UNIVERSITÉ DES MASCAREIGNES

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

DEPARTMENT OF SOFTWARE ENGINEERING

BSc (Hons) Software Engineering

Academic Year 2022/2023

Year 3 Semester 1

Mobile Application Development

Building your first React Native App

First of all create a folder on your computer where you are going to store all your codes for this module. Name the folder **MAD2022_<*your surname*>.** (**Note:** Avoid spaces in your name, use an underscore instead).

1. Navigate to the folder where you are going to store your mobile apps and type

   ```
   npx create-expo-app MyFirstApp
   ```

   at the command prompt. The name of our project is **MyFirstApp**. The command creates a blank project template. The project is created in a folder with the same name as the project name (`MyFirstApp`).

2. Navigate to the newly created folder:

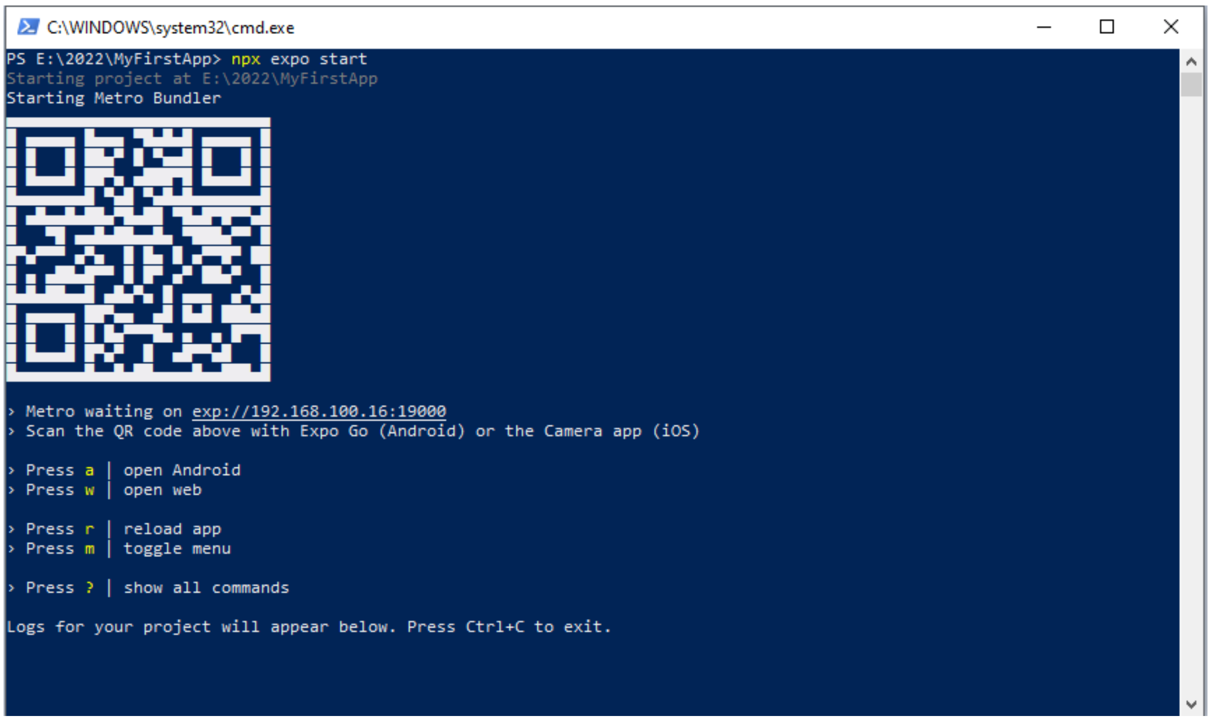
   ```
   cd MyFirstApp
   ```

3. Start the development server by typing:

   `npx expo start` (you can also use the command `npm start`)

   When you run `npx expo start`, Expo CLI uses Metro to bundle your JavaScript code and assets.

   A UI (called the Terminal UI) shows up which has a QR code and a list of keyboard shortcuts you can press. Here is a list of the important ones:



```
PS E:\2022\MyFirstApp> npx expo start
Starting project at E:\2022\MyFirstApp
Starting Metro Bundler

› Metro waiting on exp://192.168.100.16:19000
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Press a │ open Android
› Press w │ open web

› Press r │ reload app
› Press m │ toggle menu

› Press ? │ show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
```

- Press **a** to open app on an Android emulator
- Press **i** to open app on an iOS emulator
- Press **r** to reload the app

- Press **`m`** to open the dev menu on any connected device
- Press **`shift + a`** to select an Android emulator to open

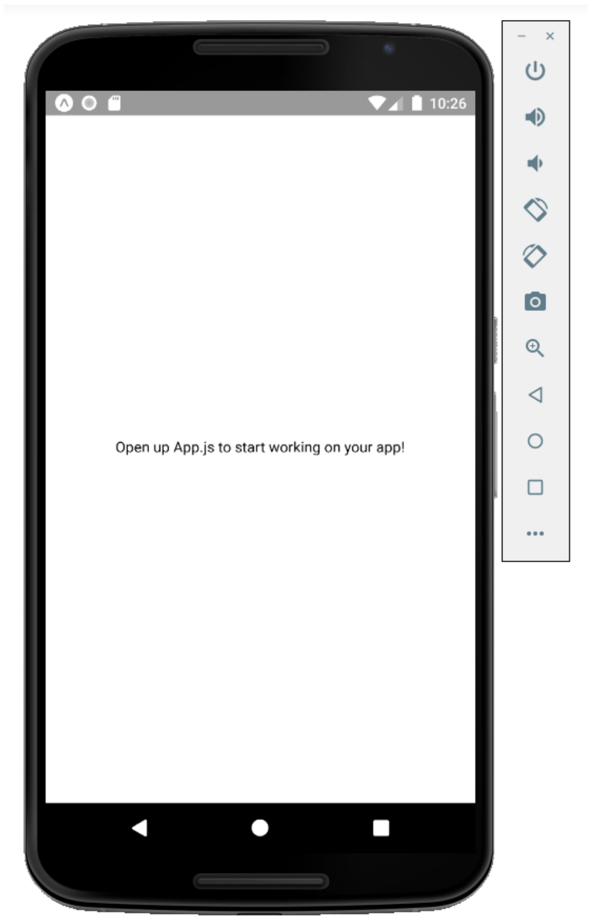
4. Run your app on an Android emulator

Open an Android Virtual Device from Android Studio.

In the Terminal UI, press **`a`** to open app on the Android emulator

In the Terminal UI you will see the JavaScript bundler begin packaging your app.

When your app is bundled, you should see something like this in your emulator.



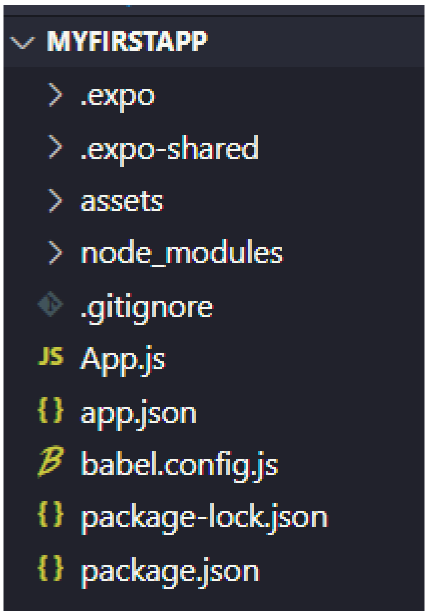
Running your app on a real device

- On your Android device, press "Scan QR Code" in the Expo Go app and scan the QR code you see in the terminal.
- On your iPhone or iPad, open the default Apple Camera app and scan the QR code you see in the terminal.

Viewing and updating the source code

1. Start Visual Studio Code.

2. From the menu, use **File > Open Folder**... to open the **MyFirstApp** folder (your project folder). You will see the following folder structure:



```
MYFIRSTAPP
  > .expo
  > .expo-shared
  > assets
  > node_modules
    .gitignore
    App.js
    app.json
    babel.config.js
    package-lock.json
    package.json
```

Let us go through each of these files/folders:

- `.expo` contains files that define configurations for the Expo packager and for device connection settings.
- `assets` contains a few image assets by default. These are the app icon and splash screen images.
- `node_modules` directory contains all the packages required by our project. Any new dependencies and development dependencies go here.
- `.gitignore` is where we specify which files should be ignored by Git. We can see that both the `node_modules/` and `.expo/` directories are already included.
- `App.js` is where our application code lives. Whatever you write inside this file, it will get displayed on the mobile device.
- `app.json` is a configuration file that allows us to add information about our Expo app. The list of properties that can be included in this file is listed in the documentation which can be found at

  https://docs.expo.dev/workflow/configuration/.

  Examples of properties that can be changed here include the app icon and splash screen.
- `babel.config.js` allows us to define presets and plugins for configuring Babel (https://babeljs.io/)

- `package.json` file is a configuration file for NPM's use. It defines the name of our project, its version and source code repository location, what JavaScript file represents as its main entry point, the dependencies it has,. Basically, it contains all the relevant metadata about our project.

Let's take a closer look at the generated `package.json` file:

```
{
  "name": "myfirstapp",
  "version": "1.0.0",
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web",
    "eject": "expo eject"
  },
  "dependencies": {
    "expo": "~46.0.9",
    "expo-status-bar": "~1.4.0",
    "react": "18.0.0",
    "react-native": "0.69.5",
    "react-native-web": "~0.13.12"
  },
  "devDependencies": {
    "@babel/core": "^7.12.9"
  },
  "private": true
}
```

The `scripts` property contains all the commands needed to run the application.

The `dependencies` and `devDependencies` properties define the application and development dependencies respectively. Expo, the React core library and the version of React Native needed for this specific version of Expo are all included as dependencies.
Let us have a look at the `App.js` file:

```
import StatusBar from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style= 'auto'>
    </View>
  );
11  }
12
13  const styles = StyleSheet.create({
14    container: {
15      flex: 1,
16      backgroundColor: '#fff',
17      alignItems: 'center',
18      justifyContent: 'center',
19    },
    });
```

- Line 1: We import the `StatusBar` component which allows us to control the status bar (change its text colour, background colour, hide it, etc.).
- Line 2: We import some basic components (`Stylesheet`, `Text` and `View`) from react-native.
- The `View` component renders a container. It maps to the fundamental native iOS (`UIView`) and Android (`View`) components. . You can think of this component as an HTML `div` element which contains other elements. A `View` component can contain nested components.
- The `Text` component allows us to render text.
- The `StyleSheet` component provides an API to create styles. There are no classes or ids in React Native like in web development. To create a new style object you use the `StyleSheet.create()` method. React Native uses camelCase instead of kebab-case for style properties (For example, we use `backgroundColor` instead of `background-color`).

  React Native uses Flexbox to handle layout. Flexbox makes it easy to distribute the UI elements in the container.

  The first style that we apply is `flex: 1`, the `flex` property will define how your items are going to "fill" over the available space along your main axis. Since we only have one container, it will take all the available space of the parent component. In this case, it is the only component, so it will take all the available screen space.

  The style `justifyContent: "center"` aligns children of a container in the centre of the container's main axis and `alignItems: "center"` aligns children of a container in the centre of the container's cross axis.

  Styling and layouts will be discussed in depth in a separate session.

- Lines 6-9 show JSX (JavaScript XML) code. JSX is an XML-like extension to the ECMAScript specification.

  A JSX Element will take the following form:

```
<JSXElement>
   <ChildJSXElement />
   <ChildJSXElement />
   <ChildJSXElement />
</JSXElement>
```

The JSX specification defines the following:

- The JSX Elements can be either self-opening `<JSXElement></JSXElement>` or self-closing `<JSXElement />`.

- An attribute of a JSX element can be an expression `{}` or a string `""`. For example:

  ```
  <JSXElement attr="value" />

  <JSXElement attr={expression} />
  ```

  Expressions are JavaScript code snippets.

- The children elements can be text, expressions, or JSX elements.

- JSX is not readable by the browser. Before our code can be interpreted by the browser, it needs to be converted from JSX into JavaScript. This is achieved by Babel.

3. Change the text "*Open up App.js to start working on your app!*" to "*My first mobile app!*" The text is immediately updated on your device/emulator.

   By default, the Expo CLI comes with live reload enabled. This means if you edit and save any file, the application on your mobile device or emulator will automatically reload. Moreover, any build errors and logs will be displayed directly in the terminal.

   If your app is not automatically reloading, try the following:

   - Close the Expo app and reopen it.
   - Show the Developer Menu. This can be achieved in several ways:

     - Terminal UI: Press **`m`** in the terminal to open the menu on connected iOS and Android
     - Android Device: Shake the device vertically a little bit.
     - Android Emulator: Press **`Ctrl + M`** or run `adb shell input keyevent 82` in your terminal window.
     - iOS Simulator: Press **`Cmd ⌘ + D`** to simulate the shake gesture
     - iOS Device: Shake the device a little bit.

   - If you see **Enable Fast Refresh**, press it. If you see **Disable Fast Refresh**, dismiss the developer menu and edit your source code and save your file.