React Native creates the corresponding Android and iOS views for those specific components.

In Figure 7 we can see that React Native does not use `<div>`-tags, instead it uses `<View>`-tags. `View` is the basic building block of the UI, and it is used like `div` or `<>`-tags. `View` is a container that supports layout with flexbox, style, some touch handlings, and accessibility controls. `<Text>`-tags were used also in Figure 7. This is equivalent to the `<p>`-tags, and text can be displayed inside this component. These are some ready to use building blocks that development team behind React Native created for developers, and they are called core components. There are over twenty core components but the most used ones are `View`, `Text`, `Image`, `ScrollView` and `TextInput`, `StyleSheet` and `Touchable`. `Image` component is straightforward, it is used for displaying an image. `ScrollView` works is like `View`, but the container itself is scrollable. `TextInput` is used for text input field, and it allows using on-screen keyboard to input text. `Touchable` is used for creating custom

button components or enables touch recognition features and `StyleSheet` is used for writing styles that are similar to CSS (React Native 2022).

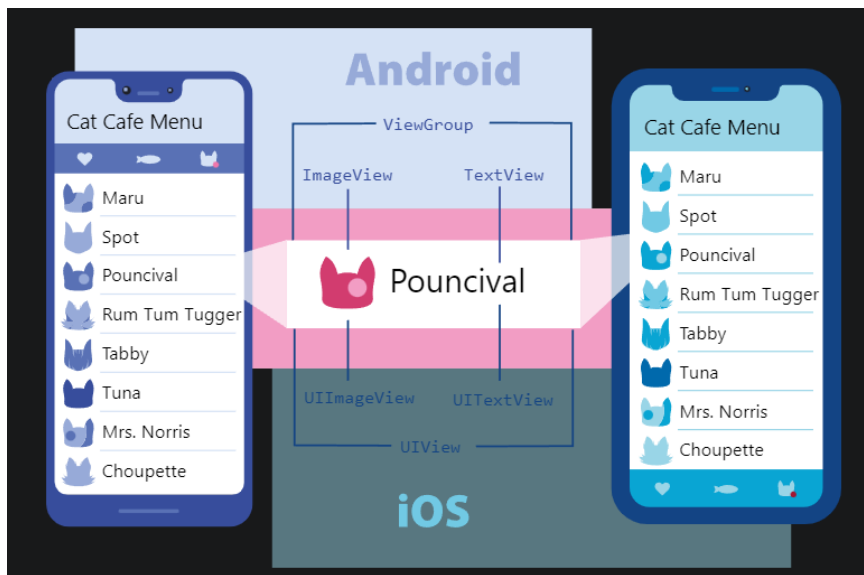


Figure 8. Compiling between two OS (React Native docs 2022)

For example, in Figure 7 `<Text>`-tags are used, when JavaScript code is compiled into an application the `Text` component is translated into corresponding native text components. For iOS, it is `<UITextView>`, and for Android, it is `<TextView>` (see Figure 8). This way application gets the native look and feeling but can continue to be developed with JSX and use only one project for two different OS.

3.2.4 Custom components

Creating React Native custom components are implemented by combining the basic core and custom components. In every component import of the React library is required (see Figure 7, row 3). In Figure 7, on the row 6 a functional component `Application` is created. On the row 9 is a return statement that has imported core component `View` wrapper and inside its `Text` and `Button` core components. Outside the return statement, is created `StyleSheet` object that is part of core components, and it is used to create style objects that are similar to CSS syntax. On the row 18, the style object container is created and it has couple of style changes that impacts the position of the items. On the row 10 the container style object is passed to the style prop. In Figure 7, the export statement is missing, but components must always be

exported if it wants to be imported into any other JavaScript file used in this project. Export is done with command `export default TheNameOfTheComponent`.

3.2.5 Platform specific code and platform-specific extensions

When developers are building application that is cross-platform there are cases where it makes sense to have platform specific parts. Some visual UI elements can be implemented differently for iOS and Android. React Native provides two ways to separate it by platform.

React Native provides Platform module that detects which platform the application is running on. One can use this detection logic to implement platform-specific code. This method is particularly good when only small part of the component is platform specific.

```
1  import {Platform, StyleSheet} from 'react-native';
2
3  const styles = StyleSheet.create({
4    height: Platform.OS === 'ios' ? 200 : 100,
5  });
6
```

Figure 9. Platform module

In Figure 9 we have example where iOS has platform specific styling. By using `Platform.OS === 'ios'` we can distinguish iOS application runs from Android. This way only the iOS devices are affected by this style change. For Android, the value will be `android`. One can also use `Platform.select` method that detects which platform is used dynamically and chooses the most fitting option. The options are `ios`, `android`, `native` and `default`. `Platform.Version` method can be used to detect the version of the Android or iOS platform.

When platform specific code is too complex to implement in conditional way, for example the code looks crowded or unreadable, one should consider splitting the file into two separate files. React Native will detect if the file has `.ios` or `.android` in the files extension and this will load the relevant platform file. For example, if there is `BigButton.ios.js` and `BigButton.android.js`, React Native will detect which one to use when `BigButton` is used in some other component.

3.3 Node.js

Node.js is free open-source back-end JavaScript runtime environment that uses V8 engine, a JavaScript engine that executes JavaScript code outside a web browser. Node.js lets developers to create command line tools with JavaScript and it is mostly used for server-side scripting. Node.js enables developers to create independent applications using JavaScript instead of running the code only in the browsers. Installing Node.js developers get NPM (node package manager) that enables the downloading and installing different JavaScript libraries (aka node modules) to the projects (Node.js 2022).

3.4 Firebase Realtime Database

Firebase Realtime Database is a cloud hosted NoSQL database that stores and syncs data between users in real-time. It is part of Firebase product family that is developed by Google. Firebase is a Backend-as-a-Service (Baas) that provides developers variety of different tools and services for development.

Instead of typical HTTP request, the Firebase Realtime Database uses data synchronization. This means that every time data changes any connected device receives the updated data in milliseconds. When device is offline Firebase Realtime Database SDK persist the data to a disk, and once the connection is restored the client device receives all the new changes that were made during offline period. The data is stored in Firebase Realtime Database in JSON format.

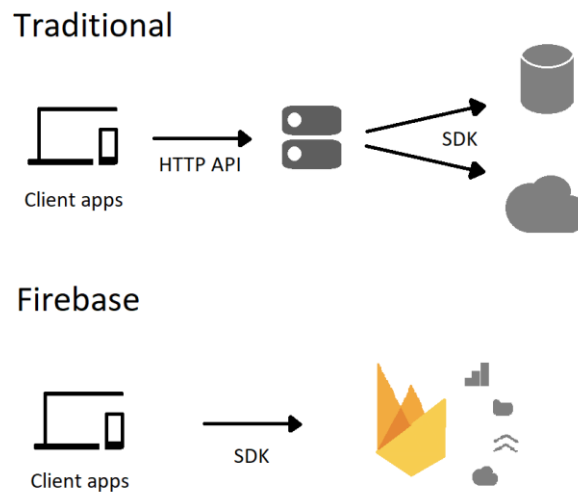


Figure 10. SDK directly calls Firebase services (Stevenson, 2018)

When setting up the Firebase Realtime Database into project client SDKs are used, provided by Firebase, and these interact directly with Firebase backend services that are operated by Google. This means that there is no need to establish any middleware between application and the service (see Figure 10), making back-end service unnecessary, and that one can write code to query the database in the client application (Google, 2022).

```

72  ✓  componentDidMount() {
73      const dbRef = firebase.database().ref();
74  ✓  dbRef.on("value", (snapshot) => {
75      let dbData = snapshot.val();

```

Figure 11. Firebase on () example

By using `firebase.database.Reference` developers can access the database instance and do write and read operations. This reference works as a listener that is triggered any time when the state of the data changes. With `set ()` method basic write operations can be made. `set ()` saves data to a specified reference and replaces any existing data at that path. To read data at path and listen for changes `on ()` method is used. When event is triggered the event call back is passed a snapshot containing all data at that location. In Figure 11 the database reference is saved into constant, on the

row 73. And on the row 74 the method is used to trigger the listener and the row 75 the current snapshot value is saved on the variable `dbData`. Database content is then accessible in variable `dbData` in JSON format. `push()` method can be used to append data to a list or replace value in the list (see Figure 12).

```
78     firebase
79     .database()
80     .ref("Restaurants/" + this.state.restaurantData.name + "/reviews")
81     .push({
82       comment: this.state.comment,
83       price: this.state.price,
84       stars: this.state.stars,
85       picture: this.state.imageUrl,
86       timestamp: date + " " + time,
87     });
```

Figure 12. Firebase push() example

4. Pizza corner applications

Pizza corner application is anonymous food review application designed specially for pizzerias. Motive behind this application was to do something new and mix it up with something very popular. At the time when the application was developed, there were not many review applications specific to pizzerias and the idea of restaurant review application was born. Messaging application called Jodel was very popular at the time when the application was developed, and Pizza Corner application got lots of inspiration from it.

Jodel is application where people can anonymously send short messages or pictures to the forum and other people can comment anonymously to them. These messages will appear people near them, for example, "here" inside one kilometer, "very close" inside two kilometres etc. Users can also follow some channels like "women", "confessions" and "fantasies" (Jodel, 2022).

Inspired by the anonymous message feature of Jodel, idea of Pizza Corner was created. There are two versions: web application and hybrid application. React.js JavaScript library was used for the web applications front-end and React Native framework was used for the hybrid application front-end. There was no need for implementing a back - end, because Firebase Realtime Database was used, and those servers are called directly

from client. For the web application Material UI component library was used, and for the hybrid application, the React Paper component library. These two had very similar styling and, at the time, there was no component library that worked on both platforms.

Development process was done on two separate projects. The React application version was developed first, and it was used as a base for the React Native version. In the following sections, we explain these two projects development process. Development of the two projects was done on the Windows operating system.

4.1 Planning

Adobe XD was used for designing the UI. Adobe XD is vector-based user experience design tool for web and mobile applications. It is developed and published by Adobe Inc. It enables website wireframing and creating interactive prototypes (Adobe, 2022) (see example in Figure 13).

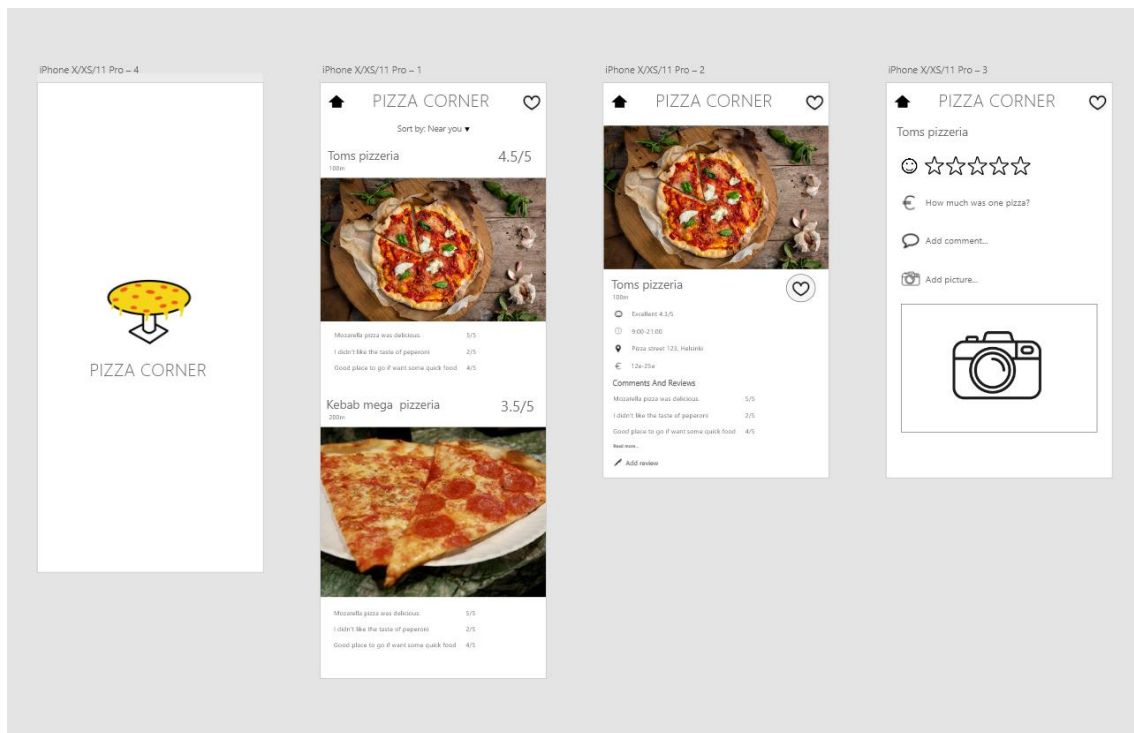


Figure 13. Pizza Corner UI protos in Adobe XD

Adobe XD was chosen for a UI design tool because it was free to use if one has only one project ongoing. Other UI and UX tools were not free, or they had too limited set of features to offer. The process of creating a prototype is straightforward. The designer

can drag and drop ready-made components like rectangles, ellipses, and text components to the screens. Own pictures can be pasted directly to the screen and modified to right fit. Transition effect can be made by choosing the element and connecting it by transition arrow to the different screen, where to move. When the design is ready the designer can run demo on computer or on the phone. For demoing the application with mobile device, separate application is needs to be downloaded.

4.2 Views

Pizza Corner application has four different views. The main page, restaurant page, review page and request new restaurant page. All the views have the quick link to the main page (the home button), the “PIZZA CORNER” title and the drawer menu button that opens drawer component from the right side. Drawer menu has only one link to the “New restaurant request” page.

In the top of the main page is search and sort tools, which the user can use to filter with the name of the restaurant or sort the data with values “All”, “Best reviews” and “Favorites”. All the restaurants are shown in their own card style component. Cards are used to display restaurant data because it groups information well and they provide large and clear links to the user. In the top part of the card are restaurants name, address, and the average review value. The most recent review’s picture is shown in the image part of the card and under that are two most recent review texts and their review score out of five (see Figure 14).

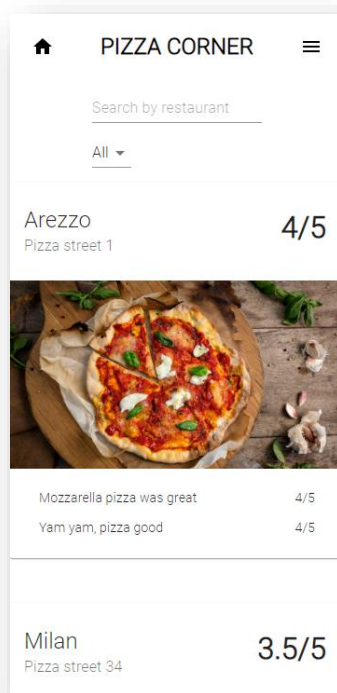


Figure 14. Pizza Corner main page

If the user pushes any of the restaurant cards in the main page, it will take them to restaurant page and shows more information about it. Restaurant page shows the most recent reviewer's picture and lists the average review score, the opening hours, the address, and the average price of pizza in that restaurant. Beside the name of the restaurant is heart icon. When clicked, the heart changes its color to pink and adds this restaurant to the user's favourite list. In the top of the main page is dropdown box that can be used to sort or filter restaurants. By choosing the "Favorites" value in the dropdown box user can see favorite restaurants. Under the restaurant info section is the "Comments and reviews" section, where each review is shown in card style. Review cards consist of picture, time stamp of the time when the review was done and the comment with the review score out of five. In the bottom of the screen is "Add a review..." button that takes user to the restaurant review page (see Figure 15).

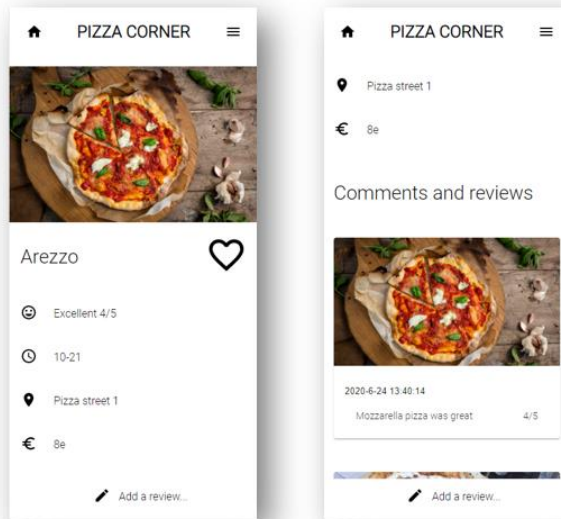


Figure 15. Restaurant page

Restaurant review page works as a form that the user can fill and send. From top to bottom filled areas are the star score of the restaurant, the price of the pizza, comment section and picture upload. When the picture is uploaded the preview of the picture is shown. Picture needs to be uploaded in png or jpg format. After all form values are filled the “Send review” button enables, and user can send the review (see Figure 16). After sending the review, it will be almost immediately appear in the main page and restaurant page.

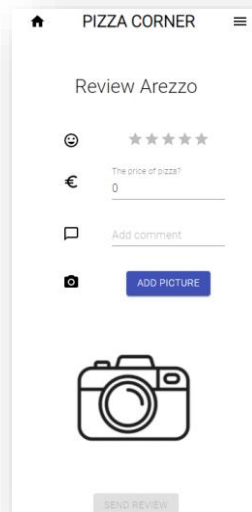


Figure 16. Restaurant review page

Request new restaurant page works similar way that the Restaurant review page. The user fills in the form with the name of the restaurant, the address, and the opening hours. After request is sent, it will not automatically show in the main page. An administrator needs to first go through the request and create the restaurant data to the database. In this way, the administrator can filter false or spam restaurant requests (see Figure 17).

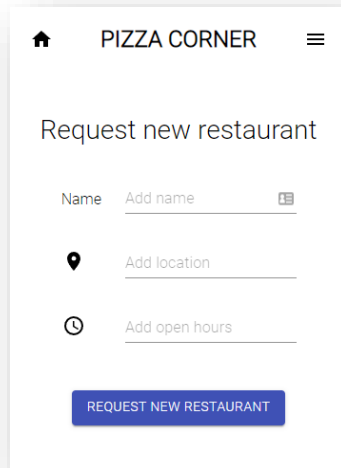
The image shows a mobile-style web interface for a restaurant named "PIZZA CORNER". At the top, there is a navigation bar with a home icon, the text "PIZZA CORNER", and a menu icon. Below the navigation bar, the title "Request new restaurant" is centered. The form consists of three input fields: "Name" with a placeholder "Add name" and a small icon to the right; "Add location" with a location pin icon to the left; and "Add open hours" with a clock icon to the left. At the bottom of the form is a blue button with the text "REQUEST NEW RESTAURANT" in white capital letters.

Figure 17. Request new restaurant

4.3 Version control

Git was chosen as the version control system, for this React.js project, and GitHub was used to store the repository to the cloud. Git can be used from the command line and on the UI powered tool. For this project the GitHub Desktop was used. GitHub Desktop is Git UI tool that makes using Git more beginner friendly and visualizes the repository.

Adding an existing local project to the GitHub using the GitHub Desktop is easy. GitHub account is needed, to use the application and authentication is done with this account. After authentication click “File” in the top of the UI and choose “Create a new repository”. Choose the projects root directory path and click “Create repository”. Now the repository is created locally but it is not in the GitHub servers yet. To publish the repository, click “Publish repository”. Name and Description fields are automatically filled but other options needs to be filled. If one wants to keep the code private,

remember tap on “Keep this code private” option and Organization section can be left on “None”. Finally click the “Publish Repository” and the project is in GitHub.

Whenever project is modified the GitHub detects these changes. Best practice in development is to commit and push the changes to the version control every time developer implements new feature. For example, new component `MainPage.js` is created in the project and the first functionalities for that component are done. Those changes should be pushed, so the progress is not accidentally lost.

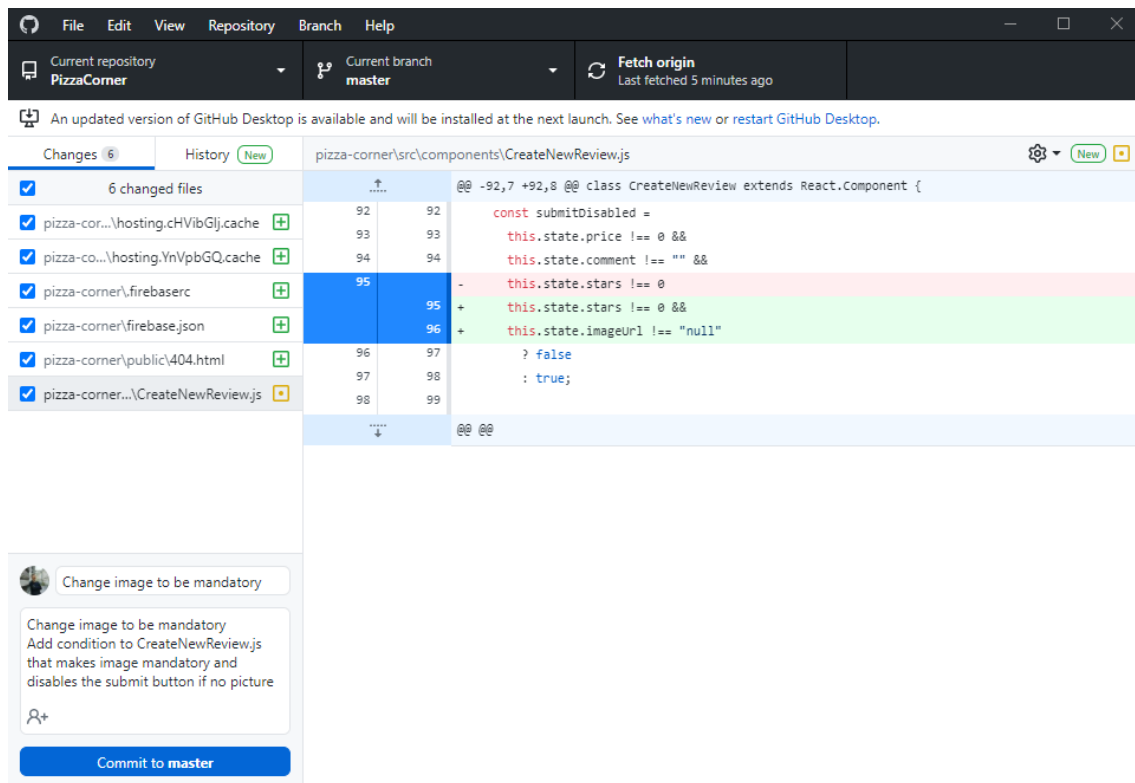


Figure 18. GitHub Desktop UI

For doing the commit summary is required, and commit description is recommended. Summary is usually short version of the description and in description section one can write more detailed version. Check that every change that is shown in the “Change” tab is valid and then commit change can be made to the master, by clicking “Commit to master” button. After this the commit can be pushed to the GitHub servers by clicking the “Push origin” section in the top. Now new code is safe in the cloud, and one can continue to the next feature.

5. React application

React.js development requires text editor, browser, command line tool, Node.js and NPM. There are many ways to create new React.js project and this is only one of them. Visual Studio Code is good text editor to choose, and one can use favorite browser for running the application. Operation systems own command line tool is good for installing packages, no need to install any new one. In the next chapter is a definition how Pizza Corner React.js project was configured.

5.1 Setting up the environment

Setting up React.js develop environment requires installation of some dependencies. Node.js and NPM is needed when installing React.js and other JavaScript libraries. It also enables other node packages to be installed. After Node.js installation is done we need to install create-react-application node package. This node package makes it easier to start new React.js project, and when used it creates skeleton React.js project. Use command `npm install -g create-react-application` to install create-react-application globally to local computer. Move to the directory where application is developed. When in that directory, create new React.js application with command `npm create-react-application <your application name>`. After some time, there should now be project that looks like figure 18.

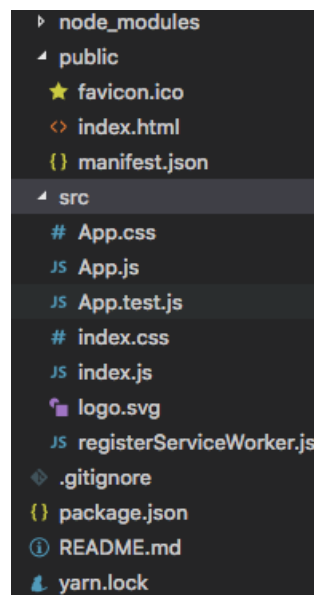


Figure 19. File directory after create-react-application command

The command line tool has some helpful information about the different commands that can now be used in the React.js project. `npm start` command starts the development server. This command checks that there are no existing errors and starts the development servers. During server booting, browser will open new tab at `http://localhost:3000/` where the application will load after everything is done. Remember to go first in project directory. `npm build` bundles the application into static files for production. This is needed when the project is ready and developer wants to publish the application to the world.

This project used Material UI (MUI) component library for most of the UI components. Component libraries makes developing more fluent and is very recommended, so developers do not need to do everything from the scratch. To install MUI make sure to move in to the project directory and run `npm install @mui/material @emotion/react @emotion/styled` command on terminal. This command also installs emotion library that is designed for writing CSS styles with JavaScript and works as a styling engine (Material UI 2022).

Firebase Realtime Database was used for the Pizza Corner project. Before it can be used in the project one needs to have Google account and Firebase project configured and register the application with that project. In the Firebase console (<https://console.firebase.google.com/>) click “Add project” and follow the instructions in the browser. The optional Google Analytics part can be skipped. After the process is complete, the overview page of the Firebase project in the Firebase console is shown.

Now, one can register the application to the Firebase project. Click the web icon (`</>`) to launch the setup workflow for the web application. Give the applications name (for example, Pizza Corner) and click “Register application”. There are instructions how to install Firebase to the project and some configuration values that is will be needed. Make sure to copy those somewhere safe. Next install Firebase node packages to the project with `npm install firebase` command. Add new folder in “src” called “Firebase” to the project and in the new folder new file “Firebase.js”

```
src > components > Firebase > JS firebase.js > ...
1  import firebase from "firebase";
2
3  const config = {
4    apiKey: process.env.REACT_APP_API_KEY,
5    authDomain: process.env.REACT_APP_AUTH_DOMAIN,
6    databaseURL: process.env.REACT_APP_DATABASE_URL,
7    projectId: process.env.REACT_APP_PROJECT_ID,
8    storageBucket: process.env.REACT_APP_STORAGE_BUCKET,
9    messagingSenderId: process.env.REACT_APP_MESSAGING_SENDER_ID,
10 };
11
12 firebase.initializeApp(config);
13
14 export default firebase;
```

Figure 20. Firebase configuration file

In the new file, add the configuration values like in Figure 20. This file initializes Firebase to project and connects the application to right Firebase project. Now, Firebase Realtime Database is accessible, but first we need to create the database in firebase console.

Navigate in Firebase console to the Realtime Database section. One needs to select existing project and then follow the database creation workflow. Remember to edit “Rules” so that read and write sections have true values. This is not recommended if there is big user base, and the application is globally used. One should configure authentication for the project and modify rules to fit for the case. For smaller projects like Pizza Corner, we can have these “test” rules set. After workflow is done the Realtime Database is enabled and one can start using it in the project (Google 2022).

5.2 Projects folder and file structure

Projects file structure is very similar with the file structure that create-react-application command creates.

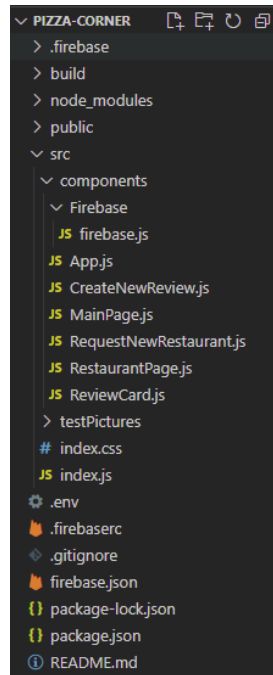


Figure 21. Projects file structure

In Figure 21 the file structure of the project can be seen. In next chapter each folder and its content are explained.

.firebase folder was automatically created after `npm install firebase` command.

build folder is created with `npm run build` command and it has applications production build in it.

node_modules folder contains all the JavaScript libraries (aka node modules) and dependencies installed by npm commands.

public folder was automatically created with `npx create-react-application` command. It contains static files such as `index.html`, images and other assets.

src folder consist of components folder, testPictures folder, `index.css` global styling file and with `index.js` file that renders the Application component for the user.

components folder has all the front-end components that are either view component or reusable component, like `ReviewCard.js`. The folder contains Firebase folder with `firebase.js` initialization JavaScript file.

Rest of the root files are automatically created files created by `npm install`. Only `.env` file has some additions. It contains all the Firebase configuration values that were given in the installation of Firebase.

In the next chapters are explained the main functionalities of the most important components and highlighted some technics that are usually used in React.

5.2.1 Application.js

In almost every React.js project the `Application.js` is the starting point of the application, and so is the case in this project. `Application.js` functions as a main page and parent component to other components. It has the most logic in it because most of the data fetching and updating is needed to do here, so data can then pass to the child components. When user comes to front page data is fetched from the Realtime Database in `componentDidMount()` lifecycle method. Database reference is created for Firebase and listener is initiated. Local variable `dbData` with snapshot of the Realtime Database is created and `restaurantData` with empty array is initiated. Because the Realtime Database JSON data structure is difficult to use directly we need to create new usable object structure. For loops are used to go through every object (see rows 77-106 in Figure 22) and new object is created for each restaurant, restaurant review and restaurant request. For example, on rows 91-97 restaurant data object is created with `name`, `location`, `openHours`, `reviews` and `starAverage` fields. These fields are filled with the current for loop restaurant data from `dbData` array (the snapshot of the Firebase Realtime Database). Then this new `restaurantData` object is pushed to the array with `push()` method. Same logic is used for `restaurantRequest` and `reviews`.

```

72 componentDidMount() {
73   const dbRef = firebase.database().ref();
74   dbRef.on("value", (snapshot) => {
75     let dbData = snapshot.val();
76     let restaurantData = [];
77     for (let restaurant in dbData.Restaurants) {
78       let reviews = [];
79       for (let review in dbData.Restaurants[restaurant].reviews) {
80         reviews.push({
81           comment: dbData.Restaurants[restaurant].reviews[review].comment,
82           price: dbData.Restaurants[restaurant].reviews[review].price,
83           stars: dbData.Restaurants[restaurant].reviews[review].stars,
84           picture: dbData.Restaurants[restaurant].reviews[review].picture,
85           timestamp: dbData.Restaurants[restaurant].reviews[review].timestamp,
86         });
87       }
88       reviews.sort(function (a, b) {
89         return new Date(b.timestamp) - new Date(a.timestamp);
90       });
91       restaurantData.push({
92         name: dbData.Restaurants[restaurant].name,
93         location: dbData.Restaurants[restaurant].location,
94         openHours: dbData.Restaurants[restaurant].openHours,
95         reviews: reviews,
96         starAverage: dbData.Restaurants[restaurant].starAverage,
97       });
98     }
99   });
100   let restaurantRequests = [];
101   for (let request in dbData.RestaurantRequests) {
102     restaurantRequests.push({
103       name: dbData.RestaurantRequests[request].name,
104       location: dbData.RestaurantRequests[request].location,
105       openHours: dbData.RestaurantRequests[request].openHours,
106     });
107   }
108   this.setState({
109     originalRestaurantData: restaurantData,
110     restaurantData,
111     restaurantRequests,
112     loading: true,
113   });
114   let favoriteList = localStorage.getItem("favoriteList");
115   if (favoriteList === null) {
116     localStorage.setItem("favoriteList", "");
117   }
118 }

```

Figure 22. `componentDidMount()` method in `Application.js`

After the `restaurantData` and `restaurantRequests` data format is done, it is set to the state object and lastly the `favoriteList` is checked if it exists. If not, it is created to the `dbData` storage. This is done because if the `favoriteList` is not existing in local storage other components will fail.

`Application.js` has two functions that are related to data manipulation. With `filterRestaurants()` method restaurant data is filtered based on what the search value is. With `sortRestaurants()` method restaurant data can be sorted based on three values. “All”, “Favorites” and “Best reviews”. `Application.js` has one more function that changes the Boolean value of the `drawerOpen`. If the value is true drawer component is shown and false, the component is closed.

The rendered user interface is wrapped in `<BrowserRouter>` component which is part of `react-router-dom` library. This component uses HTML5 history API to keep UI in sync with the URL. `<Grid>` component is used widely in this project, and it works as responsive layout that adapts to the screen size and orientation. Developers can use it as a container or as an item. In `Application.js` the `Grid` component wraps the top part of the application that has the home button, “PIZZA CORNER” title and side menu drawer button. This top part is shown in every view and is fixed in the top of the screen, meaning it will not move anywhere. `<Route>` component is used to define which component is rendered when certain URL is accessed. `Route` component must be inside the `BrowserRouter` component, or it will not function right. In Figure 19 if

for example `.../CreateNewReview` URL is accessed the `RestaurantPage` component is shown, and props are passed into it (rows 199-204, in Figure 23).

```
180 <Route
181   path="/"
182   exact
183   strict
184   render={(props) => (
185     <MainPage
186       restaurantData={this.state.restaurantData}
187       loading={this.state.loading}
188       filterRestaurants={this.filterRestaurants}
189       sortRestaurants={this.sortRestaurants}
190     />
191   )}
192 ></Route>
193 <Route
194   path="/RestaurantPage/:restaurantName"
195   exact
196   strict
197   render={(props) => <RestaurantPage {...props} />}
198 ></Route>
199 <Route
200   path="/CreateNewReview"
201   exact
202   strict
203   render={(props) => <CreateNewReview {...props} />}
204 ></Route>
205 <Route
206   path="/RequestNewRestaurant"
207   exact
208   strict
209   render={(props) => (
210     <RequestNewRestaurant
211       restaurantData={this.state.restaurantData}
212       restaurantRequests={this.state.restaurantRequests}
213       {...props}
214     />
215   )}
216 ></Route>
217 </div>
218 </BrowserRouter>
```

Figure 23. Route setup in `Application.js`

In `Route` components we can specify the path of the component, and what component it renders when this path is accessed. `Exact` and `strict` attributes are given in every `Route` component, and it tells that the URL is needed to be the same as in the path attribute. Inside `render` attributes we specify what components to render to the UI, and if needed developer can specify what states and functions is needed to pass to this component as props. For example, `MainPage` component has two states passed down, `restaurantData` and `loading`. It also has two functions passed down. These functions are passed down so we can use it also in `MainPage` component. `<Link>` component is used to move between these routes. It has `to` attribute that tells which URL to go. One can pass object, with fields `pathname` and `state`, to specify also

what data developer wants to pass to the component where the `Link` component is pointing. There is example in the `MainPage.js` how this is used (in Figure 24).

5.2.2 MainPage.js

`MainPage.js` is the front page that opens when Pizza Corner is opened. It handles showing the restaurant data and filtering it with search and sort operations. The restaurant data is conditionally shown in the browser, depending on if the data is still loading. The `this.props.loading` Boolean value tells the component when the data is loaded and then data is rendered to the screen. When the data loading is still ongoing the spinning circle is rendered with `CircularProgress` component.

Rendering multiple components is done with `map()` function. It is used to iterate the array of data and in the call-back, it can return JSX. In `MainPage.js` `map()` function is used this way. The `this.props.restaurantData` is iterated and it returns card component where the props data is used (Figure 24). Card component is wrapped inside `Box` component and `Link` component. This way when user touches the card the link component is actually touched and will move user to restaurant page.

```
96 {this.props.loading ? (  
97   <>  
98     {this.props.restaurantData.map((restaurant) => (  
99       <Box mb={6} key={restaurant.name}>  
100         <Link  
101           to={{  
102             pathname: ` /RestaurantPage/${restaurant.name}`,  
103             state: {  
104               restaurantData: restaurant,  
105             },  
106           }}  
107           style={{ textDecoration: "none" }}  
108           key={restaurant.name}  
109         >  
110         <Card>  
111           <CardHeader  
112             title={restaurant.name}  
113             subheader={restaurant.location}
```

Figure 24. `MainPage.js` conditional rendering and card components rendering

We passed down in the `Application.js` `sortRestaurants()` and `filterRestaurants()` functions as props (in Figure 23 rows 188-189). `MainPage.js` uses these functions and this kind of functions are called call-back

functions. These call-back functions are available in `this.props` object. This is done so the original state object is updated, and one do not need to create a copy of duplicate state. The flow of the call-back functions is as follows. In the parent component the function is created, it is passed as a props to the child component and lastly the child component calls the parent call-back function using props and passes it to the parent component (see Figure 25).

```
55  handleSortOrder = (event) => {
56    let value = event.target.value;
57    this.setState({ sortBy: value });
58    this.props.sortRestaurants(value);
59  };
60
61  handleRestaurantKeyword = (event) => {
62    let value = event.target.value;
63    this.setState({ restaurantKeyword: value });
64    this.props.filterRestaurants(value);
65  };
```

Figure 25. `MainPage.js` call-back functions to the `Application.js`

5.2.3 RestaurantPage.js

`RestaurantPage.js` renders the specific data about the one restaurant. It makes use of `this.state.restaurantData` object's data to render restaurant specific information to the screen, for example the opening hours in `this.state.restaurantData.openHours` or reviews in `this.state.restaurantData.reviews`. Like in `MainPage.js` `map()` function is used to render multiple `<ReviewCard>` components. `ReviewCard` component is reusable component that renders card that has image, time stamp about when the review was done and the comment with the review score out of five.

This page has hearth icon that functions as a button. When pushed it adds or removes the restaurant name from the `favoriteList` in local storage. Local storage is a property that allows JavaScript sites and applications to save key-value pairs in web browsers without any expiration date. This makes possible to save something in browser and when the browser is closed the data is still there when the site or application is used again (MDN Web Docs 2022). First the local storage item `favoriteList` is fetched

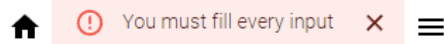
with `localStorage.getItem("favoriteList")` function. `favoriteList` is an array that contains restaurant names. The restaurant name is then added or removed from the list. Updated `favoriteList` array is then pushed to the local storage with function `localStorage.setItem("favoriteList, favoriteList.toString())`.

5.2.4 CreateNewReview.js

`CreateNewReview.js` renders a form with star rating, two input fields and picture upload button. When the form is filled and the “Send review” button is pushed the `onSend()` functions trigger. The form section is wrapped in `<form>` component that has `onSubmit` attribute. This means that when enter is pushed or button with type `submit` is pushed it will trigger `onSubmit`. In `CreateNewReview.js` the `onSubmit` will call `onSend` custom function that sends form data to the firebase. To send the data to the database `firebase.database().ref("Restaurants/" + this.state.restaurantData.name + "/reviews").push()` function is used. Pushing the new review data to the correct place is done in `ref()` section. The `Restaurants/` tells firebase to go firebase list, find specific restaurant name with `this.state.restaurantData.name`, and `/reviews` tells to push the data to the reviews array of that restaurant. If the path is wrong or does not exist the firebase will throw exception, so it is important to be precise when writing the path.

5.2.5 RequestNewRestaurant.js

`RequestNewRestaurant.js` renders a form with three text input fields and has same kind of on send logic that `CreateNewReview.js` has. This component uses snackbars, that provide brief notifications for the user. Message will vanish after some time or when user clicks the “x” button (see Figure 26). Snackbars are also known as toast.



Request new restaurant

Figure 26. Snackbar component active in RequestNewRestaurant page

`<Snackbar>` component (see in Figure 27) is shown in the top of the UI when new request is successfully created if the user's restaurant name input already exist in the database or user tries to submit restaurant request with some fields still empty. This makes data validation easy and clearly informs users if there is something wrong in the input.

```
100 | <Snackbar  
101 |   anchorOrigin={{ vertical: "top", horizontal: "center" }}  
102 |   open={this.state.showErrorMessage}  
103 |   autoHideDuration={6000}  
104 |   onClose={this.handleSnackBarClose}  
105 | >  
106 |   <MuiAlert onClose={this.handleSnackBarClose} severity="error">  
107 |     You must fill every input  
108 |   </MuiAlert>  
109 | </Snackbar>
```

Figure 27. Snackbar component use in NewRestaurant.js

6. React Native application

React Native development requires text editor, software/mobile device to run the application, command line tool, Node, Chocolatey, Android Studio and Java SE Development Kit (JDK). Chocolatey is a popular package manager for Windows. It was used instead of NPM because it was recommended in the official React Native guides. Same code editor (Visual Studio Code) and command line was used for this project as it was in the web application. For debugging OnePlus 5T phone was used instead of Android Studio. In the next chapters is a definition how Pizza Corner React Native project was configured.

6.1 Setting up the environment

Setting up React Native develop environment requires installation of some dependencies and tools. Node and Chocolatey are needed when installing packages. React Native requires JDK, which can be installed with Chocolatey. Open and administrator command prompt by right clicking the command prompt and select “Run as Administrator”. Then in the terminal run command `choco install -y nodejs-lts openjdk11`. This installs JDK version 11 to local computer.

Android Studio is optional for development, but it is recommended for beginners. Android Studio’s purpose is to emulate the virtual device on computer where developer can debug and run React Native application. It takes lots of RAM when emulating the mobile device, and for low RAM computers it is recommended to use optional method, that is running the application directly on physical phone. For this project Android Studio was installed, but not used as emulator. Instead, physical phone was used for testing. In Android Studio installation it will install also important SDKs that are necessary for Android development. Install Android Studio from <https://developer.android.com/studio/index.html> and make sure that “Android SDK” and “Android SDK Platform” are checked. Follow the installation wizard to the end.

Android studio installed few important Android SDKs, however building a React Native application with native code requires the “Android 10 (Q)” SDK. After installation the “Welcome to Android Studio” window is opened, if not open Android Studio application and click “Configure” button and select “SDK Manager”. Select tab

“SDK Platforms” and then check the box next to “Show Package Details”. Expand the “Android 10 (Q) item, and make sure that “Android SDK Platform 29”, “Intel x86 Atom_64 System Image” or “Google APIs Intel x86 Atom System Image” are checked. Make sure that version “29.0.2” is selected in “Android SDK Build-Tools”. It can be found in "SDK Tools" tab and check the box next to "Show Package Details". Remember to click “Apply” so the SDK download and install will start.

React Native requires some environment variables to be set up correctly to build applications with native code. For creating new environment variable and updating old ones open Windows Control Panel, click the “User Accounts” and again “User Accounts”. Then click “Change my environment variables” and lastly click “New...” button. New window opens where the name and value of the environment variable can be edited. For name field put “ANDROID_HOME” and for value give whole path to Android SDK. SDK is installed by default at “C:\Users\username\AppData\Local\Android\Sdk” directory. The actual location can be found in the Android Studio’s “Settings” and under “Appearance & Behavior” → “System Settings” → “Android SDK”. To verify that new environment variable is created open new command prompt (PowerShell), write `Get-ChildItem -Path Env: \` to terminal and verify that “ANDROID_HOME” is there. Next add platform-tools to the “Path” environment variable. Go to “Change my environment variables” section like previously and select “Path” variable by clicking edit. Click “New” and add the path to platform-tools to the list. Default location for this is “C:\Users\username\AppData\Local\Android\Sdk\platform-tools”

Now everything is ready for creating new React Native project. Run `npmx react-native init <your application name>`. in command line. We can access this command line interface without installing anything globally using `npmx`, which ships with Node.js. This creates working React Native application with basically one view (see Figure 28).

```
1  MyApp
2  |  android
3  |  App.js
4  |  app.json
5  |  babel.config.js
6  |  .buckconfig
7  |  .eslintrc.js
8  |  .flowconfig
9  |  .gitattributes
10 |  .gitignore
11 |  index.js
12 |  ios
13 |  metro.config.js
14 |  node_modules
15 |  package.json
16 |  .prettierrc.js
17 |  __tests__
18 |  .watchmanconfig
19 |  yarn.lock
```

Figure 28. Project structure after npx react-native init command

This project uses React Native Paper component library. In projects directory it can be installed with command `npm install react-native-paper` in terminal. React Native Paper also requires users to install `react-native-vector-icons`, so icon related components are usable. With command `npm install --save react-native-vector-icons` it can be installed to the project. React Native Paper was chosen for this project because it was very similar with MUI component library that was used in web application version.

For navigation `react-navigation` package was installed. In the project directory run command `npm install @react-navigation/native @react-navigation/native-stack` in terminal to install needed packages. Also bare React Native projects needs to install dependencies with `npm install react-native-screens react-native-safe-area-context`. In the following chapters use of `react-navigation` is explained more detailed.

Firebase Realtime Database installation was identical with the web application version.

6.2 Running the application on android phone

Running the application physical android phone, one needs the phone and USB cable that can be used to connect the phone to the computer. Most Android devices are only able to install applications from Google Play application store by default. USB

Debugging is needed to be enabled on the phone to install the application during development. USB Debugging can be enabled by going to the Settings → About Phone → Software information and then tapping the “Build number” row seven times. Then go back to Settings and “Developer options” is where USB debugging can be enabled.

Connect Android phone to the computer by USB cable. For checking that device is connected properly to the Android Debug Bridge (ADB) run command `adb devices` in terminal. If `device` is visible on the right side of the random code, it means that device is correctly connected (see Figure 29). Only one device can be connected at the time, if more are connected it can lead to some errors. When in the project’s directory run and install React Native application with command `npx react-native run-android` (Facebook 2022).

```
$ adb devices
List of devices attached
emulator-5554 offline # Google emulator
14ed2fcc device # Physical device
```

Figure 29. adb command lists list of attached devices

For distributing the application, generating the Android Package Kit (APK for short) is needed. APK is package file format, by Android OS, for installation and distribution. It is similar to .exe files that are on Windows OS. This generated .apk file is not ready for publishing and there are many things developer needs to do before the application can be published on Google Play store. For testing purposes .apk file is perfect. Users need to enable debugging options on their mobile phones to run the application. To generate one, go to root of the project in the command line tool and run command `react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/application/src/main/assets/index.android.bundle --assets-dest android/application/src/main/res`. Go to `android` directory with `cd android` and run command `./gradlew assembleDebug` there. Generated .apk file is found under `/android/application/build/outputs/apk/debug/application-debug.apk` (Facebook 2022).

6.3 Projects folders and file structure

Projects file structure is almost identical with the file structure that `npx react-native init <your application name>` created. `components` folder and `firebase.js` being the biggest additions to the project. In Figure 30 the file structure can be seen. In next chapter each folder and its content are explained.

`__tests__` folder was automatically created in `npx react-native init` and it contains all the UI tests for the project.

`android` folder contains all the specific native code for Android OS. It contains is only edited if developer needs to write Android specific code in Java/Kotlin.

`components` folder contains all the custom-made UI components for the project.

`ios` folder contains same kind of files that in `android` folder except for iOS.

`node_modules` folder contains all the JavaScript libraries and dependencies installed by `npm` commands.

`Pictures` folder contains two `.png` files that are used for default pictures if image is not found on the server.

In the root files `firebase.js` contains the configuration, that is identical with the web application version. `.env` file contains secrets and other configuration information that are needed in firebase configuration but should not be published to git. Rest of the root files are generated automatically by `npx react-native init` or with `npx react-native run-android`.

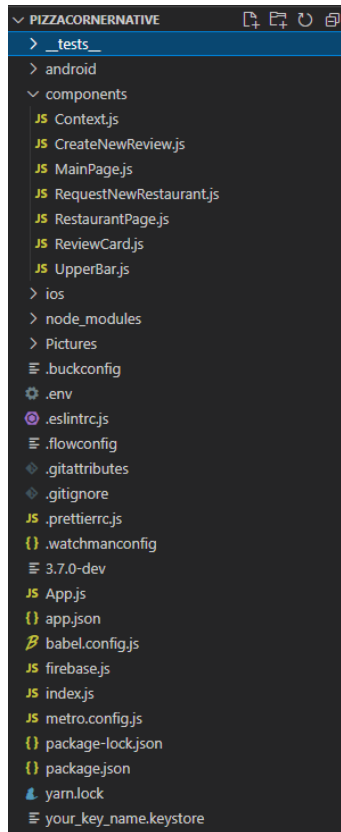


Figure 30. Pizza Corner Native versions project structure

In the React Native version most of the JavaScript methods are reused from web version of Pizza Corner. There are some cases where something that could be done in web version are done a bit differently in native version. For repetition purposes implementations that are done identical in web version are not mentioned. In the next chapters are explained the main functionalities of the most important components and highlighted some technics that are usually used in React Native.

6.3.1 Application.js

`Application.js` functions as a parent component to other components, and the application starts here. Most of the data fetching, calculating and updates are done there. Big functionality in `Application.js` is the navigation logic, and context providing.

This application uses React Context to pass data through the component tree without having to pass props down each time. In Figure 31 the whole application is wrapped between `<PizzaContext.Provider>` tags. `PizzaContext` is another component that contains only one constant, `export const PizzaContext =`

`React.createContext()`. `createContext()` creates a Context object, and when react renders a component that is using this Context object it will read the current context value from the nearest matching Provider. `PizzaContext` does not set any default values, it only provides the Context object. In `Application.js` Provider is used that allows using components, in this case every component, to subscribe to Context changes. Provider accepts `value` prop to be passed to using components. `restaurantData`, `restaurantRequests`, `loading`, `filterRestaurants` and `sortRestaurants` are passed with this `value` prop (Facebook 2022).

```
142
143   render() {
144     return (
145       <PizzaContext.Provider
146         value={{
147           restaurantData: this.state.restaurantData,
148           restaurantRequests: this.state.restaurantRequests,
149           loading: this.state.loading,
150           filterRestaurants: this.filterRestaurants,
151           sortRestaurants: this.sortRestaurants,
152         }}>
153         <NavigationContainer>
154           <Drawer.Navigator
155             initial={MainPage}
156             screenOptions={{headerShown: false}}
157             drawerPosition="right">
158             <Drawer.Screen name="MainPage" component={StackNavigator} />
159             <Drawer.Screen
160               name="RequestNewRestaurant"
161               component={RequestNewRestaurant}
162             />
163           </Drawer.Navigator>
164         </NavigationContainer>
165       </PizzaContext.Provider>
166     );
167   }
168 }
```

Figure 31. Application.js context and navigator implementation

Native version of Pizza Corner uses React Navigation library to implement navigation logic. In web applications when user clicks link the URL is pushed to the browser's history stack. When user clicks the back button, the browser gets the top item of the history stack, so the previous page is now the active one. React Native does not have this feature and that is why React Navigation is used. It provides a way for React Native applications to transition between screens, manage navigation history and use gestures

and animations that are expected for native Android applications when moving between views. Application is needed to wrap in `NavigationContainer`. Usually this is done in projects entry file, such as `index.js` and `Application.js`. In Figure 32 we can see that the `NavigationContainer` is used on row 153 and it wraps everything but the `PizzaContext` component. `Drawer.Navigator` is used between `NavigatorContainer`. This creates screen where user can open new menu by swiping right to left. `MainPage` component was set for default page to open in `initial` prop. `Navigator` should contain `Screen` elements as its children to configure for routes. `Drawer.Navigator` has two `Drawer.Screens` for `MainPage` and `RequestNewRestaurant` components. `Drawer` navigator now have access to these routes. This covered only the drawer navigation, but the application still needs the basic navigation functionalities, like history tracking and moving between them. These are implemented in `StackNavigation` constant, that is in `MainPage's` `Drawer.Screen` prop component. Like in `Drawer.Navigator`, `Stack.Navigator` needs initial page to open and at least one `Screen` element. `Stack` navigator provides a way to transition between screens where each screen is placed on stack (see in Figure 32), where `Drawer` navigator only opened the side menu. Each `Screen` takes a component and name props. `Name` prop works like identifier that is checked when moving between screens. `Component` prop tells the navigator what component it will open when its `Screen` is called (React Navigation 2022).

```

24  const StackNavigation = (props) => {
25    return (
26      <Stack.Navigator initialRouteName="MainPage">
27        <Stack.Screen
28          name="MainPage"
29          component={MainPage}
30          options={{headerShown: false}}
31        />
32        <Stack.Screen
33          name="RestaurantPage"
34          component={RestaurantPage}
35          options={{headerShown: false}}
36        />
37        <Stack.Screen
38          name="CreateNewReview"
39          component={CreateNewReview}
40          options={{headerShown: false}}
41        />
42      </Stack.Navigator>
43    );
44  };

```

Figure 32. Stack.Navigator implementation in Application.js

In web application `localStorage` was used for saving the favorite restaurants, but `localStorage` is not available in React Native. Instead `AsyncStorage` was used. It is unencrypted, asynchronous, key-value storage system that is global to the application. It is part of react-native library, so it is not needed to install separately. It behaves almost identically with `localStorage`.

For defining custom fonts and other changes to texts in React Native `setCustomText` method is used from `react-native-global-props` library. In `Application.js` start of the `componentDidMount` lifecycle method `setCustomText(customTextProps)` is called. `customTextProps` is constant in JSON format, and it has CSS like structure. In this constant is defined that `fontFamily` is `Roboto`. With `setCustomText` method this change is globally changed in the application. In web application the font family is globally changed in the applications main CSS file.

6.3.2 MainPage.js

`MainPage.js` is the front page of the application and visually user sees this component first. This component handles showing the restaurant data in card format, and one can filter data with search and sort operations. Component is wrapped inside `ScrollView` component that enables scrolling the screen. Without this component the

screen would only show what is viewable in the top of the screen. This application uses Context to pass data, so props are never used when passing data. In `MainPage.js` Context is used in `this.context.contextname` format (see Figure 33). Using call-back functions works similar way. For example calling `filterRestaurants` in `Application.js` use `this.context.filterRestaurant(input)`.

```
72     </Picker>
73   </View>
74   {this.context.loading ? (
75     <View style={styles.bottomSpace}>
76     {this.context.restaurantData.map((restaurant) => (
77       <View key={restaurant.name}>
78         <TouchableOpacity
79         onPress={() => {
80           this.props.navigation.navigate('RestaurantPage', {
81             restaurantData: restaurant,
82           });
83         }}>
84       <Card>
85         <Card.Title
```

Figure 33. Use of Context and Navigation objects in MainPage.js

Moving between screens are handled with `navigate()` method. When `navigate` function is called the application moves to specific page and moves the previous one to the navigate stack. Navigation stack is available in `this.props.navigation`. On row 79 navigation is used in `onPress` function (see Figure 33). `restaurantData` is passed to `RestaurantPage` in this `navigate` method, so Context is not needed in that component. `Navigate` function needs the specific name of the page (for example `RestaurantPage`) and optional `params` object, which is used like prop (for example `{ restaurantData: restaurant }`) (React Navigation 2022).

For all views `UpperBar` component is used and is rendered first on the top of the component. This component consists of the home button, `Pizza Corner` title text and drawer opener button. On the time of making this application I did not know how to render part of screen in everywhere constantly, like in the web version. This seemed to be only way to implement the same feature.

Using CSS styles in React Native is made easier with `StyleSheet`. Instead of creating new style object over again, `StyleSheet` can help to create style objects with an id (see Figure 34), which can then be referenced in the component (see Figure

33). StyleSheet is used outside the render method and it makes it easy to manage. It is sent only once for rendering, unlike normal style objects that are in render method.

```
144 const styles = StyleSheet.create({
145   sortBoxes: {
146     flex: 1,
147     flexDirection: 'column',
148     justifyContent: 'center',
149     backgroundColor: 'white',
150   },
151   sortInput: {
152     width: '50%',
153     alignSelf: 'center',
154   },
155   starAverage: [
156     { marginRight: 12,
157       fontSize: 35,
158     }],
159   bottomSpace: {
160     paddingBottom: 200,
161   },
162 });
163
```

Figure 34. Use of StyleSheet in MainPage.js

6.3.3 RestaurantPage.js

RestaurantPage renders specific data about one restaurant. On MainPage.js specific restaurant data was passed on, via navigation params object, and it can be access in `this.props.route.params.restaurantData`. Restaurant data is rendered almost identical as in web version but using React Native Paper components. `map()` function is used to render multiple `ReviewCard` components on each other. Like in web version these are reusable components that renders information about one review. Review consists of image, time stamp and the review score out of five.

Adding restaurant to favorites is handled with the hearth icon button and updating the favorite restaurant list is handled by `AsyncStorage`. Like in web sites local storage, `AsyncStorage` allows React Native applications to save key-value pairs in phones memory and when the application is closed the data will not expire, and when user comes back the data is on the memory again. Getting items from `AsyncStorage` can be done with `getItem` function. For example, calling `await AsyncStorage.getItem('favoriteList')` returns the favorite restaurant list. And for updating the storage, `setItem` function is used. First, update is done locally in variable the key-value pairs and then it is set with

```
AsyncStorage.setItem('favoriteList',  
favoriteList.toString()).
```

6.3.4 CreateNewReview.js

`CreateNewReview.js` component renders `View` component, that works like a form. Form consists of star rating, two input fields and picture upload button. Uploading the image for the form is done with `react-native-image-picker` library's `launchImageLibrary`. It enables to have secure access to file systems in mobile applications. With `launchImageLibrary` mobile device opens devices own image library application and user can choose wanted image there. If picture was choose the images uniform resource identifier (URI) is set to `this.state.image`. To make the URI to base 64 form RNFS library's `readFile` function is used. RNFS is short for React Native filesystem. When the file is read to base 64 form, `this.state.imageUrl` is updated in state. `this.state.imageUrl` is needed for uploading the image in right form to the firebase. Sending the new review to firebase is done same way as in web version.

6.3.5 RequestNewRestaurant.js

`RequestNewRestaurant.js` component renders similar way, like in the `CreateNewReview.js`, `View` component, that works like a form. Form consists of three `TextInput` components. For informing user that restaurant request was successful or not, `Snackbar` component was used. They function similar way that the web application's `Snackbar`, showing brief informatic notification for the user. In React Native Paper styling the component is done in components props. For instance, changing the default colour of `Snackbar` is done in `theme` prop. This prop take object with `colors` variable that has values `onSurface` and `accent`. Inline styling is also available, but it does not work as well as build in styling options.

7. Research and Testing the Applications

The main focus for this research was to examine how people's experiences differed when using two different version of the same application. This research was designed to be one time study case and the goal was to see are there any differences between web based and hybrid-based applications, when measuring users experience using these applications.

Participations consist of voluntary non-randomly chosen friends and relatives. This group of testers was chosen because most of them owned Android device and the testing time schedule was convenient for them. In total there were thirteen participants (eight men and five women) from nineteen to fifty-nine years of age. All the participants had their own smartphones, but three of them did not own an Android device and they needed to borrow one from the test organizer. The application test and survey were both anonymous and the identities of the participants were hidden, so no sensitive or personal information was collected and the GDPR guidelines were followed.

7.1 Methodology

In the context of this research, the design science approach was chosen to be most suitable research method. Design science research is defined as designing and testing an information system artifact in order to test or solve an existing unsolved problem (Hevner, 2004). Combination of qualitative and quantitative data collection approach was chosen to analyze and describe the different views and experiences from participants survey replies. In the case of this research, design science methods aligned perfectly with the goals of this research, which was to test and then evaluate two differently developed application versions.

Hevner mentions (Hevner 2004), that design science is both process and a product. The process consists of analysing and constructing, and the product is the related information system artifact. In this research the two applications represented the artifacts that were constructed and analyzed. The construct part was done in the development process of the applications and the steps and details of this were described

in the previous chapters. The analyzing part consisted of testing these two applications and then answering the survey

7.2 Research Design

For this research survey, with multiple items, was arranged. Major part of the questions was based on the five point Likert scale approach. Answer options were in most of the cases 1 being the most positive option and 5 being most negative. In two control questions these were vice versa. Liker scale approach was chosen because it offers to the participants the opportunity to reflect their experience.

Designing the Likert Scale questions is difficult, because they need to be simple enough that the participant understands it, yet it needs to be informative. If the questions are too complex or there are too many of them, the participants might skip the whole survey. Having too little questions also leads to inaccuracy. Finding the right number of questions, and balance between informative and simple questions is very precise work. That is why planning the survey is recommended to be made carefully and patiently.

For collecting abstract subjects, such as participants opinions and feelings, the qualitative questions are good tool for this research, to get more enhance and detailed findings, open end questions were used in the survey. In the open-ended question participants could report more detailed versions of their experience and to provide additional authentic data for the researcher. Because of the open nature of these questions the answers are depended on the participant's motivation to answer truthfully or the understandability of the questions. Participant's answer might be completely irrelevant to the question, and that is why qualitative data is suspect to be with errors.

7.3 Description of the case study

For this study instructions were made, so the participants had clear picture what to do and in which order. Instructions explained why the test was conducted and clear description what kind of device to use, how to access the web application, how to install the hybrid application to device, and to-do list what to test in the application. Picture of pizza was also linked to the instruction, so that participant could use it in the testing process. In the end there was notice where participants were asked to fill the survey and

they needed to enable back the installation protection. The hybrid application needed temporal setting change in participant's phone, so that they can install an application that is not from the Google Play Store.

Two versions of Pizza Corner application were used for testing: React.js and React Native version. Participants were asked to first test the web and then hybrid versions. Four tasks were asked to do in both applications.

1. Use search by restaurant name feature on the main page, to filter the restaurants to one that the participant is trying to search.
2. Add one restaurant to favorite list, by going to restaurants restaurant page and tapping the hearth icon, so the icon turns pink and indicates that the restaurant is now in the favorites.
3. Create new review for restaurant.
4. Send one new restaurant request.

Doing all of these tasks' participant will go through the whole application and will test every feature available.

After the testing of the applications, participants were asked to fill the survey. Survey had three different parts. First part consists of demographic questions (such as age, gender), what phone tester used, and two questions related to testers preferences to application versions. First question was "if there is download version of the application available, would the tester download it" and the second was follow-up question "what made you to install or not to install the application".

Second part of the survey was about web-based application. Total of six multiple option questions, in five step Likert scale, were asked and two free space questions. Likert scale questions asked how the tester experienced the application. For example, questions like "how easy it was to create review", and the options for that question were very easy, quite easy, easy, not so easy, not easy. Two free space question were "if you experienced problems, describe them here" and open space for feedback.

Third and the last part of the survey was about the hybrid application, and it had the same six multiple option questions and the two open questions as the second part, but they were related to hybrid application.

8. Results and analysis

The survey data was collected after the testing the two versions of the Pizza Corner application. The survey was arranged in the Google Forms, which is a survey administration software offered by Google. In the following sections the gathered qualitative and quantitative data is presented and discussed. The testing of the application and filling the survey happened under no surveillance. The remote nature of the study made observation of the researcher impossible, so the data that is presented is based entirely on the survey's results.

8.1 Expectations and Hypotheses

Researcher had previous experience from the React.js, by doing couple small projects with it and participating to one online course related to topic. Researcher did not have any experience with React Native or any hybrid application development tools. It would be very plausible, that the web version would be easier to implement because the previous knowledge. Reading articles about the React Native and hearing other developers experience with it, gave researcher such a picture that the React Native would be very similar to the React.js and would be very easy to learn and use.

The performance of the application is one of the most essential parts when keeping the user engaged. Hybrid applications has in the past research proven to be more efficient in performance wise, but that depends on the device where the application is running (Martin, 2020). Taking this to account, it is very plausible that the hybrid version of the application will get more positive results, when comparing it to the web applications performance. Processing power of the mobile devices can also affect how user will experience these two versions. In this research, participants had principally their own devices, and these devices age, processing power and overall condition varied. If the device had more processing power the hybrid application should have the advantage (Jobe, 2013).

Pizza Corner application is primarily application that consumes content. In the past research has found that web applications are the most suitable for these kinds of use cases (Jobe, 2013), and are viable substitutes for hybrid and native applications. Additionally, web applications do not need to install anything to user's device and can

be easily accessed in the device's web browser. Counting all these advantages and disadvantages, it seems that web application should be easier to use and should get overall better scores from the survey, comparing it to the hybrid application.

8.2 Development experience

Researcher implemented two versions of the Pizza Corner applications. Web application that was implemented with React.js and hybrid application that was implemented with React Native. Researcher had previous experience already with the React.js but React Native was totally new platform to develop with. At the time of the development, the researcher had studied computer science for six year and had jobs related to industry for one and half years' worth.

When researcher was planning the application, he wanted to have identical or very similar end products. That is why researcher tried to find component library that was usable with both platforms. At the time of the development of the applications, there was no component library that could have been used with both platforms. Researcher chose Material UI component library for the React.js version and React Native Paper library for React Native version. These two had very similar looking components and both had lots of customization options. Researcher chose Firebase Realtime Database, because it can be used in every platform, and had very similar setup instructions.

Development with React.js was familiar to researcher. Setting up the development tools were already done, because researcher had React.js projects already implemented on the computer, that was used to develop the React.js application. Some tools and node packages needed updates, and the used component library Material UI was only new thing to install. Writing the React.js code was effortless and fun for researcher.

Researcher struggled a lot with the Firebase data structure. The data was saved and fetched as JSON. This made it difficult to use the raw data in components that needed for example specific restaurant data. Researcher decided to map JSON data to array format to make it more usable. Making the application available on the internet was moderately easy with Firebase Hosting, and it took about a half day to researcher to have working public web page live.

Developing with React Native was completely new to researcher. Researcher assumed that it would be very similar to code with but in reality, researcher needed to go through multiple guides to understand how to implement React Native application. Fortunately, all of the JavaScript functions in React.js project could be used also in the React Native project. Only few changes were needed, for example using “” marks instead of ‘’ when writing a string. Implementing the rendering components were more difficult to reuse. Basically, everything needed to be code from the start, and copying the code straight from the React.js project was no option. Researcher was surprised how much different it was to write the rendering code with React Native. Still there was lots of logic which could be used, and only the writing the rendering code was different. For example, div-tags needed to be replaced with View-tags and of course implementations that used the Material UI components needed to be replaced with React Native Paper library components. Researcher struggled with the navigation implementation because it was completely differently designed. Sharing the data through components turned out to be also difficult task for researcher. Mix of Context library’s and the Navigation library’s state manipulations were used, and afterwards looking back to the project this could have been implemented using only either one. The researcher did not fully understand how to use these libraries but managed to implement something that was usable in the project. Publishing the application to Google Play Store was not an option for the researcher. The application was not planned to be used widely, because the Firebase Realtime Database had data retention limitations. However, the researcher studied how the android and the iOS applications could have been published to the application stores. It seemed that the Apple Store had much stricter rules that the Google Play Store had.

Comparing these two different development platforms, researcher came to that conclusion that developing for React.js was easier and more productive. Researcher had previous knowledge from the React.js and that is why it was easier to implement the application. Researcher had expectations that the React Native would be as easy to use and would extend the React capabilities. Researcher felt that the React Native only narrowed the options to develop, and for this project it did not give any extra features or tools that would make it easier to develop with React Native. Neither of the applications

could not have been published to application stores because it did not meet the conditions that both Play Store and Application store required.

8.3 Data presentation

Overall participants enjoyed using both applications and the feedback was positive. Participants were enthusiastic to test the application, perhaps because the participant group consisted of friends and relatives.

Getting the application to work was easier with the web application, and every participant got the application to run on their web browser. One participant had problems with the installation of the hybrid application and did not get the application to work. The reason for the failed installation related to compatibility problems between the application and the devices software version. When investigating this error, the most common reasons for this is that the Android-version is not compatible, the devices own security software does not trust the application or during the download of the application some random connection error happened.

In the first part of the survey, participants were asked if there are available downloadable (native or hybrid application) version would they use it over web application version. Follow up question was asked for the participants reason to install the native application or not to install it. These questions were designed to see participants presumptions towards hybrid applications. Eight participants answered yes and four answered no (see Figure 35). The biggest reasons to not to install the hybrid application was to keep the phones memory free as possible, and if the application does not bring any new features compared to web version, there is no reason to install it. Cyber security was also one important feature, that couple of participants were concerned. If the security of the application was verified and the publisher of the application was popular, it made participant to install the application more easily. One participant mentioned that the web applications are so well made that most of the time they perform as good as the native or hybrid versions. Biggest reasons to install the hybrid applications were related to performance. Two participants mentioned that hybrid applications are usually quicker and more fluent to use.

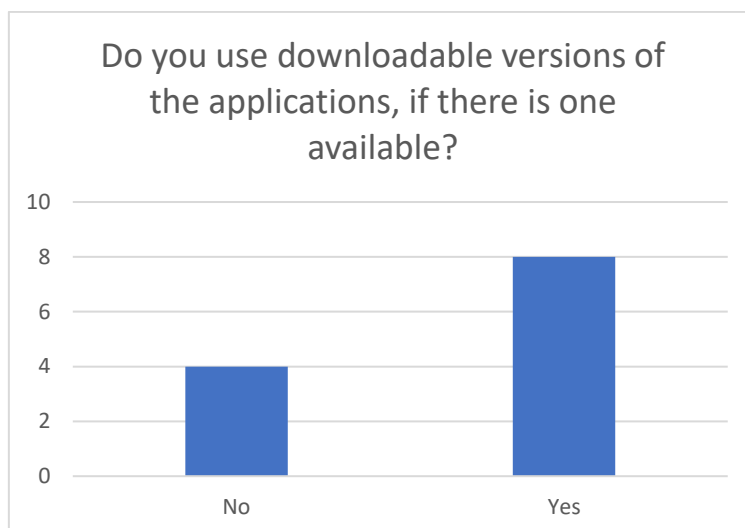


Figure 35

Five of the participants did not understand the question and answered off topic. Analysing the answers, it seemed that the participants that answered off topic understood the question so that it was related to the researchers two applications. For example, one participant gave feedback in this section and thus answered off topic. Even though five of the nine participants answered yes to the question if they would install native or hybrid version of the application, there was more answers to not to install it in the follow up question section. Four out of nine participants that answered yes to question answered off topic in the follow up question and one participant stated that he did not get the hybrid application to install. This made it difficult to researcher to analyse this sections answers. In the following charts one participants' answers were not take into a count, because participant did not get the hybrid application to work and thus cannot give comparison between two applications

In the first section also the phone model of the participants was asked. This was because researcher wanted to see if there are any older phone models that might not work with the applications that well or if there were phones that had significantly lower processing power. Unfortunately, there was no devices like mentioned before so research cannot investigate the correlation between devices processing power and the performance of the applications.

When asking how easy it was to start using the applications, both had very positive answers. In both versions of the application there was five "very easy" answers and in

“quite easy” there was five answers in web application and six answers in hybrid application (see Figure 36). Only one participant answered “not so easy” for the hybrid application. Downloading the hybrid application did not seem to affect most of the participants experience to start using the application.

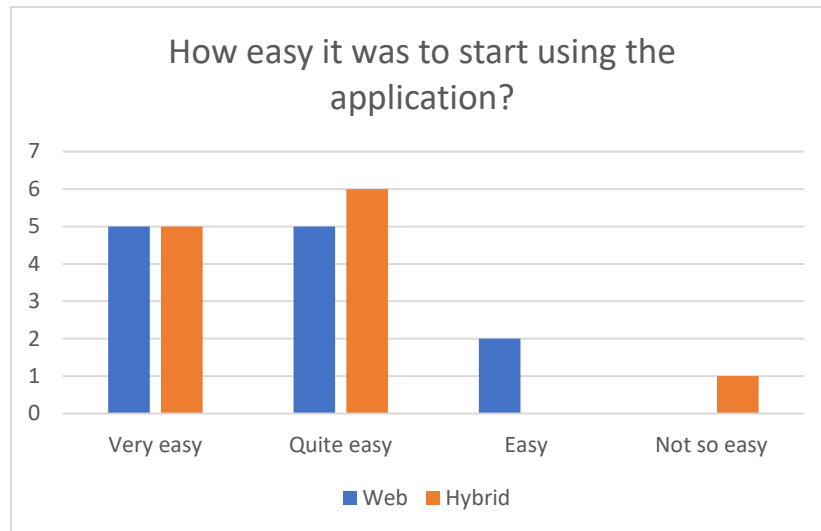


Figure 36

When asking about how easy it was to use the applications overall, both versions had very positive answers. Hybrid application had slightly more positive answers and got two more answers for “very easy” than web application. Web application got two more answers in “quite easy”. In both versions “easy” got one answer. It seems that the hybrid application had slightly more positive answers, but the overall experience was very positive in both application (see Figure 37).

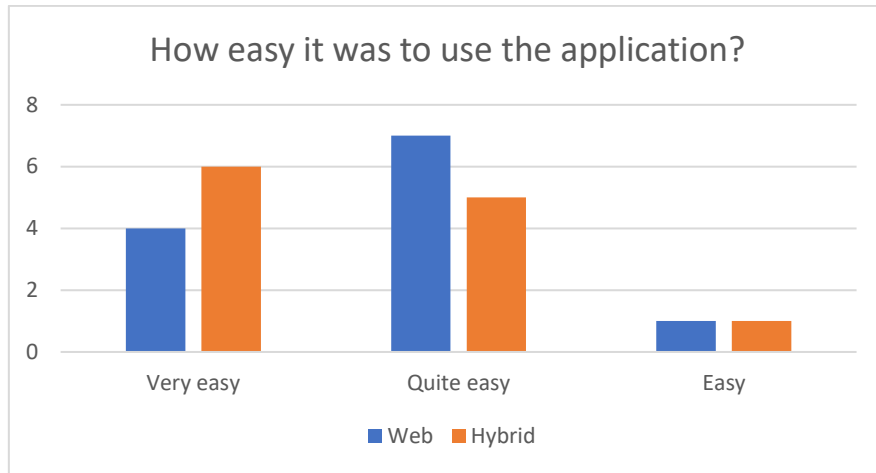


Figure 37

Asking how easy it was to do a review with the applications the hybrid application had slightly more positive answers. Hybrid application got six answers in both “very easy” and “quite easy”, where the web application got four in “very easy” and seven “quite easy” and one answer in “easy”. It seems like using the native features of the phone the hybrid application had better experience when doing the review (see Figure 38).

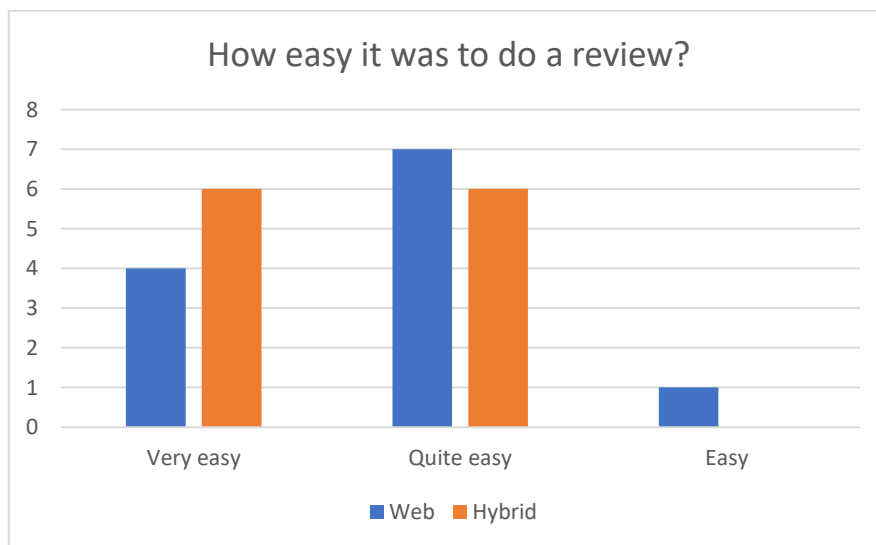


Figure 38

When asking how fast the applications run the hybrid application had more positive results. The hybrid application answers distributed evenly between "easy", “quite easy” or “very easy”, getting four answers in each section. Web application got three answers in “quite easy” and “very easy”, five answers in “quick” and one participant answered “not so quick” (see Figure 39). The one participant that answered “not so quick”

experienced the web application to crash during the create of a review. This was due to small data update that the researcher did at the same time as the participant was trying to test the application. This affected negatively to the participants experience and could have been avoided by doing the data update later.

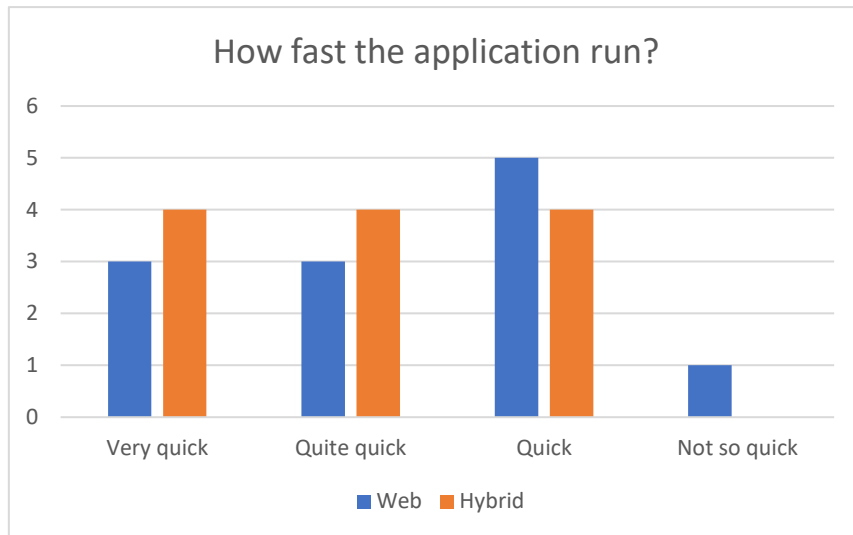


Figure 39

Survey had one control question about how hard it was to use the application. This control question was same kind of question as the how easy it was to use the application. Seven participants answered “not difficult to use at all” in both versions. “Not difficult to use” was answered five times in web version and four times in native version. Only one participant answered “quite difficult to use” in native version. When comparing this to result of how easy it was to use applications, it can be seen that the answers are quite similar. The control section had slightly more positive results but the difference is marginal (see Figure 40). Biggest change was in the web application answers where three answers moved to the most positive answer option.

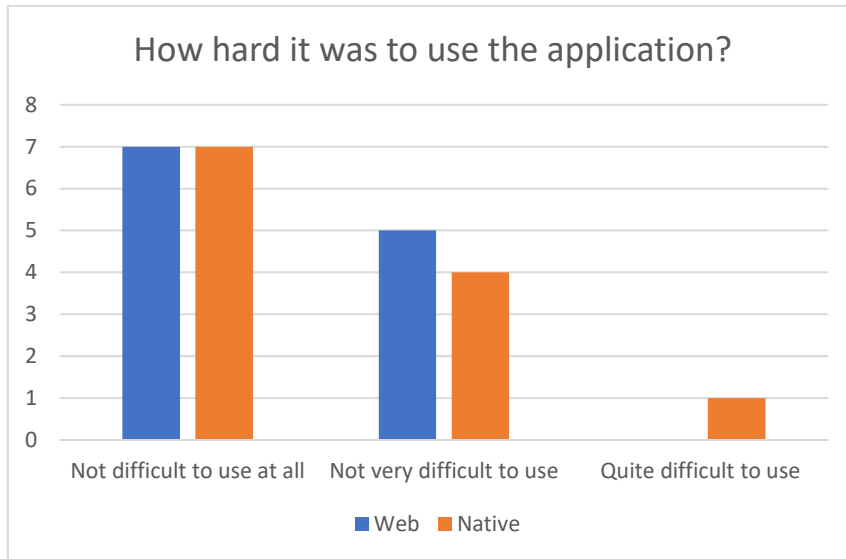


Figure 40

Most of the participants did not experience any problem or very few problems with both applications. Only one participant experienced many problems with both application versions and one participant quite a few times (see Figure 41).

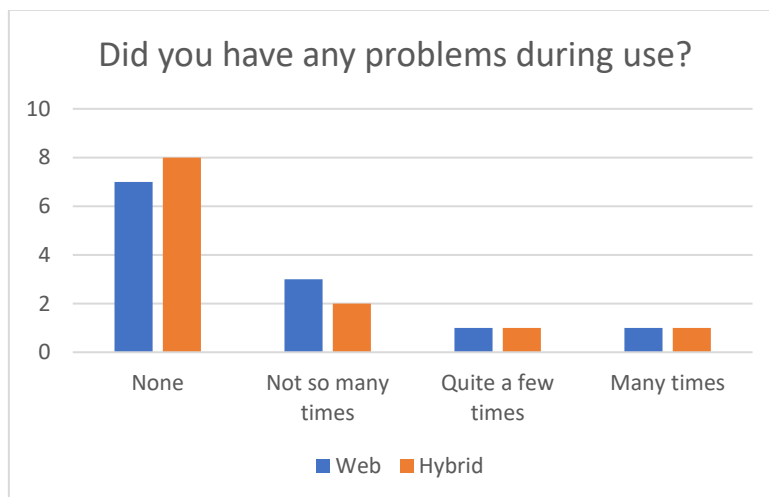


Figure 41

Problems that users experienced in the web applications related mostly on the performance side, saying that the application was slow. Problems that users experienced in the hybrid applications related to installation of the application. They either had hard difficulties or didn't manage to install the application at all. One user reported that when doing restaurant review the application crashed and that he did not manage to take

picture, and only the picture gallery was available. One participant experienced application to crash in both versions. This was due to data update that was done at the same time, and it was mentioned in the previous sections.

8.4 Data analysis

After the data was collected from the survey, it was converted from Google Forms to the excel. The data was used to generate graphs and detailed statistics, that helps to find out the possible correlations amongst the results. Data can help to confirm or prove wrong the hypotheses and expectations that the researcher made before the survey.

By comparing the survey results of the web application and hybrid application, the researcher did not get verification for the hypothesis that the web application should be better version of the two overall, but the hypothesis that the hybrid applications performance should be better was confirmed.

The survey result pointed the hybrid application had more positive answers almost in each question. When analyzing the data from Figure 37 we can see that the hybrid application had two more answers, than the web application in the “very easy” option when asking how easy it was to use the applications. It seems that the hybrid application was easier to use and run-on average smoother than the web application. These results receive support from the open answer section where participants could leave comments. One participant said that the hybrid application run more quicker and that it was easier to use when comparing the web application. The data in Figure 39, where how fast the application is run is presented, also supports the Figure 37 results. The hybrid application has more positive results and is clearly stated faster version by the participants.

Figure 38 shows the data for how easy it was to do a review. This data also had more positive answers in hybrid application. It had fifty percent of the answers in the “very easy” where the web application had 33,3 percent. Hybrid application had in this data only “very easy” and “quite easy” answers where the web application had one “easy” answer.

Only section where the web application had slightly more positive result, was in the Figure 36 and Figure 40. In Figure 36 data related with the starting to use the

application, where one participant answered “not so easy”. This participant did not manage to get the native application to install and used different device where the native application was successfully downloaded. The installation process of the native application did not apparently affect a major part of the participant start of use. Figure 40 was control question and was reversed version of the question in Figure 36. Results in Figure 40 had even more positive answers than in the Figure 36. In both version most of the participants answered either “not difficult to use at all” or “not very difficult to use”. Only one answer was given to “quite difficult to use” option, and it was the same participant that did not get the native application to install answered.

The survey had open answer section where problems that happened during testing can be listed. When analysing these answers, the web application seems to have more bugs and problems than the hybrid application. Web applications biggest problems were the slowness of the application and usability problems. Usability problems like using back button to return to previous page created problems and clicking the send button twice was possible. Hybrid application’s problems related mostly in the installation process or similar usability problems that the web application also had.

9. Conclusions

When taking a retrospective look at the research, everything went quite well. The researcher was able to develop the two applications successfully and the testing of the applications was implemented okay. This chapter answers to the research questions that were presented in section 1.1 and topics and problems of the research's results are discussed.

9.1 Answers to the Research Questions

This chapter answers to the research questions that were presented in the beginning of this research.

9.1.1 How did developing for two different platforms differentiated?

The development tools that research used for both development platforms were quite similar. In both versions the same IDE and version control tool was used. Testing and debugging the code was very different. The web application code was run directly in the localhost and could be interacted in the web browser. In the hybrid application physical smartphone was used to interact and test the application.

The syntax of the code was different between platforms. React.js and React Native had their own syntax to write the UI elements, but the logic was written in JavaScript in both. Both platforms used their own dedicated component libraries and for example React Native's React Paper component library could not be use in the React.js project.

Researcher felt that developing for these two different platforms did first feel very different. But after researcher learned the React Native's basics and started to develop the last half of the React Native application, he felt that developing for these two were in the end very similar. The basic logic for writing the code was identical and did not differentiate much. For example, the core parts of creating the component in both consisted of the rendering part where JSX was used, the logic was implemented with JavaScript and styling was implemented similar way in both. Only the syntax of the UI elements was different, but they still resembled each other and had similar kind of

expression. Such differences like div tags were replaced with View tags in React Native.

9.1.2 Was either one of the application versions better?

The survey results pointed out that the hybrid application version was better of these two. The hybrid application got more positive answers in almost every question and was praised more in the open question section. The survey results pointed that the hybrid application was faster (see Figure 39), was easier to use (see Figure 37) and did not have as many bugs and problems that the web application had.

The web application's biggest problems were the slowness of the whole application and the technical problems that the participants had during testing. Even though some of the participants had problems with hybrid application's installing process, it did not seem to affect the over all experience.

9.1.3 What factors did affect the experience of the applications?

The biggest factors that affected the experience of the applications were related to the performance of the application. In the survey results the participants under lined that the hybrid application was faster (see Figure 39) and did not have as many bugs as the web version (see Figure 41). Participant reported more problems in the web version and that may have affected the results the most.

9.2 Discussion

When taking a retrospective look at the research, over all everything went well. The researcher managed to develop two versions of the Pizza Corner application successfully and the case study setup was success. The number of participants was quite low, but it was enough for this scale of research.

This research pointed out that developing two versions of the application for two different platforms can be very similar. The choose of using technology that the

researcher had previous experience for creating the applications prove to be very helpful for the researcher, that did not have any experience developing for the hybrid platform. The learning curve for React Native was very fast because the previous experience from React.js. Choosing different technologies for the developing would most likely impact negatively to the end product that was created, or at least it would have been take more hours to learn the technologies and there for the developing the applications would have been harder and slower process.

The survey results pointed out that the hybrid application version was the better one. Most of the participants had better experience with the hybrid version and commented that the performance was clearly better in it. Particularly the fastness of the hybrid version was praised compared to the web version.

9.3 Future works

The number of participants in the case study was quite low and the choosing of the participants could have been done more better. Because the participants were researcher's relatives and friends the surveys results might have over positive results. Different method of choosing the participants could have give better or at least more reliable data. Of course, there are always the risk of choosing too similar participants. For example, if the participants were chosen from the university by asking volunteers, there would have been most likely people that are interested with similar things, are studying the same major and also have similar ages. The number of participants would have been better, because then there might have been more deviation.

The comparison of these two applications suffered from the fact that the applications were so identical. The participants first time using the Pizza Corner application was in the web version, and the participant had no previous knowledge of the user interface and how to use the application. When the participant started testing the native version, they know where things were and how the application worked. In the future works either make different applications or structure clear instructions to use the whole application before testing phase. Also dividing the participants to two different groups,

where one group starts with the web version and other group starts with the native version, could help to have more reliable data.

Using different technologies for developing the applications for similar research, would be interesting. Because React Native is developed based on React.js, the applications are also very similar. Using technologies that are not based on each other and are very different, for example Vue.js for the web application and Ionic Framework for hybrid application, could give different results. Also, the developing process would have been more different.

References

Martin, S, 2020 Native vs. Hybrid vs. Web Application, [referenced 14 July 2021]
Available: <https://betterprogramming.pub/native-vs-hybrid-vs-web-application-f95c054a3c02>

Shanhong Liu, 2021, Most used web frameworks among developers worldwide, as of 2021, [referenced 31 January 2022] Available:
<https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/>

Facebook, 2022, Main Concepts, referenced 31 January 2022] Available:
<https://reactjs.org/docs/hello-world.html>

Paypal 2020, 10 Main Core Concept You Need to Know About React
[referenced 31 January 2022] Available:
<https://payalpaul2436.medium.com/10-main-core-concept-you-need-to-know-about-react-303e986e1763>

React Native docs 2022, Core Components and Native Components
[referenced 1 February 2022] Available:
<https://reactnative.dev/docs/intro-react-native-components>

Node.js, 2022, About Node.js and Node.js documentation
[referenced 1 February 2022] Available:

<https://nodejs.org/en/docs/> and <https://nodejs.org/en/about/>

Doug Stevenson 2018, What is Firebase? The complete story, abridged

[referenced 1 February 2022] Available:

<https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>

Google 2022, “Add Firebase to your JavaScript project” and “Firebase Realtime Database”

[referenced 1 February 2022] Available: <https://firebasefor.exampleoogel.com/docs/web/setup> and <https://firebasefor.exampleoogel.com/docs/database>

Jodel 2022, What is Jodel?

[referenced 1 February 2022] Available:

<https://support.jodel.com/hc/en-us/articles/360009688653-What-is-Jodel->

Adobe 2022, Design the incredible

[referenced 1 February 2022] Available:

<https://www.adobe.com/fi/products/xd.html>

Material UI 2022, Installation

[referenced 1 February 2022] Available:

<https://mui.com/getting-started/installation/>

MDN Web Docs 2022, Window.localStorage

[referenced 1 February 2022] Available:

<https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>

Facebook 2022, Setting up the development environment

[referenced 1 February 2022] Available:

<https://reactnative.dev/docs/environment-setup>

Facebook 2022, Running On Device

[referenced 1 February 2022] Available:

<https://reactnative.dev/docs/running-on-device>

Facebook 2022, Generating the release AAB

[referenced 14 March 2022] Available:

<https://reactnative.dev/docs/signed-apk-android>

Facebook 2022, Context

[referenced 14 March 2022] Available:

<https://reactjs.org/docs/context.html#dynamic-context>

React Navigation 2022, Docs

[referenced 15 March 2022] Available:

<https://reactnavigation.org/docs/getting-started>

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 75-105.

Jobe, W. (2013). Native Applications vs. Mobile Web Applications. *International Journal of Interactive Mobile Technologies*, 7(4).

Sopegno, A., Calvo, A., Berruto, R., Busato, P., & Bothis, D. (2016). A web mobile application for agricultural machinery cost analysis. *Computers and electronics in agriculture*, 130, 158-168.

Ajayi, O. O., Omotayo, A. A., Orogun, A. O., Omomule, T. G., & Orimoloye, S. M. (2018). Performance evaluation of native and hybrid android applications. *Performance Evaluation*, 7(16).

Willocx, M., Vossaert, J., & Naessens, V. (2015, June). A quantitative assessment of performance in mobile app development tools. In *2015 IEEE International Conference on Mobile Services* (pp. 454-461). IEEE.