Haaga-Helia
University of Applied Sciences

Building a universal application with React and React Native

Ha Van

Bachelor's Thesis
Degree Programme in Business Information Technology
2020

Haaga-Helia
University of Applied Sciences

Abstract

18 May 2020

Author(s)

Ha Van

Degree programme

Business Information Technology

Report/thesis title	**Number of pages and appendix pages**
Building a universal application with React and React Native	**36 + 1**

The thesis is an instruction on how to build a universal application with React and React Native. A universal app is an application which is able to run on both native platform and web. Instead of maintaining the same application with separate projects for React and React Native respectively, the thesis provides a method to combine these popular frameworks in one single project. By this way, developers can reduce significantly effort and time in app development.

The thesis is going to help readers understand how React and React Native operates on their host platforms. Then, it guides readers to set up a universal project with a combination of React and React Native. The thesis is not only a guidance, but it also explains why the setup is feasible. In addition to this method in building a cross-platform app, the thesis also mentions another option. In the end, readers can know the pros and cons of options in building a universal application to pick a better solution in their own project.

Keywords

ReactJS, React Native, Expo, React Native for Web, Webpack, Babel, Metro Bundler

Table of contents

1 Introduction

Nowadays, more and more enterprises pick hybrid applications over native applications due to their development speed and cost reduction. Facebook caught up with this trend and decided to invent React Native to serve mobile development. The idea of React Native comes from the process while Facebook developers were building React JS – a library serves web development. However, building and maintaining both mobile and web applications at the same time lead to task duplication. The same app feature will be created twice for separate platforms – web and mobile. For example, most of the companies' solutions start with web apps. When they attract more users, they want to bring it to mobile because phones are becoming smarter and more handier. It is what social media apps are doing right now. Acknowledge the trends as well as the efforts spared for maintaining two separate projects that most of its' clients are encountering, Reactron Technologies Oy has taken a proactive approach to the cross-platform solution. Since React and React Native are proving their own position on the web and mobile platform respectively, Reactron decided to figure out the combination of React and React Native in one project.

1.1 Company introduction and its solution

Reactron is a digital consulting company that is located in Espoo. They offer technology services and solutions for their customers. The company follows the B2B business model with the following key competencies: Mobile Development, Web Development, Backend Development and Infrastructure.

React JS and React Native are ultimate solutions for web and native mobile respectively in most of front-end projects in Reactron. The reason for it is the popularity of these frameworks in their own industry. According to Stack Overflow statistics in 2019, React JS is one of the most popular web frameworks (Stack Overflow survey, 2019). React JS took the second position, just after jQuery, in the reputation in the web development. Mehul Rajput while discussing the top mobile app development frameworks in 2019-2020 asserted the top position of React Native in mobile cross-platform solutions (Rajput, 2018).

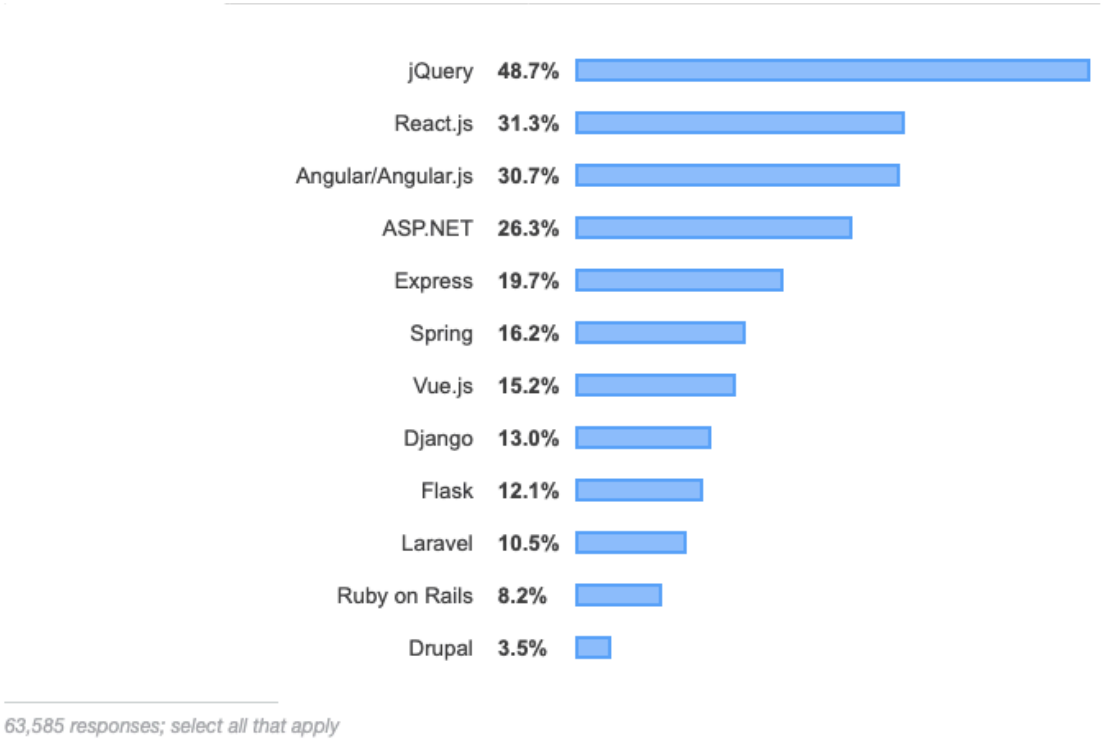


Figure 1. Most popular web frameworks in 2019 (Stack Overflow survey, 2019)

While working with these frameworks separately, Reactron recognised the task duplication and problems in managing multiple projects. In the end, it came to a solution for a better project management. The solution is, in fact, a project template, customized later based on each of the company's customers' problems. The template includes necessary files and folders to configure a universal application. A universal app is an application that can run properly on web platform and native mobile platform. With the same code base, the project can now run on multiple platforms. It is implementing successfully with a client of Reactron. So far, the application has been running smoothly on three platforms: Android, iOS and web. The thesis is also going to present all the crucial steps to build a cross-platform application.

The introduction of this cross-platform solution is going to make a significant contribution to cost and time reduction while having to build separate projects to reach the goal of obtaining universal applications. The outstanding part of this solution is that it handles the native part of the app platform without depending on any services from any third-parties. As a result, this thesis can potentially become a useful reference for other enterprises to consider their technologies for their future projects.

1.2 Objectives and goals

This thesis is conducted with the purpose of indicating constructive steps to configure a universal application with the help of a library called react-native-web. It is a product maintained by Facebook. This thesis is going to explain the infrastructure behind this library, express ideas over the pros and cons of other solutions to build a universal app and present necessary configure steps to run this library.

The thesis is supposed to supply the grip of the operational mechanism of native frameworks when it comes to separate platforms. Then, readers can understand why the following native path is a prolonged strategy. Even though hybrid apps sound charming and fast because they make use of a single code base to run on different environments, modifications can make the code more complicated and error-prone. The comparison with other solutions in this thesis will help the reader make their own business decision on picking the right technology for their applications.

1.3 Out of scope

The thesis only supports theory and figures to substantiate the instruction of building a universal application with React, React Native, and the result to confirm the feasibility of this solution. No business matters related to Reactron's clients will be mentioned in this thesis.

The thesis is going to mainly focus on infrastructure to build a universal app with React and React Native. It does not totally cover how to use React and React Native. Other solutions to universal apps are not taken into a deep dive. However, they are going to be introduced and presented with some advantages and disadvantages. Furthermore, Reactron's solution is currently implemented on only utility applications; other app categories such as games and social media have not been tested yet. That is why the thesis will give some suggestions and predictions about those issues.

1.4 Research questions:

- Which solutions available for universal applications?
- What are the pros and cons of each solution?
- In which scenarios should each solution be implemented?

2 Theoretical framework

The primary language of React and React Native is JavaScript. To run the app natively, Facebook uses different bundlers to run JavaScript on different platforms. Bundling is the process of combining multiple modules into a bundle file, making the code ready for production. In this case, React exerts Webpack and babel to compile the latest version of ECMAScript to plain JS so that different browsers can understand. For mobile, it is Metro Bundler that helps bundle JavaScript so that mobile platforms can understand.

Before jumping to the presentation of React Native for web, this section is going to unravel the infrastructure of React and React Native: How the application can run natively in different platforms with one codebase? Then, it explains why Facebook's frameworks are becoming more and more popular. Additionally, some other solutions to build the universal application are also mentioned with many advantages and disadvantages to prepare for further consideration to pick technology for business applications.

2.1 Web development with React JS

2.1.1 Infrastructure

React JS is a JavaScript library that is used to build user interfaces. Basically, a website is composed of DOM (Document Object Model) – an abstraction of structured HTML code. Elements of HTML code become nodes in DOM. The DOM provides API such as 'getElementById', 'removeChild', 'getClassById', etc to traverse and modify the nodes. The DOM is represented as a tree structure which makes the changes and updates to DOM easier. However, after each modification, the updated element and its children have to be repainted to update the UI. The more UI components, the more expensive the DOM updates become. That is why it is a real performance pain of dynamic web apps (SPA – Single Page Application) (Krajka, 2015)

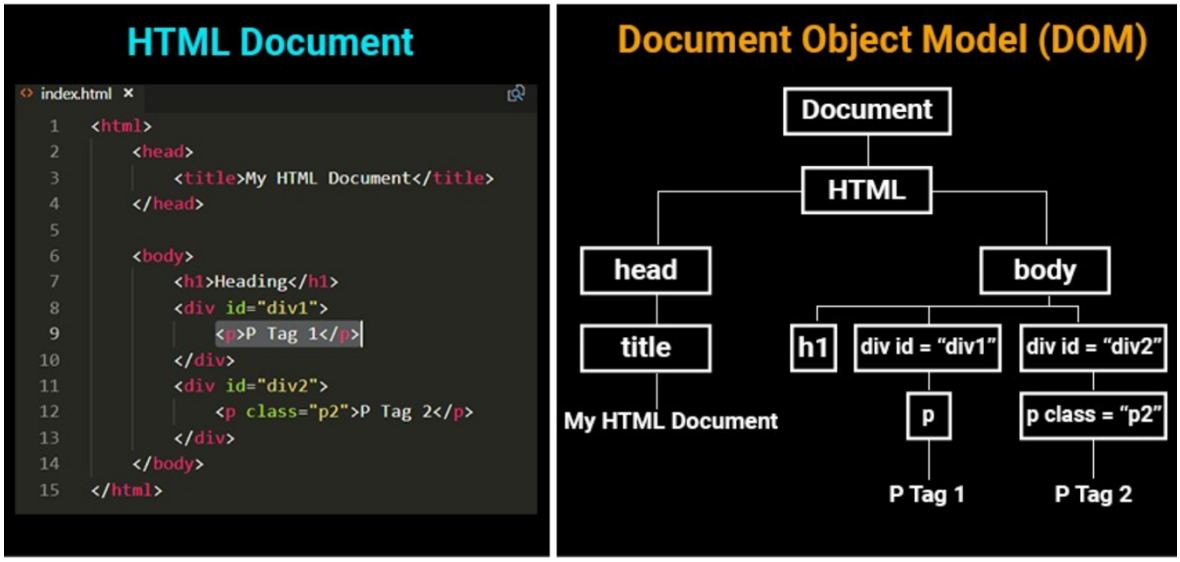


Figure 2. An example of DOM (Sakpal 2020)

React comes with a helping hand when it implements Virtual DOM (or can be called React DOM) to handle this development pain. Virtual DOM is a copy of HTML DOM. It is light-weight and detached from browser-specific implementation. When a new element is created, a virtual DOM is created. If any elements get updated, a new virtual DOM is also created. This virtual DOM then compares with the former virtual DOM to compute diff and apply these changes to the real DOM. In this way, only the updated nodes get re-rendered, not the whole tree. (Krajka, 2015)

From Facebook React documentation, the only way to update the React DOM is to create a React element and pass it to the following function:

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

Figure 3. An example of updating Virtual DOM (React Documentation, 2020)

Facebook also defines react element as “React elements are immutable. Once you create an element, you cannot change its children or attributes. An element is like a single frame in a movie: it represents the UI at a certain point in time.” React elements live in virtual DOM as basic nodes. Their immunity makes them fast to compare and update in virtual DOM. The immunity also sounds useless to dynamic web apps that are supposed to be stateful. Nevertheless, JSX- XML/HTML like syntax extension to JavaScript- comes along with a hand of compiling HTML tags to React element. It is the syntax that Facebook recommends using to describe what the UI should look like. (React Documentation, 2020)

2.1.2 Compiling and Bundling

In ReactJS, JSX is intended to be used by Babel – a transpiler to transform JSX into a standard JavaScript object. Basically, JavaScript has no definition of class, Map, Set, etc. which is created by ECMAScript. This standard JavaScript objects will be later parsed by JavaScript engine – an interpreter of the web browser. Along with Babel, Webpack is also used as a module bundler to ensure that the app's JavaScript is compatible with different browsers. It explains the cross-browser feasibility of React JS. (Babel Documentation, 2020 & Webpack Documentation, 2020)

As a matter of fact, all React components are written in JSX. A React component contains app states. Whenever the component's states change, the component is re-rendered. Then, it is converted into the React element. Now the React element can be added to virtual DOM. When the diff computation is completed, the virtual DOM is passed to the HTML DOM. (React Documentation, 2020)

2.2 Mobile development with React Native

2.2.1 Categories of mobile application

Most mobile applications nowadays fall into one of the two following categories: hybrid or native.

Native apps are exclusively built for one specific operating system, such as iOS, Android or Windows. In the manner that they are called native, these apps are built for a particular platform and cannot be used on another operating system. There are development tools and languages to support separate platforms. For instance, Xcode and Swift/ Objective-C for iOS, Android Studio and Java or Kotlin/ Java. These apps can be download from the App Store (iOS) and Google Play Stores (Android). The native app ensures high performance and excellent use experience because it uses native-device UI with full access to all device controls like contacts, camera, location services, gestures, animation etc. Native apps can run offline without getting impacted by server connection because their content and images are stored on the device. Nonetheless, due to the single platform characteristic, native apps are expensive to develop and maintain. Native languages like Swift or Java are hard to master; expenditure for hiring native developers usually cost a fortune.

Hybrid apps are a mixture of web apps and native apps. They are like a website packaged in a native application using an exclusive platform. The apps are shown in its embedded

browser, i.e. Riverview in iOS and WebView in Android. The popular platforms for building hybrid apps are Ionic, Cordova and PhoneGap. They all provide plugins to access device utilities like Bluetooth, Camera, Location, etc. Developers can build cross-platform web, mobile, desktop applications without caring about the native platforms but still ensure native feelings. A few years back, many companies preferred this solution due to its cost and time reduction. However, hybrid apps depend heavily on plugins, so access to device features can be limited. Not all built-in device features are available, or they can be out of date. The core of hybrid apps is a website, not native; it is impossible for these apps to work offline. It is the reason why hybrid apps are best suited for being content-focused. The hybrid app is also slower than native apps because each element has to download. The philosophy of “write once, run everywhere” can cause platform inconsistencies. Some features or designs do not always support both platforms, and sometimes it requires developers to write modifications on their own. (Ashley MacQuarrie, 2018)

2.2.2 Infrastructure

React Native is a JavaScript framework for writing native rendering applications for mobile platforms. It currently supports iOS and Android. React Native is based on React’s concept, but the difference is host platform API. Instead of rendering to the browser’s DOM, React Native invokes Objective-C APIs to render iOS components or Java APIs to render Android components. These platform-specific APIs play a key role in native-feeling user experience. Mobile APIs include data storage, location services, camera, Bluetooth, etc. It sets React Native app away any definition of mobile app’s categories, neither native apps or hybrid apps.

Just as in React, React Native uses JSX – a combination of markup and Javascript to control view in a single file. In React, this markup is then translated into the browser’s DOM. However, there are no HTML elements in the mobile environment. Instead, the markup is translated to platform-native elements. JSX, in this context, is like a “bridge” to connect to the specific platform and render its invoked components (Eisenman 2016, 27). The following table is an instance of some base components in different platforms:

Table 1. Base components in different platforms (React Native documentation, 2020)

Web	React Native	iOS	Android
div	View	UIView	View
span	Text	UILabel	TextView
ul, li	FlatList, SectionList	UICollectionView	ListView
image	Image	UIImage	ImageView

input	TextInput	UITextField	EditText

One thing should take into consideration while coding with React Native, styling in CSS is impossible in React Native. Instead, the StyleSheet object is provided and applied to the component's style attribute. The property and value using in StyleSheet object are quite similar to CSS, but the naming must be camel case. Nevertheless, not all CSS syntaxes are available in StyleSheet. The way to animate elements in React Native is also distinct from regular CSS. React Native supplies an Animation component to handle animation and gestures.

The following image is going to demonstrate the difference in rendering JSX in separate platform between React and React Native:

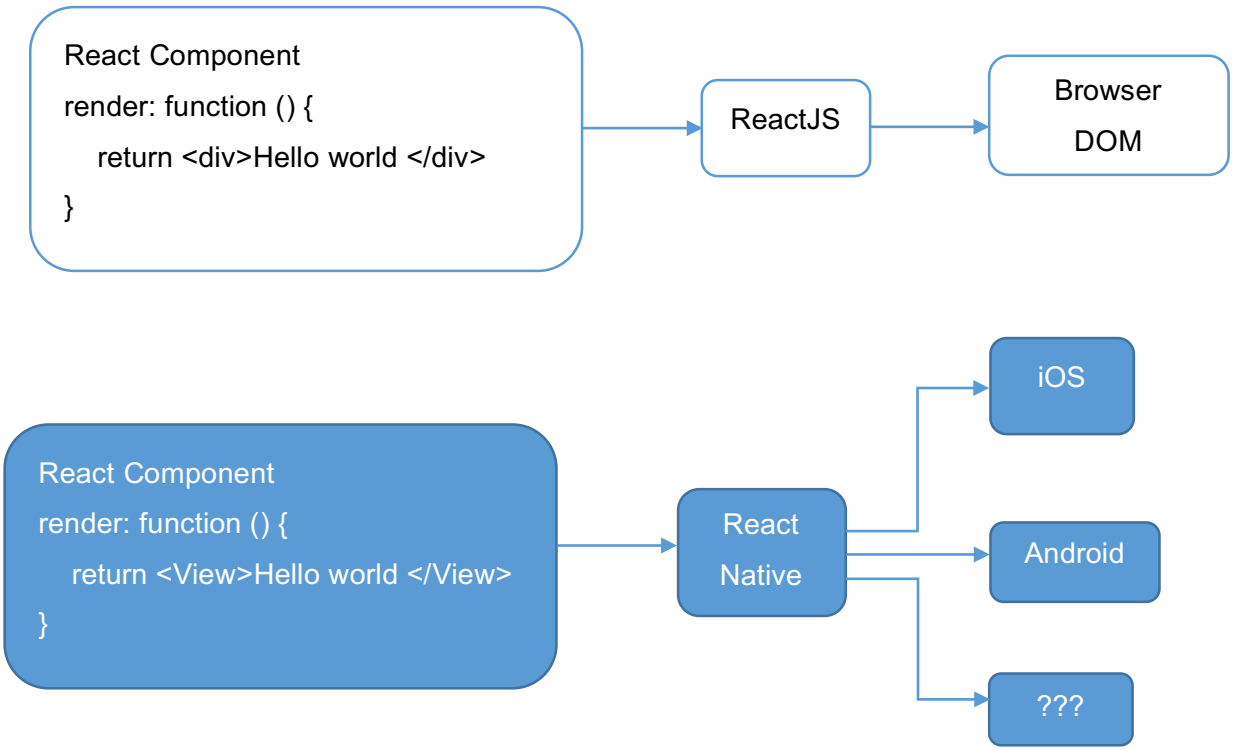


Figure 4. Demonstration of JSX rendering (Eisenman 2016, 27)

Thanks to the abstraction layer provided by the concept of Virtual DOM, React Native is able to target other platforms as long as someone needs to write the "bridge". That is why React Native can run on the web too, but not the other way around. The empirical part is going to deep dive into the "bridge" for the web.

2.2.3 Compiling and Bundling

How can the mobile operating system understand JavaScript? From Facebook's documentation, React Native runs its JavaScript through the device's JavaScript Engine called JavaScript Core. However, if debug mode (Remote JS Debugging) in a mobile simulator is activated, JavaScript will run through the browser's JavaScript Engine and using the browser's debugging tools. Not only can developers use browser's console but also network requests. Now developers can enjoy debugging as they do with the web. (React Native documentation, 2020).

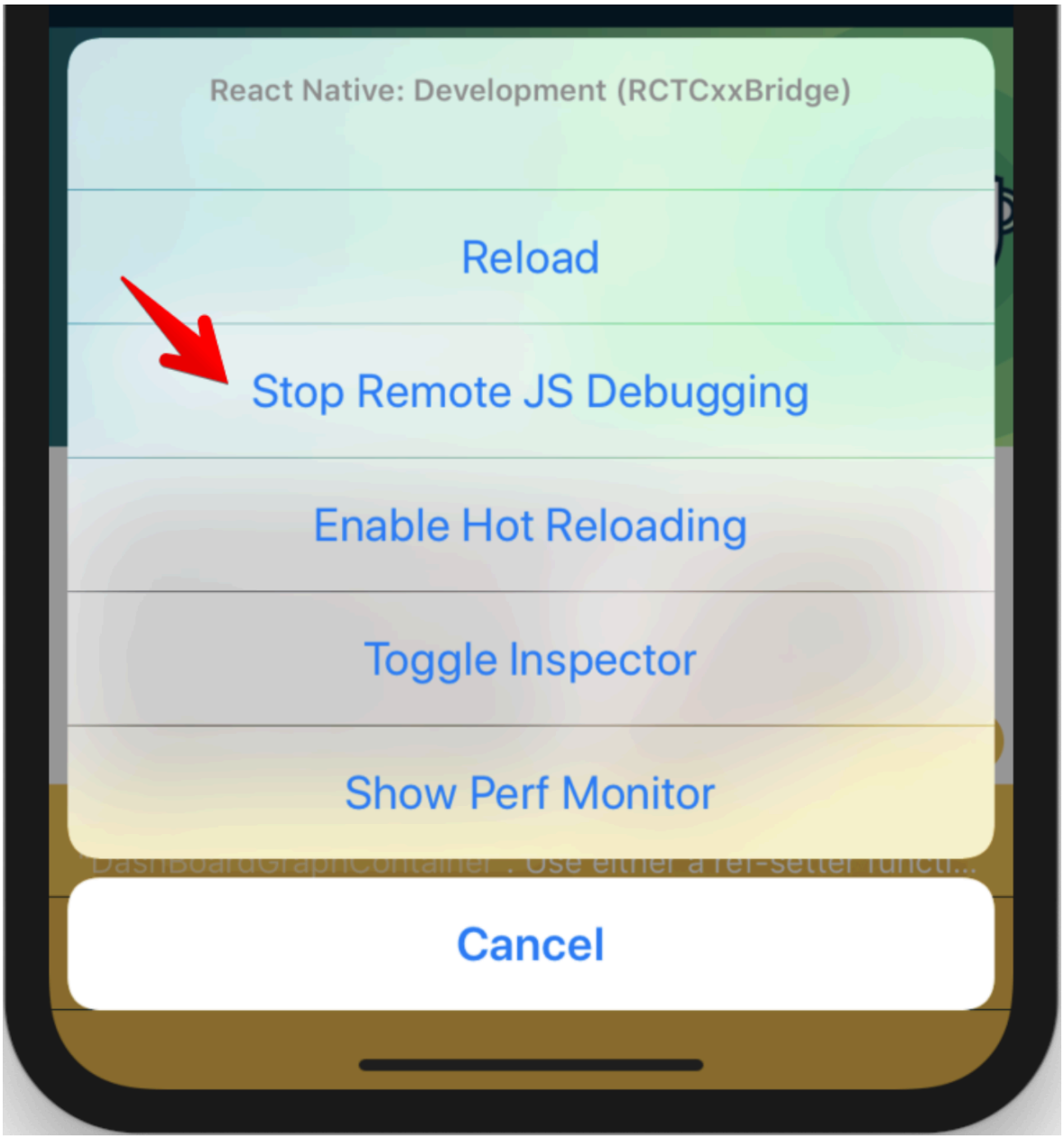


Figure 5. React Native DevTool selections

One novel feature that React Native supplies is its live reload feature. It allows to quickly recompile and load bundled JavaScript files after making changes in the code. Basically, in iOS development, the app must be killed in order for the new changes.

React Native runs on two different threads: Main thread and JavaScript thread. The main thread is in charge of rendering UI and catching user gestures, whereas the JavaScript thread is responsible for handling the app logic and defining the UI structure.

React Native also uses Babel and Polyfill for syntax transformation to make sure that the JavaScript code may not be supported natively by JavaScriptCore. Contrary to ReactJS, React Native uses Metro to bundle JavaScript. Its main task is to combine all JavaScript code into a file and translate the code so that the device can understand. Metro also converts assets into objects that can be displayed by React Native's <Image> component.

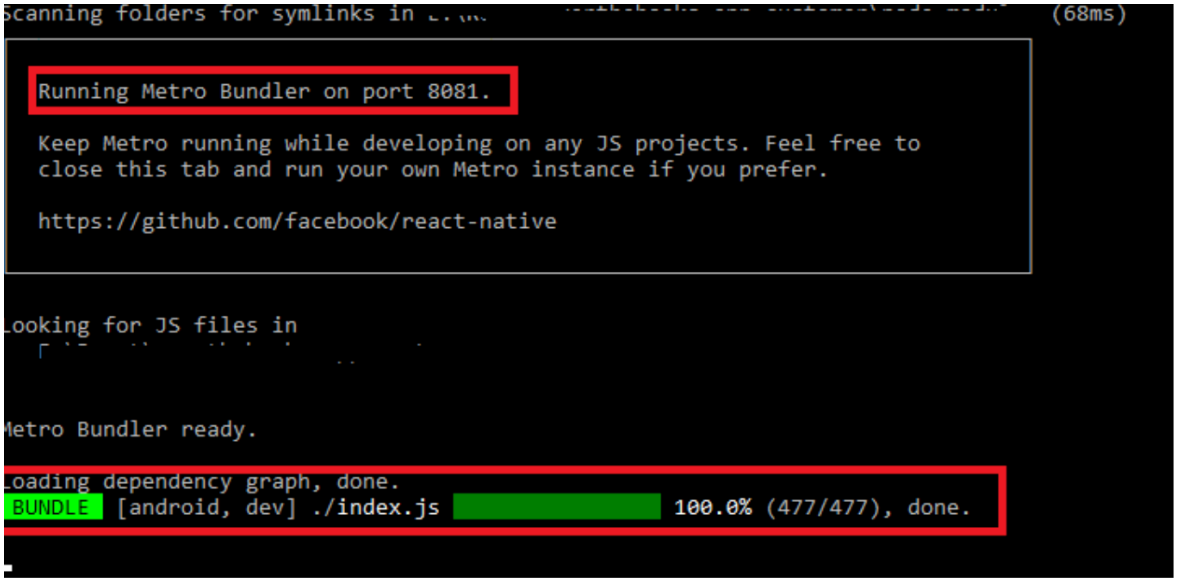


Figure 6. An example of Metro Bundler

From a blog on Medium written by Rishabh Sharma, Metro bundling goes through 3 primary processes (Sharma, 2018):

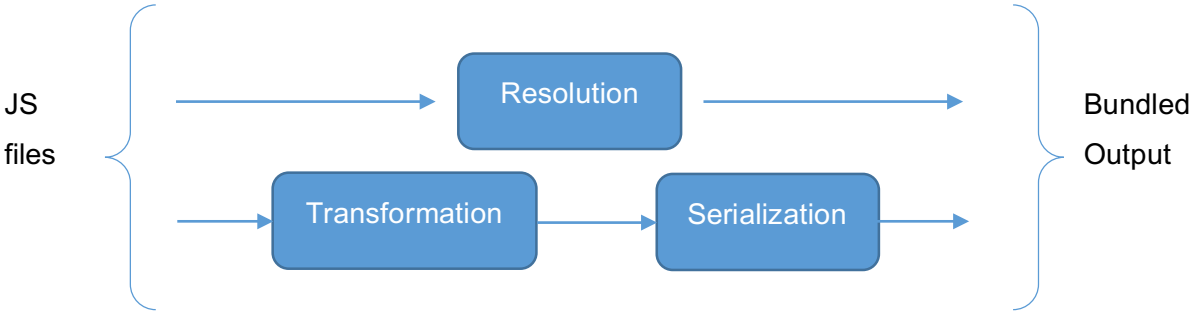


Figure 7. Infrastructure of Metro Bundler (Sharma, 2018)

- **Resolution:** is the process of building a directed graph of all modules required from the entry point. Metro utilises a resolver to detect which files are required from a file. This process happens in parallel with Transformation and Serialization.
- **Transformation:** is responsible for translating a module to a format that React Native can understand.
- **Serialization**: the process happens after the transformation completed. The formatted modules are now serialized and combined to one or a multitude of bundles. The bundled output is then passed to the target platform's JavaScript Engine to run the Universal app application with React Native.

2.3 Expo

2.3.1 Infrastructure

Expo is a toolchain built around React Native to help developers start their projects quickly. It roughly corresponds to the release of React Native, so when React Native released react-native-web to react native apps on the web platform, Expo also updates its SDK (software development kit). Expo provides a set of tools and services to develop, build, deploy, test or even run simulators on the specific platform from the same codebase. Specifically, it offers a collection of ready solutions such as device accelerometer, camera, notifications, geolocation, etc. One convenient point using Expo is that developers do not need to download XCode or Android Studio to run the app. They can run the app directly with Expo CLI. To install Expo and run a project with Expo, follow the below commands: (Expo Documentation, 2020)

```
# Install the command line tools
$ npm install --global expo-cli

# Create a new project
$ expo init my-project
```

Figure 8: Commands to setup project with Expo

Expo provides a multitude of commands for running and build apps on both mobile and web platform:

```
$ expo init # Create a new project
$ expo start # Star the Javascript bundler
$ expo ios # Run on iOS
$ expo android # Run on Android
$ expo web # Run on Web

$ expo build:ios # Build iOS app
$ expo build:android # Build Android app
$ expo build:web # Build an optimized-version of a React app
```

Figure 9: Expo available commands

After the installation, in app.js, it is recognizable that the web platform has been added under platform array.

```
{
  "expo": {
    "name": "awesome-expo-web-example",
    "slug": "awesome-expo-web-example",
    "privacy": "public",
    "sdkVersion": "33.0.0",
    "platforms": [
      "ios",
      "android",
      "web"
    ],
    ...
  }
}
```

Figure 10: An example of app.json

Developers are able to separate codebase for a specific platform as long as the file under the following format: 'filename.platform.js'. For example, file 'example.ios.js' will only run on iOS, 'example.android.js' will only run for Android, and 'example.web.js' is only for the web. The codebase is the same as React Native.

2.3.2 Compiling and Bundling

When running the app with Expo CLI, it develops and manages two server processes in the background: Expo Development Server and React Native Package Server. (Expo Documentation, 2020)

The Expo Development Server is the endpoint added to the Expo app. Basically, when the app is firstly opened on Expo, an input pop up asking for the URL, which can be found in

Expo Manifest. It is a communication layer between Expo CLI and the app running on the real device or simulator. The following is an example of Expo Manifest

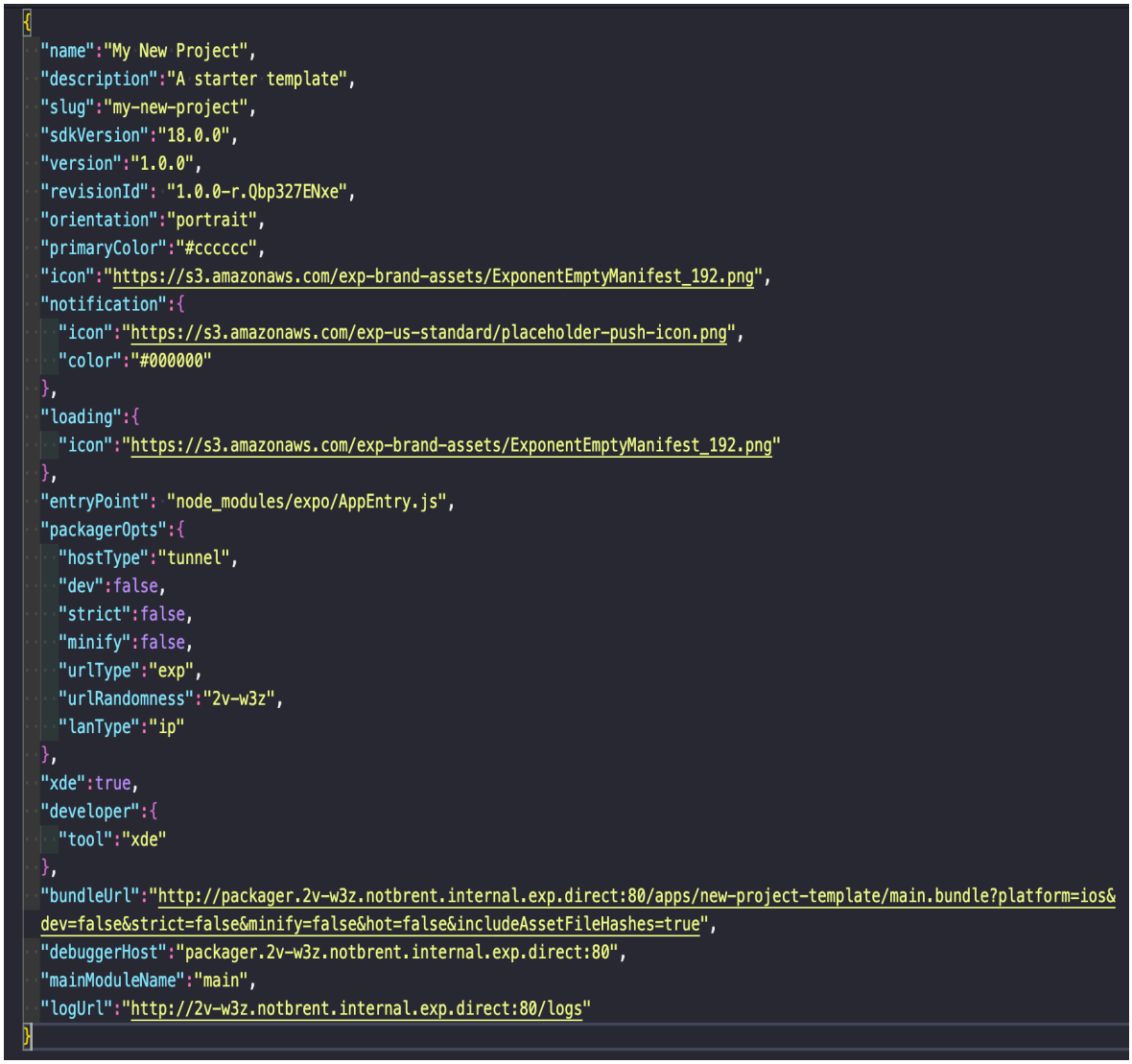


Figure 11: An example of manifest.json (Expo Documentation, 2020)

Every field in this file is the configuration for Expo to run the app. It is noticeable that few fields in this file are identical to the ones in file ‘app.json’. It is because they are generated from that file, which demonstrates how the Expo app access the configuration. At first, the app goes through a manifest file, then fetch the app’s JavaScript, which is given from the field ‘bundleUrl’. This endpoint points to the React Native Package Server. (Expo Documentation, 2020)

The function of the React Native Package Server is to compile the app’s JavaScript into a single file and translate any code that is not compatible with the platform’s JavaScript engine, JavaScriptCore. Additionally, it also serves assets to correct assets directory for screen DPI, assuming that assets exist. (Expo Documentation, 2020)

2.3.3 Deploying

The ‘build’ command from Expo CLI kick off the Expo build service. When publishing an Expo app, the app’s JavaScript code is bundled with the production flag enabled. After compilation, Expo uploads that bundle file to CloudFront. Then the ‘manifest.json’ file is uploaded to the Expo server with the revisionId key to tell the release of the app. An URL is generated when the publishing is complete to share the app with anyone as long as they are Expo clients. The next version of the app will be auto-updated the next time users open and refresh the app, provided that they have the version of Expo client that supports the SDK version, which is identified in the ‘app.json’ file. (Expo Documentation, 2020)

2.3.4 Drawbacks

Despite the convenience of Expo SDK, developers are limiting themselves to the packages Expo is offering. Not all iOS and Android APIs are available yet. Expo currently does not support integration with any other SDK, which is necessary for app customization. In case developers face Expo’s limitations, the only solution is to detach the app from Expo. It is a pain because it entails a number of errors and costs a lot of time and money. (Expo Documentation 2020)

Although Expo is able to support three different platforms, it is challenging to serve all of them equally. The web platform is the most novice out of three, so it is unstable to build web with Expo. Besides, using a simulator to communicate with the native app is still a slow process. (Expo Documentation 2020)

Even though Expo offers excellent service for building and publishing the app. Free builds can sometimes be queued. Expo servers are usually unavailable due to overload. Hence, sometimes notifications do not work because of the problem with servers. (Expo Documentation 2020)

3 Empirical part

From the previous section, the answer to how React and React Native run in web and native platforms has been discovered. React Native is a pure UI language. All React Native's components are designated as the platform's UI primitives. For instance, prevalent components of React Native such as <View>, <Text>, <Image> are primary elements that make sense to any visual interface no matter where the app is running. Meantime, React's essential elements are just DOM nodes, which define solely for the web. It is why React components do not make sense out of browsers. In order to build a universal app, React Native has a superior position to React, which means it should start from React Native, not the other way around. It is possible to translate React Native primitives to the DOM node as long as a "bridge" – APIs - is erected.

Understanding thoroughly how React and React Native works, Nicolas Gallagher has created a react-native-web library to make it possible for React Native apps to run on the web. It has been on the market for almost five years and continues to spread among the developers. In this section, react-native-web will be introduced, and the necessary steps to build a universal app will be presented.

3.1 React Native Web

React-native-web is an open-source project created by Nicolas Gallagher in 2015. It allows using React Native API components on the web. React Native has proved itself in the mobile platform, now the web is gradually entering this picture.

This library has been implemented on giant tech companies' projects such as Twitter, Major League Soccer, Flipkart, Uber, The Times, DataCamp. The browser support for this open-source is Chrome, Firefox, Edge, Safari 7+, IE 10+. Deprecated components and APIs in React Native are also not supported by React Native for Web. It is possible to integrate React Native for Web with other prevailing web tools such as Gatsby, Next.js, Phenomic, Docz, Razzle, Storybook and Styleguidist. (Nicolas Gallagher, 2020)

Even though React Native for Web allows to use React Native APIs to run on the web, not all React Native components are currently supported. The following table is going to enlist the status of React Native components presently in react-native-web.

Table 2. Status of React Native components in React Native for Web (Nicolas Gallagher, 2020)

Name	Status	Note
ActivityIndicator	✓	
Button	✓	
CheckBox	✓	
FlatList	✓	
Image	✓	Missing multiple sources and HTTP headers.
ImageBackground	✓	
KeyboardAvoidingView	(✓)	Mock. No equivalent web APIs.
Modal	✘	Not started
Picker	✓	
RefreshControl	✘	Not started
SafeAreaView	✓	
ScrollView	✓	Missing momentum scroll events.
SectionList	✓	
StatusBar	✓	Mock. No equivalent web APIs.
Switch	✓	
Text	✓	Missing onLongPress support
TextInput	✓	Missing rich text features and auto-expanding behaviour
Touchable	✓	Includes additional support for mouse and keyboard interactions.
TouchableHighlight	✓	
TouchableNativeFeedback	✘	Not started
TouchableOpacity	✓	
TouchableNativeFeedback	✓	
View	✓	
VirtualizedList	✓	
YellowBox	(✓)	Mock. No Yellowbox functionality

The following table is going describe the status of React Native modules in React Native for Web:

Table 3. Status of React Native modules in React Native for Web (Nicolas Gallagher, 2020)

Name	**Status**	**Note**
AccessibilityInfo	(✓)	Mock. No equivalent web APIs.
Alert	✘	Not started.
Animated	✓	Missing `useNativeDriver` support.
AppRegistry	✓	Includes additional support for server rendering with getApplication.
AppState	✓	
BackHandler	(✓)	Mock. No equivalent web APIs.
DeviceInfo	(✓)	Mock. No equivalent web APIs.
Clipboard	✓	
Dimensions	✓	
Easing	✓	
Geolocation	✓	
I18nManager	✓	Includes additional support for runtime switch to RTL.
InteractionManager	(✓)	
Keyboard	(✓)	Mock.
LayoutAnimation	(✓)	Missing translation to web animations.
Linking	✓	
NativeEventEmitter	✓	
NativeMethodsMixin	✓	
NativeModules	(✓)	Mock. Missing ability to load native modules
PanResponder	✓	
PixelRatio	✓	
Platform	✓	
Settings	✘	No equivalent to web APIs.
Share	✓	Only available over HTTPS.
StyleSheet	✓	
UIManager	✓	
Vibration	✓	

For those components or modules that are not currently supported, it is recommended to create two files for different platforms and use separate native or web modules to handle. As a convention, the file with suffix '.js' is for the native platform, and the file with extension '.web.js' is for the web. For example, the Modal component is now not supported. The following two images are going to demonstrate how to create Modal components for the apps.

```
import Modal from 'react-native-modal';

export default Modal;
```

Figure 12. Modal.js

```
import Modal from 'modal-enhanced-react-native-web';

export default Modal;
```

Figure 13. Modal.web.js

Figure 11 uses a Modal library for React Native, and figure 12 uses a React Modal library. Now, to use the Modal component, it will be imported as follow:

import Modal from '/path/to/file/modal'

Thanks to bundle tools, it is not necessary to add file extension while importing any files. That is why the relative path to the file is enough. In this way, the system knows which modules to take on different platforms.

3.2 Installation

So as to build a universal application, the following packages are needed to install into the project:

- react-native: Firstly, a React Native must be created. NPX is a preferred package to install the React Native app. NPX stands for Node Package Executor that ships with NPM (Node Package Manager) since its version 5.2.0. NPX can initiate a build using a tool name without the prior necessity for local or global installation. With the first line of code, a project is initiated using React Native CLI with a basic configuration for a React Native project. So as to create a React Native project, use this command "npx react-native init ProjectName". (NPX Github 2020)

- react-scripts: The heart of React JS.
- react-native-web: The library assists invoking React Native API on the web.
- React-app-rewired: Override Webpack configuration if needed.

After the project is created, a file called “package.json” contains the app information. It also enlists scripts for running and building the project in different platforms:

```
"scripts": {
  "android": "react-native run-android",
  "ios": "react-native run-ios",
  "web": "SKIP_PREFLIGHT_CHECK=true react-app-rewired start",
  "start": "react-native start",
  "test": "jest",
  "lint": "eslint ."
},
```

Figure 14. Scripts for running the project in package.json

When running the app, the system is going to find index files and start its process. In the universal case, it is better to have two distinct files, one for web and one for mobile as ‘index.js’ and ‘index.web.js’ to differentiate two platforms. React Native for Web offers a method called AppRegistry for registering, running, prerendering and unmounting all apps. App root components are recommended to be registered with ‘AppRegistry.registerComponent’. Apps can be run by calling ‘AppRegistry.runApplication’. The following images are going to display how to write those index files:

```
1  import {AppRegistry} from 'react-native';
2  import App from './src/App';
3  import {name as appName} from './app.json';
4
5  AppRegistry.registerComponent(appName, () => App);
```

Figure 15. index.js

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import App from './App';
4
5  ReactDOM.render(
6    <React.StrictMode>
7      <App />
8    </React.StrictMode>,
9    document.getElementById('root'),
10 );
```

Figure 16. index.web.js

The following picture in an instance of default folder structure of a React Native project.

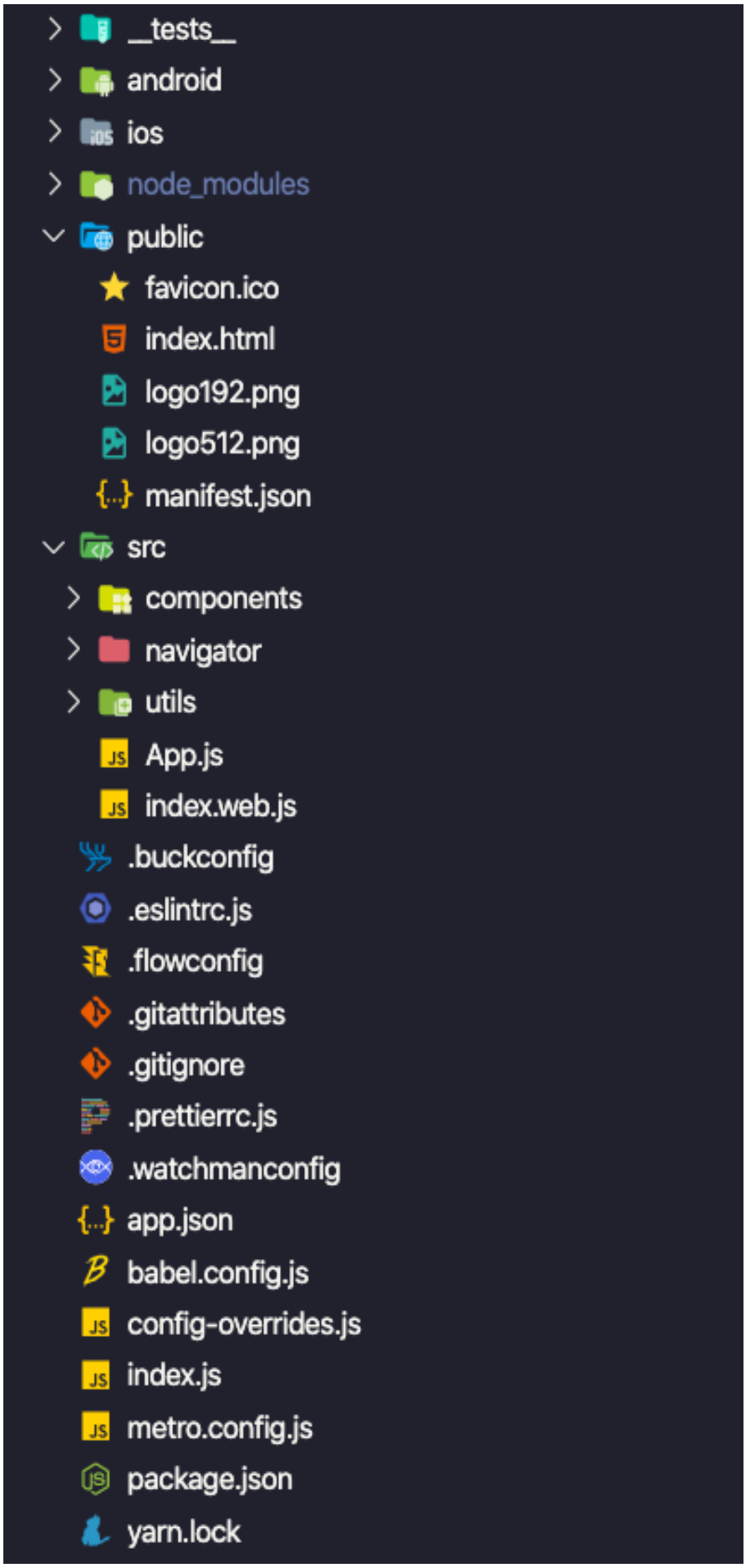


Figure 17. Default folder structure of a React Native project

By default, the web is going to look for a public folder that contains an index HTML file (Figure 17). One thing that should be noticed is that the index.web.js (Figure 16) must be created in the src folder because it is what react-scripts are programmed to seek some files to run on the web.

3.3 Compiling and bundling

One headache problem while creating a universal application is bundler management. From the previous section, it is noted that the standard method of bundling JavaScript in React Native is Metro Bundler, and the standard method for the web is Webpack. All platforms use the same tool for the JavaScript converter - Babel. With React Native for Web, it is proposed to config those two bundlers so that the project can run smoothly on a different platform.

Babel is a JavaScript transcompiler that is responsible for converting JavaScript 2015+ code into a compatible JavaScript version that the older JavaScript engines can understand. Next image is going to demonstrate how to config Babel for the project:

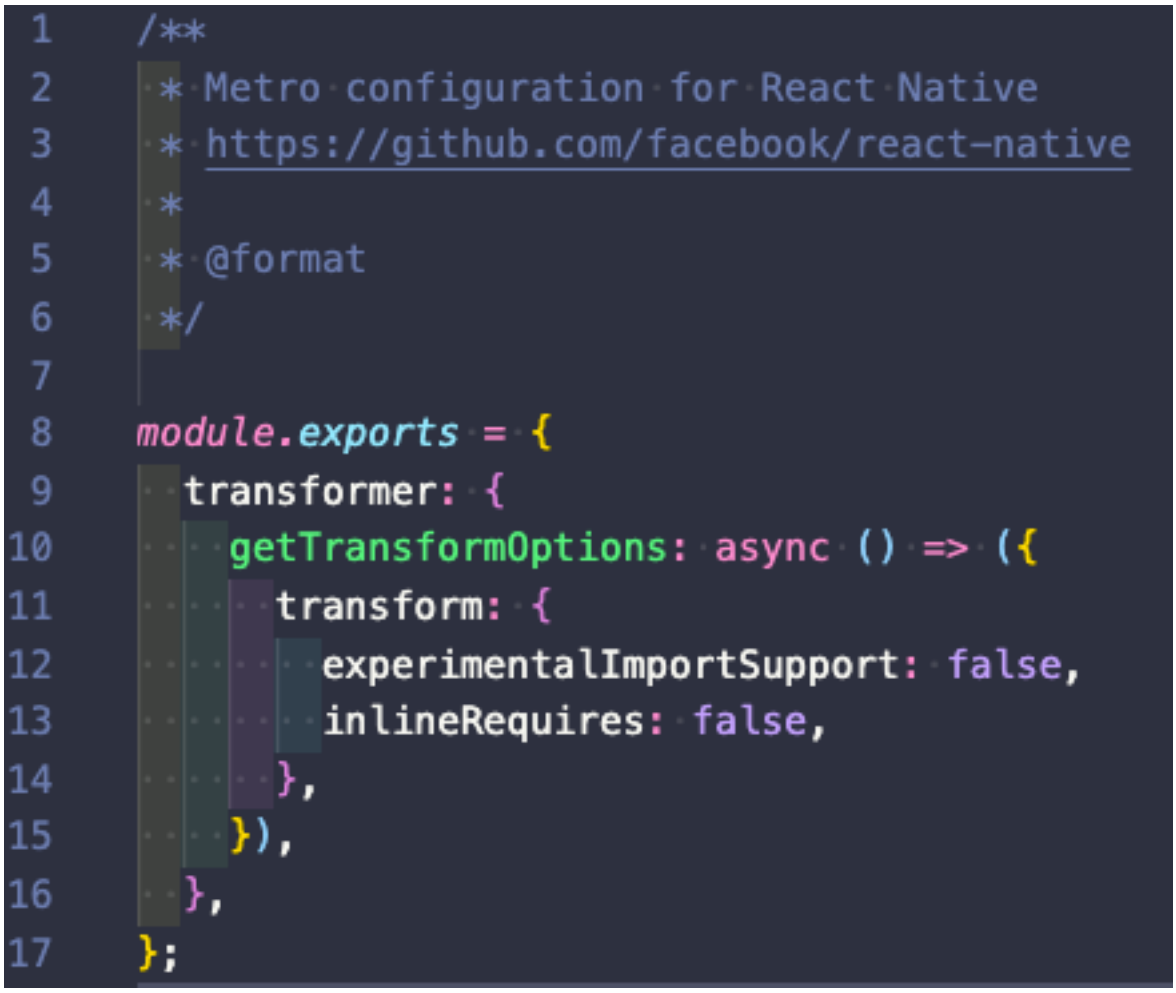
```
module.exports = {
  presets: ['module:metro-react-native-babel-preset',
  'module:react-native-dotenv'],
};
```

Figure 18. babel.config.js

Babel configuration should be created in the file ‘babel.config.js’ in the root project folder. The first module item in the presets array is a popular package used for transpiling JavaScript code in React Native projects. The second package supports environment variables import from ‘.env’ files in React Native.

Metro bundler is a JavaScript module bundler for React Native created by Facebook. It takes all JavaScript files and compiles them into a single file. From the Metro documentation, a Metro config can be created in the following three ways which are ordered by priority: (Metro Bundler Documentation, 2020)

- metro.config.js (or react-native.config.js for newer version of React Native)
- metro.config.json
- The metro field in package.json

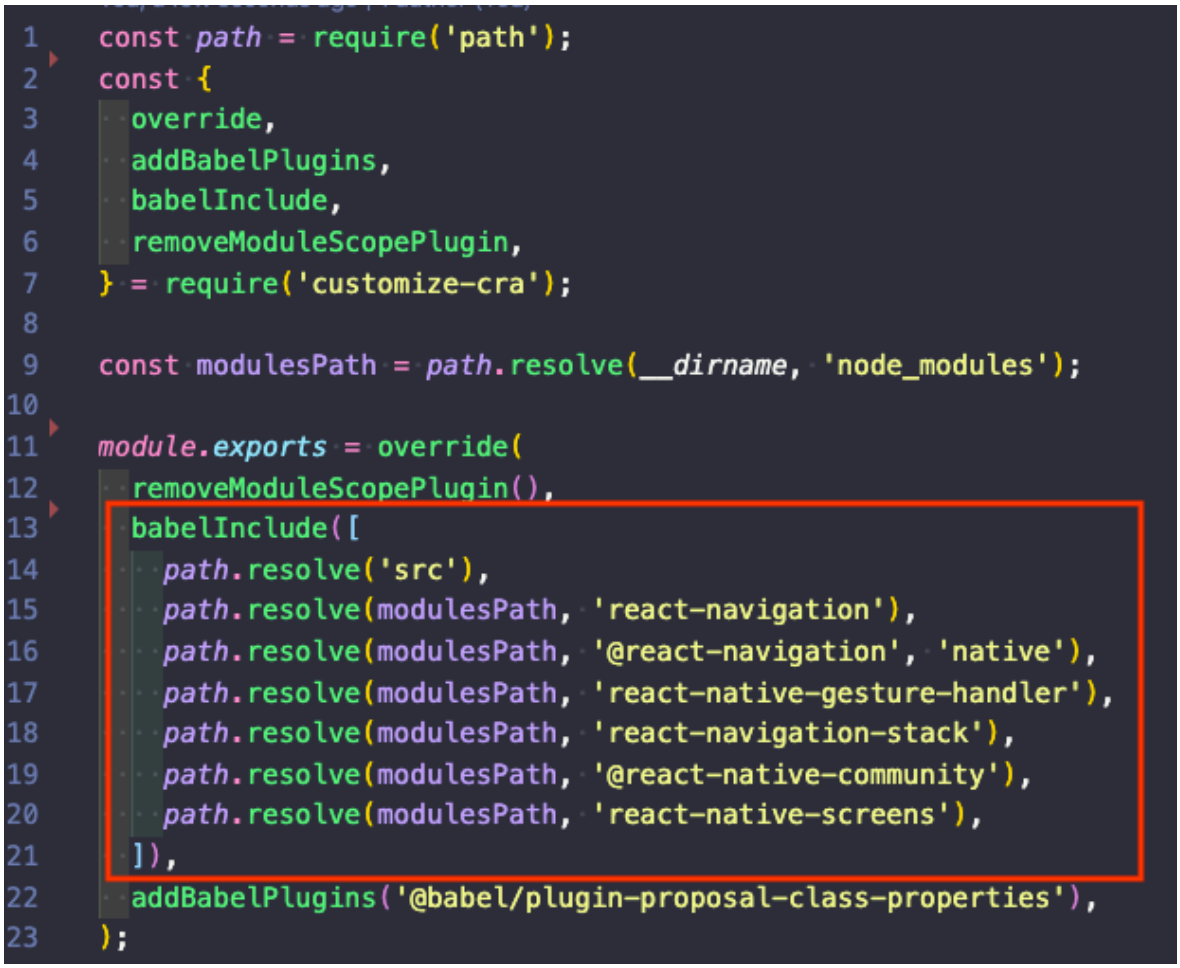


```
/**
 * Metro configuration for React Native
 * https://github.com/facebook/react-native
 *
 * @format
 */

module.exports = {
  transformer: {
    getTransformOptions: async () => ({
      transform: {
        experimentalImportSupport: false,
        inlineRequires: false,
      },
    }),
  },
};
```

Figure 19. metro.config.js

Webpack is a JavaScript module bundler for the web. Typically, Create React App – a global command-line utility – is used to create new projects with the latest version of ‘react-scripts’ – a development dependency. In fact, default Webpack config in Create React App is unable to bundle React Native modules and invoke the web API. In a universal project, Webpack needs to override so that the web can understand React Native dependencies. In fact, in the web platform, Webpack takes responsibility for transpiling the files inside node_modules folders. In order to do that, ‘react-app-rewired’ and ‘customize-cra’ are needed to install. ‘customize-cra’ takes advantage of ‘react-app-rewired’ ‘s config-override.js file to manipulate the configuration of Create React App. To summary, overriding default Webpack and Babel in the web is possible to implement in the file config-override.js.



```
const path = require('path');
const {
  override,
  addBabelPlugins,
  babelInclude,
  removeModuleScopePlugin,
} = require('customize-cra');

const modulesPath = path.resolve(__dirname, 'node_modules');

module.exports = override(
  removeModuleScopePlugin(),
  babelInclude([
    path.resolve('src'),
    path.resolve(modulesPath, 'react-navigation'),
    path.resolve(modulesPath, '@react-navigation', 'native'),
    path.resolve(modulesPath, 'react-native-gesture-handler'),
    path.resolve(modulesPath, 'react-navigation-stack'),
    path.resolve(modulesPath, '@react-native-community'),
    path.resolve(modulesPath, 'react-native-screens'),
  ]),
  addBabelPlugins('@babel/plugin-proposal-class-properties'),
);
```

Figure 20. config-override.js

From now on, in order to use React Native dependencies, it is proposed to define them in the babelInclude parameter in the override() function as the red border in the image has demonstrated.

3.4 Navigation

Navigation is one of the most critical features of an application. In React Native world, there are two well-known dependencies for navigation. They are React Navigation and React Native Navigation. Acknowledged the potential of universal application, React Navigation has supplied two dependencies: react-navigation for mobile and @react-navigation/web for the web. That is the reason this dependency is picked for universal projects.

So as to create navigation for different platforms, firstly create an app stack where defines all of the app's screens as the following figure 21:

```
import {createStackNavigator} from 'react-navigation-stack';
import Home from '../components/Home';
import About from '../components/About';

const AppStack = createStackNavigator({
  Home: {
    screen: Home,
    path: '',
    navigationOptions: {
      headerTitle: 'Home',
    },
  },
  About: {
    screen: About,
    path: 'about',
    navigationOptions: {
      headerTitle: 'About',
    },
  },
});
export default AppStack;
```

Figure 21. AppStack.js

Figure 20 demonstrates how to define the app screen. As an example, there are two screens called Home and About. The above example uses react-navigation-stack to define the stack navigator. In the native platform, the app is constructed as a stack with two screens. In the web platform, when users access the web app with the empty requested content, it directs to Home screen, and when users use '/about' as requested content, it directs to the About component.

```
import {createBrowserApp} from '@react-navigation/web';
import {createAppContainer} from 'react-navigation';
import {isWeb} from '../utils/helpers';

export const createApp = isWeb() ? createBrowserApp : createAppContainer;
```

Figure 22. index navigator

Figure 21 create a function called createApp to create the whole app navigation for platform-specific. It means that on the web, the app utilises function createBrowserApp from the library "@react-navigation/web" and on native platform, function createAppContainer from react-navigation will be used.

3.5 Deploying

Without the assistance of Expo, every configuration in the universal app is conducted manually. Before publishing the ready application on App Store (iOS) and Google Play (Android), make sure that the following steps are gone through:

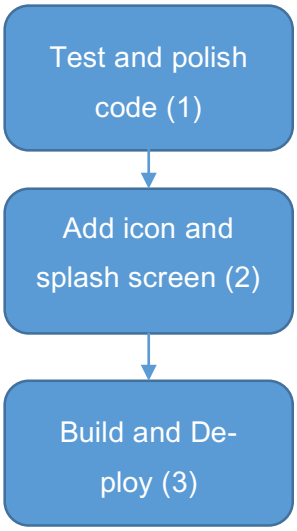


Figure 23. Deployment steps

Step 1: The code must be tested carefully and thoroughly. The app must ensure that it is responsive to a different platform. It is recommended to run the app on three platforms for every built feature to make sure that it runs smoothly.

Step 2: To complete this step, it is necessary to have XCode and Android Studio installed.

- Web: The app icon should be saved as “favicon.ico” in the public folder. Then, it must be linked in the file “index.html” with the following command:

  `<link rel=”icon” href=”%PUBLIC_URL%/favicon.ico”/>`

  The image size should be 64x64.
- XCode: Open the project with the extension “.xcworkspace”. In XCode, select “ProjectName > ProjectName > Images.xcassets ”. Folder “Images.scassets” is the place to contain all the app’s images. Make sure that the developer has all the image sizes that XCode requires. The splash screen is designed in the file “LaunchScreen.xib”. XCode already supports tools for designing with dragging and dropping. (Spencer Carli 2020)

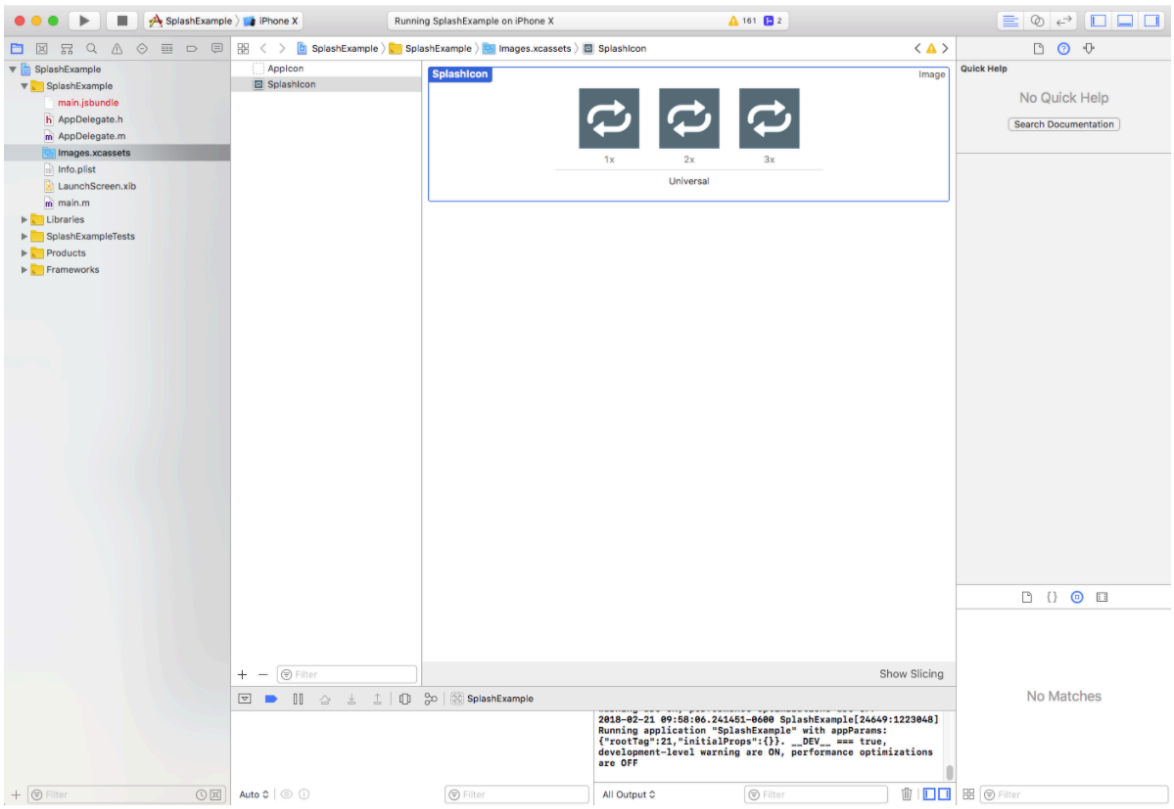


Figure 24. An example of adding icon and splash screen on Xcode (Carli, 2020)

- **Android:** Android Studio does not support dragging and dropping as XCode. It requires some files are created and modified to display SplashScreen. For adding app icon, right-click on folder ‘resources’ from the folder path “app > res”, select the path ‘New > Image Asset’. Asset Studio is pop up as the following image:

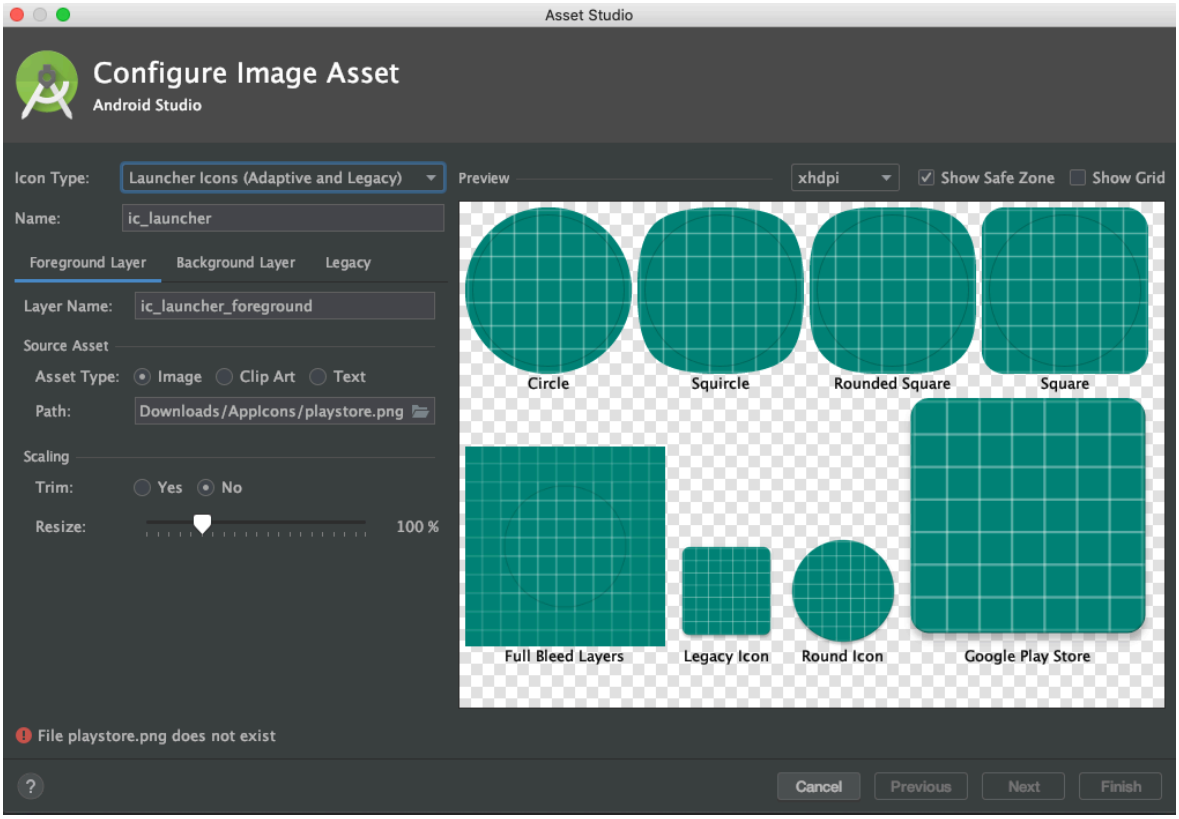


Figure 25. Asset Studio

Asset Studio allows developers to design the shape, colour or the image of the icon. To setup SplashScreen, it is a little bit complicated. Firstly, create an XML file for the splash screen. Let say “background_splash.xml” and paste the following code:

```xml
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:drawable="@color/splashscreen_bg"/>

    <item
        android:width="300dp"
        android:height="300dp"
        android:drawable="@mipmap/splash_icon"
        android:gravity="center" />

</layer-list>
```

Figure 26. background_splash.xml (Carli, 2020)

It creates a list of layers which includes a background colour and an icon with the size 300dpx300dp. All the app colours are defined in “res/values/colors.xml”:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="blue">#4F6D7A</color>
</resources>
```

Figure 27. colors.xml (Carli, 2020)

Next open “styles.xml” in the same folder as “colors.xml”, replace the code with this:

```xml
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="android:textColor">#000000</item>

        <!-- Add the following line to set the default status bar color for all the app. -->
        <item name="android:statusBarColor">@color/app_bg</item>
        <!-- Add the following line to set the default status bar text color for all the app
        to be a light color (false) or a dark color (true) -->
        <item name="android:windowLightStatusBar">false</item>
        <!-- Add the following line to set the default background color for all the app. -->
        <item name="android:windowBackground">@color/app_bg</item>
    </style>

    <!-- Adds the splash screen definition -->
    <style name="SplashTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="android:statusBarColor">@color/splashscreen_bg</item>
        <item name="android:background">@drawable/background_splash</item>
    </style>

</resources>
```

Figure 28. styles.xml (Carli, 2020)

Then, open the “AndroidManifest.xml” in the folder manifest to declare the link to splash activity:

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.splashexample"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="22" />

    <application
      android:name=".MainApplication"
      android:allowBackup="true"
      android:label="@string/app_name"
      android:icon="@mipmap/ic_launcher"
      android:theme="@style/AppTheme">

        <activity
            android:name=".SplashActivity"
            android:theme="@style/SplashTheme"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
          android:name=".MainActivity"
          android:label="@string/app_name"
          android:configChanges="keyboard|keyboardHidden|orientation|screenSize"
          android:windowSoftInputMode="adjustResize"
          android:exported="true"
        />

      <activity android:name="com.facebook.react.devsupport.DevSettingsActivity" />
    </application>

</manifest>
```

Figure 29. AndroidManifest.xml (Spencer Carli 2020)

Last but not least, create a new file called SplashActivity.java in “java/ProjectName” with the following content:

```java
package com.rn_splashscreen_tutorial; // Change this to your package name.

import android.content.Intent;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

public class SplashActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
        finish();
    }
}
```

Figure 30. SplashActivity.java (Carli, 2020)

- **Step 3:** Different platforms will have distinct ways to build and deploy the app and it may require paid account to host the app.
    - **iOS**: To build the universal app on iOS, follow the instruction on https://reactnative.dev/docs/running-on-device on section “Building your app production” for iOS. After the app is built, go to the personal Apple account on https://developer.apple.com/. A paid account is required so that the app is registered. Developer has to use Bundle Identifier of the project to register the app. Then a valid certificate ID is generated from Apple. With the registered ID, go to https://itunesconnect.apple.com/ to connect to the App Store. From there, developer can manage the app info and setup prize for the app on App Store. (React Native Documentation, 2020)

    - **Android:** To build universal app on Android, read the instruction from React Native document (https://reactnative.dev/docs/signed-apk-android#adding-signing-config-to-your-apps-gradle-config). After the build, take the release bundle with the file named “app.aab” from the directory “android/app/build/outputs/bundle/release”. Then, go to Google Play Console website, login with a paid account and register tha app with the release bundle taken from the project. Just like iTunes Connector, Google Play Console is the place where developers setup information and prize for the app. Now the app is launched in the market. . (React Native Documentation 2020)

    - **Web:** Run the command “yarn build:web” which is setup from “package.json”

      ```
      "build:web": "SKIP_PREFLIGHT_CHECK=true react-app-rewired build",
      ```

      The above command creates a build directory with the production build for the project. All the developers have to do is to pick a server host and register the app to publish it. There are a bunch of server hosting providers like Heroku, Netlify, Bluehost, Hostwinds, etc.

3.6 Result

The previous subsections of Section 3 have demonstrated a detailed instruction on how to build a universal application with React and React Native. A demo application is built and public on the writer's Github account. The app can be found at: https://github.com/NganHaVan/universaldemo. It is a simple app with basic few text and basic navigation. The next 3 images are the results of the demo app.

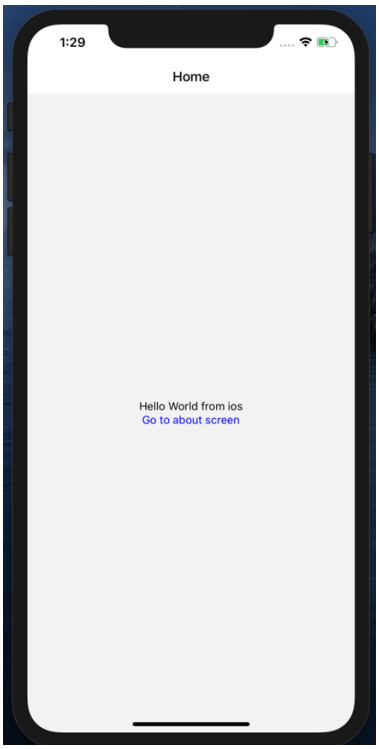


Figure 31. The demo app on iOS

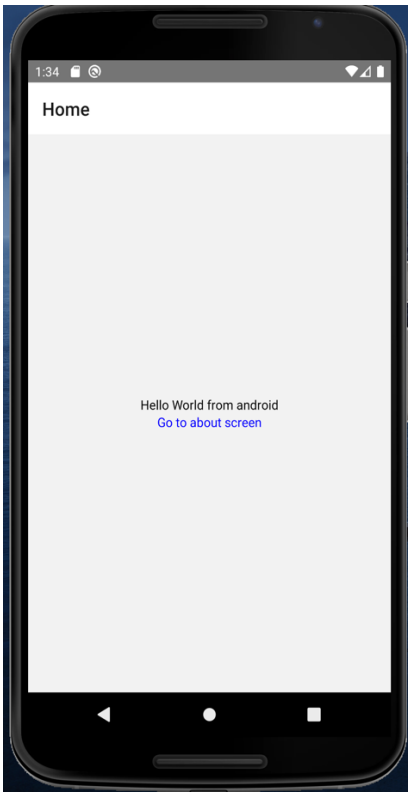


Figure 32. The demo app on Android

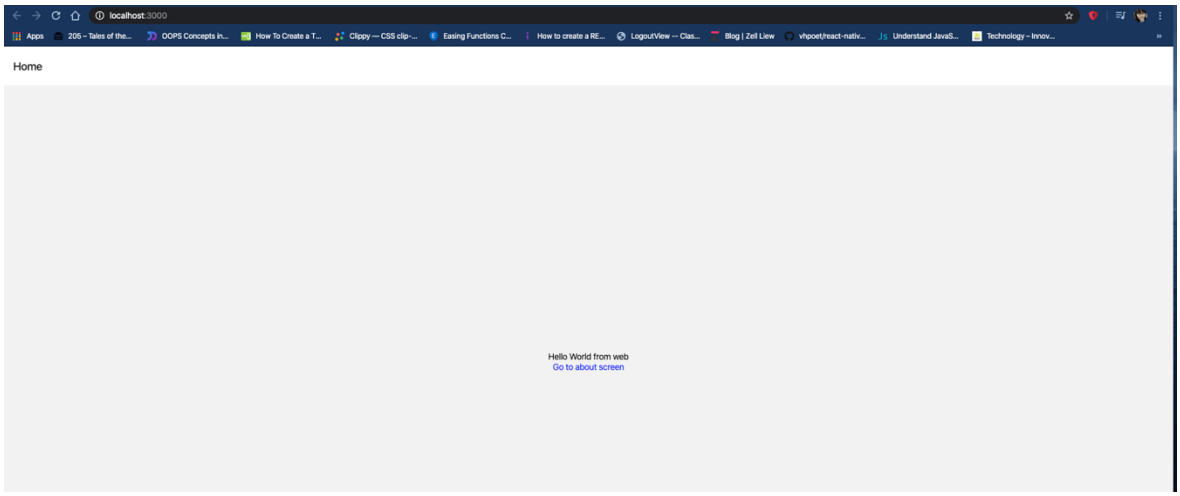


Figure 33. The demo app on web

4 Discussion

To sum up, section 2 unravelled how React and React Native operates on their host platform. From that knowledge, it is understandable how React Native for web handles platform-specific. After section 3, setting up a universal application with React and React Native has become coherent. From now on, the application can be managed under one project. Developers no longer have to duplicate their job. While maintaining and building a universal app, in addition to logic, responsiveness is a primary element that should be taken into serious consideration. In fact, there are a plethora of screen sizes on three separate platforms. Developers and designers should prepare a sharpened and flexible UI design. For long-term development, as the application becomes large, type checking tools are suggested to be implemented inside the project. The popular tools at the moment are TypeScript and Flow. These tools help mitigate errors to occur and assist developers in writing correct code.

From section 2, it is also known that Expo supports the universal application. Expo is a great option to start a universal project because it handles many headache tasks and provides smooth APIs. Vanilla React Native solution requires developers to have solid grasps in Babel and module bundlers so that they can write configuration files. Therefore, if developers have nothing to do with native modules written in native languages or customized SDK, Expo is an ideal solution. However, if native modules or any services out of Expo's support are mandatory in the project, vanilla React Native should be picked over Expo.

In the end, writing this thesis is an opportunity for the writer to reinforce and accumulate more knowledge about React and React Native. It provides more insights into picking the better technology for app development by knowing the pros and cons of different options in building a universal application. Hopefully, this thesis is helpful to developers or operations that are struggling in managing big applications. It is time they should consider universal app into their project to save cost and effort in app development.

References

Babel Documentation. URL: https://babeljs.io/docs/en/. Accessed: 19 April 2020

Carli, S 2017. How to add a Splash Screen to a React Native App (iOS and Android). URL: https://medium.com/handlebar-labs/how-to-add-a-splash-screen-to-a-react-native-app-ios-and-android-30a3cec835ae. Accessed: 24 April 2020

Eisenman, B. 2016. Learning React Native – Building Native Mobile Apps with JavaScript. O'Reilly

Expo Documentation. URL: https://docs.expo.io/versions/latest/. Accessed: 19 April 2020

Krajka, B. 2015. The difference between DOM and Virtual DOM. React Kung Fu. URL: https://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/. Accessed: 19 April 2020

Kruhlyk, Y. 2018. Expo vs Vanilla React Native: What to choose for your project. Apiko. URL: https://apiko.com/blog/expo-vs-vanilla-react-native/. Accessed: 24 April 2020.

MacQuarrie, A. 2018. Hybrid Apps vs Native Apps: What should you build? URL: https://themanifest.com/mobile-apps/hybrid-apps-vs-native-apps-which-should-you-build. Accessed: 24 April 2020

Rajput, M. 2018. Top Mobile App Development Frameworks in 2019-2020. URL: https://www.mindinventory.com/blog/mobile-app-development-framework-2019/. Accessed: 24 May 2020

React Documentation. URL: https://reactjs.org/. Accessed: 19 April 2020

React Native Documentation. Facebook. URL: https://reactnative.dev/. Accessed: 19 April 2020

React Navigation Documentation. URL: https://reactnavigation.org/docs/getting-started. Accessed: 24 April 2020

Sharma, R. 2018. Role of Metro Bundler in React Native. URL: https://medium.com/@rishabh0297/role-of-metro-bundler-in-react-native-24d178c7117e. Accessed: 24 April 2020

Sakpal, T. 2018. What is Document Object Model (DOM)? How JS interacts with DOM. URL: https://simplesnippets.tech/what-is-document-object-modeldom-how-js-interacts-with-dom/. Accessed: 19 April 2020

Stack Overflow Survey, 2019. URL: https://insights.stackoverflow.com/survey/2019. Accessed: 24 May 2020

Webpack Documentation. URL: https://webpack.js.org/concepts/. Accessed: 19 April 2020

Appendices

Appendix 1. Thesis structure

Cover page, abstract, table of contents
Introduction – Company introduction and solution – Objectives and goals – Out of scope.
Theoretical part – Web development with ReactJS – Mobile development with React Native – Expo solution for universal application
Empirical part – React Native Web – Instruction to build a universal application with React Native Web – Description of implementation – Result
Discussion – Consideration of results – Conclusions and suggestions for development – An evaluation of the thesis process and one's own learning.
References
Appendices – The product