Haaga-Helia
University of Applied Sciences

Work Diary Thesis- Working as a software developer

Nhi Duong

Bachelor's / Master's Thesis
Degree Programme in BIT
2021

Haaga-Helia
University of Applied Sciences

Abstract

18.4.2021

Author
Nhi Duong

Degree programme
Business Information Technology

Thesis title
Work Diary Thesis- Working as a software developer

Number of pages and appendix pages
90

This Diary thesis describes my twelve weeks working as a Software Developer at Geometrix Oy. The observation period of this thesis was from 01.12.2020 to 04.03.2021. The thesis's content focuses on my daily work tasks, weekly evaluation and discussion. On a daily basis, I describe my plans to carry out my work tasks, skills needed to complete the tasks and things that I learned from doing the tasks. The readers can also expect detailed analysis about frameworks, tools, technologies and skills needed to complete the tasks. Besides, I will share challenges that I face during my working process and my ideas about how to maintain work-life balance.

Here are some findings that I found when writing this thesis: Rendering map markers or annotations using Mapbox in React Native application, integrating external tile servers to Mapbox in React Native application, pros and cons of using Mapbox's Custom Header, using Redux thunk in handling async logic, handling multiple environments in React Native, automating building and releasing processes. The main results of this thesis are related to React Native and Mapbox. Additionally, I noted down some Git commands and their use cases in a real project. There may be more things that you may find helpful if you are developing a React Native app.

This thesis would be useful for those who want to know more about the experiences of working as a software developer in Finland and those who are interested in using React Native and Redux to develop mobile applications. Moreover, there are many useful information about React Native technology, Map Feature Service, Web Map Service, UX/UI design.

Keywords
Software development, mobile application, programming, React Native, Redux, Mapbox.

Table of contents

1 Introduction 1
2 Framework 2
2.1 Analysis of your current work 2
2.2 Evaluation 4
2.3 Development 5
2.4 Interest groups at work 6
2.5 Interaction skills at work 7
3 Diary entries 8
3.1 Observation week 1 (7.12.2020- 11.12.2020) 8
3.2 Observation week 2 (14.12.2020- 18.12.2020) 16
3.3 Observation week 3 (21.12.2020- 23.12.2020) 23
3.4 Observation week 4 (28.12.2020- 01.01.2021) 27
3.5 Observation week 5 (04.01.2021- 08.01.2021) 31
3.6 Observation week 6 (11.01.2021- 15.01.2021) 36
3.7 Observation week 7 (18.01.2021- 22.01.2021) 41
3.8 Observation week 8 (25.01.2021- 29.01.2021) 46
3.9 Observation week 9 (01.02.2021- 05.02.2021) 55
3.10 Observation week 10 (08.02.2021- 12.02.2021) 62
3.11 Observation week 11 (22.02.2021- 28.02.2021) 70
4 Discussion and conclusions 80
5 References 84

Table of Figures

Figure 1	Interest groups at work
Figure 2	Jira board integrated with Github
Figure 3	Example of wrapping router with Redux
Figure 4	Example of app navigation's structure
Figure 5	Example of a separate style file in React Native
Figure 6	Example of using Higher Order Component (HOC)
Figure 7	Introduction carousel
Figure 8	Example of using PropTypes in React
Figure 9	Example of map features (Mapbox 2021)
Figure 10	Performance Monitor window on iOS simulator
Figure 11	Mutable JavaScript Object example
Figure 12	Example of GeoJSON data
Figure 13	Comparison between Toast and Snackbar (UX Collective 2021)
Figure 14	React Native touchable component (About React 2021)
Figure 15	Polygon illustrated on Mapbox
Figure 16	Example of the Form component of "tcomb-form-native"
Figure 17	The use of tcomb-form-native to generate dynamic form
Figure 18	Mapbox components for rendering annotations
Figure 19	Example of render an image with Authorization Header in React Native
Figure 20	Example of Mapbox's Symbol Layer use
Figure 21	Example of Redux "thunk function"
Figure 22	An example "thunk" function demonstrating the pattern to handle Async request in Redux-thunk
Figure 23	Web Map Service layers (Land Processes Distributed Active Archive Center 2021)
Figure 24	Adding testing WMS raster layer to Mapbox
Figure 25	Adding testing water-colour raster layer to Mapbox
Figure 26	Demonstrating original map view
Figure 27	Map capabilities description table
Figure 28	Example of layer data in XML document return by making getCapabilities request to Web Map Service
Figure 29	Example of using bottom modal in Mobile Application (MAPS.ME application)
Figure 30	Applying immutable updates to nested state
Figure 31	Demonstrating form's value object and form's schema object
Figure 32	Programmatically focus cursor to the next input field

Figure 33	Compare JavaScript array iteration methods
Figure 34	Working on multiple Git branches
Figure 35	Example of creating Root Reducer in Redux
Figure 36	Exclude a specific reducer from being reset
Figure 37	The process of integration (Juha Linnanen 2019)
Figure 38	Continuous Deployment (Juha Linnanen 2019)
Figure 39	Example of CI/CD config file (Juha Linnanen 2019)
Figure 40	Example of using environment variables (Juha Linnanen 2019)
Figure 41	Fastfile structure
Figure 42	Fastlane's available lanes
Figure 43	App's icon with badges
Figure 44	Private lane to add badge to icon
Figure 45	Fastfile summary table
Figure 46	Different targets in Xcode project
Figure 47	Adding configurations for the same target in Xcode.
Figure 48	The Roadmap to setup different environments for React Native iOS project
Figure 49	The Roadmap to setup different environments for React Native iOS project (option 2)
Figure 50	Example of using Bundle Suffix ID to generate unique Bundle ID for each scheme

1 Introduction

Under the basic requirements, my thesis consists of twelve working-weeks observations. The starting date of the diary is 01.12.2020 and it is planned to end on 03.03.2021. The observation period is extended to 3 more days in April to compensate for 3 Christmas holidays in December, 1 New Year holiday and 1 Epiphany holiday in January. Following the structure of a work diary thesis, the first week of the period is reserved for writing about the starting point including required tools, technical skills, frameworks and my beginning work tasks. During the next ten weeks, the diary focuses on describing in detail my daily objectives, my plan to complete selected tasks, challenges and solutions for the problems. At the end of every week, I evaluate and discuss new things that I learned from doing the tasks as well as difficulties that I faced.

My main responsibility in this job is to implement a mobile application for iOS devices. The Android version and the backend of this application has been existing. The main technologies that were selected are React Native, Redux, Mapbox. Besides, Web Feature Service and Web Map Service are the main data sources of the application.

The backend has been built with PostgreSQL and Java. Vagrant and Virtual box are used to build my own development environment. Despite the fact that I work mainly in the front-end side, knowledge about relational databases and the backend are required to understand the whole system architecture. About the working method, Kanban is selected as a development approach. We use Jira as the tool to visualize backlog and selected tasks in Kanban board.

At the time that I started working at Geometrix Oy, I was the first foreigner in the company so I will bring here some experiences that may be useful for international students who are interested in working in a Finnish company. About my company, Geometrix Oy is a Finnish company that is located in Helsinki. Its objective is to utilize and customize geographic and spatial information to support work supervision, asset management and driving routes saving, etc. The company’s products are both web applications and mobile applications to meet various needs of customers.

The project that I am working on is a part of their system. In the Android version, there can be more functionalities that users can use depending on their user role. However, the iOS version only focuses on map-related functionalities.

2 Framework

2.1 Analysis of my current work

For all software developers, the first weeks are always the most challenging time since we need to make more effort to get familiar with the existing codes, system architecture and database structure. The good point is I was not expected to jump into developing new features in the first weeks. During the first week, the tasks that were assigned to me are: constructing my own development environment, refactoring the current codes, learning how to investigate responses from the server, learning the company working styles, customers and products, preparing for remote work in the next weeks.

Constructing my own development environment

Although this is not my first time working as a Software developer, setting up the development environment took me a lot of time and effort. In a nutshell, Vagrant is the tool used to create my own local backend. As a good practice, every developer in my company should have their own local environment so that any modification made will not affect the system and the data can be safely deleted or modified. Beside the local environment, we have an internal shared server called Ned. It is used for testing purposes and it can only be connected by using a virtual private network (VPN) to connect to the company Intranet if working remotely. We have another staging server which is used by external parties who are our customers. The production server is always kept safely from any developing test and modification.

When learning about Vagrant, I realize Vagrant is a very useful tool to create and manage virtual machine environments. The virtualization tool that I use to build my virtual environment was VirtualBox. We can understand that VirtualBox is a host location where the virtual environment can run on. Another terminology used by Vagrant is Provisioner. The Provisioner is a shell script in my case. It can be any tool that automates the software installing process when we run the ‘vagrant up’ command.

In the first week, I also got to know Jenkins for the first time. It is an automation server that provides continuous integration and continuous delivery (CI/CD) of a software by automating development processes from building, testing to deploying. However, the CI/CD workflows or pipelines of the application that I am working on has not been implemented. We are intending to implement it in the near future.

Setting up and running the React Native application was simple and fast. I spent a little time checking if the application was initialised with Expo or React Native CLI. In my case, the application was initialized with React Native CLI. In brief, Expo is a fast way to start a React Native application. It is a toolchain that provides convenient features, for example, sending notifications to users, location access, camera access control, etc (Dinuka Piyadigama 2020). Another advantage of using Expo is that it removes the need of setting up two separate builds for iOS and Android versions. In other words, developers do not need to have both XCode and Android Studio Code to build and preview two versions in two platforms. However, in most cases, I prefer using React Native CLI because I can modify and extend the native codes for each platform if needed. A very genetic example when I want to modify native codes is when I want to integrate a native library to the application. With Expo, we cannot do that because native Java codes or Swift codes are not revealed in the application.

Refactoring the current codes

I believe code refactoring is the key task of any developer when starting a new job. “The best time to consider refactoring is before adding any updates or new features to existing code” (Sydney Stone 2018). Refactoring would be much easier when the project is not developed to a level of completion or it is freshly initialised. I spent almost the whole week reading and getting familiar with the codes since the application that I am working on has been developed by other developers. From now on, I will refer to this application by its own name- Mobilenote. I believe readers can find demonstration videos about Mobilenote for Android platform on Geometrix Oy website.

My refactoring process includes reading, cleaning and editing the existing codes without changing the function of them. By doing that, I can make sure that the codes are understood well by me and it is readable and maintainable in the future. Reading previously written codes can be challenging. However, here are some tips that I got from my own experiences to make this process easier. First of all, I tried to find one piece of code that I understand well what it does and tried to trace back from here. All functions, variables are related to each other to serve an operating purpose. Discovering and tracing codes helped me go through the whole structure and understand how codes are organized. Next, by repeating this tracing process, I can see the overall picture of the application. The more pieces of code that I understand, the clearer picture that I see when connecting different parts of the codebase.

Learning how to investigate responses from the server

Here is another basic step in getting familiar with the new system. Although I can use Postman as a tool to send requests and get responses from the server. In some cases, the response can be just “Internal Error” which does not reveal details about request error. I got instructions from a senior developer in my company that in order to check what is going on with the server, I can run “vagrant ssh” to SSH into the running machine and get access to a shell. With the access, I can interact with the machine and get logs from the GlassFish server.

Learning the company working styles, customers and products

I was introduced to the company working styles, customers and products during the first two days of the first week. We have many loyal customers from STARA, KIRAVE, KYMP, GRK Rail Oy, HKL and Sipoo, etc. About working styles, we have a general meeting every week via Teams to update about what has been done in each project. Due to the pandemic that has been happening since the beginning of 2020, most of us are working remotely. We have another weekly technical meeting to select work tasks for this week and discuss technical issues. About group work and communication, we use Teams as a communication tool. With Teams, we can have separate channels for posting communication topics within one platform. About technical documentation, the company has its own Confluence wiki site that can be used for sharing and keeping documents regarding development processes, instructions to set up things, description of available tools and application’s features.

Preparing for remote work in next weeks

During the first week, I was asked to be present in the company to get support and instructions from a senior developer. The senior developer ensured that the local development environment is ready for me to work from home in upcoming weeks. We also configure the VPN network that allows my computer to connect to the company’s internal network remotely. With the VPN connection, I can use the company testing server if there is any problem with my local server.

2.2 Evaluation

Before joining the company, I have already had experiences of working on my personal React Native application, so I am pretty confident in my React Native skills. Moreover, because I have already released this application to Apple’s App Store, I know how to work with TestFlight, how to revoke certificates, provisioning profile and so on. In addition, thanks to 6 months working in a Full-stack intern position, I have accumulated knowledge about setting up the environment, developing databases, creating User Interface, tackling

business logics and algorithms on the backend side, etc. Those knowledge and experiences will help me to handle tasks in this job.

About my current situation in this company, I think I have handled everything quite well until now. Although server development is not my strength, I managed to learn basic terminologies and the system architecture. I got a lot of support from the mentor who is the senior developer of the company. The mentor advised me to learn a bit about Linux and its command lines. He also advised me to have the habit of typing the whole lines of codes without copy and paste since it helps memorize things faster and reduce dependency on other sources.

I have much more confidence in my front-end developing skills. After the first week, I understood the application's structure, data flow, state management and how Redux was integrated with React Native. I managed to draw a diagram demonstrating the data flow in the application by myself. Because the working style in my company is self-organising, I proactively arranged my backlog in Jira and listed down what should be done next.

2.3 Development

With six-months work experiences in software development, my professional level can be considered as junior level. Despite my young experiences, I have built good fundamentals about both frontend and backend development. I found knowledge about the frontend side and backend side closely supports each other. A frontend developer must know at least how to define interactions with the server side and how to handle responses from the server. And a backend developer should know at least how the data would be handled on the client side to optimize their solutions. When studying or working in software development, I avoid sticking to one side of the development since it will limit my own flexibility. My objective is to develop my professional skills comprehensively.

Moreover, since I acknowledge that the codes that I write may not be as efficient as codes written by senior developers, I always try to optimize my solutions by finding a new way to write codes that generate the same app behaviours but consume less memory and speed up the program. In addition, I am trying to improve my research and self-taught skills because they are the key skills that help developers to develop further in their software development career. There are a lot of available technologies nowadays and new frameworks keep coming out from time to time so self-taught skill will allow me to learn things fast and adapt to new trending frameworks.

How it shows in my work

At my professional level, I found myself re-write codes multiple times to make sure it is the shortest and most efficient way to implement new features. Moreover, I always keep a habit of refactoring my own codes to make sure there are no redundant variables or operations. Optimizing codes looks like a never-ending process to me. When the application keeps growing, there will be more room for improvement, not to mention the old solutions may conflict with the newer one.

Writing reusable codes is one goal in my development process. There used to be many times when I was too dedicated to write new codes without planning ahead which produces a lot of code redundancy. When looking back, I found a lot of functions or components that I can reuse to avoid “repeating myself”. From this experience, I will try to write as many reusable codes as possible from now on. In order to do that, I will make sure every method should do only one thing so that I can reuse them in the future.

Learning objectives in the future

Within the scope of my job, I am planning to learn more about Web Feature Service (WFS) and Web Map Service (WMS). WMS and WFS are new terminologies to me. I have not worked on map service before so learning about them will be my top priority. Besides, I am planning to invest time in learning how native applications are developed. I am not expecting myself to become professional in native languages soon but having basic understanding will help me understand the Android version of Mobilenote since it was written in Java.

2.4 Interest groups at work

The company involves internal groups and external groups. Within the organization, we are having eighteen developers in total. The company can be seen as a flat organisation where there are not many levels of middle management. The company has a CEO and a manager and the gap between them and staff is small. We are self-managing teams, so we organise our own work tasks and discuss important issues directly with the CEO and the manager.

The external stakeholders include customers, society, government and third-party companies, etc. In addition, we have using accounting and headhunting services provided by third-party companies. The chart below depicts all interest groups that connected with my company:

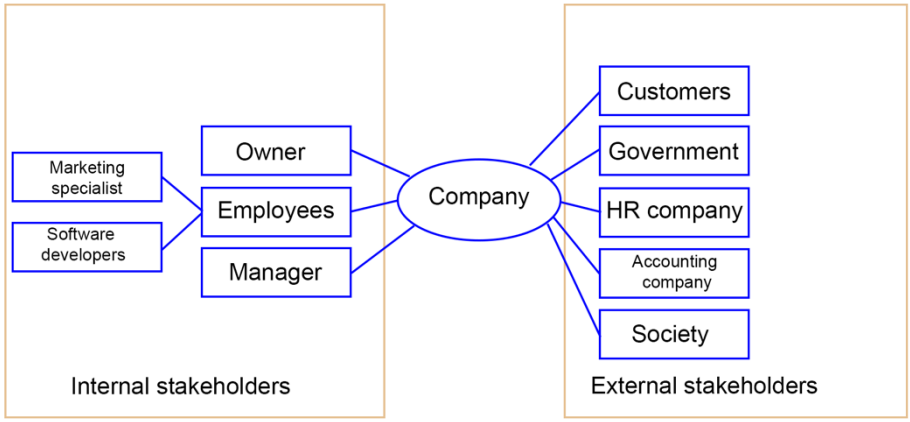

Figure 1. Interest groups at work

2.5 Interaction skills at work

We have separate channels for each project on Teams and we communicate mainly on Teams channels. Beside discussing technical issues in specific channels, we have a General channel where organisational information is shared. The company owner keeps us updated about the company's activities, holidays, upcoming events and general meetings in General channel.

Working remotely causes certain challenges in interacting with my colleagues. We do not have regular face to face meetings where discussions and sharing information are much easier. Interacting requires more effort when we have to plan everything ahead. For example, when there is a need to have a face to face meeting, I have to send messages to my colleagues or to my mentor at least one day before to reserve the meeting. Another challenge when working from home is that I must make sure I have a stable Internet connection to be available for video calls on Teams during working time.

In the Mobilenote iOS project, I mainly work independently due to the fact that I am the only React Native developer in the company. However, in technical meetings, I can interact with the backend team and raise questions or ask for support about server-side related issues. I am not required to have any interaction with the company customers, they only interact directly with the manager, owner and senior developers. The reason is I have a language boundary and I do not have experience in consulting that can support the customers.

3 Diary entries

3.1 Observation week 1 (7.12.2020- 11.12.2020)

Monday 7th December 2020

My target for today is to reconstruct the navigation and add a side drawer to the application. Navigation is the backbone in mobile applications. It defines all the routes that users can navigate around the application from a central place. Establishing a well-structured navigation requires a good plan and a well-designed layout. But it can be challenging since there are limitations about the screen size of mobile devices. There are different navigation patterns created to solve this challenge. These patterns are:

- Hamburger menu and side drawer: Hamburger menu combined with sidebar is one of the most common patterns that help save space. The drawer hides the navigation list until the user presses on the hamburger icon.

- Bottom/top tab bar is another way to free up the screen yet still allows the user to see available destinations within the application. The tab bar can either be located on top or at the bottom of the screen. The advantage of tab bars is that users can know their current location immediately at a glance. However, the downside is that the navigation options are limited. The maximum number of navigation options that can fit the tab bar is five, if there are more than five navigation options, the navigation tab bar will look very cramped.

- The Floating button is well utilized by Google since they realize it is a good way to promote user actions (Nick Babich 2017). It is a circle action button that is designed to stand out and attract user attention. It often holds the most prioritised function of the application. However, using a floating icon is a risky choice as it is not easy for users to understand what it means and what it does.

- Full screen navigation turns the entire home screen to a place where the user can navigate. It is only suitable for task-based applications where developers do not need to worry about saving screen space for other purposes.

There can be a lot more navigation patterns but those are the most common and basic one. In the application that I take responsibility for, full screen navigation has been implemented. The whole home screen is devoted to containing all action buttons that will navigate the user to other screens where they can carry out the selected action.

During the day, I have moved all the codes that define the app routes to a separated file called AppNavigator.js. They were located in App.js files before. Handling them in a separate file would make it easier for maintenance. The library which was used to handle navigation in this application is “react-native-router-flux”. It is developed based on React Navigation, however, there are more API provided to interact with.

By the end of the day, I have successfully added the hamburger menu and the side drawer to the application. The side drawer would be a suitable place to hold destinations to the user's account and sign-out button. With the side drawer, the user can sign-out anytime and from anywhere without coming back to the home screen.

Tuesday 8th December 2020

I am planning to investigate and replace all legacy React Native lifecycle methods to new life cycle methods today. The application has been developed since 2018 so there are a lot of changes and innovations in React Native methods since then. In addition, I got a lot of warnings from React Native that “componentWillReceiveProps" has been renamed and it should be replaced soon. Although it does not affect how the application works or produce any bugs, I decided that I should switch from “componentWillReceiveProps" to “componentDidUpdate” and “getDerivedStateFromProps” because “componentWillReceiveProps” may be deprecated in the future.

Before changing to new life cycle methods, I made a research about how “componentWillReceiveProps” actually works. This method receives the “nextProps” value and allows developers to compare “nextProps” and “this.props” values. We can understand that “this.props” is the value that exists in the component before “nextProps”. When we compare “nextProps” value with “this.props” value, if there are differences between them, then we can carry out some operations or perform “setState” to re-render the component. However, with the invention of async rendering, using “componentWillReceiveProps” can cause side effects.

By the end of today, I have completely switched from “componentWillReceiveProps” to “getDerivedStateFromProps” and “componentDidUpdate”. They are better alternatives for “componentWillReceiveProps”. “getDerivedStateFromProps” is static and we cannot access “this” inside it, instead, we must return an object to update state. Since “getDerivedStateFromProps” is invoked right after the component receives new props and when the component is initialised for the first time, it is an ideal place where we can update

state. Next, when a state changes, “componentDidUpdate” method will be called so it is the right place to define operations after a state get updated. Unlike “componentWillReceiveProps”, “componentDidUpdate” receives “prevProps” and “prevState” instead of “nextProps”. It can be a bit confusing about the timestamp. However, the only thing we need to memorize when migrating from “componentWillReceiveProps” to “componentDidUpdate” are replacing “nextProps” in “componentWillReceiveProps” with “this.props” in “componentDidUpdate” and replacing “this.props” in “componentWillReceiveProps” with “prevProps” in “componentDidUpdate”.

While testing the application, I recognized the login screen is always re-rendered multiple times when user logout from the application. The reason is “componentWillReceiveProps” was used in an unnecessary place. While the “Main” component is the parent component of the “Home” component, both of them have operations inside “componentWillReceiveProps” that force navigating back to the login screen when the props that indicates if the user logged in changes. My solution is removing that operation in “Home” component since it was already defined in the “Main” component.

Wednesday 9th December 2020

I work in the office today instead of working from home because the company has ordered a photographer to take photos for new employees. I have listed down some work tasks that I can start working on when I get more familiar with the application. One of them is fixing the center icon floating on the map. There are two main problems with it: Firstly, the map is centred on San Francisco instead of Helsinki when opening the map for the first time. Secondly, the GPS button or the centering icon does not navigate the map back to my current location which is Helsinki.

I spent almost half of a day trying to figure out what has been wrong with the application. Surprisingly, the basic problem is because the map is set to follow the user's device location. And the coordinates of the iPhone simulator are set to San Francisco by default. The only thing that I need to do is setting customised coordinates for the iPhone simulator. My biggest problem was not testing the application on a real device before modifying the codes. If I tested the application on a real device, I would know that it was the simulator's problem not the codes. Nevertheless, that experience will remind me to always test a mobile application on a real device beside Xcode’s simulator.

I spent the rest of the day trying to recreate the Vagrant machine by myself. The process takes a lot of time without success. I got the error “Timed out while waiting for the machine

to boot" multiple times. I understand that the installation progress always stops at copying data from the server which means that the timeout length parameter can be too short, or the database package is too large. When communicating with the senior developer who works on the server side, I got the response that he will work on Vagrant installation package soon to make it smaller and less prone for connection errors.

Thursday 10th December 2020

The plan of today is fixing the User Interface and re-defining common styles, fonts, icons and colours in one global object that any component can access. The global style object will be placed at the top of the component tree. Unlike ReactJS, styling is done with JavaScript instead of CSS. Thus, what I will do is create global objects that define the common colour, font, icon used in this application. With the global object, I can import and access global styles from any components in the component tree. The good thing of doing that is switching colours, fonts, icons will be easier because we do not need to trace back every place to apply the new styles.

During the rest of the day, I had a long technical meeting where senior developers share the whole picture about the company system and used technologies. Although I was briefly introduced before about tools, company products and technologies, today is the official day that they present in more details about these things. In the meeting, they introduced me to the best practices and rules to use Git. The rules are: Firstly, every Github branch name should include a ticket code or issue key of the Kanban board ticket. Secondly, a branch should only contain commits and changes which belong to one single ticket. Finally, every commit should also include the ticket code since it supports code review and code management.

Our Kanban board is handled by Jira which is a software that provides tools for Agile development. From Jira board, the team can review all changes, commit messages, branches, pull requests, status of the development process because Jira was integrated with Github. From Github, developers can also review references of commits to Jira tickets/ issues. In other words, commits, branches and pull requests will be linked automatically to the issue key if the issue key is included in the commit message.

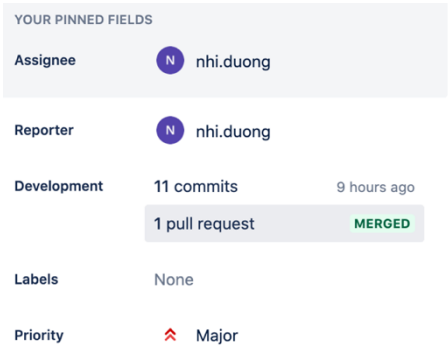

Figure 2. Jira board integrated with Github

Friday 11th December 2020

The task for today is re-installing Vagrant and setting up Redux for the project. I guess that creating a Vagrant machine will take even more time than yesterday because I am working remotely, and I have to use VPN to connect to my company intranet. Most companies have their own Intranet for sharing their network resources and keep those resources from exposing directly to the outside.

While leaving the Vagrant machine to be installed, I started working on Redux. There has been a plan that Redux should be used to handle states in this application since it offers a straightforward method to manage state. Although React Native provides the concept of local state to manage component's behaviour, when the application keeps growing, sharing of data across many components will become complicated. For example, in order to share a state among sibling components, the state should locate in the parent component and it can be passed to child components as props. In case state must be shared from one component to another that is far away from each other in the component tree, the state must be passed from parent components until it reaches the right place. Using Redux will resolve those problems since the state of the entire application will be managed in a state object (Neo Ighodaro 2020).

During the day, I installed all necessary libraries to use Redux. Then I created the basic Redux components including Reducer, Action, Types. Next, I connected Redux store to every screen by declaring a variable to wrap all the <Screen>:

```
1   const RouterWithRedux = connect()(Router);
```

Figure 3. Example of wrapping router with Redux

The final structure of the app navigation connected to Redux looks like this:

Figure 4. Example of app navigation's structure

By the end of the day, I can pass some simple states from the Redux store to components. Having Redux is a good decision especially when the application has to deal with a lot of data sharing.

Week 1 analysis

During this week, I have learned how to read previously written codes better. I think reading other people's codes is even harder than writing my own codes. However, to be able to write good codes, I have to learn how to read codes faster. By reading other people's codes, I can understand what other developers think. They may have better logic and better solutions for the same problem that I encounter.

The other skills that I developed well this week is actively asking questions and expressing personal opinions. In fact, I used to think that a developer only needs to have good programming and technical related skills until I started working full-time. If the right questions are not made at the right time, other people will implicitly think that I have understood the task thoroughly. For example, when my mentor mentioned that the first thing that I should do is installing Vagrant. At that moment, if I did not express that this is something new to me, the mentor may not help me set up the Vagrant machine and deploy the server to this virtual machine. Next, expressing personal opinions is good in brainstorming different solutions for software development. Expressing personal thoughts also gives me chances to listen to other people's opinions. For example, this week, I told my colleagues that "react-native-maps" could be more compatible to React Native than Mapbox. One of my colleagues agrees that "react-native-maps" seems to be more popular in the React Native community than Mapbox. But the others told me the reason that Mapbox was chosen is that Mapbox was developed by a commercial company, so it provides better support and maintenance in the long term. The "react-native-maps" is developed and maintained by the React Native community so it can be a risky choice to integrate to a commercial software. The "react-native-maps" team itself mentioned that due to the quick changes in the React Native ecosystem, they are not going to support the older version of

React Native. The discussion helps me to understand better about why Mapbox was chosen to develop this application.

Next, I learned about reusing and sharing styles in React Native. Generally, styles are plain JavaScript objects. When the styles become more complex, we may want to separate them from the component file. In React Native, styles are saved in a JavaScript file, not in CSS file:

```
- ComponentName
  |- index.js
  |- styles.js
```

Figure 5. Example of a separate style file in React Native

By keeping styles separate from the component files, the codes will look cleaner and it will be easier to share styles between components. Each separate style file may contain specific styles used to standardize buttons, palettes, spinner, etc. To use styles from separate files, we need to import them to the component file. In addition, styles can be passed as properties to a component. As stated by Bonnie Eisenman, passing styles as properties is an effective way to control child component's styles from its parents (Bonnie Eisenman 2017, 78).

Next, I have learned about the best way to render a scrollable list in React Native. There are different ways to render a list of items in React Native, some of them are ScrollView, FlatList, SectionList, ListView. Each of them has its own strengths and weaknesses. ScrollView and Flatlist are two most commonly used components to render lists in React Native. While ScrollView loads all data or list items immediately after the component loads, FlatList, only renders a limited number of list items that fit the visible window, the rest items of the list will be rendered when the user scroll the view (EidorianAvi 2020).

ScrollView and FlatList are used in different cases. FlatList is optimized to render a very large array because it only renders a small number of items for a moment. FlatList obviously has better performance compared to ScrollView when the application has to deal with a long list of data. To be able to use FlatList, we must provide an array of data to FlatList's data property. ScrollView can be used to render a short list and we do not need to provide an array of data to ScrollView.

In addition, I have learned that there are a lot of cases where I can reuse codes. For example, this week, I wrote one shared data validator which can be used by multiple

components. The validator is just a function that takes data value and validating rules, and then returns a Boolean value to indicate if the input data is valid or not valid.

Moreover, when looking for different techniques to write reusable codes, I learned that using higher order components (HOC) is a good way to reuse business logic (reactjs.org/docs/, Higher-Order Components). A higher order component is a function that takes a component as input and returns a new component from the input component (Eric Kim2017). There are a lot of use cases where we can use higher order components. One of them is using HOC to define a reusable error handler. The HOC can be written using ES6 arrow functions like that:

```
const withErrorHandler = Comp => ({ children, ...props }) => (
    <Comp {...props}>
        {children}
    </Comp>
)
```

Figure 6. Example of using Higher Order Component (HOC)

And the input component can be wrapped by higher order component like that:
export default withErrorHandler(MainComponent)

3.2 Observation week 2 (14.12.2020- 18.12.2020)

Monday 14th December 2020

On every Monday, we have a general meeting to update the working progress of each person to the manager. For the rest of the day, I will create an auto-scroll carousel to illustrate the application's features. The auto-scroll carousel is planned to be shown at the login screen where users can quickly review what features that the application offers at a glance.

The introduction carousel was created as a separate component and was imported to be used in login component. The carousel covers the full width and one quarter of the screen height. In React Native, we can get the width and height of the device's screen size by using Dimensions.get("window") method. The idea to implement this carousel was utilizing ScrollView to handle horizontal scrolling of images by setting the "horizontal" prop of ScrollView to true. Here are some important ScrollView's properties that I utilized to create the carousel: The first property is "pagingEnabled". This prop divides the scrolling content into sections where we can apply interval logic to enable automatic scrolling. In this case, the scrolling content is a list of images passed to ScrollView. The second one is "onMomentumScrollEnd" which is fired when the scroll view has finished moving. We can use this prop to pass an action that should be performed when the scroll view ends.

About the scroll event, as soon as the component is mounted, the selected index state will be updated every 3 seconds. The selected index decides which image is visible on the screen. Besides, the selected index can be used to highlight the circle indicator of the visible image. Figure 7 demonstrates the final result of my work:

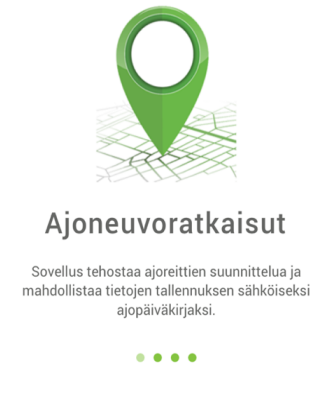

Figure 7. Introduction carousel

Tuesday 15th December 2020

At today's technical meeting, we discussed having general guidelines for User Interface (UI) designs. The guidelines cover both common rules for choosing UI designs and detailed instructions about how to optimize visualization of the application on iOS devices. According to what has been agreed upon: First of all, the UI designer will design the basic styles for buttons, layout so the developer only needs to choose the right style from the ready-made collection. Secondly, the developer should not make the UI decisions only based on their instincts and personal taste. Finally, all the used icon libraries should be documented to make sure every future developer can know which to use.

About the instructions to optimize the application visualization on iOS devices, I made a research by myself by reading Apple's Human Interface Guidelines. Apple documentation clarifies the best form of colours, fonts and resolution. Here are some main points that I noted down: Firstly, the default colour space on iOS is Standard RGB (sRGB). Secondly, in order to reduce the image file size without reducing image quality, we should use an 8-bit colour palette. Finally, the status bar that displays the current state of the iPhone device (time, battery, etc) varies among different iPhone models. Therefore, it is important to test an application on multiple models to make sure the application content does not overlap the status bar.

For the rest of the day, I focused on re-arrange the UI of the application. Because the application has been developed for only 6 months, its User Interface has not yet been polished. By the end of the day, the feature's form view of the application was somehow improved.

Wednesday 16th December 2020

I decide to continue improving the current version of the application by adding "prop-types" library to the application. Another task for today is debugging the error that users cannot open the same feature form constantly.

Using PropTypes is a good way to check React props or any data objects passed to React Native components. To use PropTypes, I defined the types of data tended to be passed to components, then PropTypes will check if the data passed to the component has the correct type. Defining the type of data helps avoid potential bugs when compiling the application (Glad Chinda 2020). I got the idea of using PropTypes when using TypeScript on the previous project, but since this application was written by JavaScript already, I had to add

the extension for it with “prop-types” library. Here is the example of defining prop types in my application:

```
73   WMSLayers.propTypes = {
74       selectedLayerName: PropTypes.string,
75       updateToken: PropTypes.func,
76       WMSCapabilities: PropTypes.array,
77       token: PropTypes.object,
78   }
```

Figure 8. Example of using PropTypes in React

The next task for today was debugging the mentioned bug. Before delving into the bug details, it is important to understand some basic terminologies used in the application:

- “Feature” refers to geographical features. They can be natural geographical landforms or artificial objects including human settlements, engineered forms. These features are visible on most kinds of map. Features can be illustrated as symbols, shapes, polygons, lines on the map. The picture below demonstrates how features are illustrated on the map. From now on, let’s call it “feature” or “map-feature” for short.

- “Spatial data” refers to geospatial data which is used to describe information about objects on geometric space (Ralf Hartmut Güting 1994, 1). Spatial database is optimized to store and query the geometric data objects.

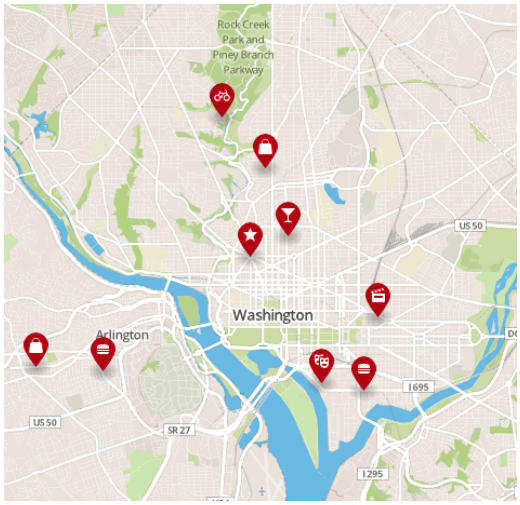

Figure 9. Example of map features (Mapbox 2021)

Our map function allows users to query all map-features of the selected map-feature type and display them on the map. When the user presses on a map-feature marker (which could be a symbol, polygon, line or even icon), a modal box will pop-up to show all details about

the selected map-feature. The problem that happened was that a selected map-feature cannot be opened constantly. By the end of the day, I figured out how to fix it. The reason for this bug was that only one Mapbox “ShapeSource” had been used to render multiple “circle layers” with different filter expressions. The solution was creating two separate “ShapeSource” layers: one layer holding all queried map-feature markers, one layer holding the selected map-feature marker.

Thursday 17th December 2020

The objective of today is using Redux to pass map data to React Native components. Last week, I added Redux to this application so the main task for today was moving local states to the Redux store. When the application is still simple, it is the best time to add Redux. Moreover, I planned to add the “immutable-js” package to the application.

The reason that I decided to use a third-party library to handle immutability was that it will guarantee the data in Redux reducer cannot be changed. In big applications, subscribing state and tracking mutation throughout the application can generate huge memory consumption which can slow down the performance. That’s why I tried to make the Redux state tree become an immutable object. I believe it will help improve the performance of the application especially when this application has to deal with a lot of data.

During the rest of the day, I extracted some local states to Redux. I only extracted the states that I understand the best as it is not simple to change the data flow. Once the state was lifted to the Redux state tree, we can get access to this state by connecting this particular component to the Redux store. The connect function receives two parameters which are “map state to props” and “dispatch to props”. “Map state to props” gets the Redux states from its store and maps states to the component’s props. “Map dispatch to props” maps the functions that allow us to update Redux states to props.

Friday 18th December 2020

I had a technical meeting with the Android team where they showed me how the application is working on Android devices. This version has been published to the Play store. During the meeting, they instructed me to clone the codes and run the application on Android studio. We got a little trouble in running the application until we figured out OpenJDK was not downloaded and configured on my computer. I did not expect that I could reuse any piece of code from the Android version, but the Android version can give me hints about what functions to be added to the iOS version next. After the meeting ended, I continued to investigate the data flow and extract local states to Redux.

Week 2 analysis

In summary, this week, I used ScrollView to create a carousel at the login screen of the application. The use of ScrollView in creating an auto scrolling carousel was explained on Monday's diary note. Next, I added a PropTypes extension to the application to check if data passed to a component has the correct type. Using PropTypes is a good way to eliminate the potential bugs when compiling the application. Moreover, to improve the performance of the application, I also applied the "immutable-js" library to guarantee the immutability of data. In this week's analysis, I will discuss more about how to improve React Native application's performance with immutability.

Before explaining more about immutability, I will describe ways that we can get the application's performance assessment. The application performance can be measured by a tool called "react-native-slowlog". It is a timer to track React Native application's performance problems. The tool can be installed with this command: "npm I -S react-native-slowlog". This tool will notify developers when it recognizes there is a performance problem with the application. A more common tool to track app performance is the built-in Performance Monitor. The Performance Monitor can be opened by launching the React Native application on iOS simulator, then pressing Command D to open the Developer Menu. From there, we can select Perf Monitor.

After these steps, the Performance Monitor should be opened as demonstrated below:

Figure 10. Performance Monitor window on iOS simulator

The RAM number indicates the memory usage of the process. The JSC number indicates JavaScript thread memory usage. In the Views column: the first number indicates the visible views at the moment, the second number shows saved view count. The UI number means the current frame rate in the UI. The JS number means the frames / second of JavaScript thread.

What is immutable data? The term “immutable” refers to values that should be unchanging over time (Esteban Herrera 2018). Most primitive values, for example, String, Integer, null, undefined, Booleans, etc are immutable values (Rooney 2021). In JavaScript, Array and Object are mutable and that can bring problems to our application if we are not aware about it. Here is an example of JavaScript’s behaviour when dealing with Object or Array:

```
export default settings;
const myPhone = {
  name: 'iPhone',
  model: 8,
};

const myPhoneUpdated = myPhone;
myPhoneUpdated.colour = 'white';

console.log(myPhone); //Result: { name: 'iPhone', modal: '8', colour: 'white }
```

Figure 11. Mutable JavaScript Object example

From the example above, we can see that the original object has been changed after we assign its value to the second object and add more value to the second object. What we actually did is just assign a reference to the first object’s location.

We should not mutate states directly because it may lead to bugs and it may prevent us from making performance optimizations in the future. There are many ways to avoid changing states that should be immutable. However, handling immutable states updates can be tricky sometimes. I used to make a lot of mistakes when handling immutable data until I made a research about it. Here are some ways that I used to update immutable states in Redux, they are also applicable to ReactJS and React Native states: Firstly, we can use “…” spread operation to update state. The spread operator is a syntax of ES6, it allows us to copy all properties from an object to another object or copy all items from one array to another array. We can also use “concat” to update the array: concat returns a copy of the array so that we can modify the array without mutating the source. Next, we can use Object.assign to create a copy of an object: It helps us to avoid directly modifying the object

that belongs to state. The final way is to use the Immutability helper. Immutability helpers can be a library for mutating a copy of data without directly changing the source.

The approaches that were listed above are the most common approaches to avoid mutations and sharing by cloning data and making data immutable. Besides, we can use pure components or Memo (if we are using hooks) to optimize the applications and reduce re-renders. Pure components can prevent re-renderings since it only makes shallow comparison on the props and state. useMemo is the hook that accepts a function and a list of dependencies as arguments. useMemo will call the function and return a value when it is called (Stephen Hartfield 2020). The best thing is that every time it is called, useMemo will check if the dependencies change. If any dependencies change, it will call the function again, otherwise, it will only return cached value.

3.3 Observation week 3 (21.12.2020- 23.12.2020)

Monday 21st December 2020

The Vagrant machine was installed successfully on my computer. My task for today is changing the application behaviour from zooming to the Helsinki region on opening the map instead of zooming to the user's location when opening the map. Unfortunately, despite having the Vagrant box running, the application kept sending the error message “Request failed with status code 500” when I opened the map. Because it is an error from the server side, I need help from the backend team.

We had a long meeting to investigate the problem. After almost two hours, we figured out the cause of the error was that one variable called “Koodisto” on the server side requires a property coming from the company’s private network. The problem was I cannot use VPN while connecting to the local running virtual machine (Vagrant machine) and vice versa when working from home. After making some searches, I got a lot of answers for the causes of that: As soon as we connect to the VPN, we get a completely new address that the local virtual machine cannot obtain communication to. At the same time, the list of known routes that VPN clients receive does not include the virtual networks, then the virtual networks are removed.

Finally, we jumped into the conclusion that there were two solutions for it: the first one was switching the network to bridge mode and restarting the virtual machine, the second choice was just changing the property of “Koodisto” to null.

Tuesday 22nd December 2020

Changing the application behaviour from zooming to the Helsinki region on opening the map instead of zooming to the user's location is the task that I intended to work on yesterday. However, due to the problem with the Vagrant machine, I had a delay to do this. Therefore, this becomes the task for today. In addition, I am about to have a weekly technical meeting to review completed tickets on the board and consider what to do next.

The plan to implement this task was setting “followUserLocation” prop of Mapbox’s Camera to “false”. That should stop the map view from following and zooming to the user's location when opening the map. The next step was using the Geolocation API to get the user's location. The values that “getCurrentPosition” returns were the longitude and latitude of the user’s current location. Based on these values, Mapbox Camera can zoom to the user’s location whenever the GPS icon gets pressed.

At today's meeting, we had listed down some important tasks. The most prioritised task was listing down all requirements of Mapbox. We wanted to make sure Mapbox can provide all the services required in the application. The Android version of this application is using the web map service (WMS) provided by our company. For some reasons, on this iOS version, we have been using the Mapbox map service. However, just like the Android version, the iOS application is using web-feature-service (WFS) provided by the company.

Wednesday 23rd December 2020

The tasks for today were selected from yesterday's meeting. I am about to handle date and time data shown on the map-feature form. To be clear, a map-feature form is a modal box that contains all details about the selected map-feature. The map-feature form can include information about date and time it was created, note, category, etc. At that moment, the date and time format that was shown to the user is US format (YYYY-MM-DD). It needed to be transformed to European format which is "D.M.YYYY".

The task was done easily by using moment which is a popular library amongst React JS and React Native communities. For the rest of the day, I spent time to re-evaluate and test the application. There were still many pieces of codes that I did not discover and understand completely. I kept reading the codes and tried to memorize the data flow until I got more ideas about their purposes.

Thursday 24th December 2020

Christmas holidays.

Friday 26th December 2020

Christmas holidays.

Week 3 analysis

This week, I have learned many things about integrating Mapbox to React Native applications. I spent most of my time handling map annotations, GeoJSON data, and getting user's location. They are all new concepts to me, but I think they are very interesting things to work on. The work tasks brought me a lot of knowledge about both technology and geography. In this analysis, I will introduce GeoJSON data, Mapbox's useful components and Geolocation API.

GeoJSON is a standard format used to encode different geographic data structures. Like described in its name, GeoJSON was invented based on JSON data format. It has become

a popular data format used in many GIS technologies and GIS services (geojson.org). GeoJSON is maintained by a GeoJSON group of developers called “Geographic JSON working group”. GeoJSON supports different geometry types, including: Point, Polygon, Line String, Multi Line String, Multi Polygon. A GeoJSON data object must have at least three properties: type, properties and geometry. Geographical data must be transformed to the correct GeoJSON format to be processed by map services or mapping libraries, for example, Mapbox, Leaflet, etc. Below is an example of GeoJSON data object:

Figure 12. Example of GeoJSON data

Next, I think it may be good to share information about the most popular Mapbox components that are available for React Native. MapView is the main component in Mapbox, it is the container of the whole map and overlays and different layers (raster layer, symbol layer, heat map layer, line layer, etc). Camera is another important component in Mapbox. Its purposes are zooming to the set coordinates and displaying the visible part of the map to the screen. Next, MarkerView is where map-markers are created. We can generate map-markers by different components depending on our needs. One of the most popular components used to render map-markers are PointAnnotation and SymbolLayer. Each of them has its own strengths and weaknesses. We must consider carefully before applying one of them to the application.

When developing maps utilising applications, I realised that there are a lot of use cases where developers want to access and get the user's device location. In React Native, a good solution to get a user's location is using Geolocation API. Geolocation API is a popular library recommended by React Native team. The installation of this library requires some

modifications to Info.plist file (for iOS version) and to AndroidManifest.xml (for Android version).

3.4 Observation week 4 (28.12.2020- 01.01.2021)

Monday 28th December 2020

The goal of today is implementing UPDATE function for the map-feature form. At the moment, the application only handles GET requests to obtain all the map-feature types and map-features details based on the user's role. Because I had moved all the states, actions and operations to Redux, managing data would be much easier now.

I started working on UPDATE function by simply creating an update icon on the User Interface. I always start working on the simplest thing when implementing a new application's feature. The update icon will toggle the form's "disabled" mode from true to false. Initially, "disabled" mode was set to be true by default which prevents users from editing the form. Once the update icon is pressed, the form will become editable and a new button that allows the user to save data will appear. What I did next was passing the form value, form schema and a valid token to a method which parses received data from JavaScript object to XML format before calling the API. The reason for this data transformation is that my company's server only accepts a request body under XML form. According to Erik T.Ray (2001), XML is a self-describing language which helps maximize data usefulness by refining data to the most structured form.

By the end of the day, I had successfully transformed the data from JavaScript object to XML format. The transformed data must match the example request body that was documented on my company's Confluence Wiki site. This was the first time that I handled XML format. Despite having difficulties reading XML documents, I was satisfied that I learned a new thing today.

Tuesday 29th December 2020

Because the UPDATE function has not yet completed, I am planning to continue creating a method for handling API requests today. For any frontend or software developer, interacting with the server side cannot be easy without Postman. It is one of the most valuable tools that makes API development become easier. It provides the quickest way to test API requests without writing testing codes.

In yesterday's report, I mentioned that examples and instructions to make API requests are documented on my company's Confluence site. Following the document, I created a method for handling API calls with a correct XML body, header and a valid token. By the end of the day, the method for sending an UPDATE request has been done. I spent the rest of the day learning different Git commands that I haven't known before. Some of those are "git stash"

and “git stash pop”. “Git stash” command is used for storing uncommitted changes since the last commit and turning the current working directory to a clean one. “Git stash pop” command is used when the developer want to re-apply the stashed changes. Moreover, I learned how to change the commit messages that have been pushed to the remote branch.

Wednesday 30th December 2020

My goal for today is handling responses from the server after the data has been updated successfully or after the update request has been failed. The response data format that the application receives is XML document. Once again, I had to parse XML response to JavaScript object. As soon as the server responds that the UPDATE request is successful, an action to update the application status in Redux states should be dispatched. By contrast, if the server responds that the UPDATE request is failing, an action to change the application status that there is error when updating will also be triggered.

What about the User Experience? I created a Toast component to inform users about the application’s status. In React Native main component, every time the application status changes, “componentDidUpdate” will render a Toast showing a notification message. Toast is a useful component when we need to send notifications to users. I used to be confused between Toast and SnackBar components as both of them almost look the same. The difference between SnackBar and Toast is that SnackBar not only shows notification messages, but it also requires an action from the user. On the other hand, Toast only shows a message to the user and it will disappear within a few seconds depending on the set duration (Shan Shen 2018).

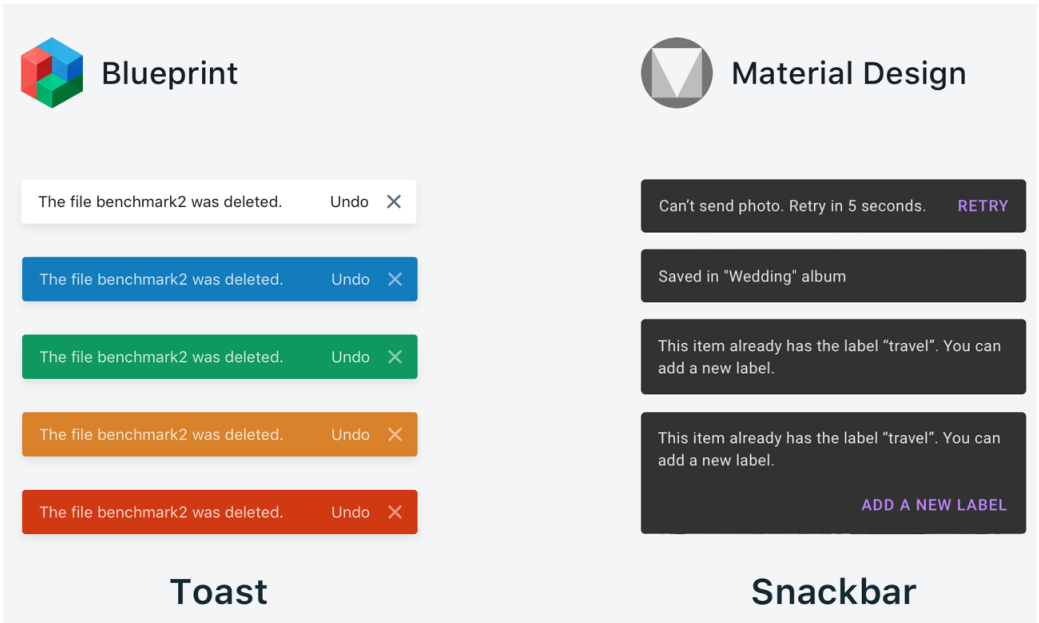

Figure 13. Comparison between Toast and Snackbar (UX Collective 2021)

Thursday 31st December 2020

I started my working day by reading the documentation from my company wiki about API requests again. I checked if I miss anything and I noticed that each geographical-feature type has its own request permission set. These permissions can be INSERT, QUERY, UPDATE, DELETE, etc. However, every map-feature type has different permission configuration. The application must send a GET request to get the feature's permission configuration before allowing the user to make UPDATE or DELETE requests.

During the rest of the day, I created another method to get permission data. The list of permissions was saved into an array. The component that takes responsibilities for rendering the map-feature details will get an array of permissions from Redux state. The permission data decides whether an UPDATE icon should be rendered to the screen or not. The work went smooth for me. I got the application working as expected by the end of the day.

Friday 1st January 2021

New Year holiday.

Week 4 analysis

This week I worked more in the User Interface design and API responses handling. I recognized that it is a good practice to notify users about the application current status after they have actions to modify the app's data. I found out there are a lot of ways to deliver notifications or messages to users, for example, we may use Toast to display a short message and disappear in a few seconds, or we can use Snack bar to send a short message and requires user's action, or we can use a modal box to pop up a long message.

The notification kind that I want to implement was in-app notifications, not push notifications displaying on the device's lock screen. Moreover, my target is to keep the notifications to be as simple as possible. And the notification may not distract users from using the app. Thus, I considered using one of the four common notification design patterns: Toast, Snack bar, banner and popup modal. A popup modal may interrupt users from reviewing data since it takes a lot of space on the screen. Therefore, I decided to consider between Toast and Snack bars. Both of them are temporary message modals that take a small amount of space on the screen. In the end, Toast was the component that I selected to use. Although it satisfies my expectation after being implemented, I found that Toast still has a small downside. It is pretty hard to measure how many seconds a Toast duration should be. If it disappears too fast, the users may miss the message. If it stays too long, it may distract the

user's view. What I did to improve is setting a long enough duration and creating a small close button on the right side of the Toast so that the user can close it if they want.

The other interesting thing that I learned this week is handling touch and gestures in React Native. React Native provides many types of APIs that allow developers to build touch-ready interfaces. However, iOS and Android may behave differently against user's interaction. Below are four kinds of Touchable components that are available in React Native:

S. No.	Touchable	Available For
1.	TouchableNativeFeedback	Only For Android
2.	TouchableHighlight	For Android/IOS Both
3.	TouchableOpacity	For Android/IOS Both
4.	TouchableWithoutFeedback	For Android/IOS Both

Figure 14. React Native touchable component (About React 2021)

In general, the TouchableHighlight and TouchableOpacity are more common than the rest. The TouchableHighlight component creates an overlay to appear on the view when it gets touched (About React 2021). The TouchableOpacity visually responds to the user's touch by changing the opacity of the wrapped view. Once again, according to About React (2021), TouchableWithoutFeedback components are not recommended to use unless the developer has a specific reason to use because it does not give any visual feedback when being touched.

3.5 Observation week 5 (04.01.2021- 08.01.2021)

Monday 4th January 2021

Since the UPDATE function has been done, I will continue testing this function with all available map-feature types today. The function worked well for most map-feature types. After updating the feature information from the application, I used Postman to double check if data has been sent successfully to the database. However, the UPDATE request failed for some certain map-feature types. Here was what I did to trace the bug: Firstly, from the terminal, I SSH into the Vagrant machine and check the logs from the server. The logs indicated that the version information is missing from the request body. To make sure that it was only the problem with the request body, I printed the request body that has been transformed to XML format and copied that to Postman. With this request body, the UPDATE request sent from Postman also failed. From those hints, I concluded that there is missing data in the XML request body. However, this was not the thing that I could fix by myself since I need more information from the backend team about that error.

Tuesday 5th January 2021

My goal for today is asking for help from the backend team about the UPDATE error. Then I can spend the rest of the day fixing that problem. I got the answer from the backend team that I need to provide a version number to UPDATE requests for certain map-feature types. And the backend team will add an example for that case to the company's wiki page.

The version data was easy to find if I sent a GET request to the server to obtain data of a single map-feature. The problem was that I was not sure where to add this version number to the UPDATE request body. Therefore, I added this debugging task to my to-do list and moved on to another task. During the rest of the day, I started researching about what services that Mapbox can offer. One of the most important services that Mapbox should satisfy our requirements is allowing polygon and polyline to be illustrated on the map from geographical data. I found an official document from MapboxGL that demonstrates how to do that. To make sure that I can draw both polygon, polyline and any symbol to the map, I created a new small project with the same React Native version and MapboxGL version as the project that I am working on. My work result after trying to draw polygon and polyline on Mapbox was demonstrated as below:

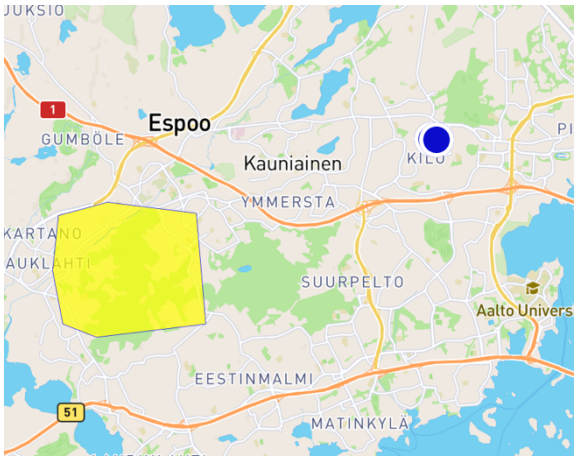

Figure 15. Polygon illustrated on Mapbox

Wednesday 6th January 2021

Epiphany holiday.

Thursday 7th January 2021

We had a plan that the application should be released to TestFlight today. That is the main reason that I came to the office since I need to work with another developer on the release. At the moment, staging and deploying are done manually because Jenkins pipeline has not yet been set up. However, since publishing an iOS application was what I had done before so I think the progress will be smooth.

Distributing an iOS application to TestFlight is something that any iOS developer has to do before publishing it to Apple's App Store. According to Apple official documentation (2021), the purpose of releasing to TestFlight is to let beta testers test the application on their own devices (Apple Store Connect 2021). In my case, the beta testers include people in my project's team, the company's owner and the manager. If a developer does not work for any organisation, the developer has to pay for an Apple Developer Account on his/her own. To start releasing the application, I need to config the application bundle from Xcode. However, if an application was not initialized by React Native CLI but by Expo, the whole releasing process can be done with command lines.

What challenged me in releasing this application was that I had to deal with Apple distribution certificates. Our goal is distributing this application under one team name, not by personal account. The progress can be frustrating and complicated if the developer does not understand how Apple handles certificates and bundles. What we did was share the same iOS distribution certificate from my colleague computer to my computer. The certificate was then added to my Keychain Access which allowed me to release the

application under my team's distribution certificate. The next step was archiving and sending the application to TestFlight. After that, we only need to wait for Apple to send emails that contain a "redeem code" for installing this application from TestFlight.

For the rest of the day, I managed to figure out how map-feature's forms are rendered dynamically. There are many available map-feature types on the database and each type has different forms. By understanding the operation, I can continue improving it in the future.

Friday 8th January 2021

The goals of today are learning how different map-feature forms are rendered dynamically and adding data validator when applicable. Before jumping into describing my work tasks, I will explain quickly about how the dynamic forms were generated. The basic idea is utilizing "tcomb-form-native" library. This library allows developers to generate a form with less code and more reusability. It means that we do not need to write a dozen input fields, but we just need to define a model for that form at once (Spencer Carli 2017). The Form component of "tcomb-form-native" receives three main values: options, value, type:

```
<Form
        ref=""
        type={}
        value={}
        onChange={}
/>
```

Figure 16. Example of the Form component of "tcomb-form-native"

The form renders the received data object into different input fields depending on their types. The diagram below demonstrates how a data object in this application is map into type, value and options:

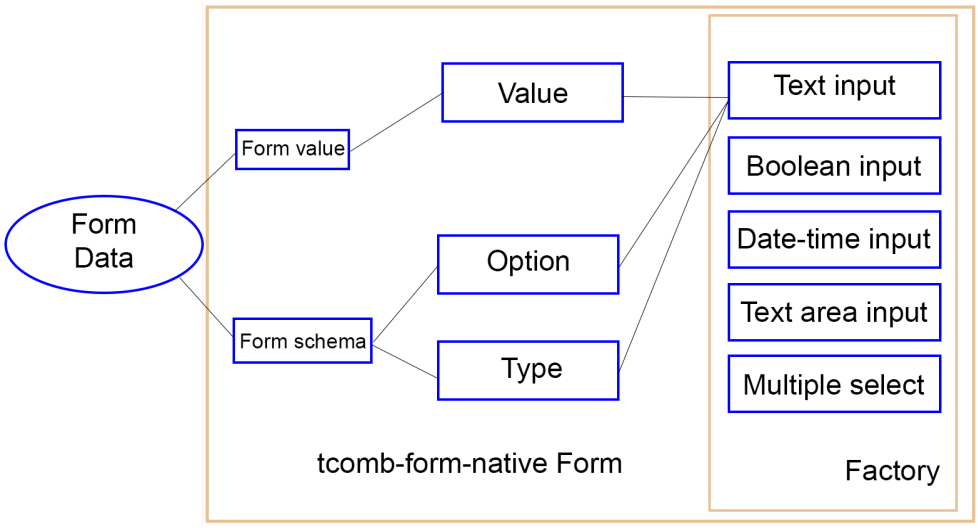

Figure 17. The use of tcomb-form-native to generate dynamic form

As demonstrated in figure 17, we can create different components called factories to render input fields. For example, if an input field is a text-input field, it will be handled by TextInputComponent. We can choose to use the default factory from “tcomb-form-native” or we can create our own factory. By creating our own factory, we can customize the input field’s style and template. The dynamic form is one of the most important functions in this application, however, there are still some missing parts that I am required to add in the future. One of them is adding a data validator if the form schema has “validation” property.

By the end of today, I have understood how dynamic form is handled in the application. There is still a lot of information about “tcomb-form-native” library on its official site that I need to learn. The second task of today was not yet implemented. I am planning to do it later when I understand thoroughly all perspectives of the dynamic form.

Week 5 analysis

This week, I interacted with the backend team and asked for their help about API request issues. Therefore, in this week’s analysis, I will share my thoughts about how to make better cooperation between frontend and backend teams. Even though working on the same project, the things that the frontend team and backend team has to deal with are completely different, so cooperation between two teams can be challenging. Here are my ways to overcome the communication barriers and interact more effectively. First of all, I always try to read the provided documents carefully before making questions and asking for help. Next, each person in the company has a different role and different tasks. What I did is try to understand how their tasks related to the overall project so that I know who I should contact to ask for support. Then, I try to explain my issue in a way that they can understand even though they do not use the framework that I use. By doing that, I can get support from the right person without waiting for an unrelated person.

Since I have started committing my codes to Github, I have learned that maintaining the best coding practices is a way to minimize confusion and help other people to understand the codes easier, especially when the pull requests that I make on Github must be approved by senior developers before they are merged into the master branch. In fact, the senior developers who are going to approve my codes only work on the backend side of the project, but they have to check codes from another application that they do not work on. Therefore, following standards and writing readable codes are important to make sure that my pull requests can be approved without causing confusion.

The application had been released to TestFlight on Thursday. What disappointed me was that the virtual keyboard overlays the app's view every time it appears. I almost forget to test the application with a virtual keyboard or test it with a real device before releasing. However, finally, I solved the problem by using the <KeyboardAvoidingView> component provided by React Native. This component makes sure that the view is pushed up when the virtual keyboard appears. However, the official documentation states that its behaviour can vary between iOS and Android. Therefore, when using <KeyboardAvoidingView>, a developer needs to configure the props of <KeyboardAvoidingView> to ensure it works fine on both platforms.

3.6 Observation week 6 (11.01.2021- 15.01.2021)

Monday 11th January 2021

The task that I selected for today is customising the symbol of map-features based on their defined styles. Each feature type has its own styles, and these styles are defined in the database. I can get the defined styles by making a GET request to the server. All styles can be found on a single API endpoint and they are described in the XML document.

The challenge of styling map-feature markers with data-driven property is choosing the right Mapbox's component. Basically, the Mapbox's component that is chosen to render our map-feature markers should meet two requirements: It can render different symbols (circle, rectangle, triangle), and it allows customising colours for those symbols. At the moment, CircleLayer is used to render map-feature markers. However, although CircleLayer allows customising colours, CircleLayer only renders a marker in circle shape.

At the end of the day, I had found three Mapbox annotation components that may satisfy the application's needs which are CircleLayer, SymbolLayer, PointAnnotation. I tried to use them and made a short table to compare their performances:

Usability/ Mapbox components	CircleLayer	SymbolLayer	PointAnnotation
Symbol sources	Native <MapboxGL.CircleLayer/>	Imported PNG images.	Symbols are generated by CSS.
Shapes	Only circle shape.	Rectangle, triangle, circle, icons, etc.	Rectangle, triangle, circle, icons, etc.
Performance	Fast, recommended by Mapbox team.	Fast, recommended by Mapbox team.	Slow (about to deprecate).
Z-index control	yes	yes	No, always on the top layer.
Filtered layer control	yes	yes	np

Figure 18. Mapbox components for rendering annotations

The table will be presented at tomorrow's meeting since I need opinions on what component to choose. Each of them has its own advantages and disadvantages, for example, PointAnnotation allows me to generate symbols with CSS. It is very easy to use because it

does not require GeoJSON data to be passed to its prop. However, the performance of PointAnnotation is slow, especially since the application has to render a large number of symbols to the map. By contrast, SymbolLayer is pretty fast but it only renders symbols under PNG image format. Additionally, SymbolLayer requires feature data to be transformed into GeoJSON data so that it can understand and render symbols to their correct coordinates on the map.

Tuesday 12th January 2021

The plan for today is clear, I will present about three Mapbox annotation component types in a weekly technical meeting and ask for people's opinions about which component I should use. Once a component is chosen, I will replace CircleLayer to the chosen component to customise the map-feature markers.

After the meeting, we decided to switch from CircleLayer to SymbolLayer. Choosing SymbolLayer generates more work for the backend team. However, we agreed that the application's performance is prioritized over any convenience. What the backend team will do is implement new API endpoints to return symbol images under PNG format. The symbol image URLs will be passed to SymbolLayer as a string. Before new endpoints are implemented, I will temporarily use local testing images in SymbolLayer.

Until the end of the working day, I have changed CircleLayer to SymbolLayer. What I learned from today is that a software developer needs presentation skill although it sounds fairly irrelevant to a programming position. The reasons are clear, firstly, it helps me to become more productive, secondly, my colleague can understand what I am doing and give me opinions if needed. From what I experienced from today's meeting, I understand that it does not matter if the listeners really know the technologies that I am working on, it is only about what I am trying to convey.

Wednesday 13th January 2021

Today's objectives are writing a GET request to get map-feature styles, the styles are defined in an XML document, so I need to transform the XML response into a JavaScript object to be used in this application. Although we understand that images will be used in SymbolLayer which means that we do not need those style data for symbol creation with CSS, the senior engineer still advised me to have those data in the application in case we want to generate styles using CSS in the future.

The XML document that I get from the server is complicated and difficult to read. But I figure out how to extract them into simple objects. When doing that, Lodash became a valuable tool for me since I have to deal with a lot of objects and arrays. I used Lodash mostly to handle deep cloning an object or to check if an object or array contains a specific property. During the rest of the day, I spent time mostly on Adobe Photoshop to generate some simple symbols images to be used for testing purposes.

Thursday 14th January 2021

Last week I mentioned that there was a minor error when updating map-feature form. I have not fixed that because I need to wait for the document about API requests to be updated so that I know exactly where to add the missing version number in the API request body. Since I am notified that the document has been updated so fixing this error is today's work task. For the rest of the day, I will reserve time to test the whole application again.

The debugging process went well for me, I added the missing data to the XML request body within a few minutes. When working more with XML, I realized that it has a lot of down sides. Firstly, the XML tags and syntax are verbose compared to other data transmission formats. The use of a lot of tags seems to be redundant and it increases the complexity of the data. Next, XML documents are obviously hard to read which leads to less productivity when working with it. Spending time to digest XML documents slowed down my working progress especially when I had to frequently convert XML data into JSON or JavaScript objects and vice versa.

Friday 15th January 2021

I have notified that new API endpoints have been implemented and I can start using server generated symbol images for SymbolLayer. The goals for today are updating my local development environment with the new added endpoints and applying those images to the Mapbox SymbolLayer. A meeting with a senior developer is also reserved since I need support to deploy new endpoints to my Vagrant.

The development environment was updated by those steps: Firstly, we need to pull the latest version of the server. Next, we may need to add Maven settings in the m2 file. Then we can install and redeploy the service by running the command “mvn -o clean install cargo:redeploy”. This command must be run in a direct directory, for example, “apache-maven-3.6.3/bin/mvn -o clean install cargo:redeploy”. Finally, if the database is updated, we must apply database changes with the command: mvn initialize Liquibase:update.

After the new service was applied successfully to my local environment, I started working on a bug that I found when testing the update function. The bug came from a third-party library called “react-native-sectioned-multi-select”. The application kept sending a message complaining that “subKey” is missing in “react-native-sectioned-multi-select”. The error was quickly fixed by upgrading the library version to its latest version. The bug actually comes from the library itself and it has been solved in its latest version.

Week 6 analysis

My skill in handling and converting XML data to JavaScript Object has become better. The converting progress itself can be done by using a third-party library called “react-native-xml2js”. But then, I have to extract usable data into a simpler object.

From last week's experiences, I realised presentation skill is as important as programming skill in software development. It supports my job more than I thought. Explaining and presenting technical concepts are not easy though. It requires a lot of practices to be able to talk about technical issues in a way that everybody can understand. Therefore, this is one of the soft skills that I am intending to practice more in the future.

The things that I had to clarify during this week are researching and selecting the most suitable Mapbox component to render customisable symbols to the map, deploying updated server to local virtual machine, debugging minor error with missing data in update request body, debugging error with the third-party library.

What bothers me the most during last week is the extensive use of third-party libraries in this application. There are many risks and vulnerabilities when an application is too dependent on libraries. Firstly, let's think about maintenance issues. I saw advertising on some library’s Github pages that they need maintainers for the library. The worst scenario that came to my mind is that the library may be abandoned by its own author or the library may be incompatible with new React Native versions. Besides, there are also issues with security and licenses when using third-party libraries.

When talking about installing third-party libraries to the React Native component, it would be good to share here what I have learned about the installing process. In most cases, after we run the “npm install <package-name>” command, we have to run “react-native link” command. The reason for that is this package requires changes to the underlying iOS and Android projects. The “react-native link” basically modifies the native files which could be AppDelegate.m (for iOS project) or MainApplication.java (for Android project).

React Native is a framework used to build cross-platform applications. However, it does not mean that we can always use the same codes for each platform without extra configuration. From my experience, if an application is intended to be used in both platforms, it should be tested carefully because one feature working on one platform may not work on the other. In the case that we have to handle platform-specific components, we need to have a parent component to wrap platform-specific components. Then, the parent component needs to present a unified API. One of the most common elements that usually has plat-form specific behaviour is “navigation UI”. The interaction patterns are very different between iOS and Android.

3.7 Observation week 7 (18.01.2021- 22.01.2021)

Monday 18th January 2021

The goal for today is applying server generated symbol images to SymbolLayer. The task seems to be simple but in fact, it is the most challenging task for me up to now. I spent almost half of the day applying the image URL to SymbolLayer without success. The image URL cannot be rendered even when I applied the path to <Image/> component of React Native.

After a few hours, I eventually figured out the problem thanks to Postman. Postman returned me the error message indicating that “access token required”. This message let me know that authorization is required in order to retrieve images from the server. This was the first time that I have to deal with an image URL that requires authorization. However, I quickly found the solution. In React Native, to provide token to Image component, we can add this token to the “headers” object, the codes should look like this:

```
<Image
    source={
        {
            uri: 'http://...',
            headers: {
                Authorization: 'Bearer ' + token
            }
        }
    }
/>
```

Figure 19. Example of render an image with Authorization Header in React Native

With the authorization values, the image is visible to the application if I use <Image/> component to render it. However, the bigger problem was that the “iconImage” property in SymbolLayer only accepts the image URL as a string so I cannot pass authorization header to the image URL:

```
<MapboxGL.SymbolLayer
    style={{
        iconImage: 'http://… ',
        iconSize: 0.35,
    }}
/>
```

Figure 20. Example of Mapbox’s Symbol Layer use

After researching, I found out many other developers have the same problem with specifying Authorization Header for a source in Mapbox. The method to add Authorisation Header for ReactJS's Mapbox (mapbox-gl-js) is "transformRequest". Unfortunately, this method cannot be used in React Native's Mapbox (react-native-mapbox-gl). I understand that using a JavaScript function that will be passed to Native code is impossible. Therefore, applying "transformRequest" method of "mapbox-gl-js" to "react-native-mapbox-gl" is not the answer. That is why I need to leave this problem and report about it in tomorrow's weekly meeting.

Tuesday 19th January 2021

I am planning to report yesterday's issue to the support team in today's meeting. A problem should always be updated as soon as possible so that everyone can understand my situation. I understand that keeping silent and delaying the work when encountering bugs or technical issues will cause confusion, especially when everybody is working from home and no one understands about the other's situation.

In the meeting, other developers confirmed that this issue had been recognised before. The problem with Mapbox is that it does not allow specifying Authorisation Header to external sources. Also, the developer who took care of this iOS application used to report that he could not apply a token to the external source used within Mapbox. That is also the main reason why my company's Web Map Service (WMS) was not integrated to this iOS app.

One of the backend developers will work on integrating the token into the URL so that I do not need to worry about passing Header to image URL. My task to customize map-feature styles can be delayed. Another task should be taken care while waiting for the server to be updated. Updating a server can take a long time though.

Wednesday 20th January 2021

The task for today is implementing the DELETE function for the map-feature form. As mentioned before, every map-feature type has its own request permissions. The method to get feature-type's permissions had been done before so I only need to work on DELETE function for now.

The work went well and it is pretty similar to the workflow of implementing UPDATE function. I did not have any trouble in completing the task.

Thursday 21st January 2021

Today is devoted to testing and cleaning the codes. I came up with a better way to handle the application status update. Thus, I decided to rewrite the codes to update the application status in componentDidUpdate method. Refactoring or reviewing codes is a good practice since it allows me to double-check the logics and optimise or shorten my codes.

What I did today was using the one single operation to handle server responses for both INSERT and UPDATE requests. The business logic to handle responses is the same so there is no reason to use two separate operations. In the end of the day, I got the application working as normal with better and shorter codes.

Friday 22nd January 2021

Because all tasks assigned to me have been done, I decided to find the solution for adding a custom Header to the external source used in Mapbox. During the day, I found a lot of cases and topics relating to my own issue. I also found an official documentation from Mapbox indicating that by using the “addCustomHeader” method, developers can configure sending custom HTTP Headers to the tile server. From the end of 2019, this method has been created to help developers pass header to their own tile servers. At that moment, I still did not know what a tile server was. After a while, I figured out a tile server is a service that returns map images or tiles. The tile server regularly requires large disk space for the rendered tiles. External tile services can be integrated into Mapbox and the added tiles can exist as a separate layer over the Mapbox map layer. The research about tile servers was interesting but I was not sure how it may help my work.

During the day, I made many attempts to add the authorization token for the image URL by using “addCustomHeader”. However, there was no success, the symbol images were not visible on the map yet.

Week 7 analysis

Despite having trouble in specifying Authorization Header for external sources in Mapbox, I have successfully implemented a new function to the application. However, as the previous task has not been completed, my brain keeps thinking about how to solve it on the client side even though the backend team mentioned that they will also be working on integrating the token into the URL. I believe my extreme concern towards this unsolved problem may turn into stress which is what I am trying to avoid. Therefore, whenever I encounter troubles from a task, I try to switch to another task. If the new task is done successfully, I will feel better since I remain working productively. From psychology perspectives, my brain

behaviour towards unfinished jobs is understandable. In fact, this behaviour is called the Zeigarnik Effect. This effect may impact our life negatively since “the stress of daily hassles and frustrations often stem from incomplete tasks (Mentalhelp 2021). Under the scope of this thesis, I will only list some strategies that helped me to reduce anxiety when the work task is undone. Firstly, we need to switch to a new work task when the current one is unsolvable. Next, we can divide the unsolvable task into smaller tasks and perform one by one. The most important thing is that we should understand our own limit.

During this week, I have learned more thoroughly the patterns to handle async requests and how to use Redux “thunk” middleware in handling Async logic. So far, I have been working well with using AJAX requests to get and update data from the server. In this weekly analysis, I will explain how to use Middleware to handle Async logic. In fact, the asynchronous operations only happen outside the Redux store and the Redux store has nothing to do with async logic, the store’s job is just synchronously dispatching actions (Redux Fundamentals 2021, Part 6).

When developing this application, I realised that there are many cases where I want to have asynchronous logics to interact with the store. What helps me to do that is using Redux “thunk”. “Redux thunk” is one of the most popular middleware which allows developers to write async codes to dispatch and check the store state. Let’s put it simple, “Redux thunk” is used when we want to call an action that returns a function instead of the action object. The “thunk” package can be installed easily by using either npm or yarn: “npm install redux-thunk” or “yarn add redux-thunk”. When writing a thunk function, we need to pass “dispatch” and “getState” as parameters and use them inside the “thunk”. Below is an example of writing a “thunk function”:

```
1  const thunkFunction = (dispatch, getState) => {
2      const initialState = getState();
3      dispatch(increment());
4      const updatedState = getState();
5  }
6  store.dispatch(thunkFunction)
7
```

Figure 21. Example of Redux “thunk function”

The example has not included Async logic yet, the pattern to handle Async requests in Redux normally looks like this: Firstly, we can create an action to indicate the request has been loaded before the request. The action is normally used to update the “isLoading” state in the store. Secondly, we can create the Async request. After getting the response from

the server, the Async logic dispatches either an action to handle “success response” or “failure response”. Then we can update the loading state.

```
const saveNewFeature = (token, itemData) => dispatch => {
  dispatch(actions.fetchingData());
  if (itemData) {
    api
      .saveNewItem(token, itemData)
      .then(response => {
        if (response.message == "success") {
          dispatch(actions.itemSaved(response, itemData));
        }
        if (response.message == "error") {
          dispatch(actions.saveError(response, itemData));
        }
      })
      .catch(err => {
        dispatch(actions.requestError(err));
      });
  }
};
```

Figure 22. An example “thunk” function demonstrating the pattern to handle Async request in Redux-thunk

3.8 Observation week 8 (25.01.2021- 29.01.2021)

Monday 25th January 2021

Last week I mentioned that DELETE function has been done, however, it can be dangerous to let users delete a feature right away without asking for confirmation. Therefore, the task for today is adding an alert modal box to ask for user's confirmation before deleting a feature. This task did not take much time for me since React Native has a built-in alert modal box that is pretty easy to use. The other task that I did was releasing the application to TestFlight. After the first release, the second release was much simpler. I did not need to handle the team distribution certificate anymore. The distribution certificate only needs to be configured one time. What I did to release the application this time was incrementing the build number of this application in Xcode, archiving the application and sending the archive file to TestFlight.

Tuesday 26th January 2021

In the morning, I did not have any special tasks. I have finished all tasks assigned to me. New tasks will be brainstormed and generated in today's meeting. Therefore, in the morning, I only searched and read topics relating to "How to specify a custom header for a source used in Mapbox" and "How to integrate external Web Map Service to Mapbox". As I mentioned in last Tuesday's diary, my company's own Web Map Service should have been included in the iOS app a long time ago. But due to the problem with passing a token for an external source used in Mapbox, it was not integrated.

In the meeting, I demonstrated what I have done last week and what I found when searching about "adding custom header" to a source used in Mapbox. As I mentioned in last week's diary, Mapbox developed a new method to pass custom header to tile server. When I mentioned this method, the senior developer told me that I can try to integrate the company's Web Map Service into Mapbox using that method. That could be the solution for passing a token into the requested server. But before discussing further about what I will do, it would be good to explain briefly about "tile server" and "Web Map Service". Basically, we can say that both of these terminologies refer to the same thing. Let's make it clear, according to the Land Processes Distributed Active Archive Center (2021), a Web Map Service (WMS) is a standard protocol developed by Open Geospatial Consortium to provide access to georeferenced map images over HTTPS requests. A Web Map Service provides a set of map layers. The figure below demonstrates WMS layers:

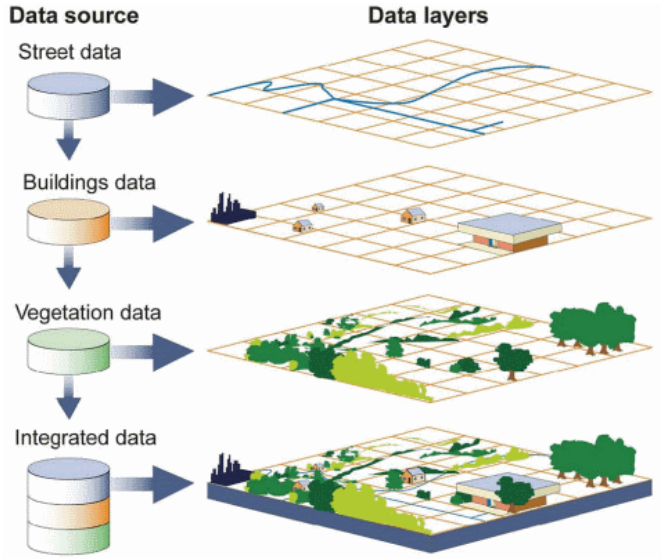

Figure 23. Web Map Service layers (Land Processes Distributed Active Archive Center)

After the meeting, I formed a plan to integrate my company's WMS to Mapbox:
Firstly, I passed the user's token to "addCustomHeader" method: *MapboxGL.addCustomHeader('Authorization', '{auth header}').*

To enable "addCustomHeader", we also need to call "initHeaders" in AppDelegate.m. Here are two mandatory lines that we must add to AppDelegate.m:

(1) Include the header file: *#import "MGLCustomHeaders.h"*

(2) Init headers, add swizzle method: *[[MGLCustomHeaders sharedInstance] initHeaders];*

Note that, addCustomHeader() should be initiated in the componentDidMount() method and it should be called every time we make a request to the tile server because we need to update the token. In addition, we can use removeCustomHeader() method to remove the header at runtime.

Next, I created a RasterSource layer to contain retrieved map layers from WMS. Finally, I passed the tile source to the RasterSource component under TileJSON form.

Here are the results after I added testing raster layers retrieved from Web Map Service:

Figure 24. Adding testing WMS raster layer to Mapbox

In the next example, I added another testing “water-colour” raster layer to Mapbox:

Figure 25. Adding testing water-colour raster layer to Mapbox

And this is the original map view without any raster layer:

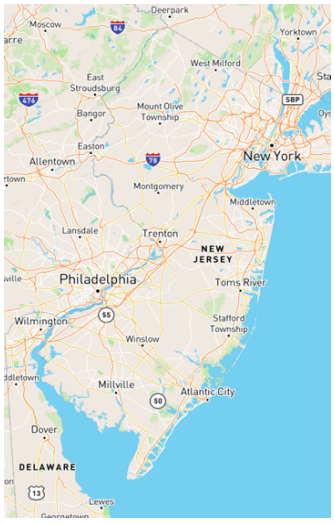

Figure 26. Demonstrating original map view

The Web Map Service servers used in these examples can be accessed without authorization header. Therefore, developers can ignore the first step which is passing a token to the request.

Wednesday 27th January 2021

Yesterday I figured out how to integrate my company's WMS into Mapbox. My plan for today is to get WMS capabilities. Like most OGC (Open Geospatial Consortium) protocols, Web Map Service allows developers to make GetCapabilities requests to get all available maps that the service offers. The GetCapabilities request returns a large XML document.

The XML document has three main sections. The service section describes the service abstract and contact information. The capability section lists all requests that the server can handle. The content section includes data about available map layers, each map layer has properties about its name, title, bounding box, etc. The table below lists all important properties of a map layer data and its purpose:

Properties	Purposes
Service	Name of the service
Version	The service's version
Request	The request type, in most case, the request is GetMap
Layers	Name of the layers, layers can be a list separated by comma
Styles	List of styles
Bbox	Determine the extent of the map
Width	Width of the output layer in pixels
Height	Height of the output layer in pixels
SRC	The coordinate reference system. At the moment, Mapbox only supports EPSG:3857
Format	The format of the output (it can be image/png, image/jpeg or others)

Figure 27. Map capabilities description table

Here is an example of layer data in XML document return by Web Map Service (note that I erased some important values to avoid exposing my company's data):

```
<Layer queryable="1" opaque="0" cascaded="1">
    <Name>              </Name>
    <Title>                     </Title>
    <Abstract></Abstract>
    <SRS>          </SRS>
    <SRS>          </SRS>
    <SRS>EPSG:3857</SRS>
    <LatLonBoundingBox minx="                  " miny="                 " maxx="                " maxy="                " />
    <BoundingBox SRS="          " minx="          " miny="          " maxx="          " maxy="          " />
    <ScaleHint min="0.0" max="1000000.0" />
    <Format>image/png</Format>
    <UpdateDate>                  </UpdateDate>
</Layer>
```

Figure 28. Example of layer data in XML document return by making a getCapabilities request to Web Map Service.

With the capabilities, we can make a request to get map layers, the request may look like this:
"http://example/wms?bbox={bbox-epsg-3857}&format=image/png&service=WMS&version=1.1.1&request=GetMap&srs=EPSG:3857&width=256&height=256&layers=example"

By the end of the day, I have finished getting WMS capabilities and made a request to get map layers available on my company server.

Thursday 28th January 2021

Yesterday, I made a pull request for one of the branches that I have worked on. I got a comment about the updated codes from one of the reviewers. He suggested that I change the way I compare two arrays in componentDidUpdate() method. Thus, this is one of the tasks for today. The other task that I want to do today is to extract codes in one component to a new component.

About the first task, the way that I used to compare two arrays was converting arrays into strings and comparing them as strings. On one hand, it is a quick solution especially when I want to make sure that the state should be updated as soon as two arrays (one from the previous state, one from the current props) are not identical. On the other hand, it may cause unnecessary re-renders if two arrays contain exactly the same elements but do not have the same element order. In addition, the code reviewers commented that it is not a good practice to compare two sets of values within a line of code, I should have a separate method to compare them because it will make it easier to read the codes. Thus, I created a method to compare two arrays by looping through them and check if they have the same elements. The task was done quickly, and I learned a lesson that it is the developer's responsibility to write “trustworthy” codes.

One of the components in this application has more than 900 lines of codes which is not a good practice. Therefore, I decided to split components that have too many lines of codes into smaller components. The task was not complicated but it takes time to re-organise everything.

Friday 29th January 2021

My goal for today is finishing the User Interface for map layers selection. As clarified in Wednesday's diary, I finally knew how to integrate my company's web map service into Mapbox. What I want to do today is creating a bottom modal to list all available map layers for the user to choose. The figure below demonstrates a bottom modal which is commonly used in map-related application:

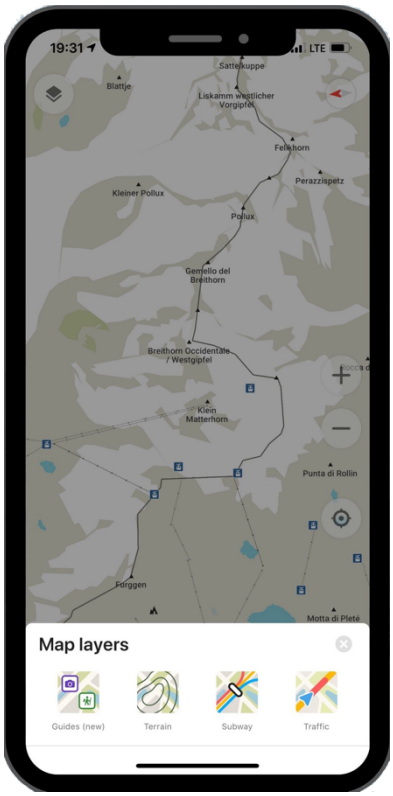

Figure 29. Example of using bottom modal in Mobile Application (MAPS.ME application)

In my application, the bottom modal was created to contain a list of available map layers for the user to choose. Once a list item is selected, the modal will be closed automatically to avoid distracting users from using the map. Bottom modal is widely used for map-related applications because it does not take a lot of space and it allows users to review the map while configuring the settings.

In the middle of the day, I had a meeting with the whole company. The meeting was called a "knowledge sharing" meeting where everybody demonstrated about the application they are working on. The purpose of the meeting was to motivate each developer to

communicate and cooperate with others. In the meeting, we discussed that all applications (both mobile and desktop applications) should look similar if they belong to the same application family. When it comes to Marketing and Branding, the feeling of users that applications on different platforms are developed by the same company and serve the same purposes are very important. In the weekly analysis, the branding perspective in UI design will be discussed more.

Week 8 analysis

The whole week was dedicated to integrating my company's Web Map Service into the application. The task used to be unsolvable because "react-native-mapbox-gl" does not allow specifying custom header to an external source used in Mapbox. But thanks to the new method- addCustomHeader, a valid token can be passed to the authorization header for authenticated tile server used in Mapbox now. Although addCustomHeader helps us to add custom headers to external tile servers, this method still has a drawback. In fact, addCustomHeader will add the specified header to every Mapbox tile request, not only a specific tile source. This behaviour is not good in terms of security because this leaks security data to Mapbox's servers and other servers that should not receive the headers.

While splitting large components into smaller parts on Thursday, I made a quick research about how to write clean and concise codes. Robert Martin mentioned in his book that "The first rule of functions is that they should be small, the second rule of functions is that they should be smaller than that" (Robert C.Martin 2008). In another chapter about writing clean classes, he emphasized again: "The first rule of classes is that they should be small, the second rule of classes is that they should be smaller than that" (Robert C.Martin 2008). Writing small functions and small classes is the key factor in maintaining clean codes. What about my application situation? At the moment, some of the main components in this application have over 900 lines of codes. Those long components are not readable and maintainable, especially when React Native's components have its vanilla JSX which tends to make the code even harder to predict and understand. When reading these long components, I realised that each of them has too many responsibilities and they contain many complicated modal boxes. What I did is just extract these modal boxes into separate components and import them back to their parent components. After all, the codes look a bit better. From this unpleasant experience, I learned a lesson that I should always keep the components small and make sure each of them does only one thing.

This week, when handling state updates, I finally understood why using nested objects in state is not recommended. In week 2 analysis, I mentioned some immutable approaches

that we can use to handle immutable state updates, those approaches are: using spread (…) operator, using Object.assign method to clone an object, etc. Those approaches only make a shallow copy of an object. If an object contains nested child objects, we have to copy all levels of those nested objects. Otherwise, those nested objects will only be copied by its reference instead of its value. Unfortunately, handling immutable updates to nested state can be complicated. The codes below demonstrate how a nested state is updated:

```
function updateNestedState(state, action) {
  return {
    ...state,
    first: {
      ...state.first,
      second: {
        ...state.first.second,
        [action.id]: {
          ...state.first.second[action.id],
          three: action.value
        }
      }
    }
  }
}
```

Figure 30. Applying immutable updates to nested state

As demonstrated above, the codes to update deeply nested state are hard to read and quite verbose. Handling immutable updates to nested state generates more chances to make mistakes (Manujith Pallewatte 2019). Moreover, deep copy may cause unexpected re-renders when React Native thinks that everything has changed while only a nested child object has changed. Thus, we should avoid deep cloning as much as possible.

On Friday, I mentioned the role of branding in UI design is something any developer should understand. In short, branding is a set of marketing techniques in which a company creates and develops its name, image, designs, symbols that make customers identify the company easily (Evan Tarver 2020). For a software company, the User Interfaces of their own products are the most powerful tool to create visual identity. The User Interface includes visual elements such as: logo, typography, templates, colour palettes, shapes, icons, etc. These elements decide how customers recognize and distinguish a company from other companies on the market. Because of the importance of brand identity in UI design, we have decided that all software products would have a unified interface to promote the company's image. The biggest challenge for us is that cooperation between different teams working on different software projects is difficult. In addition, while the iOS version of Mobilenote application has been developing, its Android and Desktop versions have been

developed so it will require extra work to change the UI for completed apps. Moreover, it will take time to maintain the consistency of UI elements throughout the application family.

3.9 Observation week 9 (01.02.2021- 05.02.2021)

Monday 1st February 2021

My plan for today is rendering read-only data to the feature form. I was asked to do this task in the last meeting since the beta testers recognized that the feature form was missing some fields. For some reasons, the application has only rendered editable data. This task could be the only task that I am about to take care for today since I think it would take time to figure out the data flow before adding new codes. As soon as I make read-only data visible on the form, I will work on the UI to make sure that read-only data looks different from editable data. For example, read-only data will have a grey background and the user cannot focus the cursor on these fields.

The work went well for me, I found the piece of code that filters out all data that has the "READONLY" flag. The filtering method stays in a "utility" component where any data retrieved from the server can be transformed into simpler objects or readable objects. Instead of filtering out, I changed the codes to make sure that both editable and read-only data are available in the Redux store. Those steps were not complicated, what really took time was filtering out read-only data when sending an UPDATE request. Because read-only data was not filtered out from the beginning, I have to get rid of it from the UPDATE request body, otherwise, the UPDATE request cannot be successful. The challenge was that the form's values were separated from the form's schema. And data indicating if a form's value is read-only data or not is kept in the form's schema. Below are examples of a form's value object and a form's schema that are separate from each other:

Figure 31. Demonstrating form's value object and form's schema object

Now it comes to the question of how I can check if an item in form's value object is a read-only item or not. There are many options to do that, I could choose forEach(), filter(), reduce(), map() or a simple for loop to go through all the keys of the values object. With each key, I can access the schema object and check if its "isReadOnly" prop is true or false. Amongst all those listed methods, I finally picked reduce() method since it is the shortest way to achieve my purpose.

Tuesday 2nd February 2021

The tasks assigned to me are almost done, I'm planning to double-check everything before today's technical meeting. Although everything is working normally, I am still concerned about the fact that there are so many things inside the componentDidUpdate() method of the main component. This method has been utilized to track and update the application's status. For example, when the application's status is "data_saved", a series of events and actions will happen. Not to mention there are different kinds of status beside "data_saved" so the codes inside componentDidUpdate() are completely long, complicated and hard to read. Therefore, once again, I need to spend time to re-organise and split this part into smaller functions. After the meeting, I will be assigned to other tasks and I can continue working on the selected tasks during the rest of the day.

What I was required to clarify in the meeting are: fixing UI issues reported by beta testers, implementing landscape mode, checking if it is possible to render the company's map layers without Mapbox's default map layer, researching about automating builds with Jenkins. Regarding the last task, I was not required to set up the hardware myself, but I need to know some basic steps, for example, compiling React Native application from the command line in a way that it can be deployed to TestFlight and AppStore. Because it is needed to write a script to make Jenkins handle this compiling.

I ended my working day with a long to-do list. Despite not having any new feature implemented today, I have learned many things related to setting up a Jenkins agent for React Native app.

Wednesday 3rd February 2021

Yesterday, I received a detailed report about the current problems with the User Interface and User Experience from the beta tester. Firstly, the tester mentioned that the application looks complicated to use as the user may not understand how to update a map-feature form. At the moment, a map-feature form is not editable if the user does not switch from

“view-only-mode” to “editable-mode” of the form by clicking on a pen icon. The other issue is that when the user enters the cursor on a text input field, the virtual keyboard will open up and cover the field where he/she wants to write. Next, when the user finishes filling a field and presses Enter, the cursor does not move to the next field programmatically. Moreover, the “save” button is located at the bottom of the form which is hard to reach. Fixing those problems are my tasks for today.

About the first issue, the reason that users are required to unlock “editable-mode” to be able to edit is that it helps avoid saving data unintendedly, for example, the user may open the form, press the keyboard by mistake and save the form without double checking. But since this case can be rare and the tester gave many good reasons for reducing the app’s complexity. The tester mentioned that this app is intended to be used in harsh conditions where there could be heavy snow while users are working outside transferring cars or cleaning the roads off snow. The app must be easy to use in "heavy snow with a big man's finger" - as one of our clients mentioned. Therefore, during the day, I decided to enable “editable-mode” of the form without asking the user to click any button.

The next issue is text input elements are hidden by the virtual keyboard. I fixed that by using the <KeyboardAvoidingView/> component of React Native. This component automatically adjusts the view’s position based on the keyboard height. The other solution for that is using “react-native-keyboard-aware-scrollview” library which is a helper for handling ScrollView’s content when the keyboard shows up. To fix the last problem, I decided to make the position of the save, update, delete button “absolute” to make sure that these buttons are not going to move along with the view scrolling.

Thursday 4th February 2021

I have to continue working on the User Interface today to make it look simpler to use. In yesterday’s diary, I mentioned that there are still a lot of issues with the User Interface. Handling styling can be frustrating sometimes, especially when there are no available designs for it. In this case, the developer may need to create UI prototyping, collect feedback and change the UI accordingly.

The work took me the whole day to finish. I got an issue that the fixed bottom footer was still overlapped by the virtual keyboard. Although React Native’s <KeyboardAvoidingView/> component has helped prevent text input fields from being overlapped by the keyboard, other elements that have an “absolute” position are still hidden when the keyboard shows up. The solution for this is to add an event listener to identify the native keyboard events.

The “event listener” can be initiated in componentWillMount() and it can be removed in componentWillUnmount(). With the event listener, we can call a function to detect the keyboard’s height and re-calculate the margin, padding, top, bottom, right, left attributes of “absolute” position.

At the end of the day, the issue that the keyboard overlaps the bottom footer was fixed. As a junior developer who used to work on full stack applications, I do not believe working on the frontend side can be easier than the backend side. The perception of many people that working on the front-end is easier than working on backend may be true on entry level when people can step into visual stuff and see the results right away. However, when it comes to API handling, state management, routers, component rendering lifecycles, application optimization, etc, the work can be complicated. While a frontend developer has to make sure that the application integrates well with the backend and takes care of the app’s performance and user experiences, the work tasks of a backend developer can be more straightforward and clearer.

Friday 5th February 2021

The first goal for today is making it possible for the cursor to focus on the next text field when the user presses on the ”Enter” key. The second goal is finding out if it is possible to use the company’s tile layers without Mapbox’s default map layer. At the moment, although we can use our own Web Map Service to render different map layers, raster layers and overlays, we still have to keep Mapbox’s default map layers as a base layer. I’m planning to test if I can remove this layer and render a blank view as the base layer instead.

For the first task, I could not resolve it today because I know it would be complicated. Normally, when we want to programmatically focus to the next text input, we can use “ref” to move the cursor (the picture below shows how to do it in a regular form), however, this application’s forms are dynamically rendered. We have separate components to handle rendering form fields based on their types. For example, we have: “TextInput” component to render all text input fields, “BooleanInput” to render all Boolean input fields, etc. Therefore, passing a ref to each input field can be complicated. I am intending to ask my team if this feature is really necessary.

```
return (
    <>
        <TextInput
            placeholder="Input1"
            autoFocus={true}
            returnKeyType="next"
            onSubmitEditing={() => ref_input2.current.focus()}
        />
        <TextInput
            placeholder="Input2"
            returnKeyType="next"
            onSubmitEditing={() => ref_input3.current.focus()}
            ref={ref_input2}
        />
        <TextInput
            placeholder="Input3"
            ref={ref_input3}
        />
    </>
)
```

Figure 32. Programmatically focus cursor to the next input field

About the second task, I found out that Mapbox's MapView is the container for all map layers, raster layers, etc. And it will always render the default Mapbox's map layer to the screen. We cannot remove the MapView component if we want to render maps. However, the interesting thing is that Mapbox's default map layer is always visible to the screen even when I remove Mapbox Access Token. It means that using this layer is free of charge and we do not need to have a Mapbox Access Token if we host our own vector tiles/ map layers or we use tiles hosted by third-party services. Mapbox Access Token is only needed if we want to use the Mapbox tool, SDKS, API, tile sources, styles, etc hosted by Mapbox's servers.

Week 9 analysis

Last week was a busy week for me. When testers started to give feedback, there were more tasks to handle which almost stressed me out. I usually worked overtime to complete my tasks. There were many times that I had to tell myself that I should stop this workday at least before 5pm. But then, I still pushed myself to work until night. I have learned that work-life balance in software development may be complicated. While the work time is limited, we always want to find out the solution quickly and we think that if we spend 1 more hour, we are going to "make it happen". Although my university teacher gave me many nice tips about maintaining work-life balance, when it comes to reality, I'm still too devoted to work because I wanted to compensate for my lack of work experience and to assure my productivity. Nevertheless, I realised that staying up late to work is bad for the brain which results in less productivity for the next workday. Therefore, I am trying to keep a consistent routine to end my workday and improve my work-life balance.

First of all, since I am still working from home during this pandemic, I normally did not have a mentally feeling that I am going to “leave the office”. Therefore, at the end of a workday, I try to close all open tabs on my browser, “halt” my virtual machine and close the running apps. If there are some important tabs that I want to reference on the next workday, I will create bookmarks to save them. On the next working day, I can start running up everything to begin a “fresh” workday. Although it’s quite time-consuming, these actions train my brain to not think about work tasks after the work time and allow it to take a break.

Secondly, when working remotely, most of us are more flexible with the schedule, however, it can make us unaware about the time we spend on working. Thus, I set boundaries for my start and end time. Normally, I try to start my workday before 9am and end my workday before 5pm. Next, as suggested on Flexjobs.com, it’s worth finding a third space where I can make a transition between work and personal life (Emily Courtney 2020). Regularly, when working in the office, we may have commutes from office to home in which we can get rid of stresses of the workday and mentally prepare for a shift to personal-life activities. But due to working from home, we may hardly have a transition from work to personal life. Here are some tips introduced by Flexjobs.com that may help me build a transition space when working from home: I may take a walk with my dog, I may physically leave my house for a while for any purposes, I may spend time to meditate, work-out and so on. I may choose any peaceful activities but pushing myself right into household chores in order to calm my brain.

The final thing to keep in mind when improving work-life balance is enhancing the sleep quality. In the book “Leader-shift- The work-life balance program” by Don Clayton, he mentioned that it is important to avoid any engagement with work and study immediately before going to bed and this should be considered as a personal policy (Don Clayton, 2005, 86). Although I sometimes fail in following the routines and policies mentioned above, I am trying my best to apply these into my every workday.

The thing that I learned during last week is understanding when to use filter, map, reduce, foreach methods. Although all of these methods serve the same purposes- iterating through a JavaScript array, executing a function in each item of the given array and returning a result. There are still a slightly difference amongst them, the comparison table below clarify the differences amongst them:

Methods	Return type of result	Number of elements in the result in comparison with the given array
.forEach	Return undefined.	
.map	Return an array.	Equal to the number of elements in the given array.
.filter	Return an array containing all elements that satisfy the conditions.	Equal or less than the number of elements in the given array.
.reduce	Can be anything.	Reduce transform an array into anything

Figure 33. Compare JavaScript array iteration methods

In brief, foreach method works like a for loop, it executes the provided callback function to each element in an array one by one. The next method is map, just like foreach, map executes the given callback function to every element in the array. However, unlike foreach, map returns a new array based on the given array (Manoj Singh Negi 2017). The third method is filter, it not only allows iterating through an array, but it also selects certain elements in the original array based on the given conditions and it returns a brand-new array filled with elements that pass the conditional checking. The final method and also the most interesting method to me is reduce. Just like its name, it reduces an array into a single value which could be a string or a number. This method holds two arguments- let's call them "sum" and "element". The "sum" is the final result that we get after the callback has run on each element in the array and added them into the "sum". Last Monday, I mentioned that I used the reduce method to solve my task, it is because I wanted to receive a single object after iterating the original array. In this case, reduce is the most elegant method since we do not need to write extra and verbose codes to transform an array into a new object.

In conclusion, although I got stressed out during this week, I have learned the hard way to control my work-life balance. For me, work-life balance is more about the mindset on how people can lead themselves to live a life that they want to live, rather than a life they have to live. Besides, I learned many things about JavaScript array iteration methods and choosing the most efficient method in a specific case. Next week, I am planning to research how to compile React Native iOS applications from the command line to support writing Jenkins pipeline scripts.

3.10 Observation week 10 (08.02.2021- 12.02.2021)

Monday 8th February 2021

Today I am going to handle the resetting state when the user switches from one account to another account. Although this scenario can be rare (every employee normally has only one account to use this application), handling resetting state on logout will make sure the application acts as expected. The next thing in my today's to-do list is switching from React Native Geolocation API to Google Location Services API. The reason for that is React Native Geolocation API is less accurate and slower than Google Location Services API (Emmanuel Oaikhenan 2021). The same statement has been made by React Native: "This API is not recommended by Google because it is less accurate and slower than the recommended Google Location Services API" (React Native 2021).

During the day, I have switched from React Native Geolocation API to Google Location Services API. The first task also went well for me. What I did was create a new action that will be dispatched to reset Redux states to its initial states as soon as the user logout. What challenged me today is that I noticed there is a bug in the previous branch, but I was not sure how to come back to this branch to edit and apply those changes into newer branches. The picture below demonstrates my Git branches:

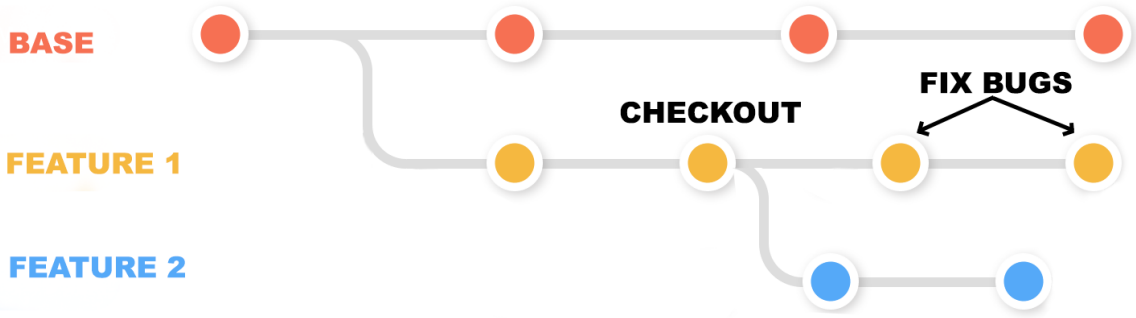

Figure 34. Working on multiple Git branches

To put it simply, I have been working on 3 branches: base, feature1, feature2. I checked out to branch "feature2" and wrote new codes here, but then I noticed there are bugs related to branch "feature1". I wanted to fix bugs on branch "feature1" and add those changes to the branch that I have been working on- "feature2". After searching for the best practices to do that, I found out there could be three strategies:

- I could use git cherry-pick command to pick the commit from branch “feature1” and apply it to “feature” branch after I fix the issues on branch “feature1”. The command should be:

  git checkout feature2

  git cherry-pick <commit-hash>

- I could switch to the branch I want to fix- branch “feature1”, create a new branch- let’s call it “fix-feature1”- to fix issues on branch “feature1”. After fixing the bug, I can merge “fix-feature1” into branch “feature1”, and then merge into “feature2”.

- The third strategy is using “rebase” to base the changes from one branch into the current branch. According to the official document of Atlassian (2021), “git rebase solves the same problem as git merge” (Git stash, Atlassian.com).

Both “git rebase” and “git merge” are created to integrate changes from a branch to another branch. There are many arguments about which command is better. Some people believe using “git rebase” will make the git tree structure look cleaner because it does not generate extra commits as “git merge” does. However, there are good arguments to use “git merge” since “merge” preserves all the changes history. Therefore, “git merge” is better for code reviews and teamwork.

Tuesday 9th February 2021

Yesterday I worked on resetting states in Redux stores and local states in React components when the user logout. Today I will double check the codes again to make sure everything is working fine. In addition, I am looking for the best practices and the correct way to reset Redux states. Yesterday, I mentioned that I have reset Redux states once the user logout, but I had to write multiple actions to reset states in different reducers. That is ok if there are two or three reducers, but there are five reducers in this application, and I do not want to repeat the same or similar codes for each reducer.

I found a lot of nice articles writing about resetting Redux State using a Root Reducer. Initially, I reset states in the Redux store by adding the RESET_ON_LOGOUT condition to four out of five reducers in this application. But those verbose codes do not follow the best practices- “Don’t Repeat Yourself”. Creating a Root Reducer can help me to centralize the resetting of state. The Root Reducer stays on top of other reducers. We need to import “combineReducers” from Redux in order to combine multiple reducers into a single Root Reducer. According to Redux official document (2021), “the combineReducers helper function turns an object whose values are different reduxing functions into a single reducing

function you can pass to createStore". The codes below are an example of how to create a Root Reducer:

```
import { combineReducers } from 'redux';
import AppReducer from './AppReducer';

import UsersReducer from './UsersReducer';
import OrderReducer from './OrderReducer';
import NotificationReducer from './NotificationReducer';
import CommentReducer from './CommentReducer';

const appReducer = combineReducers({
    /* your app's top-level reducers */
    users: UsersReducer,
    orders: OrderReducer,
    notifications: NotificationReducer,
    comment: CommentReducer,
});

const rootReducer = (state, action) => {
    // when a logout action is dispatched it will reset redux state
    if (action.type === 'USER_LOGGED_OUT') {
        state = undefined;
    }

    return appReducer(state, action);
};

export default rootReducer;
```

Figure 35. Example of creating Root Reducer in Redux

As demonstrated above, "undefined" is assigned to the state, it is because reducers will return to initial state whenever "undefined" is passed to the first argument (Asher Cohen 2019). In this case, the whole reducer tree will return to initial states when the user logs out. However, in my case, I just wanted to reset some specific reducers, so I have to use "destructuring" to maintain references to Redux store's properties that we want to exclude from resetting. The codes would look like this:

```
const rootReducer = (state, action) => {
    if (action.type === 'USER_LOGGED_OUT') {
      const { users, comment } = state;
      //Exclude this from being reset
      state = { users, comment };
    }

    return appReducer(state, action);
  };

export default rootReducer;
```

Figure 36. Exclude a specific reducer from being reset

Wednesday 10th February 2021

Here are the tasks that I want to do today: checking the last time if the application's states are cleared when user logouts, filtering out deleted map-feature symbols on map view (or make the deleted map-features symbol invisible on map view). I mentioned before that if a feature was deleted, it should not be shown on the map anymore. Although I have implemented that already, however, at that time, I only filtered one single deleted feature, I should filter out all deleted features in case the user deletes multiple features at the same time.

In order to solve the filtering deleted features task, I added a state to hold all deleted feature's ID in an array. Then, I changed the filter expression in Symbol Layer to make sure that Mapbox understands that it should filter out a list of features instead of one feature. The initial filter expression that I used is:

filter={['all'], ['==', 'gml:id', this.state.deletedFeatureID]}

This expression means that Mapbox should render all features, except the one that has an ID which is equal to the "deletedFeatureID" saved on state.

Below is the expression that I changed to:

filter={['!', ['match', ['get', 'gml:id'], this.state.deletedFeatureID, true, false]]}

This expression means that Mapbox should render all features, except features that have an ID matching one of the IDs saved on the array.

At the end of the day, I have completed all tasks. The most challenging thing is understanding how to write appropriate expressions that Mapbox can understand. This is pretty new to me, I had to spend time to read the document carefully and select the right expression to use.

Thursday 11th February 2021

My plan for today is fixing the bug of multiple choices input fields and releasing a newer version to TestFlight. I actually detected the bug a long time ago, but it's not very urgent to fix and I was too busy with other tasks. Let's me describe the bug, the multiple choices input field is supposed to allow user to choose multiple items at once, and then, it should combine values of item's objects into one single string separated by hash marks- "#". At the moment, multiple choice input fields work normally when I insert a new map-feature, but when I update the map-feature, it does not save values of the item's objects. Instead, it saves the ID of the item's objects.

At the end of the day, the issue was fixed. But I haven't had enough time to release a new version to TestFlight. That will be handled tomorrow.

Friday 12th February 2021

Today, my task was researching how to set up CI/ CD in React Native. After a day of reading different tutorials, blogs and documents, I summarized basic steps on how to set up CI/CD for React Native project. Generally, CI stands for Continuous Integration. It is a process of merging code in small increments to prevent integration problems. The process of integration: Developer makes add new features and makes pull requests, the code will then be compiled and reviewed. Once the codes are approved, it will be merged.

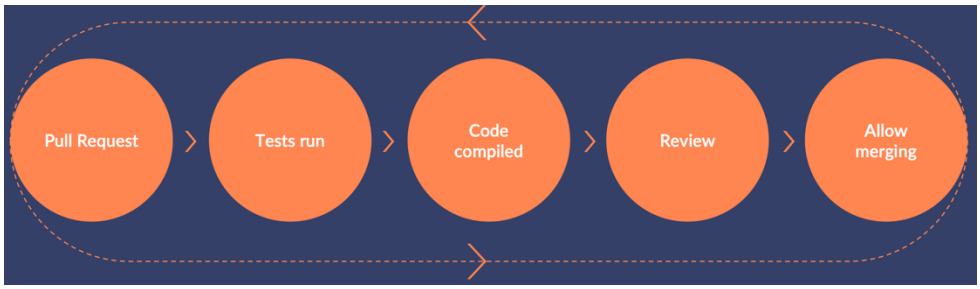

Figure 37. The process of integration (Juha Linnanen 2019)

CD stands for Continuous Development. It is a process of automating the deployment process to release the product and get feedback faster.

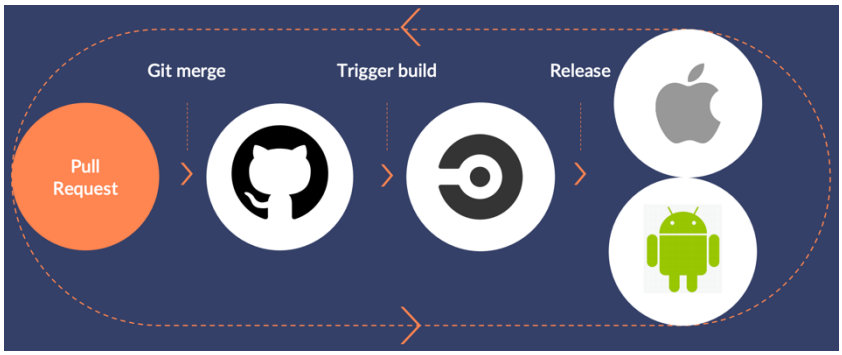

Figure 38. Continuous Deployment (Juha Linnanen 2019)

Normally, when setting up CI/CD for a project, the development team usually builds 3 flavours: Dev, Staging and Production. Anytime new codes have been pushed to the master branch, a newer version of Dev will be released to the store, once the features look good to go, we can push the codes to the staging branch which will trigger automatic build to the staging environment. And once everything is working well in the staging, we can be confident to publish the production version. But before setting up CI/ CD, we need a cloud

service for automatic triggering, some popular services are Circle CI, Travis, AppCenter, Gitlab CI/CD, Nevercode, Bitrise, etc.

Here are the basic steps to set up CI/CD for React Native project:
Firstly, we need to integrate CI/CD server to version control. Secondly, we need to configure CI/CD server pipelines. Below is an example of a config file (config.yml):

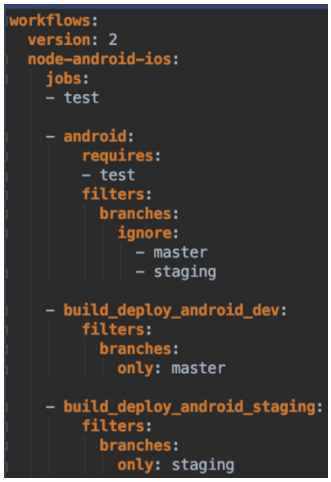

Figure 39. Example of CI/CD config file (Juha Linnanen 2019)

Next, we can add build flavours to the app. There will be a difference between iOS and Android, in Android there is better support for product flavours. In iOS, we have to create our own schemes and build configurations and we need a library called “react-native-schemes-manager”. The library manages all dependencies needed in new schemes that we created. On the React Native side, if we want our codes to work differently and depending on the flavour we have, we need to use a library called “react-native-config” to create environment file for different flavours.

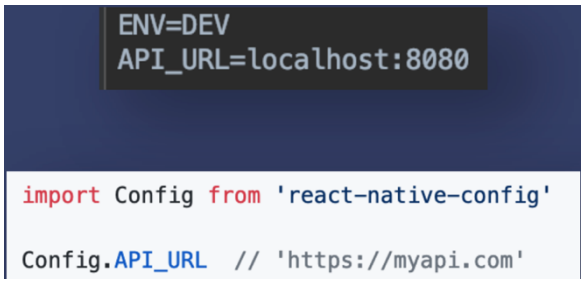

Figure 40. Example of using environment variables (Juha Linnanen 2019)

To be able to share secrets or certain passwords or secrets we do not want to commit to repositories. We can share them with cloud service by using private storage or encoded variables. The next step is to set up app signing and certificates. In Android, we can use the Keystore and Keystore property file which has passwords and secrets. In the

build.gradle file, we can add signing config. In iOS, we need to use Fastlane matches which can generate signing and certificates and store them in a private key top repo. Handle version numbering is another important step: We need to auto increment version numbers every time we build a new version, otherwise, the App Store will not accept a build that has the same version number. In addition, we can use different icon badges for dev and staging builds. With Fastlane plugin, we can add icons with extra information to differentiate between build version, staging version and production version. Finally, we can release the Deployment to Apple App Store.

Week 10 analysis

This week, I have learned a lot of new stuff regarding git best practices to modify older branches and apply changes to current branches. I have also learned about using the conditional expression of Mapbox. Using Mapbox is complicated sometimes because it has its own syntax and rules. The most important thing that I learned is the basic steps to setup CI/CD for the React Native project. There are multiple tools created around React Native to support the automating deployment process. One of them is Fastlane- an open source platform that supports both Android and iOS deployment. On Friday, I tried to use Fastlane and I successfully released a new beta version to TestFlight with just a command. Fastlane saves a lot of time for developers as it handles code signing, certificates, compiling, archiving, etc. Normally, those tasks are handled manually from Xcode, but Fastlane allows developers to write a script with defined actions. We can find all available action types from Fastlane official website.

As I have learned how to automate the compiling and releasing process, next week, I am intending to learn how to create different environments or flavours (development, staging, release) in React Native app. There should be something like a properties file that contains all environment variables, for example, API URL, Bundle ID, etc. With that, I can switch between different environments without changing the codes. This is an important task because once the Jenkins agent is set up, it should be able to automatically integrate and build the application for different environments.

Although there are many new things to learn, I managed to teach myself by reading documents, watching tutorials, learning from online courses, and reading books. I have read books more recently. I actually did not read words by words, but I just scanned the table of contents to find the topic that I'm interested in. Books often reveal some perspectives that I haven't thought about before. For example, I haven't known that we can integrate persistent database functionality with Realm, Async Storage, SQLite, etc. Actually, there could be

more options to store and retrieve data locally. Regarding my work-life balance management, I managed my time and schedule better this week. I did not work overtime on any day except Friday. Keeping a consistent schedule and sleeping early helped me work more productively the next day.

3.11 Observation week 11 (22.02.2021- 28.02.2021)

Monday 22nd February 2021

My goal for today is to configure the React Native iOS project for three build environments: development, staging, production. In addition, I want to automate compiling and releasing processes from the command line. Currently, on Apple's iTunes Connect site, we only have one testing version published for internal testing. We want to have a production version here besides the beta version so that two applications can be installed to the same device. In the future, once Jenkins agent has been set up, it can compile and distribute beta testing and production versions automatically. But before writing a script to automate compiling and releasing processes, I must make sure that the pipeline can dynamically switch between different environments (or targets). At the moment, managing different environments is done manually which means that I have to change the API host to the correct one before uploading the app. In addition, I have to manually increment the build number, version number from Xcode, revoke certificates, archive the application to Zip file, etc. Those repeating tasks are pretty time consuming and complicated, but this will be resolved soon.

In the end of the day, I have successfully created three schemes for this project. I have installed "react-native-config" to manage environment variables so that I can choose a build environment from the command line to run the app on simulators. In addition, Fastlane has been integrated into the app to simplify the deployment process. I will continue to work on the Fastfile to define more automatic actions that are necessary for the releasing process.

Tuesday 23rd February 2021

Now that I have integrated Fastlane, I am planning to add extra actions to Fastfile to handle specific actions for alpha build, beta build and production build. For the rest of the day, I will document step by step on what I did to automate the compiling and releasing process so that I can review in the future when needed.

Before describing what I did today, I will introduce Fastlane. According to Fastlane official document (2021), it is "an open source platform aimed at simplifying Android and iOS deployment, Fastlane lets you automate every aspect of your development and release workflow". Fastlane use "lanes" to manage actions, normally, there are three major lanes: alpha (development), beta (staging) and store (release) (Faiz Mokhtar 2018). The structure of a Fastfile looks like this:

```
fastlane_version "2.66.2"

default_platform :ios

platform :ios do
  before_all do
  end

  desc "Create a new Alpha build"
  lane :alpha do
  end

  desc "Create a new Beta build"
  lane :beta do
  end

  desc "Create a new App Store build"
  lane :store do
  end

  after_all do |lane|
  end

  error do |lane, exception|
  end
end
```

Figure 41. Fastfile structure

After config Fastfile with different lanes, I can run “Fastlane” command to see available lanes in my project:

```
+--------+-----------+-----------------------------+
|              Available lanes to run               |
+--------+-----------+-----------------------------+
| Number | Lane Name | Description                 |
+--------+-----------+-----------------------------+
| 1      | ios beta  | Push a new beta build to    |
|        |           | TestFlight                  |
| 2      | ios store | Create a new App Store build |
| 0      | cancel    | No selection, exit fastlane! |
+--------+-----------+-----------------------------+
/usr/local/Cellar/fastlane/2.174.0/libexec/gems/highli
ment as keyword parameters is deprecated
[12:01:55]: Which number would you like run?
```

Figure 42. Fastlane’s available lanes

As mentioned in yesterday's diary, I already added some actions to Fastfile so that every time I run “Fastlane ios beta”, it will automatically execute those actions. The actions that I added are: Ensure the current git branch is “master”, ensure the git status is clean, obtain certificates and provisioning profile to invoke to the app, fetch the latest build number and increment the build number. For the beta specific lane, I added those actions: update app identifier, update info.plist file, build, archive and upload to TestFlight. The action that I added today is generate an overlay/badge to reveal the app version and build number. This badge will make it easier for developers and testers to identify the app version:

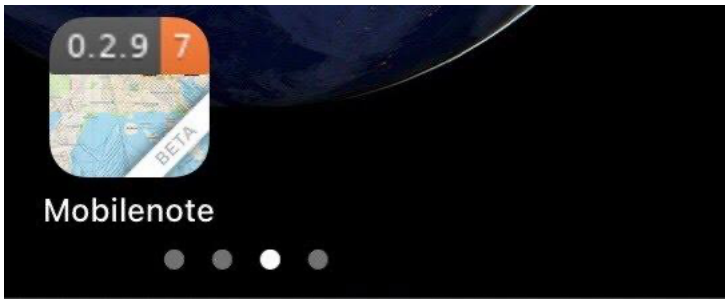

Figure 43. App's icon with badges

In order to add badges to the app's icon, we must install Fastlane's badge plugin (Fastlane 2021). Then we should define the action of generating badges in Fastfile. Here are the codes that I write to dynamically add app version to the app's icon:

```
#Add badges
private_lane :add_badge_to_icon do |options|
  version_number = lane_context[SharedValues::VERSION_NUMBER]
  build_number = lane_context[SharedValues::BUILD_NUMBER]
  if options[:release] == "alpha"
      add_badge(
          shield: "#{version_number}-#{build_number}-orange",
          no_badge: false,
          alpha: true,
      )
  elsif options[:release] == "beta"
      add_badge(
          shield: "#{version_number}-#{build_number}-orange",
          no_badge: false,
      )
  end
end
```

Figure 44. Private lane to add badge to icon

Those codes create a private lane that handle adding version badges to the app's icon. In order to use that, we must call this lane inside the lane that we want to use. In my case, I call this private lane inside my "beta lane" with this line of code:
add_badge_to_icon(release: "beta")

Another important thing is that we must install ImageMagick in order to use Fastlane's badge plugin. If everything is setup correctly, Fastlane will return a table of actions that it has executed after we run Fastfile's script:

```
+------+--------------------------------+-------------+
|                fastlane summary                      |
+------+--------------------------------+-------------+
| Step | Action                         | Time (in s) |
+------+--------------------------------+-------------+
| 1    | default_platform               | 0           |
| 2    | get_certificates               | 5           |
| 3    | get_provisioning_profile       | 2           |
| 4    | latest_testflight_build_number | 4           |
| 5    | increment_build_number         | 0           |
| 6    | update_app_identifier          | 0           |
| 7    | update_info_plist              | 0           |
| 8    | build_app                      | 261         |
| 9    | upload_to_testflight           | 117         |
| 10   | clean_build_artifacts          | 0           |
| 11   | notification                   | 0           |
+------+--------------------------------+-------------+

[20:47:20]: fastlane.tools just saved you 7 minutes! 🎉
nhiduong@Nhis-MacBook-Pro mobilenote-ios %
```

Figure 45. Fastfile summary table

Wednesday 24th February 2021

Yesterday, I read many tutorials about the creation of build variants of targets to share the common code base between different environments. And I just noticed that there are two options to do that in iOS: The first option is creating a new target for each pipeline by duplicating the main target and adding a new scheme for the newly created target. In the picture below, there are two more targets that I created for staging and development environments which are MobilenoteStaging and MobilenoteDev. The main target-MobileNote was saved for the production environment.

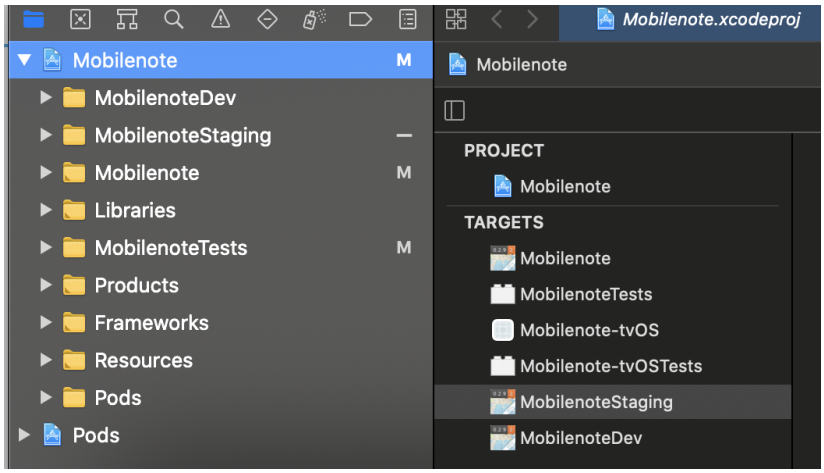

Figure 46. Different targets in Xcode project

The second option is adding iOS build configurations and schemes for the main target without adding new targets. It means that instead of creating a new target for each scheme, we can add multiple schemes for the same target. We can do that by duplicating the default “Debug” and “Release” configurations respectively. After that, we can add new schemes

and make sure that the new scheme uses its own build configuration options that are replicated from the initial configurations.

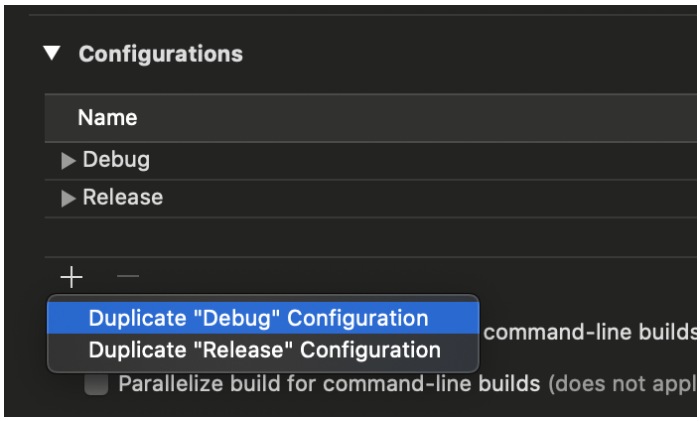

Figure 47. Adding configurations for the same target in Xcode

On Monday, I created three schemes for the project using the first method. It is working pretty well now. The project has three targets and three schemes for each target. My goal for today is to try the second method because I want to make sure that I understand the pros and cons of each method. I could not find any article or blog writing about which method is better, so I will experience it by myself today.

Just like the first method, the second method requires a few steps of manually configuring the project in Xcode including: replicating configuration options, adding scheme, configuring Info.plist file, adding Bundle ID Suffix to Bundle ID to make sure that Bundle ID is unique. The extra step is to add a “react-native-scheme-manager” library to add build configuration to all libraries in the Xcode project. After the day of working on the second method, I can run the project with the selected scheme from Xcode. However, I could not run the project with the selected scheme from the Command Line. I will continue to find the problem when I have time.

Thursday 25th February 2021

Yesterday, I spent time implementing the other option of adding build variants. But after configuring everything, I can only run the app from Xcode, not from the Command Line. However, I need to move on to the next task today, I may come back and try this option again when I have time. The task for today is uploading the application to TestFlight and testing the new added features. There are a lot of tickets in the “Staging” column waiting for approval. They will be marked as “Done” once the application is working fine on real devices and testers approve them.

This time, the staging version of this application has been configured to use the “staging server”. This server is normally used by external testers who are our customers. I haven’t used this server before because it requires a few steps to add Keycloak configuration for this application in our staging server. In short, Keycloak is an open source Identity and Access Management. It allows the user to authenticate via Keycloak so the application does not need to handle login form and storing users. The Keycloak configuration for this application in the staging server was done yesterday so I can start using it from now on. During the day, I have tested all new features before adding other testers to download this application from TestFlight and test it.

Friday 26th February 2021

My plan for today is to continue testing the application that I uploaded to TestFlight. Once everything is working fine, I will create a new branch and start implementing camera accessing and photo taking tasks. Every map-feature form has been allowing users to upload photos from Camera Roll now, but it should also allow them to take photos directly from their phone’s camera and upload photos to the server. The use case for that is when the cleaning staff or car moving workers want to take photos as proofs for their finished tasks, they can choose to take photos directly from their phone instead of loading photos from photo storage.

When testing the app with multiple accounts, I recognised that the app crashes when I login with some specific accounts. Since I could not see the logs from the device, I have to test it from the simulator. The logs showed “Internal Error” message which means that it is server’s fault not client’s fault. But in order to avoid app crashes, I managed to create a new screen to show error message instead of “throw error”. This will let user understands what is happening in case there is server issues. I did report this error to backend team so that they can check that. After making sure that 500 status error can be handled, I started learning how to access camera in React Native. I will continue working on that on next week.

Week 11 analysis

This week, I learned one of the most important aspects in software development which are handling different app flavors/ environments and automating the deployment process. I was truly struggling in the beginning since I did not have experiences about that. I read tons of articles and blogs on how to set up CI/CD and how to manage staging, production environments. After one week of working on these tasks, I am more confident about what I am saying now. I understand the overall picture as well as the detailed steps to set up Fastlane, config schemes and generate environment variables, etc. The blogs and tutorials

from the Internet recommend different roadmaps and techniques to setup environments and automate the deployment process. But some of them were written years ago and may be outdated now because React Native changes a lot since then. Because of that, I only followed the latest blogs and tutorials. I will explain the roadmap to setup environments and automate the releasing process that I used. And I will also explain the second option to serve the same purpose.

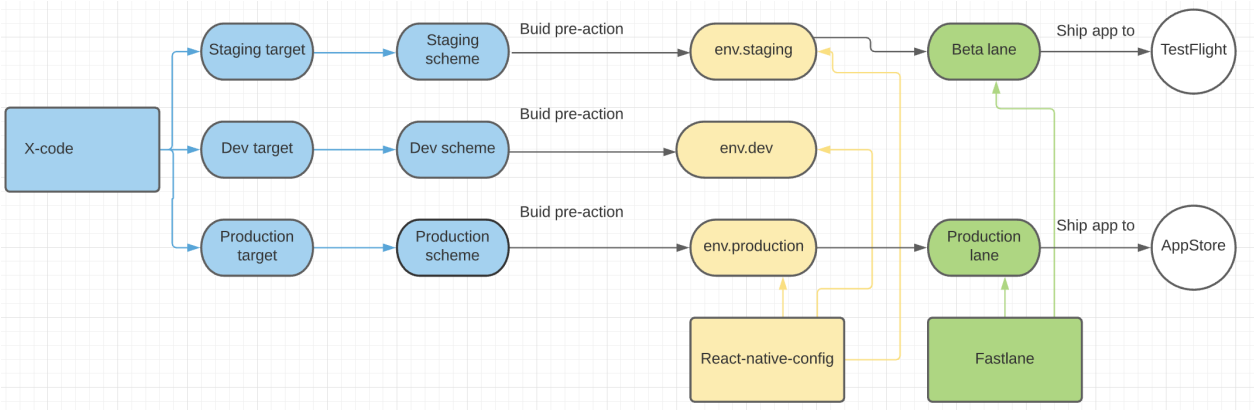

Figure 48. The Roadmap to setup different environments for React Native iOS project

The three environments that I created are Development, Staging, Production. The Development environment is where I use my local API host and develop new features without interrupting the users. The Staging environment is where the staging server is used. Staging versions will be shipped to TestFlight so that testers can download and test it. The Production version is the one that will be submitted to Apple App Store.

As demonstrated in the picture, the first step to do is creating three separate targets by duplicating the main target. We can create a different Bundle Identifier for each target so that we can upload all app versions to TestFlight and run them in the same device. A Bundle ID helps iOS and MacOS to recognize updates to the application and it can be changed at any point (Microblink 2021). Each new target will generate an Info.plist file automatically. Therefore, we have to handle three Info.plist files when developing new features. For example, if we develop a feature that requires access to the user's camera, we have to add the key and string purpose of accessing the user's camera in all three Info.plist files.

The second step is duplicating the main scheme so that each target has its own scheme. A scheme is a blueprint for the whole build process. It tells Xcode which configuration to build, run and test the app. What we must do next is creating environment files for each scheme. An environment file contains information about API host, app’s display name, version, etc which will be used by a specific scheme. Thanks to react-native-config library, storing and

accessing environment variables is much easier. Once the environment files are ready, we can define pre-action for the Build process in each scheme. The pre-action will tell Xcode to run a specific environment script that we just created with react-native-config.

Setting up Fastlane is the final step. We can define different lanes for each version. In most cases, we may want to have a beta lane (lane :beta) and a production lane (lane :store). The alpha lane is available, but I don't see I need to use this because the alpha version uses a local API host so uploading this version to TestFlight does not make sense.
For the second option, we need to config it quite differently in Xcode. However, the following steps are the same. The below pictures show the roadmap to setup environments and Fastlane if we choose to implement the second option:

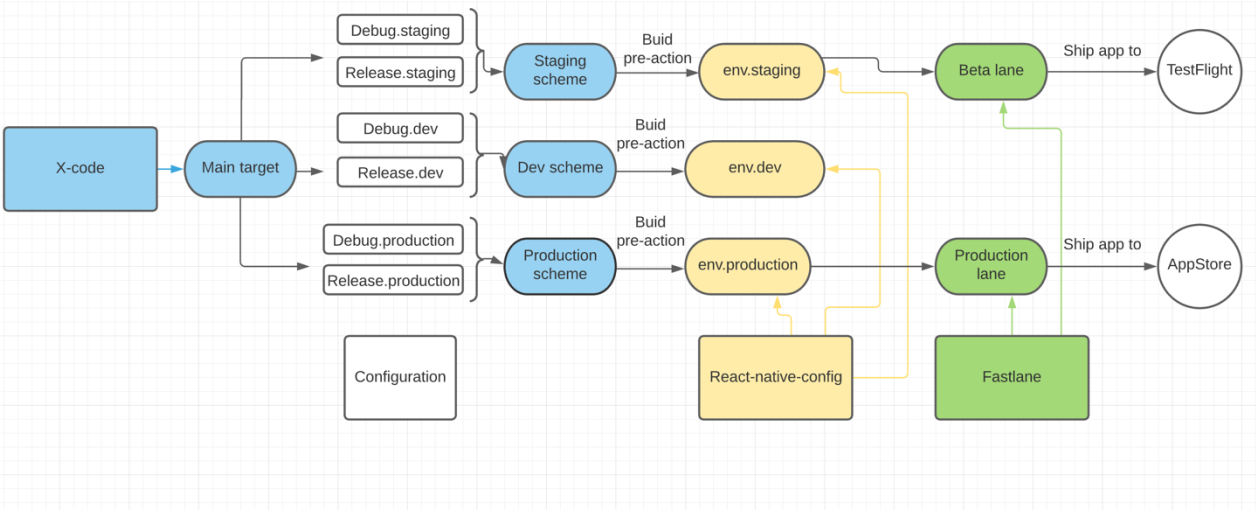

Figure 49. The Roadmap to setup different environments for React Native iOS project (option 2)

I explained quickly about the second option on Wednesday. With this roadmap picture, I can explain more thoroughly. In the main target, we have to replicate the set of configurations (Debug and Release) so that each scheme can use their own configurations. Then, we must install a library called "react-native-scheme-manager" to make sure that the replicated configurations will always copy the value from initial configurations (React Native Schemes Manager). After that, just like the first option, we can create schemes for each environment and each scheme will use their own set of configurations. How about Bundle Identifier? We need different unique Bundle IDs to be able to upload both staging and production builds to App Store Connect. It is very simple to create new Bundle ID for each target when we implement the first option. In this option, we have only one target which means that we have only one Bundle ID, let's say, the Bundle ID that we have is: "com.haagahelia.fi.mobilenote". We need to create user-defined "Bundle ID Suffix" for each

scheme to generate a unique Bundle ID for each of these schemes (Ronit Kadwane 2020). With the ID Suffix, we can change the value of Bundle ID into $(PRODUCT_BUNDLE_IDENTIFIER)$(BUNDLE_ID_SUFFIX). That's how we generate a brand-new Bundle ID for each scheme of the same target by utilizing Bundle Suffix ID.

Scheme	Bundle ID Suffix	Generated Bundle ID
Development	.dev	com.haagahelia.fi.mobilenote**.dev**
Staging	.staging	com.haagahelia.fi.mobilenote**.staging**
Production	.production	com.haagahelia.fi.mobilenote**.production**

Figure 50. Example of using Bundle Suffix ID to generate unique Bundle ID for each scheme

After using both the above options to setup environments, I can tell the pros and cons of each option. For me, the first option is more straightforward and easier to implement. However, we must keep in mind that any time we have to modify Info.plist files, we need to take care of all Info.plist files of all targets. The advantage of the second option is that there is only Info.plist file since we only have one target. However, when it comes to changing configurations of each environment, it will be quite complicated and messy.

Let's talk a bit about Fastlane. Setting up Fastlane is quite simple. However, we have to design our own patterns of actions. There are several actions listed on Fastlane's website that we can choose. If we are not sure what an action is going to do with the app, it may be dangerous. Yesterday, I tried to use the "reset_git_repo" action in "after all" lane. It means that I requested Fastlane to remove all changes made by Fastlane after it runs the Fastfile script. I found this action useful because I don't want to keep the app icons with overlaying version numbers generated by Fastlane in the app folder. The problem is this action removed everything including newly generated files, modifications it made to update Info.plist, xcworkspace and xcodeproj files. It is a drastic action that I did not expect. But after all, this is my own fault when not reading the document carefully before applying this action. Fastlane's official doc actually mentioned that "reset_git_repo" will not only discard files but it will also "reset your git to a clean state, discarding any uncommitted and untracked changes".

You may notice that there is a missing piece in the CI/CD setup picture. The missing piece is Jenkins. After lanes and build options are ready, we need Jenkins to automate Fastlane build flows. I am not a specialist in setting up agent hardware and creating remote access to cloud source control. But I pushed myself to learn about that a bit. Jenkins will be set up by my colleagues, and we will collaborate at some points to create Jenkins jobs. There may

be two Jenkins jobs to handle staging and production build flows. For each job, we may have to create a script, set up Git access, pass Fastlane parameters, refresh node modules, execute shell for each build phase. That would be another challenging journey for me since I am the main person who is responsible for that.

In conclusion, there are still a lot of things to learn, and of course, there will be challenges. I found myself becoming more independent and active in this role. I know when to seek help and what issues I should bring to the table to discuss in technical meetings. When I need to carry out a big task, I understand that I should start from the overall picture before breaking this task into smaller one so that I can implement them one by one. That's what I did to solve the "Automating deployment for different environments" task. If you read my diary, I can see that I did not go into details about how I should configure Xcode to have different versions of the app with the same code base and so on. What I did first on last Friday was watch a workshop video introducing CI/CD implementing a roadmap from react-finland.fi. The roadmap is introduced and implemented by a company called Gofore. The speaker does not go into details about "how" we do this and that, he mentioned "what" we need to do. And this introduction gave me some hints about what I should look up next. If I started by looking into details, for example, about how to create and access environment variables, I could be lost. I actually tried to look up for that, and there are several suggestions to setup environment files and access environment variables (For example, besides using react-native-config, we can use react-native-dotenv, or we can create JSON files to include environment variables and utilizing node script to switch environment file according to the used environment).

4 Discussion and conclusions

At the beginning of the job, I had only 6 months of work experience. At that time, I was very unconfident about my technical skills. I understand that the knowledge that I gained from studying at school is not enough to handle complicated work tasks, so I put a lot of effort in learning from online courses, bootcamps and Internet tutorials. Even though having 6-month work experiences, every day at work was extremely stressful to me. The most stressful period was the first two weeks when I had to learn the old codes and the system architecture. Nevertheless, after three months of working, I have grown a lot.

When I started my job, I listed down some main goals which I want to achieve. Firstly, I want to write cleaner and maintainable codes. Secondly, I want to get familiar with the git workflow when working with a team. Thirdly, I want to use Hook in this React Native app instead of using Class. In addition, I want to understand Web Map Service and Web Feature Service to handle my job. Moreover, I want to know how to set up CI/CD by myself. Finally, I want to use Redux more effectively.

So far, I have achieved 5/6 goals. I am always aware that codes written by Junior or fresher developers like me may not be as clean as senior developers, so I try to refactor my codes after every time I complete a new feature. By doing that, I can double check the logic and re-write the codes if necessary. About the second mission, although I was struggling in using Git to work with my team in the beginning, in the end, I understood more thoroughly about git workflow and git version control. In the third goal, I wanted to write a new component using Hooks since it allows developers to reuse stateful logic easily. But I have not been too motivated to use it because I am more familiar with classes. React's official document states that “Hooks work side by side with existing code so you can adopt them gradually”. It means that I can start to use Hooks and I do not need to change complex existing class components to Hooks. I will try to complete this mission in the upcoming months. From now on, I will practice “thinking in Hooks” and start using it. Next, I have been working well with Web Map Service and Web Feature Service and Redux. Last but not least, I have set up a part of the CI/CD system for this React Native app.

I believe that I have made a lot of changes and improvements to the project, especially to map-related features. There are not so many documents and available detailed tutorials about using Mapbox in React Native app, so I have to make my own research for every Mapbox component that I use. In brief, the most outstanding solutions that I brought to the project are: comparing different components to render annotations in mobile map view and choose the most suitable one for each use case, integrating my company tile server to the

application with Custom Header method provided by Mapbox, researching about Mapbox and alternatives to make sure that the selected Map provider can fulfil all requirements of the company, applying Redux thunk to handle async logic to interact with the Redux store, handling multiple environments for the project to speed up CI/CD setup progress.

The diary thesis turns out to be a good chance for me to review my working progress. I usually make quick notes about the commands, syntax, technical issues and their solutions, etc. But I have never spent time to document thoroughly what I did, how I came up with the final solutions. Writing the work diary thesis allowed me to analyse how I find out the solutions, how I made mistakes, what mistakes that I regularly made, etc. This is great to write down all of those because it helps me to remember the mistakes and avoid them in the future.

Moreover, when writing the diary thesis, I have an opportunity to set and review short term and long-term goals for my work. I have not ever thought about long-term goals for my career, but writing this thesis forced me to think about it. Needless to say, the short-term goals are my daily work tasks. The long-term goals are more abstract. They are not just about which frameworks or language I should follow in the future, but they are also about career preparation and plan: How I want to see myself in the future? Should I keep working on mobile development? Should I follow technical trends such as Machine learning and data science? etc.

The interesting issue with “react-native-mapbox-gl” is that it is not possible to pass a header to a title request except when we add Custom Header to componentDidMount(). However, the big problem of this is that passing authorization header to Custom Header can leak security data to other servers. We cannot choose to pass a token to a specific server, using Custom Header means that we will add the specified header to all servers.

I was limited to work mainly on the Frontend side in this job. But I want to work as Full-stack developer to get good fundamentals about all aspects of the system. I used to work with PHP and MySQL to develop the backend side for the application in my previous job. And I am really interested in working on the backend stuff. Furthermore, I want to broaden my knowledge in Cloud computing services such as Microsoft Azure, Amazon Web Services, Google Cloud, IBM Cloud, etc. I believe they are the future of software development. Many businesses have started to switch to using Cloud because it provides a better way to handle data centre operations. Not to mention that cloud services help businesses to reduce a huge amount of money on maintaining equipment, facilities and employing developers to build

their own data centre and fixing potential issues related to downtime. One of my first steps to learn more about Cloud computing is getting Cloud certificates. From Microsoft or Amazon websites, I can see that there are a lot of free courses that train people about Cloud computing from basic to advanced levels.

The other thing that I really want to learn is Linux. I will spend time exploring Linux OS since it will help me get closer to the lower level of the system and understand how computers work. At the moment, I just do not know where to begin learning about Linux. I think the possible path is to install it to my Virtualbox and learn the basic commands. I think that would be incredibly interesting to learn.

With the analysing of my own work, I am able to review my working progress and understand every perspective of the problems that I encountered. The work analysis gave me the sense of accomplishment. Since I wrote down my daily work targets in daily diary entries and weekly evaluations at the end of every week, I can see how much work was done and how my skills have been improved. When working in the company, I was not really given feedback about how fast I have been completing my tasks, so analysing my own work is good for self-tracking and self-evaluation.

I made a lot of comparisons amongst different methods, Mapbox's components, React Native's components in this thesis. These comparisons either come from official documents or from my own experiences. To be more specific, I compare Expo and React Native CLI, Mapbox's annotation rendering components (Point Annotation- Symbol Layer- Circle Layer), React Native's touchable component (TouchableNativeFeedback, TouchableWithoutFeedback, TouchableOpacity, TouchableHighlight), setup multiple environment methods in React Native (multiple targets or one single target), React's mapping methods (filter, map, reduce, foreach methods), React Native's list rendering methods (ScrollView, FlatList, SectionList, ListView), different ways to handle Immunity, mobile UI design patterns (Bottom/top tab bar, Hamburger menu, Floating icon, Toast, Snackbar, etc).

For me, these comparisons are priceless. I will need to review them in the future. If the comparisons were taken from official documents, there will be references for them and they are the most trustworthy information. If the comparison were rooted in blogs and articles, I will experience myself to check if I have the same opinions as the author. In case the comparisons were based on my own experiences, I will soon recognize if they are correct once I gain more experiences.

The diary also allowed me to concern more seriously about my work-life balance and career plan. In every week evaluation, I review how good I am in dealing with pressure and how I manage to end my workday with a positive mind. In conclusion, making analysis for my work task brought me a lot of values, it gave me a chance to document both technical things and psychological issues arising when dealing with stress.

References

About React 2021. React Native Touchable- 4 different type of touchable. URL: https://aboutreact.com/react-native-touchable/. Accessed 23 February 2021.

Amah, D. 2020. Guide to Adding Point Annotations on Mapbox View. URL: https://www.codementor.io/@danielamah/guide-to-adding-point-annotations-on-mapbox-view-1a0pbgc8i6. Accessed: 26 December 2020.

Apple 2021. Beta Testing Made Simple with TestFlight. URL: https://developer.apple.com/testflight/#:~:text=TestFlight%20makes%20it%20easy%20to,by%20sharing%20a%20public%20link. Accessed: 23 February 2021.

Atlassian 2021. Git stash. URL: https://www.atlassian.com/git/tutorials/saving-changes/git-stash. Accessed: 29 December 2020.

Avi, E. 2020. React Native: Scroll View and Flat List. URL: https://dev.to/eidorianavi/react-native-scroll-view-and-flat-list-2d7m. Accessed: 7 December 2020.

Babich, N. 2017. Basic Patterns for Mobile navigation: Pros and Cons. URL: https://www.smashingmagazine.com/2017/05/basic-patterns-mobile-navigation/. Accessed: 7 December 2020.

Carli, S. 2017. Easily Build Forms in React Native. URL: https://www.reactnativeschool.com/easily-build-forms-in-react-native. Accessed: 23 February 2021.

Ceddia, D. 2018. Immutability in React and Redux: The Complete Guide. URL: https://daveceddia.com/react-redux-immutability-guide/#how-to-update-state-in-redux. Accessed: 17 December 2020.

Chakraborty, P. 2020. Automatic Deployment of React Native iOS Apps with Fastlane and Github Actions. URL: https://www.gitstart.com/post/automatic-deployment-of-react-native-ios-apps-with-fastlane-and-github-actions. Accessed: 16 February 2021.

Chinda, G. 2020. PropTypes in React: A complete guide. URL: https://blog.logrocket.com/validating-react-component-props-with-prop-types-ef14b29963fc/. Accessed 16 December 2020.

Clayton, D. 2005. Leader-shift- The work-life balance program.

Cohen, A. 2019. React — Resetting Redux State with a Root Reducer (bonus: how to reset state selectively). URL: https://medium.com/@asher.cassetto.cohen/resetting-redux-state-with-a-root-reducer-bonus-how-to-reset-state-selectively-e2a008d0de61. Accessed: 9 February 2021.

Courtney, E. 2020. 8 Tips to End Your Day While Working From Home. URL: https://www.flexjobs.com/blog/post/ending-your-workday-remote-employee/. Accessed: 02 February 2021.

Eisenman, B. 2017. Learning React Native- Building React Native Mobile Apps with JavaScript. 2nd edition. O'Reilly Media.

Fastlane 2020. Badge. URL: http://docs.fastlane.tools/actions/badge/#badge. Accessed: 23 February 2021.

Fastlane 2021. Fastlane actions. URL: https://docs.fastlane.tools/actions/. Accessed: 16 February 2021.

GeoJSON 2021. URL: https://geojson.org/. Accessed: 23 February 2021.

Geoserver 2021. Introduction to Web Map Service. URL: https://geoserver.geo-solutions.it/educational/en/pretty_maps/wms.html. Accessed: 27 January 2021.

Guerra, J. 2018. Multiple Schemes and Configurations in a React Native iOS App. URL: https://medium.com/@guerrix/multiple-schemes-and-configurations-in-a-react-native-ios-app-fb1812b940c8. Accessed: 16 February 2021.

Güting, R. 1994. An introduction to spatial database systems. The VLDB Journal.

Hakulinen, R. 2018. Why you should replace forEach with map and filter in JavaScript. URL: https://gofore.com/en/why-you-should-replace-foreach/. Accessed: 2 February 2021.

Hartfield, S. 2020. Understanding the React useMemo Hook. URL: https://www.digitalocean.com/community/tutorials/react-usememo. Accessed: 17 December 2020.

Herrera, E. 2018. Immutability in React: There's nothing wrong with mutating objects. URL: https://blog.logrocket.com/immutability-in-react-ebe55253a1cc/. Accessed: 19 December 2020.

Herrera, E. 2018. Using Immutable.JS with Redux. URL: https://redux.js.org/recipes/using-immutablejs-with-redux. Accessed: 17 December 2020.

Hupe, A. 2020. The problem with snackbars and toast messages. URL: https://adamsilver.io/articles/the-problem-with-snackbars-and-toast-messages/#:~:text=Snackbars%E2%80%94also%20known%20as%20toast,disappear%20after%20a%20few%20seconds. Accessed: 30 December 2020.

Idaszak, D. 2019. Improve React Native performance with immutability. URL: https://blog.logrocket.com/improve-react-native-performance-with-immutability/. Accessed: 17 December 2020.

Ighodaro, N. 2020. Why uses Redux? A tutorial with examples. URL: https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/. 11 December 2020.

Jackson, M. 2017. Adding multiple target pipelines for React Native apps and Fastlane CircleCI deployment pt1. URL: https://medium.com/@jacks205/adding-multiple-target-pipelines-for-react-native-apps-and-fastlane-circleci-deployment-pt-1-ae9590ae52f2. Accessed: 18 February 2021.

Julian, M. 2020. Setting up Multiple Environments on React Native for iOS and Android. URL: https://medium.com/swlh/setting-up-multiple-environments-on-react-native-for-ios-and-android-c43f3128754f. Accessed: 16 February 2021.

Kadwane, R. 2020. Configure multiple flavor/schema for React Native Apps. URL: http://blog.logicwind.com/adding-multiple-target/. Accessed: 16 February 2021.

Kim, E. 2017. Tackling HOC in React Nattive. URL: https://medium.com/@bosung90/use-higher-order-component-in-react-native-df44e634e860. Accessed: 5 December 2020.

Ledesma, M. 2019. React Native: Best Practices When Using FlatList or SectionList. URL: https://dev.to/m4rcoperuano/react-native-best-practices-when-using-flatlist-or-sectionlist-4j41. Accessed: 7 December 2020.

Linnanen, J. 2019. React Native CI/CD. URL: https://slides.react-finland.fi/2019/juha-linnanen.pdf. Accessed: 12 February 2021.

Magalhães, M. 2020. Comparing JS iteration methods (map, filter, forEach, reduce + loops). URL: https://dev.to/manumagalhaes/comparing-js-iteration-methods-map-filter-foreach-reduce-loops-le. Accessed: 2 February 2021.

Martin, R. 2008. Clean code: A handbook of Agile Software craftsmanship. Prentice Hall.

MetalHelp 2021. The Zeigarnik Effect and Completing Everything. URL: https://www.mentalhelp.net/stress/the-zeigarnik-effect-and-completing-everything/. Accessed: 23 February 2021.

Microblink 2021. Application Identifiers: Bundle ID And Package Name, URL: https://help.microblink.com/hc/en-us/articles/360021533574-Application-identifiers-Bundle-ID-and-Package-name#:~:text=The%20bundle%20ID%20can%20be,ID%20as%20it's%20unique%20identifier. Accessed: 16 February 2021.

Mokhtar, F. 2018. How to improve your beta deployment flow with Fastlane Badge plugin. URL: https://medium.com/@faizmokhtar/how-to-improve-your-beta-deployment-flow-with-fastlane-badge-plugin-289e2ea5d2bb. Accessed: 16 February 2021.

Morales, A. 2018. Resetting Redux State with a Root Reducer. URL: https://www.digitalocean.com/community/tutorials/redux-reset-state-redux. Accessed: 9 February 2021.

Muller, N. 2020. Automate your React Native App with Fastlane. URL: https://levelup.gitconnected.com/automate-your-react-native-app-with-fastlane-ea516b4a893. Accessed: 16 February 2021.

Negi, M. 2017. Array Methods Explained: Filter vs Map vs Reduce vs Foreach. URL: https://codeburst.io/array-methods-explained-filter-vs-map-vs-reduce-vs-foreach-ea3127c6d319. Accessed: 2 February 2021.

Nhammad 2020. Mark a Coordinate on Mapbox Map in React Native. URL: https://medium.com/javascript-in-plain-english/mark-a-coordinate-on-mapbox-map-in-react-native-5d21c71ed46e. Accessed: 26 December 2020.

Oaikhenan, E. 2021. React Native geolocation: A complete tutorial. URL: https://blog.logrocket.com/react-native-geolocation-a-complete-tutorial/. Accessed: 23 February 2021

Pallewatte, M. 2019. Deeply Nested Objects and Redux. URL: https://www.pluralsight.com/guides/deeply-nested-objectives-redux. Accessed: 23 February 2021.

Particle41 Team 2020. How to set up Fastlane to manage multiple build environments. URL: https://particle41.com/insights/fastlane-automation-multiple-environments/. Accessed: 16 February 2021.

Pate, T. 2018. Running iOS builds — Part 3, React Native DevOps Guide. URL: https://medium.com/@tgpski/running-ios-builds-part-3-react-native-devops-guide-4b1fdc40b49. Accessed: 16 February 2021.

Pate, T. 2018. Setting up a Jenkins agent— Part 1, React Native DevOps Guide. URL: https://medium.com/@tgpski/setting-up-a-jenkins-agent-part-1-react-native-devops-guide-4c8b763b0961. Accessed: 2 February 2021.

Piyadigama, D. 2020. Expo vs React-Native-CLI. URL: https://dinukapiyadigama.medium.com/expo-vs-react-native-cli-7a3019b2760d. Accessed: 16 February 2021.

Ray, E. 2001, Learning XML. O'Reilly Media.

React Native 2021. Geolocation, URL: https://reactnative.dev/docs/geolocation. Accessed: 23 February 2021.

ReactJS 2021. Higher-Order Components. URL: https://reactjs.org/docs/higher-order-components.html. Accessed: 5 December 2020.

Redux 2021. Redux Fundamentals, Part 6: Async Logic and Data Fetching. URL: https://redux.js.org/tutorials/fundamentals/part-6-async-logic. Accessed: 25 January 2021.

Rooney 2021. Mutable and Immutable Types in JavaScript (With Examples). URL: https://howtocreateapps.com/mutable-and-immutable-types-in-javascript-with-examples/. Accessed: 23 February 2021.

Ryan Kh. 2017. Fostering Better Communication Between Back-End and Front-End Developers. URL: https://dzone.com/articles/fostering-better-communication-between-back-end-am. Accessed: 8 January 2021.

Shen, S. 2018. Toasts or snack bars? — designing organic notifications. URL: https://uxdesign.cc/toasts-or-snack-bars-design-organic-system-notifications-1236f2883023. Accessed: 20 December 2020.

Shien, T. 2020. How to setup fastlane for build & release React-Native apps (iOS & Android) Part 1. URL: https://medium.com/codespace69/how-to-setup-fastlane-for-build-release-react-native-apps-ios-android-part-1-fd9af5ce6693. Accessed: 16 February 2021.

Stone, S. 2018. Code Refactoring Best Practices: When (and When Not) to Do It. URL: https://www.altexsoft.com/blog/engineering/code-refactoring-best-practices-when-and-when-not-to-do-it/. Accessed: 5 December 2020.

Talwaria, A. 2020. Array's Avengers: forEach(), filter(), map() and reduce(). URL: https://dev.to/aasthatalwaria/array-s-avengers-foreach-filter-map-and-reduce-10nd. Accessed: 2 February 2021.

Tamas, C. 2020. How to manage staging and production environments in React Native. URL: https://dev.to/calintamas/how-to-manage-staging-and-production-environments-in-a-react-native-app-4naa. Accessed: 16 February 2021.

Tan, L. 2018. Continuous Integration and Continuous Delivery with Jenkins and Fastlane. URL: https://www.appcoda.com/ci-cd-jenkins-fastlane/. Accessed: 16 February 2021.

Tarver, E. 2020. Brand Identity. URL: https://www.investopedia.com/terms/b/brand-identity.asp. Accessed: 23 February 2021.

USGS 2021. Web Map Service (WMS), URL: https://lpdaac.usgs.gov/tools/web-map-service-wms/#:~:text=A%20Web%20Map%20Service%20(WMS,transfer%20protocol%20(HTTPS)%20requests.&text=These%20layers%20can%20be%20requested,available%20on%20a%20WMS%20server. Accessed: 26 January 2021.

Vičkus, G. 2019. React17, or how to get rid of “componentWillReceiveProps”?. URL: https://itnext.io/react17-or-how-to-get-rid-of-componentwillreceiveprops-c91f9a6f6f03. Accessed: 8 December 2020.

Wong, Y. 2017. Building React Native app for multiple environments (updated for RN 0.61.4). URL: https://medium.com/@ywongcode/building-multiple-versions-of-a-react-native-app-4361252ddde5. Accessed: 16 February 2021.