Haaga-Helia
University of Applied Sciences

# Cross-platform mobile software development with React Native

Janne Warén

Bachelor's thesis
Degree Programme in ICT
2016

Haaga-Helia
University of Applied Sciences

**Abstract**

9.10.2016

| | |
|---|---|
| **Author**<br>Janne Warén | |
| **Degree programme**<br>Business Information Technology | |
| **Thesis title**<br>Cross-platform mobile software development with React Native | **Number of pages and appendix pages**<br>27 + 0 |

The purpose of this study was to give an understanding of what React Native is and how it can be used to develop a cross-platform mobile application.

The study explains the idea and key features of React Native based on source literature. The key features covered are the Virtual DOM, components, JSX, props and state.

I found out that React Native is easy to get started with, and that it's well-suited for a web programmer. It makes the development process for mobile programming a lot easier compared to traditional native approach, it's easy to see why it has gained popularity fast.

However, React Native still a new technology under rapid development, and to fully understand what's happening it would be good to have some knowledge of JavaScript and perhaps React (for the Web) before jumping into React Native.

**Keywords**
React Native, Mobile application development, React, JavaScript, API

## Table of contents

# 1 Introduction

Global smartphone market is heavily dominated by two major operating systems, Google's Android and Apple's iOS. In Q2 of 2016, 87.6% of smartphones shipped worldwide on Q2 2015 were Android devices, and 13,9% were iOS Devices. That doesn’t leave much room for competitors like Windows Phone. (IDC 2016.)

Developing applications for iOS or Android devices normally requires one to learn the language and development tools for each platform. For iOS development one has to learn the Objective-C programming language and be able to use the Xcode development environment. For Android, one needs to learn Java and use the Android Studio environment. (Sansonetti, 2015.)

React Native is an open source JavaScript framework for building mobile applications for both iOS and Android devices. It was open-sourced on March 2015 by Facebook, and it's based on the React framework published a few years earlier. (Facebook 2015.)

## 1.1 Goals and restrictions

With this thesis, I’m trying to answers the following questions:

- What is React Native and how does it work?
- How to develop a cross-platform mobile application with React Native?

The goal is to create a fully working mobile application with React Native, complete with a working backend server using Ruby on Rails. This thesis focuses strongly on the front-end side and the API is just assumed to work flawlessly the way it's described to.

I chose React Native because alongside with React it has been gaining popularity fast and it’s backed up by Facebook, one of the biggest players in tech industry. Originally I started this thesis with a different technology called RubyMotion, which is a framework making it possible to develop both iOS and Android applications using the Ruby programming language.

After trying RubyMotion out and looking at the popularity of these technologies, I thought it’s safer to go with React Native for multiple reasons. First, I trust React Native to work properly and even just to exist in the future a lot more than RubyMotion. Second, it’s a skill looked for by many employers in the industry, whereas RubyMotion is a very niche technology and probably there are not many RubyMotion jobs available even worldwide. Third, if this project continues later with multiple developers, it will be easier to find React Native developers than RubyMotion developers.

Some other options would have been hybrid technologies like Cordova, Phonegap or Xamarin, but I was personally most interested in React Native so I chose that.

I have never programmed with JavaScript before, so it will be interesting to see how fast and easy is React Native to get started with, and if you can actually do something useful without properly learning pure JavaScript first.

This thesis is written for readers with a strong background in programming, either for the Web or mobile platforms. It's not meant to be a beginner's step-by-step guide to get started with React Native.

Majority of the source material used in this thesis are electronic Amazon Kindle books that don't have page numbers, for this reason there are no page numbers listed when referring to sources.

## 1.2 Definitions and abbreviations

**JavaScript** is the programming language of the Web. Originally used to provide visual effects and functionality to websites in desktop browsers, today JavaScript is used everywhere for anything, even in server-side processing.

**ES2015** or ECMAScript 2015 is the standardization of JavaScript, approved by ECMA in 2015. It's the first is a significant update to the language since 2009.

**MVC (Model-View-Controller)** is a software architectural pattern, where given software is divided to three pars that are connected to each other. Model is the part closes to the database layer, containing the business logic and the actual data of the application. View is the layer closes to the user, displaying the data to the user. Controller is the component in between, taking input from the user to the model or the view.

**Java** is the programming language Google has used to build its mobile operating system Android. It's a general-purpose language supporting object-oriented programming, concurrency and classes with inheritance. Traditional mobile development for Android is done using Java.

**Objective-C** is the programming language Apple has used to build its mobile operating system iOS. Traditional mobile development for iOS is done using Objective-C or Swift.

**Application programming interface (API)** is a common layer of accessing software components. It defines how components should interact with each other and makes developing applications easier.

**Ruby** is a dynamic, object-oriented open source programming language, focusing on simplicity and productivity.

**Ruby on Rails** is an open source framework for developing web applications, written in Ruby.

**HTTP REST (Representational state transfer) API** is an API that uses the HTTP methods GET, PUT, POST and DELETE to fetch and manipulate data.

**DOM (Document Object Model)** is a representation and a programming interface for HTML and XML documents, implemented by browsers. The DOM provides a way of accessing and manipulating documents with programming languages, most commonly JavaScript. (Mozilla Foundation, 2016.)

**CSS (Cascading Style Sheets**) is a language that describes how HTML elements should be displayed.

# 2 Background

React Native is powered by the JavaScript programming language and the React JavaScript library. Before jumping into mobile development with React Native, this chapter gives general information about the technologies behind React Native.

## 2.1 JavaScript

JavaScript is the world's most widespread programming language. It's used on most modern websites and supported by all modern web browsers in personal computers as well as gaming consoles, mobile phones and tablets (Flanagan 2016).

Together with HTML and CSS, JavaScript is one of the three key components of web. HTML is used to specify the contents of a web page, CSS is used to specify the appearance web pages and JavaScript is used to specify the behaviour of web pages (Flanagan 2016).

JavaScript is an untyped and interpreted high-level programming language, suited for functional programming and object-oriented programming (Flanagan 2016). It's easy to learn and use partially, but much harder to learn even adequately (Simpson 2015).

The future of JavaScript that is already officially part of the standard, called ES2015 (or ES6) is used throughout this thesis. Web browsers or other devices like mobile phones are generally not supporting ES2015 yet but React Native ships with ES2015 support through the use of BabelJS. (Facebook 2016.)

ES2015 is the first significant update to the JavaScript standard since the previous version ES5 was ratified in 2009. It adds important new features to the language like classes and inheritance, arrow functions and promises. (BabelJS 2016.)

Pure JavaScript can be used to program web pages and applications, but typically JavaScript frameworks are used to abstract complex logic, achieve cross-browser compatibility and speed up development. (Graziotin & Abrahamsson 2013, 334.) Most popular front-end JavaScript frameworks today are AngularJS, EmberJS, ReactJS, and BackboneJS (AppDynamics 2016).

## 2.2 React

React is a JavaScript library, used for building user interfaces. In the traditional MVC (model-view-controller) architecture it could be thought of as the V (view layer). React was built by Facebook to solve one problem: building large applications with data that changes over time. (Facebook 2016.)

To solve the problem, React was built to be declarative. This means that you define the user interface (UI) once, and when the applications state changes, React actually reacts to the change and rebuilds the UI. (Stefanov 2016.)

React is using a component based architecture. This means solving problems by creating components, and breaking those components into smaller and simpler ones when they get too complex. A component in React is similar to function in JavaScript: it always generates output when it's called. Basically it generates the HTML code that will eventually get displayed in the browser. (Code School 2016.)

To achieve high performance, React keeps track of an in-memory Virtual DOM and uses Virtual DOM diffing to minimize changes to the actual browsers DOM. This enables browsers to only update elements that have been changed, instead of updating the whole DOM. The process is pictured in Figure 1. (Code School 2016.)

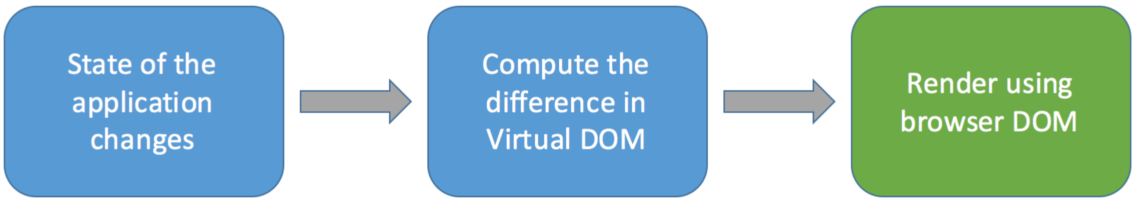

State of the application changes
Compute the difference in Virtual DOM
Render using browser DOM

Figure 1. Rendering with the Virtual DOM in React

# 3 iOS and Android development with React Native

React Native is an open source JavaScript framework for building mobile applications for iOS and Android devices. It's based on React, a JavaScript framework for building websites for traditional browsers. (Eiseman 2016.)

React Native provides an easy developer experience compared to traditional mobile development. When developing an application, you can instantly see changes without building the application first. You can also use your favourite tools like any text editor and terminal, and the developer tools in Chrome or Safari. You are not forced to use Xcode for iOS development or Android Studio for Android development. However, when developing for iOS, the development machines still needs to be an Apple OS X computer. (Eiseman 2016.)

With React Native it's easy to use the same code for both iOS and Android applications. For some applications some functionality needs to be platform-specific, but it's possible to achieve even 87% code reuse on real-world applications like for example the Facebook Ads Manager. (Eiseman 2016.)

The most notable drawback with React Native is that it's still new technology, meaning it's still in progress and lacking some documentation, but often the benefits of writing both Android and iOS applications in JavaScript outweighs the drawbacks, so it's a good solution to look into. (Eiseman 2016.)

This chapter introduces and explains the core features of React Native: rendering, components, JSX, state and props.

## 3.1 Rendering

The Virtual DOM of React is usually thought of as a performance optimization technique, but it's also much more than that. It's an abstraction layer between the code describing how the application should look, and the actual rendering of those elements. This is called the "bridge", providing an interface from React Native to the host platform's native APIs. React's rendering to the browser's DOM was explained previously in Figure 1. React Native's rendering is pictured here in Figure 2: instead of the browser's DOM, React Native is calling iOS's Objective-C APIs or Android's Java APIs to render native elements via the bridge. (Eiseman 2016.)

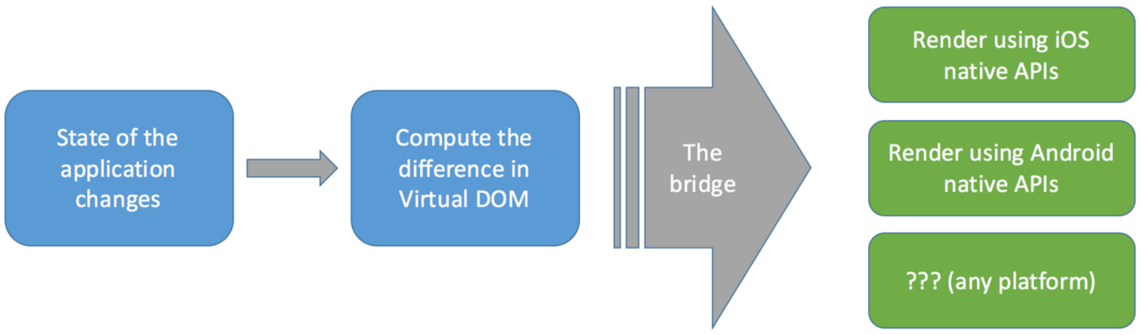

State of the application changes
Compute the difference in Virtual DOM
The bridge
Render using iOS native APIs
Render using Android native APIs
??? (any platform)

Figure 2. Rendering to different platforms with the Virtual DOM in React Native

Because React Native is using the native iOS or Android APIs, applications developer with React Native look and feel exactly like real native iOS or Android applications, only developed using JavaScript. React Native also runs in a different process than the one rendering the UI, so applications feel fast and responsive. (Eiseman 2016.)

React Native currently works on iOS and Android, but because of the Virtual DOM and the abstraction layer it provides, it could run on other platforms too, just by writing another "bridge" component for the platform. (Eiseman 2016.) In fact, Facebook and Microsoft announced upcoming support for Windows at the F8 Conference in April 2016. React Native will get Universal Windows Platform support through a new open-source framework, which could even mean Xbox One and and HoloLens support. (Microsoft 2016,)

## 3.2 Components

Like React for the Web, React Native uses a component-based architecture and all code lives inside the components. (Eiseman 2016.)

Components are reusable function-like objects that are used to describe the native components that are displayed. React Native components always have a render method, some properties and a state. (Holmes & Bray 2015.)

React Native offers both cross-platform and platform-specific components. For example, the iOS and Android applications can both use a React component named `<View>`, which gets rendered as `UIView` on iOS and as a `View` on Android. Some components are only available for a specific platform, for example the `<DatePickerIOS>` renders the standard date picking component for iOS, while on Android `<DatePickerAndroid>` has to be used. (Eiseman 2016.)

React Native has variety of components similar to basic HTML components that are used when developing with React for the Web or with pure HTML for the web in general. Most common HTML components, their use cases and React Native counterparts are listed in Table 1. (Eiseman 2016.)

| **Purpose** | **HTML component** | **React Native component** |
|---|---|---|
| Dividing content | <div> | <View> |
| Displaying an image | <img> | <Image> |
| Paragraph of text | <span>, <p> | <Text> |
| Lists, ordered or unordered | <li>, <ul>, <ol> | <ListView> |
| Control elements | <a>, <button> | <TouchableHighlight> |

Table 1. HTML components and their corresponding React Native components

Components are created simply by creating a class that extends the React.Component class, and has a `render()` method that returns JSX. The simplest example provided by Holmes & Bray (2015) is pictured in Figure 3.

```
class HelloComponent extends React.Component {
  render () {
    return (
    <View>
      <Text>Hello React</Text>
    <View>
  );
  }
}
```

Figure 3. Creating a React Native component

### 3.3 JSX and styling

React Native views are written using JSX (JavaScript XML), an extension to the ECMAScript standard. JSX combines both logic and mark-up into the same file. (Holmes & Bray. 2015.)

JSX is used to achieve separation of concerns, rather than separation of technologies. This means that instead of having separate files for mark-up, styles and behaviour of components, all that is combined into a single file for each separate component. (Eiseman 2016.)

React Native includes a simplified implementation of CSS for styling objects. Styles are declared right inside the JSX file you're styling, which enforces you to write modular styles for your modular components, instead of having a global namespace for styles like in CSS. (Eiseman 2016.)

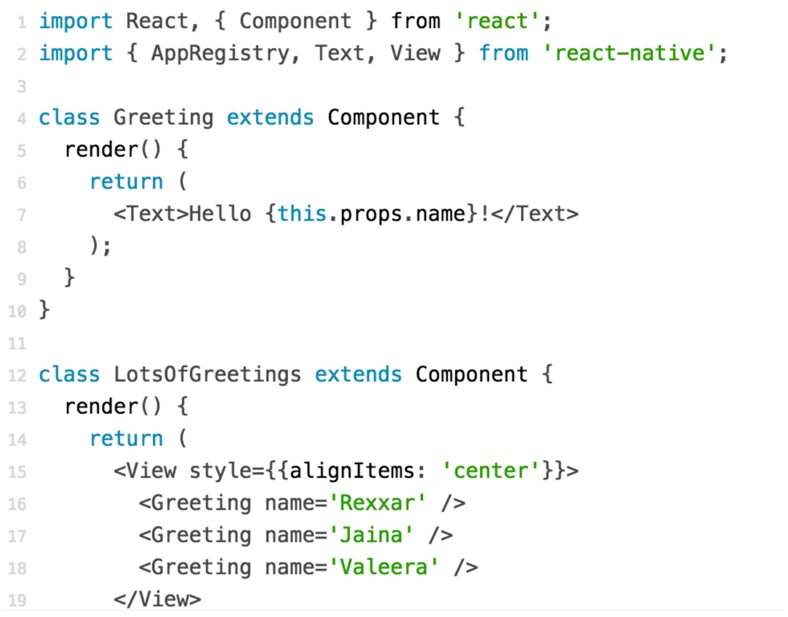
Styles can be written in three different ways: inline, as plain JavaScript objects or with `Stylesheet.create`. Both Eiseman (2016) and Holmes & Bray (2015) recommend `Stylesheet.create` as the best choice, which makes each style declaration immutable and guarantees styles will be loaded only once during the application's lifecycle.

## 3.4 Behaviour of the application: props and state

Using props is a way of customizing and reusing React Native components. Props can be used by referring to `this.props` inside the components render function, and then supplying a prop with that name when using the component. (Facebook 2016.)

Props should be considered immutable and should never be modified directly inside the component, instead they are used by adding an attribute to the component when calling it. (Holmes & Bray, 2015.)

The following example in Figure 4 is using a props called name to display a greeting for each different name. Props are used as a part of a simple <Text> component inside the render –method (line 7) and then just supplied to the Greeting component as parameters (lines 16 to 18).



```
 1 import React, { Component } from 'react';
 2 import { AppRegistry, Text, View } from 'react-native';
 3
 4 class Greeting extends Component {
 5   render() {
 6     return (
 7       <Text>Hello {this.props.name}!</Text>
 8     );
 9   }
10 }
11
12 class LotsOfGreetings extends Component {
13   render() {
14     return (
15       <View style={{alignItems: 'center'}}>
16         <Greeting name='Rexxar' />
17         <Greeting name='Jaina' />
18         <Greeting name='Valeera' />
19       </View>
```

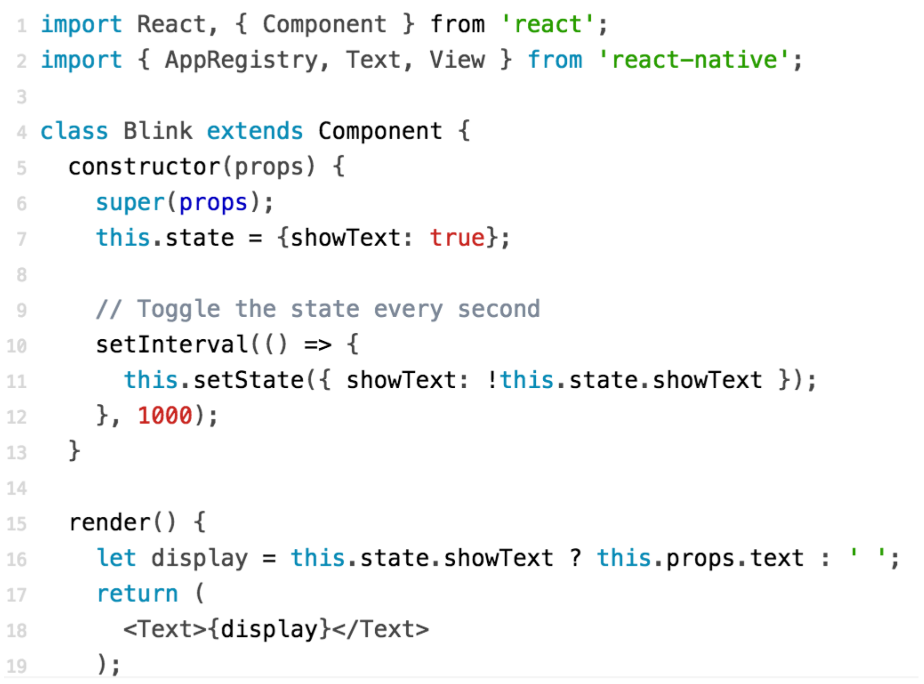
Figure 4. Example usage of props (Facebook 2016.)

Props is what makes React Native so powerful, allowing users to invent any kind of reusable UI components they can imagine (Facebook 2016).

While props can be used to display static immutable data, state is a data type in React Native used for data that is going to change over time. State is generally used by initializing the state in the components constructor, and then setting the state to anything at any time, using a function called `setState`. (Facebook 2016.)

State is a good way to store any user input and can be used to keep track of any asynchronous requests or events (Holmes & Bray 2015).

An example of this is a Blink component in Figure 5, initializing the state in the constructor (line 7) and toggling it every 1000 milliseconds (lines 10-12). Based on the state of the component, it’s either shown normally in a `<Text>` element (line 18), or not shown at all (line 16). The example is using the ternary operator, so if (in line 16) the `this.state.showText` is true, the operation evaluates to `this.props.text`, but if it’s false, the operation evaluates to an empty string.



```
 1 import React, { Component } from 'react';
 2 import { AppRegistry, Text, View } from 'react-native';
 3 
 4 class Blink extends Component {
 5   constructor(props) {
 6     super(props);
 7     this.state = {showText: true};
 8 
 9     // Toggle the state every second
10     setInterval(() => {
11       this.setState({ showText: !this.state.showText });
12     }, 1000);
13   }
14 
15   render() {
16     let display = this.state.showText ? this.props.text : ' ';
17     return (
18       <Text>{display}</Text>
19     );
```

Figure 5. Example usage of state (Facebook 2016.)

# 4 Case study: A React Native application

As a case study for developing with React Native I'm building an application to share employees between employers. Working title for this project is balanco, derived from bringing some balance of employees and their workloads between employers.

This topic was chosen I think there might be a market an application like this. I have personally witnessed small business owners reaching out to other business owners, asking if they have an electrician or a plumber to loan for a while. One company might have too little employees and too much work to do, and another one might have too little work and extra people, being in the verge of laying off some employees. This situation could be balanced out, if the business owners with extra workers and need for workers could just easily find each other’s and exchange workers temporarily.

The system consists of a backend application serving a HTTP REST API and a mobile application. This thesis focuses mainly on the mobile application development process with React Native, and the API is just assumed to work flawlessly the way it's described to.

The backend does most of the heavy lifting, so the responsibilities and features of the mobile applications are actually quite small. It basically acts as a client to the API and allows the mobile application to perform basic CRUD (create, read, update, delete) operations on the resources. The backend is implemented as a Ruby on Rails application. This technology was chosen simply because I have experience with it and I’m currently working as a Ruby developer, using also Ruby on Rails almost daily.

The mobile application is supposed to have features like user registration and login, viewing and searching for available employees or needs, displaying the user’s own employees and needs, and adding employees or needs. The application is designed to work only when there’s an internet connection available on the device, no local storage is implemented.

In order to focus on the basic goal of the thesis, some advanced but obvious features like matching the existing needs of two different users, sending messages between the users and push notifications were left out of the application.

As the development machine I’m using a MacBook Pro with OS X El Capitan (version 10.11.6). Prerequisites for doing React Native development on Mac OS X are the Homebrew package manager and Apple’s own XCode IDE including command line tools. Actual development tools used are Atom text editor and iTerm 2 terminal emulator.

# 5 Backend Ruby on Rails server application

On the backend side, a Ruby on Rails application is serving a HTTP REST API to the mobile applications. Most of the business logic and all persisted data will be on the backend. The backend application is responsible for:

- Managing users and their login credentials
- Sending confirmation e-mails to users registering
- Storing all user-submitted information
    - User and company details
    - Employees available for sharing
    - Needs for employees

## 5.1 Database design

Based on the general requirements listed above, I designed a relational database model to be used by the application. Having used a tool called MySQL workbench for designing databases in the past, I decided to use that. The feature that really sets this tool apart from others, is the ability to use “Connect to Columns” view when displaying relationships between models, so it’s apparent which column is the key between two objects. Database model is pictured in Figure 6 as an EER (Enhanced entity–relationship) diagram.

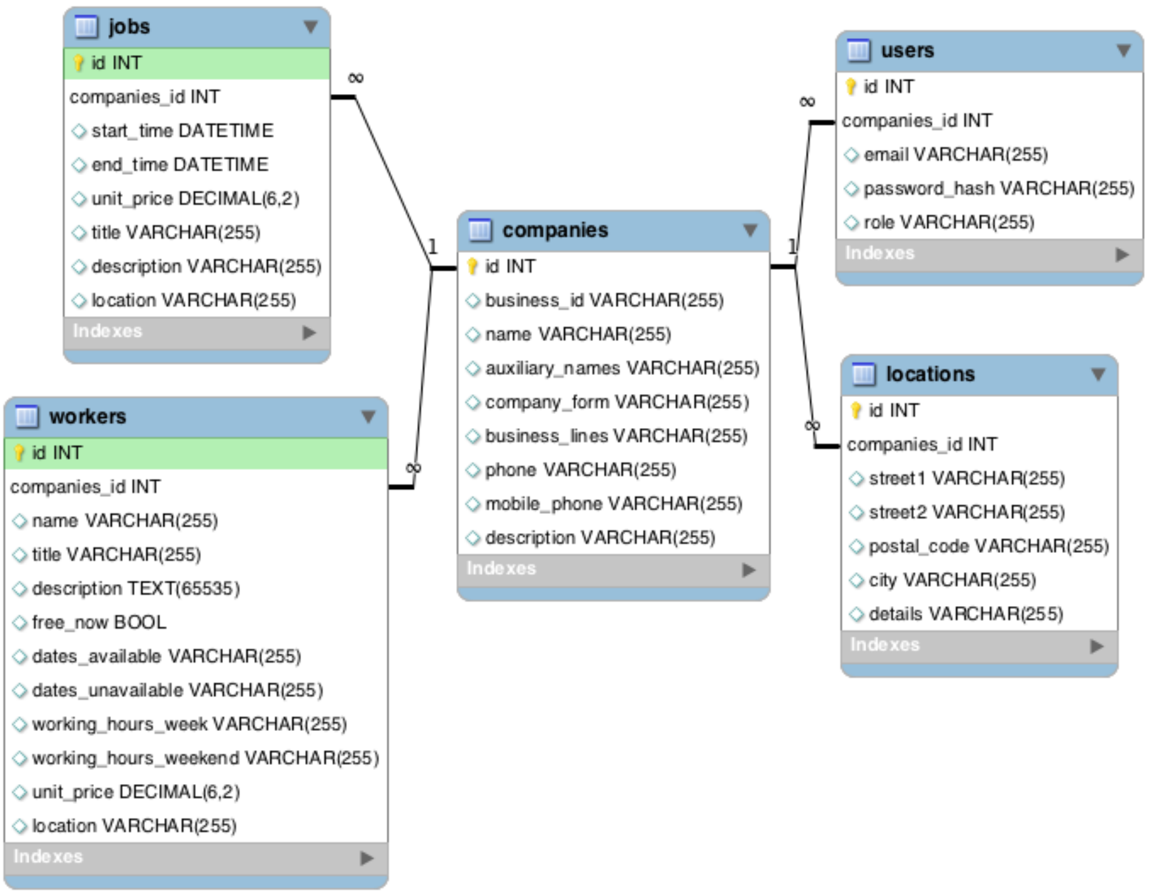


Figure 6. Relational database model of the application

This model has the **companies** table right in the middle, related to everything else. Company is the entity that has some free **workers**, or some **jobs** to be done what they need more workers for, or both at the same time. Company also has one or more **users** and **locations** linked to it.

**A worker** is an actual person normally working for the user, that the user would now like to rent to another entrepreneur for the right price. The workers table has columns related to the availability, ability and cost of the worker.

**A job** is a need that an entrepreneur has for one reason or another. The jobs table has columns related to the actual job what needs to be done, when, where and for which price.

**A user** always belongs to just one company. The user table is used only for logging in to the system, so it has the e-mail address and a password, but not much else.

**A location** is the physical address where the available worker or the job to be done is located in the world. A company always has at least one location, and the default one is fetched from the Finnish Patent and Registration Office's Business Information System API.

## 5.2 API specification

The backend application provides HTTP methods GET, POST, PUT and DELETE for each resource workers, jobs, companies and users. The last one is used only for authentication so it doesn't have all the methods, for example it's impossible to list users as that would be really insecure.

The GET method is used to request data for a resource. The DELETE method is used for deleting resources. The POST and PUT methods are used for creating or updating resources, the difference being that PUT is always used to completely replace the resource and is an idempotent method. (W3C 2014.)

This means that calling a POST method to create an object 10 times, will create 10 of those objects with different resource identifiers. Calling a PUT method assumes the caller already knows the desired object identifier to be used, and the result of calling it 10 times will only be one object created, and that same object completely replaced over and over again 9 times.

The backend API provides these GET, POST, PUT and DELETE methods listed in Table 2 for different resources to be consumed by the frontend client.

| **Resource** | **Method** | **URL** |
|---|---|---|
| Companies | GET | https://<domain>/companies |
| Companies | GET | https://<domain>/companies/[id] |
| Companies | POST | https://<domain>/companies |
| Companies | PUT | https://<domain>/companies/[id] |
| Companies | DELETE | https://<domain>/companies/[id] |
| Jobs | GET | https://<domain>/jobs |
| Jobs | GET | https://<domain>/jobs/[id] |
| Jobs | POST | https://<domain>/jobs |
| Jobs | PUT | https://<domain>/jobs/[id] |
| Jobs | DELETE | https://<domain>/jobs/[id] |
| Workers | GET | https://<domain>/workers |
| Workers | GET | https://<domain>/workers/[id] |
| Workers | POST | https://<domain>/workers |
| Workers | PUT | https://<domain>/workers/[id] |
| Workers | DELETE | https://<domain>/workers/[id] |
| Users | POST | https://<domain>/auth/ |
| Users | POST | https://<domain>/auth/sign_in/ |

Table 2. Methods and resources provided by the API

Each method is explained in detail during the following chapters, together with the UI screen the method is used in.

# 6 iOS application

Because I’m just getting started with React Native and mobile development in general, I used a process of first planning and drawing out a rough layout sketch of each view, and then implementing it with React Native components.

In my experience there’s no better medium for the first draft of any design than pen and paper, so first I draw the layout on paper. Having used a tool called Balsamiq Mockups 3 before, I then transferred this layout into electronic format that can be nicely embedded into this thesis.

This chapter will cover the design and implementation of the different views of the application, but first we’ll quickly go through installing React Native and creating the project.

## 6.1 Hello world

React Native can be installed on an OS X computer using Homebrew and npm (Node package manager) with the following commands:

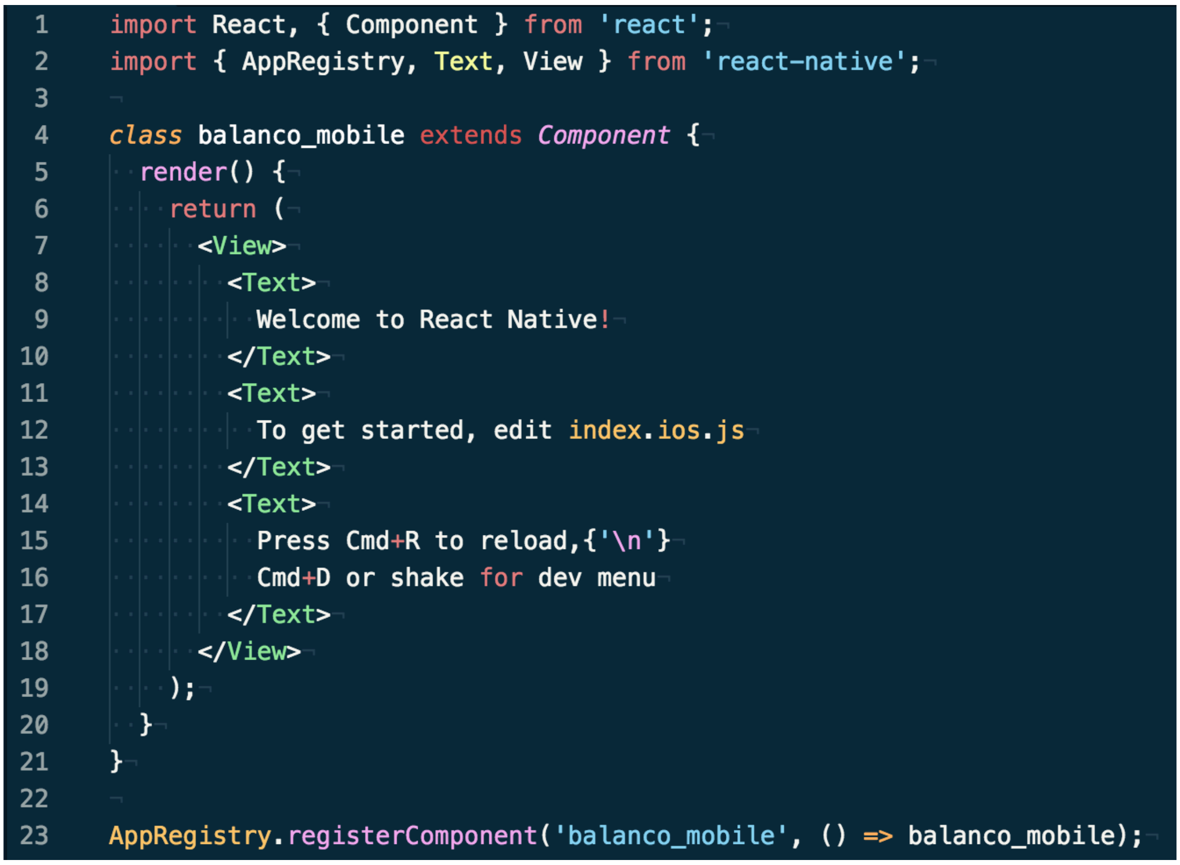
```
brew install node

brew install watchman

sudo npm install -g react-native-cli
```

After successfully installing the prerequisites and React Native, I created a new React Native application by issuing this command in Terminal:

```
react-native init balanco_mobile
```

After doing this, a directory named balanco_mobile has been created and a sample React Native project is already in place and working, only displaying a default greeting inside a `<Text>` element.

The most important file at this stage is index.ios.js, pictured in Figure 7. For demonstration purposes, all styling has been removed from the default example.



```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class balanco_mobile extends Component {
  render() {
    return (
      <View>
        <Text>
          Welcome to React Native!
        </Text>
        <Text>
          To get started, edit index.ios.js
        </Text>
        <Text>
          Press Cmd+R to reload,{'\n'}
          Cmd+D or shake for dev menu
        </Text>
      </View>
    );
  }
}

AppRegistry.registerComponent('balanco_mobile', () => balanco_mobile);
```

Figure 7. File index.ios.js from the default React Native application

To get this simple starter application running, I simply ran the following command in Terminal:

```
react-native run-ios
```

This starts up a development web server serving the application, and the XCode iOS Simulator running the application, showed in Figure 8.

iPhone 6 – iOS 10.0 (14A345)
Carrier
8:29 AM

Welcome to React Native!
To get started, edit index.ios.js
Press Cmd+R to reload,
Cmd+D or shake for dev menu

Figure 8. Sample React Native application running in iOS Simulator

## 6.2 User registration and login

I wanted to keep the layout of the application simple, and especially because it’s a mobile application I wanted to avoid users having to type their passwords twice even when registering an account. That’s why I decided to use the same view and fields for both registering and logging in. The idea is pictured as a layout sketch in Figure 9.

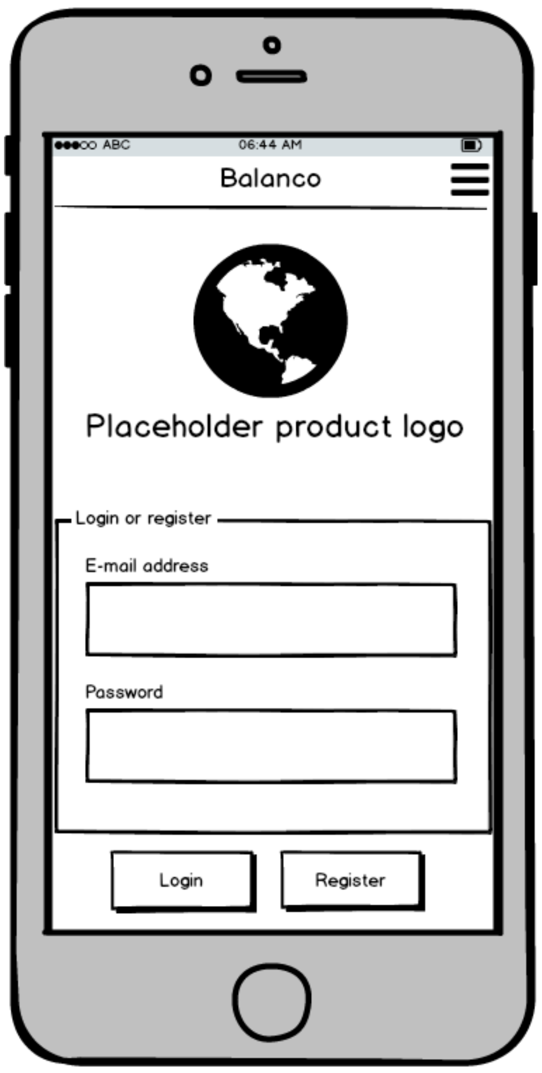


Figure 9. Login and registration screen – layout sketch

To achieve its purpose, the login and registration screen is taking use of two API method listed in table 3.

| Resource | HTTP method | Path | Input | Output |
|---|---|---|---|---|
| user | POST | auth/ | • email<br>• password<br>• password_confirmation | • User object (JSON) |
| user | POST | auth/sign_in/ | • email<br>• password | • User object (JSON) |

Table 3. API methods used by the login and registration screen

This is done by component named Login, implemented in login.js file. It works by binding the Login and Register buttons to corresponding methods `login()` and `register()` by giving an attribute `onPress` to the `TouchableHighlight` component with the methods name as value. These methods then make those API calls with the values user has typed into the text inputs.

The values of input fields “E-mail address” and “Password” are stored in the state, first initializing simply by calling `this.state` in the constructor method. Then these email and password props are updated in the `<TextInput>` component by giving an attribute `onChangeText` that updates the state using `this.setState`.

Finished implementation with the two input fields and buttons is presented in Figure 10.

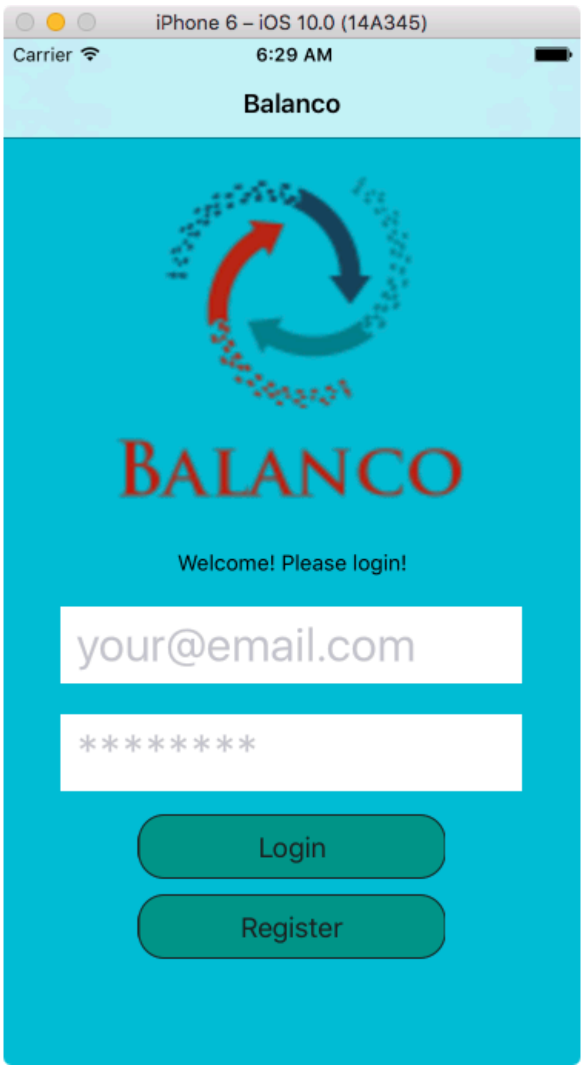


Figure 10. Finished login and registration screen – running in iOS Simulator

The `login()` and `register()` methods use the built in `fetch()` method to make a POST request to the API. After getting a response back, the response inspected and if it contains no errors, user is presented with a notification to check their e-mail for a confirmation mail (when registering), or sent to the next view (Main) covered in the next chapter.

## 6.3 Viewing available workers

After successfully logging, the user is brought to the main view, which at this point is a list of workers available from other employers. I sketched a simple draft of the layout pictured in Figure 11.

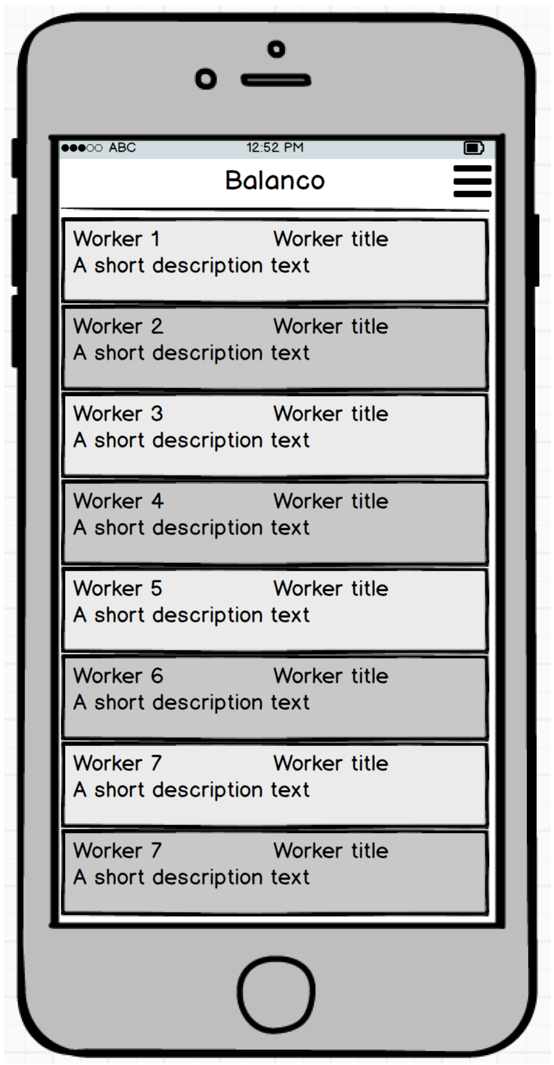


Figure 11. Viewing available workers – layout sketch

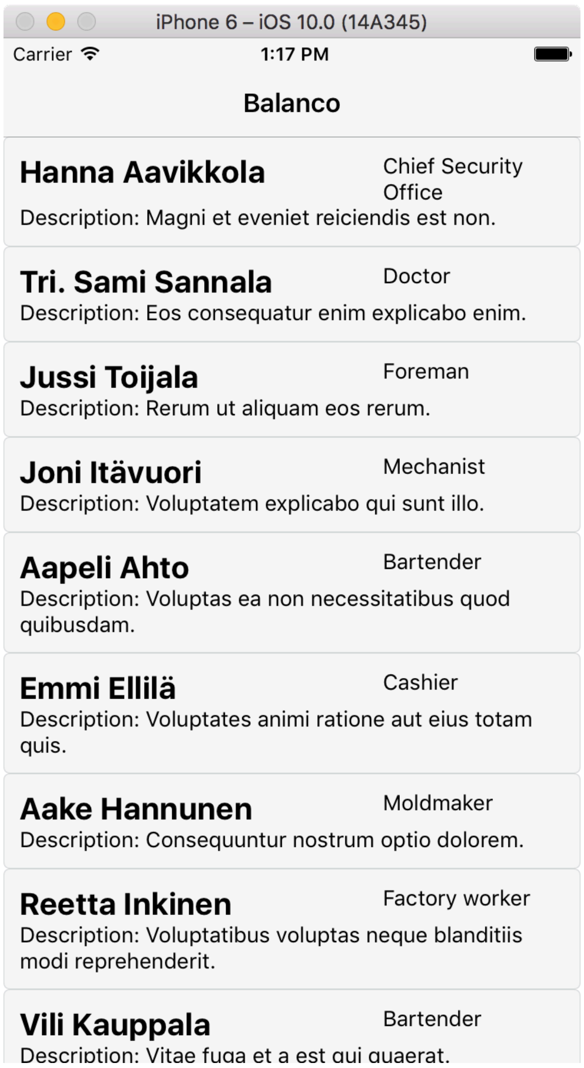
This view is built by calling just one API method, shown in Table 4.

| Resource | HTTP method | Path | Input | Output |
|---|---|---|---|---|
| workers | GET | workers/ | - | • Array of workers (JSON) |

Table 4. API methods used by the main view

This is implemented by triggering the data fetching in a function called `componentDid–Mount`, which React Native will call automatically. It will call our method `fetchData` which simply calls the API with a GET request and puts the resulting JSON data into the Main components state. This change in the state makes React Native automatically re-render the view, allowing user to see the fetched data. The data is displayed by rendering a `ListView` component filled with `Worker` components.

The end result is pictured in Figure 12, a simple view listing all the workers (name, title, description) with some fake data fetched from the API.



Hanna Aavikkola — Chief Security Office
Description: Magni et eveniet reiciendis est non.

Tri. Sami Sannala — Doctor
Description: Eos consequatur enim explicabo enim.

Jussi Toijala — Foreman
Description: Rerum ut aliquam eos rerum.

Joni Itävuori — Mechanist
Description: Voluptatem explicabo qui sunt illo.

Aapeli Ahto — Bartender
Description: Voluptas ea non necessitatibus quod quibusdam.

Emmi Ellilä — Cashier
Description: Voluptates animi ratione aut eius totam quis.

Aake Hannunen — Moldmaker
Description: Consequuntur nostrum optio dolorem.

Reetta Inkinen — Factory worker
Description: Voluptatibus voluptas neque blanditiis modi reprehenderit.

Vili Kauppala — Bartender
Description: Vitae fuga et a est qui quaerat.

Figure 12. Finished main view – running in iOS Simulator

Now I have a working registration & login view, and a main view displaying some information from the API. Unfortunately, I have to leave out rest of the features from this thesis due to limited time I have to finish my degree.

I feel that this is already enough for getting a basic understanding of React Native concepts, and this is a good base to continue development of the application.

# 7 Results and conclusions

My goals were to find out what React Native is and how does it work, and to find out how to develop a cross-platform application with React Native. In my opinion, I reached my goals only partly. Due to limited time left to finish my degree, I had to leave out the whole part of *cross-platform* development, so I'm going to have to explore the Android part of React Native later. Also I didn't yet get to deploy the application on a real iPhone device, not to even mention publishing it at Apple's App Store.

I found out React Native is a good choice for creating mobile applications and especially well-suited for programmes with a background in web development. It basically allows you to develop iOS and Android applications just like you would develop a web page, but still being a real native application using the native API's instead of just rendering a HTML view. It's still new technology though, and I think this is visible from the result of this thesis also. There are not that many books available about React Native, so the sources are a bit thin.

Based on my experiences while developing the application, React Native is really easy to get started with. It provides a clean project skeleton to get started with and the overall development experience with live code reloading is truly enjoyable. Eiseman (2016) lists a lot of benefits compared to traditional mobile development. For example, you don't have to wait for the application to build, instead you can just reload and see changes instantly. Also you can use intelligent debugging tools of Chrome or Safari. Even Apple deals with React Native nicely, permitting JavaScript changes to applications without having to wait for a review.

In retrospective, it would have been a good idea to learn a little bit of JavaScript and React first, before jumping straight into React Native development. The biggest problem for me was not knowing the JavaScript syntax at all, making silly mistakes when creating and importing classes. But I would like to put some of the blame to React Native itself too, the biggest being outdated documentation and tutorials. For example, some of the tutorials are using the older `var Classname = React.createClass` syntax but the boilerplate sample application already comes with the newer `class Classname extends Component` syntax, and sometimes it's hard to mix these two together.

It has been a long-running project, writing this thesis. I've had some trouble making up my mind about the topic and technologies used, but in the end I'm still happy with my choices. This topic was useful, fun and provided a real learning experience, getting to know a side of programming I previously knew nothing about. I'm quite happy with this learning experience, and I think this thesis gives a good view on what's the general idea of React Native and how to use it.

I'm currently not that happy with the end result though, this application is far from being released or being useful to anyone. Only listing some data from the API is not that useful, so there are a lot of features in the to-do list.

Summing everything up, I still think the project was a success, or at least a good start. I managed to get the most important features technically working. I'm already submitting data to the API, acting based on the reply to that, and then displaying some data to the user from the API. I think the majority of mobile applications are just this, just repeated over and over for more functionality, and polished a lot.

The application's development will for sure continue. I will be working on adding features such as listing the needs of other users, adding your own needs and workers, and managing you company information including physical addresses. There will most probably be some kind of a map where you can see the needs and workers, and some kind of a category system based on the industry of the user's own company. Also searching for workers by keywords or any other details is a must have feature. The applications visual design also needs a ton of work.

# 8 Sources

AppDynamics. 2016. Comparing the 4 Most Popular Client-Side JavaScript Frameworks. Available at: https://blog.appdynamics.com/apm/comparing-the-4-most-popular-client-side-javascript-frameworks/ Read: 25.10.2016

BabelJS. 2016. Learn ES2015. Available at: https://babeljs.io/docs/learn-es2015/. Read 5.10.2016.

Code School. 2016. Powering up with React (online course).

Eisenman, B. 2016. Learning React Native. O´Reilly. Sebastopol.

Ethan Holmes & Tom Bray, 2015. Getting Started with React Native. Packt Publishing. Birmingham.

Facebook. 2016. Props. Available at: https://facebook.github.io/react-native/docs/props.html. Read 7.10.2016.

Facebook. 2015. React Native: Bringing modern web techniques to mobile. Available at: https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/. Read: 8.12.2016

Facebook. 2016. State. Available at: https://facebook.github.io/react-native/docs/state.html. Read 7.10.2016.

Facebook. 2016. Tutorial. Available at: https://facebook.github.io/react-native/docs/tutorial.html. Read: 5.10.2016

Facebook. 2016. Why React? Available at: https://facebook.github.io/react/docs/why-react.html. Read: 4.9.2016.

Flanagan, D. 2016. JavaScript: The Definitive Guide: Activate Your Web Pages. O´Reilly. Sebastopol.

Github. 2016. Front-end JavaScript frameworks. Available at: https://github.com/showcases/front-end-javascript-frameworks/. Read: 4.9.2016.

Graziotin, D. & Abrahamsson, P. 2013. Product-Focused Software Process Improvement. 14th International Conference, PROFES 2013, Paphos, Cyprus, June 12-14, 2013. Proceedings. Springer Berlin Heidelberg.

IDC. 2016. Smartphone OS Market Share, 2016 Q2. Available at: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Read: 07.10.2016

Microsoft. 2016. React Native on the Universal Windows Platform. Available at: https://blogs.windows.com/buildingapps/2016/04/13/react-native-on-the-universal-windows-platform/#rF1GOv7MIXdQ4S9I.97 Read: 8.12.2016

Mozilla Foundation, 2016. Introduction to the DOM. Available at https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction Read 17.11.2016

Sansonetti, L. 2015. RubyMotion: Cross-Platform Mobile Development the Right Way. RedDotRuby 2015 / Confreaks. Available at https://www.youtube.com/watch?v=ZV5zCX-HIqNY. Watched 5.10.2016.

Simpson, K. 2015. You Don’t Know JS: Up & Going. O´Reilly. Sebastopol.

Stefanov, S. 2016. React: Up & Running. O´Reilly. Sebastopol.

World Wide Web Consortium (W3C), 2014. RFC7231 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Available at: https://tools.ietf.org/html/rfc7231 Read: 10.11.2016