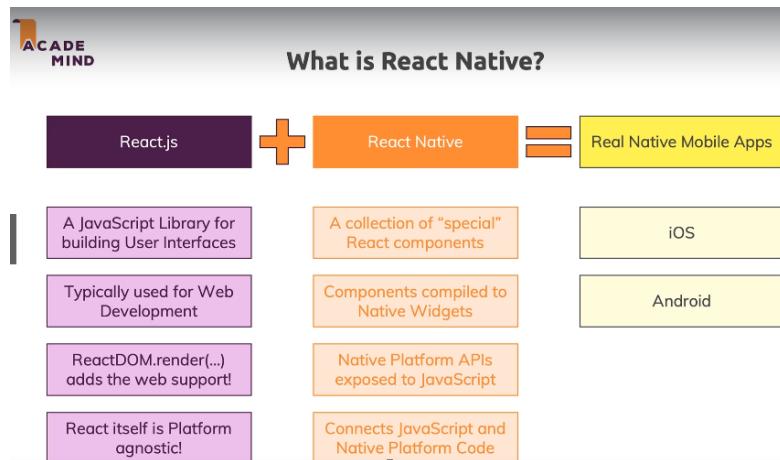


React Native - The Practical Guide

Intro

What is React Native



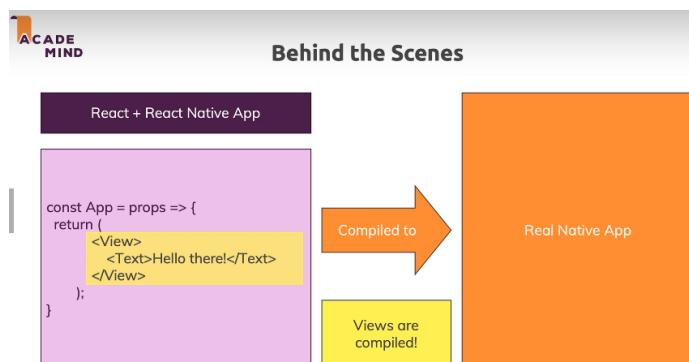
Remember React.js is only a library to build user interfaces. It does not render at any specific environment (it's platform agnostic)! React for JS development uses ReactDOM library to mount itself, for example.

Which means React is an abstract conceptualization, which is capable of rendering anywhere given the proper library to do so!

React Native is a collection of React components that compile to Native Widgets, that is, to render in a native environment. It also gives you access to native apps (like the device camera and GPS location), and give you tools to bridge Javascript to native language.

If you combine ReactJS and React Native, there you have everything you need to create and run Real Native Mobile Apps

How React Native Works



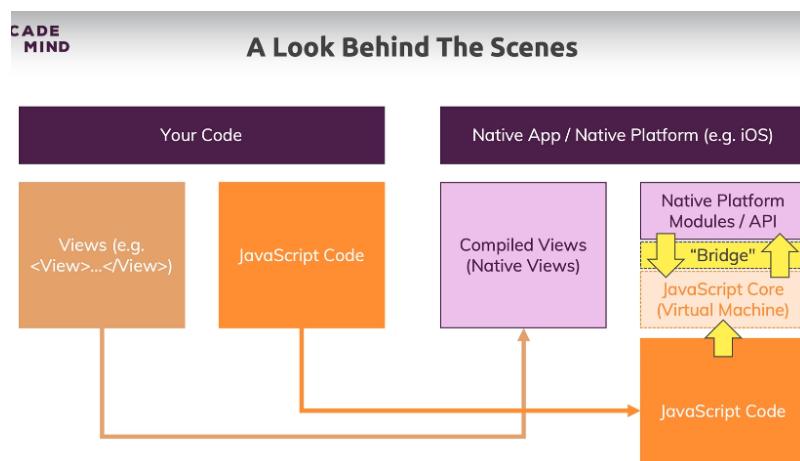
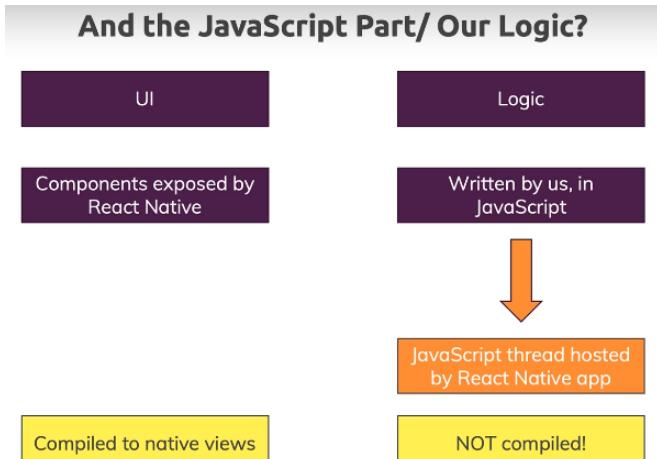
Unlike normal React.JS components in Web, React Native gives you custom components that are already linked to -and handles- the whole native logic. They compile to native widgets, elements, code.

A table of translations between web - android - iOS - React Native:

ACADE MIND

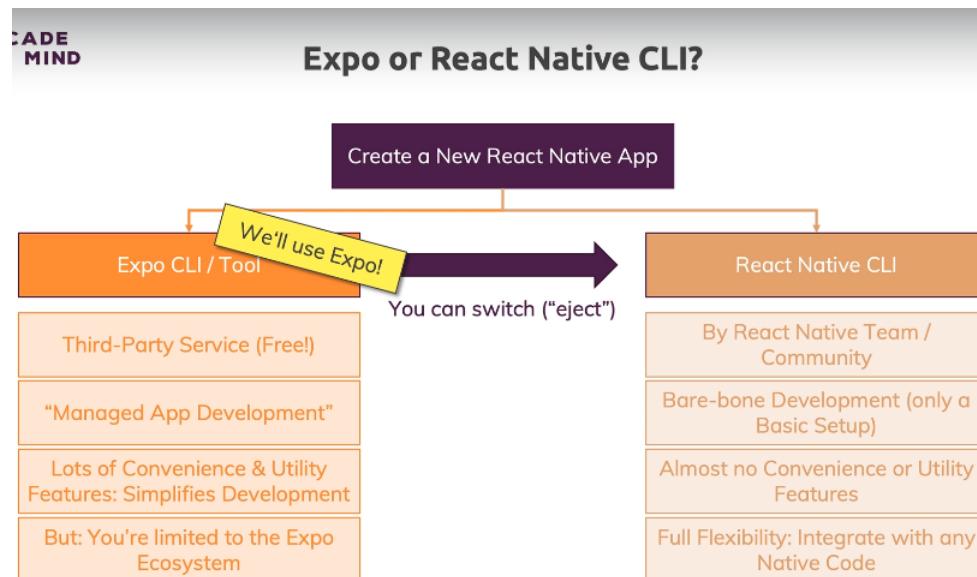
React for the Web	Native Component (Android)	Native Component (iOS)	React Native
<div>	android.view	UIView	<View>
<input>	EditText	UITextField	<TextInput>
...

Your additional Javascript code is NOT compiled! Instead, it is run in a special thread of React Native. This means that all views are compiled into native widgets, but the logic is constantly run and handled by React Native Thread!



So, to sum it up: (1) Views are compiled to native code, (2) Javascript code constantly runs in a native thread which "bridges" (communicates) with Native Modules / APIs.

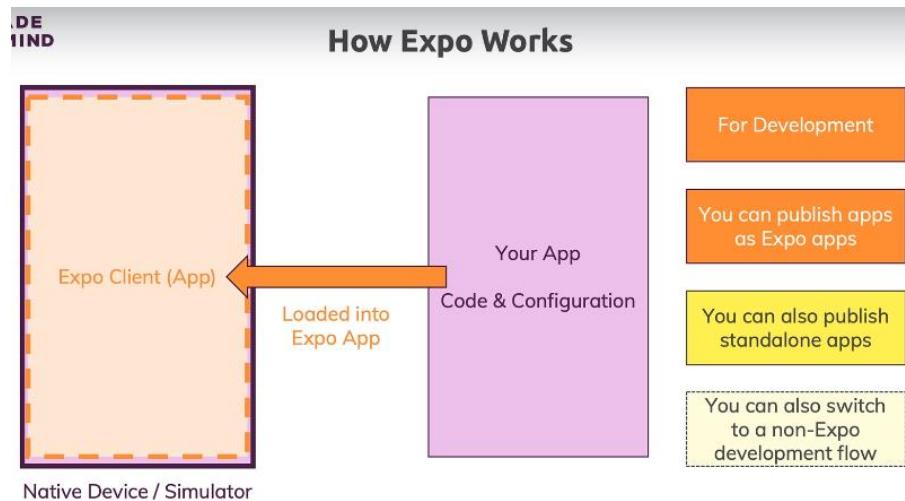
Expo or React Native CLI?



Expo is a wrapper around your app, an extra layer. It gives you access to plenty of native widgets and apis, but you are limited to its ecosystem. It removes some of the fine grain control, but offers you a lot of flexibility in time, in return. Plenty of components and utilities are there bridged and ready to use.

RN CLI is made by the React Team, it gives you a bare-bone setup, which needs the respective setups (IDEs), no features or convenience utilities, but it offers you fine grain control on all native widgets, as well as easier integration with any native code already in existence (Expo would require you to eject).

If you are getting started, Expo is the right choice. You can always eject to React Native CLI by ejecting. That's why we are to use Expo to build apps here.



Behind the scenes, Expo uses a simulator to run your app (code and configs). You can also publish your app to Expo, so anyone developing with expo can also run it without downloading it.

Of course, Expo helps you to build and distribute standalone apps, directly to the iOS and Android stores.

Node JS download

At the moment, the Node webpage (nodejs.org) which we're going to use in the next lecture looks different.

This will only be the case temporarily (as you can tell if you visit the page) but in the meantime, since you're going to need to download NodeJS from the site, here's the download link: <https://nodejs.org/en/download/>

Creating our first React Native App

1. Go to nodejs.org and install it.

2. On a new folder, install expo cli globally:

```
$ (sudo) npm install expo-cli --global
```

3. Create a project:

```
$ expo init rn-first-app  
> expo-template-blank
```

4. **\$ cd rn-first-app && npm start**

Leave the process running in the terminal. This is such because an auto-updater runs each time you save your project, as it constantly runs in a watcher server.

5. **Download the Expo Client app** from the app store.

6. Open the app and **scan the QR code** given to you by Expo when you initialized the app. This compiles your app shows it in your device.

Working our first App

View and Text are compiled to native code, while StyleSheet is a utility function that takes an object with styles and assigns those styles to components rendered as UI.

Starting example:

```
// import { StatusBar } from 'expo-status-bar';  
import React, { useState } from "react"  
import { StyleSheet, Text, View, Button } from "react-native"  
  
export default function App() {  
  const [text, setText] = useState("")  
  return (  
    <View style={styles.container}>
```

```

        <Text>{text}</Text>
        <Button title="Change text" onPress={() => setText("asd")}></Button>
        /* <StatusBar style="auto" /> */
    </View>
)
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        backgroundColor: "#fff",
        alignItems: "center",
        justifyContent: "center"
    }
})

```

React Native Apps are Hard Work

Expo does make things a lot easier, but it's important to know that React Native is not about Writing code once and run it everywhere, but all about Learning once and writing everywhere.

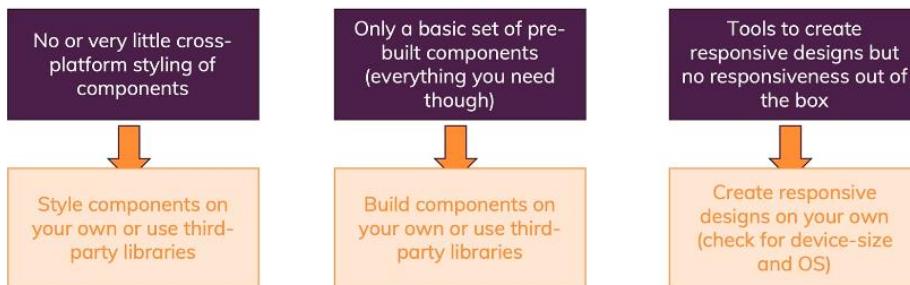
Components adjust themselves to any platform with Expo, but you will have to learn code that's flexible depending the platform on the go.



React Native Apps are Hard Work!

— Write JavaScript code and run it everywhere?

“Learn once, write everywhere”

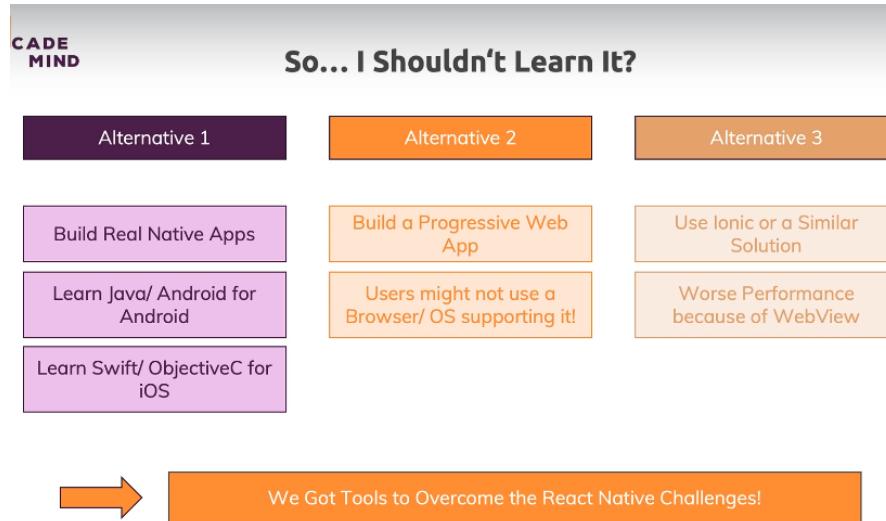


React Native alternatives

React Native keeps on changing. It is inevitable to go back to an app you build half a year ago to update it, breaking changes can and will happen.

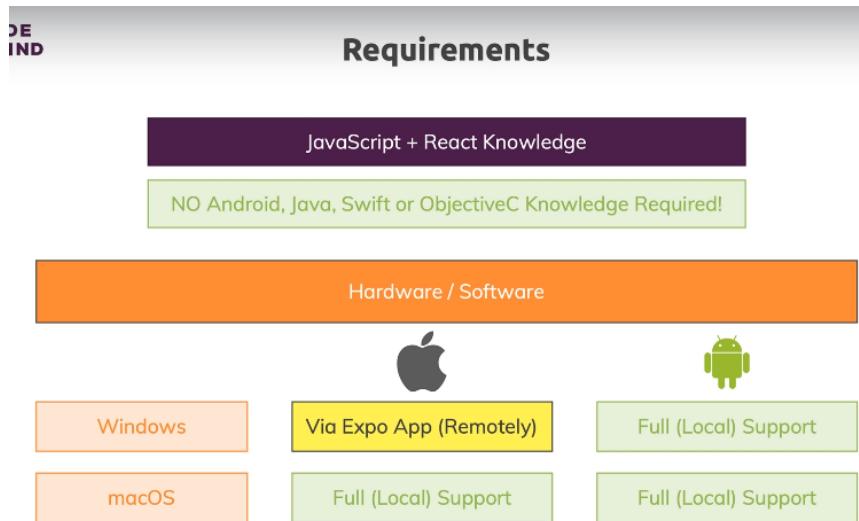
It is also highly dependant on third-party packages, that can also change. Though Expo updating generally solves this issue. However, you still have the extra dependencies you add, to be adjusted.

Sometimes, there also are bugs in RN, which you will have to keep an eye on and workaround.



Ionic provides a native wrapper for web apps to be shipped to app stores. Apps can be ran as mobile ones, but they will still appear as web apps.

Course requirements



Running the app on Android Emulator

For development, we need a simulator. It is probably not too efficient to fully develop using your own device.

For those, we will either need XCode (iOS simulator) to be run on MacOS, and/or Android Studio, for Android development, on any device.

<https://expo.io> documentation holds instructions for both.

1. Download and install Android Studio.

You need these steps on MacOS and Linux:

- If you are on macOS or Linux, add the Android SDK location to your PATH using `~/.bash_profile` or `~/.bashrc`. You can do this by adding a line like `export ANDROID_SDK=/Users/myuser/Library/Android/sdk`.
- On macOS, you will also need to add `platform-tools` to your `~/.bash_profile` or `~/.bashrc`, by adding a line like `export PATH=/Users/myuser/Library/Android/sdk/platform-tools:$PATH`

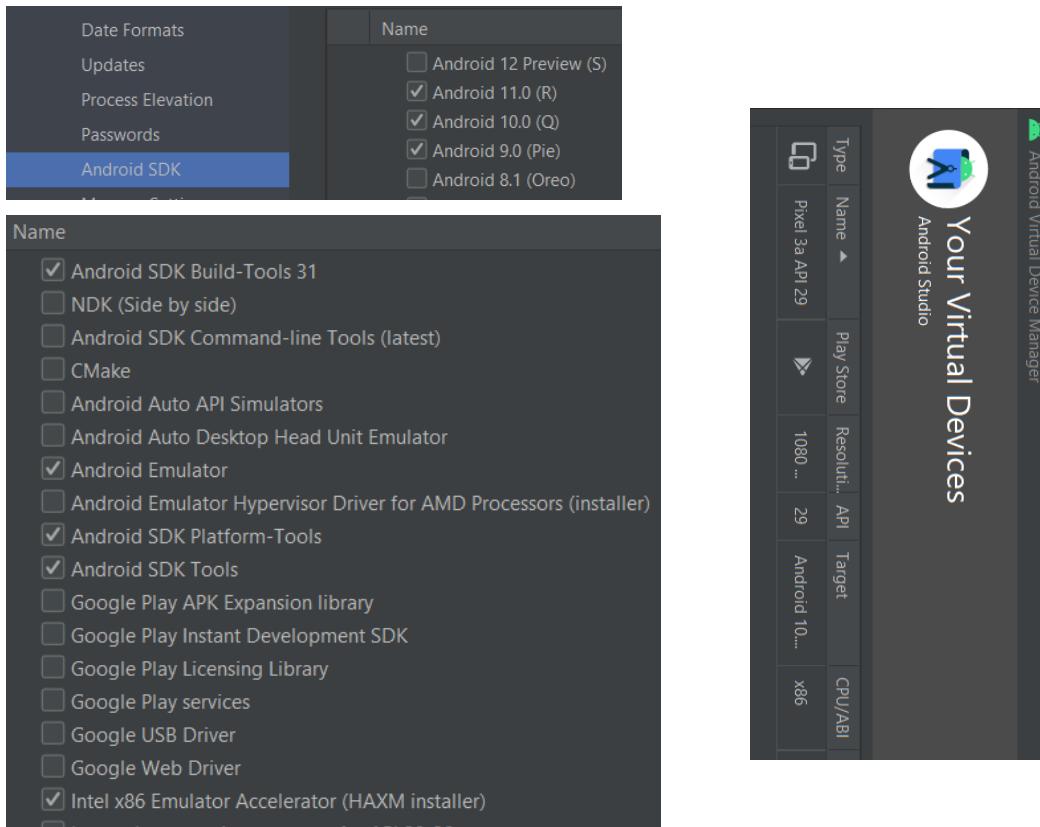
2. Open AS and on the opened window, tap Configure > SDK Manager

Download the latest stable SDKs, and also Android Emulator, SDK Platform, SDK Tools and SDK Build Tools on SDK Tools. Additionally, you need the intel x86 Emulator Accelerator. Tap apply.

3. Now, Configure > ADB manager.

Create different devices with different sizes (select ones with play store included), and configure them with the latest stable Android OS (these are device images, independent of the other ones you installed before).

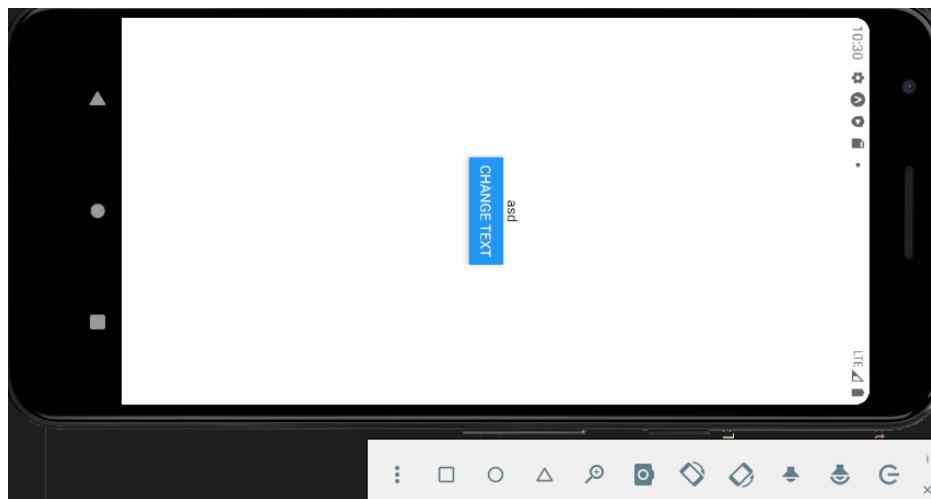
Once installed, emulators can be launched with the play button on the emulator screen.



4. Once the emulator launched, run it. Go to Expo in localhost, and run the app on Android Emulator, it should run in the emulated device.

Or, on your terminal, press "`a`", and the emulator should open.

Expo client will automatically install and run in the emulator.



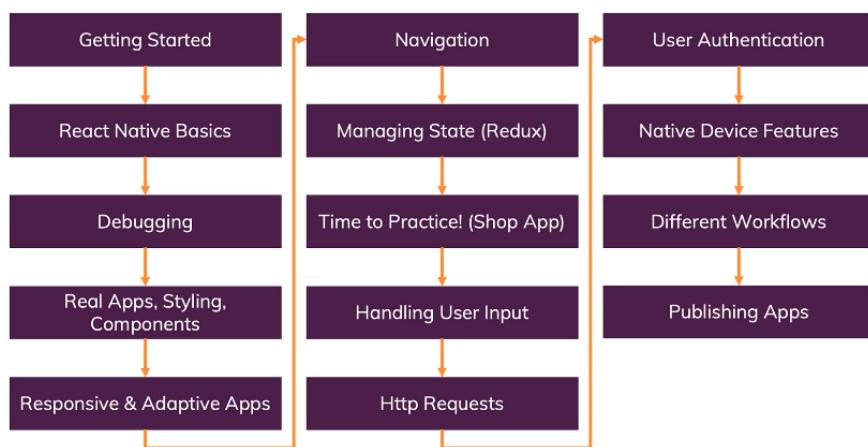
Running App on iOS simulator

Note that all this only works in MacOS.

1. Go to app store and search for "XCode", install the developer tools.
2. Open XCode and go to Preferences. Enable Command-Line Tools there.
3. Launch the simulator

Developer Tools > Simulator
Hardware > device > (select device) -> changes the device
4. Go to Expo devtools and launch the app on iOS simulator.
Alternatively, you can tap "i" on the command line.

Course outline



Useful links

Official Expo Docs:

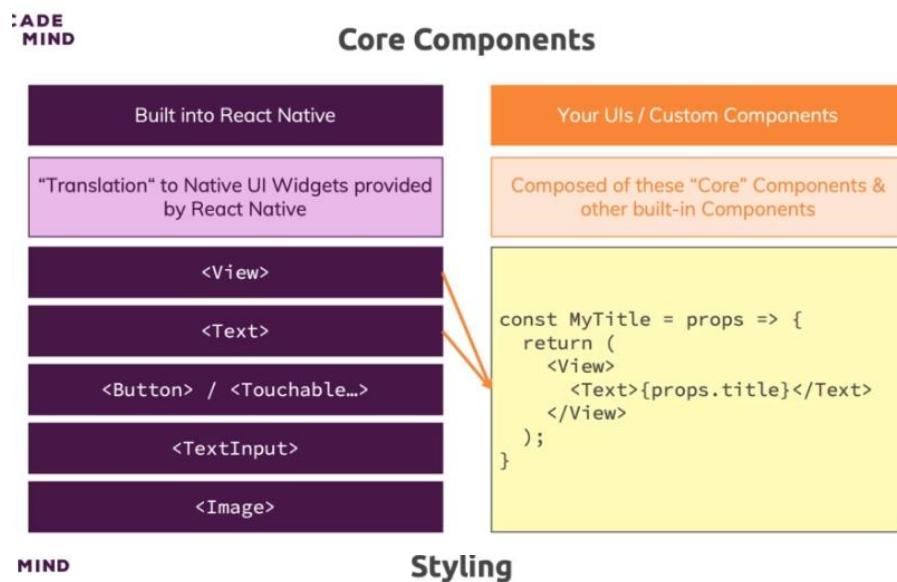
<https://docs.expo.io/versions/v34.0.0/introduction/installation/>

Official React Native Docs:

<https://facebook.github.io/react-native/docs/getting-started>

Diving into basics

How to work with React Native Components



Setting up a new project

1. `$ expo init rn-complete-guide && npm start`



Working with core components

View is like a div for web development. You can wrap other components and add styles.

Text is a generic component to display text. You cannot output text without being wrapped in this component, this is a strong difference from web development.

Getting started with styles

Let's start by creating a TextInput and a Button with some stylings.

Note you can always check the available styles for each component in the documentation.

App.js

```
import React from "react"
import { StyleSheet, TextInput, View, Button } from "react-native"

export default function App() {
  return (
    <View>
      <View style={{ padding: 50 }}>
        <TextInput
          placeholder="Example placeholder"
          style={{
            borderBottomColor: "black",
            borderBottomWidth: 1,
            padding: 10,
            marginVertical: 10
          }}
        />
      </View>
    </View>
  )
}
```

```

        } }

      />
      <Button title="add" />

    </View>
    <View></View>

  </View>

)

}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#fff",
    alignItems: "center",
    justifyContent: "center"
  }
})

```

Example placeholder



Flexbox and layouts

Flex stretches over the cross axis by default.

If `flexDirection` is `row`, components are stretched over the Y axis, and viceversa.

Flex: 1 will stretch the container as much as possible. All of its siblings will only take the necessary width to render themselves visible (with their intrinsic dimensions, or their children).

The number value is itself * 100 of the available space once the least necessary space to render everything is calculated.

If one child has `flex 2`, another one `flex 1`, and the last one, `flex 3` -> first child takes 2/6 of the available space, child 2, 1/6 and last child 3/6

App.js

```

export default function App() {
  return (
    <View
      style={{
        padding: 70, width: "80%", height: 200,
        flexDirection: "row",
        justifyContent: "space-between",
        alignItems: "stretch"
      }}
    >

```



```

<View
  style={ {
    backgroundColor: "blue",
    flex: 1, justifyContent: "center", alignItems: "center"
  } }
>
  <Text>1</Text>
</View>
<View
  style={ {
    backgroundColor: "green",
    flex: 2, justifyContent: "center", alignItems: "center"
  } }
>
  <Text>2</Text>
</View>
<View
  style={ {
    backgroundColor: "red",
    flex: 3, justifyContent: "center", alignItems: "center"
  } }
>
  <Text>3</Text>
</View>
</View>
)
}

```

Inline styles and stylesheet objects

We cannot use inline styles all the time, it gets tricky to follow. Instead, we use StyleSheet objects, which configures a CSS-in-JS style syntax to pass as classes to RN components.

App.js

```

import React from "react"
import { StyleSheet, TextInput, View, Button } from "react-native"

export default function App() {
  return (
    <View style={styles.container}>
      <View style={styles.inputContainer}>

```

```

        <TextInput
            placeholder="Example placeholder" style={styles.input} />
        <Button title="add" />
    </View>
</View>
)
}

const styles = StyleSheet.create({
    container: { padding: 70 },
    inputContainer: {
        flexDirection: "row", justifyContent: "space-evenly",
        alignItems: "center"
    },
    input: {
        borderBottomColor: "black", borderBottomWidth: 1,
        padding: 10, width: "75%"
    }
})

```

Working with state and events

Enough stylings for now, let's add some functionality, starting with the input.

App.js

```

import React, { useState } from "react"
import {
    StyleSheet, TextInput, Text, View, Button, ScrollView
} from "react-native"

export default function App() {
    const [text, setText] = useState("")
    const [texts, setTexts] = useState([])

    function changeText(text) {
        setText(text) // value is already passed as a string, not an event
    }

    function handleSubmit() {
        setTexts((prevTexts) => [...prevTexts, text])
    }
}

```

```

return (
  <View style={styles.container}>
    <View style={styles.inputContainer}>
      <TextInput
        placeholder="Example placeholder" style={styles.input}
        value={text} onChangeText={changeText}
      />
      <Button title="add" onPress={handleSubmit} />
    </View>
    <ScrollView>
      {texts.map((t, i) => (
        <View style={styles.listItems} key={i}>
          <Text>{t}</Text>
        </View>
      )))
    </ScrollView>
  </View>
)
}

const styles = StyleSheet.create({
  container: { padding: 70 },
  inputContainer: {
    flexDirection: "row", justifyContent: "space-around",
    alignItems: "center"
  },
  input: {
    borderBottomColor: "black", borderBottomWidth: 1,
    padding: 10, width: "75%"
  },
  listItems: {
    padding: 10, margin: 10, borderColor: "crimson", borderWidth: 1
  }
})

```

FlatList

ScrollView is great to scroll a little amount of components, as it renders them all to the screen. So, on a list that's capable of having many components, we need a more efficient solution: FlatList.

FlatList handles lots of rendering logic, but expects an object as item inside the array state, with a key to use when mapping.

App.js

```
import React, { useState } from "react"
import {
  StyleSheet, TextInput, Text, View, Button, FlatList
} from "react-native"

export default function App() {
  const [text, setText] = useState("")
  const [texts, setTexts] = useState([])

  function changeText(text) { setText(text) }

  function handleSubmit() {
    setTexts((prevTexts) => [
      ...prevTexts,
      { key: Math.random().toString(), value: text }
    ])
  }

  return (
    <View style={styles.container}>
      <View style={styles.inputContainer}>
        <TextInput
          placeholder="Example placeholder"
          value={text} onChangeText={changeText}
          style={styles.input}
        />
        <Button title="add" onPress={handleSubmit} />
      </View>
      <FlatList
        data={texts}
        renderItem={(data) => ( // -> automatically sets key
          <View style={styles.listItems}> //           if defined in data
            <Text>{data.item.value}</Text> // -> data.item!
          </View>
        ) }
      />
    </View>
  )
}
```

Alternatively, you can pass a `keyExtractor` function to calculate the key for each item.

```
<FlatList
  keyExtractor={(data, i) => data.key} // -> this is default
  data={texts}
  renderItem={(data) =>
    <View style={styles.listItems}>
      <Text>{data.item.value}</Text>
    </View>
  }
/>
```

Splitting the app into components

Time to build our own components to reuse them and spread the logic into different files.

App.js

```
import React, { useState } from "react"
import { StyleSheet, View, FlatList } from "react-native"
import TextItem from "./components/TextItem"
import TextField from "./components/TextField"

export default function App() {
  const [texts, setTexts] = useState([])

  function handlePress(text) {
    setTexts((prevTexts) => [
      ...prevTexts,
      { key: Math.random().toString(), value: text }
    ])
  }

  return (
    <View style={styles.container}>
      <TextField
        buttonProps={{ title: "add", onPress: handlePress }} />
      <FlatList
        keyExtractor={(data, i) => data.key}
        data={texts}
```

```

        renderItem={(data) => <TextItem>{data.item.value}</TextItem>}
      />
    </View>
  )
}

const styles = StyleSheet.create({ container: { padding: 70 } })

TextField.js
import React, { useState } from "react"
import { View, Button, TextInput, StyleSheet } from "react-native"

export default function TextField{
  inputProps = {}, buttonProps = {} )
{
  const [text, setText] = useState("")

  function handleChangeText(text) { setText(text) }

  return (
    <View style={styles.container}>
      <TextInput
        placeholder={inputProps?.placeholder || "Example placeholder"}
        value={text} onChangeText={handleChangeText}
        style={styles.input} {...inputProps}
      />
      <Button
        {...buttonProps}
        onPress={buttonProps?.onPress?.bind(this, text)}
      />
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flexDirection: "row", justifyContent: "space-evenly",
    alignItems: "center"
  },
  input: {

```

```

        borderBottomColor: "black", borderBottomWidth: 1,
        padding: 10, width: "75%"
    }
))

TextItem.js

import React from "react"
import { View, Text, StyleSheet } from "react-native"

export default function TextItem({ children }) {
    return (
        <View style={styles.container}>
            <Text>{children}</Text>
        </View>
    )
}

const styles = StyleSheet.create({
    container: {
        padding: 10, margin: 10, borderColor: "crimson",
        borderWidth: 1
    }
})

```

Working with touchable components

Let's try to delete elements now, and also use this opportunity to introduce touchable functionality.

`View` is not pressable by default. There are special components for this, which conveniently wrap on almost any other children: **Touchable**.

And even better, if you wish to use them as full components with added UI visibility feedback you have **TouchableOpacity**, **TouchableHighlight**, **TouchableWithoutFeedback** (has touchable functionality but no visual feedback) and **TouchableNativeFeedback** (this last one is the ripple effect that only works on Android).

activeOpacity prop controls the opacity intensity of Opacity.
underlayColor props controls the background color of Highlight.

App.js

```

export default function App() {
    const [texts, setTexts] = useState([])

    function handlePress(text) {

```

```

    setTexts((prevTexts) => [
      ...prevTexts,
      { key: Math.random().toString(), value: text }
    ])
  }

  function handleDeleteText(targetKey) {
    setTexts(texts.filter((t) => t.key !== targetKey))
  }

  return (
    <View style={styles.container}>
      <TextField buttonProps={{ title: "add", onPress: handlePress }} />
      <FlatList
        data={texts}
        renderItem={({ data }) => (
          <TextItem
            onDeleteText={handleDeleteText.bind(this, data.item.key)}
            data.item.value
          </TextItem>
        )}
      />
    </View>
  )
}


```

The screenshot shows a mobile application interface. At the top right is a blue 'ADD' button. Below it is a list of four text items, each enclosed in a red-bordered box:

- adsas12344
- adsas1
- adsas123
- adsas12344

TextItem.js

```

export default function TextItem({ children, onDeleteText }) {
  return (
    <TouchableOpacity activeOpacity={0.8} onPress={onDeleteText}>
      <View style={styles.container}>
        <Text>{children}</Text>
      </View>
    </TouchableOpacity>
  )
}

```

Adding a modal overlay

Modal is a built in, easy to use, RN component capable of adding a modal screen. Let's use it to move input functionality there.

App.js

```
import React, { useState } from "react"
import { StyleSheet, View, FlatList, Button } from "react-native"
import TextItem from "./components/TextItem"
import TextField from "./components/TextField"

export default function App() {
  const [texts, setTexts] = useState([])
  const [isModalOpen, setIsModalOpen] = useState(false)

  function handlePress(text) {
    setTexts((prevTexts) => [
      ...prevTexts,
      { key: Math.random().toString(), value: text }
    ])
    setIsModalOpen(false)
  }

  function handleDeleteText(targetKey) {
    setTexts(texts.filter((t) => t.key !== targetKey))
  }

  function closeModal() { setIsModalOpen(false) }

  return (
    <View style={styles.container}>
      <Button title="Open" onPress={() => setIsModalOpen(true)} />
      <TextField
        modalProps={{ visible: isModalOpen }}
        cancelButtonProps={{ onPress: closeModal }}
        addButtonProps={{ title: "add", onPress: handlePress }}
      />
      <FlatList
        data={texts}
        renderItem={({ data }) => (
          <TextItem
```

```

        onDeleteText={handleDeleteText.bind(this, data.item.key)}>
        {data.item.value}
      </TextItem>
    ) }
  />
</View>
)
}

```

```
const styles = StyleSheet.create({ container: { padding: 70 } })
```

TextField.js

```

import React, { useState } from "react"
import { View, Button, TextInput, StyleSheet, Modal } from "react-native"

```

```

export default function TextField({
  modalProps = {},
  inputProps = {},
  addButtonProps = {},
  cancelButtonProps = {}
}) {
  const [text, setText] = useState("")

```

```
  function handleChangeText(text) { setText(text) }
```

```
  function addNewText() {
```

```
    addButtonProps?.onPress?.(text)
```

Example placeholder

```
    setText("")
  }
```

```
  return (
```

```
<Modal animationType="slide" {...modalProps}>
```

```
  <View style={styles.container}>
```

```
    <TextInput
```

```
      placeholder={inputProps?.placeholder || "Example"}
```

```
      placeholder"}
```

```
      value={text}
```

```
      onChangeText={handleChangeText}
```

```
      style={styles.input}
```

```
      {...inputProps}
```

CANCEL

ADD

OPEN



```

        />

      <View style={styles.buttonsContainer}>
        <View style={styles.button}>
          <Button
            title="cancel"
            color="red"
            onPress={cancelButtonProps?.onPress}
          />
        </View>
        <View style={styles.button}>
          <Button {...addButtonProps} onPress={addNewText} />
        </View>
      </View>
    </Modal>
  )
}

```

```

const styles = StyleSheet.create({
  container: {
    flex: 1, // <- as much space as parent gives it
    justifyContent: "center",
    alignItems: "center"
  },
  input: {
    borderBottomColor: "black", borderBottomWidth: 1,
    padding: 10, width: "75%", marginBottom: 10
  },
  buttonsContainer: {
    width: "75%", flexDirection: "row", justifyContent: "space-evenly"
  },
  button: { width: "40%" // <- buttons need View to apply styles! }
})

```

TextItem.js:

```

import React from "react"
import { View, Text, StyleSheet, TouchableOpacity } from "react-native"

export default function TextItem({ children, onDeleteText }) {
  return (

```

```

<TouchableOpacity activeOpacity={0.8} onPress={onDeleteText}>
  <View style={styles.container}>
    <Text>{children}</Text>
  </View>
</TouchableOpacity>
)
}

const styles = StyleSheet.create({
  container: {
    padding: 10, margin: 10, borderColor: "crimson", borderWidth: 1
  }
})

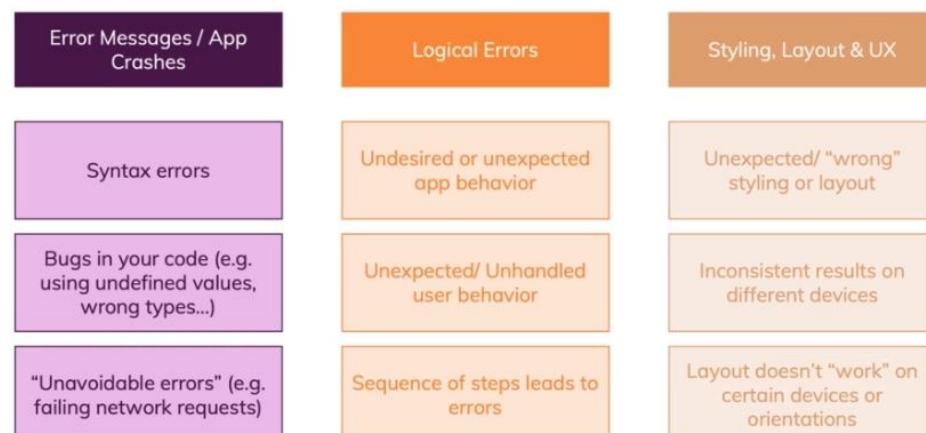
```

Debugging React Native apps

What to and how to debug

:ADE
MIND

What To Debug?



Read the error messages (seriously)!

Often, you the error message contains the solution or a (pretty) exact pointer at the problematic code line.

console.log()

Get a feeling for the "flow" of your code (What happens when? Which value is used where?)

Chrome Debugger (+ Breakpoints)

Dive into the code in great detail and step by step.

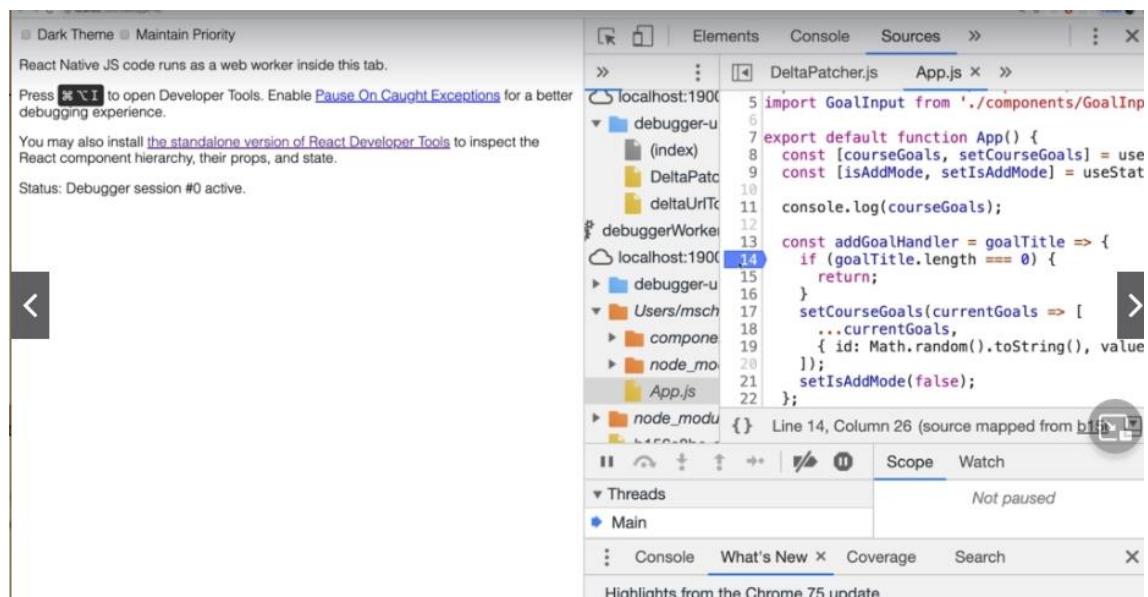
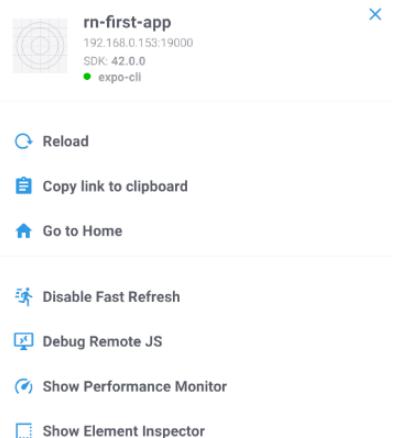
Remotely debugging

To open the developer tools, you **CMD+D** on iOS, or **CTRL+M** on Android.

Go to “**Debug JS remotely**” (make sure Connection is LAN or Local in Expo Devtools, since Tunnel is slow).

If you enable it, a web browser tab Will open where you can debug the app On a traditional JS fashion (adding Breakpoints).

You can also add “**debugger**” statements In the code itself.



Device Devtools Overlay

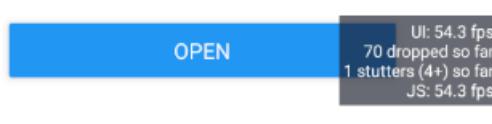
Side note: app can be fast reloaded typing “**rr**” on Android or **CMD+R** in iOS.

Now, You can also debug on the emulated device.

On Developer tools, tap “Performance Monitor”, a screen will appear with data on how your app is performing (FPS, render time)

Be aware that this is an INDICATOR!

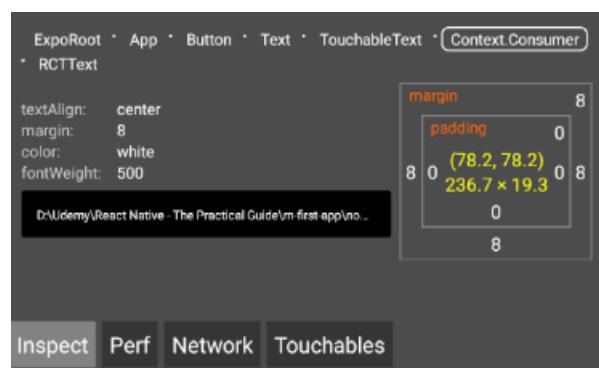
It is the performance in development Mode, in production it usually drops.



Debugging UI and using React Native Debugger

UI can be debugged by tapping “**Show Element Inspector**”, and then tapping on the element you want to inspect.

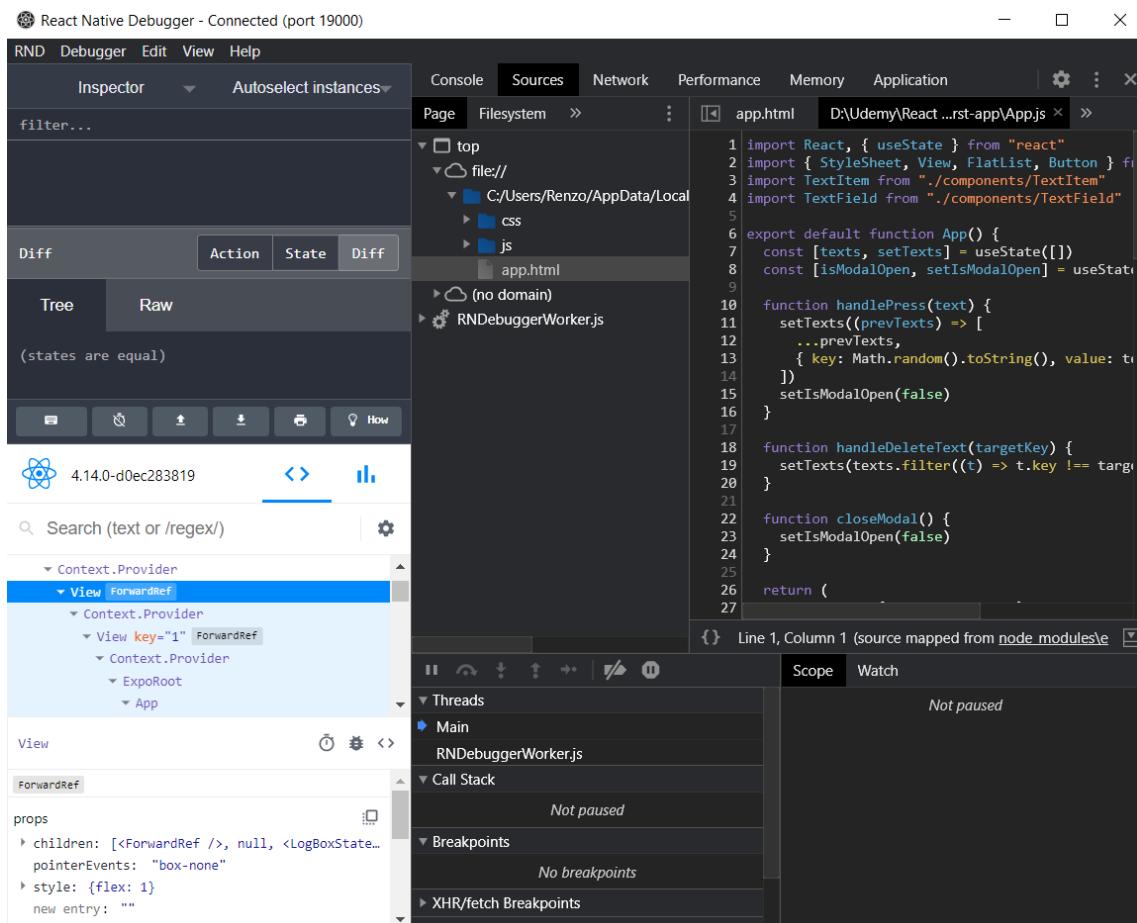
You'll see the layout, position in tree and styles.



However, a better experience can be achieved by downloading the React Native Debugger.

<https://github.com/jhen0409/react-native-debugger/blob/master/docs/getting-started.md>

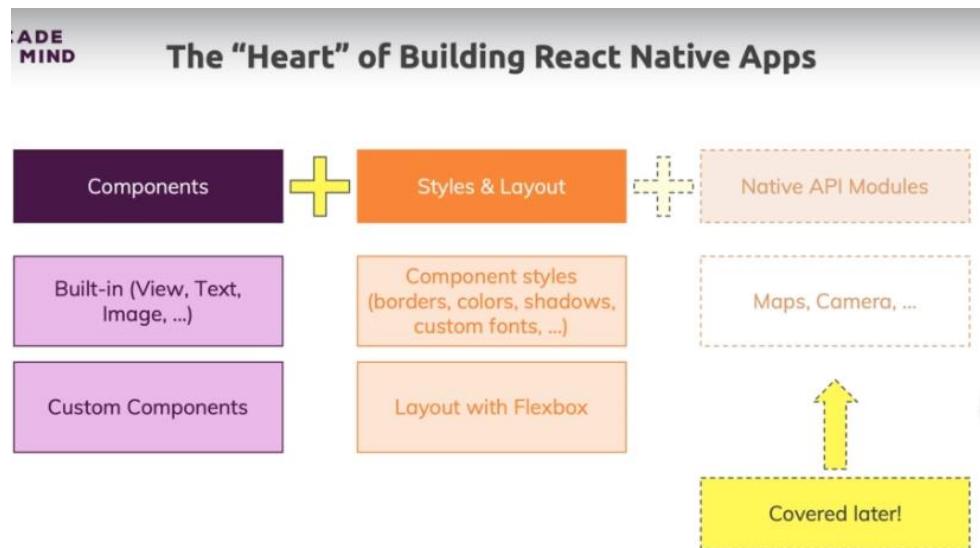
- > Install the correct binary according to your setup.
- > Enable Javascript debugging in Development Mode.
- > Open React Native Debugger, CMD+T / CTRL+T and type the port (19000)



You can right click to enable network requests, which lets you check for requests sent and responses got.

Components, stylings, layouts

Intro



Setup



Adding header and first screen components

App.js

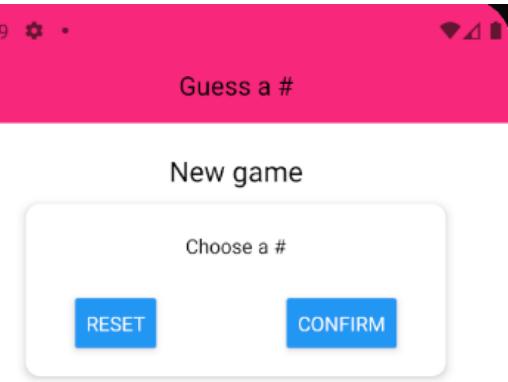
```
import React from "react"
import { StyleSheet, View } from "react-native"
import Header from "./components/Header"
import StartGame from "./screens/StartGame"
```

```

export default function App() {
  return (
    <View style={styles.container}>
      <Header title="Guess a #" />
      <StartGame />
    </View>
  )
}

const styles = StyleSheet.create({ container: { flex: 1 } })

```



components/Header.js:

```

import React from "react"
import { View, Text, StyleSheet } from "react-native"

export default function Header(props) {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>{props.title}</Text>
    </View>
  )
}

```

```

const styles = StyleSheet.create({
  container: {
    width: "100%", height: 90, paddingTop: 36,
    backgroundColor: "#f7287b", alignItems: "center",
    justifyContent: "center"
  },
  title: { color: "black", fontSize: 18 }
})

```

screens/StartGame.js:

```

import React from "react"
import { View, Text, StyleSheet, TextInput, Button } from "react-native"

export default function StartGame(props) {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>New game</Text>

```

```

        <View style={styles.inputContainer}>
            <Text>Choose a #</Text>
            <TextInput />
            <View style={styles.buttonsContainer}>
                <Button title="Reset" onPress={() => {}} />
                <Button title="Confirm" onPress={() => {}} />
            </View>
        </View>
    )
}

const styles = StyleSheet.create({
    container: { flex: 1, padding: 10, alignItems: "center" },
    title: { fontSize: 20, marginVertical: 10 },
    inputContainer: {
        width: 300, maxWidth: "80%", alignItems: "center", padding: 20,
        backgroundColor: "white", // always white bgColor.
        borderRadius: 10,
        // these shadow styles only work on iphone
        shadowColor: "black",
        shadowOffset: { width: 0, height: 2 }, // right, top
        shadowRadius: 6,
        shadowOpacity: 0.26,
        // and these ones only on android
        elevation: 5
    },
    buttonsContainer: {
        flexDirection: "row", width: "100%",
        justifyContent: "space-between", paddingHorizontal: 15
    }
})

```

Screens are views for entire pages, so they go on their separate folder, other than components/. They are similar to pages/ in web development.

React Native Styling vs CSS Styling

Styling in React Native is inspired by CSS - but it's not equivalent!

You must never forget that React Native in the end is all about translating React components (like <View> or <Text>) to native widgets (like `UIView` or `widget.view`)

These native widgets don't understand CSS. They have nothing to do with the web, HTML or anything like that!

What the React Native team does, is the following: They also provide "style translations" => CSS-inspired styling commands/ properties which also are translated to styling configurations those native widgets understand.

Hence `backgroundColor: 'black'` works - it simply targets the platform-specific configurations for the native widget that will result in a black background to be drawn. Even if these native instructions look nothing like CSS. React Native does the heavy lifting behind the scenes.

That's why many but absolutely not all CSS properties are supported in React Native. That's also why styling is done via JavaScript and not CSS. In addition, not all React Native components support all style properties.

<Text> doesn't support flexbox-related properties for example

Extracting Card component

Card.js

```
import React from "react"
import { View, StyleSheet } from "react-native"

export default function Card({ children, style }) {
  return <View style={{ ...styles.container, ...style }}>
    {children}
  </View>
}

const styles = StyleSheet.create({
  container: {
    padding: 20, backgroundColor: "white", // always white bgColor.
    borderRadius: 10,
    // these shadow styles only work on iphone
    shadowColor: "black",
    shadowOffset: { width: 0, height: 2 }, // right, top
    shadowRadius: 6,
```

```

        shadowOpacity: 0.26,
        // and these ones only on android
        elevation: 5
    }
})
}

StartGame.js
import React from "react"
import { View, Text, StyleSheet, TextInput, Button } from "react-native"
import Card from "../components/Card"

export default function StartGame(props) {
    return (
        <View style={styles.container}>
            <Text style={styles.title}>New game</Text>
            <Card style={styles.card}>
                <Text>Choose a #</Text>
                <TextInput />
                <View style={styles.buttonsContainer}>
                    <Button title="Reset" onPress={() => {}} />
                    <Button title="Confirm" onPress={() => {}} />
                </View>
            </Card>
        </View>
    )
}

const styles = StyleSheet.create({
    container: { flex: 1, padding: 10, alignItems: "center" },
    title: { fontSize: 20, marginVertical: 10 },
    card: { width: 300, maxWidth: "80%", alignItems: "center" },
    buttonsContainer: {
        flexDirection: "row", width: "100%",
        justifyContent: "space-between", paddingHorizontal: 15
    }
})

```

Adding TextInput and Button functionality

Input.js

```
import React from "react"
import { TextInput, StyleSheet } from "react-native"

export default function Input({ style, ...otherProps }) {
  return <TextInput
    {...otherProps} style={{ ...styles.container, ...style }}
  />
}

const styles = StyleSheet.create({
  container: {
    height: 30, borderRightColor: "gray",
    borderBottomWidth: 1, marginVertical: 10
  }
})
```

StartGame.js

```
import React, { useState } from "react"
import {
  View, Text, StyleSheet, Button, TouchableWithoutFeedback, Keyboard,
  Alert
} from "react-native"
import Card from "../components/Card"
import Input from "../components/Input"
import colors from "../constants/colors"

export default function StartGame(props) {
  const [num, setNum] = useState("")
  const [confirm, setConfirm] = useState(false)
  const [selectedNum, setSelectedNum] = useState()

  const handleChangeText = (text) => setNum(
    text.replace(/[^0-9]/g, "")
  )
  const handleResetInput = () => setNum("")

  const handleConfirm = () => {
    if (isNaN(chosenNum) || chosenNum <= 0 || chosenNum > 99) {
```

```
        return Alert.alert(
            "Invalid #",
            "It has to be between 1 and 99 inclusive",
            [
                {
                    text: "Back",
                    style: "destructive",
                    onPress: handleResetInput
                }
            ]
        )}}
    }

let confirmedOutput

if (confirm) confirmedOutput = <Text>Chosen #: {selectedNum}</Text>

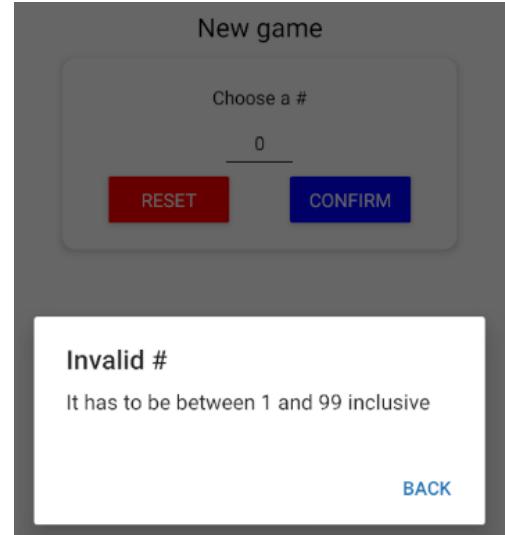
return (
    <TouchableWithoutFeedback onPress={Keyboard.dismiss}>
        <View style={styles.container}>
            <Text style={styles.title}>New game</Text>
            <Card style={styles.card}>
                <Text>Choose a #</Text>
                <Input
                    blurOnSubmit    autoCapitalize="none"
                    keyboardType="numeric"    maxLength={2}
                    value={num}    onChangeText={handleChangeText}
                    style={styles.input}
                />
                <View style={styles.buttonsContainer}>
                    <View style={styles.button}>
                        <Button
                            title="Reset" color={colors.DANGER}
                            onPress={handleResetInput}
                        />
                </View>
                <View style={styles.button}>
                    <Button
                        title="Confirm" color={colors.PRIMARY}
                        onPress={handleConfirm}
                    />
                </View>
            </Card>
        </View>
    )
}
```

```

        </View>
    </Card>
    {confirmedOutput}
</View>
</TouchableWithoutFeedback>
)
}

const styles = StyleSheet.create({
    container: { flex: 1, padding: 10, alignItems: "center" },
    title: { fontSize: 20, marginVertical: 10 },
    card: { width: 300, maxWidth: "80%", alignItems: "center" },
    input: { width: 50, textAlign: "center" },
    buttonsContainer: {
        flexDirection: "row", width: "100%",
        justifyContent: "space-between", paddingHorizontal: 15
    },
    button: { width: "40%" }
})

```



Finishing confirmation box

NumberContainer.js

```

import React from "react"
import { StyleSheet, View, Text } from "react-native"
import colors from "../constants/colors"

export default function NumberContainer({ children }) {
    return (

```

```

        <View style={styles.container}>
            <Text style={styles.text}>{children}</Text>
        </View>
    )
}

const styles = StyleSheet.create({
    container: {
        borderWidth: 2, borderColor: colors.SECONDARY,
        padding: 10, borderRadius: 10, marginVertical: 10,
        alignItems: "center", justifyContent: "center"
    },
    text: { color: colors.SECONDARY, fontSize: 22 }
})

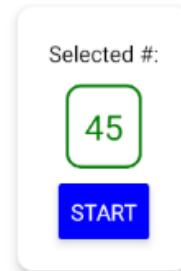
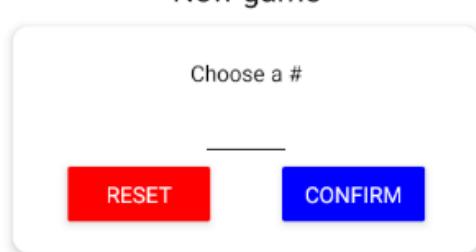
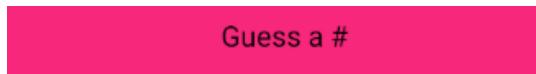
```

StartGame.js

```

...
if (confirm)
    confirmedOutput = (
        <Card style={styles.summaryCard}>
            <Text>Selected #:</Text>
            <NumberContainer>
                {selectedNum}
            </NumberContainer>
            <Button title="Start"
                color={colors.PRIMARY} onPress
            />
        </Card>
    )
...
const styles = StyleSheet.create({
    ...
    summaryCard: { marginTop: 20, alignItems: "center" }
})

```



Adding GameScreen and game logic

App.js

```

import React, { useState } from "react"
import { StyleSheet, View } from "react-native"
import Header from "./components/Header"

```

```
import GameScreen from "./screens/Game"
import GameOverScreen from "./screens/GameOver"
import StartGameScreen from "./screens/StartGame"

export default function App() {
  const [userNum, setUserNum] = useState(0)
  const [guessCounts, setGuessCounts] = useState(0)

  const handleRestart = () => {
    setGuessCounts(0)
    setUserNum(0)
  }

  const handleStartGame = (selectedNum) => setUserNum(selectedNum)

  const handleGameOver = (numGuesses) => setGuessCounts(numGuesses)

  let content = <StartGameScreen onStartGame={handleStartGame} />

  if (userNum && guessCounts <= 0) {
    content = <GameScreen
      userChoice={userNum} onGameOver={handleGameOver} />
  } else if (guessCounts > 0) {
    content = (
      <GameOverScreen
        {...{ userNum, guessCounts }} onRestart={handleRestart} />
    )
  }
}

return (
  <View style={styles.container}>
    <Header title="Guess a #" />
    {content}
  </View>
)
}

const styles = StyleSheet.create({ container: { flex: 1 } })
```

GameOverScreen.js

```
import React from "react"
import { View, Text, StyleSheet, Button } from "react-native"

export default function GameOverScreen({ guessCounts, userNum, onRestart }) {
  return (
    <View style={styles.container}>
      <Text>Game over!</Text>
      <Text>PC Guesses: {guessCounts}</Text>
      <Text>Number: {userNum}</Text>
      <Button title="new game" onPress={onRestart} />
    </View>
  )
}

const styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" }
})
```

GameScreen.js

```
import React, { useState, useRef, useEffect } from "react"
import { View, Text, StyleSheet, Button, Alert } from "react-native"
import Card from "../components/Card"
import NumberContainer from "../components/NumberContainer"

export default function GameScreen({ userChoice, onGameOver }) {
  const [curGuess, setCurGuess] = useState(rng(1, 100, userChoice))
  const [guesses, setGuesses] = useState(0)
  const curLow = useRef(1)
  const curHi = useRef(100)

  useEffect(() => {
    if (curGuess === userChoice) onGameOver(guesses)
  }, [curGuess, userChoice, onGameOver])

  const handleNextGuess = (direction) => {
    if (
      (direction === "lower" && curGuess < userChoice) ||
      (direction === "greater" && curGuess > userChoice)
    ) {
```

```

        return Alert.alert("No cheating", "You know that's not true.", [
          { text: "Sorry :c", style: "cancel" }
        ])
      }

      if (direction === "lower") curHi.current = curGuess
      else curLow.current = curGuess

      const newGuess = rng(curLow.current, curHi.current, curGuess)

      setCurGuess(newGuess)
      setGuesses((prevGuesses) => prevGuesses + 1)
    }

    return (
      <View style={styles.container}>
        <Text>PC's guess</Text>
        <NumberContainer>{curGuess}</NumberContainer>
        <Card style={styles.buttonsContainer}>
          <Button
            title="lower"
            onPress={handleNextGuess.bind(this, "lower")} />
          <Button
            title="greater"
            onPress={handleNextGuess.bind(this, "greater")}
          />
        </Card>
      </View>
    )
  }

  const styles = StyleSheet.create({
    container: { flex: 1, padding: 10, alignItems: "center" },
    buttonsContainer: {
      flexDirection: "row", justifyContent: "space-evenly",
      marginTop: 20, width: 300, maxWidth: "80%"
    }
  })
}

```

```

function rng(min, max, exclude) {
  min = Math.ceil(min) // include 1
  max = Math.floor(max) // exclude 100
  const rndNum = Math.floor(Math.random() * (max - min)) + min

  if (rndNum === exclude) return rng(min, max, exclude)
  else return rndNum
}

```

Guess a #

New game

Choose a #

—

RESET **CONFIRM**

Guess a #

PC's guess

88

LOWER **GREATERT**

Selected #: 50

START

Game over!
PC Guesses: 6
Number: 50

NEW GAME

Adding custom fonts

1. Extract and add two fonts in **.ttf** format to `assets/fonts/`
2. To import, `expo-font` should be available by default, but if not:
`$ expo install expo-font`

App.js

```

import React, { useEffect, useState } from "react"
import { ActivityIndicator, StyleSheet, View } from "react-native"
import * as Font from "expo-font"

...

export default function App() {
  const [userNum, setUserNum] = useState(0)
  const [guessRounds, setGuessRounds] = useState(0)
  const [isLoaded, setIsLoaded] = useState(false)

  // useFont hook from "expo-font" is the default today!
  useEffect(() => {

```

```

fetchFonts()
  .then(() => setIsLoaded(true))
  .catch(() => setIsLoaded(true))
}, [])

if (!isLoading) {
  return (
    <View style={...styles.container, ...styles.loadingContainer}>
      >
        <ActivityIndicator size="large" color={colors.SECONDARY} />
      </View>
    )
}

const handleRestart = () => { setGuessRounds(0); setUserNum(0) }

...
return (
  <View style={styles.container}>
    <Header title="Guess a #" /> {content} </View>
)
}

const styles = StyleSheet.create({
  container: { flex: 1 },
  loadingContainer: { justifyContent: "center", alignItems: "center" }
})

function fetchFonts() {
  return Font.loadAsync({
    "open-sans": require("./assets/fonts/OpenSans-Regular.ttf"),
    "open-sans-bold": require("./assets/fonts/OpenSans-Bold.ttf")
  })
}

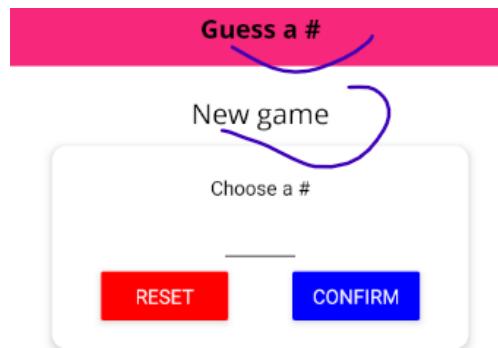
```

Header.js

```

...
export default function Header(props) {
  return (
    <View style={styles.container}>

```



```

        <Text style={styles.title}>{props.title}</Text>
    </View>
)
}

const styles = StyleSheet.create({
...
title: { color: "black", fontSize: 18, fontFamily: "open-sans-bold" }
})

```

StartGame.js

```

...
export default function StartGameScreen({ onStartGame }) {
...
const styles = StyleSheet.create({
  container: { flex: 1, padding: 10, alignItems: "center" },
  title: { fontSize: 20, marginVertical: 10, fontFamily: "open-sans" },
},
...
})

```

Installing expo-font

Depending on the version of Expo you're using, you very likely need to install the expo-font package.

You can do this in two different ways and it's important to understand the difference:

- 1) npm install --save expo-font
- 2) expo install expo-font
- 2) is recommended - but what is the difference?

npm install installs a packages a dependency into the project - we use this command for most packages which we do install.

Some packages (typically all expo-* packages) can break the app if you install the wrong version though - because they closely work together with Expo itself.

To get the right package version for the specific version of Expo your app relies on, expo install is the right "tool". It also just executes npm install behind the scenes but it picks a specific (i.e. the correct) version of the package to be installed.

Hence for all expo-* packages, npm install can be used but expo install is the preferred command to avoid errors. Of course you could always try npm install first and only run expo install if you thereafter do face any errors.

Synthetic Style Cascade custom component

If you want to use the same styles for one component (example with fontFamily used before), you have two alternatives:

1. Create a custom component and re-use it everywhere.

NormalText.js

```
import React from "react"
import { Text, StyleSheet } from "react-native"

export default function NormalText(
  { children, style, ...otherProps }
) {
  return (
    <Text style={{ ...styles.container, ...style }} {...otherProps}>
      {children}
    </Text>
  )
}

const styles = StyleSheet.create({
  container: { fontFamily: "open-sans" }
})
```

GameOverScreen.js

```
import React from "react"
import { View, StyleSheet, Button } from "react-native"
import NormalText from "../components/NormalText"

export default function GameOverScreen({ guessRounds, userNum, onRestart }) {
  return (
```

```

<View style={styles.container}>
  <NormalText>Game over!</NormalText>
  <NormalText>PC Guesses: {guessCounts}</NormalText>
  <NormalText>Number: {userNum}</NormalText>
  <Button title="new game" onPress={onRestart} />
</View>
)
}
}

const styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" }
})

```

Game over!
PC Guesses: 2
Number: 50
NEW GAME

2. Create a constants file with a default exported StyleSheet to be imported anywhere as styles.

constants/defaultStyles.js

```

import { StyleSheet } from "react-native"

const defaultStyles = StyleSheet.create({
  NORMAL_TEXT: { fontFamily: "open-sans", fontSize: 18 },
  BOLD_TEXT: { fontFamily: "open-sans-bold", fontSize: 20 }
})

```

Guess a #

export default defaultStyles

PC's guess
14

LOWER **GREATERT**

Game.js

```

...
export default function GameScreen({ userChoice, onGameOver }) {
  ...
  return (
    <View style={styles.container}>
      <Text style={defaultStyles.BOLD_TEXT}>PC's guess</Text>
      <Text style={defaultStyles.NORMAL_TEXT}>{curGuess}</Text>
      <Card style={styles.buttonsContainer}>
        <Button
          title="lower" onPress={handleNextGuess.bind(this, "lower")}>
        />
        <Button
          title="greater"
        >
      </Card>
    </View>
  )
}

```

```

        onPress={handleNextGuess.bind(this, "greater")}

    />
</Card>
</View>
)
}

```

Adding local images

Two ways:

1. Locally:

GameOver.js

```

import React from "react"
import { View, StyleSheet, Button, Image } from "react-native"
import BoldText from "../components/BoldText"
import NormalText from "../components/NormalText"

export default function GameOverScreen(
  { guessRounds, userNum, onRestart }
) {
  return (
    <View style={styles.container}>
      <BoldText>Game over!</BoldText>
      <View style={styles.imageContainer}>
        <Image
          source={require("../assets/images/success.png")}
          style={styles.image}
          resizeMode="cover" // 'contain' 'center' 'repeat' 'stretch'
        />
      </View>
      <NormalText>PC Guesses: {guessRounds}</NormalText>
      <NormalText>Number: {userNum}</NormalText>
      <Button title="new game" onPress={onRestart} />
    </View>
  )
}

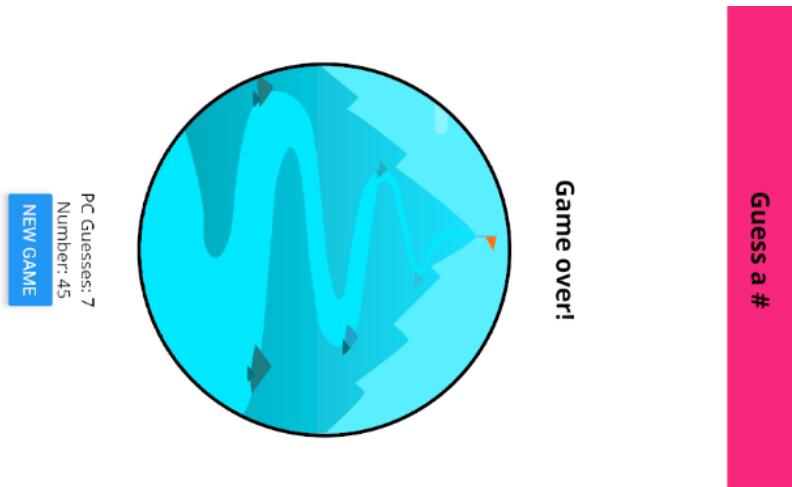
const styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" },

```

```

imageContainer: {
  width: 300, // hard coded for now. Will media query later!
  height: 300,
  borderRadius: 150, borderWidth: 3, borderColor: "black",
  overflow: "hidden", marginVertical: 30
},
image: { width: "100%", height: "100%" }
})

```



2. Remotely:

GameOver.js

```

...
export default function GameOverScreen(
  { guessRounds, userNum, onRestart }
) {
  return (
    <View style={styles.container}>
      <BoldText>Game over!</BoldText>
      <View style={styles.imageContainer}>
        <Image
          source={{
            uri: "https://explorersweb.com/wp-content/
              uploads/2021/05/Summit-Everest-MingmaG.jpg"
          }}
          fadeDuration={500} // fadein effect on img load, 300 default
          style={styles.image}
          resizeMode="cover" // 'contain' 'center' 'repeat' 'stretch'
        />
    </View>
  )
}

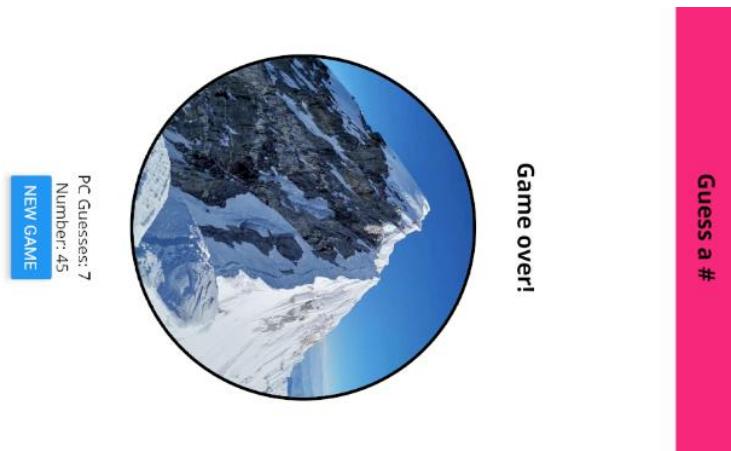
```

```

        </View>
        <NormalText>PC Guesses: {guessCounts}</NormalText>
        <NormalText>Number: {userNum}</NormalText>
        <Button title="new game" onPress={onRestart} />
    </View>
)
}

const styles = StyleSheet.create({
    container: { flex: 1, justifyContent: "center", alignItems: "center" },
    imageContainer: {
        width: 300, // hard coded for now. Will media query later!
        height: 300,
        borderRadius: 150, borderWidth: 3, borderColor: "black",
        overflow: "hidden", marginVertical: 30
    },
    // RN is able to calculate width/height of local images, but NOT
    // from remote ones! These last ones ALWAYS need height and width!
    image: { width: "100%", height: "100%" }
})

```



Closer look on <Text>

You can nest `<Text />` components!

This is used to style texts inside other ones.

Also, this way, inner Texts "inherit" styles from parents.

> One of the few instances this apply in RN!

`<View>` can also be nested inside `<Text>`, but there can be issues with that.

Text does NOT use flexbox by default, like other RN components

Text wraps into a new line if it does not fit inside the screen.

If you do not want this behavior, you can set **numberOfLines** prop, combined with **ellipsizeMode** 'truncate'

GameOver.js

```
import React from "react"
import { View, StyleSheet, Button, Image, Text } from "react-native"
import BoldText from "../components/BoldText"
import NormalText from "../components/NormalText"
import colors from "../constants/colors"

export default function GameOverScreen({ guessRounds, userNum, onRestart }) {
  return (
    <View style={styles.container}>
      <BoldText>Game over!</BoldText>
      <View style={styles.imageContainer}>
        <Image
          source={require("../assets/images/success.png")}
          style={styles.image}
          resizeMode="cover"
        />
      </View>
      <NormalText style={styles.textWrapper}>
        PC Guesses:
        <Text style={styles.highlight}>{guessRounds}</Text>.
        # was:
        <Text style={styles.highlight}>{userNum}</Text>
      </NormalText>
      <Button title="new game" onPress={onRestart} />
    </View>
  )
}

const styles = StyleSheet.create({
  ...
  textWrapper: { marginVertical: 20, textAlign: "center" },
  highlight: {
    color: colors.PRIMARY, fontFamily: "open-sans-bold",
    marginHorizontal: 30
  }
})
```

<View> vs <Text>

<Text> and <View> are probably **THE most important/most-used components** built into React Native.

<View> is your #1 component if you need to group and structure content (= provide a layout) and/ or if you want to style something as a container (e.g. the <Card> look we built in our custom <Card> component).

<View> uses **Flexbox** to organize its children - have a look at the Flexbox deep dive earlier in this course (**in module 2**) to learn more about how that works.

A <View> can hold as many child components as you need and it also works with any kind of child component - it can hold <Text> components, other <View>s (for nested containers/ layouts), <Image>s, custom components etc.

If you need scrolling, you should consider using a <ScrollView> - you could wrap your <View> with it or replace your <View> (that depends on your layout and styling). Please note, that due to its scrollable nature, Flexbox works a bit differently on a
<ScrollView>: <https://stackoverflow.com/questions/46805135/scrollview-with-flex-1-makes-it-un-scrollable>

<Text> is also super important. As its name suggests, you use it for outputting text (of any length). You can also **nest other <Text> components** into a <Text>. Actually, you can also have nested <View>s inside of a <Text> but that comes with certain caveats you should watch out for: <https://github.com/facebook/react-native/commit/a2a03bc68ba062a96a6971d3791d291f49794dfd>

Unlike <View>, <Text> does **NOT use Flexbox** for organizing its content (i.e. the text or nested components). Instead, text inside of <Text> automatically fills a line as you would expect it and wraps into a new line if the text is too long for the available <Text> width.

You can avoid wrapping by setting the `numberOfLines` prop, possibly combined with `ellipsizeMode`.

Example:

```
1 | <Text numberOfLines={1} ellipsizeMode="tail">
2 |   This text will never wrap into a new line, instead it will b
3 | </Text>
```

Also important: When adding styles to a `<Text>` (no matter if that happens via inline styles or a `StyleSheet` object), the styles will actually be shared with any nested `<Text>` components.

This **differs from the behavior** of `<View>` (or actually any other component - `<Text>` is the exception): There, any styles are **only applied to the component to which you add them**. Styles are never shared with any child component!

```
<Text numberOfLines={1} ellipsizeMode="tail">
  Text will be cut off like this if it is too long...
</Text>
```

Custom Button component and icons

There are many icon packages in `expo: Ionicons, EvilIcons, MaterialIcons, AntDesign`, and such. **Check the docs!**

MainButton.js

```
import React from "react"
import { View, Text, StyleSheet, TouchableOpacity } from "react-native"
import colors from "../constants/colors"

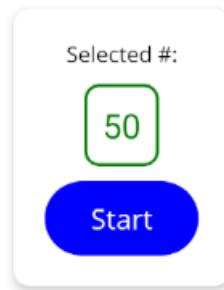
export default function MainButton({
  children, onPress, styles = {}, ...rest
}) {
  return (
    <TouchableOpacity
      style={{ ..._styles.container, ...styles.container }}
      activeOpacity={0.4} onPress={onPress} {...rest}
    >
      <View style={{ ..._styles.view, ...styles.view }}>
```

```

        <Text style={{ ..._styles.text, ...styles.text }}>
            {children}
        </Text>
    </View>
</TouchableOpacity>
)
}

const _styles = StyleSheet.create({
    container: {},
    view: {
        backgroundColor: colors.PRIMARY, paddingVertical: 12,
        paddingHorizontal: 30, borderRadius: 25
    },
    text: { color: "white", fontFamily: "open-sans", fontSize: 18 }
})

```



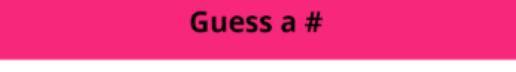
StartGameScreen.js

```

...
import MainButton from "../components/MainButton"
...

export default function StartGameScreen({ onStartGame }) {
    ...
    if (confirm)
        confirmedOutput = (
            <Card style={styles.summaryCard}>
                <NormalText>Selected #:</NormalText>
                <NumberContainer>{selectedNum}</NumberContainer>
                <MainButton onPress={() => onStartGame(selectedNum)}>
                    Start
                </MainButton>
            </Card>
        )
    ...
}

```



PC's guess

64



GameScreen.js

```

...
import { Ionicons } from "@expo/vector-icons"
import Card from "../components/Card"

```

```

export default function GameScreen({ userChoice, onGameOver }) {
  ...
  return (
    <View style={styles.container}>
      <Text style={defaultStyles.BOLD_TEXT}>PC's guess</Text>
      <Text style={defaultStyles.NORMAL_TEXT}>{curGuess}</Text>
      <Card style={styles.buttonsContainer}>
        <MainButton onPress={handleNextGuess.bind(this, "lower")}>
          <Ionicons name="md-remove" size={24} color="white" />
        </MainButton>
        <MainButton onPress={handleNextGuess.bind(this, "greater")}>
          <Ionicons name="md-add" size={24} color="white" />
        </MainButton>
      </Card>
    </View>
  )
}

const styles = StyleSheet.create({
  container: { flex: 1, padding: 10, alignItems: "center" },
  buttonsContainer: {
    flexDirection: "row", justifyContent: "space-evenly",
    marginTop: 20, width: 400, maxWidth: "90%"
  }
})

```

Exploring UI libraries

Keep in mind most RN components do require you styling lots.

You can always do so if you need or want to, but there are many UI libraries that have components ready to use in RN, like AntDesign, Ionic, Material Design, Bootstrap, and others.

<https://reactnativeelements.com/>

<https://github.com/GeekyAnts/NativeBase>

Managing past guesses as list

Note: If you wish to add width to a ScrollView, **wrap it in View!**

Note: When using ScrollView, you need to set **flexGrow** (not flex).

This ensures that all space is taken, AS WELL AS being able to exceed boundaries, which will show first and last items on screen.

GameScreen.js

```
import React, { useState, useRef, useEffect } from "react"
import { View, Text, StyleSheet, Alert, ScrollView } from "react-native"
import { Ionicons } from "@expo/vector-icons"
import Card from "../components/Card"
import MainButton from "../components/MainButton"
import NormalText from "../components/NormalText"
import defaultStyles from "../constants/default-styles"

export default function GameScreen({ userChoice, onGameOver }) {
  const [curGuess, setCurGuess] = useState(rng(1, 100, userChoice))
  const [guesses, setGuesses] = useState([curGuess])
  const curLow = useRef(1)
  const curHi = useRef(100)

  useEffect(() => {
    if (curGuess === userChoice) onGameOver(guesses.length)
  }, [curGuess, userChoice, onGameOver])

  const handleNextGuess = (direction) => {
    if (
      (direction === "lower" && curGuess < userChoice) ||
      (direction === "greater" && curGuess > userChoice)
    ) {
      return Alert.alert("No cheating", "You know that's not true.", [
        { text: "Sorry :c", style: "cancel" }
      ])
    }

    if (direction === "lower") curHi.current = curGuess
    else curLow.current = curGuess + 1 // as not to repeat keys

    const newGuess = rng(curLow.current, curHi.current, curGuess)

    setCurGuess(newGuess)
    setGuesses((prevGuesses) => [newGuess, ...prevGuesses])
  }

  const renderItem = (value, index) => (
    <View key={value} style={styles.listItem}>
```

```

        <NormalText>#{guesses.length - index}</NormalText>
        <NormalText>{value}</NormalText>
    </View>
}

return (
    <View style={styles.container}>
        <Text style={defaultStyles.BOLD_TEXT}>PC's guess</Text>
        <Text style={defaultStyles.NORMAL_TEXT}>{curGuess}</Text>
        <Card style={styles.buttonsContainer}>
            <MainButton onPress={handleNextGuess.bind(this, "lower")}>
                <Ionicons name="md-remove" size={24} color="white" />
            </MainButton>
            <MainButton onPress={handleNextGuess.bind(this, "greater")}>
                <Ionicons name="md-add" size={24} color="white" />
            </MainButton>
        </Card>
        <View style={styles.listContainer}>
            <ScrollView contentContainerStyle={styles.list}>
                {guesses.map(renderListItem)}
            </ScrollView>
        </View>
    </View>
)
}

const styles = StyleSheet.create({
    container: { flex: 1, padding: 10, alignItems: "center" },
    buttonsContainer: {
        flexDirection: "row", justifyContent: "space-evenly",
        marginTop: 20, width: 400, maxWidth: "90%"
    },
    listContainer: {
        width: "80%",
        flex: 1 // stretch to the whole available space
    },
    list: {
        flexGrow: 1, // keeps styles of ScrollView. Exceeds boundaries.
        alignItems: "center",
        justifyContent: "flex-end"
    }
})

```

```
        } ,  
  
    listItem: {  
        borderColor: "#ccc", borderWidth: 1, padding: 15,  
        marginVertical: 10, backgroundColor: "white",  
        flexDirection: "row",  
        justifyContent: "space-evenly",  
        width: "60%"  
    }  
})
```

```
function rng(min, max, exclude) {  
    min = Math.ceil(min) // include 1  
    max = Math.floor(max) // exclude 100  
    const rndNum = Math.floor(Math.random() * (max - min)) + min  
  
    return rndNum === exclude ? rng(min, max, exclude) : rndNum  
}
```

Guess a #

PC's guess
57

#8	57
#7	60
#6	69
#5	49
#4	76
#3	78
#2	48
#1	39

Using FlatList instead of ScrollView

ScrollView always keeps its components rendered, making it an efficient component only when there are not too many items for it to show.

If we need to display plenty of items, we use a FlatList.

GameScreen.js

```
import React, { useState, useRef, useEffect } from "react"
import { View, Text, StyleSheet, Alert, FlatList } from "react-native"
...
export default function GameScreen({ userChoice, onGameOver }) {
  const [curGuess, setCurGuess] = useState(rng(1, 100, userChoice))
  const [guesses, setGuesses] = useState([{ guess: curGuess }])
  const curLow = useRef(1)
  const curHi = useRef(100)

  useEffect(() => {
    if (curGuess === userChoice) onGameOver(guesses.length)
  }, [curGuess, userChoice, onGameOver])

  const handleNextGuess = (direction) => {
    if (
      (direction === "lower" && curGuess < userChoice) ||
      (direction === "greater" && curGuess > userChoice)
    ) {
      return Alert.alert("No cheating", "You know that's not true.",
        [{ text: "Sorry :c", style: "cancel" }])
    }

    if (direction === "lower") curHi.current = curGuess
    else curLow.current = curGuess + 1 // as not to repeat keys

    const newGuess = rng(curLow.current, curHi.current, curGuess)

    setCurGuess(newGuess)
    setGuesses((prevGuesses) => [{ guess: newGuess }, ...prevGuesses])
  }

  const renderListItem = ({ item, index }) => (
    <View style={styles.listItem}>
```

```

        <NormalText>#{guesses.length - index}</NormalText>
        <NormalText>{item.guess}</NormalText>
    </View>
)

return (
<View style={styles.container}>
    <Text style={defaultStyles.BOLD_TEXT}>PC's guess</Text>
    <Text style={defaultStyles.NORMAL_TEXT}>{curGuess}</Text>
    <Card style={styles.buttonsContainer}>
        <MainButton onPress={handleNextGuess.bind(this, "lower")}>
            <Ionicons name="md-remove" size={24} color="white" />
        </MainButton>
        <MainButton onPress={handleNextGuess.bind(this, "greater")}>
            <Ionicons name="md-add" size={24} color="white" />
        </MainButton>
    </Card>
    <View style={styles.listContainer}>
        <FlatList
            contentContainerStyle={styles.list}
            data={guesses}
            keyExtractor={({ guess }) => guess.toString()}
            renderItem={renderListItem}
        />
    </View>
</View>
)
}

const styles = StyleSheet.create({
    container: { flex: 1, padding: 10, alignItems: "center" },
    buttonsContainer: {
        flexDirection: "row", justifyContent: "space-evenly",
        marginTop: 20, width: 400, maxWidth: "90%"
    },
    listContainer: {
        width: "60%", // FlatList container is the real width for items
        flex: 1
    },
})

```

```

list: {
  flexGrow: 1,
  // nothing to align to center, listItems are 100% width now
  justifyContent: "flex-end"
},
listItem: {
  borderColor: "#ccc", borderWidth: 1, padding: 15,
  marginVertical: 10, backgroundColor: "white",
  flexDirection: "row", justifyContent: "space-evenly",
  width: "100%" // 100% of 60%, now in listContainer
}
})

```

Wrap up

ADE
MIND

Core Components & Styling - Summary

<View> Container & Layout Component	<Text> Output (Nested) Text, Control Text Style	<TextInput> Receive User Input (via Soft Keyboard)
<Button> Platform-adaptive Button	<FlatList> Performance-optimized Scrollable List	<ScrollView> Non-optimized Scrollable View/ Container
<Image> Output Local or Network Images	StyleSheet Validated, Potentially Optimized Style Config	

Responsive and Adaptive interfaces and Apps

Intro

How to Detect the Device Dimensions & Orientation

Adjusting Layouts based on Size & Orientation

How to Detect the Device Platform (iOS or Android)

Adjusting Code / UI based on the Platform

Working with more flexible style rules

First thing first, it is convenient to create some device simulator images with different sizes to be tested.

Now, you can set width as %, then set min and max widths to make sure the item is never too large or small, regardless of the device.

Like this:

StartGameScreen.js

```
...
const styles = StyleSheet.create({
...
  card: {
    width: "80%", minWidth: 300, maxWidth: "95%", alignItems: "center"
  },
...
})
```



Dimensions API

Sometimes, we cannot get away with just using percentages. We might need to know how many pixels are available, and here is when Dimensions API comes in handy.

Mind that percentages calculate over parent containers, **while Dimensions API can be executed on window and on screen**.

`Dimensions.get()` returns an object with values for current `fontScale`, `scale`, `width` and `height`. Both for screen and window as arguments.

StartGameScreen.js

```
...
const styles = StyleSheet.create({
...
  button: { width: Dimensions.get("window").width / 3 },
...
})
```



We can also include if checks anywhere in the javascript file.

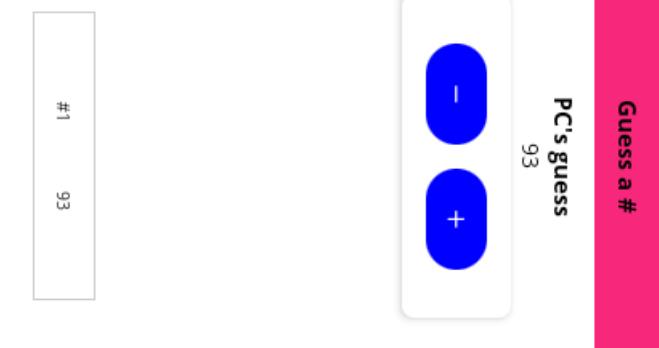
GameScreen.js

```
...
const styles = StyleSheet.create({
...
  listContainer: {
```

```

width: Dimensions.get("window").width > 500 ? "60%" : "80%",
flex: 1 // stretch to the whole available space
},
...
})

```



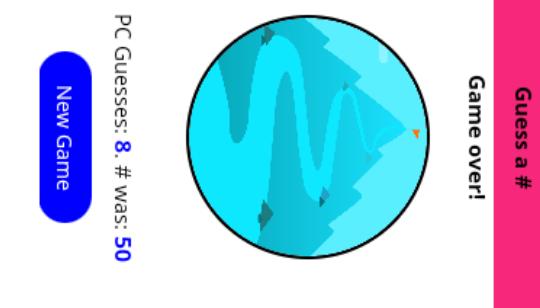
And of course, we can dynamically calculate values.

GameOver.js

```

...
export default function GameOverScreen({ guessCounts, userNum,
onRestart }) {
return (
/* if device is extra small, make sure it is scrollable */
<ScrollViewScrollView

```



```

const styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" },
  imageContainer: {
    width: Dimensions.get("window").width * 0.7,
    height: Dimensions.get("window").width * 0.7,
    borderRadius: (Dimensions.get("window").width * 0.7) / 2,
  }
})

```

```

borderWidth: 3, borderColor: "black", overflow: "hidden",
marginVertical: Dimensions.get("window").height / 20 // 5%
},
image: { width: "100%", height: "100%" },
textWrapper: {
  marginVertical: Dimensions.get("window").height / 40, // 2.5%
  textAlign: "center",
  fontSize: Dimensions.get("window").height < 400 ? 16 : 20
},
highlight: {
  color: colors.PRIMARY, fontFamily: "open-sans-bold",
  marginHorizontal: 30
}
}
)

```

Problems with device orientations

Some apps have to be locked to a certain device orientation, being portrait or landscape. But most times we would want it to be used in both modes.

This is configurable in app.json

app.json

```
{
  "expo": {
    "name": "guess-a-number",
    "slug": "guess-a-number",
    "version": "1.0.0",
    "orientation": "default",
    "icon": "./assets/icon.png",
    "splash": {
      "image": "./assets/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#ffffff"
    },
    "updates": {
      "fallbackToCacheTimeout": 0
    },
    "assetBundlePatterns": ["**/*"],
    "ios": {
      "supportsTablet": true
    }
  }
}
```

```

},
"android": {
  "adaptiveIcon": {
    "foregroundImage": "./assets/adaptive-icon.png",
    "backgroundColor": "#FFFFFF"
  }
},
"web": {
  "favicon": "./assets/favicon.png"
}
}
}
}

```

KeyboardAvoidingView

This breaks all stylings, so lets start adjusting them, and we will start with KeyboardAvoidingView, a wrapper View component which makes sure the screen adjusts itself when keyboard opens.

Note it must be used inside ScrollView if we are applying the latter.

Behaviors can be:

'**position**', which adjusts an offset of certain amount of pixels, controlled by keyboardVerticalOffset

<KeyboardAvoidingView behavior="position" keyboardVerticalOffset={30}>



'**padding**', adding padding styling to the bottom of the screen.

<KeyboardAvoidingView behavior="padding">

'**height**', changing the overall height of the screen.

Typically, on iOS '**position**' works best, and on android, '**padding**'.

Listening to orientation changes

`Dimensions.get()` calculates once when app starts.

This means that if the device rotates, values will not re-calculate, and styles will break.

To recalculate Dimensions, we have to pluck it off StyleSheet variable, and add checks in the component itself, with `Dimensions.addEventListener`. Like this:

StartGameScreen.js

```
...
export default function StartGameScreen({ onStartGame }) {
  const [num, setNum] = useState("")
  const [confirm, setConfirm] = useState(false)
  const [selectedNum, setSelectedNum] = useState()
  const [buttonWidth, setButtonWidth] = useState(
    Dimensions.get("window").width / 3.5
  )

  useEffect(() => {
    // create a function that returns the width of the window / 3.5
    const updateLayout = () =>
      setButtonWidth(Dimensions.get("window").width / 3.5)

    // add a listener to call for it
    Dimensions.addEventListener("change", updateLayout)

    // clear the event listener on unmount
    return () => Dimensions.removeEventListener("change",
      updateLayout)
  }, [])

  ...
  return (
    <ScrollView>
      <KeyboardAvoidingView
        behavior="position" keyboardVerticalOffset={30}>
        <TouchableWithoutFeedback onPress={Keyboard.dismiss}>
          <View style={styles.container}>
            ...
            <View style={styles.buttonsContainer}>
              <View style={{ width: buttonWidth }}>
                <Button
```

```

        title="Reset"
        color={colors.DANGER}
        onPress={handleResetInput}
      />
    </View>
<View style={{ width: buttonWidth }}>
  <Button
    title="Confirm"
    color={colors.PRIMARY}
    onPress={handleConfirm}
  />
</View>
</View>
...
)
}

```



Rendering different layouts

GameScreen.js

```

...
export default function GameScreen(( userChoice, onGameOver )) {
...
  const [currDeviceWidth, setCurrDeviceWidth] = useState(
    Dimensions.get("window").width
  )
  const [currDeviceHeight, setCurrDeviceHeight] = useState(
    Dimensions.get("window").height
  )
...
  useEffect(() => {
    const updateLayout = () => {
      setCurrDeviceHeight(Dimensions.get("window").height)
      setCurrDeviceWidth(Dimensions.get("window").width)
    }
  })
}

```

```

        }

    Dimensions.addEventListener("change", updateLayout)

    return () => Dimensions.removeEventListener("change",
        updateLayout)
}, [])
...

if (currDeviceHeight < 500) {
    return (
        <View style={styles.container}>
            <Text style={defaultStyles.BOLD_TEXT}>PC's guess</Text>
            <Card style={styles.buttonsContainer}>
                <View style={styles.control}>
                    <MainButton onPress={handleNextGuess.bind(this, "lower")}>
                        <Ionicons name="md-remove" size={24} color="white" />
                    </MainButton>
                <Card>
                    <Text style={defaultStyles.NORMAL_TEXT}>{curGuess}</Text>
                </Card>
                    <MainButton onPress={handleNextGuess.bind(this, "greater")}>
                        <Ionicons name="md-add" size={24} color="white" />
                    </MainButton>
                </View>
            </Card>
            <View style={styles.listContainer}>
                <FlatList
                    contentContainerStyle={styles.list}
                    data={guesses}
                    keyExtractor={({ guess }) => guess.toString()}
                    renderItem={renderListItem}
                />
            </View>
        </View>
    )
}

return (
    <View style={styles.container}>
        <Text style={defaultStyles.BOLD_TEXT}>PC's guess</Text>

```

```

<Card style={styles.guessWrapper}>
  <Text style={defaultStyles.NORMAL_TEXT}>{curGuess}</Text>
</Card>
<Card style={styles.buttonsContainer}>
  <MainButton onPress={handleNextGuess.bind(this, "lower")}>
    <Ionicons name="md-remove" size={24} color="white" />
  </MainButton>
  <MainButton onPress={handleNextGuess.bind(this, "greater")}>
    <Ionicons name="md-add" size={24} color="white" />
  </MainButton>
</Card>
<View style={styles.listContainer}>
  <FlatList
    contentContainerStyle={styles.list}
    data={guesses}
    keyExtractor={({ guess }) => guess.toString()}
    renderItem={renderListItem}
  />
  <View>
    <Text>PC's guess</Text>
    <Text>19</Text>
    <View>
      <Text>-</Text>
      <Text>+</Text>
    </View>
  </View>
</View>
)
}

const styles = StyleSheet.create({
  container: { flex: 1, padding: 10, alignItems: "center" },
  control: {
    flexDirection: "row", justifyContent: "space-around",
    alignItems: "center", width: "80%"
  },
  guessWrapper: { marginVertical: 10 },
  ...
})

```

Expo's ScreenOrientation API

Built-in Dimensions API does not give you the device orientation, but expo's ScreenOrientation does.

```
$ expo install expo-screen-orientation
```

This gives you the screen orientation and screen lock capabilities.

You have diverse functionalities, as `getOrientationAsync`, `lockAsync`, `addOrientationListener`, `lockPlatformAsync`, and many more. All async because they all for the underlying native API.

You also get some constants, like `OrientationLock.PORTRAIT` and `LANDSCAPE` strings.

A simple example to lock orientation to portrait mode would be:

```
ScreenOrientation.lockAsync(  
  ScreenOrientation.OrientationLock.PORTRAIT  
)
```

Platform API

Some APIs and components behave differently in Android and in iOS, so inevitably, we need a way to detect on which platform are we running. Platform API does just that.

It offers back values as `OS`, `Version`, `constants`, `isTV`, `isTVOS`, `isPad`, `isTesting`; and a function to `select` a platform.

Header.js

```
import React from "react"  
import { View, StyleSheet, Platform } from "react-native"  
import BoldText from "./BoldText"  
import colors from "../constants/colors"  
  
export default function Header(props) {  
  return (  
    <View style={styles.container}>  
      <BoldText style={styles.title}>{props.title}</BoldText>  
    </View>  
  )  
}  
  
const styles = StyleSheet.create({  
  container: {  
    width: "100%", height: 90, paddingTop: 36,  
    alignItems: "center", justifyContent: "center",  
    ...Platform.select({  
      ios: {  
        backgroundColor: "white",  
        borderBottomColor: "gray",  
        borderBottomWidth: 1  
      }  
    })  
  }  
})
```

```

        },
        android: { backgroundColor: colors.SECONDARY }
    })
},
title: { color: Platform.OS === "ios" ? colors.PRIMARY : "white" }
))

```

MainButton.js

```

import React from "react"
import { View, Text, StyleSheet, TouchableOpacity,
    TouchableNativeFeedback, Platform
} from "react-native"
import colors from "../constants/colors"

export default function MainButton({
    children, onPress, styles = {}, ...rest
}) {
    let ButtonComponent = TouchableOpacity

    // 'ripple' effect was introduced in android v.21
    if (Platform.OS === "android" && Platform.Version >= 21) {
        ButtonComponent = TouchableNativeFeedback
    }

    return (
        <View style={[_styles.container, styles.container]}>
            <ButtonComponent
                activeOpacity={0.4} style={styles.button}
                onPress={onPress} {...rest}
            >
                <View style={[_styles.view, styles.view]}>
                    <Text style={[_styles.text, styles.text]}>{children}</Text>
                </View>
            </ButtonComponent>
        </View>
    )
}

const _styles = StyleSheet.create({

```

```

// make 'ripple' effect do not exceed boundaries of visible button
container: { borderRadius: 25, overflow: "hidden" },
view: {
  backgroundColor: colors.PRIMARY, paddingVertical: 12,
  paddingHorizontal: 30, borderRadius: 25
},
text: { color: "white", fontFamily: "open-sans", fontSize: 18 }
})

```

Or, if the file ends up clogged with platform checks, **you can create two independent files width .ios.js and .android.js extensions.**

React Native will compile the standard file to the correct platform, under the hood.

MainButton.android.js

```

...
export default function MainButton({
  children, onPress, styles = {}, ...rest
}) {
  let ButtonComponent = TouchableOpacity

  // 'ripple' effect was introduced in android v.21
  if (Platform.Version >= 21) { // <- no more Platform.OS check
    ButtonComponent = TouchableNativeFeedback
  }

  return (
    <View style={[_styles.container, styles.container]}>
      <ButtonComponent
        activeOpacity={0.4} style={styles.button}
        onPress={onPress} {...rest}>
      </ButtonComponent>
    </View>
  )
}

```

```

const _styles = StyleSheet.create({
  // make 'ripple' effect do not exceed boundaries of visible button
  container: { borderRadius: 25, overflow: "hidden" },
  view: {
    backgroundColor: colors.PRIMARY,
    paddingVertical: 12,
    paddingHorizontal: 30,
    borderRadius: 25
  },
  text: { color: "white", fontFamily: "open-sans", fontSize: 18 }
})

```

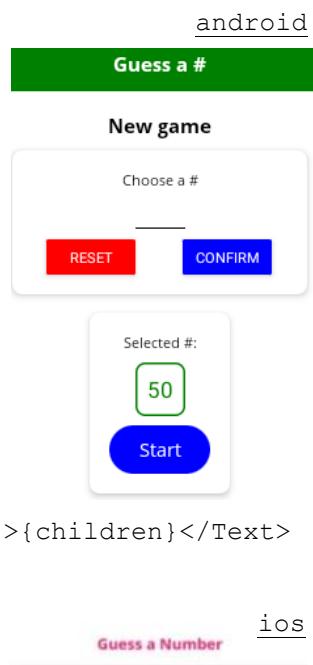
MainButton.ios.js

```

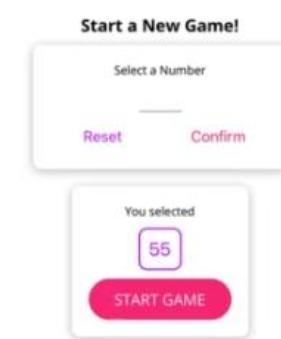
...
export default function MainButton({
  children, onPress, styles = {}, ...rest
}) {
  return (
    <TouchableOpacity
      activeOpacity={0.4} style={styles.button}
      onPress={onPress} {...rest}
    >
      <View style={[_styles.view, styles.view]}>
        <Text style={[_styles.text, styles.text]}>{children}</Text>
      </View>
    </TouchableOpacity>
  )
}

const _styles = StyleSheet.create({
  view: {
    backgroundColor: colors.PRIMARY,
    paddingVertical: 12,
    paddingHorizontal: 30, borderRadius: 25
  },
  text: { color: "white", fontFamily: "open-sans", fontSize: 18 }
})

```

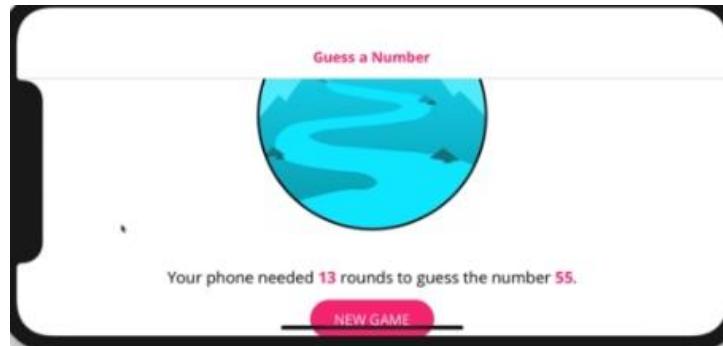


ios



SafeAreaView

Many times, general features of the phone can overlap layout shown on screen for our app, like the notch (thin line) or camera section in iOS devices.



To make sure content is adjusted to fit on the screen, there is a component built in React Native: **SafeAreaView**.

Note: You might not need to manage it all the time, as many libraries do it under the hood, like React Navigation.

App.js

```
import React, { useEffect, useState } from "react"
import { ActivityIndicator, SafeAreaView, StyleSheet } from "react-native"
import * as Font from "expo-font"

import Header from "./components/Header"
import GameScreen from "./screens/Game"
import GameOverScreen from "./screens/GameOver"
import StartGameScreen from "./screens/StartGame"
import colors from "./constants/colors"

export default function App() {
  const [userNum, setUserNum] = useState(0)
  const [guessRounds, setGuessRounds] = useState(0)
  const [isLoaded, setIsLoaded] = useState(false)

  // useFont hook from "expo-font" is the default today!
  useEffect(() => {
    fetchFonts()
      .then(() => setIsLoaded(true))
      .catch(() => setIsLoaded(true))
  }, [])
}
```

```

if (!isLoading) {
  return (
    <SafeAreaView style={[ styles.container, styles.loadingContainer ]}>
      <ActivityIndicator size="large" color={colors.SECONDARY} />
    </SafeAreaView>
  )
}

const handleRestart = () => {
  setGuessRounds(0)
  setUserNum(0)
}

const handleStartGame = (selectedNum) => setUserNum(selectedNum)

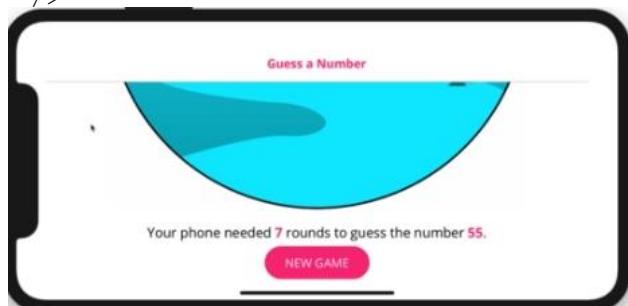
const handleGameOver = (numGuesses) => setGuessRounds(numGuesses)

let content = <StartGameScreen onStartGame={handleStartGame} />

if (userNum && guessRounds <= 0) {
  content = <GameScreen userChoice={userNum} onGameOver={handleGameOver} />
} else if (guessRounds > 0) {
  content = (
    <GameOverScreen {...{ userNum, guessRounds }} onRestart={handleRestart} />
  )
}

return (
  <SafeAreaView style={styles.container}>
    <Header title="Guess a #" />
    {content}
  </SafeAreaView>
)
}

```



```

const styles = StyleSheet.create({
  container: { flex: 1 },
  loadingContainer: { justifyContent: "center", alignItems: "center" }
})

function fetchFonts() {
  return Font.loadAsync({
    "open-sans": require("./assets/fonts/OpenSans-Regular.ttf"),
    "open-sans-bold": require("./assets/fonts/OpenSans-Bold.ttf")
  })
}

```

Navigation and React Navigation

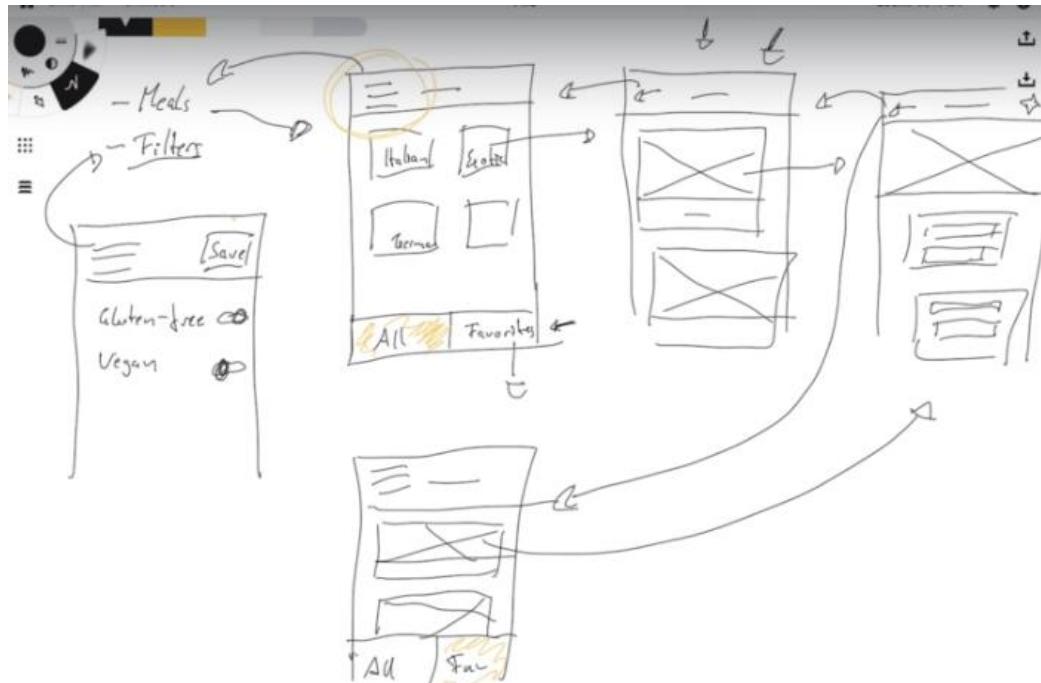
Intro

So far we created an app that renders a component depending on if checks.

While this was sufficient enough for a simple game app as we did, which only shares a couple of components and almost no state, this is not sufficient for big and organized apps.

For that, we need some way to navigate between screens passing params. There are lots of solutions out there, but the most used one is **React Navigation**.

Here's the structure of the app we are to create to showcase it:



Adding screens

First, we are to create some screen files.

Then, since we need some fonts to load before app starts, we require a way to show loading status.

In the last app, we used `ActivityIndicator` from RN. Now, we introduce `AppLoading`.

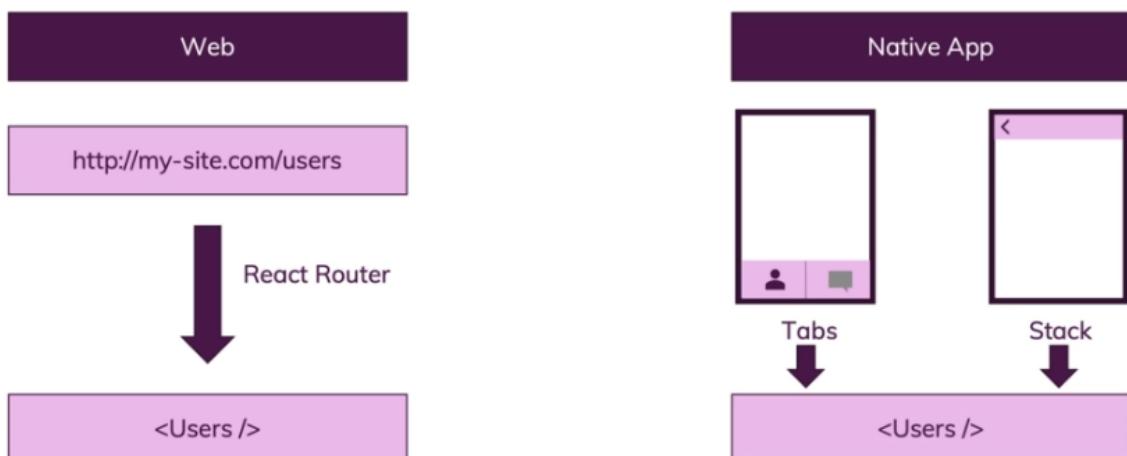
```
$ expo install expo-app-loading
```

This is a wrapper component that makes sure app does not start until the required async asset finished loading.

The required assets we need are two fonts, so we also require `expo-font` to process it.

```
$ expo install expo-font
```

Adding navigation



First, let's install React Navigation.

```
$ npm install @react-navigation/native
```

Note: always refer to the official docs!!

This way, we ensured react navigation for expo needs some other extra libraries to work for gestures, animations, screen changing and so on (and React Native cli needs some other ones)

```
$ expo install react-native-screens react-native-safe-area-context react-native-reanimated @react-native-community/masked-view
```

If you're using React Navigation v4 or higher, everything works as shown in this module but there is one important difference: You need to install the different navigators which we'll use in this module (`StackNavigator`, `DrawerNavigator`, `TabsNavigator`) separately.

So when we use the `StackNavigator` (= next lecture), run

```
$ npm install --save react-navigation-stack
```

before you start using it (with v3 and lower, it was part of react-navigation itself).

Also add this import in the file where you are using `createStackNavigator`:

```
import { createStackNavigator } from 'react-navigation-stack'
```

Same for `TabsNavigator` (used a little bit later in this module):

```
$ npm install --save react-navigation-tabs
```

```
import { createBottomTabNavigator } from 'react-navigation-tabs';
```

And also for `DrawerNavigator` (also used later in this module):

```
$ npm install --save react-navigation-drawer
```

```
import { createDrawerNavigator } from 'react-navigation-drawer'
```

Note: This was necessary for React Navigator versions 4-, which we will learn now as many projects still use it. Later we will learn how to convert to the most recent v5+.

Navigation/MealsNavigator.js

```
import { createAppContainer } from "react-navigation"
import { createStackNavigator } from "react-navigation-stack"
import CategoriesScreen from "../screens/Categories"
import CategoryMealsScreen from "../screens/CategoryMeals"
import MealDetailsScreen from "../screens/MealDetails"

const MealsNavigator = createStackNavigator({
  Categories: CategoriesScreen,
  CategoryMeals: CategoryMealsScreen,
```

```

MealDetails: MealDetailsScreen
})

export default createAppContainer(MealsNavigator)

App.js
import React, { useEffect, useState } from "react"
import { StyleSheet } from "react-native"
import * as Font from "expo-font"
import AppLoading from "expo-app-loading"
import MealsNavigator from "./navigation/MealsNavigator"

export default function App() {
  const [fontLoaded, setFontLoaded] = useState(false)

  useEffect(() => {
    fetchFonts()
      .then(() => setFontLoaded(true))
      .catch(console.error)
  }, [])

  return !fontLoaded ? <AppLoading /> : <MealsNavigator />
}

const styles = StyleSheet.create({
  container: {
    flex: 1, alignItems: "center", justifyContent: "center"
  }
})

async function fetchFonts() {
  return await Font.loadAsync({
    "open-sans": require("./assets/fonts/OpenSans-Regular.ttf"),
    "open-sans-bold": require("./assets/fonts/OpenSans-Bold.ttf")
  })
}

```

Screens/Categories.js

```
import React from "react"
import { View, Text, StyleSheet, Button } from "react-native"

export default function Categories(props) {
  return (
    <View style={_styles.container}>
      <Text>Categories</Text>
      <Button
        title="Go to meals"
        onPress={() =>
          props.navigation.navigate({ routeName: "CategoryMeals" })
        }
      />
    </View>
  )
}

const _styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" }
})
```

Screens/CategoryMeals.js

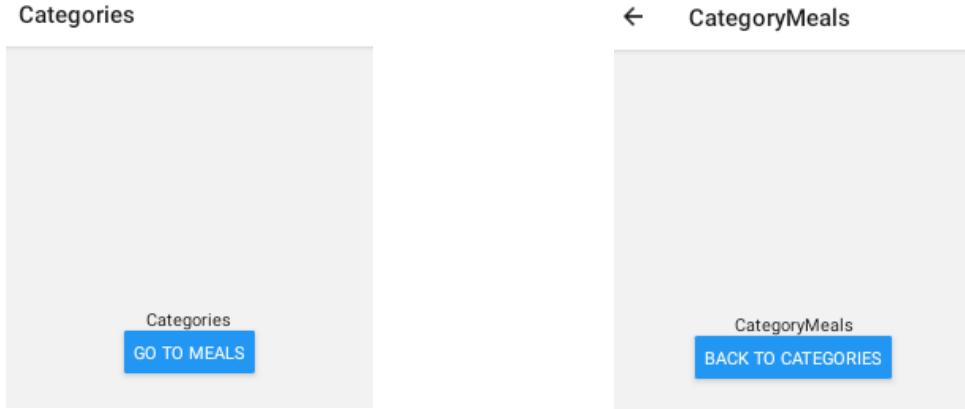
```
import React from "react"
import { View, Text, StyleSheet, Button } from "react-native"

export default function CategoryMeals(props) {
  return (
    <View style={_styles.container}>
      <Text>CategoryMeals</Text>
      <Button
        title="Back to categories"
        onPress={() => props.navigation.navigate("Categories")}
      />
    </View>
  )
}

const _styles = StyleSheet.create({
```

```
    container: { flex: 1, justifyContent: "center", alignItems: "center"
  }
}

})
```



Pushing, popping, replacing

Instead of using 'navigate', you can resort to '**push**', which can navigate to the same screen you are on ('navigate' does not allow this).

```
props.navigation.push("Categories")
```

This is useful, for example, in apps like DropBox which uses the same screen for different folders. So this way you can "reload" the screen.

Now, if you want to trigger a 'go back' navigation manually, you use 'goBack'

```
props.navigation.goBack()
```

'pop' will remove the current top-most screen in the stack (and hence, triggering a 'go back' behavior, but now incapable of returning to the popped screen. This can only be used in a stack navigator.

```
props.navigation.pop()
```

To return to the bottom-most screen in the stack, popping everything on top, use 'popToTop'.

```
props.navigation.popToTop()
```

And to replace one screen in the stack (not add a new one):

```
props.navigation.replace("CategoryMeals")
```

```
screens/MealDetails.jsx
import React from "react"
import { View, Text, StyleSheet, Button } from "react-native"

export default function MealDetails(props) {
  return (
    <View style={_styles.container}>
      <Text>MealDetails</Text>
      <Button title="Go to Categories"
        onPress={() => props.navigation.push('Categories')} />
      <Button title="Go back"
        onPress={() => props.navigation.goBack()} />
      <Button title="Pop screen"
        onPress={() => props.navigation.pop()} />
      <Button title="To root"
        onPress={() => props.navigation.popToTop()} />
      <Button title="Replace with Categories"
        onPress={() => props.navigation.replace("Categories")}>
        />
    </View>
  )
}
```

Categories grid

```
Constants/colors.js
export const colors = {
  PRIMARY: "#4a148c",
  SECONDARY: "#FFCF00"
}
```

models/Category.js

```
export class Category {
  constructor(id, title, color) {
    this.id = id
    this.title = title
    this.color = color
  }
}
```

Data/dummy.js

```
import { Category } from "../models/Category"

export const CATEGORIES = [
  new Category("c1", "Italian", "#f5428d"),
  new Category("c2", "Quick & Easy", "#f54242"),
  new Category("c3", "Hamburgers", "#f5a442"),
  new Category("c4", "German", "#f5d142"),
  new Category("c5", "Light & Lovely", "#368dff"),
  new Category("c6", "Exotic", "#41d95d"),
  new Category("c7", "Breakfast", "#9eecff"),
  new Category("c8", "Asian", "#b9ffb0"),
  new Category("c9", "French", "#ffc7ff"),
  new Category("c10", "Summer", "#47fcfd")
]
```

Screens/Category.jsx

```
import React from "react"
import {
  View, Text, StyleSheet, TouchableOpacity, FlatList, Platform
} from "react-native"
import { CATEGORIES } from "../data/dummy"
import { colors } from "../constants/colors"

export default function Categories(props) {
  const renderItem = (data) => (
    <TouchableOpacity
      style={_styles.gridItem}
      onPress={() =>
        props.navigation.navigate({ routeName: "CategoryMeals" })
      }
    >
      <View>
        <Text>{data.item.title}</Text>
      </View>
    </TouchableOpacity>
  )

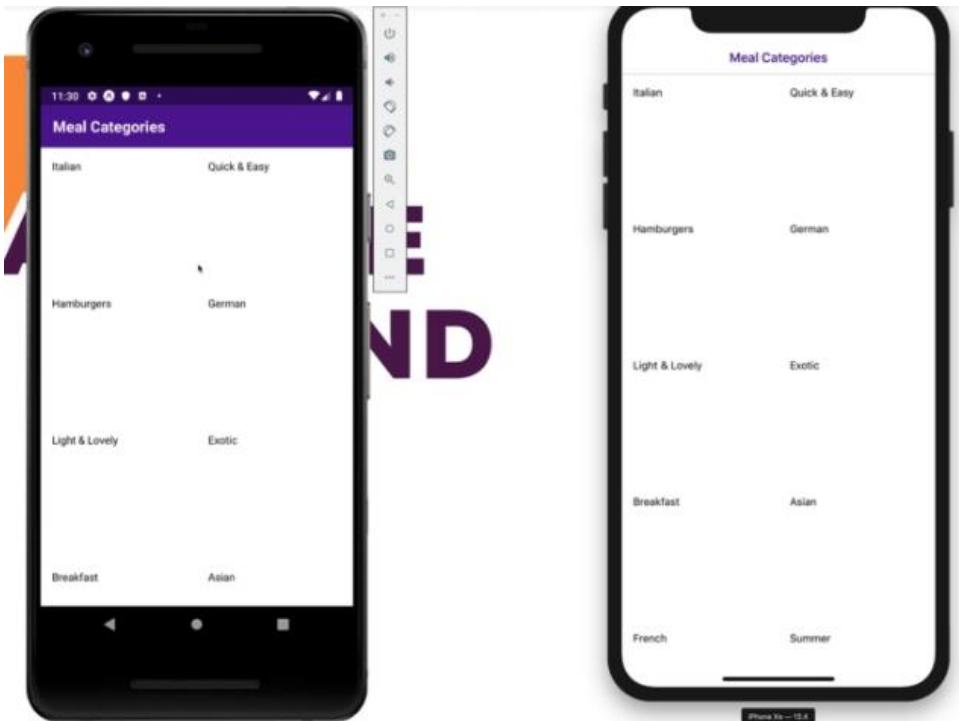
  return <FlatList
    data={CATEGORIES} renderItem={renderItem} numColumns={2} />
}
```

```

Categories.navigationOptions = {
  headerTitle: "Meal Categories",
  headerStyle: {
    backgroundColor: Platform.OS === "android" ? colors.PRIMARY : ""
  },
  headerTintColor:
    Platform.OS === "android" ? "white" : colors.PRIMARY
}

const _styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" },
  gridItem: { flex: 1, margin: 15, height: 150 }
})

```



Passing and reading navigation params

Screens/Categories.jsx

```

...
export default function Categories(props) {
  const renderItem = (data) => (
    <TouchableOpacity
      style={_styles.gridItem}

```

```

        onPress={() =>
          props.navigation.navigate({
            routeName: "CategoryMeals",
            params: { categoryId: data.item.id }
          })
        }
      >
      <View> <Text>{data.item.title}</Text> </View>
    </TouchableOpacity>
  )
}

return <FlatList
  data={CATEGORIES} renderItem={renderItem} numColumns={2} />
}
...

```

CategoryMeals.jsx

```

...
export default function CategoryMeals(props) {
  const categoryId = props.navigation.getParam("categoryId")

  const selectedCategory =
    CATEGORIES.find((cat) => cat.id === categoryId)

  return (
    <View style={_styles.container}>
      <Text>CategoryMeals</Text>
      <Text>{selectedCategory.title}</Text>
      <Button title="Go to details"
        onPress={() => props.navigation.navigate("MealDetails")}
      />
    </View>
  )
}

// vvv can be a function which receives react navigation obj!
CategoryMeals.navigationOptions = ({ navigation }) => {
  const categoryId = props.navigation.getParam("categoryId")

  const selectedCategory =
    CATEGORIES.find((cat) => cat.id === categoryId)

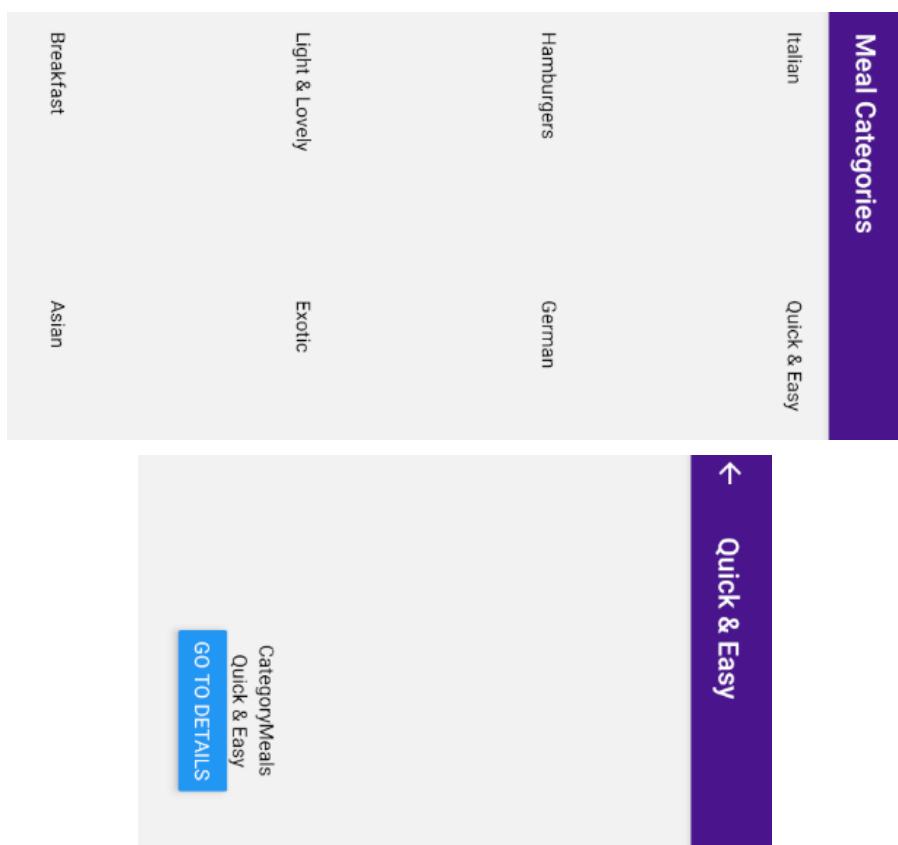
```

```

        return {
          headerTitle: selectedCategory.title,
          headerStyle: {
            backgroundColor: Platform.OS === "android" ? colors.PRIMARY : ""
          },
          headerTintColor:
            Platform.OS === "android" ? "white" : colors.PRIMARY
        }
      }

const _styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" }
})

```



Default Navigation options and configs

If you have navigation options that repeat across screens, you can set them up in "createStackNavigator" as a second argument.

Note that everything you set there can be overridden by props with the same name at each individual component's "navigatorOptions".

Also, navigator options set on each individual "createStackNavigator" properties will also override "defaultNavigatorOptions", and ALSO override navigation options on each single component.

So, to keep it short, precedences are as follows:

- (1) "createStackNavigator" first argument properties' "navigatorOptions"s
- (2) Individual component's "navigatorOptions"
- (3) "createStackNavigator" second argument "defaultNavigatorOptions" property

Second argument of "createStackNavigator" accepts "**mode**" property, which modifies the way transition between screens occur.

Defaults to "**card**", which slides to the sides, but can also be "**modal**", which transitions from top to bottom.

"**initialRouteName**" sets the first screen to load in the app.

Screens/MealsNavigator.jsx

```
import { Platform } from "react-native"
import { createAppContainer } from "react-navigation"
import { createStackNavigator } from "react-navigation-stack"
import CategoriesScreen from "../screens/Categories"
import CategoryMealsScreen from "../screens/CategoryMeals"
import MealDetailsScreen from "../screens/MealDetails"
import { colors } from "../constants/colors"

const MealsNavigator = createStackNavigator(
  {
    Categories: {
      screen: CategoriesScreen,
      navigationOptions: { headerTitle: "Meal Categories" } // (1)
    },
    CategoryMeals: CategoryMealsScreen,
    MealDetails: MealDetailsScreen
  },
  {
    // defaults "createStackNavigator" for every screen
    defaultNavigationOptions: {} // (3)
    navigationOptions: {
      headerStyle: {
        backgroundColor:
          Platform.OS === "android" ? colors.PRIMARY : ""
      }
    }
  }
)
```

```

        } ,
        headerTintColor:
          Platform.OS === "android" ? "white" : colors.PRIMARY
      } ,
      mode: "modal",
      initialRouteName: "Categories"
    }
  )
}

export default createAppContainer(MealsNavigator)

```

Also, '**react-native-screens**' package ensures performance is optimized when transitioning between screens, since it switches to, and controls, native animations for such.

App.js

```

import React, { useEffect, useState } from "react"
import { StyleSheet } from "react-native"
import * as Font from "expo-font"
import AppLoading from "expo-app-loading"
import { enableScreens } from "react-native-screens"
import MealsNavigator from "./navigation/MealsNavigator"

enableScreens()

export default function App() {
  const [fontLoaded, setFontLoaded] = useState(false)

  useEffect(() => {
    fetchFonts()
      .then(() => setFontLoaded(true))
      .catch(console.error)
  }, [])

  return !fontLoaded ? <AppLoading /> : <MealsNavigator />
}

const styles = StyleSheet.create({
  container: {
    flex: 1, alignItems: "center", justifyContent: "center"
  }
}

```

```

        }
    })

async function fetchFonts() {
    return await Font.loadAsync({
        "open-sans": require("./assets/fonts/OpenSans-Regular.ttf"),
        "open-sans-bold": require("./assets/fonts/OpenSans-Bold.ttf")
    })
}

```

Grid styling and refactoring

Screens/Categories.jsx

```

import React from "react"
import { FlatList } from "react-native"
import { CATEGORIES } from "../data/dummy"
import CategoryGridItem from "../components/CategoryGridItem"

export default function Categories(props) {
    const renderItem = (data) => (
        <CategoryGridItem
            title={data.item.title}
            color={data.item.color}
            onPress={() =>
                props.navigation.navigate({
                    routeName: "CategoryMeals",
                    params: { categoryId: data.item.id }
                })
            }
        />
    )

    return <FlatList
        data={CATEGORIES} renderItem={renderItem} numColumns={2} />
}

```

Components/CategoryGridItem.jsx

```

import React from "react"
import { TouchableOpacity, TouchableNativeFeedback, Platform, View,
Text, StyleSheet } from "react-native"

```

```

export default function CategoryGridItem({ title, onPress, color }) {
  const Component =
    Platform.OS === "android"
      ? TouchableNativeFeedback
      : TouchableOpacity

  return (
    <View style={_styles.container}>
      <Component style={{ flex: 1 }} onPress={onPress}>
        <View style={[_styles.view, { backgroundColor: color }]}>
          <Text style={_styles.title} numberOfLines={2}>
            {title}
          </Text>
        </View>
      </Component>
    </View>
  )
}

const _styles = StyleSheet.create({
  container: {
    flex: 1, margin: 15, height: 150,
    borderRadius: 10, // delimiter for ripple effect
    overflow: "hidden" // ripple effect's overflow block
  },
  view: {
    flex: 1, justifyContent: "flex-end", alignItems: "flex-end",
    padding: 15,
    // for iOS
    shadowColor: "black",
    shadowOpacity: 0.25,
    shadowOffset: { width: 0, height: 2 },
    shadowRadius: 10,
    // for android
    elevation: 3
  },
  title: {
    fontFamily: "open-sans-bold", fontSize: 16, textAlign: "right"
  }
})

```



Meal Models & data, loading and styling Meals

Models/meal.js

```
export class Meal {  
  constructor(  
    id, categoryIds, title, affordability, complexity,  
    imageUrl, duration, ingredients, steps, isGluttenFree,  
    isVegan, isVegetarian, isLactoseFree  
) {  
  this.id = id  
  this.categoryIds = categoryIds  
  this.title = title  
  this.affordability = affordability  
  this.complexity = complexity  
  this.imageUrl = imageUrl  
  this.duration = duration  
  this.ingredients = ingredients  
  this.steps = steps  
  this.isGluttenFree = isGluttenFree  
  this.isVegan = isVegan  
  this.isVegetarian = isVegetarian  
  this.isLactoseFree = isLactoseFree  
}  
}  
}
```

Screens/CategoryMeals.jsx

```
import React from "react"  
import { View, StyleSheet, FlatList } from "react-native"  
import MealItem from "../components/MealItem"  
import { CATEGORIES, MEALS } from "../data/dummy"  
  
export default function CategoryMeals(props) {  
  const categoryId = props.navigation.getParam("categoryId")  
  
  const displayedMeals = MEALS.filter(  
    (meal) => meal.categoryIds.indexOf(categoryId) !== -1  
  )  
  
  const renderItem = (data) => (  
    <MealItem  
      title={data.item.title}  
    >  
  )  
}
```

```

        duration={data.item.duration}
        complexity={data.item.complexity}
        affordability={data.item.affordability}
        imageUrl={data.item.imageUrl}
        onPress={() => {}}
    />
)

return (
<View style={_styles.container}>
    <FlatList
        data={displayedMeals}
        renderItem={renderItem}
        keyExtractor={(item, i) => item.id} // not necessary!
        style={{ width: "95%", marginTop: "2%" }}
    />
</View>
)
}

CategoryMeals.navigationOptions = ({ navigation }) => {
    const categoryId = navigation.getParam("categoryId")
    const selectedCategory =
        CATEGORIES.find((cat) => cat.id === categoryId)

    return { headerTitle: selectedCategory.title }
}

const _styles = StyleSheet.create({
    container: { flex: 1, justifyContent: "center", alignItems: "center" }
})

```

Components/MealItem.jsx

```

import React from "react"
import { View, Text, StyleSheet, TouchableOpacity, ImageBackground
} from "react-native"

export default function MealItem{
    title, imageUrl, duration, complexity, affordability, onPress
}
```

```

}) {
  return (
    <View style={_styles.container}>
      <TouchableOpacity onPress={onPress}>
        <View style={[_styles.row, _styles.header]}>
          <ImageBackground
            source={{ uri: imageUrl }} style={_styles.bgImg}>
        </>
        <View style={_styles.titleContainer}>
          <Text numberOfLines={1} style={_styles.title}>
            {title}
          </Text>
        </View>
        </ImageBackground>
      </View>
      <View style={[_styles.row, _styles.detail]}>
        <Text>{duration}m</Text>
        <Text>{complexity.toUpperCase()}</Text>
        <Text>{affordability.toUpperCase()}</Text>
      </View>
    </TouchableOpacity>
  </View>
)
}

const _styles = StyleSheet.create({
  container: {
    height: 200, width: "100%", backgroundColor: "lightgray",
    borderRadius: 10, overflow: "hidden", marginBottom: "2%"
  },
  row: { flexDirection: "row" },
  header: { height: "85%" },
  titleContainer: {
    backgroundColor: "rgba(0,0,0,0.5)",
    paddingVertical: 5, paddingHorizontal: 10
  },
  title: {
    fontFamily: "open-sans-bold", fontSize: 20,
    color: "white", textAlign: "center"
  },

```

```

        bgImg: { width: "100%", height: "100%", justifyContent: "flex-end"
    } ,
    detail: {
        paddingHorizontal: 10, justifyContent: "space-between",
        alignItems: "center", height: "15%"
    }
})

```



Passing data to meal details screen

Screens/MealDetails.jsx

```

import React from "react"
import { View, Text, StyleSheet, Button } from "react-native"
import { MEALS } from "../data/dummy"

export default function MealDetails(props) {
    const mealId = props.navigation.getParam("mealId")
    const selectedMeal = MEALS.find(meal) => meal.id === mealId

    return (
        <View style={_styles.container}>
            <Text>{selectedMeal.title}</Text>
            <Button
                title="Go back" onPress={() => props.navigation.goBack()} />
        </View>
    )
}

MealDetails.navigationOptions = ({ navigation }) => {
    const mealId = navigation.getParam("mealId")
}

```

```

const selectedMeal = MEALS.find((meal) => meal.id === mealId)

return { headerTitle: selectedMeal.title }
}

const _styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" }
})

```

Screens/CategoryMeals.jsx

```

import React from "react"
import { View, StyleSheet, FlatList } from "react-native"
import MealItem from "../components/MealItem"
import { CATEGORIES, MEALS } from "../data/dummy"

export default function CategoryMeals(props) {
  const categoryId = props.navigation.getParam("categoryId")

  const displayedMeals = MEALS.filter(
    (meal) => meal.categoryIds.indexOf(categoryId) !== -1
  )

  const renderItem = (data) => (
    <MealItem
      title={data.item.title} duration={data.item.duration}
      complexity={data.item.complexity}
      affordability={data.item.affordability}
      imageUrl={data.item.imageUrl}
      onPress={() => props.navigation.navigate({
        routeName: "MealDetails",
        params: { mealId: data.item.id }
      })}
    />
  )

  return (
    <View style={_styles.container}>
      <FlatList
        data={displayedMeals}

```

```

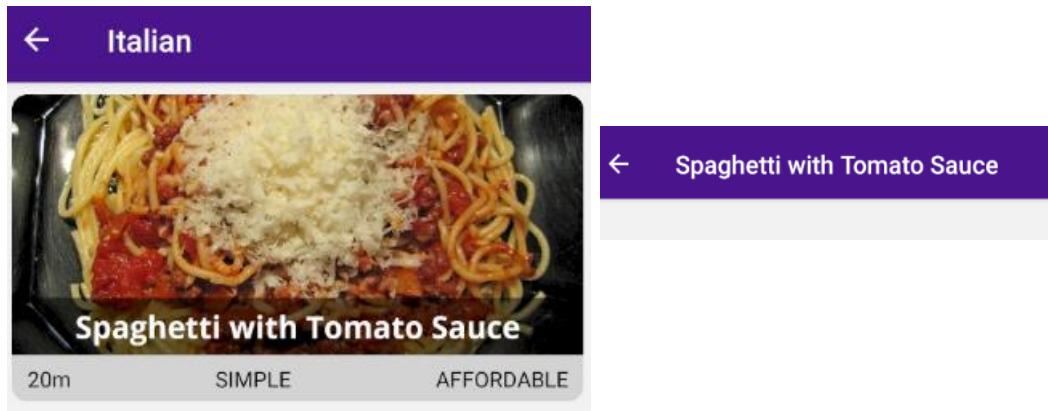
        renderItem={renderItem}
        keyExtractor={(item, i) => item.id} // not necessary!
        style={{ width: "95%", marginTop: "2%" }}
      />
    </View>
  )
}

CategoryMeals.navigationOptions = ({ navigation }) => {
  const categoryId = navigation.getParam("categoryId")
  const selectedCategory =
    CATEGORIES.find((cat) => cat.id === categoryId)

  return { headerTitle: selectedCategory.title }
}

const _styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" }
})

```



Adding buttons to header

You can use `headerRight` and `headerLeft` in `navigationOptions` to add controls, but it can be really cumbersome to make them look right for all devices.

In order to save the effort, we can use a `navigation buttons` package, also from React Navigation:

```
$ npm install --save react-navigation-header-buttons
```

Components/CustomHeaderButton.jsx

```
import React from "react"
import { Platform } from "react-native"
import { HeaderButton } from "react-navigation-header-buttons"
import { Ionicons } from "@expo/vector-icons"

import { colors } from "../constants/colors"

export default function CustomHeaderButton(props) {
  // IconComponent expects an import from @expo/vector-icons!
  return (
    <HeaderButton
      {...props}
      IconComponent={Ionicons}
      iconSize={23}
      color={Platform.OS === "iOS" ? colors.PRIMARY : "white"}
    />
  )
}
```

Screens/MealDetails.jsx

```
import React from "react"
import { View, Text, StyleSheet, Button } from "react-native"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import CustomHeaderButton from "../components/HeaderButton"
import { MEALS } from "../data/dummy"

export default function MealDetails(props) {
  const mealId = props.navigation.getParam("mealId")
  const selectedMeal = MEALS.find((meal) => meal.id === mealId)

  return (
    <View style={_styles.container}>
      <Text>{selectedMeal.title}</Text>
      <Button
        title="Go back" onPress={() => props.navigation.goBack()} />
    </View>
  )
}
```

```

MealDetails.navigationOptions = ({ navigation }) => {
  const mealId = navigation.getParam("mealId")
  const selectedMeal = MEALS.find((meal) => meal.id === mealId)

  return {
    headerTitle: selectedMeal.title,
    headerRight: () => (
      <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
        {/* you can have multiple '*Item*' inside '*HeaderButton*' */}
        {/* `title` is used as key and for aria */}
        {/* `iconName` is the vector-icons target icon name */}
        <Item
          title="Favorite" iconName="ios-star" onPress={() => {}} />
      </HeaderButtons>
    )
  }
}

```



```

const _styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" }
})

```

Correcting shadows

Shadows for Categories got lost in the process, and that's because they were cropped by overflow 'hidden'.

That property is necessary for android, but not for iOS, so we leverage on Platform API.

Components/CategoryGridItem.jsx

```

const _styles = StyleSheet.create({
  container: {
    flex: 1, margin: 15, height: 150, borderRadius: 10,
    // for ripple effect's overflowing issue, use 'hidden'. For iOS,
    // 'visible', since 'hidden' crops the shadow
    overflow:
      Platform.OS === "android" && Platform.Version >= 21
        ? "hidden"
        : "visible",
    // for android as not to conflict with overflow: "hidden"
    elevation: 5 ,
  }
})

```

```

view: {
  flex: 1, justifyContent: "flex-end", alignItems: "flex-end",
  padding: 15,
  // for iOS
  shadowColor: "black", shadowOpacity: 0.25,
  shadowOffset: { width: 0, height: 2 }, shadowRadius: 10
},
title: {
  fontFamily: "open-sans-bold", fontSize: 16, textAlign: "right"
}
})

```



Tab-based navigation

MealsNavigator.jsx

```

import React from "react"
import { Platform } from "react-native"
import { createAppContainer } from "react-navigation"
import { createStackNavigator } from "react-navigation-stack"
import { createBottomTabNavigator } from "react-navigation-tabs"
import { Ionicons } from "@expo/vector-icons"
import CategoriesScreen from "../screens/Categories"
import CategoryMealsScreen from "../screens/CategoryMeals"
import MealDetailsScreen from "../screens/MealDetails"
import FavoritesScreen from "../screens/Favorites"
import { colors } from "../constants/colors"

```

```

const MealsNavigator = createStackNavigator(
{
  Categories: {
    screen: CategoriesScreen,
    navigationOptions: { headerTitle: "Meal Categories" }
  },
  CategoryMeals: CategoryMealsScreen,
  MealDetails: MealDetailsScreen
},
{
// second arg of "createStackNavigator" are defaults for every screen
  defaultNavigationOptions: {
    headerStyle: {
      backgroundColor:
        Platform.OS === "android" ? colors.PRIMARY : ""
    },
    headerTintColor:
      Platform.OS === "android" ? "white" : colors.PRIMARY
  },
  mode: "modal",
  initialRouteName: "Categories"
}
)

const MealsFavTabNavigator = createBottomTabNavigator(
{
  // you can use a whole navigator stack!
  Meals: {
    screen: MealsNavigator,
    // you can also set `navigatorOptions` in the component file, or
    // in `createStackNavigator` second argument
    navigationOptions: {
      // tabConfigs are configs set in `tabBarOptions`
      tabBarIcon: (tabConfigs) => (
        <Ionicons name="ios-restaurant" size={25}>
          color={tabConfigs.tintColor}
        />
      )
    }
  },
}
)

```

```

// and single screens, also shorthanded `Favorites: FavoritesScreen`
Favorites: {
  screen: FavoritesScreen,
  navigationOptions: {
    tabBarLabel: "Favs",
    tabBarIcon: (tabConfigs) => (
      <Ionicons name="ios-star" size={25}
      color={tabConfigs.tintColor} />
    )
  }
},
{
  tabBarOptions: { activeTintColor: colors.SECONDARY }
}
)
}

// createAppContainer fuses both navigators into a root one
export default createAppContainer(MealsFavTabNavigator)

```



navigatorOptions inside of a Navigator

When defining a navigator, you can also add **navigationOptions** to it:

```

const SomeNavigator = createStackNavigator({
  ScreenIdentifier: SomeScreen
}, {
  navigationOptions: {
    // You can set options here!
    // Please note: This is NOT defaultNavigationOptions!
  }
});

```

Don't mistake this for the **defaultNavigationOptions** which you could also set there (i.e. in the second argument you pass to **createWhateverNavigator()**).

The **navigationOptions** you set on the navigator will NOT be used in its screens! That's the difference to **defaultNavigationOptions** - those options WILL be merged with the screens.

So what's the use of navigationOptions in that place then?

The options become important once you use the navigator itself as a screen in some other navigator - for example if you use some stack navigator (created via **createStackNavigator()**) in a tab navigator (e.g. created via **createBottomTabNavigator()**).

In such a case, the **navigationOptions** configure the "nested navigator" (which is used as a screen) for that "parent navigator". For example, you can use **navigationOptions** on the nested navigator that's used in a tab navigator to configure the tab icons.

Adding MaterialBottomTabs

The actual look-and-feel is iOS'ish, not so suited for Android devices. Let's use a package that converts tabs to material ui.

```
$ npm install --save react-navigation-material-bottom-tabs
```

And to really mimic material stylings, let's use paper too.

```
$ npm install --save react-native-paper
```

Navigation/MealsNavigator.jsx

```

...
import { createMaterialBottomTabNavigator } from "react-navigation-
material-bottom-tabs" ...

```

```

const MealsNavigator = createStackNavigator(
... // check in lesson above
)

const tabsScreenConfigs = {
  // you can use a whole navigator stack!
  Meals: {
    screen: MealsNavigator,
    // you can also set `navigatorOptions` in the component file, or
    // in `createStackNavigator` second argument
    navigationOptions: {
      // tabConfigs are configs set in `tabBarOptions`
      tabBarIcon: (tabConfigs) => (
        <Ionicons
          name="ios-restaurant" size={25}
          color={tabConfigs.tintColor}
        />
      ),
      tabBarColor: colors.PRIMARY
    }
  },
  // and single screens, also shorthanded `Favorites: FavoritesScreen`
  Favorites: {
    screen: FavoritesScreen,
    navigationOptions: {
      tabBarLabel: "Favs",
      tabBarIcon: (tabConfigs) => (
        <Ionicons
          name="ios-star" size={25} color={tabConfigs.tintColor} />
      ),
      tabBarColor: colors.SECONDARY
    }
  }
}

const MealsFavTabNavigator =
Platform.OS === "android"
? createMaterialBottomTabNavigator(tabsScreenConfigs, {
  activeColor: "white",

```

```

    // enables bgcolor shift using `tabBarColor` when toggling
    // tabs, with a rippling effect
    shifting: true
    // or use this if `shifting: false` (no ripple color change)
    // barStyle: { backgroundColor: colors.PRIMARY}
  ))
: createBottomTabNavigator(tabsScreenConfigs, {
  tabBarOptions: { activeTintColor: colors.SECONDARY }
})

```

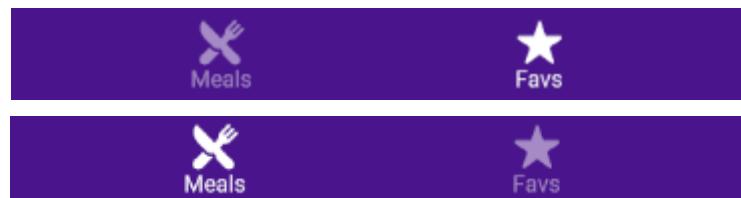
// createAppContainer fuses both navigators into a root one

```
export default createAppContainer(MealsFavTabNavigator)
```

// with `shifting = true`



// with `shifting = false` and `barStyle`



Adding a Favorites Stack

To navigate using the favorite option (star icon on each meal), we create a new navigator stack, which will be used complementary to the tab and stack navigation we already have.

Navigation/MealsNavigator.jsx

```

import React from "react"
import { Platform } from "react-native"
import { createAppContainer } from "react-navigation"
import { createStackNavigator } from "react-navigation-stack"

```

```
import { createBottomTabNavigator } from "react-navigation-tabs"
import { createMaterialBottomTabNavigator } from "react-navigation-material-bottom-tabs"
import { Ionicons } from "@expo/vector-icons"
import CategoriesScreen from "../screens/Categories"
import CategoryMealsScreen from "../screens/CategoryMeals"
import MealDetailsScreen from "../screens/MealDetails"
import FavoritesScreen from "../screens/Favorites"
import { colors } from "../constants/colors"

const defaultStackNavigatorOptions = {
  headerStyle: {
    backgroundColor: Platform.OS === "android" ? colors.PRIMARY : ""
  },
  headerTintColor: Platform.OS === "android" ? "white" : colors.PRIMARY,
  headerTitle: "Screen"
}

const MealsNavigator = createStackNavigator(
{
  Categories: {
    screen: CategoriesScreen,
    navigationOptions: { headerTitle: "Meal Categories" }
  },
  CategoryMeals: CategoryMealsScreen,
  MealDetails: MealDetailsScreen
},
{
  defaultNavigationOptions: defaultStackNavigatorOptions,
  mode: "modal", initialRouteName: "Categories"
}
)

const FavNavigation = createStackNavigator(
{
  Favorites: FavoritesScreen,
  MealDetails: MealDetailsScreen
},
{ defaultNavigationOptions: defaultStackNavigatorOptions }
)
```

```

const tabsScreenConfigs = {
  Meals: {
    screen: MealsNavigator,
    navigationOptions: {
      tabBarIcon: (tabConfigs) => (
        <Ionicons
          name="ios-restaurant"
          size={25}
          color={tabConfigs.tintColor}
        />
      ),
      tabBarColor: colors.PRIMARY
    }
  },
  Favorites: {
    screen: FavNavigation,
    navigationOptions: {
      tabBarLabel: "Favs",
      tabBarIcon: (tabConfigs) => (
        <Ionicons
          name="ios-star" size={25} color={tabConfigs.tintColor} />
      ),
      tabBarColor: colors.SECONDARY
    }
  }
}

const MealsFavTabNavigator =
  Platform.OS === "android"
    ? createMaterialBottomTabNavigator(tabsScreenConfigs, {
        activeColor: "white",
        shifting: true
      })
    : createBottomTabNavigator(tabsScreenConfigs, {
        tabBarOptions: { activeTintColor: colors.SECONDARY }
      })

export default createAppContainer(MealsFavTabNavigator)

```

components/MealList.jsx

```
import React from "react"
import { View, FlatList, StyleSheet } from "react-native"
import MealItem from "./MealItem"

export default function MealList({ data, navigation }) {
  const renderItem = (data) => (
    <MealItem
      title={data.item.title}
      duration={data.item.duration}
      complexity={data.item.complexity}
      affordability={data.item.affordability}
      imageUrl={data.item.imageUrl}
      onPress={() =>
        navigation.navigate({
          routeName: "MealDetails",
          params: { mealId: data.item.id }
        })
      }
    />
  )

  return (
    <View style={_styles.container}>
      <FlatList
        {...{ data, renderItem }}
        // keyExtractor={(item, i) => item.id} // not necessary!
        style={{ width: "95%", marginTop: "2%" }}
      />
    </View>
  )
}

const _styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" }
})
```

Screens/CategoryMeals.jsx

```
...
import MealList from "../components/MealList"
```

```

export default function CategoryMeals(props) {
  const categoryId = props.navigation.getParam("categoryId")

  const displayedMeals = MEALS.filter(
    (meal) => meal.categoryIds.indexOf(categoryId) !== -1
  )

  return (
    <MealList data={displayedMeals} navigation={props.navigation} />
  )
}

```

```

CategoryMeals.navigationOptions = ({ navigation }) => {
  const categoryId = navigation.getParam("categoryId")
  const selectedCategory =
    CATEGORIES.find((cat) => cat.id === categoryId)

  return { headerTitle: selectedCategory.title }
}

```

Screens/FavoritesScreen.jsx

```

import React from "react"
import MealList from "../components/MealList"
import { MEALS } from "../data/dummy"

export default function Favorites(props) {
  const dummyFavMeals = MEALS.filter(
    (meal) => meal.id === "m1" || meal.id === "m2"
  )

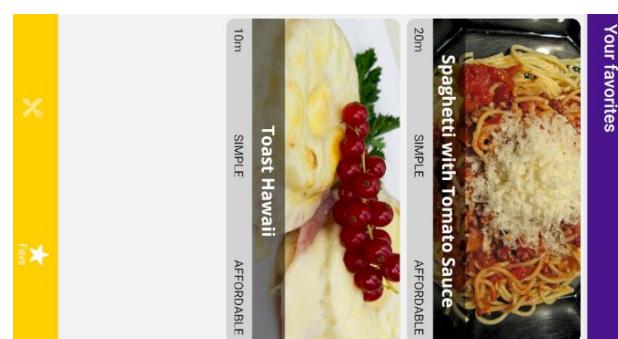
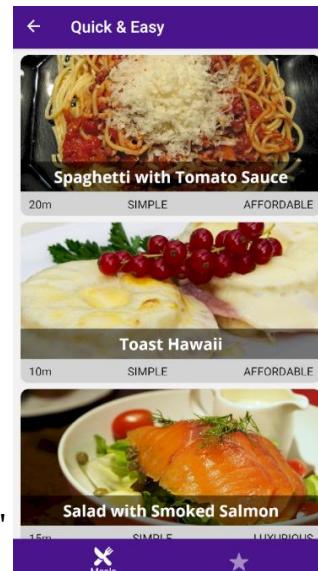
  return <MealList data={dummyFavMeals} navigation={props.navigation}>
/>
}

```

```

Favorites.navigationOptions = { headerTitle: "Your favorites" }

```



Menu button and Drawer navigation

Navigator/MealsNavigator.jsx

```
import React from "react"
import { Platform, SafeAreaView, StatusBar } from "react-native"
import { createAppContainer } from "react-navigation"
import { createStackNavigator } from "react-navigation-stack"
import { createBottomTabNavigator } from "react-navigation-tabs"
import { createDrawerNavigator, DrawerItems } from "react-navigation-drawer"
import { createMaterialBottomTabNavigator } from "react-navigation-material-bottom-tabs"
import { Ionicons } from "@expo/vector-icons"
import CategoriesScreen from "../screens/Categories"
import CategoryMealsScreen from "../screens/CategoryMeals"
import MealDetailsScreen from "../screens/MealDetails"
import FavoritesScreen from "../screens/Favorites"
import FiltersScreen from "../screens/Filters"
import { colors } from "../constants/colors"

const defaultStackNavigatorOptions = {
  // targets container <View>
  headerStyle: {
    backgroundColor: Platform.OS === "android" ? colors.PRIMARY : ""
  },
  // targets title <Text>
  headerTitleStyle: { fontFamily: "open-sans-bold" },
  // targets 'back' <Button> text in iOS
  headerBackTitleStyle: { fontFamily: "open-sans" },
  headerTintColor:
    Platform.OS === "android" ? "white" : colors.PRIMARY,
  headerTitle: "Screen"
}

const MealsNavigator = createStackNavigator(
{
  Categories: {
    screen: CategoriesScreen,
    navigationOptions: { headerTitle: "Meal Categories" }
  },
  CategoryMeals: CategoryMealsScreen,
  MealDetails: MealDetailsScreen
},
```

```

        {
          defaultNavigationOptions: defaultStackNavigatorOptions,
          mode: "modal",
          initialRouteName: "Categories"
        }
      )

const FavNavigation = createStackNavigator(
{
  Favorites: FavoritesScreen,
  MealDetails: MealDetailsScreen
},
{ defaultNavigationOptions: defaultStackNavigatorOptions }
)

const FiltersNavigator = createStackNavigator(
{
  Filters: FiltersScreen
},
{
  // navigationOptions is when it is actively used as a screen
  // > navigationOptions: { drawerLabel: 'Filters' },
  // defaultNavigation applies passively to all screens
  defaultNavigationOptions: defaultStackNavigatorOptions
}
)

const tabsScreenConfigs = {
  // you can use a whole navigator stack!
  Meals: {
    screen: MealsNavigator,
    // you can also set `navigatorOptions` in the component file, or
    // in `createStackNavigator` second argument
    navigationOptions: {
      // tabConfigs are configs set in `tabBarOptions`
      tabBarIcon: (tabConfigs) => (
        <Ionicons
          name="ios-restaurant"
          size={25}
          color={tabConfigs.tintColor}

```

```

        />
    ),
tabBarColor: colors.PRIMARY,
// targets label <Text> on each tab in android
tabBarLabel:
Platform.OS === "android" ? (
<Text style={{ fontFamily: "open-sans-bold" }}> Meals </Text>
) : (
"Meals"
)
}
},
// and single screens, also shorthanded `Favorites: FavoritesScreen
// | FavNavigation`
Favorites: {
screen: FavNavigation,
navigationOptions: {
tabBarLabel: "Favs",
tabBarIcon: (tabConfigs) => (
<Ionicons
name="ios-star" size={25} color={tabConfigs.tintColor} />
),
tabBarColor: colors.SECONDARY,
tabBarLabel:
Platform.OS === "android" ? (
<Text style={{ fontFamily: "open-sans-bold" }}> Meals </Text>
) : (
"Meals"
)
}
}
}

const MealsFavTabNavigator =
Platform.OS === "android"
? createMaterialBottomTabNavigator(tabsScreenConfigs, {
activeColor: "white",
// enables bgcolor shift using `tabBarColor` when toggling

```



```

        // tabs, with a rippling effect
        shifting: true
        // or use this if `shifting: false` (no ripple color change)
        // barStyle: { backgroundColor: colors.PRIMARY}
    })
: createBottomTabNavigator(tabsScreenConfigs, {
    tabBarOptions: {
        // targets label <Text> on each tab in iOS
        labelStyle: "open-sans",
        activeTintColor: colors.SECONDARY
    }
})
}

const MainNavigator = createDrawerNavigator(
{
    MealsFavs: {
        screen: MealsFavTabNavigator,
        navigationOptions: { drawerLabel: "Meals" }
    },
    Filters: FiltersNavigator
},
{
    // drawer items
    contentOptions: {
        activeTintColor: colors.SECONDARY,
        labelStyle: { fontFamily: "open-sans-bold" }
    },
    contentComponent: (props) => (
        <SafeAreaView
            style={{ flex: 1, paddingTop: StatusBar.currentHeight }}>
            <DrawerItems {...props} />
        </SafeAreaView>
    )
}
)

// createAppContainer fuses both navigators into a root one
export default createAppContainer(MainNavigator)

```

Screens/CategoriesScreen.jsx

```
import React from "react"
import { FlatList } from "react-native"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import { CATEGORIES } from "../data/dummy"
import CategoryGridItem from "../components/CategoryGridItem"
import CustomHeaderButton from "../components/HeaderButton"

export default function Categories(props) {
  const renderItem = (data) => (
    <CategoryGridItem
      title={data.item.title} color={data.item.color}
      onPress={() =>
        props.navigation.navigate({
          routeName: "CategoryMeals",
          params: { categoryId: data.item.id }
        })
      }
    />
  )

  return
  <FlatList data={CATEGORIES} renderItem={renderItem} numColumns={2} />
  Categories.navigationOptions = ({ navigation }) => ({
    headerLeft: (
      <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
        <Item
          title="menu"
          iconName="ios-menu"
          onPress={navigation.toggleDrawer}
        />
      </HeaderButtons>
    )
  })
}
```

Screens/FiltersScreen.jsx

```
import React from "react"
import { View, Text, StyleSheet } from "react-native"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
```

```

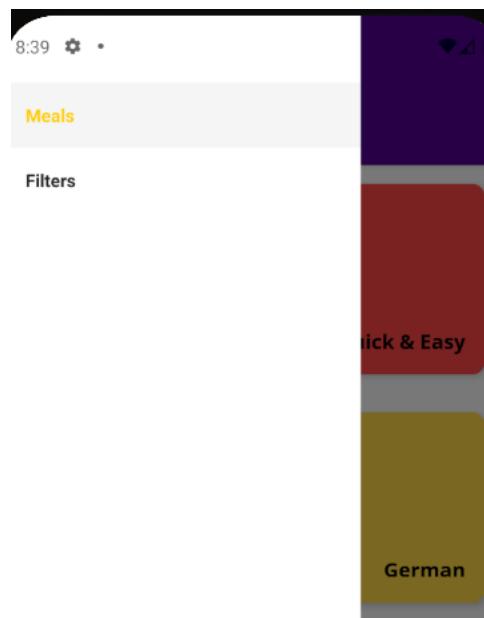
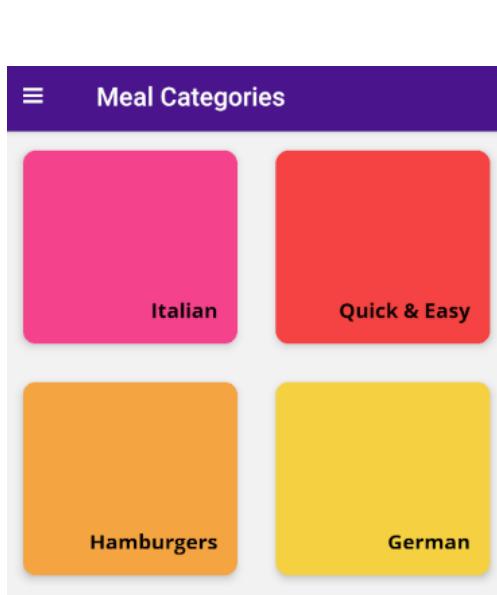
import CustomHeaderButton from "../components/HeaderButton"

export default function Filters(props) {
  return (
    <View style={_styles.container}>
      <Text>Filters</Text>
    </View>
  )
}

Filters.navigationOptions = ({ navigation }) => ({
  headerTitle: "Filter meals",
  headerLeft: (
    <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
      <Item
        title="menu"
        iconName="ios-menu"
        onPress={navigation.toggleDrawer}
      />
    </HeaderButtons>
  )
})
}

const _styles = StyleSheet.create({
  container: { flex: 1, justifyContent: "center", alignItems: "center" }
})

```



Adding styles and content

Components/DefaultText.jsx

```
import React from "react"
import { Text, StyleSheet } from "react-native"

export default function DefaultText({ children }) {
  return <Text style={_styles.container}>{children}</Text>
}

const _styles = StyleSheet.create({
  container: { fontFamily: "open-sans" } })
```

components/MealItem.jsx

```
import React from "react"
import {
  View, Text, StyleSheet, TouchableOpacity, ImageBackground
} from "react-native"
import DefaultText from "./DefaultText"

export default function MealItem({
  title, imageUrl, duration, complexity, affordability, onPress
}) {
  return (
    <View style={_styles.container}>
      <TouchableOpacity onPress={onPress}>
        <View style={[_styles.row, _styles.header]}>
          <ImageBackground
            source={{ uri: imageUrl }} style={_styles.bgImg}>
            <View style={_styles.titleContainer}>
              <Text numberOfLines={1} style={_styles.title}>
                {title}
              </Text>
            </View>
          </ImageBackground>
        </View>
        <View style={[_styles.row, _styles.detail]}>
          <DefaultText>{duration}m</DefaultText>
          <DefaultText>{complexity.toUpperCase()}</DefaultText>
        </View>
      </TouchableOpacity>
    </View>
  )
}
```

```

        <DefaultText>{affordability.toUpperCase()}</DefaultText>
    </View>
</TouchableOpacity>
</View>
)
}

const _styles = StyleSheet.create({
  container: {
    height: 200, width: "100%", backgroundColor: "lightgray",
    borderRadius: 10, overflow: "hidden", marginBottom: "2%"
  },
  row: { flexDirection: "row" },
  header: { height: "85%" },
  titleContainer: {
    backgroundColor: "rgba(0,0,0,0.5)", paddingVertical: 5,
    paddingHorizontal: 10
  },
  title: {
    fontFamily: "open-sans-bold", fontSize: 20,
    color: "white", textAlign: "center"
  },
  bgImg: { width: "100%", height: "100%", justifyContent: "flex-end" },
  detail: {
    paddingHorizontal: 10, justifyContent: "space-between",
    alignItems: "center", height: "15%"
  }
})

```



MealDetails screen content

Components/MealDetails.jsx

```

import React from "react"
import { ScrollView, View, Text, StyleSheet, Image } from "react-native"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import DefaultText from "../components/DefaultText"
import CustomHeaderButton from "../components/HeaderButton"
import { MEALS } from "../data/dummy"

```

```

export default function MealDetails(props) {
  const mealId = props.navigation.getParam("mealId")
  const selectedMeal = MEALS.find((meal) => meal.id === mealId)

  return (
    <ScrollView>
      <Image source={{ uri: selectedMeal.imageUrl }} style={_styles.image} />
      <View style={[_styles.details]}>
        <DefaultText>{selectedMeal.duration}m</DefaultText>
        <DefaultText>
          {selectedMeal.complexity.toUpperCase()}</DefaultText>
        <DefaultText>
          {selectedMeal.affordability.toUpperCase()}</DefaultText>
        </View>
        <BasicList
          data={selectedMeal.ingredients} title="Ingredients" />
        <BasicList
          data={selectedMeal.steps} title="Steps" />
      </ScrollView>
    )
  }

  MealDetails.navigationOptions = ({ navigation }) => {
    const mealId = navigation.getParam("mealId")
    const selectedMeal = MEALS.find((meal) => meal.id === mealId)

    return {
      headerTitle: selectedMeal.title,
      headerRight: () => (
        <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
          <Item
            title="Favorite" iconName="ios-star" onPress={() => {}} />
        </HeaderButtons>
      )
    }
  }

  const _styles = StyleSheet.create({
    image: { width: "100%", height: 200 },
  }
)
}

```

```

details: {
  flexDirection: "row", padding: 15, justifyContent: "space-around"
},
title: {
  fontFamily: "open-sans-bold", fontSize: 20, textAlign: "center"
},
listItem: {
  marginVertical: 10, marginHorizontal: 20, borderColor: "#ccc",
  borderWidth: 1, padding: 10
}
}

function BasicList({ data, title }) {
  return (
    <>
      <Text style={_styles.title}>{title}</Text>
      {data.map((item) => (
        <View key={item} style={_styles.listItem}>
          <DefaultText>{item}</DefaultText>
        </View>
      ))}
    </>
  )
}

```

The screenshot displays a mobile application interface for a meal planning or recipe app. At the top, there's a purple header bar with a back arrow, the title "Classic Hamburger", and a star icon. Below the header is a large image of a "Classic Hamburger" sandwich.

Below the image, the recipe details are listed:

- Time: 45m
- Difficulty: SIMPLE
- Cost: PRICEY

Ingredients

- 300g Cattle Hack
- 1 Tomato
- 1 Cucumber
- 1 Onion

Steps

- Form 2 patties
- Fry the patties for c. 4 minutes on each side
- Quickly fry the buns for c. 1 minute on each side
- Brush buns with ketchup
- Serve burger with tomato, cucumber and onion

At the bottom, there's a purple footer bar with a "Meals" icon and a star icon.

Filters screen content

Screens/FiltersScreens.jsx

```
import React, { useCallback, useEffect, useState } from "react"
import { View, Text, StyleSheet, Switch, Platform } from "react-native"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import CustomHeaderButton from "../components/HeaderButton"
import { colors } from "../constants/colors"

export default function Filters({ navigation }) {
  const [isGluttenFree, setIsGluttenFree] = useState(false)
  const [isLactoseFree, setIsLactoseFree] = useState(false)
  const [isVegan, setIsVegan] = useState(false)
  const [isVegetarian, setIsVegetarian] = useState(false)

  const saveFilters = useCallback(() => {
    const appliedFilters = {
      isGluttenFree, isLactoseFree, isVegan, isVegetarian
    }

    console.log(appliedFilters)
    , [isGluttenFree, isLactoseFree, isVegan, isVegetarian])
  }

  useEffect(() => {
    // setParams merges navigation param values for current screen
    navigation.setParams({ save: saveFilters })
    // vvv no navigation on dependencies! It always reconstructs
  , [saveFilters])

  return (
    <View style={_styles.container}>
      <Text style={_styles.title}>Filters</Text>
      <FilterSwitch
        title="Gluten-free" value={isGluttenFree}
        onValueChange={setIsGluttenFree}
      />
      <FilterSwitch
        title="Lactose-free" value={isLactoseFree}
        onValueChange={setIsLactoseFree}
      />
    
```

```

        <FilterSwitch
            title="Vegan" value={isVegan} onChange={setIsVegan} />
        <FilterSwitch
            title="Vegetarian" value={isVegetarian}
            onChange={setIsVegetarian}
        />
    </View>
)
}

Filters.navigationOptions = ({ navigation }) => ({
    headerTitle: "Filter meals",
    headerLeft: () => (
        <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
            <Item title="menu" iconName="ios-menu"
                onPress={navigation.toggleDrawer}
            />
        </HeaderButtons>
    ),
    headerRight: () => (
        <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
            <Item title="save" iconName="ios-save"
                // trigger the function passed above as new param
                onPress={navigation.getParam("save")}
            />
        </HeaderButtons>
    )));
}

const _styles = StyleSheet.create({
    container: { flex: 1, alignItems: "center" },
    title: {
        fontFamily: "open-sans-bold", fontSize: 20,
        margin: 20, textAlign: "center"
    },
    filterContainer: {
        flexDirection: "row", justifyContent: "space-between",
        alignItems: "center", width: "80%", marginVertical: 15
    }
})

```

```

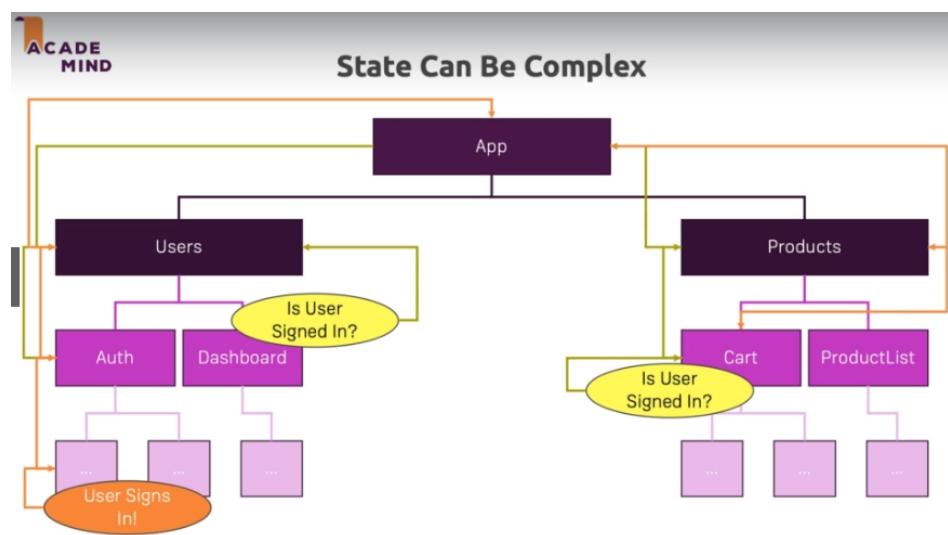
function FilterSwitch({ title, value, onValueChange }) {
  return (
    <View style={_styles.filterContainer}>
      <Text>{title}</Text>
      <Switch
        trackColor={{
          true: Platform.OS === "android" ? colors.PRIMARY : ""
        }}
        thumbColor={Platform.OS === "android" ? colors.PRIMARY : ""}
        {...{ value, onValueChange } }
      />
    </View>
  )
}

INFO    Object {
19:14      "isGluttenFree": false,
          "isLactoseFree": true,
          "isVegan": false,
          "isVegetarian": true,
        }

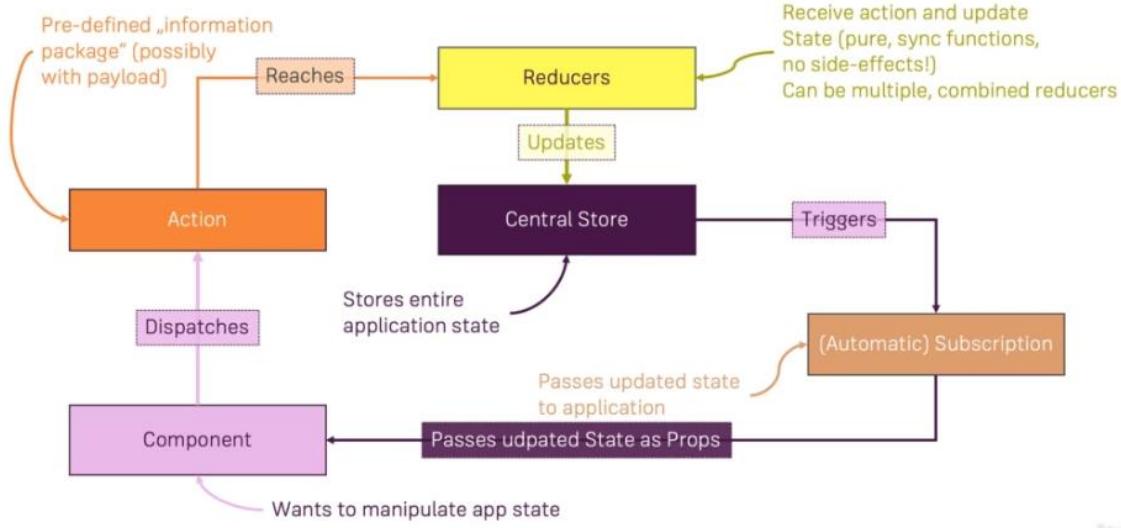
```

State management and redux

Intro



Redux to the Rescue!



Redux and store setup

First, install the packages needed.

```
$ expo install redux react-redux
```

App.js

```
import React, { useEffect, useState } from "react"
import { StyleSheet } from "react-native"
import * as Font from "expo-font"
import AppLoading from "expo-app-loading"
import { enableScreens } from "react-native-screens"
import { createStore, combineReducers } from "redux"
import { Provider } from "react-redux"
import MealsNavigator from "./navigation/MealsNavigator"
import mealsReducer from "./store/reducers/meals"

enableScreens()

const rootReducer = combineReducers({ meals: mealsReducer })
const store = createStore(rootReducer)

export default function App() {
  const [fontLoaded, setFontLoaded] = useState(false)
```

```

useEffect(() => {
  fetchFonts().then(() => setFontLoaded(true)).catch(console.error)
}, [])

return !fontLoaded ? (
  <AppLoading />
) : (
  <Provider store={store}>
    <MealsNavigator />
  </Provider>
)
}

const styles = StyleSheet.create({
  container: { flex: 1, alignItems: "center", justifyContent: "center" }
})

async function fetchFonts() {
  return await Font.loadAsync({
    "open-sans": require("./assets/fonts/OpenSans-Regular.ttf"),
    "open-sans-bold": require("./assets/fonts/OpenSans-Bold.ttf")
  })
}

```

Screens/CategoryMeals.jsx

```

import React from "react"
import { useSelector } from "react-redux"
import MealList from "../components/MealList"
import { CATEGORIES } from "../data/dummy"

export default function CategoryMeals(props) {
  const categoryId = props.navigation.getParam("categoryId")

  const availableMeals =
    useSelector((state) => state.meals.filteredMeals)

  const displayedMeals = availableMeals.filter(
    (meal) => meal.categoryIds.indexOf(categoryId) !== -1
  )
}

```

```

        return <MealList data={displayedMeals} navigation={props.navigation}>
    />
}

CategoryMeals.navigationOptions = ({ navigation }) => {
    const categoryId = navigation.getParam("categoryId")
    const selectedCategory =
        CATEGORIES.find((cat) => cat.id === categoryId)

    return { headerTitle: selectedCategory.title }
}

```

Screens/FavoritesScreen.jsx

```

import React from "react"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import { useSelector } from "react-redux"
import CustomHeaderButton from "../components/HeaderButton"
import MealList from "../components/MealList"

export default function Favorites(props) {
    const favoriteMeals =
        useSelector((state) => state.meals.favoriteMeals)

    return <MealList data={favoriteMeals} navigation={props.navigation}>
    />
}

Favorites.navigationOptions = ({ navigation }) => ({
    headerTitle: "Your favorites",
    headerLeft: (
        <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
            <Item title="menu" iconName="ios-menu"
                onPress={navigation.toggleDrawer}>
            />
        </HeaderButtons>
    )
})

```

*Notice navigationOptions does NOT have access to data stored in redux, and **useEffect to set navigation params is not a viable option** since it will not be available on first render.*

The way to solve this is to add the necessary params on each screen that has the ability to load the one that needs the proper param to work.

For example:

Components/MealItem.jsx

```
...
export default function MealList({ data, navigation }) {
  const renderItem = (data) => (
    <MealItem
      title={data.item.title} duration={data.item.duration}
      complexity={data.item.complexity}
      affordability={data.item.affordability}
      imageUrl={data.item.imageUrl}
      onPress={() =>
        navigation.navigate({
          routeName: "MealDetails",
          params: { mealId: data.item.id, mealTitle: data.item.title }
        })
      }
    />
  )
...
}
```

Screens/MealDetailsScreen.jsx

```
...
import { useSelector } from "react-redux"
...

export default function MealDetails(props) {
  const meals = useSelector((state) => state.meals.meals)
  const mealId = props.navigation.getParam("mealId")
  const selectedMeal = meals.find(meal) => meal.id === mealId

  useEffect() => {
    props.navigation.setParams({ mealTitle: selectedMeal.title })
  , [selectedMeal]
}

return ...
}
```

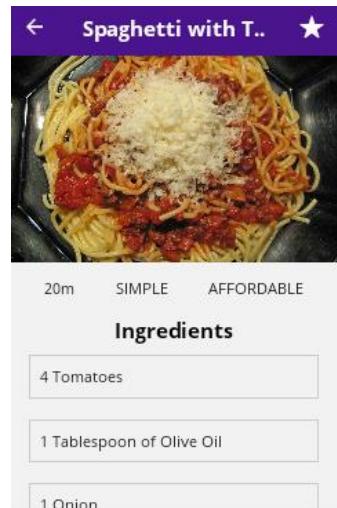
```

MealDetails.navigationOptions = ({ navigation }) => {
  // here, param comes from MealItem.jsx
  const mealTitle = navigation.getParam("mealTitle")

  return {
    headerTitle: mealTitle,
    headerRight: () => (
      <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
        <Item title="Favorite" iconName="ios-star" onPress={() => {}} />
      </HeaderButtons>
    )
  }
}

{
  <Image alt="Spaghetti with Tomato Sauce" style={{ width: '100%', height: 150 }} />
  <Text style={{ color: 'white', position: 'absolute', top: 0, left: 0, padding: 5px, background: 'black', opacity: 0.8, width: 150, height: 30 }}>Italian</Text>
  <Text style={{ color: 'white', position: 'absolute', bottom: 0, left: 0, padding: 5px, background: 'black', opacity: 0.8, width: 150, height: 30 }}>Spaghetti with Tomato Sau..</Text>
  <Text style={{ color: 'white', position: 'absolute', bottom: 0, left: 0, padding: 5px, background: 'black', opacity: 0.8, width: 150, height: 30 }}>20m SIMPLE AFFORDABLE</Text>
}

```



Dispatching actions and reducer logic

Store/actions/meals.js

```

export const TOGGLE_FAVORITE = "TOGGLE_FAVORITE"

export const toggleFavorite =
  (mealId) => ({ type: TOGGLE_FAVORITE, mealId })

```

Store/reducer/meals.js

```

import { MEALS } from "../../data/dummy"
import { TOGGLE_FAVORITE } from "../actions/meals"

const initialState =
  { meals: MEALS, filteredMeals: MEALS, favoriteMeals: [] }

const mealsReducer = (state = initialState, action) => {
  switch (action.type) {
    case TOGGLE_FAVORITE:

```

```

        const existingIdx = state.favoriteMeals.findIndex(
            (meal) => meal.id === action.mealId
        )
        if (existingIdx !== -1) {
            return {
                ...state,
                favoriteMeals: state.favoriteMeals
                    .slice(0, existingIdx)
                    .concat(state.favoriteMeals.slice(existingIdx + 1))
            }
        } else {
            const mealToAdd =
                state.meals.find((meal) => meal.id === action.mealId)
            return {
                ...state,
                favoriteMeals: state.favoriteMeals.concat(mealToAdd)
            }
        }
    }

    default: return state
}
}

export default mealsReducer

```

components/MealList.jsx

```

import React from "react"
import { View, FlatList, StyleSheet } from "react-native"
import { useSelector } from "react-redux"
import MealItem from "./MealItem"

export default function MealList({ data, navigation }) {
    // forward favoriteMeal from here to preload the star on MealDetails
    // load. Otherwise, the star will flicker (loads on screen render)
    const favoriteMeals =
        useSelector((state) => state.meals.favoriteMeals)

    const renderItem = (data) => {
        const isFav =
            favoriteMeals.find((meal) => meal.id === data.item.id)

```

```

        return (
            <MealItem
                title={data.item.title}
                duration={data.item.duration}
                complexity={data.item.complexity}
                affordability={data.item.affordability}
                imageUrl={data.item.imageUrl}
                onPress={() =>
                    navigation.navigate({
                        routeName: "MealDetails",
                        params: { mealId: data.item.id,
                            mealTitle: data.item.title, isFav }
                    })
                }
            />
        )
    }

    return (
        <View style={_styles.container}>
            <FlatList {...{ data, renderItem }}>
                style={{ width: "95%", marginTop: "2%" }}
            />
        </View>
    )
}

...

```

Screens/MealDetailsScreen.jsx

```

...
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import { useSelector, useDispatch } from "react-redux"
import DefaultText from "../components/DefaultText"
import CustomHeaderButton from "../components/HeaderButton"
import { toggleFavorite } from "../store/actions/meals"

export default function MealDetails(props) {
    const meals = useSelector((state) => state.meals.meals)
    const mealId = props.navigation.getParam("mealId")
    const selectedMeal = meals.find((meal) => meal.id === mealId)

```

```

const isCurrMealFav = useSelector((state) =>
  state.meals.favoriteMeals.some((meal) => meal.id === mealId)
)
const dispatch = useDispatch()

const toggleFavoriteHandler = useCallback(
  () => dispatch(toggleFavorite(mealId)),
  [dispatch, mealId]
)

useEffect(() => {
  props.navigation.setParams({ toggleFav: toggleFavoriteHandler })
}, [toggleFavoriteHandler])

useEffect(() => {
  props.navigation.setParams({ isFav: isCurrMealFav })
}, [isCurrMealFav])

return (
  <ScrollView>
    <Image source={{ uri: selectedMeal.imageUrl }} style={_styles.image} />
    <View style={[_styles.details]}>
      <DefaultText>{selectedMeal.duration}m</DefaultText>
      <DefaultText>{selectedMeal.complexity.toUpperCase()}</DefaultText>
      <DefaultText>{selectedMeal.affordability.toUpperCase()}</DefaultText>
    </View>
    <BasicList data={selectedMeal.ingredients} title="Ingredients" />
    <BasicList data={selectedMeal.steps} title="Steps" />
  </ScrollView>
)
}

MealDetails.navigationOptions = ({ navigation }) => {
  const mealTitle = navigation.getParam("mealTitle")
  const toggleFav = navigation.getParam("toggleFav")
  const isFav = navigation.getParam("isFav")
}

```

```

        return {

            headerTitle: mealTitle,
            headerRight: () => (
                <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
                    <Item
                        title="Favorite" onPress={toggleFav}
                        // `isFav` comes from MealList component and from above!
                        iconName={isFav ? "ios-star" : "ios-star-outline"}
                    />
                </HeaderButtons>
            )
        }
    }
}

```

Screens/FavoritesScreen.jsx

```

import React from "react"
import { View, StyleSheet } from "react-native"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import { useSelector } from "react-redux"
import DefaultText from "../components/DefaultText"
import CustomHeaderButton from "../components/HeaderButton"
import MealList from "../components/MealList"

export default function Favorites(props) {
    const favoriteMeals =
        useSelector((state) => state.meals.favoriteMeals)

    if (!favoriteMeals || !favoriteMeals.length) {
        return (
            <View style={_styles.container}>
                <DefaultText>
                    No favorite meals. Start adding some!</DefaultText>
            </View>
        )
    }

    return <MealList data={favoriteMeals} navigation={props.navigation} />
}
}

Favorites.navigationOptions = ({ navigation }) => ({

```

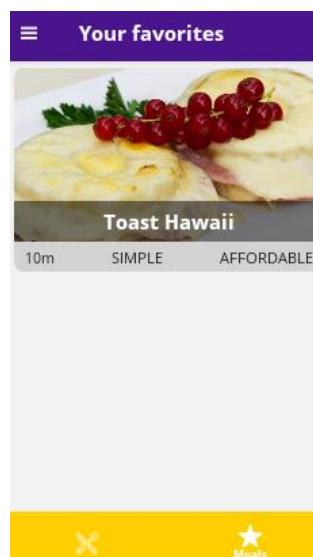
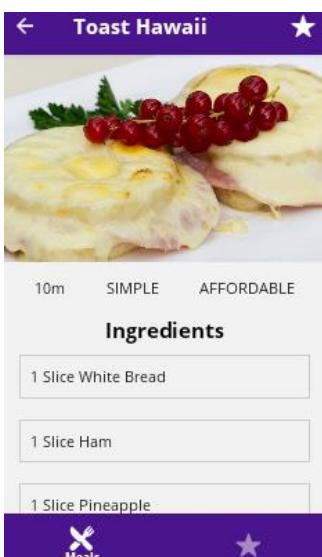
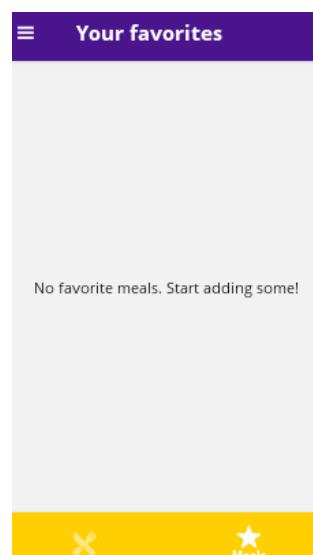
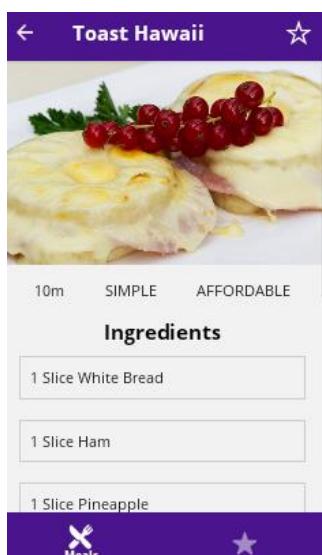
Favorites.navigationOptions = ({ navigation }) => ({

```

headerTitle: "Your favorites",
headerLeft: () => (
  <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
    <Item title="menu" iconName="ios-menu"
      onPress={navigation.toggleDrawer}
    />
  </HeaderButtons>
)
}

const _styles = StyleSheet.create({
  container: { flex: 1, alignItems: "center", justifyContent: "center" }
})

```



Adding filtering logic

State/actions/meals.js

```
export const TOGGLE_FAVORITE = "TOGGLE_FAVORITE"
export const SET_FILTERS = "SET_FILTERS"

export const toggleFavorite =
(mealId) => ({ type: TOGGLE_FAVORITE, mealId })
export const setFilters =
(filters) => ({ type: SET_FILTERS, filters })
```

State/reducer/meal.js

```
import { MEALS } from "../../data/dummy"
import { SET_FILTERS, TOGGLE_FAVORITE } from "../actions/meals"

const initialState =
{ meals: MEALS, filteredMeals: MEALS, favoriteMeals: [] }

const mealsReducer = (state = initialState, action) => {
switch (action.type) {
...
case SET_FILTERS:
const filters = action.filters

const filteredMeals = state.meals.filter((meal) => {
if (
(filters.isGlutenFree && !meal.isGlutenFree) ||
(filters.isLactoseFree && !meal.isLactoseFree) ||
(filters.isVegan && !meal.isVegan) ||
(filters.isVegetarian && !meal.isVegetarian)
) {
return false
}
return true
})

return { ...state, filteredMeals }

default: return state
}
```

```
}
```

```
export default mealsReducer
```

screens/FiltersScreen.jsx

```
...

import { useDispatch } from "react-redux"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import CustomHeaderButton from "../components/HeaderButton"
import { setFilters } from "../store/actions/meals"

export default function Filters({ navigation }) {
  const [isGluttenFree, setIsGluttenFree] = useState(false)
  const [isLactoseFree, setIsLactoseFree] = useState(false)
  const [isVegan, setIsVegan] = useState(false)
  const [isVegetarian, setIsVegetarian] = useState(false)
  const dispatch = useDispatch()

  const saveFilters = useCallback(() => {
    const appliedFilters = {
      isGluttenFree, isLactoseFree, isVegan, isVegetarian
    }
    dispatch(setFilters(appliedFilters))
  }, [isGluttenFree, isLactoseFree, isVegan, isVegetarian, dispatch])

  useEffect(() => {
    // merges navigation param values for current screen
    navigation.setParams({ save: saveFilters })
    // no navigation on dependencies! It always reconstructs
  }, [saveFilters])

  return (
    <View style={_styles.container}>
      <Text style={_styles.title}>Filters</Text>
      <FilterSwitch title="Gluten-free" value={isGluttenFree}
        onValueChange={setIsGluttenFree} />
      <FilterSwitch title="Lactose-free" value={isLactoseFree}
        onValueChange={setIsLactoseFree} />
      <FilterSwitch title="Vegan" value={isVegan}
        onValueChange={setIsVegan} />
    
```

```

        <FilterSwitch title="Vegetarian" value={isVegetarian}
            onValueChange={setIsVegetarian} />
    </View>
)
}

Filters.navigationOptions = ({ navigation }) => ({
    headerTitle: "Filter meals",
    headerLeft: () => (
        <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
            <Item title="menu" iconName="ios-menu"
                onPress={navigation.toggleDrawer} />
        </HeaderButtons>
),
    headerRight: () => (
        <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
            <Item title="save" iconName="ios-save"
                onPress={navigation.getParam("save")} />
        </HeaderButtons>
)
)
})

const _styles = StyleSheet.create({
    container: { flex: 1, alignItems: "center" },
    title: {
        fontFamily: "open-sans-bold", fontSize: 20,
        margin: 20, textAlign: "center"
    },
    filterContainer: {
        flexDirection: "row", justifyContent: "space-between",
        alignItems: "center", width: "80%", marginVertical: 15
    }
})

const FilterSwitch = ({ title, value, onValueChange }) => {
    return (
        <View style={_styles.filterContainer}>
            <Text>{title}</Text>
            <Switch
                trackColor={{

```

```

        true: Platform.OS === "android" ? colors.PRIMARY : "",
        false: ""
    } }
    thumbColor={Platform.OS === "android" ? colors.PRIMARY : ""}
    value={value}
    onValueChange={onValueChange}
/>
</View>
)
}

```

Screens/CategoryMealsScreens.jsx

```

...
import { useSelector } from "react-redux"
import DefaultText from "../components/DefaultText"

export default function CategoryMeals(props) {
    const categoryId = props.navigation.getParam("categoryId")

    const availableMeals =
        useSelector((state) => state.meals.filteredMeals)

    const displayedMeals = availableMeals.filter(
        (meal) => meal.categoryIds.indexOf(categoryId) !== -1
    )

    return !displayedMeals.length ? (
        <View style={_styles.container}>
            <DefaultText>Nothing to show. Checked filters?</DefaultText>
        </View>
    ) : (
        <MealList data={displayedMeals} navigation={props.navigation} />
    )
}

CategoryMeals.navigationOptions = ({ navigation }) => {
    const categoryId = navigation.getParam("categoryId")

    const selectedCategory =
        CATEGORIES.find((cat) => cat.id === categoryId)

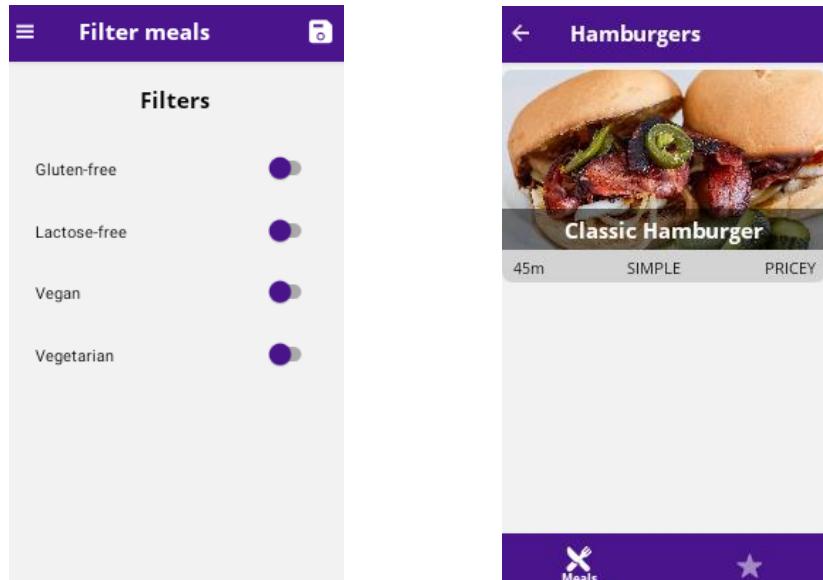
```

```

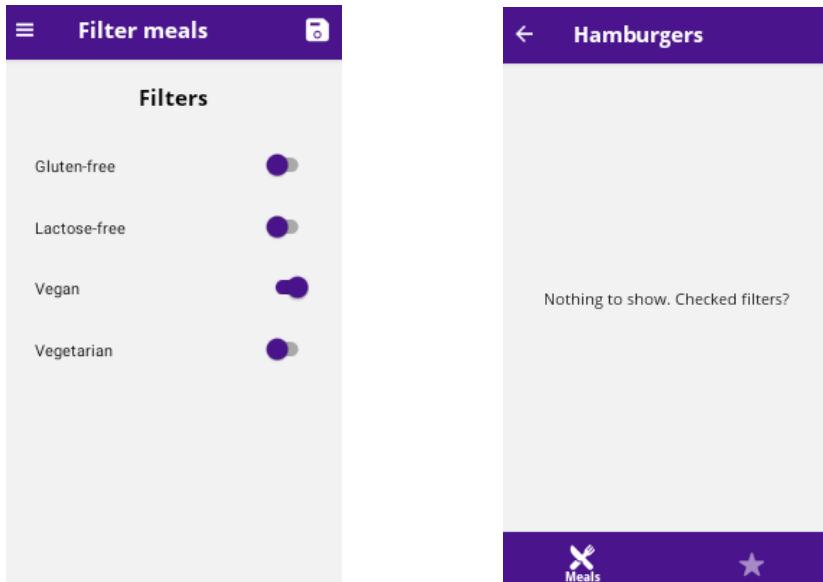
        return { headerTitle: selectedCategory.title }
    }

const _styles = StyleSheet.create({
    container: { flex: 1, alignItems: "center", justifyContent: "center" }
})

```



Change filter and tap the save icon



Debugging Redux in React Native

You can debug Redux in React Native apps with help of the React Native Debugger tool: <https://github.com/jhen0409/react-native-debugger/blob/master/docs/redux-devtools-integration.md>

- 1) Make sure you got the React Native Debugger installed (<https://github.com/jhen0409/react-native-debugger>)
- 2) Enable JS Debugging in the running app (open development overlay via **CTRL + M** / **CMD + M** on Android devices, **CMD + D** on iOS devices)
- 3) Install the **redux-devtools-extension** package via


```
$ npm install --save-dev redux-devtools-extension
```

<https://www.npmjs.com/package/redux-devtools-extension>
- 4) Enable Redux debugging in your code:

```
import { createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';

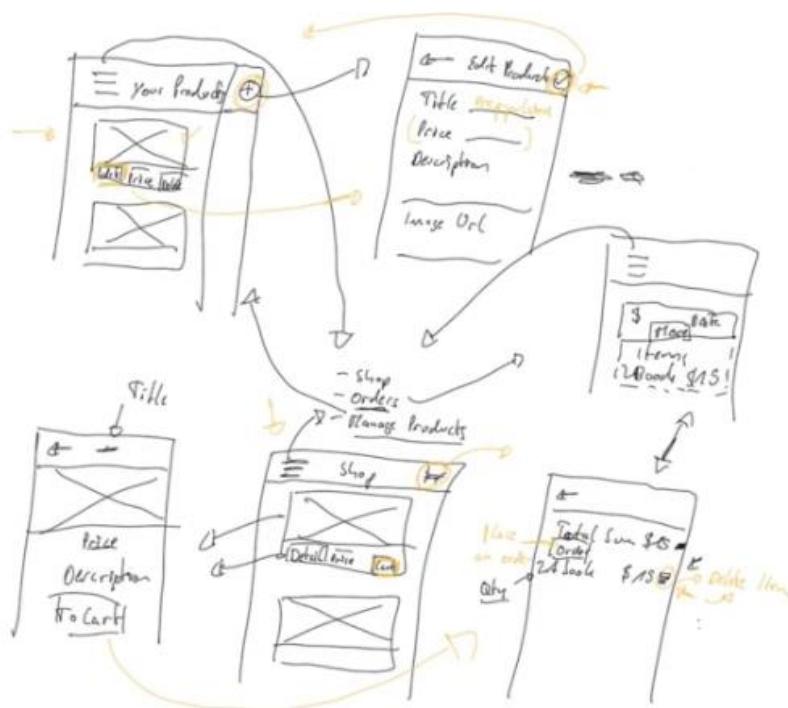
const store = createStore(reducer, composeWithDevTools());
```

Important:

Make sure you remove this code when building your app for production!

The Shop App

Intro



Create the basic setup

Start with the scaffold.

```
$ expo init shopping-app

$ expo install react react-redux react-navigation react-
navigation-buttons react-native-reanimated
react-native-reanimated react-native-gesture-handler
```

App.js

```
import React from "react"
import { createStore, combineReducers } from "redux"
import { Provider } from "react-redux"

import ShopNavigator from "./navigation/Shop"
import productsReducer from "./store/reducers/products"

// reducers are empty for now
const rootReducer = combineReducers({ products: productsReducer })
const store = createStore(rootReducer)

export default function App() {
  return (
    <Provider store={store}><ShopNavigator /></Provider>
  )
}
```

Navigation/Shop.jsx

```
import { Platform } from "react-native"
import { createAppContainer } from "react-navigation"
import { createStackNavigator } from "react-navigation-stack"
import ProductsOverviewScreen from "../screens/shop/ProductsOverview"
import colors from "../constants/colors"

const ProductsNavigator = createStackNavigator(
  { ProductsOverview: ProductsOverviewScreen },
  {
    defaultNavigationOptions: {
      headerStyle: {
        backgroundColor: colors.header,
      },
      headerTitleStyle: {
        color: "white",
      },
      headerTintColor: "white",
    },
  },
)
```

```
        headerStyle: {
          backgroundColor:
            Platform.OS === "android" ? colors.PRIMARY : ""
        },
        headerTintColor:
          Platform.OS === "android" ? "white" : colors.PRIMARY
      }
    }
  )
}

export default createAppContainer(ProductsNavigator)
```

screens/ProductsOverview.jsx

```
import React from "react"
import { FlatList } from "react-native"
import { useSelector } from "react-redux"
import ProductItem from "../../components/shop/ProductItem"

export default function ProductsOverview(props) {
  const availableProducts = useSelector(
    (state) => state.products.availableProducts
  )

  return (
    <FlatList
      data={availableProducts}
      // keyExtractor={(item) => item.id} // not on newer versions!
      renderItem={({ item }) => (
        <ProductItem
          image={item.imageUrl} title={item.title}
          price={item.price} onViewDetails={() => {}}
          onAddToCart={() => {}}
        />
      )}
    />
  )
}

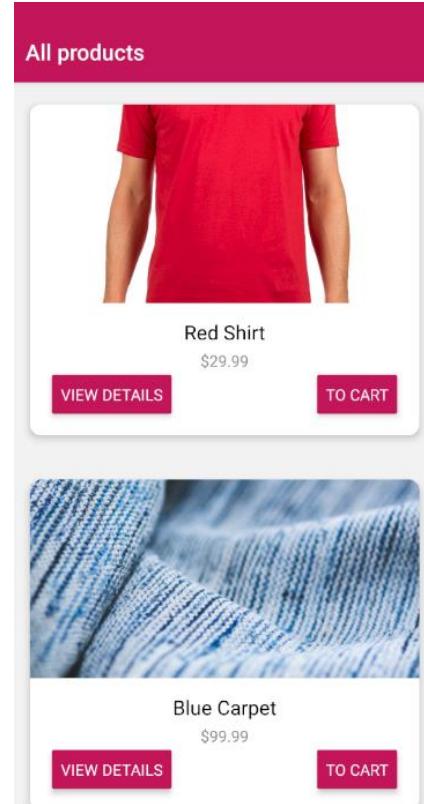
ProductsOverview.navigationOptions = { headerTitle: "All products" }
```

components/shop/ProductItem.jsx

```
import React from "react"
import { View, Text, Image, StyleSheet, Button } from "react-native"
import colors from "../../constants/colors"

export default function ProductItem(props) {
  return (
    <View style={_styles.container}>
      <View style={_styles.imageContainer}>
        <Image style={_styles.image} source={{ uri: props.image }} />
      </View>
      <View style={_styles.textsContainer}>
        <Text style={_styles.title}>{props.title}</Text>
        <Text style={_styles.price}>$ {props.price.toFixed(2)}</Text>
      </View>
      <View style={_styles.buttonsContainer}>
        <Button
          color={colors.PRIMARY} title="View details"
          onPress={props.onViewDetails}
        />
        <Button
          color={colors.PRIMARY} title="To cart"
          onPress={props.onAddToCart}
        />
      </View>
    </View>
  )
}

const _styles = StyleSheet.create({
  container: {
    height: 300, margin: 20,
    borderRadius: 10,
    backgroundColor: "white",
    elevation: 5, // android
    shadowColor: "black", // iOS
    shadowOpacity: 0.26,
    shadowOffset: { width: 0, height: 2 },
    shadowRadius: 8
  },
})
```



```

imageContainer: {
  width: "100%", height: "60%",
  borderTopLeftRadius: 10, borderTopRightRadius: 10,
  overflow: "hidden"
},
image: { width: "100%", height: "100%" },
textsContainer: { alignItems: "center", height: "15%", padding: 10
},
title: { fontSize: 18, marginVertical: 4 },
price: { fontSize: 14, color: "#888" },
buttonsContainer: {
  flexDirection: "row", justifyContent: "space-between",
  alignItems: "center", height: "25%", paddingHorizontal: 20
}
})

```

models/product.js

```

export class Product {
  constructor(id, ownerId, title, imageUrl, description, price) {
    this.id = id; this.ownerId = ownerId
    this.title = title; this.imageUrl = imageUrl
    this.description = description; this.price = price
  }
}

```

Store/shop/reducer.js

```

import PRODUCTS from "../../data/dummy"

const initialState = {
  availableProducts: PRODUCTS,
  userProducts: PRODUCTS.filter((p) => p.ownerId === "u1") }

const productsReducer = (state = initialState, action) => {
  switch (action.type) {
    default: return state
  }
}

export default productsReducer

```

Adding touchable components

App.jsx

```
import React, { useState, useEffect } from "react"
import { ActivityIndicator, View, StyleSheet } from "react-native"
import { createStore, combineReducers } from "redux"
import { Provider } from "react-redux"
import ShopNavigator from "./navigation/Shop"
import productsReducer from "./store/reducers/products"
import * as Font from "expo-font"

const rootReducer = combineReducers({ products: productsReducer })
const store = createStore(rootReducer)

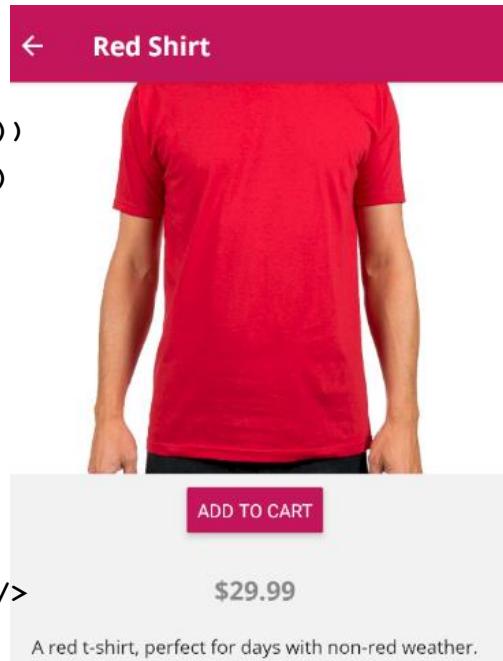
export default function App() {
  const [isFontLoaded, setIsFontLoaded] = useState(false)

  useEffect(() => {
    fetchFonts()
      .then(() => setIsFontLoaded(true))
      .catch((err) => console.log(err))
  }, [])

  return isFontLoaded ? (
    <Provider store={store}>
      <ShopNavigator />
    </Provider>
  ) : (
    <View style={_styles.container}>
      <ActivityIndicator size="large" />
    </View>
  )
}

const _styles = StyleSheet.create({
  container: { flex: 1, alignItems: "center", justifyContent: "center" }
})

function fetchFonts() {
  return Font.loadAsync({
```



```

    "open-sans": require("./assets/fonts/OpenSans-Regular.ttf"),
    "open-sans-bold": require("./assets/fonts/OpenSans-Bold.ttf")
  })
}

Navigator/Shop.jsx

...
import ProductDetailsScreen from "../screens/shop/ProductDetails"

const ProductsNavigator = createStackNavigator(
{
  ProductsOverview: ProductsOverviewScreen,
  ProductDetails: ProductDetailsScreen
},
{
  initialRouteName: "ProductsOverview",
  defaultNavigationOptions: {
    headerStyle: {
      backgroundColor:
        Platform.OS === "android" ? colors.PRIMARY : ""
    },
    headerTitleStyle: { fontFamily: "open-sans-bold" },
    headerBackTitleStyle: { fontFamily: "open-sans" },
    headerTintColor:
      Platform.OS === "android" ? "white" : colors.PRIMARY
  }
}
)

export default createAppContainer(ProductsNavigator)

```

Screens/ProductsOverview.jsx

```

...
export default function ProductsOverview(props) {
  const availableProducts = useSelector(
    (state) => state.products.availableProducts
  )
  return (
    <FlatList
      data={availableProducts}

```

```

renderItem={({ item }) => (
  <ProductItem
    image={item.imageUrl} title={item.title}
    price={item.price} onAddToCart={() => {}}
    onViewDetails={() =>
      props.navigation.navigate("ProductDetails", {
        productId: item.id,
        productTitle: item.title
      })
    }
  />
) }
/>
)
}

```

```
ProductsOverview.navigationOptions = { headerTitle: "All products" }
```

Screens/ProductsOverview.jsx

```

import React from "react"
import { View, Text, Image, StyleSheet, Button, TouchableOpacity,
  TouchableNativeFeedback, Platform } from "react-native"
import colors from "../../constants/colors"

export default function ProductItem(props) {
  const TouchableComponent =
    Platform.OS === "android" && Platform.Version >= 21
    ? TouchableNativeFeedback
    : TouchableOpacity

  return (
    <View style={_styles.container}>
      {/* cannot have Touchable as single view child */}
      <View style={_styles.touchableContainer}>
        {/* `useForeground` enables ripple effect over all elements */}
        <TouchableComponent
          onPress={props.onViewDetails} useForeground>
          {/* cannot have multiple elements as Touchable children */}
          <View>
            <View style={_styles.imageContainer}>

```

```

        <Image style={_styles.image}
            source={{ uri: props.image }} />
        </View>
        <View style={_styles.textsContainer}>
            <Text style={_styles.title}>{props.title}</Text>
            <Text style={_styles.price}>
                ${props.price.toFixed(2)}</Text>
        </View>
        <View style={_styles.buttonsContainer}>
            <Button color={colors.PRIMARY} title="View details"
                onPress={props.onViewDetails} />
            <Button color={colors.PRIMARY} title="To cart"
                onPress={props.onAddToCart} />
        </View>
    </View>
</TouchableComponent>
</View>
</View>
)
}

const _styles = StyleSheet.create({
    container: {
        height: 300, margin: 20, borderRadius: 10,
        backgroundColor: "white",
        elevation: 5, // android
        shadowColor: "black", shadowRadius: 8, // iOS
        shadowOpacity: 0.26, shadowOffset: { width: 0, height: 2 },
    },
    touchableContainer: { overflow: "hidden", borderRadius: 10 },
    imageContainer: {
        width: "100%", height: "60%", overflow: "hidden"
        borderTopLeftRadius: 10, borderTopRightRadius: 10,
    },
    image: { width: "100%", height: "100%" },
    textsContainer: { alignItems: "center", height: "15%", padding: 10 },
    title: {
        fontSize: 18, marginVertical: 2, fontFamily: "open-sans-bold"
    },
    price: { fontSize: 14, color: "#888", fontFamily: "open-sans" },
})

```

```

        buttonsContainer: {
            flexDirection: "row", justifyContent: "space-between",
            alignItems: "center", height: "25%", paddingHorizontal: 20
        }
    })
}

```

Adding items to cart

Install redux devtools extension to see redux running in debugger.

```
$ npm install --save-dev redux-devtools-extension
```

App.jsx

```

...
import { composeWithDevTools } from "redux-devtools-extension"
import productsReducer from "./store/reducers/products"
import cartReducer from "./store/reducers/cart"
...
const rootReducer = combineReducers({
    products: productsReducer, cart: cartReducer
})
// remember to remove `composeWithDevtools` before deploying!
const store = createStore(rootReducer, composeWithDevTools())
export default function App() { ... }

```

Models/cart.js

```

export default class CartItem {
    constructor(quantity, price, title, sum) {
        this.quantity = quantity
        this.price = price
        this.title = title
        this.sum = sum
    }
}

```

Store/actions/cart.js

```

export const ADD_TO_CART = "ADD_TO_CART"
export const addToCart = (product) => ({ type: ADD_TO_CART, product })

```

```

store/reducers/cart.js

import { ADD_TO_CART } from "../actions/cart"
import CartItem from "../../models/cart"

const initialState = { items: {}, total: 0 }

const cartReducer = (state = initialState, action) => {
  switch (action.type) {
    case ADD_TO_CART:
      const { id, price, title } = action.product
      let updatedOrNewCartItem = {}

      if (state.items[id]) {
        updatedOrNewCartItem = new CartItem(
          state.items[id].quantity + 1, price,
          title, state.items[id].sum + price
        )
      } else {
        updatedOrNewCartItem = new CartItem(1, price, title, price)
      }
      return {
        ...state,
        items: { ...state.items, [id]: updatedOrNewCartItem },
        total: state.total + price
      }
    default:
      return state
  }
}

export default cartReducer

```

Screens/ProductOverview.jsx

```

...
import { useSelector, useDispatch } from "react-redux"
import * as cartActions from "../../store/actions/cart"

export default function ProductsOverview(props) {
  const availableProducts = useSelector(

```

```

        (state) => state.products.availableProducts
    )
const dispatch = useDispatch()

return (
    <FlatList
        data={availableProducts}
        renderItem={({ item }) => (
            <ProductItem image={item.imageUrl} title={item.title}
                price={item.price}
                onViewDetails={() =>
                    props.navigation.navigate("ProductDetails", {
                        productId: item.id, productTitle: item.title
                    })
                }
            onAddToCart={() => dispatch(cartActions.addToCart(item))}
        />
    ) }
    />
)
}

```

```
ProductsOverview.navigationOptions = { headerTitle: "All products" }
```

Screens/ProductDetails.jsx

```

...
import { useSelector, useDispatch } from "react-redux"
import * as cartActions from "../../store/actions/cart"

export default function ProductDetails(props) {
    const productId = props.navigation.getParam("productId")
    const selectedProduct = useSelector((state) =>
        state.products.availableProducts.find(
            (prod) => prod.id === productId))
const dispatch = useDispatch()
return (
    <ScrollView>
        <Image source={
            { uri: selectedProduct.imageUrl } } style={_styles.image} />

```

```

        <View style={_styles.buttonContainer}>
          <Button color={colors.PRIMARY} title="Add to cart"
            onPress={() =>
              dispatch(cartActions.addToCart(selectedProduct))
            }
          />
        </View>
        <Text style={_styles.price}>
          ${selectedProduct.price.toFixed(2)}</Text>
        <Text style={_styles.description}>
          {selectedProduct.description}</Text>
      </ScrollView>
    )
}

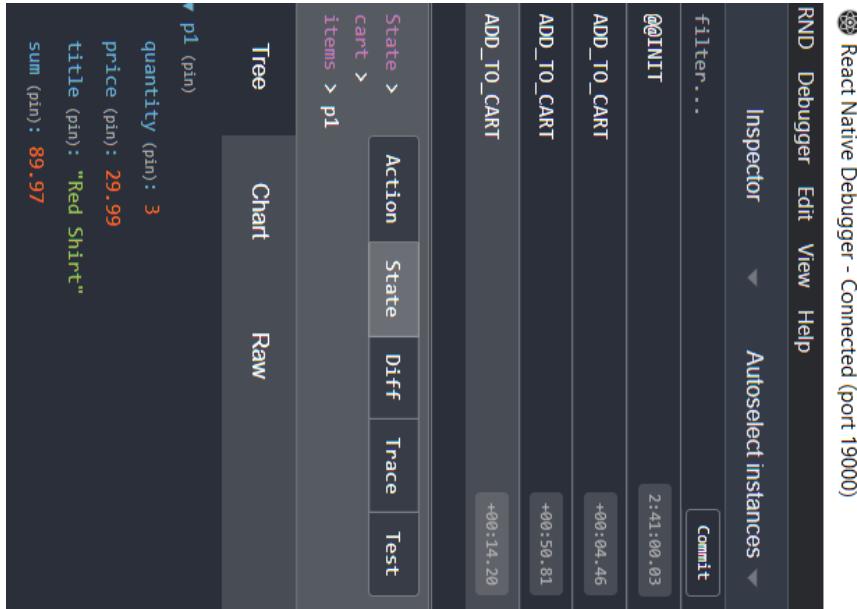
```

ProductDetails.navigationOptions = ({ navigation }) => ({

```

  headerTitle: navigation.getParam("productTitle")
})
```

...



Implementing Header buttons and remove from cart logic

UI/CustomHeaderButton.jsx

```

import { Platform } from "react-native"
import React from "react"

```

```

import { HeaderButton } from "react-navigation-header-buttons"
import { IonIcons } from "@expo/vector-icons"
import colors from "../constants/colors"

export default function CustomHeaderButton(props) {
  return (
    <HeaderButton
      {...props} IconComponent={IonIcons} iconSize={23}
      color={Platform.OS === "android" ? "white" : colors.PRIMARY}
    />
  )
}

```

Navigation/Shop.jsx

```

...
import CartScreen from "../screens/shop/Cart"

const ProductsNavigator = createStackNavigator(
  {
    ProductsOverview: ProductsOverviewScreen,
    ProductDetails: ProductDetailsScreen,
    Cart: CartScreen
  },
  {
    initialRouteName: "ProductsOverview",
    defaultNavigationOptions: {
      headerStyle: {
        backgroundColor:
          Platform.OS === "android" ? colors.PRIMARY : ""
      },
      headerTitleStyle: { fontFamily: "open-sans-bold" },
      headerBackTitleStyle: { fontFamily: "open-sans" },
      headerTintColor:
        Platform.OS === "android" ? "white" : colors.PRIMARY
    }
  }
)

export default createAppContainer(ProductsNavigator)

```

```

screens/shop/ProductsOverview.jsx

...
export default function ProductsOverview(props) {
...
return <FlatList ... />
}

ProductsOverview.navigationOptions = ({ navigation }) => ({
headerTitle: "All products",
headerRight: () => (
// needs `npm i --save @react-navigation/native`!
<HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
<Item
  title="cart"
  iconName={Platform.OS === "android" ? "md-cart" : "ios-cart"}
  onPress={() => navigation.navigate("Cart")}
/>
</HeaderButtons>
)
})
}

```

Screens/Cart.jsx

```

import React from "react"
import { View, Text, FlatList, Button, StyleSheet } from "react-native"
import { useDispatch, useSelector } from "react-redux"
import CartItem from "../../components/shop/CartItem"
import * as cartActions from "../../store/actions/cart"
import * as sharedStyles from "../../constants/styles"
import colors from "../../constants/colors"

export default function CartScreen(props) {
const total = useSelector((state) => state.cart.total)
const dispatch = useDispatch()

const items = useSelector((state) => {
const transformedCartItems = []
for (const key in state.cart.items) {
const currentItem = state.cart.items[key]
transformedCartItems.push({

```

```

        id: key, title: currentItem.title,
        price: currentItem.price, quantity: currentItem.quantity,
        sum: currentItem.sum
    })
}

//  always keep the items order in array
return transformedCartItems.sort((a, b) => (a.id > b.id ? 1 : -1))
})

return (
<View style={_styles.container}>
    <View style={_styles.summary}>
        <Text style={_styles.summaryText}>
            Total:{" "}
            <Text style={_styles.amount}>${total.toFixed(2)}</Text>
        </Text>
        <Button
            color={colors.SECONDARY} title="Order"
            disabled={items.length === 0} onPress={null}
        />
    </View>
    <FlatList
        data={items}
        keyExtractor={({ id }) => id}
        renderItem={({ item: { id, quantity, title, sum } }) => (
            <CartItem
                quantity={quantity} title={title} amount={sum}
                onDelete={() => dispatch(cartActions.removeFromCart(id))}
            />
        )}
    />
</View>
)
}

const _styles = StyleSheet.create({
    container: { margin: 20 },
    summary: {
        flexDirection: "row", alignItems: "center",
        justifyContent: "space-around",
    }
})

```

```

        marginBottom: 20, marginHorizontal: 10,
        padding: 10, ...sharedStyles.CARD_SHADOW
    },
    summaryText: { fontFamily: "open-sans-bold", fontSize: 18 },
    amount: { color: colors.PRIMARY }
)

```

Constants/styles.js

```

export const cardShadow = {
    borderRadius: 10,
    backgroundColor: "white",
    elevation: 5, // android
    shadowColor: "black", // ios
    shadowOpacity: 0.26,
    shadowOffset: { width: 0, height: 2 },
    shadowRadius: 8
}

```

Components/CartItem.jsx

```

import React from "react"
import { View, Text, StyleSheet, Platform } from "react-native"
import { Ionicons } from "@expo/vector-icons"
import TouchableComponent from "../../UI/TouchableComponent"

export default function CartItem(
    { quantity, title, amount, onDelete }
) {
    return (
        <View style={_styles.container}>
            <View style={_styles.itemData}>
                <Text style={_styles.quantity}>{quantity}</Text>
                <Text style={_styles.mainText}>{title}</Text>
            </View>
            <View style={_styles.itemData}>
                <Text style={_styles.mainText}>
                    ${amount.toFixed(2)}</Text>
                <TouchableComponent
                    onPress={onDelete} style={_styles.deleteButton}>
                    <Ionicons

```

```

        name={

            Platform.OS === "android" ? "md-trash" : "ios-trash"
        size={23} color="red"

        />
    </TouchableComponent>
</View>
</View>

)

}

const _styles = StyleSheet.create({
    container: {

        padding: 10, backgroundColor: "white", flexDirection: "row",
        justifyContent: "space-between", marginHorizontal: 20,
        paddingHorizontal: 20
    },
    itemData: { flexDirection: "row", alignItems: "center" },
    quantity: { fontFamily: "open-sans", color: "#888", fontSize: 16 },
    mainText: {

        fontFamily: "open-sans-bold", fontSize: 16, marginHorizontal: 10
    },
    deleteButton: { marginLeft: 20 }
})

```

UI/TouchableComponent.jsx

```

import { Platform, TouchableNativeFeedback, TouchableOpacity
} from "react-native"

export default function TouchableComponent(props) {
    return Platform.OS === "android" && Platform.Version >= 21 ? (
        <TouchableNativeFeedback {...props} />
    ) : (
        <TouchableOpacity {...props} />
    )
}

```

Store/actions/cart.js

```
export const ADD_TO_CART = "ADD_TO_CART"
```

```
export const REMOVE_FROM_CART = "REMOVE_FROM_CART"
export const addToCart = (product) => ({ type: ADD_TO_CART, product })
export const removeFromCart = (id) => ({ type: REMOVE_FROM_CART, id })

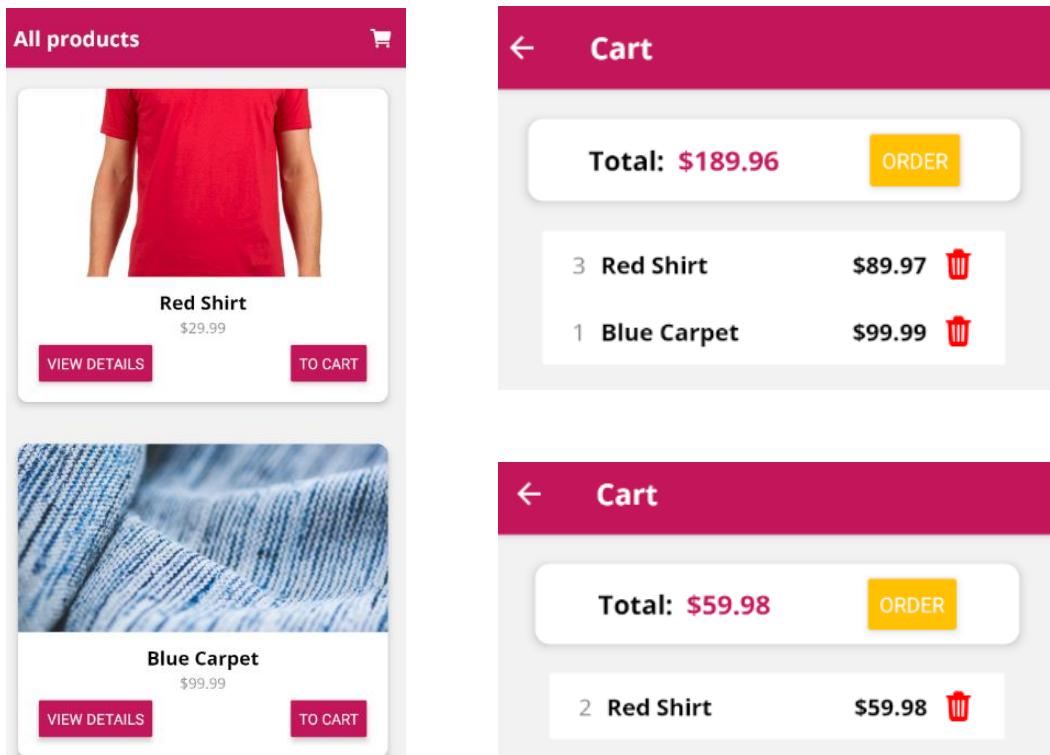
Store/reducer/cart.js
import { ADD_TO_CART, REMOVE_FROM_CART } from "../actions/cart"
import CartItem from "../../models/cart"

const initialState = { items: {}, total: 0 }

const cartReducer = (state = initialState, action) => {
  switch (action.type) {
    case ADD_TO_CART: ...
    case REMOVE_FROM_CART:
      const selectedItem = state.items[action.id]
      let updatedItems

      if (selectedItem.quantity > 1) {
        const { price, title, sum } = selectedItem
        updatedItems = {
          ...state.items,
          [action.id]: new CartItem(
            selectedItem.quantity - 1, price, title, sum - price
          )
        }
      } else {
        updatedItems = { ...state.items }
        delete updatedItems[action.id]
      }
      return {
        ...state,
        items: updatedItems,
        total: state.total - selectedItem.price
      }
    default: return state
  }
}

export default cartReducer
```



Adding redux logic for orders

Models/orders.js

```
export default class Order {
  constructor(id, items, total, date) {
    this.id = id
    this.items = items
    this.total = total
    this.date = date
  }

  get stringDate() {
    return this.date.toLocaleDateString("en-EN", {
      year: "numeric",
      month: "long",
      day: "numeric",
      hour: "2-digit",
      minute: "2-digit"
    })
  }
}
```

Store/actions/orders.js

```
export const ADD_ORDER = "ADD_ORDER"

export const addOrder = (items, total) => ({
  type: ADD_ORDER, payload: { items, total }
})
```

Store/reducers/orders.js

```
import Order from "../../models/Order"
import { ADD_ORDER } from "../actions/orders"

const initialState = { items: [] }

export default function ordersReducer(state = initialState, action) {
  switch (action.type) {
    case ADD_ORDER:
      const newOrder = new Order(
        new Date().toString(), action.payload.items,
        action.payload.total, new Date()
      )
      return { ...state, items: state.items.concat(newOrder) }

    default:
      return state
  }
}
```

Store/reducers/cart.js

```
import { ADD_TO_CART, REMOVE_FROM_CART } from "../actions/cart"
import { ADD_ORDER } from "../actions/orders"
import CartItem from "../../models/cart"

const initialState = { items: {}, total: 0 }

const cartReducer = (state = initialState, action) => {
  switch (action.type) {
    case ADD_TO_CART: ...
    case REMOVE_FROM_CART:...
```

```
        case ADD_ORDER:
            return initialState // clear cart on add order
        default:
            return state
    }
}

export default cartReducer
```

App.js

```
...
import ordersReducer from "./store/reducers/orders"

const rootReducer = combineReducers({
    products: productsReducer, cart: cartReducer, orders: ordersReducer
})

const store = createStore(rootReducer, composeWithDevTools())
...
```

Screens/Cart.jsx

```
...
import * as orderActions from "../../store/actions/orders"

export default function CartScreen(props) {
    const total = useSelector((state) => state.cart.total)
    const dispatch = useDispatch()

    const items = useSelector((state) => {
        const transformedCartItems = []
        for (const key in state.cart.items) {
            const currentItem = state.cart.items[key]
            transformedCartItems.push({
                id: key, title: currentItem.title, price: currentItem.price,
                quantity: currentItem.quantity, sum: currentItem.sum
            })
        }
        return transformedCartItems.sort((a, b) => (a.id > b.id ? 1 : -1))
    })
}
```

```

        return (
            <View style={_styles.container}>
                <View style={_styles.summary}>
                    <Text style={_styles.summaryText}>
                        Total:{" "}
                        <Text style={_styles.amount}>${total.toFixed(2)}</Text>
                    </Text>
                    <Button disabled={items.length === 0}
                        color={colors.SECONDARY} title="Order"
                        onPress={
                            () => dispatch(orderActions.addOrder(items, total))
                        }
                    />
                </View>
                <FlatList data={items} keyExtractor={({ id }) => id}
                    renderItem={({ item: { id, quantity, title, sum } }) => (
                        <CartItem quantity={quantity} title={title} amount={sum}
                        onDelete={() => dispatch(cartActions.removeFromCart(id))}
                    )
                ) }
            />
        </View>
    )
}

```

Note: To make drawer works, we need a couple of packages:

```
$ npm install --save react-navigation-drawer react-native-screens react-native-gesture-handler react-native-reanimated
```

navigator/Shop.js

```

import React from "react"
import { Platform, SafeAreaView, StatusBar } from "react-native"
import { createAppContainer } from "react-navigation"
import { createStackNavigator } from "react-navigation-stack"
import { createDrawerNavigator, DrawerItems } from "react-navigation-drawer"
import { Ionicons } from "@expo/vector-icons"
import ProductsOverviewScreen from "../screens/shop/ProductsOverview"
import ProductDetailsScreen from "../screens/shop/ProductDetails"
import OrdersScreen from "../screens/shop/Orders"
import CartScreen from "../screens/shop/Cart"
import colors from "../constants/colors"

```

```

// this setup only overrides `defaultNavigatorOptions` when the active
// screen is used as the top screen of another stack screen
const createSharedNavOptions = (iconName) => (
  // drawerConfigs comes from DrawerNavigator package
  drawerIcon: (drawerConfigs) => (
    <Ionicons
      name={Platform.OS === "android"
        ? `md-${iconName}`
        : `ios-${iconName}`}
      size={23} color={drawerConfigs.tintColor}
    />
  )
)

const sharedDefaultNavOptions = {
  headerStyle: {
    backgroundColor: Platform.OS === "android" ? colors.PRIMARY : "",
  },
  headerTitleStyle: { fontFamily: "open-sans-bold" },
  headerBackTitleStyle: { fontFamily: "open-sans" },
  headerTintColor:
    Platform.OS === "android" ? "white" : colors.PRIMARY
}

const ProductsNavigator = createStackNavigator(
{
  ProductsOverview: ProductsOverviewScreen,
  ProductDetails: ProductDetailsScreen,
  Cart: CartScreen
},
{
  initialRouteName: "ProductsOverview",
  navigationOptions: createSharedNavOptions("cart"),
  defaultNavigationOptions: sharedDefaultNavOptions
}
)

const OrdersNavigator = createStackNavigator(
{ Orders: OrdersScreen },
{

```

```

        navigationOptions: createSharedNavOptions("list"),
        defaultNavigationOptions: sharedDefaultNavOptions
    }
)

const ShopNavigator = createDrawerNavigator(
    { Products: ProductsNavigator, Orders: OrdersNavigator },
    {
        contentOptions: { activeTintColor: colors.PRIMARY },
        contentComponent: (props) => (
            <SafeAreaView
                style={{ flex: 1, paddingTop: StatusBar.currentHeight }}>
                <DrawerItems {...props} />
            </SafeAreaView>
        )
    }
)

export default createAppContainer(ShopNavigator)

```

Screens/Order.jsx

```

import React from "react"
import { FlatList, Platform, Text } from "react-native"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import { useSelector } from "react-redux"
import OrderItem from "../../components/shop/OrderItem"
import CustomHeaderButton from "../../UI/CustomHeaderButton"

export default function Orders(props) {
    const itemsInCart = useSelector((state) => state.orders.items)

    return (
        <FlatList
            data={itemsInCart}
            renderItem={({ item }) => (
                <OrderItem
                    total={item.total} date={item.stringDate} items={item.items}
                />
            )}
        )
    )
}

```

```

        />
    )
}

Orders.navigationOptions = ({ navigation }) => ({
  headerTitle: "Your orders",
  headerLeft: () => (
    <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
      <Item
        title="menu" onPress={navigation.toggleDrawer}
        iconName={Platform.OS === "android" ? "md-menu" : "ios-menu"}
      />
    </HeaderButtons>
  )
})

```

Note: remember that shape of items in `state.orders.items` array are "transformed" `CartItem` objects, as converted in `screens/shop/Cart.jsx`:

```

...
const items = useSelector((state) => {
  const transformedCartItems = []
  for (const key in state.cart.items) {
    const currentItem = state.cart.items[key]
    transformedCartItems.push({
      id: key, title: currentItem.title, price: currentItem.price,
      quantity: currentItem.quantity, sum: currentItem.sum
    })
  }
  return transformedCartItems.sort((a, b) => (a.id > b.id ? 1 : -1))
})
...

```

Components/OrderItem.jsx

```

import React, { useState } from "react"
import { View, Text, Button, StyleSheet } from "react-native"
import CartItem from "./CartItem"
import colors from "../../constants/colors"
import * as sharedStyles from "../../constants/styles"

```

```

export default function OrderItem({ total, date, items }) {
  const [showDetails, setShowDetails] = useState(false)

  return (
    <View style={_styles.container}>
      <Text style={_styles.date}>{date}</Text>
      <View style={_styles.totalAndButton}>
        <Text style={_styles.total}>Total: $ {total.toFixed(2)}</Text>
        <View style={_styles.detailsButton}>
          <Button
            title={showDetails ? "Hide details" : "Show details"}
            color={colors.PRIMARY}
            onPress={() => setShowDetails((st) => !st)}
          />
        </View>
      </View>
      {showDetails && (
        <View style={_styles.cartItemsContainer}>
          {items.map((item) => (
            <CartItem
              key={item.id} quantity={item.quantity}
              amount={item.sum} title={item.title}
            />
          )))
        </View>
      ) }
    </View>
  )
}

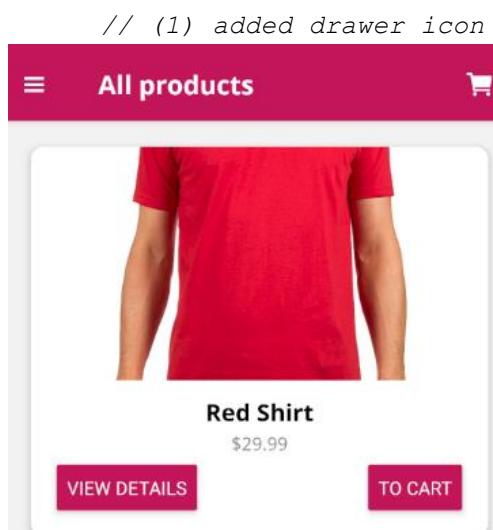
const _styles = StyleSheet.create({
  container: {
    ...sharedStyles.CARD_SHADOW, margin: 20,
    paddingHorizontal: 20, paddingVertical: 15
  },
  date: {
    marginLeft: "auto", marginBottom: 10,
    fontFamily: "open-sans", fontSize: 16, color: "#888"
  },
  totalAndButton: {

```

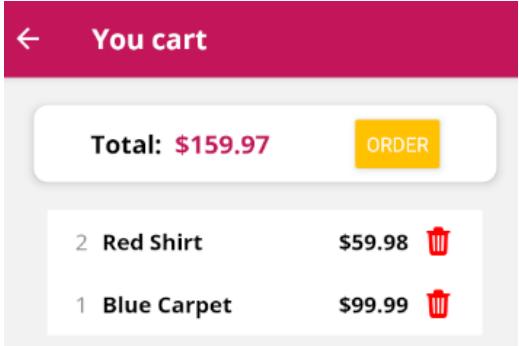
```
flexDirection: "row", justifyContent: "space-between",
alignItems: "center", width: "100%"

},
total: { fontFamily: "open-sans-bold", fontSize: 16 },
detailsButton: { width: "40%" },
cartItemsContainer: { marginTop: 5 }

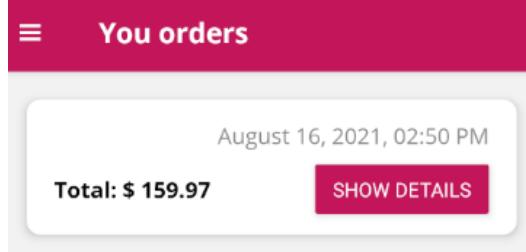
})
```



```
// (1) added drawer icon
```

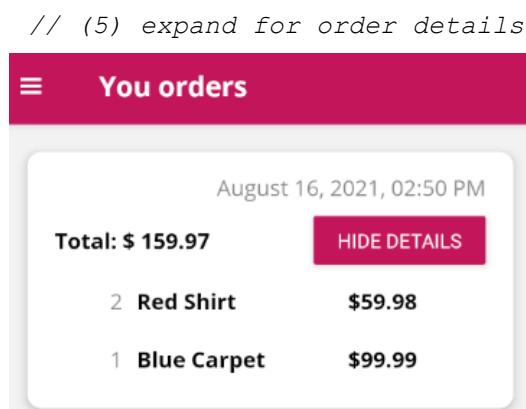


```
// (2) place and submit order
```



```
Products
```

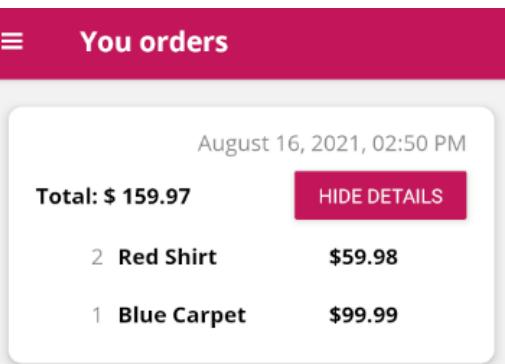
```
Orders
```



```
// (3) toggle "Orders" drawer menu
```



```
// (4) check order summary
```



```
// (5) expand for order details
```

"User products" screen

Navigation/Shop.jsx

```
...
import UserProductsScreen from "../screens/user/UserProducts"

// this setup only overrides defaultNavigator options when the active
// screen is used as the top screen of another stack screen
const createSharedNavOptions = (iconName) => ({
  drawerIcon: (drawerConfigs) => (
    <Ionicons
      name={Platform.OS === "android" ?
        `md-${iconName}` :
        `ios-${iconName}`}
      size={23}
      color={drawerConfigs.tintColor}
    />
  )
})()

const sharedDefaultNavOptions = {
  headerStyle: {
    backgroundColor: Platform.OS === "android" ? colors.PRIMARY : "",
  },
  headerTitleStyle: { fontFamily: "open-sans-bold" },
  headerBackTitleStyle: { fontFamily: "open-sans" },
  headerTintColor:
    Platform.OS === "android" ? "white" : colors.PRIMARY
}

const ProductsNavigator = createStackNavigator(
{
  ProductsOverview: ProductsOverviewScreen,
  ProductDetails: ProductDetailsScreen,
  Cart: CartScreen
},
{
  initialRouteName: "ProductsOverview",
  navigationOptions: createSharedNavOptions("cart"),
  defaultNavigationOptions: sharedDefaultNavOptions
})
```

```

        )

const OrdersNavigator = createStackNavigator(
  { Orders: OrdersScreen },
  {
    navigationOptions: createSharedNavOptions("list"),
    defaultNavigationOptions: sharedDefaultNavOptions
  }
)

const AdminNavigator = createStackNavigator(
  { UserProducts: UserProductsScreen },
  {
    navigationOptions: createSharedNavOptions("create"),
    defaultNavigationOptions: sharedDefaultNavOptions
  }
)

const ShopNavigator = createDrawerNavigator(
{
  Products: ProductsNavigator,
  Orders: OrdersNavigator,
  Admin: AdminNavigator
},
{
  contentOptions: { activeTintColor: colors.PRIMARY },
  contentComponent: (props) => (
    <SafeAreaView
      style={{ flex: 1, paddingTop: StatusBar.currentHeight }}>
      <DrawerItems {...props} />
    </SafeAreaView>
  )
}
)
}

export default createAppContainer(ShopNavigator)

```

UI/CustomHeaderButtons.jsx

```
import React from "react"
```

```

import { Platform } from "react-native"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import CustomHeaderButton from "./CustomHeaderButton"

export default function CustomHeaderButtons (
  { onPress, title, iconName }
) {
  return (
    <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
      <Item
        title={title} onPress={onPress} iconName={
          Platform.OS === "android" ?
          `md-${iconName}` : `ios-${iconName}`
        } />
    </HeaderButtons>
  )
}

```

Components/shop/ProductItem.jsx

```

...
export default function ProductItem(props) {
  return (
    <View style={_styles.container}>
      {/* cannot have Touchable as single view child */}
      <View style={_styles.touchableContainer}>
        {/* `useForeground` enables ripple effect over all elements */}
        <TouchableComponent onPress={props.onSelect} useForeground>
          {/* cannot have multiple elements as Touchable children */}
          <View>
            <View style={_styles.imageContainer}>
              <Image
                style={_styles.image} source={{ uri: props.image }} />
            </View>
            <View style={_styles.textsContainer}>
              <Text style={_styles.title}>{props.title}</Text>
              <Text style={_styles.price}>
                ${props.price.toFixed(2)}</Text>
            </View>
            <View style={_styles.buttonsContainer}>
              {props.children}</View>
            
```

```

        </View>
    </TouchableComponent>
</View>
</View>
)

}

const _styles = StyleSheet.create({...})

screens/user/userProducts.jsx
import React from "react"
import { Button, FlatList } from "react-native"
import { useDispatch, useSelector } from "react-redux"
import ProductItem from "../../components/shop/ProductItem"
import CustomHeaderButtons from "../../UI/CustomHeaderButtons"
import * as productActions from "../../store/actions/products"
import colors from "../../constants/colors"

export default function UserProducts(props) {
    const userProducts = useSelector(
        (state) => state.products.userProducts)
    const dispatch = useDispatch()

    return (
        <FlatList
            data={userProducts} keyExtractor={(item) => item.id}
            renderItem={({ item }) => (
                <ProductItem
                    image={item.imageUrl} title={item.title} price={item.price}
                    onSelect={null}
                >
                    <Button color={colors.PRIMARY} title="Edit"
                        onPress={null} />
                    <Button
                        color={colors.PRIMARY} title="Delete"
                        onPress={() =>
                            dispatch(productActions.deleteProduct(item.id)) }
                    />
                </ProductItem>
            ) }
    )
}

```

```

        />
    )
}

UserProducts.navigationOptions = ({ navigation }) => ({
  headerTitle: "Your products",
  headerLeft: () => (
    <CustomHeaderButtons
      title="Menu" iconName="menu" onPress={navigation.toggleDrawer}
    />
  )
})

```

Store/actions/user.js

```

export const DELETE_PRODUCT = "DELETE_PRODUCT"
export const deleteProduct = (id) => ({ type: DELETE_PRODUCT, id })

```

Store/reducer/user.js

```

import PRODUCTS from "../../data/dummy"
import { DELETE_PRODUCT } from "../actions/products"

const initialState = {
  availableProducts: PRODUCTS,
  userProducts: PRODUCTS.filter(
    (p) => p.ownerId === "u1") // only user 1
}

const productsReducer = (state = initialState, action) => {
  switch (action.type) {
    case DELETE_PRODUCT:
      return {
        ...state,
        userProducts: state.userProducts.filter(
          (prod) => prod.id !== action.id ),
        availableProducts: state.availableProducts.filter(
          (prod) => prod.id !== action.id )
      }
  }
}

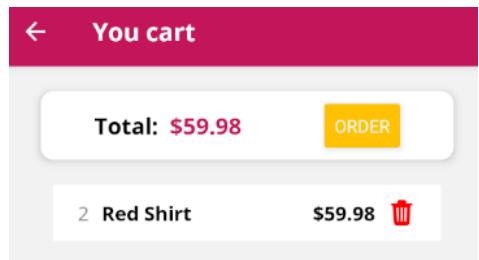
```

```

        default: return state // (1) items in cart
    }
}

export default productsReducer

```



Store/reducer/cart.js

```

...
import { DELETE_PRODUCT } from "../actions/products"

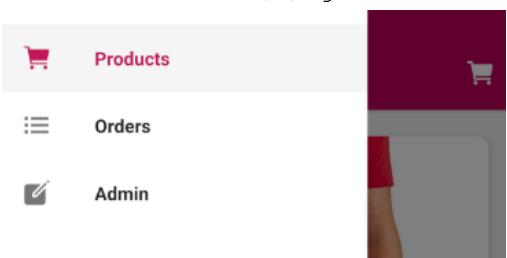
const initialState = { items: {}, total: 0 }

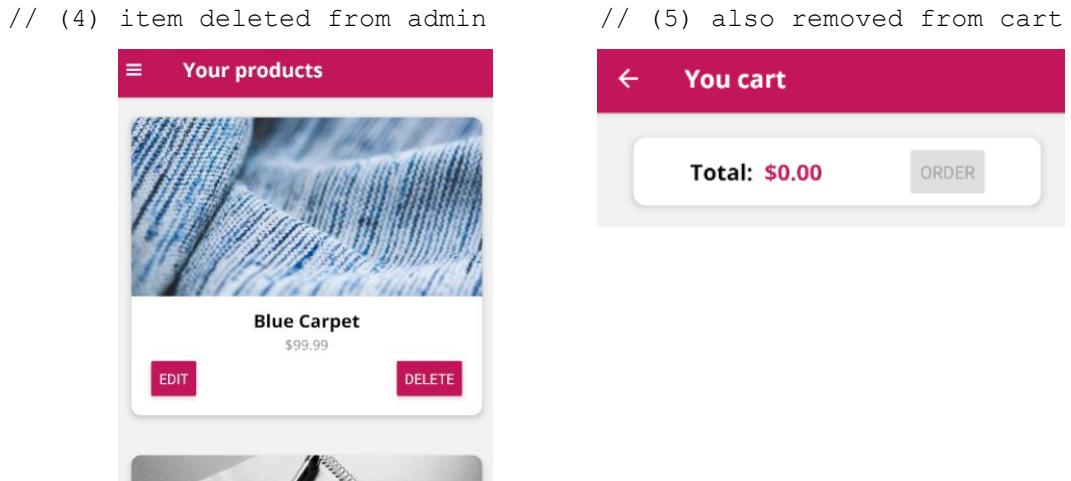
const cartReducer = (state = initialState, action) => {
    switch (action.type) {
        ...
        case ADD_ORDER:
            return initialState // clear cart on add order
        case DELETE_PRODUCT:
            // only if cart item exists in cart, we delete it
            if (state.items[action.id]) {
                const updatedItems = { ...state.items }
                const itemTotal = state.items[action.id].sum

                delete updatedItems[action.id]
                    // (2) go to admin
                return {
                    ...state,
                    items: updatedItems,
                    total:
                        state.total - itemTotal
                }
            }
            default: return state
    }
}
...

// (3) delete the item

```





Basic editing and Navigation logic

Navigation/Shop.jsx

```
...
import EditProductScreen from "../screens/user/EditProduct"
...

const AdminNavigator = createStackNavigator(
{
  UserProducts: UserProductsScreen ,
EditProduct: EditProductScreen
},
{
  navigationOptions: createSharedNavOptions("create"),
  defaultNavigationOptions: sharedDefaultNavOptions
}
)
```

Screens/user/UserProduct.jsx

```
...
export default function UserProducts(props) {
  ...
  const handleEditProduct = (id) => {
    props.navigation.navigate("EditProduct", { productId: id })
  }
}
```

```

        return (
          <FlatList
            data={userProducts}
            keyExtractor={(item) => item.id}
            renderItem={({ item }) => (
              <ProductItem
                image={item.imageUrl} title={item.title} price={item.price}
                onSelect={() => handleEditProduct(item.id)}
              >
                <Button color={colors.PRIMARY} title="Edit"
                  onPress={() => handleEditProduct(item.id)}
                />
                <Button color={colors.PRIMARY} title="Delete"
                  onPress={() =>
                    dispatch(productActions.deleteProduct(item.id))
                  }
                />
              </ProductItem>
            ) }
          />
        )
      }
    }

UserProducts.navigationOptions = ({ navigation }) => ({
  headerTitle: "Your products",
  headerLeft: () => (
    <CustomHeaderButtons title="Menu" iconName="menu"
      onPress={navigation.toggleDrawer}
    />
  ),
  headerRight: () => (
    <CustomHeaderButtons title="Add" iconName="create"
      onPress={() => navigation.navigate("EditProduct")} // NO params!
    />
  )
})

```

Screens/user/EditProduct.jsx

```

import React, { useCallback, useEffect, useState } from "react"
import { View, Text, TextInput, ScrollView, StyleSheet } from "react-native"

```

```
import { useSelector } from "react-redux"
import CustomHeaderButtons from "../../UI/CustomHeaderButtons"

export default function EditProduct(props) {
  const prodId = props.navigation.getParam("productId")
  const targetProduct = useSelector((state) =>
    state.products.userProducts.find((prod) => prod.id === prodId)
  )
  const [title, setTitle] = useState(targetProduct?.title || "")
  const [imageUrl, setImageUrl] = useState(
    targetProduct?.imageUrl || ""
  )
  const [price, setPrice] = useState("")
  const [description, setDescription] = useState(
    targetProduct?.description || ""
  )

  const handleSubmit = useCallback(() => console.log("submit"), [])

  useEffect(() => {
    props.navigation.setParams({ handleSubmit })
  }, [handleSubmit])

  return (
    <ScrollView>
      <View style={_styles.form}>
        <FormElement label="Title" value={title}
          onChangeText={setTitle} />
        <FormElement label="Image URL" value={imageUrl}
          onChangeText={setImageUrl} />
        {
          // edit price only in 'Add product' mode
          !targetProduct && (
            <FormElement label="Price" value={price}
              onChangeText={setPrice} />
          )
        }
        <FormElement label="Description" value={description}
          onChangeText={setDescription} />
      </View>
    
```

```

        </ScrollView>
    )
}

EditProduct.navigationOptions = ({ navigation }) => {
  const handleSubmit = navigation.getParam("handleSubmit")

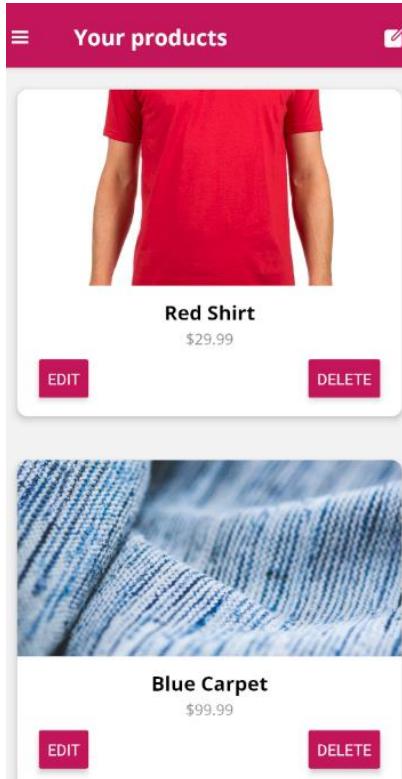
  return {
    headerTitle: navigation.getParam("productId")
      ? "Edit Product"
      : "Add product",
    headerLeft: () => (
      <CustomHeaderButtons title="Menu" iconName="menu"
        onPress={navigation.toggleDrawer}
      />
    ),
    headerRight: () => (
      <CustomHeaderButtons title="Save" iconName="checkmark"
        onPress={handleSubmit}
      />
    )
  }
}

const _styles = StyleSheet.create({
  form: { margin: 20 },
  formControl: { width: "100%" },
  label: { fontFamily: "open-sans-bold", marginVertical: 8 },
  input: { paddingHorizontal: 2, paddingVertical: 5,
    borderBottomColor: "#ccc", borderBottomWidth: 1
  }
})

function FormElement({ label, value, onChangeText }) {
  return (
    <View style={_styles.formControl}>
      <Text style={_styles.label}>{label}</Text>
      <TextInput style={_styles.input} value={value}
        onChangeText={onChangeText} />
    </View>  )
}

```

```
// (1) either edit product with "EDIT", or add with 'Add' icon
```



```
// (2) 'Add' product screen
```

A screenshot of a mobile application interface titled 'Add product'. It contains four input fields: 'Title', 'Image URL', 'Price', and 'Description'. There is also a checkmark icon in the top right corner.

```
// (3) 'edit' product screen (pre-populated)
```

A screenshot of a mobile application interface titled 'Edit Product'. The 'Title' field is populated with 'Red Shirt'. The 'Image URL' field contains the URL 'photo/2016/10/02/22/17/red-t-shirt-1710578_1280.jpg'. The 'Description' field contains the text 'A red t-shirt, perfect for days with non-red weather.' There is a checkmark icon in the top right corner.

Actions for creating and updating

Store/actions/products.js

```
export const DELETE_PRODUCT = "DELETE_PRODUCT"
export const CREATE_PRODUCT = "CREATE_PRODUCT"
export const UPDATE_PRODUCT = "UPDATE_PRODUCT"

export const deleteProduct = (id) => ({ type: DELETE_PRODUCT, id })
export const createProduct = (title, description, imageUrl, price) =>
({
  type: CREATE_PRODUCT, payload: { title, description, imageUrl, price }
})
export const updateProduct = (id, title, description, imageUrl) =>
({
  type: UPDATE_PRODUCT, payload: { id, title, description, imageUrl }
})
```

Store/reducer/products.js

```
import PRODUCTS from "../../data/dummy"
import { Product } from "../../models/product"
import { DELETE_PRODUCT, UPDATE_PRODUCT, CREATE_PRODUCT }
} from "../actions/products"

const initialState = {
  availableProducts: PRODUCTS,
  userProducts:
    PRODUCTS.filter((p) => p.ownerId === "u1") // only user 1
}

const productsReducer = (state = initialState, action) => {
  switch (action.type) {
    case DELETE_PRODUCT: ...
    case CREATE_PRODUCT:
      const newProduct = new Product(
        new Date().toString(), "u1",
        action.payload.title, action.payload.imageUrl,
        action.payload.description, action.payload.price
      )
      return {
        ...state,
        availableProducts: state.availableProducts.concat(newProduct),
        userProducts: state.userProducts.concat(newProduct)
      }
    case UPDATE_PRODUCT:
      const idxInUserProducts = state.userProducts.findIndex(
        (prod) => prod.id === action.payload.id
      )
      console.log(state.userProducts, action.payload.id)
      const idxInAvailableProducts =
        state.availableProducts.findIndex(
          (prod) => prod.id === action.payload.id
        )
      const updatedProduct = new Product(
        action.payload.id,
        state.userProducts[idxInUserProducts].ownerId,
        action.payload.title, action.payload.imageUrl,
```

```

        action.payload.description,
        state.userProducts[idxInUserProducts].price
    )
    const updatedUserProducts = [...state.userProducts]
    const updatedAvailableProducts = [...state.availableProducts]
    updatedUserProducts[idxInUserProducts] = updatedProduct
    updatedAvailableProducts[idxInAvailableProducts] =
        updatedProduct
    return {
        ...state,
        userProducts: updatedUserProducts,
        availableProducts: updatedAvailableProducts
    }
    default: return state
}
}

export default productsReducer

```

screens/user/EditProduct.jsx

```

...
import { useSelector, useDispatch } from "react-redux"
import * as productActions from "../../store/actions/products"

export default function EditProduct(props) {
    const prodId = props.navigation.getParam("productId")
    const targetProduct = useSelector((state) =>
        state.products.userProducts.find((prod) => prod.id === prodId)
    )
    const dispatch = useDispatch()
    ...

    const handleSubmit = useCallback(() => {
        if (targetProduct) {
            dispatch( productActions.updateProduct(
                prodId, title, description, imageUrl)
            )
        } else {
            dispatch( productActions.createProduct(
                title, description, imageUrl, +price)
            )
        }
    })
}
```

```

        )
    }

    props.navigation.goBack()
}, [dispatch, prodId, title, description, imageUrl, price])

useEffect(() => { props.navigation.setParams({ handleSubmit }) },
[handleSubmit])

return (...)

}

EditProduct.navigationOptions = ({ navigation }) => {
  const handleSubmit = navigation.getParam("handleSubmit")

  return {
    headerTitle: navigation.getParam("productId")
      ? "Edit Product" : "Add product",
    headerLeft: () => (
      <CustomHeaderButtons title="Menu" iconName="menu"
        onPress={navigation.toggleDrawer}
      />
    ),
    headerRight: () => (
      <CustomHeaderButtons title="Save"
        iconName="checkmark" onPress={handleSubmit}
      />
    )
  }
}

...
// (1) add product
// (2) product added

```

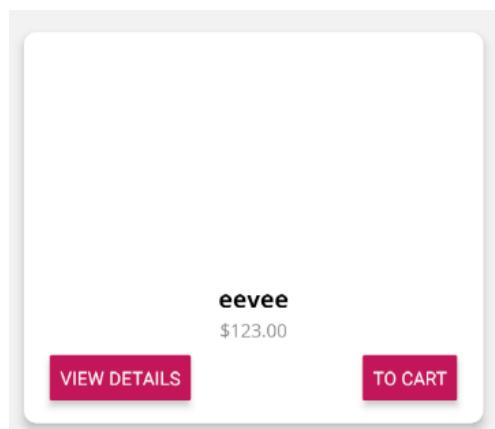
Add product

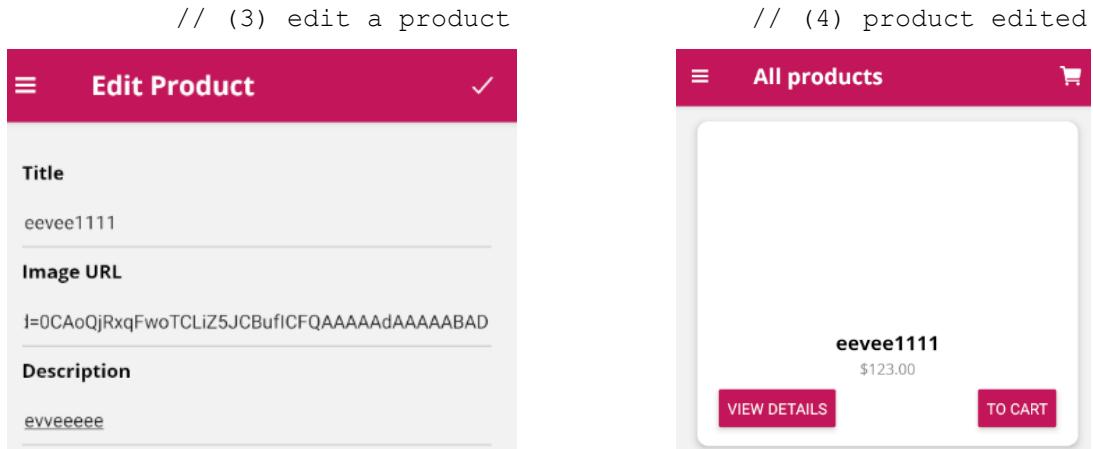
Title
eevee

Image URL
2/pokemon/Eevee_1491460844_733117_1024x576.jpg

Price
123

Description
evveeeee

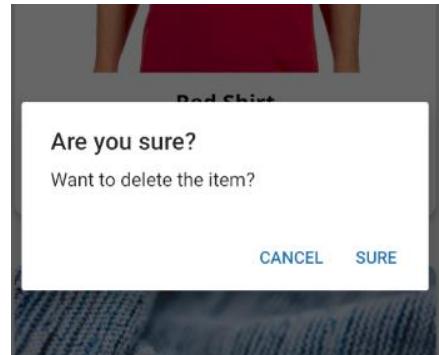




Improving the app

Screens/user/UserProducts.jsx

```
...
export default function UserProducts(props) {
...
const handleDelete = (id) =>
  Alert.alert("Are you sure?", "Want to delete the item?", [
    { text: "Cancel", style: "default" },
    { text: "Sure", style: "destructive",
      onPress: () => dispatch(productActions.deleteProduct(id))
    }
  ])
...
return (
  <FlatList
    data={userProducts}
    keyExtractor={(item) => item.id}
    renderItem={({ item }) => (
      <ProductItem image={item.imageUrl} title={item.title}
        price={item.price} onSelect={() => handleEditProduct(item.id)}
      >
        <Button color={colors.PRIMARY} title="Edit"
          onPress={() => handleEditProduct(item.id)} />
        <Button color={colors.PRIMARY} title="Delete"
          onPress={handleDelete.bind(this, item.id)} />
      </ProductItem>
    )} /> )
)
```



```
constants/styles.js
export const CARD_SHADOW = {
  borderRadius: 10, backgroundColor: "white",
  elevation: 5, // android
  shadowColor: "black", // iOS
  shadowOpacity: 0.26, shadowRadius: 8,
  shadowOffset: { width: 0, height: 2 }
}
```

```
UI/Card.jsx
import React from "react"
import { View, StyleSheet } from "react-native"
import * as sharedStyles from "../constants/styles"

export default function Card({ children, styles }) {
  return <View style={[_styles.container, styles]}>{children}</View>
}

const _styles = StyleSheet.create({
  container: sharedStyles.CARD_SHADOW })
```

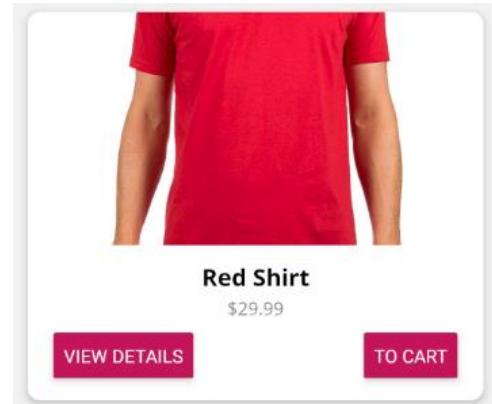
```
components/shop/ProductItem.jsx
...
import Card from "../../UI/Card"

export default function ProductItem(props) {
  return (
    <Card style={_styles.container}> {/* Card styles abstracted */}
      <View style={_styles.touchableContainer}>
        <TouchableComponent onPress={props.onSelect} useForeground>
          <View>
            <View style={_styles.imageContainer}>
              <Image style={_styles.image}
                source={{ uri: props.image }} />
            </View>
            <View style={_styles.textsContainer}>
              <Text style={_styles.title}>{props.title}</Text>
              <Text style={_styles.price}>
```

```

        ${props.price.toFixed(2)}</Text>
    </View>
    <View style={_styles.buttonsContainer}>
        {props.children}</View>
    </View>
</TouchableComponent>
</View>
</Card>
)
}
const _styles = StyleSheet.create({
    container: { height: 300, margin: 20 },
...
})

```



Handling user input

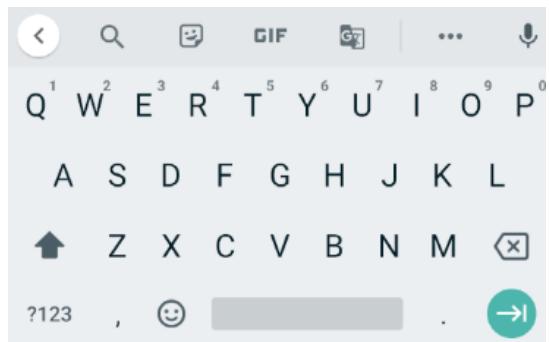
Configuring

Screens/user/EditProduct.jsx

```

...
<FormElement
    label="Image URL"
    value={imageUrl}
    onChangeText={setImageUrl}
    textInputProps={{
        keyboardType: "default",
        autoCapitalize: "sentences",
        autoCorrect: true,
        returnKeyType: "next",
        onEndEditing: () => console.log("closed keyboard"),
        onSelectionChange: () => console.log("selected something"),
        onSubmitEditing: () => console.log("hit 'submit' button ")
    }}
/>

```

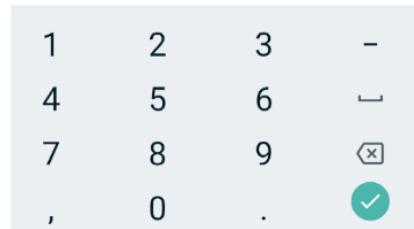


```

{
  !targetProduct && (
    <FormElement
      label="Price"
      value={price}
      onChangeText={setPrice}
      textInputProps={{ keyboardType: "decimal-pad" }}
    />
  )
}

...
function FormElement({ label, value, onChangeText, textInputProps }) {
  return (
    <View style={_styles.formControl}>
      <Text style={_styles.label}>{label}</Text>
      <TextInput
        style={_styles.input} value={value}
        onChangeText={onChangeText}
        {...textInputProps}
      />
    </View>
  )
}

```



Basic validation

Screens/user/EditProduct.jsx

```

...
const [title, setTitle] = useState(targetProduct?.title || "")
const [isValid, setIsisValid] = useState(false)

...
const handleSubmit = useCallback(() => {
  if (!isValid) {
    return Alert.alert(
      "Submit failed", "There are validation errors", [
        { text: "OK", style: "default" }
      ]
    )
  }
  if (targetProduct) {
    dispatch(

```

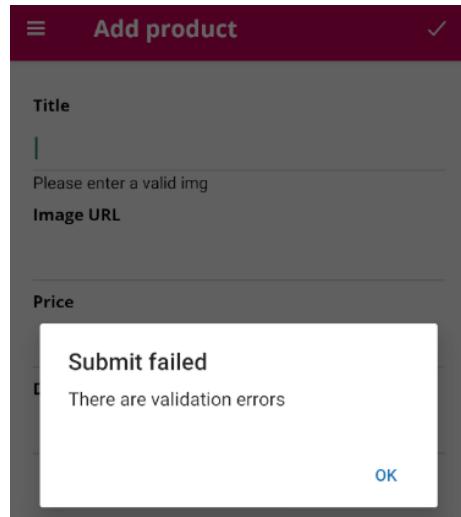
```

        productActions.updateProduct(prodId, title, description, imageUrl)
    )
} else {
    dispatch(
        productActions.createProduct(title, description, imageUrl, +price)
    )
}
props.navigation.goBack()
}, [dispatch, prodId, title,
description, imageUrl, price])

const handleOnChangeText = (text) => {
if (text.trim().length === 0) {
    setIsTitleValid(false)
} else {
    setIsTitleValid(true)
}
setTitle(text)
}

...
return (
...
<FormElement label="Title"
    value={title} onChangeText={handleOnChangeText}
    validationMsg={isTitleValid ? "" : "Please enter a valid img"}
/>
...

```



useReducer and outsourcing input component

UI/Input.jsx

```

import React, { useEffect, useReducer } from "react"
import { View, Text, TextInput, StyleSheet } from "react-native"

const INPUT_CHANGE = "INPUT_CHANGE"
const INPUT_BLUR = "INPUT_BLUR"

const inputReducer = (state, action) => {
    switch (action.type) {
        case INPUT_CHANGE:

```

```
        return { ...state, value: action.value, isValid: action.isValid
    }

    case INPUT_BLUR: return { ...state, touched: true }

    default: return state
}

}

export default function Input({
    id, initialValue, initialIsValid, label, value, onChangeText,
    validationMsg, required, email, min, max, minLength,
    ...textInputProps
}) {
    const [state, dispatch] = useReducer(inputReducer, {
        value: initialValue || "",
        isValid: initialIsValid,
        touched: false
    })

    useEffect(() => {
        if (state.touched) onChangeText(id, state.value, state.isValid)
    }, [state, id, onChangeText])

    const handleOnChangeText = (text) => {
        const emailRegex =
            /^(([^<>()\\[\\]\\.,;:\\s@"]+|(\.[^<>()\\[\\]\\.,;:\\s@"]+)+)|(.+))@((\[[0-9]{1,3}\].[0-9]{1,3}\).[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-\-0-9]+\.)+[a-zA-Z]{2,}))$/;

        let isValid = true

        if (required && text.trim().length === 0) isValid = false
        if (email && !emailRegex.test(text.toLowerCase())) isValid = false
        if (Number.isInteger(min) && +text < min) isValid = false
        if (Number.isInteger(max) && +text > max) isValid = false
        if (Number.isInteger(minLength) && text.length < minLength) isValid = false

        dispatch({ type: INPUT_CHANGE, value: text, isValid })
    }
}
```

```

const handleOnBlur = () => dispatch({ type: INPUT_BLUR })

return (
  <View style={_styles.formControl}>
    <Text style={_styles.label}>{label}</Text>
    <TextInput
      style={_styles.input}
      value={state.value}
      onChangeText={handleOnChangeText}
      onBlur={handleOnBlur}
      {...textInputProps}
    />
    {!state.isValid && state.touched &&
      <Text>{validationMsg || "Invalid input"}</Text>}
  </View>
)
}

const _styles = StyleSheet.create({
  formControl: { width: "100%" },
  label: { fontFamily: "open-sans-bold", marginVertical: 8 },
  input: {
    paddingHorizontal: 2, paddingVertical: 5,
    borderBottomColor: "#ccc", borderBottomWidth: 1
  }
})

```

Screens/user/EditProduct.jsx

```

import React, { useCallback, useEffect, useReducer } from "react"
import { View, ScrollView, StyleSheet, Alert } from "react-native"
import { useSelector, useDispatch } from "react-redux"
import CustomHeaderButtons from "../../UI/CustomHeaderButtons"
import Input from "../../UI/Input"
import * as productActions from "../../store/actions/products"

const UPDATE_FORM_INPUT = "UPDATE_FORM_INPUT"

const formReducer = (state, action) => {
  if (action.type === UPDATE_FORM_INPUT) {

```

```
const updatedValues = {
  ...state.values, [action.input]: action.value
}
const updatedValidations = {
  ...state.validations, [action.input]: action.isInputValid
}

let isFormValid = true

for (const key in updatedValidations) {
  isFormValid = isFormValid && updatedValidations[key]
}

return {
  values: updatedValues,
  validations: updatedValidations,
  isFormValid
}
}

return state
}

export default function EditProduct(props) {
  const prodId = props.navigation.getParam("productId")
  const targetProduct = useSelector((state) =>
    state.products.userProducts.find((prod) => prod.id === prodId)
  )
  const dispatch = useDispatch()

  const [formState, formDispatch] = useReducer(formReducer, {
    values: {
      title: targetProduct?.title || "",
      imageUrl: targetProduct?.imageUrl || "",
      description: targetProduct?.description || "",
      price: ""
    },
    validations: {
      title: targetProduct ? true : false,
      imageUrl: targetProduct ? true : false,
    }
  })
}
```

```
        description: targetProduct ? true : false,
        // price starts false on 'new product' mode, always true on
        // 'edit' mode
        price: targetProduct ? true : false
    },
    isFormValid: targetProduct ? true : false
})

const handleSubmit = useCallback(() => {
    if (!formState.isFormValid) {
        return Alert.alert(
            "Submit failed", "There are validation errors", [
                { text: "OK", style: "default" }
            ]
        )
    }

    const [title, description, imageUrl, price] = formState.values

    if (targetProduct) {
        dispatch(
            productActions.updateProduct(prodId, title, description, imageUrl)
        )
    } else {
        dispatch(
            productActions.createProduct(title, description, imageUrl, +price)
        )
    }

    props.navigation.goBack()
}, [dispatch, prodId, formState])

const handleOnChangeText = useCallback(
    (input, value, isValid) => {
        formDispatch({
            type: UPDATE_FORM_INPUT, value, isValid, input
        })
    },
    [formDispatch]
)
```

```

useEffect(() => {
  props.navigation.setParams({ handleSubmit })
}, [handleSubmit])

return (
  <ScrollView>
    <View style={_styles.form}>
      <Input
        id="title"
        initialValue={targetProduct?.title || ""}
        initialIsValid={Boolean(targetProduct)}
        label="Title"
        value={formState.values.title}
        // binding sets default args at the end!!! ← ← ←
        onChangeText={handleOnchangeText}
        isValid={formState.validations.title}
        validationMsg="Please enter a valid title"
        keyboardType="default"
        autoCapitalize="sentences"
        autoCorrect
        required
      />
      <Input
        id="imageUrl"
        initialValue={targetProduct?.imageUrl || ""}
        initialIsValid={Boolean(targetProduct)}
        label="Image URL"
        value={formState.values.imageUrl}
        onChangeText={handleOnchangeText}
        isValid={formState.validations.imageUrl}
        validationMsg="Please enter a valid image url"
        required
        // {
        //   returnKeyType="next"
        //   onEndEditing: () => console.log("closed keyboard"),
        //   onSelectionChange: () => console.log("selected something"),
        //   onSubmitEditing: () => console.log("hit 'submit' button ")
        // }
      />
    {
      // edit price only in 'Add product' mode
    }
  
```

```

        !targetProduct && (
            <Input
                id="price"
                label="Price"
                value={formState.values.price}
                onChangeText={handleOnChangeText}
                isValid={formState.validations.price}
                validationMsg="Please enter a valid price"
                keyboardType="decimal-pad"
                required
                min={0.1}
            />
        )
    }
    <Input
        id="description"
        initialValue={targetProduct?.description || ""}
        initialIsValid={Boolean(targetProduct)}
        label="Description"
        value={formState.values.description}
        onChangeText={handleOnChangeText}
        validationMsg="Please enter a valid description"
        isValid={formState.validations.description}
        autoCapitalize="sentences"
        autoCorrect
        multiline
        numberOfLines={3}
        required
        minLength={3}
    />
</View>
</ScrollView>
)
}

EditProduct.navigationOptions = ({ navigation }) => {
    const handleSubmit = navigation.getParam("handleSubmit")

    return {
        headerTitle: navigation.getParam("productId")
    }
}

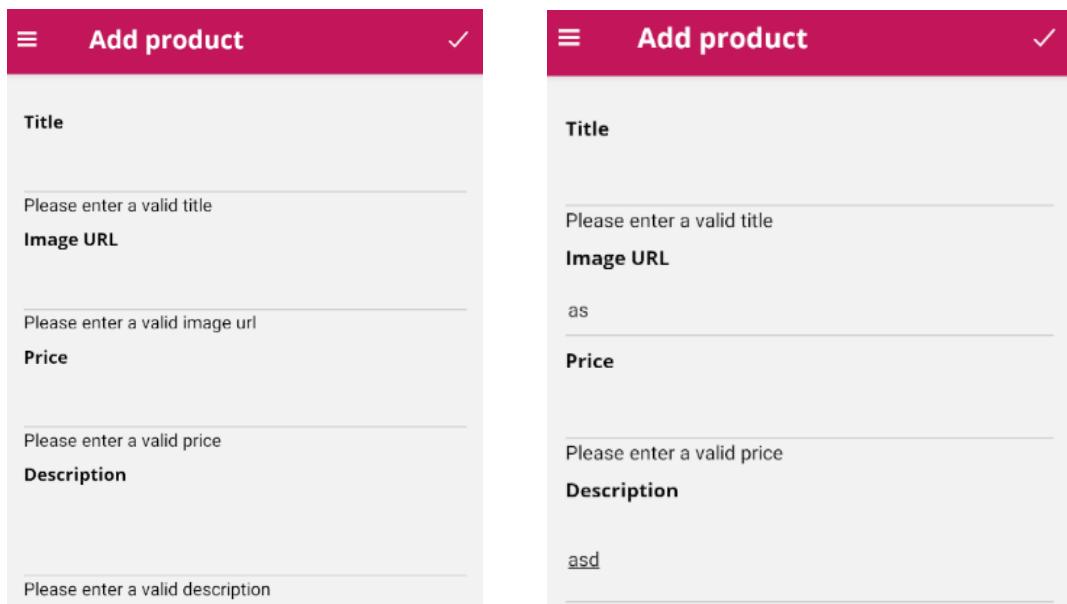
```

```

        ? "Edit Product" : "Add product",
      headerLeft: () => (
        <CustomHeaderButtons title="Menu" iconName="menu"
          onPress={navigation.toggleDrawer}
        />
      ),
      headerRight: () => (
        <CustomHeaderButtons title="Save" iconName="checkmark"
          onPress={handleSubmit}
        />
      )
    }
  }

const _styles = StyleSheet.create({ form: { margin: 20 } })

```



Tweaking styles and handling soft keyboard

Screens/user/EditProduct.jsx

```

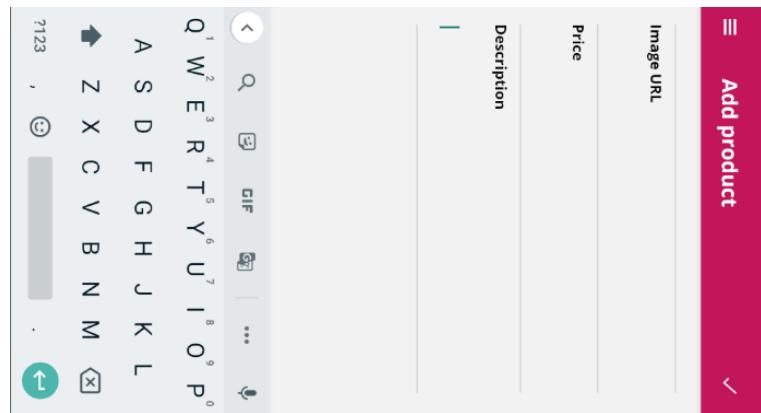
...
<KeyboardAvoidingView
  style={{ flex: 1 }} // KeyboardAvoidingView needs the whole screen
  behavior="padding"

```

```

    keyboardVerticalOffset={10} // padding of 10
  >
  <ScrollView>
    <View style={_styles.form}>
      <Input id="title" ... />
      ...
    </View>
  </ScrollView>
</KeyboardAvoidingView>

```

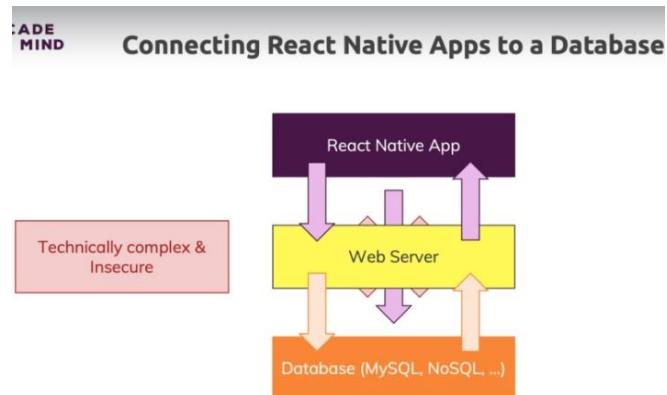


Alternatives

Validation can be handled by libraries instead of reinventing the wheel. Awesome alternatives are **formik** or **validate.js**.

HTTP Requests and adding server + database

Firebase

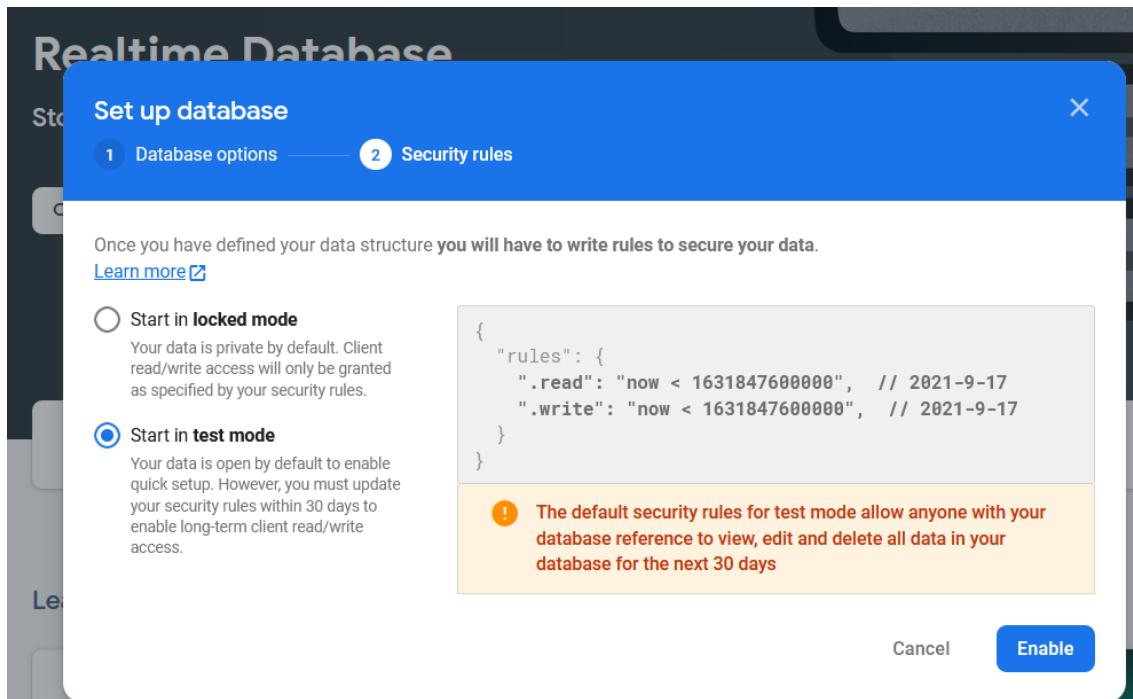


Firebase gives you a REST API and a database. Lots of other things too, but we only need those two for now.

Create a google account, and [go to Google Firebase console](#).

Add a [project](#), name it as you desire.

Create a [real time database](#) (not Firestore). Start in [test mode](#).



It will offer you an endpoint used to communicate with the database.

It looks like you are talking to the database due to the structure of the code used, but [under the hood you are writing to that endpoint handled](#).

The screenshot shows the 'Realtime Database' dashboard. At the top, there are tabs: 'Data' (selected), 'Rules', 'Backups', and 'Usage'. Below the tabs, there's a note: 'Protect your Realtime Database resources from abuse, such as billing fraud or phishing'. Under the 'Data' tab, there's a URL: 'https://rnmcshoppingapp-default-rtbd.firebaseio.com/'. At the bottom, there's a status message: 'rnmcshoppingapp-default-rtbd: null'.

Redux thunk

Redux thunk is a middleware that allows us to change the action creators to be functions instead of plain objects.

This is extremely useful to handle asynchronous code (handling side effects), and more now that we are to deal with a database.

Thunk ensures all redux code is handled asynchronously in a structured way under the hood for actions to reach the reducer in sequence, as in the end they must be processed synchronously.

When actions resolve, they are dispatched. No sooner or later.

```
$ npm install --save redux-thunk
```

App.js

```
...
import { createStore, combineReducers, applyMiddleware } from "redux"
import ReduxThunk from "redux-thunk"
import { Provider } from "react-redux"
import { composeWithDevTools } from "redux-devtools-extension"
...
const rootReducer = combineReducers({
  products: productsReducer, cart: cartReducer, orders: ordersReducer
})

// remember to remove `composeWithDevtools` before deploying!
const store = createStore(
  rootReducer,
  composeWithDevTools(applyMiddleware(ReduxThunk))
)
...
...
```

Store/actions/product.js

```
...
export const createProduct = (title, description, imageUrl, price) =>
{
  return async (dispatch) => {
    /**
     * you can fire any async code here! Redux thunk handles it.
     */
}
```

```

* Firebase sends requests to any node.
*
* By default, using just the uri will translate to a GET request,
* > If the resource does not exist, Firebase creates a document
*   for it.
* > All endpoints need `json` extension. This is exclusive to
*   Firebase.
* > Return value is a promise, which resolves to the data from
*   db.
* > That data is a raw string. It needs to be parsed to JSON.
*
* Adding an object as a second argument converts the request to
* the specified method. We use POST here.
* > Firebase auto-generates an ID. Convenient.
*/
const newProduct = { title, description, imageUrl, price }

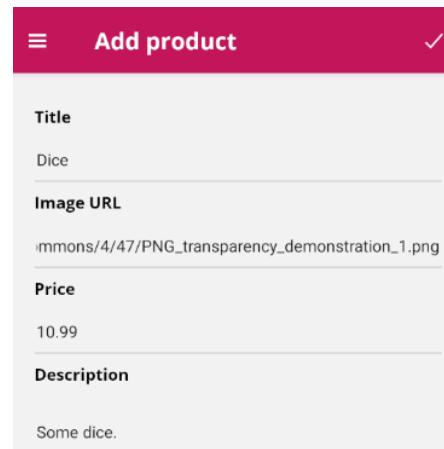
try {
  const response = await fetch(
    "https://rnmcshoppingapp-default-firebase.firebaseio.com/products.json",
    {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(newProduct)
    }
  )

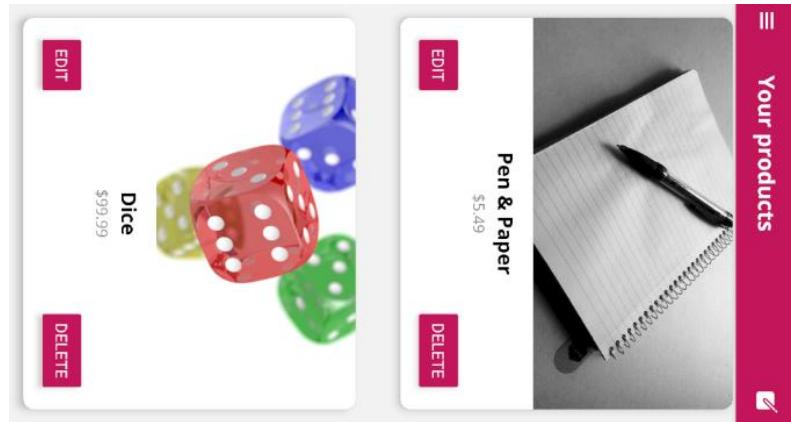
  const data = await response.json()

  dispatch({ type: CREATE_PRODUCT, payload: newProduct })
} catch (err) {
  console.log(err)
}
}

// generated id is in returned "name" key
Object {
  "name": "-Ll1HHs9EZNarP6up3fu",
}

```





<https://rnmcshoppingapp-default-firebase.com/>

```
rnmcsShoppingApp-default-firebase
  ...
  products
    ...
    -MhQUOKz3XWoTvUAsXWm
      ...
      description: "Dice!"
      imageUrl: "https://upload.wikimedia.org/wikipedia/commons/..."
      price: 99.99
      title: "Dice"
```

Fetching products from server

Note: This is how Firebase returns documents from a collection when fetched by GET.

```
Object {
  "-Ll1HHs9EZNarP6up3fu": Object {
    "description": "This is a white shirt which is pretty stylish!",
    "imageUrl": "https://cdn.pixabay.com/photo/2017/01/13/04/56/blank-1976334_1280.png",
    "price": 39.99,
    "title": "A White Shirt",
  },
}
```

They come back as an object with each document in the database as inner objects mapped to their firebase ID as keys.

Store/actions/products.js

```
import { Product } from "../../models/product"

export const DELETE_PRODUCT = "DELETE_PRODUCT"
export const CREATE_PRODUCT = "CREATE_PRODUCT"
export const UPDATE_PRODUCT = "UPDATE_PRODUCT"
export const SET_PRODUCTS = "SET_PRODUCTS"

export const deleteProduct = (id) => ({ type: DELETE_PRODUCT, id })
```

```

export const createProduct = (title, description, imageUrl, price) =>
{
  return async (dispatch) => {
    try {
      const response = await fetch(
        "https://rnmcshoppingapp-default-firebase.firebaseio.com/products.json",
        {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ title, description, imageUrl, price })
        }
      )

      const data = await response.json()

      dispatch({
        type: CREATE_PRODUCT,
        payload: { id: data.name, title, description, imageUrl, price }
      })
    } catch (err) { console.log(err) }
  }
}

export const updateProduct = (id, title, description, imageUrl) => ({
  type: UPDATE_PRODUCT,
  payload: { id, title, description, imageUrl }
})

export const fetchProducts = () => {
  return async (dispatch) => {
    try {
      const response = await fetch(
        "https://rnmcshoppingapp-default-firebase.firebaseio.com/products.json"
      )

      // Firebase's response status boolean (400, 500)
      // > throwing here will be caught on <ProductOverview />
      if (!response.ok) throw new Error("Something went wrong")

      const data = await response.json()
      const products = []
    }
  }
}

```

```

        for (const key in data) {
            const values = data[key]
            products.push(
                new Product( key, "u1", values.title, values.imageUrl,
                values.description, values.price
            )
        )
    }

    dispatch({
        type: SET_PRODUCTS,
        payload: { id: data.name, products }
    })
} catch (err) {
    console.log(err)
    throw err // send to custom analytics server
}
}

}
}

```

Store/reducer/products.js

```

import { Product } from "../../models/product"
import { DELETE_PRODUCT, UPDATE_PRODUCT, CREATE_PRODUCT, SET_PRODUCTS
} from "../actions/products"

const initialState = { availableProducts: [], userProducts: [] }

const productsReducer = (state = initialState, action) => {
    switch (action.type) {
        case SET_PRODUCTS:
            return {
                availableProducts: action.payload.products,
                userProducts:
                    action.payload.products.filter((p) => p.ownerId === "u1")
            }
        case DELETE_PRODUCT: ...
        case CREATE_PRODUCT:
            const newProduct = new Product(
                action.payload.id, // created in firebase, as "name" (see action)

```

```

        "u1",
        action.payload.title,
        action.payload.imageUrl,
        action.payload.description,
        action.payload.price
    )
}

return {
    ...state,
    availableProducts: state.availableProducts.concat(newProduct),
    userProducts: state.userProducts.concat(newProduct)
}
}

case UPDATE_PRODUCT: ...
default: return state
}

}

export default productsReducer

```

screen/shop/ProductsOverview.jsx

```

import React, { useEffect, useState, useCallback } from "react"
import {
    FlatList, Button, ActivityIndicator, View, StyleSheet, Text
} from "react-native"
...
import * as productsActions from "../../store/actions/products"
import * as sharedStyles from "../../constants/styles"
import colors from "../../constants/colors"

export default function ProductsOverview(props) {
    const [isLoading, setIsLoading] = useState(false)
    const [errorMsg, setErrorMsg] = useState("")
    const availableProducts = useSelector(
        (state) => state.products.availableProducts
    )
    const dispatch = useDispatch()

    const loadProducts = useCallback(async () => {
        setIsLoading(true)
        setErrorMsg("")

```

```
try {
    await dispatch(productsActions.fetchProducts())
} catch (err) {
    // error object comes from Firebase. It has "message" key
    setErrorMsg(err.message)
}

setIsLoading(false)
}, [setIsLoading, dispatch, setErrorMsg])

useEffect(() => {
    loadProducts() // cannot pass a pointer. It's an async function
}, [dispatch, loadProducts])

const handleSelectItem = (id, title) => {
    props.navigation.navigate("ProductDetails", {
        productId: id, productTitle: title
    })
}

if (errorMsg) { // on 400/500 error
    return (
        <View style={_styles.loadingContainer}>
            <Text>Error while fetching data</Text>
            <Button title="Refetch"
                onPress={loadProducts} color={colors.PRIMARY} />
        </View>
    )
}

if (isLoading) { // on loading from firebase
    return (
        <View style={_styles.loadingContainer}>
            <ActivityIndicator size="large" colors={colors.PRIMARY} />
        </View>
    )
}
```

```

// on no items in firebase db
if (!isLoading && !Boolean(availableProducts.length)) {
  return (
    <View style={_styles.loadingContainer}>
      <Text>No products found</Text>
    </View>
  )
}

return (
  <FlatList
    data={availableProducts}
    renderItem={({ item }) => (
      <ProductItem
        image={item.imageUrl} title={item.title}
        price={item.price}
        onSelect={() => handleSelectItem(item.id, item.title)}
      >
        <Button color={colors.PRIMARY} title="View details"
          onPress={() => handleSelectItem(item.id, item.title)}
        />
        <Button color={colors.PRIMARY} title="To cart"
          onPress={() => dispatch(cartActions.addToCart(item))}
        />
      </ProductItem>
    )}
  />
)
}

```

```

ProductsOverview.navigationOptions = ({ navigation }) => ({
  headerTitle: "All products",
  headerRight: () => (
    // need `npm i --save @react-navigation/native`!
    <CustomHeaderButtons title="Cart" iconName="cart"
      onPress={() => navigation.navigate("Cart")}
    />
  ),
  headerLeft: () => (
    <CustomHeaderButtons title="Menu" iconName="menu"

```

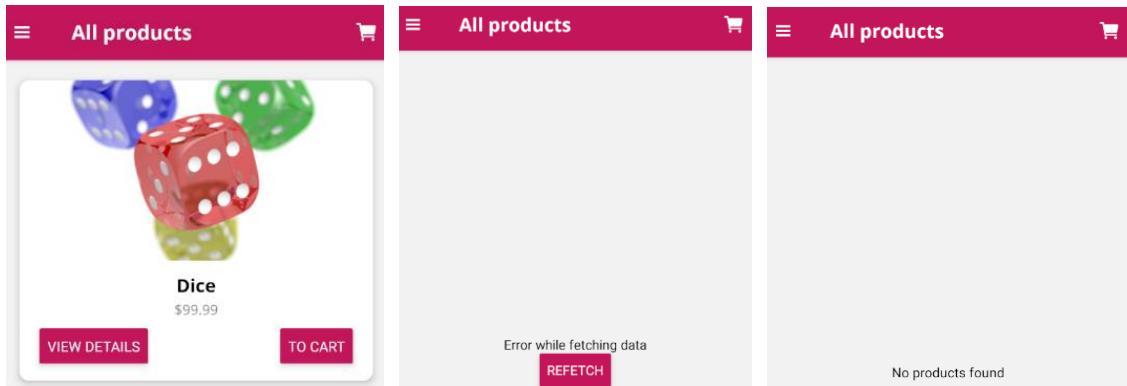
```

        onPress={navigation.toggleDrawer}
      />
    )
}

const styles = StyleSheet.create({
  // flex: 1, alignItems: 'center', justifyContent: 'center'
  loadingContainer: sharedStyles.STRETCH_AND_CENTER
})
```

// spinner loads at start, couldn't catch it, firebase is too fast :P

// on successful load // on error on request // on no products on fb



Navigation listener

Data successfully changes in Firebase as we modify it in RN, but it does not re-fetch with the new data.

Fetched data is stored in memory, and will not change state unless specifically told so. That's why we need navigation listeners for when an async event targeting the database happens.

Screens/user/ProductsOverview.jsx

```

...
const loadProducts = useCallback(async () => {
  setIsLoading(true)
  setErrorMsg("")

  try { await dispatch(productsActions.fetchProducts())
  } catch (err) { setErrorMsg(err.message) }
```

```

    setIsLoading(false)
}, [setIsLoading, dispatch, setErrorMsg])

useEffect(() => { loadProducts( ), [dispatch, loadProducts] )

useEffect(() => {
  // willFocus, willBlur, didFocus, didBlur
  // > there are new methods on react-navigation ^6: focus, blur
  // > also, hooks like `useIsFocused`, `useFocusEffect`
  const willFocusSub =
    props.navigation.addListener("willFocus", loadProducts)
  return () => willFocusSub.remove()
}, [loadProducts])

...

```

Updating and deleting products

```

import { Product } from "../../models/product"

export const DELETE_PRODUCT = "DELETE_PRODUCT"
export const CREATE_PRODUCT = "CREATE_PRODUCT"
export const UPDATE_PRODUCT = "UPDATE_PRODUCT"
export const SET_PRODUCTS = "SET_PRODUCTS"

export const deleteProduct = (id) => {
  return async (dispatch) => {
    try {
      await fetch(
        `https://rnmcsShoppingApp-default-firebase.firebaseio.com/
          products/${id}.json`,
        { method: "DELETE" }
      )

      if (!response.ok) throw new Error("Something went wrong")

      return dispatch({ type: DELETE_PRODUCT, id })
    } catch (err) { console.log(err) }
  }
}

```

```

export const createProduct = (title, description, imageUrl, price) =>
{
  return async (dispatch) => {
    try {
      const response = await fetch(
        "https://rnmcshoppingapp-default-firebase.firebaseio.com/products.json",
        {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ title, description, imageUrl, price })
        }
      )

      const data = await response.json()

      dispatch({
        type: CREATE_PRODUCT,
        // id is returned as "name" key in generated document
        payload: { id: data.name, title, description, imageUrl, price }
      })
    } catch (err) { console.log(err) }
  }
}

export const updateProduct = (id, title, description, imageUrl) => {
  return async (dispatch) => {
    try {
      await fetch(
        `https://rnmcshoppingapp-default-firebase.firebaseio.com/
products/${id}.json`,
        {
          // 'PUT' overrides, 'PATCH' updates where differs
          method: "PATCH",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ title, description, imageUrl })
        }
      )
    }

    if (!response.ok) throw new Error("Something went wrong")
  }
}

```

```

    return dispatch({
      type: UPDATE_PRODUCT,
      payload: { id, title, description, imageUrl }
    })
  } catch (err) { console.log(err) }
}

export const fetchProducts = () => {
  return async (dispatch) => {
    try {
      const response = await fetch(
        "https://rnmcshoppingapp-default-rtbd.firebaseio.com/products.json"
      )
      if (!response.ok) throw new Error("Something went wrong")

      const data = await response.json()
      const products = []

      for (const key in data) {
        const values = data[key]
        products.push(
          new Product( key, "ul", values.title, values.imageUrl,
            values.description, values.price
          )
        )
      }
    }

    dispatch({ type: SET_PRODUCTS, payload: { id: data.name, products } })
  } catch (err) {
    console.log(err)
    throw err
  }
}
}

```

Handling additional errors

UI/FetchViews.jsx

```

import React from "react"
import { View, Text, Button, ActivityIndicator, StyleSheet } from
"react-native"
import * as sharedStyles from "../constants/styles"

```

```
import colors from "../constants/colors"

export default function FetchViews({
  errorMsg, errorButtonProps, emptyResponseMsg, isLoading,
  response, children
}) {
  if (errorMsg) {
    return <Error message={errorMsg} buttonProps={errorButtonProps} />
  }

  if (isLoading) return <Loading />

  if (!isLoading && !Boolean(response)) {
    return <EmptyResponse message={emptyResponseMsg} />
  }

  return children
}

export function Loading() {
  return (
    <View style={_styles.container}>
      <ActivityIndicator size="large" colors={colors.PRIMARY} />
    </View>
  )
}

export function Error({ message, buttonProps }) {
  return (
    <View style={_styles.container}>
      <Text>{message}</Text>
      {buttonProps && <Button color={colors.PRIMARY} {...buttonProps} />}
    </View>
  )
}

export function EmptyResponse({ message }) {
  return (
    <View style={_styles.container}><Text>{message}</Text></View>
  )
}
```

```

const _styles = StyleSheet.create({
  container: sharedStyles.STRETCH_AND_CENTER
})

Screens/user/EditProduct.jsx

...
import FetchViews from "../../UI/FetchViews"
...

export default function EditProduct(props) {
  const [isLoading, setIsLoading] = useState(false)
  const [errorMsg, setErrorMsg] = useState("")

  ...
  useEffect(() => {
    if (errorMsg)
      Alert.alert("Error on submit", errorMsg, [{ text: "OK" }])
    }, [errorMsg])

  const handleSubmit = useCallback(async () => {
    if (!formState.isValid) {
      return Alert.alert(
        "Submit failed", "There are validation errors", [
          { text: "OK", style: "default" }
        ]
      )
    }
    setErrorMsg("")
    setIsLoading(true)
    const { title, description, imageUrl, price } = formState.values

    try {
      if (targetProduct) {
        await dispatch(
          productActions.updateProduct(prodId, title, description, imageUrl)
        )
      } else {
        await dispatch(
          productActions.createProduct(title, description, imageUrl, +price)
        )
      }
    } catch (err) {
      setErrorMsg(`An error occurred: ${err.message}`)
    }
  }, [formState, prodId, targetProduct])
}


```

```

        }

        props.navigation.goBack() // only on success

    } catch (err) {
        setErrorMsg(err.message)
    }

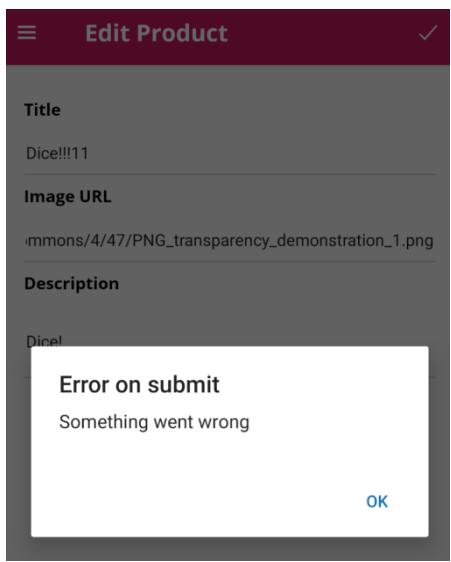
    setIsLoading(false)
}, [dispatch, prodId, formState])

...

return (
<FetchViews isLoading={isLoading} response={!isLoading}>
    <KeyboardAvoidingView style={{ flex: 1 }}

        behavior="padding" keyboardVerticalOffset={10} // on error
    >
        <ScrollView>
            <View style={_styles.form}>
                <Input id="title" ... />
                ...
                </View>
            </ScrollView>
        </KeyboardAvoidingView>
    </FetchViews>
)
}

```



The screenshot shows a mobile application interface titled 'Edit Product'. It has fields for 'Title' (Dice!!!11), 'Image URL' (immons/4/47/PNG_transparency_demonstration_1.png), and 'Description' (Dice). A modal dialog box is displayed with the title 'Error on submit' and the message 'Something went wrong'. There is an 'OK' button at the bottom right of the dialog.

Storing orders

Store/actions/orders.js

```

export const ADD_ORDER = "ADD_ORDER"

export const addOrder = (items, total) => {
    return async (dispatch) => {
        const date = new Date()

        try {
            const response = await fetch(
                "https://rnmcsShoppingApp-default-firebase.firebaseio.com/orders/u1.json",
                {
                    method: "POST",

```

```

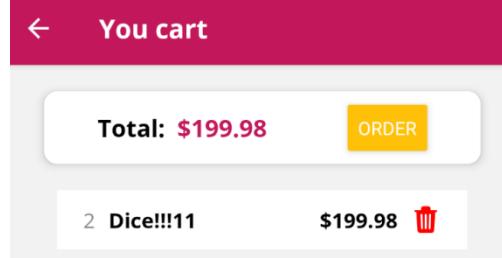
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          items, total, date: date.toISOString() })
      }
    )

    if (!response.ok) {
      throw new Error("Error in `addOrder` @ actions/orders.js")
    }

    const data = await response.json()

    dispatch({
      type: ADD_ORDER,
      payload: { id: data.name, items, total, date }
    })
  } catch (err) {
    throw new Error("Error in `addOrder` @ actions/orders.js.", err)
  }
}
}

```



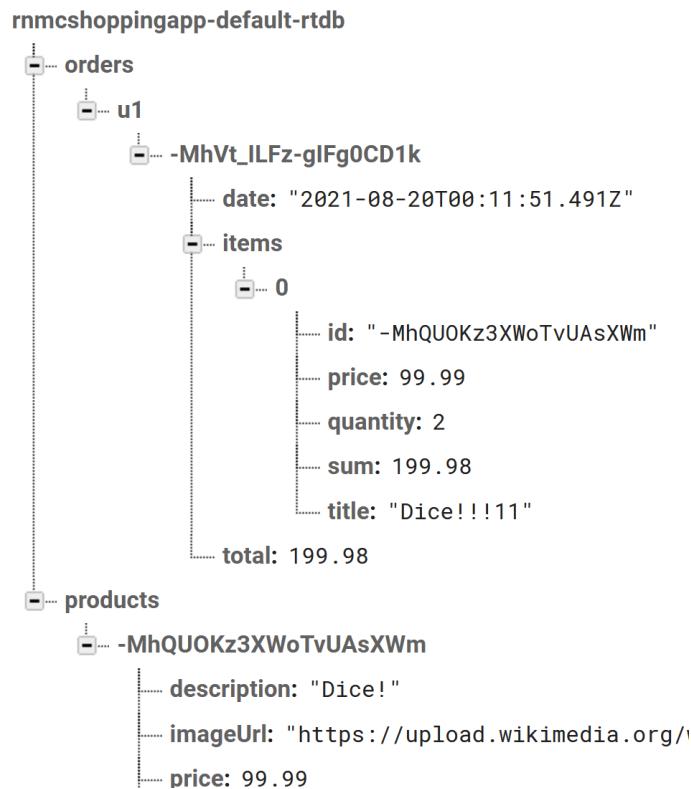
Store/reducers/orders.js

```

...
const initialState = { items: [] }

export default function ordersReducer(state = initialState, action) {
  switch (action.type) {
    case ADD_ORDER:
      return {
        ...state,
        items: state.items.concat(
          new Order(
            action.payload.id, action.payload.items,
            action.payload.total, action.payload.date
          )
        )
      }
    default: return state
  }
}

```



Adding an Activity indicator

Screen/shop/Cart.jsx

```

...
export default function Cart(props) {
  const [isLoading, setIsLoading] = useState(false)
  const total = useSelector((state) => state.cart.total)
  const dispatch = useDispatch()

  const items = useSelector((state) => {
    const transformedCartItems = []
    for (const key in state.cart.items) {
      const currentItem = state.cart.items[key]
      transformedCartItems.push({
        id: key, title: currentItem.title, price: currentItem.price,
        quantity: currentItem.quantity, sum: currentItem.sum
      })
    }
    return transformedCartItems.sort((a, b) => (a.id > b.id ? 1 : -1))
  })
}

```

```

const handleSendOrder = async () => {
  setIsLoading(true)
  await dispatch(orderActions.addOrder(items, total))
  setIsLoading(false)
}

return (
  <View style={_styles.container}>
    <Card style={_styles.summary}>
      <Text style={_styles.summaryText}>
        Total:{" "}
        <Text style={_styles.amount}>
          ${Math.round(total.toFixed(2) * 100) / 100}
        </Text>
      </Text>
      {isLoading ? (
        <ActivityIndicator size="small" color={color.PRIMARY} />
      ) : (
        <Button
          color={colors.SECONDARY} title="Order"
          disabled={items.length === 0} onPress={handleSendOrder}
        />
      )}
    </Card>
    <FlatList
      data={items}
      renderItem={({ item: { id, quantity, title, sum } }) => (
        <CartItem
          quantity={quantity} title={title} amount={sum}
          onDelete={() => dispatch(cartActions.removeFromCart(id))}
        />
      )}
    />
  </View>
)
}

```



2 A White Shirt!

\$79.98



Fetching stored orders

Store/actions/orders.js

```
import Order from "../../models/order"

export const ADD_ORDER = "ADD_ORDER"
export const SET_ORDERS = "SET_ORDERS"

export const fetchOrders = () => {
  return async (dispatch) => {
    try {
      const response = await fetch(
        "https://rnmcshoppingapp-default-firebase.firebaseio.com/orders/ul.json"
      )

      if (!response.ok) throw new Error("Something went wrong")

      const data = await response.json()
      const orders = []

      for (const key in data) {
        const values = data[key]
        orders.push(
          // date comes from fb as a string, we need an object. `new Date`
          new Order(
            key, values.items, values.total, new Date(values.date))
        )
      }

      dispatch({ type: SET_ORDERS, orders })
    } catch (err) {
      console.log(err)
      throw err
    }
  }
}

export const addOrder = (items, total) => {...}
```

Store/reducer/orders.js

```
import Order from "../../models/order"
import { ADD_ORDER, SET_ORDERS } from "../actions/orders"

const initialState = { items: [] }

export default function ordersReducer(state = initialState, action) {
  switch (action.type) {
    case SET_ORDERS:
      return { items: action.orders }
    case ADD_ORDER:
      return {
        ...state,
        items: state.items.concat(
          new Order(
            action.payload.id, action.payload.items,
            action.payload.total, action.payload.date
          )
        )
      }
    default: return state
  }
}
```

screens/shop/Orders.js

```
...
import FetchViews from "../../UI/FetchViews"
import * as orderActions from "../../store/actions/orders"

export default function Orders(props) {
  const [isLoading, setIsLoading] = useState(true)
  const itemsInCart = useSelector((state) => state.orders.items)
  const dispatch = useDispatch()

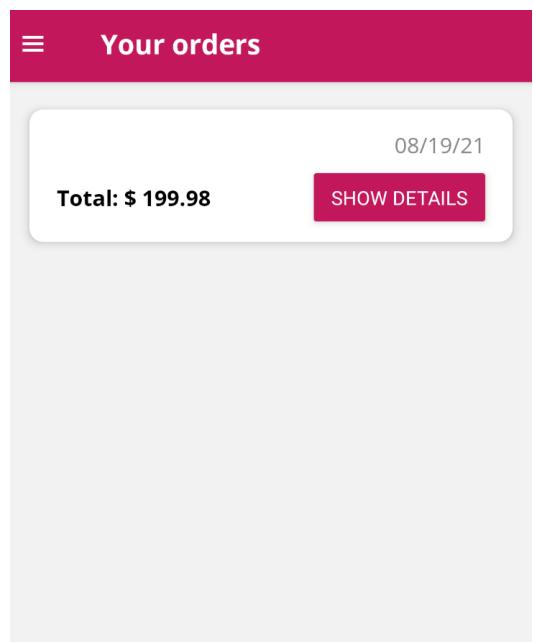
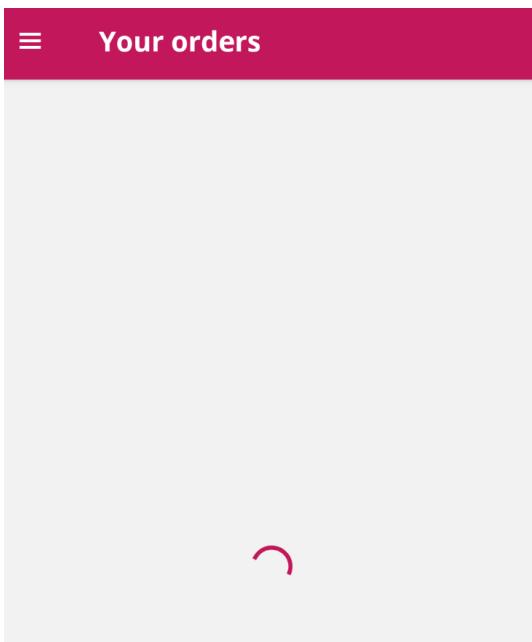
  useEffect(() => {
    dispatch(orderActions.fetchOrders())
      .then(() => setIsLoading(false))
  }, [dispatch, setIsLoading])
```

```

return (
  <FetchViews isLoading={isLoading} response={itemsInCart}>
    <FlatList
      data={itemsInCart}
      renderItem={({ item }) => (
        <OrderItem items={item.items}
          total={item.total} date={item.stringDate}
        />
      ) }
    />
  </FetchViews>
)
}

Orders.navigationOptions = ({ navigation }) => ({
  headerTitle: "Your orders",
  headerLeft: () => (
    <CustomHeaderButtons
      title="Menu"
      iconName="menu"
      onPress={navigation.toggleDrawer}
    />
  )
})
// at mount // after fb response

```



Adding "pull to refresh"

Pull to refresh is the functionality where you gesture a "pull down" motion and a reload action is triggered.

It is built in `FlatList`.

Screens/shop/ProductsOverview.jsx

```
...
export default function ProductsOverview(props) {
  const [isLoading, setIsLoading] = useState(false)
  const [isRefreshing, setIsRefreshing] = useState(false)
  const [errorMsg, setErrorMsg] = useState("")
  const availableProducts = useSelector(
    (state) => state.products.availableProducts
  )
  const dispatch = useDispatch()

  const loadProducts = useCallback(async () => {
    setErrorMsg("")
    setIsRefreshing(true)

    try { await dispatch(productsActions.fetchProducts()) }
    catch (err) { setErrorMsg(err.message) }

    setIsRefreshing(false)
  }, [setIsRefreshing, dispatch, setErrorMsg])

  useEffect(() => {
    setIsLoading(true)
    loadProducts().then(() => setIsLoading(false))
  }, [dispatch, loadProducts, setIsLoading])

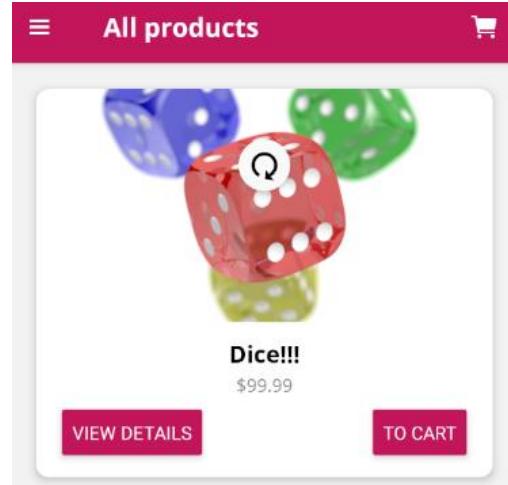
  useEffect(() => {
    const willFocusSub =
      props.navigation.addListener("willFocus", loadProducts)
    return () => willFocusSub.remove()
  }, [loadProducts])
```

```

const handleSelectItem = (id, title) => {
  props.navigation.navigate("ProductDetails", {
    productId: id, productTitle: title
  })
}

return (
  <FetchViews
    errorMsg={errorMsg} isLoading={isLoading}
    errorButtonProps={{ title: "Refetch", onPress: loadProducts }}
    emptyResponseMsg="No products found"
    response={availableProducts}
  >
  <FlatList
    data={availableProducts}
    refreshing={isRefreshing}
    onRefresh={loadProducts}
    renderItem={({ item }) => (
      <ProductItem
        image={item.imageUrl} title={item.title} price={item.price}
        onSelect={() => handleSelectItem(item.id, item.title)}
      >
      <Button color={colors.PRIMARY} title="View details"
        onPress={() => handleSelectItem(item.id, item.title)}
      />
      <Button
        color={colors.PRIMARY} title="To cart"
        onPress={() => dispatch(cartActions.addToCart(item)) }
      />
    </ProductItem>
  )}
  />
</FetchViews>
)
}
...

```



User authentication

Intro

Mobile applications do not use session persistence in the way the web does.

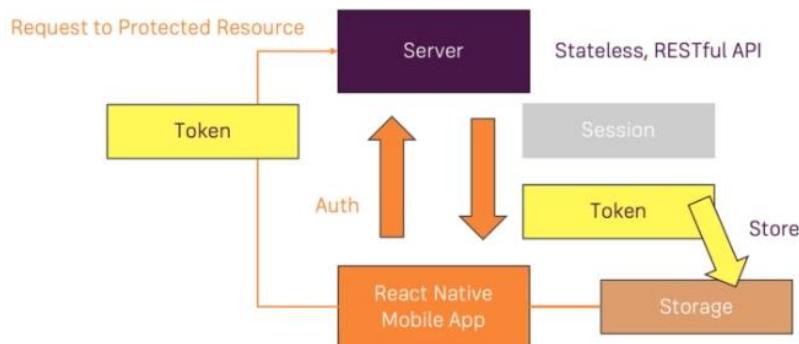
When a user logs in, the server authenticates the action, stores a private key only it knows, and returns a token that must be saved in the app running on the device (on state, AsyncStorage, or any other valid option).

As long as the token is the device, the user is considered logged in. This means, they can travel to private routes they have permissions to. And this is achieved by sending a request to the server meaning to resolve access to the private URL, along with the token.

The server validates the token with the private key, checks it did not expire, and performs other necessary validations to determine if the user is valid. If it is not, it debounces the request with an error, normally handled by the frontend, redirecting to a login screen.

If the request and tokens are both valid, access is granted and the resource can be accessed.

How Authentication Works



Basic auth screens

Note: `createStackNavigator` is specially created to handle Auth cases.
> It adds functionalities like disallowing "go back" when user is not authenticated.

We are to use linear gradient effects, and those come with `expo-linear-gradient` package.

```
$ npm install --save expo-linear-gradient
```

```

Navigation/Shop.jsx

...
const AuthNavigator = createStackNavigator(
  { Auth: AuthScreen },
  { defaultNavigationOptions: sharedDefaultNavOptions }
)

const ShopNavigator = createDrawerNavigator(
{
  Products: ProductsNavigator,
  Orders: OrdersNavigator,
  Admin: AdminNavigator
},
{
  contentOptions: { activeTintColor: colors.PRIMARY },
  contentComponent: (props) => (
    <SafeAreaView style={{ flex: 1, paddingTop: StatusBar.currentHeight }}>
      <DrawerItems {...props} />
    </SafeAreaView>
  )
}
)
)

const MainNavigator = createSwitchNavigator({
  Auth: AuthNavigator, Shop: ShopNavigator
})

export default createAppContainer(MainNavigator)

```

```

screens/user/Auth

import React from "react"
import { ScrollView, StyleSheet, KeyboardAvoidingView, View, Button } from "react-native"
import { LinearGradient } from "expo-linear-gradient"
import Card from "../../UI/Card"
import Input from "../../UI/Input"
import colors from "../../constants/colors"
import * as sharedStyles from "../../constants/styles"

```

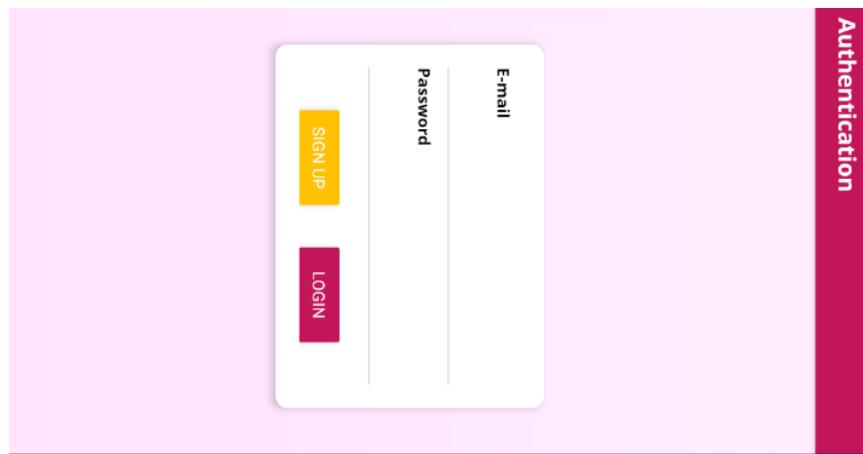
```
export default function Auth(props) {
  return (
    <KeyboardAvoidingView style={_styles.container} behavior="height">
      <LinearGradient
        colors={[ "#ffedff", "#ffe3ff" ]} style={_styles.gradient}>
        <Card style={_styles.card}>
          <ScrollView>
            <Input
              id="email" label="E-mail" keyboardType="email-address"
              required email autoCapitalize="none"
              validationMsg="Enter a valid email address"
              onChangeText={null} initialValue=""
            />
            <Input
              id="password" label="Password"
              keyboardType="default"
              secureTextEntry // hides characters, like ****
              required minLength={5} autoCapitalize="none"
              validationMsg="Enter a valid password"
              onChangeText={null} initialValue=""
            />
            <View style={_styles.buttonsContainer}>
              <View style={_styles.button}>
                <Button title="Sign up" color={colors.SECONDARY}>
                  onPress={null}
                />
              </View>
              <View style={_styles.button}>
                <Button title="Login" color={colors.PRIMARY}>
                  onPress={null} />
              </View>
            </View>
          </ScrollView>
        </Card>
      </LinearGradient>
    </KeyboardAvoidingView>
  )
}

Auth.navigationOptions = { headerTitle: "Authentication" }
```

```

const _styles = StyleSheet.create({
  container: { flex: 1 }, // must grab the whole screen
  gradient: sharedStyles.STRETCH_AND_CENTER, // must stretch!
  card: { width: "80%", maxWidth: 400, maxHeight: 400, padding: 20 },
  buttonsContainer: {
    flexDirection: "row", justifyContent: "space-evenly",
    marginTop: 20
  },
  button: { width: "30%" }
})

```



Adding user signup

Enable Authentication by email in Firebase

Provider	Status	Actions
Email/Password	Enabled	
Phone	Disabled	
Google	Disabled	

To learn how to create/sign in users, google **Firebase auth api**. There are lots of endpoints for many things, we need **Sign in with email/pass**

MISSING_REFRESH_TOKEN: no refresh token provided.

Sign up with email / password

You can create a new email and password user by issuing an HTTP POST request to the Auth `signupNewUser` endpoint.

Method: POST

Content-Type: application/json

Endpoint

```
https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=[API_KEY]
```

Request Body Payload

Property Name	Type	Description
email	string	The email for the user to create.
password	string	The password for the user to create.
returnSecureToken	boolean	Whether or not to return an ID and refresh token. Should always be true.

Response Payload

Property Name	Type	Description
idToken	string	A Firebase Auth ID token for the newly created user.
email	string	The email for the newly created user.
refreshToken	string	A Firebase Auth refresh token for the newly created user.
expiresIn	string	The number of seconds in which the ID token expires.
localId	string	The uid of the newly created user.

Table of contents

- API Usage
- Exchange custom token for an ID and refresh token
- Exchange a refresh token for an ID token
- Sign up with email / password**
- Sign in with email / password
- Sign in anonymously
- Sign in with OAuth credential
- Fetch providers for email
- Send password reset email
- Verify password reset code
- Confirm password reset
- Change email
- Change password
- Update profile
- Get user data
- Link with email/password
- Link with OAuth credential
- Unlink provider
- Send email verification
- Confirm email verification
- Delete account

So, there is the endpoint.

`https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=[API_KEY]`

API_KEY is available in project settings

Your project

Project name: rnmshoppingapp

Project ID: rnmshoppingapp

Project number: 1038498654483

Default GCP resource location: Not yet selected

Web API Key: AlzaSyCYMZ0dYeN1G2i72aG0GboxR6uM-B_Ql7w

That endpoint has **specific requests and response fields**.

Request Body Payload

Property Name	Type	Description
email	string	The email for the user to create.
password	string	The password for the user to create.
returnSecureToken	boolean	Whether or not to return an ID and refresh token. Should always be true.

Response Payload

Property Name	Type	Description
idToken	string	A Firebase Auth ID token for the newly created user.
email	string	The email for the newly created user.
refreshToken	string	A Firebase Auth refresh token for the newly created user.
expiresIn	string	The number of seconds in which the ID token expires.
localId	string	The uid of the newly created user.

Firebase's database starts with global rules where everyone can read or write. This is NOT what we want.

So, let's change the rules to allow anyone to **read** (see products), but only auth users to **write** (add/update/delete products).

```
1 ▾ {  
2 ▾   "rules": {  
3     ".read": "now < 1631847600000",  
4     ".write": "now < 1631847600000",  
5   }  
6 }
```

->

```
{  
  "rules": {  
    ".read": true,  
    ".write": "auth != null",  
  }  
}
```

However, just like that, reading will be allowed but writing will throw an error. That is because even though we log in, firebase is not aware that the user is authenticated due to not passing a valid token to it. We have to leverage on the token.

The screenshot shows the Firebase documentation for the Realtime Database REST API. On the left, there's a sidebar with navigation links for Authentication, Realtime Database (with sub-links for Introduction, Choose a Database, iOS, Android, Web, Admin, and REST), and a 'Authenticate REST Requests' section which is currently selected. The main content area is titled 'Generate an ID token' and provides instructions on how to retrieve a Firebase ID token from a client. It also notes that ID tokens expire after a short period of time. Below this, there's another section titled 'Authenticate with an ID token' which explains how to send authenticated requests to the REST API by passing the ID token in the 'auth' query string parameter. It includes a curl command example:

```
curl "https://<DATABASE_NAME>.firebaseio.com/users/ada/name.json?auth=<ID_TOKEN>"
```

At the bottom of the page, there are two small sections: one about replacing placeholder values and another about successful HTTP status codes.

Store/actions/auth.js

```
import { LOGIN, SIGNUP } from "../actions/auth"

const initialState = { token: null, userId: null }

export default function authReducer(state = initialState, action) {
  switch (action.type) {
    case LOGIN:
    case SIGNUP:
      return { token: action.token, userId: action.userId }
    default:
      return state
  }
}
```

```

Store/reducer/auth.js

const FB_API_KEY = `AIzaSyCYMZ0dYeN1G2i72aG0GboxR6uM-B_Q17w`


export const SIGNUP = "SIGNUP"
export const LOGIN = "LOGIN"

async function authUser(authType, email, password) {
  const fbAuthType =
    authType === "signin" ? "signInWithPassword" : "signUp"

  const response = await fetch(
    `https://identitytoolkit.googleapis.com/v1/
      accounts:${fbAuthType}?key=${FB_API_KEY}`,
    {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body:
        JSON.stringify({ email, password, returnSecureToken: true })
    }
  )

  const data = await response.json()

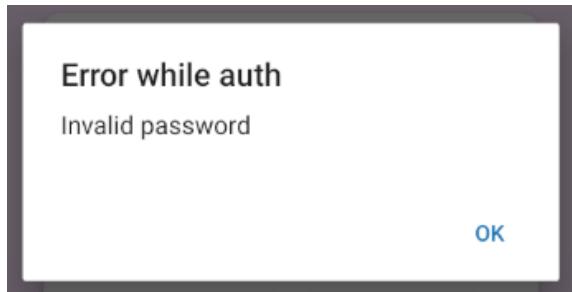
  if (!response.ok) {
    const error = data.error.message
    let message = "Something went wrong"

    // login errors
    if (error === "EMAIL_NOT_FOUND") message = "Invalid email"
    else if (error === "INVALID_PASSWORD") message = "Invalid password"
    // signup errors
    else if (error === "EMAIL_EXISTS") message = "Email already in use"
    else if (error.includes("WEAK_PASSWORD")) message = "Weak password"

    throw new Error(message)
  }

  return data
}

```



```

export const signup = (email, password) => {
  return async (dispatch) => {
    try {
      const data = await authUser("signup", email, password)

      dispatch({
        type: SIGNUP, token: data.idToken, userId: data.localId })
    } catch (err) {
      console.error(err)
      throw err
    }
  }
}

export const login = (email, password) => {
  return async (dispatch) => {
    try {
      const data = await authUser("signin", email, password)

      dispatch({
        type: LOGIN, token: data.idToken, userId: data.localId })
    } catch (err) {
      console.error(err); throw err
    }
  }
}

```

App.js

```

...
import authReducer from "./store/reducers/auth"

const rootReducer = combineReducers({
  products: productsReducer, cart: cartReducer,
  orders: ordersReducer, auth: authReducer
})

const store = createStore(
  rootReducer, composeWithDevTools(applyMiddleware(reduxThunk)))
...

```

Screens/user/Auth.jsx

```
import React, { useState, useCallback, useReducer, useEffect } from "react"
import {
  ScrollView, StyleSheet, KeyboardAvoidingView, View,
  Button, ActivityIndicator, Alert
} from "react-native"
import { LinearGradient } from "expo-linear-gradient"
import { useDispatch } from "react-redux"
import Card from "../../UI/Card"
import Input from "../../UI/Input"
import * as authActions from "../../store/actions/auth"
import colors from "../../constants/colors"
import * as sharedStyles from "../../constants/styles"

const UPDATE_FORM_INPUT = "UPDATE_FORM_INPUT"

const formReducer = (state, action) => {
  if (action.type === UPDATE_FORM_INPUT) {
    const updatedValues = {
      ...state.values, [action.input]: action.value
    }
    const updatedValidations = {
      ...state.validations, [action.input]: action.isInputValid
    }
    let isFormValid = true

    for (const key in updatedValidations)
      isFormValid = isFormValid && updatedValidations[key]

    return {
      values: updatedValues,
      validations: updatedValidations,
      isFormValid
    }
  }
  return state
}

export default function Auth(props) {
  const [isSignup, setIsSignup] = useState(false)
  const [isLoading, setIsLoading] = useState(false)
  const [error, setError] = useState("")
```

```
const dispatch = useDispatch()

const [formState, formDispatch] = useReducer(formReducer, {
  values: { email: "asd@asd.com", password: "asdfgh" },
  validations: { email: false, password: false },
  isFormValid: false
})

useEffect(() => {
  if (error) Alert.alert("Error while auth", error, [{ text: "OK" }])
}, [error])

const handleOnChangeText = useCallback(
  (input, value, isValid) => {
    formDispatch({
      type: UPDATE_FORM_INPUT, value, isValid, input
    })
  ,
  [formDispatch]
)

const handleAuth = async () => {
  setIsLoading(true)
  setError("")
  const { email, password } = formState.values

  try {
    await dispatch(
      authActions[isSignup ? "signup" : "login"](email, password)
    )
    props.navigation.navigate("Shop")
  } catch (err) {
    setError(err.message)
    setIsLoading(false)
  }
}

return (
  <KeyboardAvoidingView style={_styles.container} behavior="height">
```

```

<LinearGradient
  colors={[ "#ffedff" , "#ffe3ff" ]} style={_styles.gradient}>
<Card style={_styles.card}>
  <ScrollView>
    <Input id="email" label="E-mail"
      keyboardType="email-address"
      required email autoCapitalize="none"
      validationMsg="Enter a valid email address"
      value={formState.values.email}
      onChangeText={handleOnchangeText}
      initialValue="" />
    <Input
      id="password" label="Password"
      keyboardType="default"
      secureTextEntry required minLength={6}
      autoCapitalize="none"
      validationMsg="Enter a valid password"
      value={formState.values.password}
      onChangeText={handleOnchangeText}
      initialValue="" />
    <View style={_styles.buttonsContainer}>
      <View style={_styles.button}>
        {isLoading ? (
          <ActivityIndicator
            size="small" color={colors.PRIMARY} />
        ) : (
          <Button
            title={isSignup ? "Go signin" : "Go signup"}
            color={colors.SECONDARY}
            onPress={() => setIsSignup((st) => !st)} />
        ) }
      </View>
      <View style={_styles.button}>
        <Button
          title={isSignup ? "Register" : "Login"}
          color={colors.PRIMARY}
          onPress={handleAuth} />
      </View>
    </View>
  </ScrollView>
</Card>

```

```

        </Card>
    </LinearGradient>
</KeyboardAvoidingView>
)
}

Auth.navigationOptions = { headerTitle: "Authentication" }

const _styles = StyleSheet.create({
  container: { flex: 1 },
  gradient: sharedStyles.STRETCH_AND_CENTER,
  card: { width: "80%", maxWidth: 400, maxHeight: 400, padding: 20 },
  buttonsContainer: {
    flexDirection: "row", justifyContent: "space-evenly",
    marginTop: 20
  },
  button: { width: "40%" }
})

```

Store/actions/products.js

```

...
export const deleteProduct = (id) => {
  return async (dispatch, getState) => {
    try {
      const token = getState().auth.token

      const response = await fetch(
        `https://rnmcshoppingapp-default-rtdb.firebaseio.com/
          products/${id}.json?auth=${token}`,
        { method: "DELETE" }
      )
      ...
    }
  }
}

export const createProduct = (title, description, imageUrl, price) =>
{
  return async (dispatch, getState) => {
    try {
      // second arg incoming from redux thunk is a method to get the
      // current store state. Use it to fetch the token in state.auth
      const token = getState().auth.token
    }
  }
}

```

```

        const response = await fetch(
          `https://rnmcsShoppingApp-default-firebase.firebaseio.com/
            products.json?auth=${token}`,
          {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ title, description, imageUrl, price })
          }
        )
      ...
    }

export const updateProduct = (id, title, description, imageUrl) => {
  return async (dispatch, getState) => {
    try {
      const token = getState().auth.token

      const response = await fetch(
        `https://rnmcsShoppingApp-default-firebase.firebaseio.com/
          products/${id}.json?auth=${token}`,
        {
          method: "PATCH", // 'PUT' overrides, 'PATCH' updates where differs
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ title, description, imageUrl })
        }
      )
      ...
    }
  }
}

export const fetchProducts = () => {
  return async (dispatch) => {
  ...
}

Store/user/orders.js
import Order from "../../models/order"

export const ADD_ORDER = "ADD_ORDER"
export const SET_ORDERS = "SET_ORDERS"

export const fetchOrders = () => {...}

```

```

export const addOrder = (items, total) => {
  return async (dispatch, getState) => {
    const date = new Date()

    try {
      const token = getState().auth.token

      const response = await fetch(
        `https://rnmcshoppingapp-default-firebase.firebaseio.com/
          orders/u1.json?auth=${token}`,
        {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ items, total, date: date.toISOString() })
        }
      )
      ...
    }
  }
}

// on register (signup)
Object {
  "email": "asd@asd.com",
  "expiresIn": "3600",
  "idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjM2NGU4NTQ1NzI50WQ5NzI
    Vlk2wPeP014LSUs36jQmLdfST0BgrTDmCBzCoKD75DdYmvQkh-er
    "kind": "identitytoolkit#SignupNewUserResponse",
    "localId": "6N6tEuJ5z8hMmApk98KdPbkxGwt1",
    "refreshToken": "ACzBnCirAb_MnMquQnhJUcRfR1Szczj0H
  }

// on login (signin)
Object {
  "displayName": "",
  "email": "asd@asd.com",
  "expiresIn": "3600",
  "idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjM2NGU4NTQ1NzI50WQ5NzI
    szxWXeSVeIGH0hltUL92cX6kp8sd5-S8fQftcTfznM2r1B8HgIOCo
    "kind": "identitytoolkit#VerifyPasswordResponse",
    "localId": "6N6tEuJ5z8hMmApk98KdPbkxGwt1",
    "refreshToken": "ACzBnCiqeay1VDRivDz9LEj0pVe4EzaiXR
  "registered": true,
}

// on login 404 error
Object {
  "error": Object {
    "code": 400,
    "errors": Array [
      Object {
        "domain": "global",
        "message": "EMAIL_NOT_FOUND",
        "reason": "invalid",
      },
    ],
    "message": "EMAIL_NOT_FOUND",
  },
}

```

```
// when logged in, data in firebase can be modified.  
// > before, description was Dice!, now Dice!!!1234123
```



Mapping orders to users

Store/actions/orders.js

```
import Order from "../../models/order"  
  
export const ADD_ORDER = "ADD_ORDER"  
export const SET_ORDERS = "SET_ORDERS"  
  
export const fetchOrders = () => {  
  return async (dispatch, getState) => {  
    const userId = getState().auth.userId  
  
    try {  
      const response = await fetch(  
        `https://rnmcshoppingapp-default-rtdb.firebaseio.com/  
        orders/${userId}.json`  
      )  
      ...  
    }  
  
    export const addOrder = (items, total) => {  
      return async (dispatch, getState) => {  
        const date = new Date()  
  
        try {  
          const { token, userId } = getState().auth
```

```

    const response = await fetch(
      `https://rnmcshoppingapp-default-firebase.firebaseio.com/
        orders/${userId}.json?auth=${token}`,
      {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ items, total, date: date.toISOString() })
      }
    )
    ...
  }
}

```

Store/actions/products.js

```

...
export const deleteProduct = (id) => {}

export const createProduct = (title, description, imageUrl, price) =>
{
  return async (dispatch, getState) => {
    const { token, userId } = getState().auth

    const response = await fetch(
      `https://rnmcshoppingapp-default-firebase.firebaseio.com/
        products.json?auth=${token}`,
      {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ title, description, imageUrl, price, userId })
      }
    )

    const data = await response.json()

    dispatch({
      type: CREATE_PRODUCT,
      payload: {
        id: data.name, title, description, imageUrl, price, userId
      }
    }) catch (err) { console.log(err) }
  }
}

```

```

export const updateProduct = ...

export const fetchProducts = () => {
  return async (dispatch, getState) => {
    try {
      const response = await fetch(
        "https://rnmcshoppingapp-default-firebase.firebaseio.com/products.json"
      )

      if (!response.ok) throw new Error("Something went wrong")

      const userId = getState().auth.userId
      const data = await response.json()
      const products = []

      for (const key in data) {
        const values = data[key]
        products.push(
          new Product(
            key, values.userId, values.title,
            values.imageUrl, values.description, values.price
          )
        )
      }
    }

    dispatch({
      type: FETCH_PRODUCTS,
      payload: {
        availableProducts: products,
        userProducts: products.filter((p) => p.userId === userId)
      }
    })
  } catch (err) { console.log(err); throw err }
}
}

```

Screens/shop/Orders.jsx

```

...
export default function Orders(props) {

```

```

const [isLoading, setIsLoading] = useState(true)
const itemsInCart = useSelector((state) => state.orders.items)
const dispatch = useDispatch()

useEffect(() => {
  dispatch(orderActions.fetchOrders()).then(() => setIsLoading(false))
}, [dispatch, setIsLoading])

return (
  <FetchViews isLoading={isLoading}
    responseGate={itemsInCart.length}
    emptyResponseMsg="No items found"
  >
    <FlatList
      data={itemsInCart}
      renderItem={({ item }) => (
        <OrderItem total={item.total} date={item.stringDate}
          items={item.items}
        />
      ) }
    />
  </FetchViews>
)
}
...

```

Screens/user/UserProducts.jsx

```

...
import * as sharedStyles from "../../constants/styles"

export default function UserProducts(props) {
  ...

  return userProducts.length ? (
    <FlatList
      data={userProducts}
      keyExtractor={(item) => item.id}
      renderItem={({ item }) => (
        <ProductItem image={item.imageUrl} title={item.title}

```

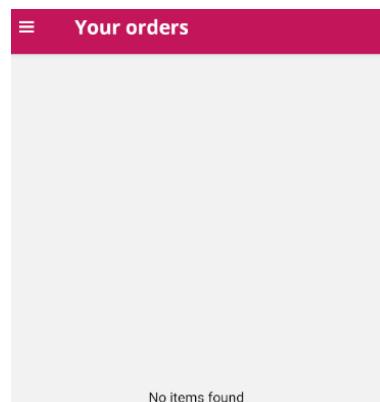
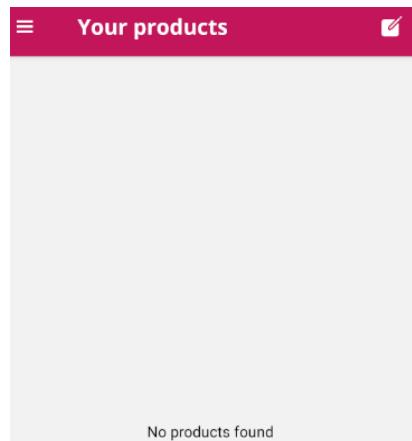
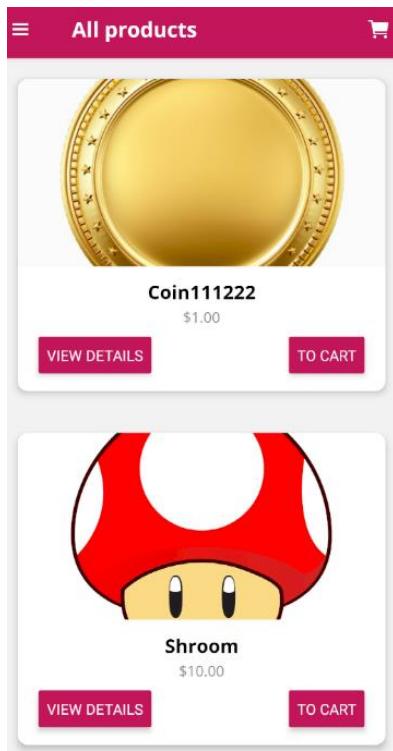
```

        price={item.price} onSelect={() => handleEditProduct(item.id)}
      >
      <Button color={colors.PRIMARY} title="Edit"
        onPress={() => handleEditProduct(item.id)}
      />
      <Button color={colors.PRIMARY} title="Delete"
        onPress={handleDelete.bind(this, item.id)}
      />
    </ProductItem>
  ) }
/>
) : (
<View style={sharedStyles.STRETCH_AND_CENTER}>
  <Text>No products found</Text>
</View>
)
}

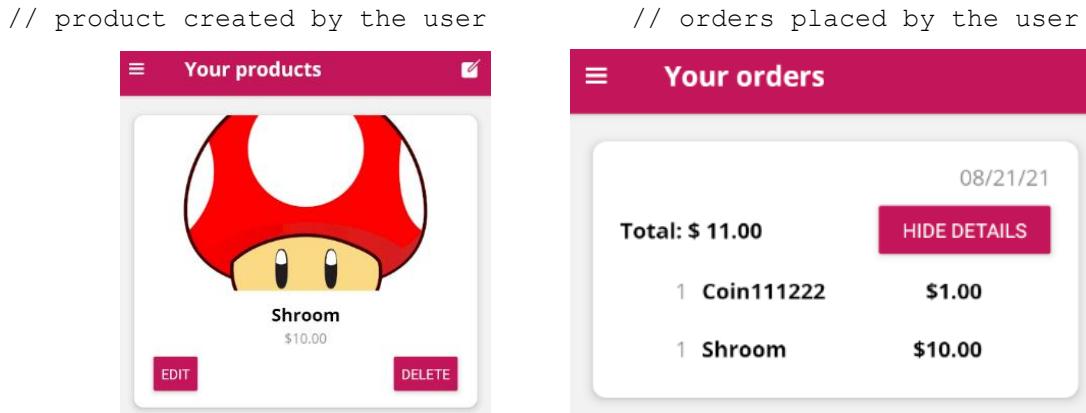
```

```
UserProducts.navigationOptions = ({ navigation }) => ({...})
```

```
// all available products // on no products created by the user
```



```
// on no orders created by the user ->
```



AsyncStorage

It's local storage, but for mobile devices. Let's use it to auto-login users after signing up.

First, install the package.

```
$ expo install @react-native-async-storage/async-storage
```

Navigation/Shop.jsx

```
...
import {StartupScreen} from "../screens/Startup"

// this setup only overrides defaultNavigator options when the active
// screen is used as the top screen of another stack screen
const createSharedNavOptions = (iconName) => ({
  drawerIcon: (drawerConfigs) => (
    <Ionicons
      name={Platform.OS === "android" ? `md-${iconName}` : `ios-${iconName}`}
      size={23} color={drawerConfigs.tintColor} />
  )
})

const sharedDefaultNavOptions = {
  headerStyle: {
    backgroundColor: Platform.OS === "android" ? colors.PRIMARY : "",
  },
  headerTitleStyle: { fontFamily: "open-sans-bold" },
}
```

```
        headerBackTitleStyle: { fontFamily: "open-sans" },
        headerTintColor: Platform.OS === "android" ? "white" : colors.PRIMARY
    }

const ProductsNavigator = createStackNavigator(
{
    ProductsOverview: ProductsOverviewScreen,
    ProductDetails: ProductDetailsScreen,
    Cart: CartScreen
},
{
    initialRouteName: "ProductsOverview",
    navigationOptions: createSharedNavOptions("cart"),
    defaultNavigationOptions: sharedDefaultNavOptions
}
)

const OrdersNavigator = createStackNavigator(
{ Orders: OrdersScreen },
{
    navigationOptions: createSharedNavOptions("list"),
    defaultNavigationOptions: sharedDefaultNavOptions
}
)

const AdminNavigator = createStackNavigator(
{
    UserProducts: UserProductsScreen, EditProduct: EditProductScreen
},
{
    navigationOptions: createSharedNavOptions("create"),
    defaultNavigationOptions: sharedDefaultNavOptions
}
)

const AuthNavigator = createStackNavigator(
{ Auth: AuthScreen },
{ defaultNavigationOptions: sharedDefaultNavOptions }
)

const ShopNavigator = createDrawerNavigator(
```

```

    {
      Products: ProductsNavigator,
      Orders: OrdersNavigator,
      Admin: AdminNavigator
    },
    {
      contentOptions: { activeTintColor: colors.PRIMARY },
      contentComponent: (props) => (
        <SafeAreaView style={{ flex: 1, paddingTop: StatusBar.currentHeight }}>
          <DrawerItems {...props} />
        </SafeAreaView>
      )
    }
  )
}

const MainNavigator = createSwitchNavigator({
  Startup: StartupScreen, Auth: AuthNavigator, Shop: ShopNavigator
})

export default createAppContainer(MainNavigator)

```

store/actions/auth.js

```

import AsyncStorage from "@react-native-async-storage/async-storage"

const FB_API_KEY = `AIzaSyCymZ0dYeN1G2i72aG0GboxR6uM-B_Q17w`

export const AUTHENTICATE = "AUTHENTICATE"
...

export const authenticate = (userId, token) => ({
  type: AUTHENTICATE, userId, token
})

export const signup = (email, password) => {
  return async (dispatch) => {
    try {
      const data = await authUser("signup", email, password)
      dispatch(authenticate(data.localId, data.idToken))
      saveDataToStorage(data.idToken, data.localId, data.expiresIn)
    } catch (err) { console.error(err); throw err }}
}
```

```

export const login = (email, password) => {
  return async (dispatch) => {
    try {
      const data = await authUser("signin", email, password)
      dispatch(authenticate(data.localId, data.idToken))
      saveDataToStorage(data.idToken, data.localId, data.expiresIn)
    } catch (err) { console.error(err); throw err }}
  }

function saveDataToStorage(token, userId, expiresIn) {
  // current time in ms + expiration time in ms, converted to Date()
  const expirationDate = new Date(
    new Date().getTime() + parseInt(expiresIn) * 1000
  ).toISOString()

  return AsyncStorage.setItem(
    "userData", JSON.stringify({ token, userId, expirationDate })
  )
}

```

Store/reducers/auth.js

```

import { AUTHENTICATE } from "../actions/auth"

const initialState = { token: null, userId: null }

export default function authReducer(state = initialState, action) {
  switch (action.type) {
    case AUTHENTICATE:
      return { token: action.token, userId: action.userId }
    default:
      return state
  }
}

```

Screens/Startup.jsx

```

/**
 * Screen to show at load, while checking if user is logged in using
 * AsyncStorage.
 */

```

```

...
export default function Startup(props) {
  const dispatch = useDispatch()

  useEffect(() => {
    const tryAuth = async () => {
      const userData = await AsyncStorage.getItem("userData")

      if (!userData) return props.navigation.navigate("Auth")

      const { token, userId, expirationDate } = JSON.parse(userData)
      // expiration date is an ISO string, we need a date object
      const expiration = new Date(expirationDate)

      // invalid auth if session expired, on no token or no user
      if (expiration <= new Date() || !token || !userId) {
        return props.navigation.navigate("Auth")
      }

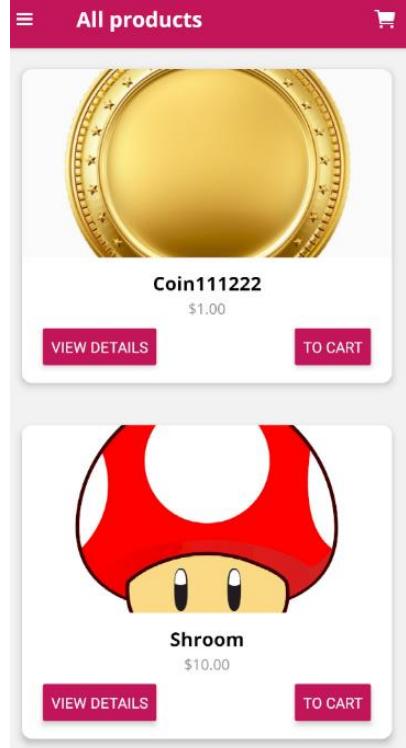
      props.navigation.navigate("Shop")
      dispatch(authActions.authenticate(userId, token))
    }
  })

  tryAuth()
}, [])

return (
  <View style={_styles.container}>
    <ActivityIndicator
      size="large"
      color={colors.PRIMARY} />
  </View>
)
}

const _styles = StyleSheet.create({
  container:
    sharedStyles.STRETCH_AND_CENTER
})

```



Logging out

Store/actions/auth.js

```
import AsyncStorage from "@react-native-async-storage/async-storage"

const FB_API_KEY = `AIzaSyCymZ0dYeN1G2i72aG0GboxR6uM-B_Q17w`

export const AUTHENTICATE = "AUTHENTICATE"
export const LOGOUT = "LOGOUT"

async function authUser(authType, email, password) {
  const fbAuthType = authType === "signin" ? "signInWithPassword" : "signUp"

  const response = await fetch(
    `https://identitytoolkit.googleapis.com/v1/
      accounts:${fbAuthType}?key=${FB_API_KEY}`,
    {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ email, password, returnSecureToken: true })
    }
  )

  const data = await response.json()

  if (!response.ok) {
    const error = data.error.message
    let message = "Something went wrong"

    // login errors
    if (error === "EMAIL_NOT_FOUND") message = "Invalid email"
    else if (error === "INVALID_PASSWORD") message = "Invalid password"
    // signup errors
    else if (error === "EMAIL_EXISTS") message = "Email already in use"
    else if (error.includes("WEAK_PASSWORD")) message = "Weak password"

    throw new Error(message)
  }

  return data
}
```

```
export const authenticate = (userId, token, expirationTime) =>
(dispatch) => {
  dispatch(setLogoutTimeout(expirationTime))
  dispatch({ type: AUTHENTICATE, userId, token })
}

export const signup = (email, password) => {
  return async (dispatch) => {
    try {
      const data = await authUser("signup", email, password)

      dispatch(
        authenticate(
          data.localId, data.idToken, parseInt(data.expiresIn) * 1000
        )
      )

      saveDataToStorage(data.idToken, data.localId, data.expiresIn)
    } catch (err) { console.error(err); throw err }
  }
}

export const login = (email, password) => {
  return async (dispatch) => {
    try {
      const data = await authUser("signin", email, password)

      dispatch(
        authenticate(
          data.localId, data.idToken, parseInt(data.expiresIn) * 1000
        )
      )

      saveDataToStorage(data.idToken, data.localId, data.expiresIn)
    } catch (err) { console.error(err); throw err }
  }
}

let logoutTimeout
```

```

function setLogoutTimeout(expirationTime) {
  return (dispatch) => {
    logoutTimeout = setTimeout(() => {
      dispatch(logout())
    }, expirationTime)
  }
}

function clearLogoutTimeout() {
  logoutTimeout && clearTimeout(logoutTimeout)
}

export const logout = () => {
  clearLogoutTimeout()
  AsyncStorage.removeItem("userData")
  return { type: LOGOUT }
}

function saveDataToStorage(token, userId, expiresIn) {
  // current time in ms + expiration time in ms, converted to Date()
  const expirationDate = new Date(
    new Date().getTime() + parseInt(expiresIn) * 1000
  ).toISOString()

  return AsyncStorage.setItem(
    "userData", JSON.stringify({ token, userId, expirationDate })
  )
}

```

Store/reducer/auth.js

```

import { AUTHENTICATE, LOGOUT } from "../actions/auth"

const initialState = { token: null, userId: null }

export default function authReducer(state = initialState, action) {
  switch (action.type) {
    case AUTHENTICATE:
      return { token: action.token, userId: action.userId }

    case LOGOUT: return initialState
  }
}

```

```

        default: return state
    }
}

}

Screens/Startup.jsx

...
export default function Startup(props) {
    const dispatch = useDispatch()

    useEffect(() => {
        const tryAuth = async () => {
            const userData = await AsyncStorage.getItem("userData")

            if (!userData) return props.navigation.navigate("Auth")

            const { token, userId, expirationDate } = JSON.parse(userData)
            const expiration = new Date(expirationDate)

            if (expiration <= new Date() || !token || !userId) {
                return props.navigation.navigate("Auth")
            }

            const expirationTime =
                expiration.getTime() - new Date().getTime()

            props.navigation.navigate("Shop")
            dispatch(
                authActions.authenticate(userId, token, expirationTime)
            )
        }

        tryAuth()
    }, [])
}

return (
    <View style={_styles.container}>
        <ActivityIndicator size="large" color={colors.PRIMARY} />
    </View>
) } ...

```

Navigation/Shop.jsx

```
...  
  
const ShopNavigator = createDrawerNavigator()  
{  
  Products: ProductsNavigator, Orders: OrdersNavigator,  
  Admin: AdminNavigator  
},  
{  
  contentOptions: { activeTintColor: colors.PRIMARY },  
  contentComponent: (props) => {  
    const dispatch = useDispatch()  
  
    return (  
      <View style={{ flex: 1 }}>  
        <SafeAreaView  
          forceInset={{ top: "always", horizontal: "never" }}  
          style={{ paddingTop: StatusBar.currentHeight }}  
        >  
          {/* `DrawerNavigationItems` on latest React Navigation */}  
          <DrawerItems {...props} />  
          <Button title="logout" color={colors.PRIMARY}  
            onPress={() => {  
              dispatch(authActions.logout())  
              props.navigation.navigate("Auth")  
            }}  
          />  
        </SafeAreaView>  
      </View>  
    )  
  }  
}  
  
const MainNavigator = createSwitchNavigator({  
  Startup: StartupScreen, Auth: AuthNavigator, Shop: ShopNavigator  
})  
  
export default createAppContainer(MainNavigator)
```

```

navigation/NavigationContainer.jsx

import React, { useEffect, useRef } from "react"
import { useSelector } from "react-redux"
import { NavigationActions } from "react-navigation"
import ShopNavigator from "./Shop"

export default function NavigationContainer(props) {
  const navRef = useRef()
  const isAuthenticated = useSelector((state) => Boolean(state.auth.token))

  // this useEffect triggers when `!token`, which happens when the
  // user is logged out from anywhere in the app.
  // > Logout action is handled in reducer. Here we navigate to auth.
  useEffect(() => {
    if (!isAuthenticated) {
      /**
       * A ref attached to a component wrapped with
       * `createAppContainer` gains access to `dispatch`, to fire
       * navigation actions declared there with the respective
       * import from 'react-navigation'!!!! ←
       */
      navRef.current.dispatch(
        NavigationActions.navigate({ routeName: "Auth" }) ←
      )
    }
  }, [isAuthenticated])

  return <ShopNavigator ref={navRef} /> // ref to access nav methods
}

```

App.js

```

...
import NavigationContainer from "./navigation/NavigationContainer"
...
const rootReducer = combineReducers({
  products: productsReducer, cart: cartReducer,
  orders: ordersReducer, auth: authReducer
})

```

```

const store = createStore(
  rootReducer, composeWithDevTools(applyMiddleware(reduxThunk))
)

export default function App() {
  const [isFontLoaded, setIsFontLoaded] = useState(false)

  useEffect(() => {
    fetchFonts()
      .then(() => setIsFontLoaded(true))
      .catch((err) => console.log(err))
  }, [])
}

return isFontLoaded ? (
  <Provider store={store}><NavigationContainer /></Provider>
) : (
  <View style={_styles.container}>
    <ActivityIndicator size="large" />
  </View>
)
}

const _styles = StyleSheet.create({
  container: { flex: 1, alignItems: "center", justifyContent: "center" }
})

function fetchFonts() {
  return Font.loadAsync({
    "open-sans": require("./assets/fonts/OpenSans-Regular.ttf"),
    "open-sans-bold": require("./assets/fonts/OpenSans-Bold.ttf")
  })
}

// on login, expiration time starts // auto-logout on logout or expire finish

```



Auto-logout and Android Warning

With the "Auto Logout" approach implemented in the previous lecture, you'll very likely get a **warning** like this on Android:

Setting a timer for a long period of time, i.e. multiple minutes, is a performance and correctness issue on Android as it keeps the timer module awake, and timers can only be called when the app is in the foreground. See <https://github.com/facebook/react-native/issues/12981>

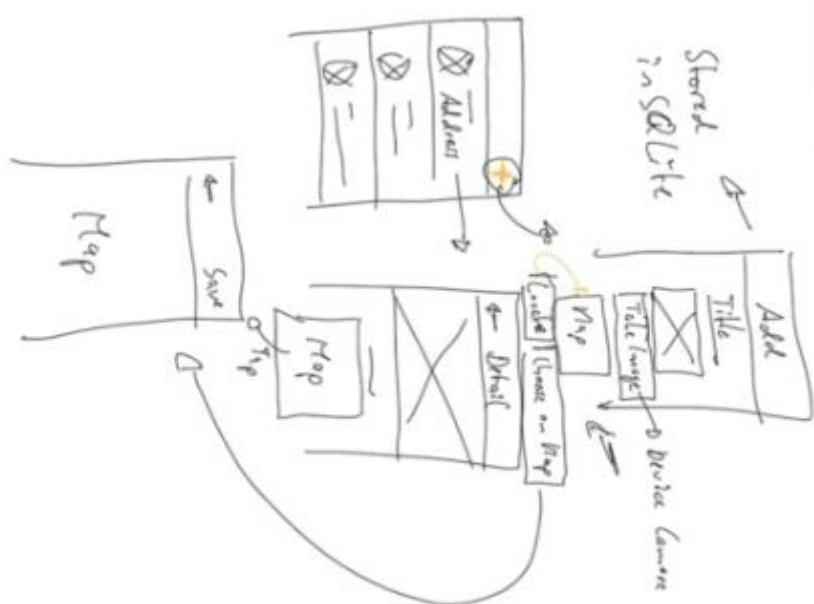
What's the problem?

Using `setTimeout()` with a big timeout duration (as we're setting it \geq 1 hour), can't be handled in the most efficient way by Android.

Unfortunately, there is no perfect solution available though but you can browse the referenced issue thread for possible workarounds:
<https://github.com/facebook/react-native/issues/12981>

Native device features (camera, maps, location, SQLite, File Storage)

Intro



Navigation setup

Navigation/Places.jsx

```
import { createAppContainer } from "react-navigation"
import { Platform } from "react-native"
import { createStackNavigator } from "react-navigation-stack"
import PlacesList from "../screens/PlacesList"
import PlacesDetails from "../screens/PlacesDetails"
import NewPlace from "../screens/NewPlace"
import Map from "../screens/Map"
import colors from "../constants/colors"

const PlacesNavigator = createStackNavigator(
  { PlacesList, PlacesDetails, NewPlace, Map },
  {
    defaultNavigationOptions: {
      headerStyle: {
        backgroundColor: Platform.OS === "android" ? colors.PRIMARY : ""
      },
      headerTintColor:
        Platform.OS === "android" ? "white" : colors.PRIMARY
    }
  }
)

export default createAppContainer(PlacesNavigator)
```

App.js

```
import React from "react"
import { applyMiddleware, combineReducers, createStore } from "redux"
import { Provider } from "react-redux"
import thunk from "redux-thunk"
import { StatusBar } from "react-native"
import PlacesNavigator from "./navigation/Places"
import placesReducer from "./store/reducer/places"

const rootReducer = combineReducers({ places: placesReducer })

const store = createStore(rootReducer, applyMiddleware(thunk))
```

```

export default function App() {
  return (
    <Provider store={store}>
      <PlacesNavigator />
      <StatusBar barStyle="default" />
    </Provider>
  )
}

```

Components/CustomHeaderButton.jsx

```

import React from "react"
import { HeaderButton } from "react-navigation-header-buttons"
import { Ionicons } from "@expo/vector-icons"
import colors from "../constants/colors"
import { Platform } from "react-native"

export default function CustomHeaderButton(props) {
  return (
    <HeaderButton {...props} IconComponent={Ionicons} iconSize={23}
      color={Platform.OS === "android" ? "white" : colors.PRIMARY}
    />
  )
}

```

Components/CustomHeaderButtons.jsx

```

import React from "react"
import { Platform } from "react-native"
import { HeaderButtons, Item } from "react-navigation-header-buttons"
import CustomHeaderButton from "./CustomHeaderButton"

export default function CustomHeaderButtons({ title, iconName, onPress }) {
  return (
    <HeaderButtons HeaderButtonComponent={CustomHeaderButton}>
      <Item
        iconName={
          Platform.OS === "android" ? `md-${iconName}` : `ios-${iconName}`
        }
        {...{ title, onPress }}
      />
  )
}

```

```

        </HeaderButtons>
    )
}

Components/PlacesItem.jsx

import React from "react"
import { View, Text, Image, StyleSheet, TouchableOpacity } from "react-native"
import colors from "../constants/colors"

export default function PlaceItem({ onSelect, image, title, address }) {
    return (
        <TouchableOpacity onPress={onSelect} style={_styles.placeItem}>
            <Image style={_styles.image} source={{ uri: image }} />
            <View style={_styles.infoContainer}>
                <Text style={_styles.title}>{title}</Text>
                <Text style={_styles.address}>{address}</Text>
            </View>
        </TouchableOpacity>
    )
}

const _styles = StyleSheet.create({
    placeItem: {
        borderBottomColor: "#ccc", borderBottomWidth: 1,
        paddingVertical: 15, paddingHorizontal: 30,
        flexDirection: "row", alignItems: "center"
    },
    image: {
        width: 70, height: 70, borderRadius: 35, backgroundColor: "#ccc",
        borderColor: colors.primary, borderWidth: 1
    },
    infoContainer: {
        marginLeft: 25, width: 250, justifyContent: "center",
        alignItems: "flex-start"
    },
    title: { color: "black", fontSize: 18, marginBottom: 5 },
    address: { color: "#666", fontSize: 16 }
})

```

Screens/PlacesList.jsx

```
import React from "react"
import { StyleSheet, FlatList } from "react-native"
import { useSelector } from "react-redux"
import CustomHeaderButtons from "../components/CustomHeaderButtons"
import PlaceItem from "../components/PlaceItem"

export default function PlacesList(props) {
  const places = useSelector((state) => state.places.items)

  return (
    <FlatList
      data={places}
      keyExtractor={(item) => item.id}
      renderItem={({ item }) => (
        <PlaceItem title={item.title} image={null} address={null}
        onSelect={() =>
          props.navigation.navigate("PlaceDetails", {
            title: item.title, id: item.id
          })
        } />
      )} />
  )
}

PlacesList.navigationOptions = ({ navigation }) => ({
  headerTitle: "All places",
  headerRight: () => (
    <CustomHeaderButtons title="Add place" iconName="add"
    onPress={() => navigation.navigate("NewPlace")}
  />
)
})
```

const _styles = StyleSheet.create({ container: {} })

screens/NewPlace.jsx

```
import React, { useState } from "react"
import { View, Text, TextInput, ScrollView, StyleSheet, Button
```

```

} from "react-native"

import { useDispatch } from "react-redux"
import * as placesActions from "../store/actions/places"
import colors from "../constants/colors"

export default function NewPlace(props) {
  const [title, setTitle] = useState("")
  const dispatch = useDispatch()

  const handleSave = () => {
    dispatch(placesActions.addPlace(title))
    props.navigation.goBack()
  }

  return (
    <ScrollView>
      <View style={_styles.container}>
        <Text style={_styles.label}>Title</Text>
        <TextInput value={title} onChangeText={setTitle}
          style={_styles.input}
        />
        <Button
          title="Save" color={colors.PRIMARY} onPress={handleSave} />
      </View>
    </ScrollView>
  )
}

NewPlace.navigationOptions = { headerTitle: "Add place" }

const _styles = StyleSheet.create({
  container: { margin: 30 },
  label: { fontSize: 18, marginBottom: 15 },
  input: {
    borderBottomColor: "#ccc", borderBottomWidth: 1,
    marginBottom: 15, paddingVertical: 5,
    paddingHorizontal: 2
  }
})

```

models/place.js

```
export default class Place {
  constructor(id, title) { this.id = id; this.title = title }
}
```

Store/actions/place.js

```
export const ADD_PLACE = "ADD_PLACE"

export const addPlace = (title) =>
  ({ type: ADD_PLACE, payload: { title } })
```

Store/reducer/place.js

```
import Place from "../../models/place"
import { ADD_PLACE } from "../actions/places"

const initialState = { items: [] }

export default function placesReducer(state = initialState, action) {
  switch (action.type) {
    case ADD_PLACE:
      return {
        items: state.items.concat(
          new Place(new Date().toString(), action.payload.title)
        )
      }
    default: return state
  }
}
```

Screens/PlaceDetails.jsx

```
import React from "react"
import { View, Text, StyleSheet } from "react-native"

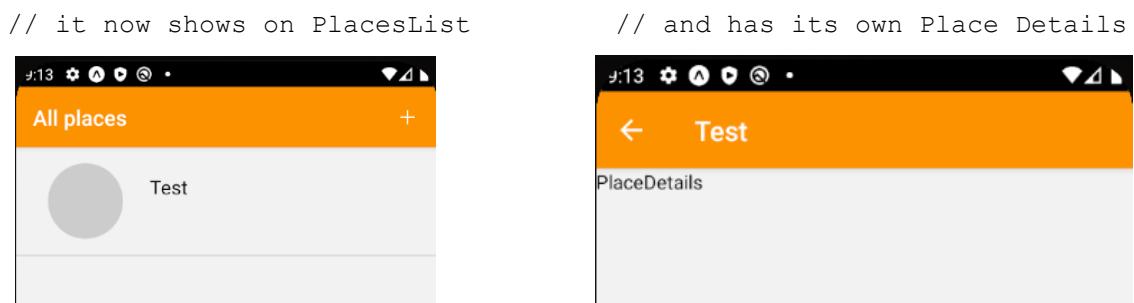
export default function PlaceDetails(props) {
  return <View><Text>PlaceDetails</Text></View>
}
```

```

PlaceDetails.navigationOptions = ({ navigation }) => ({
  headerTitle: navigation.getParam("title")
})

const _styles = StyleSheet.create({ container: {} })

```



Device camera

Expo wraps React Native and adds loads of functionalities. The docs explains the built in modules and features, with no manual configurations other than the ones required by expo (no device configs of coding from scratch).

Even though [Camera](#) module is supported, we will use [ImagePicker](#), which allows us to access image gallery (which also includes the ability to take a photo).

```
$ expo install expo-image-picker
```

Expo also has a convenient [Permissions](#) package, used to ask for permissions to access device functionalities (like camera, contacts, location) at runtime.

```
$ expo install expo-permissions
```

Permissions.CAMERA is the device's camera, while **MEDIA_LIBRARY** is the gallery. ImagePicker requires both.

Note: Once permission is tested in Expo, you need to re-launch expo app to clear given permissions.

Warning!

Expo permissions is deprecated! Functionality was divided into different packages that handle device features (like *expo-camera*, *expo-location*)

Components/ImagePicker.jsx

```
import React, { useState } from "react"
import { Button, View, Text, Image, StyleSheet, Alert } from "react-native"
import * as ImagePicker from "expo-image-picker"
import * as Permissions from "expo-permissions"
import colors from "../constants/colors"

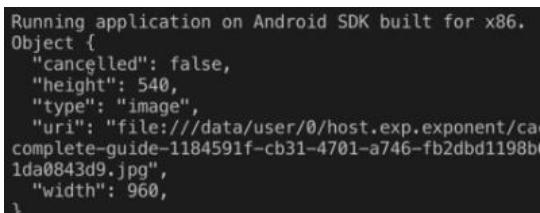
export default function ImgPicker({ onImageTaken }) {
  const [PickedImageUri, setPickedImageUri] = useState(null)

  const handleTakeImage = async () => {
    const hasPermission = await verifyPermission()

    if (!hasPermission) return

    const image = await ImagePicker.launchCameraAsync({
      allowsEditing: true, // cropping enabled
      aspect: [16, 9], // aspect ratio
      quality: 0.5 // quality (from 0 to 1)
    })
    // method return value
    setPickedImageUri(image.uri)
    onImageTaken?.(image.uri)
  }

  return (
    <View style={_styles.container}>
      <View style={_styles.preview}>
        {PickedImageUri ? (
          <Image
```



The screenshot shows the application running on an Android emulator. It displays a single image preview in the center of the screen. The image is a small portrait of a person's face. Below the image, there is some text output from the console log, which includes the file path and dimensions of the image.

```
Running application on Android SDK built for x86.
Object {
  "cancelled": false,
  "height": 540,
  "type": "image",
  "uri": "file:///data/user/0/host.exp.exponent/callcomplete-guide-1184591f-cb31-4701-a746-fb2dbd1198b1da0843d9.jpg",
  "width": 960,
```

```

        source={{ uri: PickedImageUri }} style={_styles.image} />
      ) : ( <Text>No image yet</Text> )
    </View>
    <Button title="Take pic" color={colors.PRIMARY}>
      onPress={handleTakeImage}
    />
  </View>
)
}

const _styles = StyleSheet.create({
  container: { alignItems: "center" },
  preview: {
    width: "100%", height: 200, marginBottom: 10,
    justifyContent: "center", alignItems: "center",
    borderColor: "#ccc", borderWidth: 1
  },
  image: { width: "100%", height: "100%" }
})

// Note: permissions are asked only once, even if the function is
// re-triggered. You need to re-launch expo.
async function verifyPermission() {
  const result = await Permissions.askAsync(
    Permissions.CAMERA, Permissions.MEDIA_LIBRARY
  )

  if (result.status !== "granted") {
    Alert.alert("Failed access", "Camera permission is required",
      [{ text: "OK" }])
    return false
  }
  return true
}

```

Screens/NewPlace.jsx

```

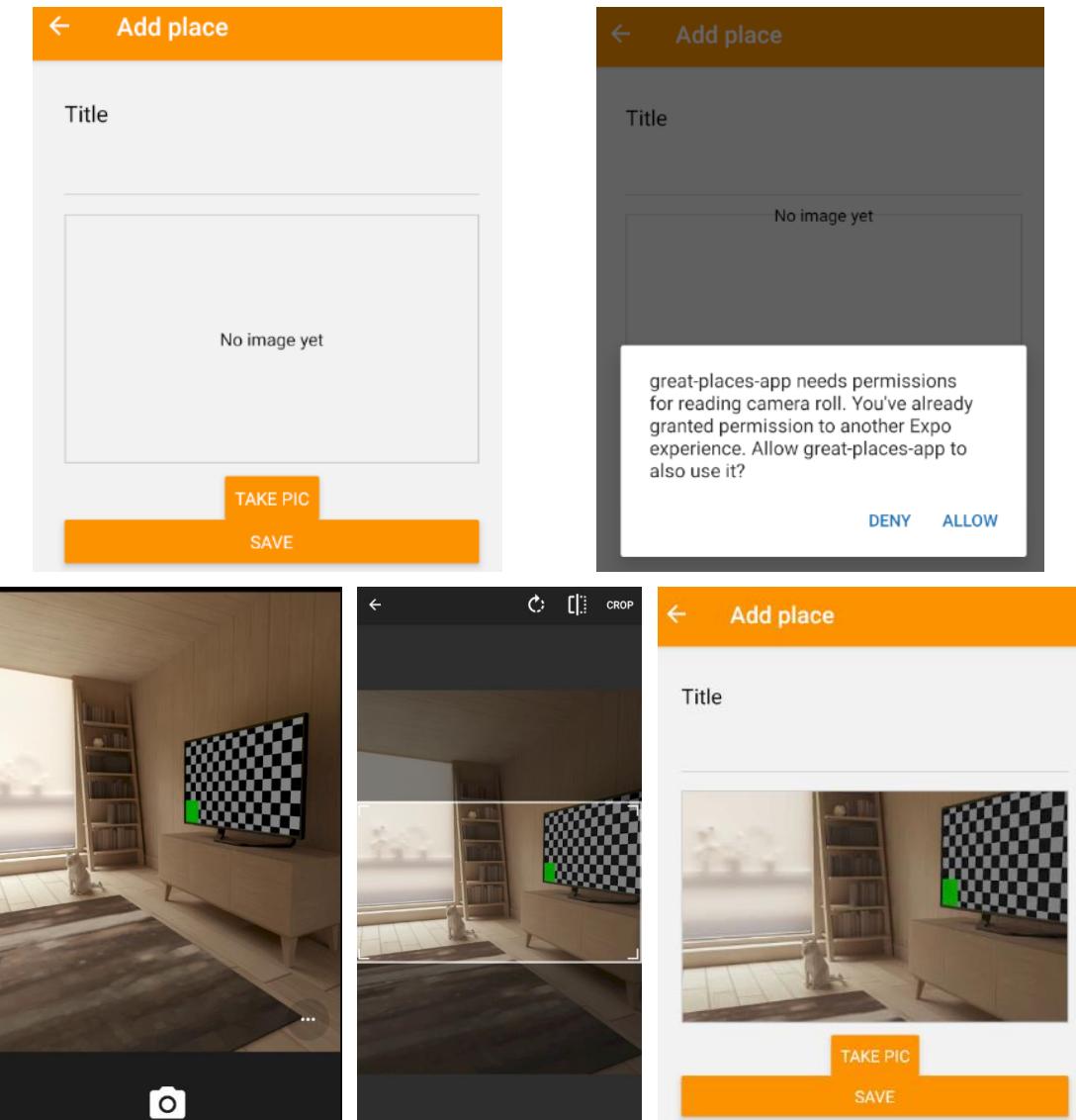
...
import ImagePicker from "../components/ImagePicker"
export default function NewPlace(props) {
...

```

```

        return (
      <ScrollView>
        <View style={_styles.container}>
          <Text style={_styles.label}>Title</Text>
          <TextInput value={title} onChangeText={setTitle}
            style={_styles.input}
          />
          <ImagePicker />
          <Button
            title="Save" color={colors.PRIMARY} onPress={handleSave} />
        </View>
      </ScrollView>
    )
  }
...

```



Using the picked image

Models/Place.js

```
export default class Place {
  constructor(id, title, imageUri) {
    this.id = id; this.title = title; this.imageUri = imageUri
  }
}
```

Store/actions/place.js

```
export const ADD_PLACE = "ADD_PLACE"

export const addPlace = (title, imageUri) => ({
  type: ADD_PLACE, payload: { title, imageUri }
})
```

Store/reducer/places.js

```
import Place from "../../models/place"
import { ADD_PLACE } from "../actions/places"

const initialState = { items: [] }

export default function placesReducer(state = initialState, action) {
  switch (action.type) {
    case ADD_PLACE:
      return {
        items: state.items.concat(
          new Place(
            new Date().toString(),
            action.payload.title,
            action.payload.imageUri
          )
        )
      }
    default: return state
  }
}
```

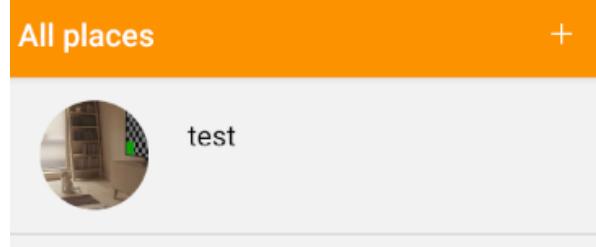
Screens/NewPlace.jsx

```
...
export default function NewPlace(props) {
  const [title, setTitle] = useState("")
  const [selectedImage, setSelectedImage] = useState(null)
  const dispatch = useDispatch()

  const handleImageTaken = (imageUri) => setSelectedImage(imageUri)

  const handleSave = () => {
    dispatch(placesActions.addPlace(title, selectedImage))
    props.navigation.goBack()
  }

  return (
    <ScrollView>
      <View style={_styles.container}>
        <Text style={_styles.label}>Title</Text>
        <TextInput value={title} onChangeText={setTitle}
          style={_styles.input} />
        <ImagePicker onImageTaken={handleImageTaken} />
        <Button title="Save"
          color={colors.PRIMARY} onPress={handleSave} />
      </View>
    </ScrollView>
  )
}
```



Screens/PlacesList.jsx

```
...
export default function PlacesList(props) {
  const places = useSelector((state) => state.places.items)

  return (
    <FlatList data={places} keyExtractor={(item) => item.id}
      renderItem={({ item }) => (
        <PlaceItem
          title={item.title} image={item.imageUri} address={null}
        </PlaceItem>
      )}
    )
}
```

```

        onSelect={() =>
          props.navigation.navigate("PlaceDetails", {
            title: item.title, id: item.id, imageUri: item.imageUri
          })
        } />
      ) } />
    )
}

```

Storing image (FileSystem and SQLite)

Expo got us covered with native file system storage, with expo-file-system package. We will use it to store the taken images.

\$ expo install expo-file-system

We cannot simply store images in cache or memory, since both will be wiped out when app resets. Hence, we need a more permanent solution, which for us will come in shape of SQLite, a light-weight database capable of communicating with the app and the device.

\$ expo install expo-sqlite

When we first instantiate SQLite in the app, the database will be created before connecting to it. Further calls will just connect. Afterwards, we can execute SQL queries.

Storage/db.js

```

import * as SQLite from "expo-sqlite"

const db = SQLite.openDatabase("places.db")

export function initializeDatabase() {
  return new Promise((resolve, reject) => {
    db.transaction(tx => {
      tx.executeSql(
        "CREATE TABLE IF NOT EXISTS places
        (id INTEGER PRIMARY KEY NOT NULL, title TEXT NOT NULL,
        imageUri TEXT NOT NULL, address TEXT NOT NULL, lat REAL NOT
        NULL, lng REAL NOT NULL);",
        [], // args
      )
    })
  })
}

export function getPlaces() {
  return db.transaction(tx => {
    tx.executeSql("SELECT * FROM places")
  })
}

export function insertPlace(place) {
  return db.transaction(tx => {
    tx.executeSql("INSERT INTO places (title, imageUri, address, lat, lng) VALUES (?, ?, ?, ?, ?)", [place.title, place.imageUri, place.address, place.lat, place.lng])
  })
}

```

```

        (executedQuery, result) => {
            // success callback
            resolve()
        },
        (executedQuery, err) => {
            // error callback
            reject(err)
        }
    )
}
}

export function insertPlace(title, imageUri, address, lat, lng) {
    return new Promise((resolve, reject) => {
        db.transaction((tx) => {
            tx.executeSql(
                `INSERT INTO places (title, imageUri, address, lat, lng)
                VALUES (?, ?, ?, ?, ?);`,
                [title, imageUri, address, lat, lng],
                (_, result) => resolve(result),
                (_, err) => reject(err)
            )
        })
    })
}

export function getAllPlaces() {
    return new Promise((resolve, reject) => {
        db.transaction((tx) => {
            tx.executeSql(
                "SELECT * FROM places",
                [],
                (_, result) => resolve(result),
                (_, err) => reject(err)
            )
        })
    })
}

```

App.js

```
...
import { initializeDatabase } from "./storage/db"

initializeDatabase()
  .then(() => console.log("SQLite initialized."))
  .catch((err) => console.log("SQLite initialization failed.", err))
...
...
```

Store/actions/places.js

Store/actions/places.js

```
import * as FileSystem from "expo-file-system"
import { getAllPlaces, insertPlace } from "../../storage/db"

export const ADD_PLACE = "ADD_PLACE"
export const LOAD_PLACES = "LOAD_PLACES"

export const addPlace = (title, imageUri) => async (dispatch) => {
  // split path and grab the image file name with extension
  const fileName = imageUri.split("/") .pop()

  // `FS.cacheDirectory`, `FS.bundleDirectory`, `FS.documentDirectory`
  // > The last one being the permanent one for the app
  const newPath = FileSystem.documentDirectory + fileName

  try {
    // move the file from its cache to its permanent path in memory.
    // > it can fail! (e.g. no space). Hence trycatch.
    await FileSystem.moveAsync({ from: imageUri, to: newPath })

    // storing data in DB can also fail
    const res =
      await insertPlace(title, newPath, "dummy address", 11.11, 22.22)
```

```

        dispatch({
          type: ADD_PLACE,
          payload: { id: res.insertId, title, pathToImg: newPath }
        })
      } catch (err) { console.log(err); throw err }
    }

export const loadPlaces = () => async (dispatch) => {
  try {
    const res = await getAllPlaces()

    dispatch({ type: LOAD_PLACES, payload: { places: res.rows._array } })
  } catch (err) { console.log(err); throw err }
}

```

Store/reducer/places.js

```

import Place from "../../models/place"
import { LOAD_PLACES, ADD_PLACE } from "../actions/places"

const initialState = { items: [] }

export default function placesReducer(state = initialState, action) {
  switch (action.type) {
    case LOAD_PLACES:
      return {
        items: action.payload.places.map(
          (p) => new Place(p.id.toString(), p.title, p.imageUri)
        )
      }

    case ADD_PLACE:
      return {
        items: state.items.concat(
          new Place(
            action.payload.id.toString(),
            action.payload.title, action.payload.pathToImg
          )
        )
      }
  }
}

```

```

        default:  return state
    }
}

}

Screens/PlacesList.jsx
...
import * as placesActions from "../store/actions/places"

export default function PlacesList(props) {
    const places = useSelector((state) => state.places.items)
    const dispatch = useDispatch()

    useEffect(() => {dispatch(placesActions.loadPlaces()) , [dispatch]}

    return (
        <FlatList
            data={places} keyExtractor={(item) => item.id}
            renderItem={({ item }) =>
                <PlaceItem title={item.title} image={item.imageUri}
                    address={null}
                    onSelect={() =>
                        props.navigation.navigate("PlaceDetails", {
                            title: item.title, id: item.id, imageUri: item.imageUri
                        })
                    } />
            )} />
    )}

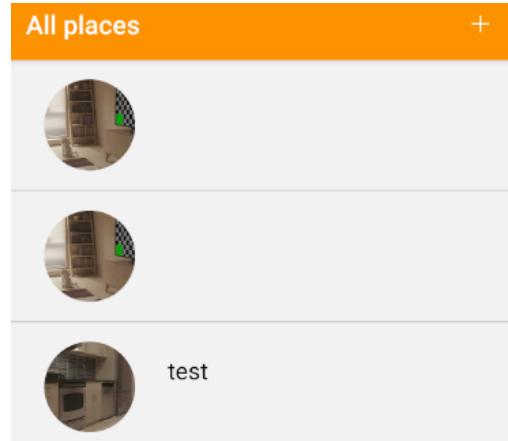
PlacesList.navigationOptions = ({ navigation }) => ({
    headerTitle: "All places",
    headerRight: () => (
        <CustomHeaderButtons title="Add place" iconName="add"
            onPress={() => navigation.navigate("NewPlace")}>
        )
    )
})

const _styles = StyleSheet.create({ container: {} })

```

```
// on app load, getAllPlaces() db method is called, which returns:
Running application on Android SDK built for x86.
SQLite initialized.
WebSQLResultSet {
  "insertId": undefined,
  "rows": WebSQLRows {
    "_array": Array [
      Object {
        "address": "dummy address",
        "id": 1,
        "imageUri": "file:///data/user/0/host.exp.exponent/eat-places-app/1069a23c-cf66-4cd9-b0fa-70c57253599c.jpg",
        "lat": 11.11,
        "lng": 22.22,
        "title": ""
      },
      Object {
        "address": "dummy address",
        "id": 2,
        "imageUri": "file:///data/user/0/host.exp.exponent/eat-places-app/46829eba-50f3-4fbb-b438-b7b9ee02d359.jpg",
        "lat": 11.11,
        "lng": 22.22,
        "title": ""
      }
    ],
    "length": 2,
  },
  "rowsAffected": 0,
}

// on data insertion with insertPlace(), result is ^
WebSQLResultSet {
  "insertId": 2,
  "rows": WebSQLRows {
    "_array": Array [],
    "length": 0,
  },
  "rowsAffected": 1,
}
```



Getting the user's location

To get user's location, expo has expo-location.

\$ expo install expo-location

And for static maps, nothing better than Google Maps itself.

<https://developers.google.com/maps/documentation/maps-static/overview>

There you will find a link you can copy, modify and paste to get a map location. However, you need an API key, which can be got as a free trial in "GET STARTED" section.

There, enable Map Static API and Places API.

The screenshot shows the Google Maps Platform API console. On the left, there's a sidebar with links like Overview, APIs, Metrics, Quotas, Credentials, and Support. The 'APIs' link is highlighted. In the main area, there's a heading 'Authentication, quotas, pricing, and policies'. Below it, under 'Authentication', there's a note: 'To use the Maps Static API, you must first enable the API and obtain the proper authentication credentials. For more information, see [Get Started with Google Maps Platform](#)'. To the right, there's a section titled 'Enabled APIs' with a dropdown menu showing 'Maps Static API' and 'Places API'.

Generate an API key going to Google Maps > Credentials > Create credentials

The screenshot shows the Google Cloud Platform Credentials page for the 'Maps Static API'. The 'Credentials' tab is selected. A modal dialog box is open, titled 'API key created', containing the message: 'Use this key in your application by passing it with the key=API_KEY parameter.' It shows a 'Your API key' field with the value 'AIzaSyBsF_hs0RXI6WN1EJeYo-uPehEBjw4Y2HI'. There are buttons for 'CLOSE' and 'RESTRICT KEY'. Below the modal, there's a 'Secret Generator' section with a note about URL signing secrets and a 'REGENERATE SECRET' button. The background shows the 'Credentials compatible with this API' section and a 'CONFIGURE CONSENT SCREEN' button.

Copy the big URL presented at the start of the page, and replace the api placeholder with your key.

https://maps.googleapis.com/maps/api/staticmap?center=Brooklyn+Bridge, New+York,NY&zoom=13&size=600x300&maptype=roadmap&markers=color:blue%7Clabel:S%7C40.702147,-74.015794&markers=color:green%7Clabel:G%7C40.711614,-74.012318&markers=color:red%7Clabel:C%7C40.718217,-73.998284&key=YOUR_API_KEY

Env.js

```
const API_KEYS = { GOOGLE: "your_api_key" }
```

```
export default API_KEYS
```

components/MapPreview.jsx

```
import React from "react"
import { Image, View, StyleSheet } from "react-native"
import API_KEYS from "../env"

export default function MapPreview({ location, children, style }) {
  const imgPreviewUrl = !location.lat
    ? ""
    : `https://maps.googleapis.com/maps/api/staticmap?
      center=${location.lat},${location.lng}&zoom=13&size=400x200
      &maptype=roadmap&markers=color:red%7Clabel:A%7C${location.lat},
      ${location.lng}&key=${API_KEYS.GOOGLE}`

  return (
    <View style={[_styles.container, style]}>
      {imgPreviewUrl ? (
        <Image
          source={{ uri: imgPreviewUrl }} style={_styles.image} />
      ) : (
        children
      )}
    </View>
  )
}

const _styles = StyleSheet.create({
  container: { justifyContent: "center", alignItems: "center" },
  // network images (async) needs 100% width and height
  image: { width: "100%", height: "100%" }
})
```

Components/LocationPicker.jsx

```
import React, { useState } from "react"
import { View, Button, Text, ActivityIndicator, Alert, StyleSheet
} from "react-native"
import MapPreview from "./MapPreview"
import * as Location from "expo-location"
import colors from "../constants/colors"

export default function LocationPicker(props) {
  const [pickedLocation, setPickedLocation] =
    useState({ lat: 0, lng: 0 })
  const [isFetching, setIsFetching] = useState(false)

  const handleGetLocation = async () => {
    const hasPermission = await verifyPermission()

    if (!hasPermission) return

    try {
      setIsFetching(true)

      const res =
        await Location.getCurrentPositionAsync({ timeInterval: 5000 })

      setPickedLocation(
        { lat: res.coords.latitude, lng: res.coords.longitude })
    } catch (err) {
      Alert.alert(
        "Could not fetch location",
        "Try again or pick location from map",
        [{ text: "Ok" }])
    }
    console.log(err); throw err
  }

  setIsFetching(false)
}

return (
  <View style={_styles.container}>
    <MapPreview location={pickedLocation} style={_styles.preview}>
      {isFetching ? (

```

```

        <ActivityIndicator size="large" color={colors.PRIMARY} />
    ) : (
        <Text>No location</Text>
    )}

</MapPreview>
<Button title="Get location" color={colors.PRIMARY}
    onPress={handleGetLocation}
/>
</View>
)
}

const _styles = StyleSheet.create({
  container: { marginBottom: 15 },
  preview: {
    marginBottom: 10, width: "100%", height: 150,
    borderColor: "#ccc", borderWidth: 1
  }
})

async function verifyPermission() {
  const result = await Location.requestForegroundPermissionsAsync()

  if (result.status !== "granted") {
    Alert.alert("Failed access", "Location permission is required", [
      { text: "OK" }
    ])
    return false
  }
  return true
}

```

Screens/NewPlace.jsx

```

...
import LocationPicker from "../components/LocationPicker"
...
export default function NewPlace(props) {
...
  return (

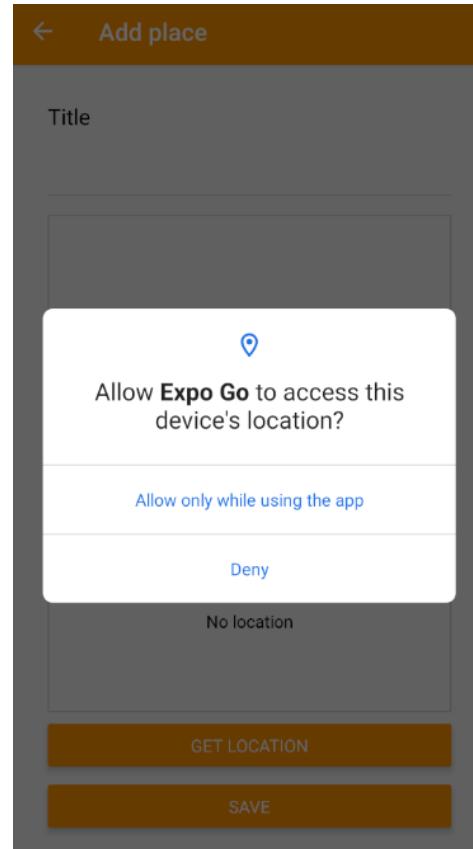
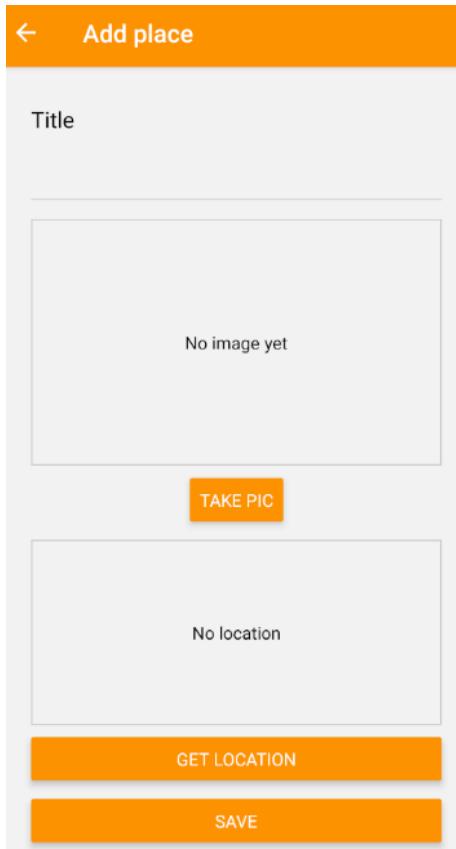
```

```

<ScrollView>
  <View style={_styles.container}>
    <Text style={_styles.label}>Title</Text>
    <TextInput value={title} onChangeText={setTitle}
      style={_styles.input}>
    />
    <ImagePicker onImageTaken={handleImageTaken} />
    <LocationPicker />
    <Button title="Save"
      color={colors.PRIMARY} onPress={handleSave} />
  </View>
</ScrollView>
)
}
...

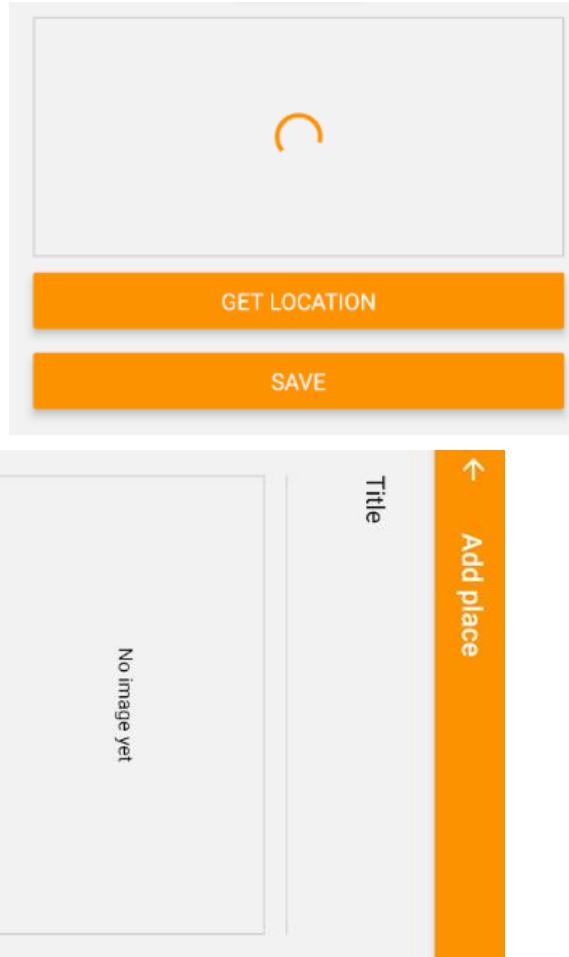
```

// on request location permissions



```
// location response and getting location
```

```
Object {
  "coords": Object {
    "accuracy": 603,
    "altitude": 0,
    "altitudeAccuracy": 0,
    "heading": 0,
    "latitude": 37.4219861,
    "longitude": -122.0840279,
    "speed": 0,
  },
  "mocked": false,
  "timestamp": 1629763369787,
}
```



More on environment variables

In the previous lecture, we created a basic environment variables file (`env.js`).

This basic file just exports a JS object - but you could get more fancy and for example export different environment variables for your development flow (i.e. for testing/ developing your app) and for production (i.e. for when you publish your app).

The special `_DEV_` global variable offered by Expo helps you - it's a variable which you can always access anywhere in your Expo-driven React Native project to determine whether you're running this app in development mode or not.

Therefore, you could create a more elaborate environment variables file like this one:

```
env.js
const variables = {
  development: { googleApiKey: 'abc' },
  production: { googleApiKey: 'xyz' }
}

const getEnvVariables = () => {
  // return this if in development mode
  if (__DEV__) return variables.development

  return variables.production // otherwise, return this
}

export default getEnvVariables // export a function reference
```

You would use that file like this:

someOtherFile.js

```
import ENV from './env'
...
const apiKey = ENV().googleApiKey
```

Displaying an interactive map

Even though Google Maps API is more than enough to handle displaying map, **react-native-maps** is the de-facto solution to interactive map functionality.

```
$ expo install react-native-maps
```

And to translate google locations to a human-readable format, we can use google's geocoding API.

It's reverse geocoding section (get address) is the one we need

The screenshot shows the Google Maps Platform Documentation page for "Reverse geocoding (address lookup)". The left sidebar contains links for Overview, Get Started, Set up in Cloud Console, Using API Keys, Best Practices Geocoding Addresses, Geocoding FAQ, Web Services, Best Practices, Client Libraries, Policies and Terms, Usage and Billing, and Optimizing Quota Usage. The main content area has a heading "Reverse geocoding (address lookup)" and a paragraph explaining what geocoding is. It lists required parameters (latlng, key), optional parameters, and a note about viewport biasing. A table of contents on the right includes links for What is geocoding?, Before you begin, Geocoding API request format, Geocoding (latitude/longitude lookup), Geocoding responses, Status codes, Error messages, Results, Address types and address component types, Viewport biasing, Region biasing, Component filtering, and Reverse geocoding (address lookup).

The endpoint is at the bottom of the section:

```
https://maps.googleapis.com/maps/api/geocode/json?latlng=40.714224,-73.961452&key=YOUR_API_KEY
```

You can pass a bunch of configurations. **Check the docs.**

For it to work, you must enable it in Google Cloud.

The screenshot shows the Google Cloud Platform APIs page for Google Maps Platform. The left sidebar has sections for Overview, APIs (which is selected), Metrics, Quotas, Credentials, and Support. The main content area is titled "Enabled APIs" and says "Select an API to view details. Figures are for the last 30 days." It shows a list of APIs: Geocoding API, Maps Static API, and Places API. Each API has a "View Details" button.

The returned object consists of { plus_code, results, status }
 > "status" key being a string with the request status.

```
  ▼ plus_code:
    compound_code:      "P27Q+MC New York, NY, USA"
    global_code:        "87G8P27Q+MC"
  ▼ results:
    ▼ 0:
      ▼ address_components:
        ▼ 0:
          long_name:      "277"
          short_name:     "277"
        ▼ types:
          0:            "street_number"
      ▼ 1:
        long_name:      "Bedford Avenue"
```

Each object in "results" holds "**formatted_address**" key, the display name for the address, our target to show in the app.

Models/places.js

```
export default class Place {
  constructor(id, title, imageUri, address, latitude, longitude) {
    this.id = id
    this.title = title
    this.imageUri = imageUri
    this.address = address
    this.latitude = latitude
    this.longitude = longitude
  }
}
```

Store/actions/places.js

```
import * as FileSystem from "expo-file-system"
import API_KEYS from "../../env"
import { getAllPlaces, insertPlace } from "../../storage/db"

export const ADD_PLACE = "ADD_PLACE"
export const LOAD_PLACES = "LOAD_PLACES"

export const addPlace =
  (title, imageUri, selectedLocation) => async (dispatch) => {
  try {
    const { latitude, longitude } = selectedLocation
    const response = await fetch(
      `https://maps.googleapis.com/maps/api/geocode/json
      ?latlng=${latitude},${longitude}&key=${API_KEYS.GOOGLE_MAPS}`
    )

    if (!response.ok) throw new Error("Error while fetching map.")

    const data = await response.json()

    // no results means no places returned from the app
    if (!data.results)
      throw new Error("No results from Google Maps.")

    // our central marker (selected or default) is the target
    const address = data.results[0].formatted_address
```

```

// split path and grab the image file name with extension
const fileName = imageUri.split("/").pop()

// `FS.cacheDirectory`, `FS.bundleDirectory`, `FS.documentDirectory`
// > The last one being the permanent one for the app
const newPath = FileSystem.documentDirectory + fileName

// move the file from its cache to its permanent path in memory.
// > it can fail! (e.g. no space). Hence trycatch.
await FileSystem.moveAsync({ from: imageUri, to: newPath })

// storing data in DB can also fail
const res = await insertPlace(
  title, newPath, address, latitude, longitude
)

dispatch({
  type: ADD_PLACE,
  payload: {
    id: res.insertId, title, pathToImg: newPath,
    address, coords: selectedLocation
  }
})
} catch (err) { console.log(err); throw err }
}

export const loadPlaces = () => async (dispatch) => {
try {
  const res = await getAllPlaces()

  dispatch({ type: LOAD_PLACES, payload: { places: res.rows._array } })
} catch (err) { console.log(err); throw err }
}
}

```

Store/reducers/places.js

```

import Place from "../../models/place"
import { LOAD_PLACES, ADD_PLACE } from "../actions/places"

const initialState = { items: [] }

```

```

export default function placesReducer(state = initialState, action) {
  switch (action.type) {
    case LOAD_PLACES:
      return {
        items: action.payload.places.map(
          (p) => new Place(
            p.id.toString(), p.title, p.imageUri,
            p.address, p.latitude, p.longitude
          )
        )
      }
    case ADD_PLACE:
      return {
        items: state.items.concat( new Place(
          action.payload.id.toString(),
          action.payload.title,
          action.payload.pathToImg,
          action.payload.address,
          action.payload.coords.latitude,
          action.payload.coords.longitude
        )
      )
    }
    default: return state
  }
}

```

Components/MapPreview.jsx

```

...
export default function MapPreview({
  location, children, onPress, style
}) {
  const imgPreviewUrl = !location.latitude
    ? ""
    : `https://maps.googleapis.com/maps/api/staticmap?
      center=${location.latitude},${location.longitude}&zoom=13
      &size=400x200&maptype=roadmap
      &markers=color:red%7Clabel:A%7C
      ${location.latitude},${location.longitude}`
}

```

```

    &key=${API_KEYS.GOOGLE_MAPS}

    `

return (
<TouchableOpacity
  onPress={onPress} style={[_styles.container, style]}>
{imgPreviewUrl ? (
  <Image source={{ uri: imgPreviewUrl }} style={_styles.image} />
) : (
  children
)}
</TouchableOpacity>
)
}

const _styles = StyleSheet.create({
  container: { justifyContent: "center", alignItems: "center" },
  // network images (async) needs 100% width and height
  image: { width: "100%", height: "100%" }
})

```

Components/LocationPicker.jsx

```

import * as Location from "expo-location"
...

export default function LocationPicker({ navigation, onPickLocation })
{
  const [pickedLocation, setPickedLocation] = useState({
    latitude: 0, longitude: 0
  })
  const [isFetching, setIsFetching] = useState(false)

  // comes from 'Map' screen.
  // > defined when saving a marker, undefined when navigating back
  const selectedLocation = navigation.getParam("location")

  useEffect(() => {
    if (selectedLocation?.latitude) {
      setPickedLocation(selectedLocation)
      onPickLocation(selectedLocation)
    }
  }, [selectedLocation])
}

```

```

        }

    , [selectedLocation, onPickLocation])

const handleGetLocation = async () => {
    const hasPermission = await verifyPermission()
    if (!hasPermission) return
    try {
        setIsFetching(true)
        const res =
            await Location.getCurrentPositionAsync({ timeInterval: 5000 })
        const coordinates = {
            latitude: res.coords.latitude,
            longitude: res.coords.longitude
        }
        setPickedLocation(coordinates)
        onPickLocation(coordinates)
    } catch (err) {
        Alert.alert("Could not fetch location",
            "Try again or pick location from map", [{ text: "Ok" }])
        console.log(err)
        throw err
    }
    setIsFetching(false)
}

const handleAddMarker = () => navigation.navigate("Map")

return (
    <View style={_styles.container}>
        <MapPreview location={pickedLocation}
            onPress={handleAddMarker} style={_styles.preview}>
        >
            {isFetching ? (
                <ActivityIndicator size="large" color={colors.PRIMARY} />
            ) : (
                <Text>No location</Text>
            )}
        </MapPreview>
        <View style={_styles.buttonsContainer}>
            <Button title="Pick location" color={colors.PRIMARY}>

```

```

        onPress={handleAddMarker}
    />
    <Button title="Get location" color={colors.PRIMARY}
        onPress={handleGetLocation}
    />
</View>
</View>
)
}

const _styles = StyleSheet.create({
    container: { marginBottom: 15, alignItems: "flex-end" },
    preview: { marginBottom: 10, width: "100%", height: 150,
        borderColor: "#ccc", borderWidth: 1
    },
    buttonsContainer: { width: "100%", flexDirection: "row",
        justifyContent: "space-between"
    }
})

async function verifyPermission() {
    const result = await Location.requestForegroundPermissionsAsync()
    if (result.status !== "granted") {
        Alert.alert("Failed access", "Location permission is required", [
            { text: "OK" }
        ])
        return false
    }
    return true
}

```

Screens/NewPlace.jsx

```

...
import LocationPicker from "../components/LocationPicker"
...
export default function NewPlace(props) {
    const [title, setTitle] = useState("")
    const [selectedImage, setSelectedImage] = useState(null)
    const [selectedLocation, setSelectedLocation] = useState({
        latitude: null, longitude: null
    })
}
```

```

        })

const dispatch = useDispatch()

const handleImageTaken = (imageUri) => setSelectedImage(imageUri)

const handlePickLocation = useCallback((coordinates) => {
    setSelectedLocation(coordinates)
}, [])

const handleSave = () => {
    if (!selectedImage) return showAlert("Please select an image")
    if (!selectedLocation.latitude)
        return showAlert("Please pick a location")

    dispatch(placesActions.addPlace(
        title, selectedImage, selectedLocation))
    props.navigation.goBack()
}

return (
<ScrollView>
    <View style={_styles.container}>
        <Text style={_styles.label}>Title</Text>
        <TextInput value={title} onChangeText={setTitle}
            style={_styles.input}>
        />
        <ImagePicker onImageTaken={handleImageTaken} />
        <LocationPicker
            navigation={props.navigation}
            onPickLocation={handlePickLocation}>
        />
        <Button
            title="Save" color={colors.PRIMARY} onPress={handleSave} />
    </View>
</ScrollView>
)
}

NewPlace.navigationOptions = { headerTitle: "Add place" }

```

```

const _styles = StyleSheet.create({
  container: { margin: 30 },
  label: { fontSize: 18, marginBottom: 15 },
  input: { borderBottomColor: "#ccc", borderBottomWidth: 1,
            marginBottom: 15, paddingVertical: 5, paddingHorizontal: 2
  }
})

function showAlert(msg) {
  Alert.alert("Failed to save", `${msg}`, [{ text: "Ok" }])
}

```

Screens/Map.jsx

```

...
import MapView, { Marker } from "react-native-maps"
...
const region = {
  latitude: 37.78, // point
  longitude: -122.43,
  latitudeDelta: 0.0922, // radius
  longitudeDelta: 0.0421
}

export default function Map(props) {
  const [location, setLocation] = useState({
    ...region, latitude: null, longitude: null
  })

  // event has a bunch of props, including `nativeEvent` with
  // coordinates, target and position
  const handleSelectLocation = (e) => {
    setLocation((prevSt) => ({
      ...prevSt,
      latitude: e.nativeEvent.coordinate.latitude,
      longitude: e.nativeEvent.coordinate.longitude
    }))
  }

  const saveLocation = useCallback(() => {

```

```

    if (!location.latitude) {
      return Alert.alert(
        "Failed to save location",
        "Please select a location in map before attempting to save",
        [{ text: "Ok" }]
      )
    }
    props.navigation.navigate("NewPlace", {
      location: {
        latitude: location.latitude, longitude: location.longitude
      }
    })
  }, [location])

useEffect(() => {
  props.navigation.setParams({ saveLocation })
}, [saveLocation])

return (
  <MapView
    region={!location.latitude ? region : location}
    style={_styles.container}
    onPress={handleSelectLocation}
  >
  {location.latitude &&
    <Marker title="Target" coordinate={location} />}
  </MapView>
)
}

Map.navigationOptions = ({ navigation }) => {
  const saveLocation = navigation.getParam("saveLocation")

  return {
    headerRight: () => (
      <TouchableOpacity onPress={saveLocation}>
        style={_styles.headerRightContainer}
      >
        <Text style={_styles.headerRightText}>Save</Text>
      </TouchableOpacity>
    )
  }
}

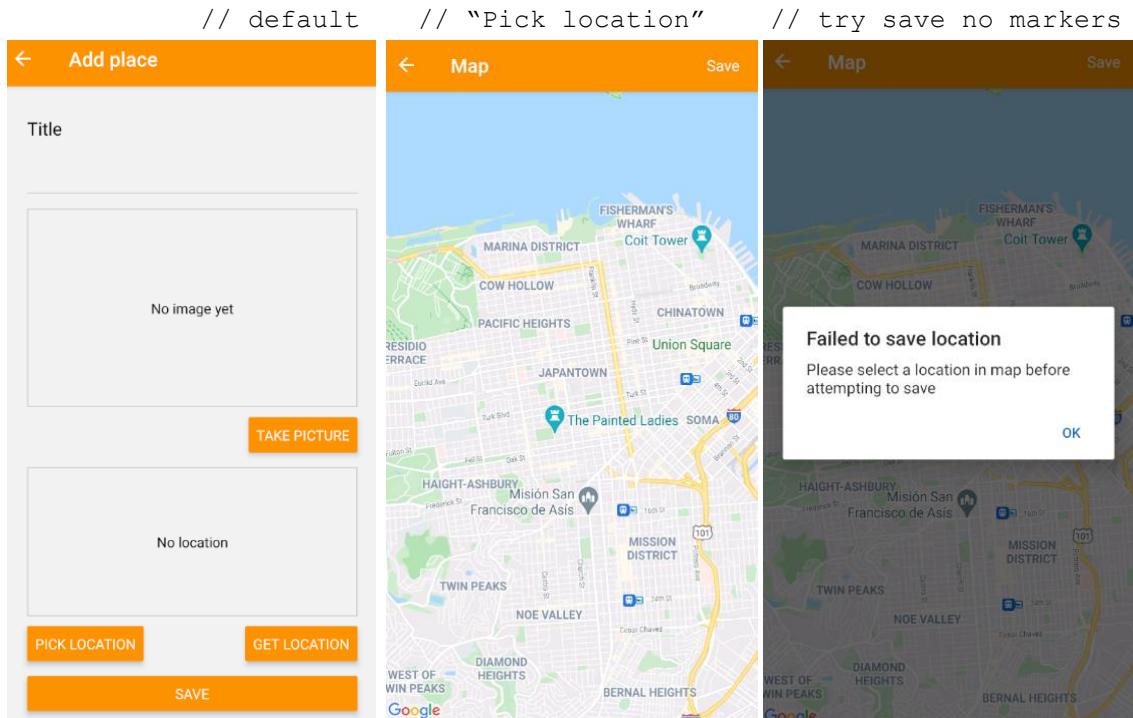
```

```

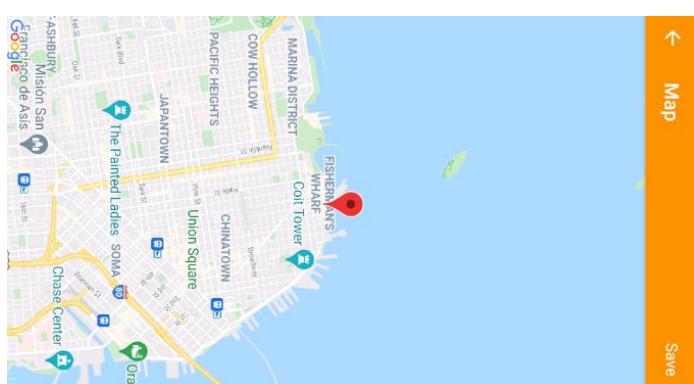
        )
    }
}

const _styles = StyleSheet.create({
  container: { flex: 1 }, // required to be displayed
  headerRightContainer: { marginHorizontal: 20 },
  headerRightText: {
    fontSize: 16,
    color: Platform.OS === "android" ? "white" : colors.PRIMARY
  }
})

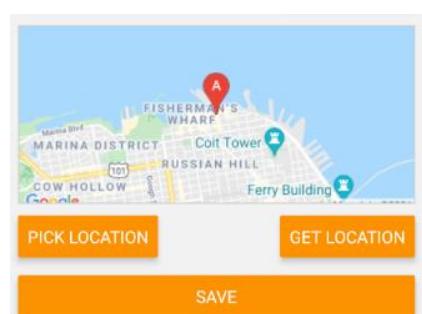
```



```
// on location select
```



```
// on Map screen "save"
```



Displaying "Details" screen

Screens/PlaceDetails.jsx

```
...
export default function PlaceDetails(props) {
  const id = props.navigation.getParam("id")
  const { imageUri, address, latitude, longitude } =
    useSelector((state) =>
      state.places.items.find((p) => p.id === id)
    )
  const location = { latitude, longitude }

  const showMap = () =>
    props.navigation.navigate("Map", {
      readonly: true, initialLocation: location
    })

  return (
    <ScrollView contentContainerStyle={{ alignItems: "center" }}>
      <Image source={{ uri: imageUri }} style={_styles.image} />
      <View style={_styles.locationContainer}>
        <View style={_styles.addressContainer}>
          <Text style={_styles.addressText}>{address}</Text>
        </View>
        <MapPreview location={location} style={_styles.mapPreview}>
          onPress={showMap}
        />
      </View>
    </ScrollView>
  )
}

PlaceDetails.navigationOptions = ({ navigation }) => ({
  headerTitle: navigation.getParam("title")
})

const _styles = StyleSheet.create({
  image: {
    height: "35%", minHeight: 300,
    width: "100%", backgroundColor: "#ccc" },
  ...
})
```

```

locationContainer: {
  marginVertical: 20, width: "90%", maxWidth: 350,
  justifyContent: "center", alignItems: "center",
  shadowColor: "black", shadowOpacity: 0.26,
  shadowOffset: { width: 0, height: 2 }, shadowRadius: 8,
  elevation: 5, backgroundColor: "white", borderRadius: 10
},
addressContainer: { padding: 20 },
addressText: { color: colors.PRIMARY, textAlign: "center" },
mapPreview: {
  width: "100%", maxWidth: 350, height: 300,
  borderBottomLeftRadius: 10, borderBottomRightRadius: 10
}
)

```

Screens/Map.jsx

```

...
import MapView, { Marker } from "react-native-maps"

const region = {
  latitude: 37.78, // point
  longitude: -122.43,
  latitudeDelta: 0.0922, // radius
  longitudeDelta: 0.0421
}

export default function Map({ navigation }) {
  // comes from 'PlaceDetails' screen
  const initialLocation = navigation.getParam("initialLocation")
  const readonly = navigation.getParam("readonly")

  const [location, setLocation] = useState({
    ...region,
    latitude: initialLocation?.latitude, // will be defined if coming
    longitude: initialLocation?.longitude // from 'PlaceDetails'
  })

  // event has a bunch of props, including `nativeEvent` with
  // coordinates target and position
}

```

```
const handleSelectLocation = (e) => {
  if (readonly) return // picking location is disabled on `readonly`

  setLocation((prevSt) => ({
    ...prevSt,
    latitude: e.nativeEvent.coordinate.latitude,
    longitude: e.nativeEvent.coordinate.longitude
  }))
}

const saveLocation = useCallback(() => {
  if (!location.latitude) {
    return Alert.alert(
      "Failed to save location",
      "Please select a location in map before attempting to save",
      [{ text: "Ok" }]
    )
  }
  navigation.navigate("NewPlace", {
    location: {
      latitude: location.latitude, longitude: location.longitude
    }
  }, [location])
}

useEffect(() => {
  navigation.setParams({ saveLocation })
}, [saveLocation])

return (
  <MapView
    region={!location.latitude ? region : location}
    style={_styles.container} onPress={handleSelectLocation}
  >
  {location.latitude &&
    <Marker title="Target" coordinate={location} />}
  </MapView>
)
}
```

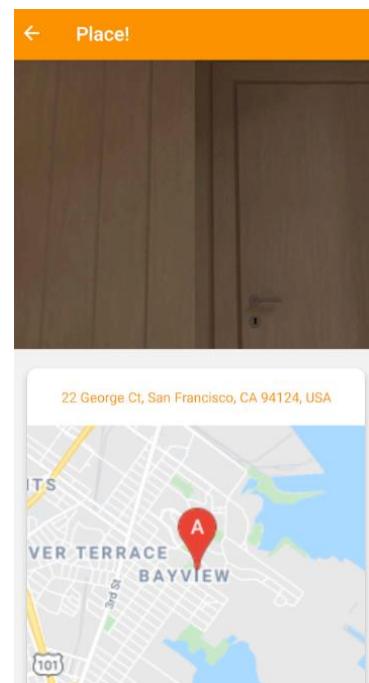
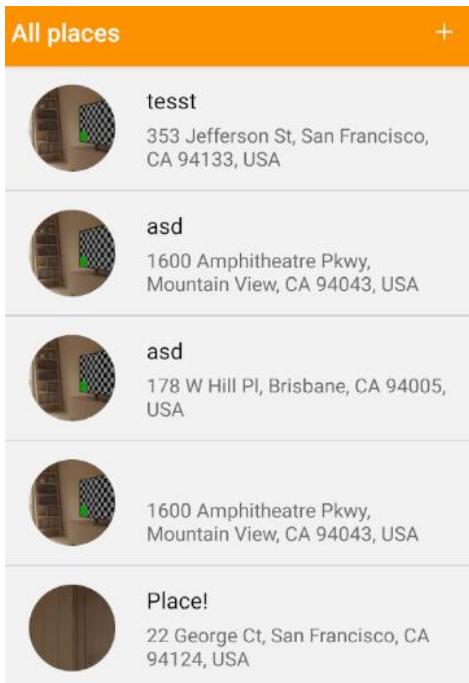
```

Map.navigationOptions = ({ navigation }) => {
  const saveLocation = navigation.getParam("saveLocation")
  const readonly = navigation.getParam("readonly")

  // if coming from 'PlaceDetails' screen, no 'Save' option
  return readonly
  ? {}
  : {
    headerRight: () => (
      <TouchableOpacity onPress={saveLocation}
        style={_styles.headerRightContainer}
      >
        <Text style={_styles.headerRightText}>Save</Text>
      </TouchableOpacity>
    )
  }
}

const _styles = StyleSheet.create({
  container: { flex: 1 }, // required to be displayed
  headerRightContainer: { marginHorizontal: 20 },
  headerRightText: { fontSize: 16,
    color: Platform.OS === "android" ? "white" : colors.PRIMARY
  }
}) // "Places" screen // "Place Details" screen

```



Note on iPhone permissions

While android gets away with only asking for camera permissions explicitly (or asking permissions automatically on last OS versions), iOS still needs both `CAMERA` and `MEDIA GALLERY` permissions to work.

So, do not forget to do so. In our app, they are asked on `ImagePicker` component:

Components/ImagePicker.jsx

```
...
import * as ImagePicker from "expo-image-picker"
...
export default function ImgPicker({ onImageTaken }) {
  const [PickedImageUri, setPickedImageUri] = useState(null)

  const handleTakeImage = async () => {
    const hasPermission = await verifyPermission()
    if (!hasPermission) return
    const image = await ImagePicker.launchCameraAsync({
      allowsEditing: true, aspect: [16, 9], quality: 0.5
    })
    setPickedImageUri(image.uri)
    onImageTaken?.(image.uri)
  }

  return (
    <View style={_styles.container}>
      <View style={_styles.preview}>
        {PickedImageUri ? (
          <Image
            source={{ uri: PickedImageUri }} style={_styles.image} />
        ) : (
          <Text>No image yet</Text>
        )}
      </View>
      <Button title="Take picture" color={colors.PRIMARY}
        onPress={handleTakeImage} />
    </View>
  )
}
```

```

const _styles = StyleSheet.create({...})

// Note: permissions are asked only once, even if the function is
// re-triggered. You need to re-launch expo.

async function verifyPermission() {
  // const result = await Permissions.askAsync(
  //   Permissions.CAMERA,
  //   Permissions.MEDIA_LIBRARY
  // )
  const cameraPermissions =
    await ImagePicker.requestCameraPermissionsAsync()
  const mediaPermissions =
    await ImagePicker.requestMediaLibraryPermissionsAsync()

  if (
    cameraPermissions.status !== "granted" ||
    mediaPermissions.status !== "granted"
  ) {
    Alert.alert(
      "Failed access",
      "Camera and media library permissions are required",
      [{ text: "OK" }]
    )
    return false
  }
  return true
}

```

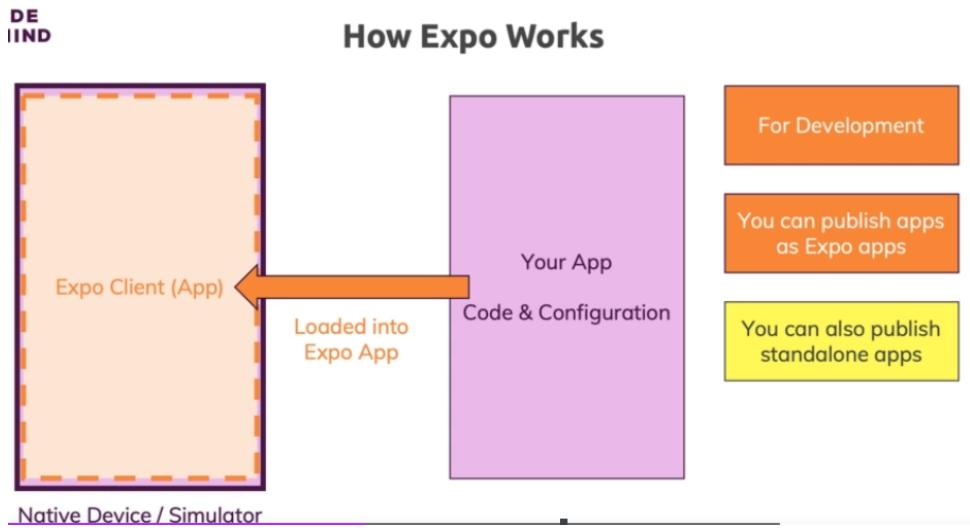
Building apps without expo

Alternatives to expo

Even though Expo is a library that offers you an incredible amount of flexibility and speed to your native development, there are reasons for which you might opt not to use it as your default.

One is the bundle size, as when you ship your app, expo wrapper is still there (the code is not tree-shaked). Thus, many expo functionalities you might not need are still bundled.

Another reason is that you might need a native functionality not provided by Expo, and only available on native Java/Swift/Objective C code. Thus, such code must be written in that language and added to the compiled package, thing we cannot do with Expo as native code is hidden away.



So, to circumvent these issues, you can choose to build an **Expo app with a bare workflow** (that is, not managed -the same as ejecting from managed-), or directly using the standard **React Native CLI**.

Alternatives		
Expo – Managed Workflow	Expo – Bare Workflow Can be created from scratch or by ejecting	React Native CLI
Easy to use, "zero setup"	Non-Expo app	Non-Expo app
Lots of native modules built-in	Still uses imported Expo packages	Can use Expo packages, manual setup required
Controlled via Expo CLI	Relatively easy to configure & manage	Manage on your own
Standalone app can be deployed (incl. Expo wrapper)	Any native module can be used (incl. non-Expo)	Any native module can be used (incl. non-Expo)
Android Studio & XCode technically not required	Build & Deploy with Android Studio / XCode	Build & Deploy with Android Studio / XCode

Keep in mind that everything written in React Native language is NOT lost. That includes components, screens, utilities and such.

What changes is the way you import and use packages, specially the ones from expo.

That includes lines like: `import { SQLite } from 'expo-sqlite'`

And of course, there will be some packages you cannot use at all.

React Native CLI

React Native official docs are the way to go. In there you find the detailed installation instructions, for all development OS and mobile target OS.

Expo cli managed the availability to emulate the app on a real device and in the web, however, that option is not available in React Native CLI. Thus, we must install XCode and/or Android Studio.

There are some steps you need to follow to the step, like setting up environment variables for AndroidSDK, and updating environment Path variable to AndroidSDK project-tools. Check the docs.

And obviously, you need React Native CLI to create a project, which can be accessed on its latest stable version with npx. So, start a project:

```
$ npx react-native init RNWithoutExpo
```

Access the folder and launch the app.

```
$ npx react-native start
```

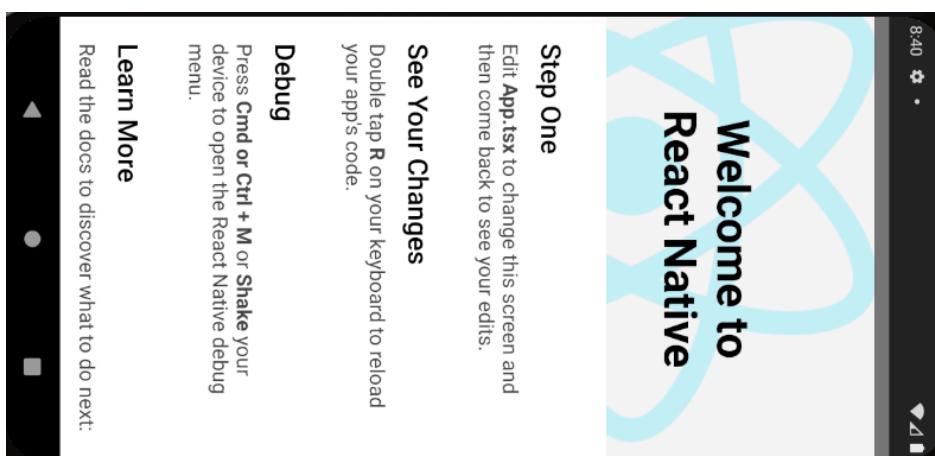
A terminal opens. That's the server running. Do NOT close it.

Now, launch the app on target simulator:

```
$ npx react-native run-android | run ios
```

The simulator should automatically open and the bundle, built.

The startup screen looks as follows (I changed "App.js" text for "App.tsx" to test if hot reloading worked).



Take a look at the project's structure:

It looks like the one managed by Expo,
But has quite a bit of differences.

For starters, we have one folder for
Android and another one for ios, which
Holds all files relative to each
device's OS files to be bundled.

App.js is the starter file, as we had
In Expo. There, you have some imports
You are already familiar with, that
Combined together create the startup
Screen shown before.

React-Native is React-Native.
Everything we learned before with Expo
Applies here. Expo is just a thin
Wrapper that adds loads of
Customization to facilitate and speed
The creation process up.

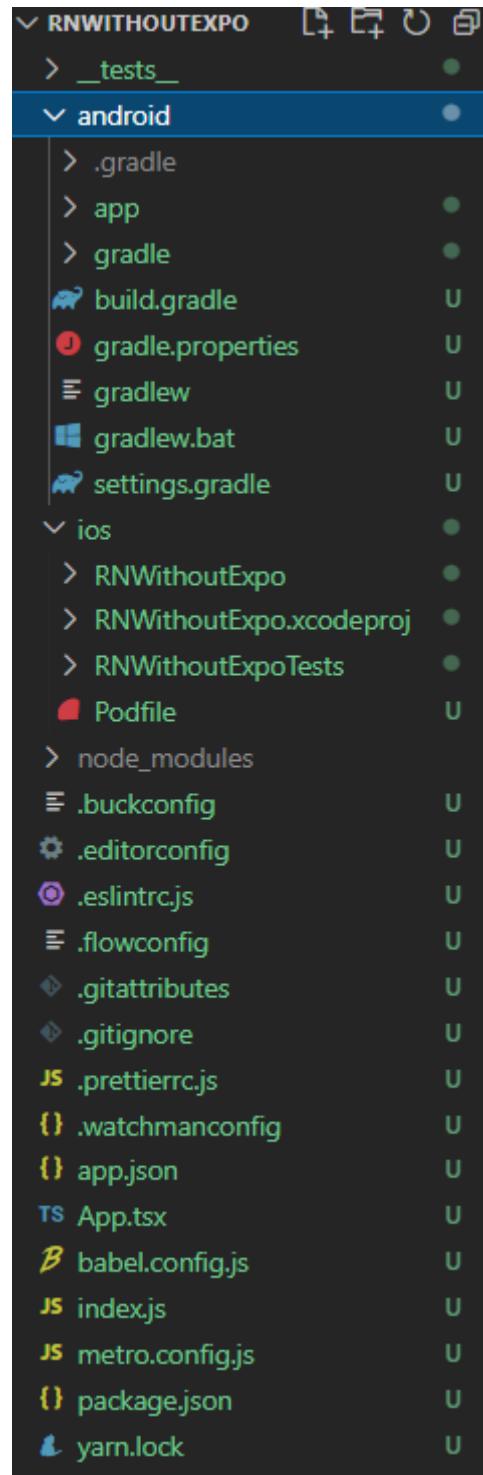
Index.js file is revealed here, with
Some weird syntax we did not see before
Due to it being hidden and handled by
Expo. Nothing to fear, this file is
Used to register components to be
Reckoned by React-Native compiler.

Check, for example, how App.js
Component is mapped to 'appName' string:

Index.js

```
/**  
 * @format  
 */  
import {AppRegistry} from 'react-native';  
import App from './App';  
import {name as appName} from './app.json';
```

```
AppRegistry.registerComponent(appName, () => App);
```



Note on live-reloading:

If you are using an older version of react-native-cli, the app wouldn't reload when changing code.

Such case, and unlike in Expo apps, "live reload" needs to be enabled.

To do this, open the developer menu on the device.

- For Android Emulators: CTRL + M (or CMD + M on a Mac)
- For iOS Simulators: CMD + D
- For physical devices: Shake the device

Then "Enable Live Reload"

Installing packages

Previously, with expo, we were limited to Javascript packages. Now we can bring any package we want, no matter if it taps inside native functionalities or not.

Let's use ImagePicker package (react-native one, not expo like we did before).

```
$ npm install --save react-native-image-picker
```

And the package must be linked to the project (expo did it by default).

```
$ npx react-native link react-native-image-picker
```

Some packages does not support link command. Always check the docs for what you want to install to know how to manage it.

Open *android/src/settings.gradle* and you will notice the package was added along its instructions to compile it:

android/src/settings.gradle

```
rootProject.name = 'RNWithoutExpo'  
include ':react-native-image-picker'  
project(':react-native-image-picker').projectDir = new  
File(rootProject.projectDir, '../node_modules/react-native-  
image-picker/android')  
apply from: file("../node_modules/@react-native-community/cli-  
platform-android/native_modules.gradle");  
applyNativeModulesSettingsGradle(settings)  
include ':app'
```

However, another thing handled by Expo were the package permissions (not user permissions, but packages to the app itself).

Those ones are to be configured by us too.

Check the docs in the target package to be installed. Everything installation-related should be there (what proceeds now is subtracted from there).

android/app/src/main/AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.rnwithoutexpo">

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

    <application
        android:name=".MainApplication"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:allowBackup="false"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:configChanges=
                "keyboard|keyboardHidden|orientation|screenSize|uiMode"
            android:launchMode="singleTask"
            android:windowSoftInputMode="adjustResize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Ios/RNWithoutExpo/Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
```

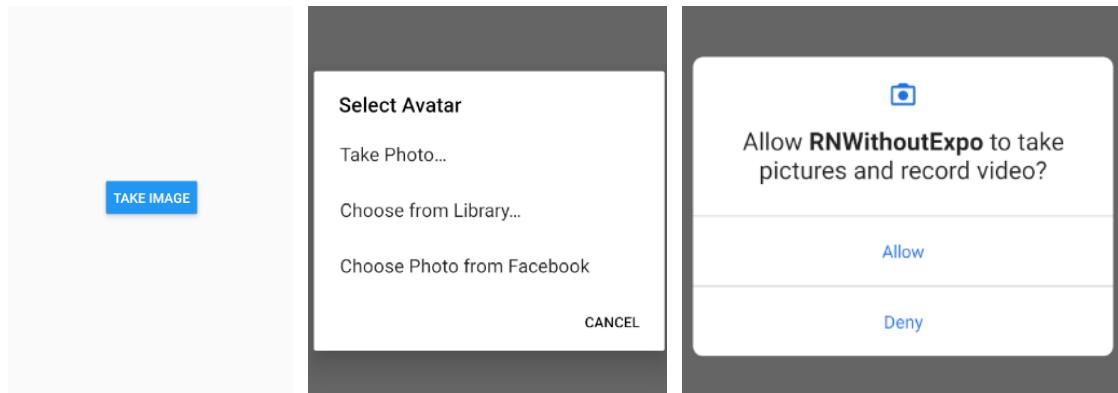
```
<key>CFBundleDevelopmentRegion</key>
<string>en</string>
<key>CFBundleDisplayName</key>
<string>RNWithoutExpo</string>
<key>CFBundleExecutable</key>
<string>$(EXECUTABLE_NAME)</string>
<key>CFBundleIdentifier</key>
<string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
<key>CFBundleInfoDictionaryVersion</key>
<string>6.0</string>
<key>CFBundleName</key>
<string>$(PRODUCT_NAME)</string>
<key>CFBundlePackageType</key>
<string>APPL</string>
<key>CFBundleShortVersionString</key>
<string>1.0</string>
<key>CFBundleSignature</key>
<string>????</string>
<key>CFBundleVersion</key>
<string>1</string>
<key>LSRequiresiPhoneOS</key>
<true/>
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSExceptionDomains</key>
    <dict>
        <key>localhost</key>
        <dict>
            <key>NSExceptionAllowsInsecureHTTPLoads</key>
            <true/>
        </dict>
    </dict>
</dict>
<key>NSPhotoLibraryUsageDescription</key>
<string>${PRODUCT_NAME} would like access to your photo gallery</string>
<key>NSCameraUsageDescription</key>
<string>${PRODUCT_NAME} would like access to use your camera</string>
<key>NSPhotoLibraryAddUsageDescription</key>
<string>
    ${PRODUCT_NAME} would like save photos to your photo gallery
</string>
<key>NSMicrophoneUsageDescription</key>
<string>
```

```

${PRODUCT_NAME} would like access to your microphone (for videos)
</string>

<key>NSLocationWhenInUseUsageDescription</key>
<string></string>
<key>UILaunchStoryboardName</key>
<string>LaunchScreen</string>
<key>UIRequiredDeviceCapabilities</key>
<array>
    <string>armv7</string>
</array>
<key>UISupportedInterfaceOrientations</key>
<array>
    <string>UIInterfaceOrientationPortrait</string>
    <string>UIInterfaceOrientationLandscapeLeft</string>
    <string>UIInterfaceOrientationLandscapeRight</string>
</array>
<key>UIViewControllerBasedStatusBarAppearance</key>
<false/>
</dict>
</plist>

```



Understanding Expo's bare workflow

Expo bare workflow is an in-between in Expo managed workflow and React Native CLI.

It includes React Native app as the created with the CLI, but comes with many pre-configured Expo packages. The idea comes from when you need to configure android and ios manually in development, but also when wanting to add native functionality in an easier way (not so many configurations for those).

Of course, the best ally is always the docs. Read them.

That being said, let's create an example.

First, you need the setup described in the docs for both React Native CLI and expo CLI. Not to create a project, but to get the PC ready to do so. This includes dependencies, Android Studio, XCode, Expo, and some configurations.

```
$ expo init <project-name> --template bare-minimum
```

Or simply:

```
$ expo init <project-name>
```

And choose the bare-minimum workflow template as the option.

The magic happens with the help of **react-native-unimodules** package, which adds supported apis (like Accelerometer, Permissions, Asset, AV, and many more) capable of linking with a React Native bare app.

This means, Expo bare workflow is just a React Native CLI app with react-native-unimodules as a middleware capable of adding expo functionalities to the project. So, many expo-related packages we have been using up to now and plenty more can be added with the help of external libraries belonging to expo too.

To include them, go to each library and read the docs, but it normally consists of one generic npm command, and maybe some other ones for iOS and Adroid. Like:

```
$ npm install expo-location
$ pod install // ios
// No extra configs for android
```

The github page also offers instructions to add them if you started from React Native CLI.

You run the app as always:

```
$ react-native run-android | run-ios
```

Ejecting from expo managed workflow

The magic of the bare workflow is that you can eject it into a react-native CLI anytime you want.

Warning! There is NO going back to a bare workflow after this.

This is good because you can always work in development mode with it, and then when you are ready for production, you can eject to perform tree-shaking.

Also, at any time you need a native addition (bare code or library) impossible to handle with Javascript, you also get the possibility to eject.

Let's do it on the great-places app we created earlier.

```
$ expo eject
```

Since great-places app is a project created with a managed flow, it will ask you for the target eject build: Bare React Native or Expo-Kit.

Expo-kit has been deprecated, so choose Bare, which gives you a React-Native CLI app with each single package we used across the whole project already pre-configured.

After ejecting, you will be presented with a list that tells you which previously included packages need manual setup before they can get going again.

In our case:

```
Warning: your app includes 2 packages that require additional setup. See the following URLs for instructions.  
Your app may not build/run until the additional setup for these packages has been completed  
*  
- expo-image-picker: https://github.com/expo/expo/tree/master/packages/expo-image-picker  
- react-native-maps: https://github.com/react-native-community/react-native-maps
```

Which, in these cases, it just includes running `pod install` in ios folder, and some **activity tags** in `AndroidManifest.xml`.

Note: You should remove all permissions your app does NOT require. They are in `AndroidManifest.xml`

Android/app/src/main/AndroidManifest.xml

```
...  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>  
<uses-permission android:name="android.permission.CAMERA"/>  
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>  
<uses-permission android:name="android.permission.INTERNET"/>  
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>  
<uses-permission android:name="android.permission.RECORD_AUDIO"/>  
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>  
<uses-permission android:name="android.permission.VIBRATE"/>  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>  
...
```

Additionally, maps also needs some meta-data in that same file, for the device to access google api key:

Android/app/src/main/AndroidManifest.xml

```
...
<meta-data
    Android:name='com.google.android.geo.API_KEY'
    Android:value:'<api-key-here>' />
...
...
```

Note that since we are using Google maps, we need to enable the google cloud Android API from there too.

Cloud platform for your project > Menu > APIs and Services > Library

Enable Maps SDK for Android

The screenshot shows the Google Cloud Platform API Library interface. At the top, there's a blue header bar with the text "Google Cloud Platform" and "great-places". Below the header, a left sidebar shows a navigation tree with "API Library" selected. The main area has a background image of a map with colored dots. In the center, it says "Welcome to the API Library" and "The API Library has documentation, links, and a smart search experience.". A search bar at the bottom has the placeholder "Search for APIs & Services". On the left, a sidebar titled "Filter by" has sections for "VISIBILITY" (Public 349, Private 3) and "CATEGORY" (Advertising 14, Analytics 5, Big data 17, Blog & CMS 1, Compute 7, CRM 1, Databases 6). The main content area is titled "Maps" and shows four API cards: "Maps SDK for Android" (Google), "Maps SDK for iOS" (Google), "Maps JavaScript API" (Google), and "Places API" (Google Enterprise API). The "Places API" card includes a note: "Get detailed information about 100 million places".

Filter by

VISIBILITY

Public (349)
Private (3)

CATEGORY

Advertising (14)
Analytics (5)
Big data (17)
Blog & CMS (1)
Compute (7)
CRM (1)
Databases (6)

Maps

Maps SDK for Android
Google
Maps for your native Android app.

Maps SDK for iOS
Google
Maps for your native iOS app.

Maps JavaScript API
Google
Maps for your website

Places API
Google Enterprise API ?
Get detailed information about 100 million places

Machine learning

Mind you must follow similar steps for iOS.

After all that, initialize the app.

```
$ react-native run-android | run-ios
```

This will build the app using XCode/Android Studio, and run it in an emulator there. Now without Expo as a bridge, which means, the functional build is fully native (like done in React Native CLI).

When to use which?

CODE
MIND

Expo vs “Non-Expo”

Expo	Non-Expo
Easy to use + develop	More manual setup required
Easy to deploy (incl. iOS on Windows/ Linux!)	Manual deployment, no iOS apps on Windows/ Linux
Rich suite of native modules (which are easy to use)	Any third-party library can be used (but needs to be configured manually)
Wrapper around your app => Size + performance “impact”	No wrapper
Restricted to built-in native modules	No limitations but also “out of the box” functionality

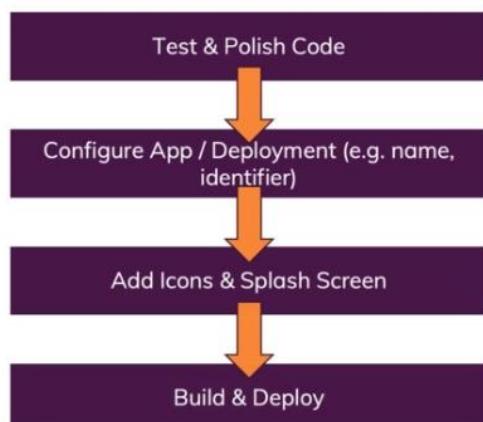
In short, work with expo for most apps, you can always eject later.

Starting with a bare workflow or a React Native CLI app is a good idea only when you know your bundle size matters and/or you know beforehand you will use lots of native modules.

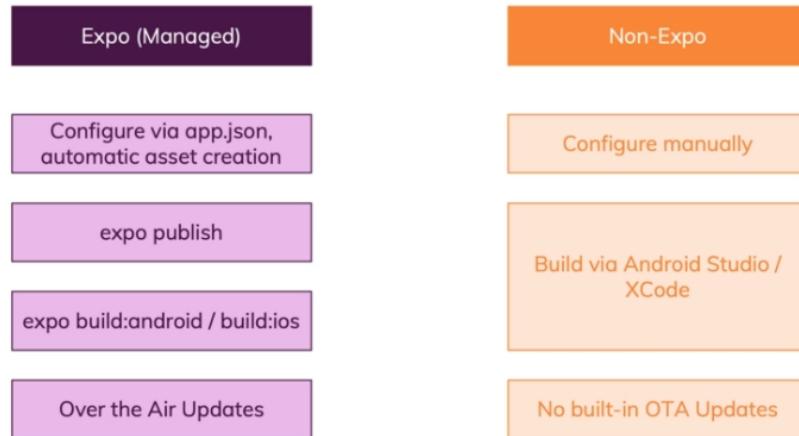
Publishing React Native apps

Steps

Deployment Steps



Expo Apps vs “Non-Expo” Apps



Over the air updates means you can push code updates of the app online, and users do not need to install new versions of the app to have them reflected on their devices. The app will update itself given the proper permissions.

Configuring the app for publishing

App.json in the root folder is the file you want to target to configure the project for deployment.

Side note: Always refer to the official docs for instructions.

Docs for this section: "Configuration with app.json"

sdkVersion: Defining a version will always update the app using that Android SDK version and its instructions.

Undefined means the latest stable version available is used each time the app is updated, along its instructions.

Platforms: 'ios', 'web', 'android'. The target platforms the app can be released to.

Version: The platform version. Consists of 3 numbers separated by a ‘.’ each (like 1.2.0).

- > The last number is reserved for tiny changes, like bug fixes.
- > The middle number is used to introduce new features or major Changes.
- > The first number is changed when a version with breaking-changes is introduced.

Orientation: "default", "portrait" or "landscape". Locks the device to that orientation.

Icon: the app's icon when not opened (displayed on the device")

Splash: The splash screen configuration object. Will explain later.

updates: configures "on the fly" updates. Will explain later.

assetBundlePatterns: configures patterns to target extra assets when the app builds up. Will explain later.

"ios" / "android" / "web": extra configuration objects that targets those platforms.

App.json

```
{  
  "expo": {  
    "name": "Great places",  
    "slug": "great-places",  
    "description": "Take pictures and locate them in map.",  
    "privacy": "public",  
    "sdkVersion": "34.0.0",  
    "platforms": ["ios", "android", "web"],  
    "version": "1.0.0",  
    "orientation": "portrait",  
    "icon": "./assets/icon.png",  
    "splash": {  
      "image": "./assets/splash.png",  
      "resizeMode": "contain",  
      "backgroundColor": "#ffffff"  
    },  
    "updates": {  
      "fallbackToCacheTimeout": 0  
    },  
    "assetBundlePatterns": [  
      "**/*"  
    ],  
    "ios": {  
      "supportsTablet": true  
    },  
    "android": {  
    }  
  }  
}
```

```

    "adaptiveIcon": {
      "foregroundImage": "./assets/adaptive-icon.png",
      "backgroundColor": "#FFFFFF"
    }
  },
  "web": {
    "favicon": "./assets/favicon.png"
  }
}

```

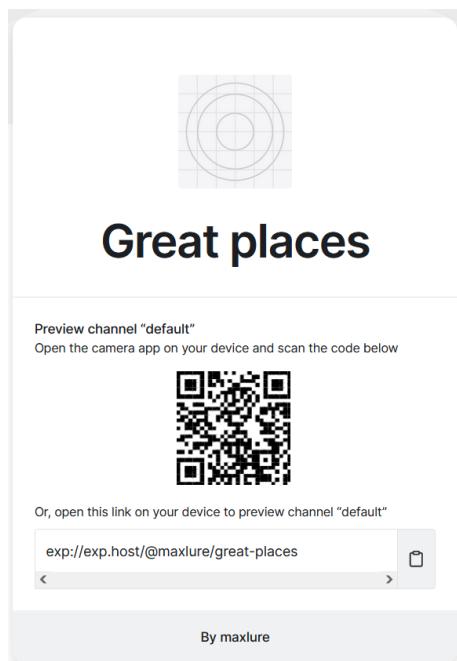
To publish the app, create an expo account, go to your project folder in a terminal and run:

\$ expo publish

It will not immediately publish it to app stores, no worries. It just bundles everything both for android, ios and/or web (according to what you specify)

A deployment link was created, which links to your app being now hosted in expo.

Scan the code to open the published App in Expo. This is no different From scanning the QR code in expo Client on development, but it is Now in production mode, and you Can share it with whoever has Expo installed on their devices.



Icon and Splash screen

Check the docs. Instructions on how to prepare icons and splash screens are there.

Expo provides a couple of placeholders in assets/ folder, but we will use some of our own to give the app some personality.

Icons are best suited for devices when they are 124x124 resolution.

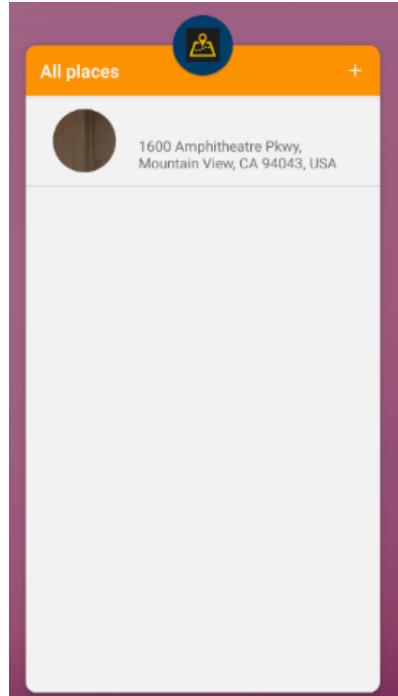
"resizeMode" property inside "**Splash**" in **app.json** has two values:

- > **"cover"**: stretches the icon to cover the screen.
- > **"contain"**: uses the provided icon size as is, centers it on Screen and paints the background with the provided color.

As you can see in the image, the icon does Not look so good on android, and this is Because we are missing some extra configs To make it adaptive (that is, to use an Transparent icon with a designed background Color.

Note: changing the icon will not take effect On emulators, but will be replaced on the When app builds (**\$ expo publish**).

You can also apply them in iOS if you so Desire, but adaptive icons are not supported There. Thus, some workarounds on the icon Image and background should be added to a Regular PNG file to look and feel like Android.



Offline asset bundles

Your app might also use other assets which need to be bundled in the final app (like images, audios, fonts, and such).

This is not the case for this one we created, but for further reference, Expo compiles and uploads them to their servers. They are downloaded when anyone launches the app.

This is good for your bundle size, but keep in mind that if somebody runs the app with no internet connection, your assets will not be available to them.

Therefore, you might want to add assets to the final bundle to download, and that is controlled by "**assetBundlePatterns**" key in **app.json**. Whichever path you specify on its value array will be excluded by expo's instant server upload on publish process.

Default value is "**assetBundlePatterns": ["**/*"]**", ****/*** being a wildcard stating "all files in all paths of the project" are NOT to be uploaded to expo servers.

If you stated "**assetBundlePatterns": ["assets/**/*"]**", then only folders and/or files inside assets/ will be included in the bundle. The rest is uploaded to expo servers to decrease the size of the final bundle. But remember: no internet connection, no files.

Over the air updates

As long as you built your app using Expo, you get Over the Air updates (OTAU). This means that once you change the code and re-publish it (no matter if it is a line or a whole new and large module), all users will get the update the next time they open the app.

OTAU configs can be found in app.json, like all other app's configs, under "**updates**" key.

It comes preconfigured with `fallbackToCacheTimeout` key set to 0. This tells the app to instantly download new updates as soon as it is opened (no timeout on splash screen). Changing it to 5000 will leave the splash on hold for five seconds before starting the download.

A business app which updates constantly and where UX is not that important can take higher values, but a widespread app open to the public needs to have lower values (preferably 0): a faster loading time leaving downloaded updates to take effect on the next startup.

So, let's show how OTAU work

First, publish it again and launch it by scanning the QR. This allows us to start fresh.

\$ expo publish

Now, change a small feature.

Screens/NewPlace.jsx

```
// NewPlace.navigationOptions = { headerTitle: "Add place" }
NewPlace.navigationOptions = { headerTitle: "Add New place" }
```

Republish the app, and with the app open on your device, close it and open it again.

The title will NOT have changed, and that makes sense. The app did not wait for the new changes to be downloaded and installed due to `fallbackToCacheTimeout` set to 0, which means (UX comes first, open the app, download in the background and apply changes on the next launch).

So, restart it one more time, and title should have changed by then.

Building for Android and iOS deployment

Up to now, everything built up was to publish the app in Expo, but we need to configure and build the app to launch it to stores.

Permissions

Our app uses plenty of native modules, and those require permissions which have been handled across the code with Permissions API.

When publishing for stores, we cannot use Expo as the app is standalone. So, we have to tell Android and iOS files which permissions are to be handled. This is also done in `app.json`, under `permissions` property, on “`android`” and “`ios`” keys.

Read the docs! *Managed Workflow > Configuration with App.json*

It is REALLY important you explicitly and specifically ask which permissions your apps uses, as if omitted, all of them will be requested, which is horrible UX.

Keys

Keys are to be manifested explicitly both for Android and iOS. In our case, we have Google Maps one. This is also added in `app.json`, on “`googleMaps`” key, inside “`apiKey`” Property, inside generic “`config`”.

Make sure Map SDK for Android is activated in Google Cloud Store,

like specified on React Native CLI section before.



Package / Bundle Identifier

A unique reverse URL used by android to identify your project. It must not be used by anyone else and MUST be present.

We will use `com.rnmc.great_places`, in “`android`” key.

For iOS, key is “`bundleIdentifier`”. The reverse URL is the same.

Version code / Build Number

For iOS, we have to provide a “`buildNumber`” with format similar to “`version`” (1.0.0). It is normally the same as version if you keep up with updates.

For Android we use “`versionCode`” and it is NOT a version number, but just a number (integer) which increases by one on each release (no matter if the change is tiny or if we are dealing with a major update).

That's the basics, but READ THE DOCS! With these changes, we are ready to publish again (managed by expo). Publish sets the default bundle up for Expo, which serves as a “pre-build” process for Android and iOS.

\$ expo publish

Now, once there are no errors after publishing, then we can leverage on cloud servers to create the bundles for both platforms as a standalone native app for further upload to both stores.

Side note: App bundles that are released will still use Expo as mediator, especially for OTAUs.

To build, check the docs for standalone apps. **Android** process is:

```
$ expo build:android -t app-bundle
```

App-bundle is a flag google recommends for efficiency and **TO REDUCE THE BUNDLE SIZE**. It's in the docs. USE IT!

The command pushes the app to expo cloud servers that links to Android ones to generate the final bundle.

Apps need to be signed with a key that identifies you as the creator. The build process instructs you on this.

Generate one and DO NOT LOSE IT, as only the owner of this key can publish updates, further publishes will require you to provide this key.

If you know what you are doing, create one with the keystore. Otherwise, simply let Expo handle the process.

```
Building optimized bundles and generating sourcemaps...
Starting Metro Bundler
Finished building JavaScript bundle in 9901ms.

Bundle           Size
├ index.ios.js   1.84 MB
├ index.android.js 1.85 MB
└ index.ios.js.map 5.19 MB
└ index.android.js.map 5.21 MB

💡 JavaScript bundle sizes affect startup time. Learn more: https://expo.fyi/javascript-bundle-sizes

Analyzing assets
Saving assets
No assets changed, skipped.

Processing asset bundle patterns:
- D:\Udemy\React Native\React Native - The Practical Guide\great-places-app\*\*

Uploading JavaScript bundles
Publish complete

📝 Manifest: https://exp.host/@maxlure/great-places/index.expo?sdkVersion=42.0.0 Learn more: https://expo.fyi/manifest-url
⚙️ Project page: https://expo.io/@maxlure/great-places Learn more: https://expo.fyi/project-page

Checking if this build already exists...

Build started, it may take a few minutes to complete.
You can check the queue length at https://expo.io/turtle-status

You can monitor the build at

https://expo.io/accounts/maxlure/projects/great-places/builds/27c77007-8c6f-45f8-bd57-b7039cd36cc

Waiting for build to complete.
You can press Ctrl+C to exit. It won't cancel the build, you'll be able to monitor it at the printed URL.
```

The final step is queued until it completes. This is such due to the first build process for standalone apps naturally taking longer to finish.

Once it does, you will get the link to download the .aab file corresponding to the app you have just created and bundled.

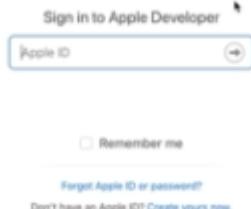
<https://expo.io/artifacts/523551fa-f108-45e6-9523-9aed93646da4>

Afterwards, you MUST run:

```
$ expo fetch:android:keystore
```

Which will **return the key you MUST SAVE AND NOT LOSE**, required to publish updates. It comes in the shape of a file.

Building for iOS is similar, but you will need a paid developer account there. Google “**apple developer account**” and follow the steps to set one up.



Afterwards, just `$ expo build:ios`

Credentials asked there will not be stored in cloud servers, but just be used to start the build process up.

Let expo handle the rest.

Once you have the packages built up, you can proceed to upload them to stores.

```
$ expo upload:android | upload:ios
```

Follow the additional steps stated in the docs, under "[Uploading Apps to the Apple App Store and Google Play](#)" section.

App.json

```
"privacy": "public",
"version": "1.0.0",
"orientation": "portrait",
"icon": "./assets/places.png",
"splash": {
  "image": "./assets/splash_icon.png",
  "resizeMode": "contain",
  "backgroundColor": "#171717"
},
"updates": { "fallbackToCacheTimeout": 0 },
"assetBundlePatterns": ["**/*"],
"ios": {
  "bundleIdentifier": "com.rnmc.great-places",
  "buildNumber": "1.0.0",
  "supportsTablet": true
},
"android": {
  "package": "com.rnmc.great_places",
  "versionCode": 1,
  "adaptiveIcon": {
    "foregroundImage": "./assets/places-adaptive.png",
    "backgroundColor": "#171717"
  },
  "config": {
    "googleMaps": {
      "apiKey": "AIzaSyD92UVwYMhcVfBvSiPu_yxXP-3oPbQAAFY"
    }
  },
  "permissions": [
    "ACCESS_FINE_LOCATION",
    "CAMERA",
    "WRITE_EXTERNAL_STORAGE"
  ]
},
"web": {
  "favicon": "./assets/favicon.png"
}
}
```

Publishing iOS apps without Expo

Docs here are "**Running on device**" - "**Running your app on iOS devices**"

Create an Apple developer account (right now not a paid one, that's for development).

Open the project in **XCode**, and navigate to **ios/XCWorkspace** file, choose to open project's folder.

Right there you will find a bunch of options to configure, which correlate to the ones we setup in **app.json** before (project name, bundle identifier, version, increment number for the version, and other configs).

Icons configuration cannot be missed there, you need to provide them on each size they are asked there. Expo handled this for us before, now we are on our own.

Icons also need to be configured for long screen. Expand the folder there and you will find a **xib file**, which you can edit.

Test the generated app on simulators and build it. You have the configuration there.

Once the app is built, it is automatically launched for you to check it works.

Next step, as stated in docs, is **Building your app for production**.

Two important things to do:

First, you need to locate **info.plist** file, and in there, the "App Transport Security Key" which controls to which web pages you can communicate with. There are default domains like localhost.

Production does NOT need localhost as a production server (that's development), so, remove it.

Second, go to **Product -> Scheme -> Add scheme**. Toggle "debug" to "release".

Afterwards, **Product -> build**, which in term will re-build the app, this time for release (optimized for production).

The app is ready, **you can go to App Store to upload it**. Of course, you need configs there too:

Go to "Certificates, Identifiers & Profiles" tab, select "Identifiers" and "Add a new App id". **The Bundle ID of your project in XCode needs to be set up there**.

You can set up a description on the fly there too.

Finally there, register your app.

Next, go to **App Store Connect** webpage. Select “My Apps” tab, and add a new one. Choose the name, language, and the bundle id you just set up as certificate.

Your app is managed in this same page (like price listings, statistics).

Once you finish, in XCode, go to **Archives**. It will automatically build the latest bundle with all final configurations.

The generated archives there are your app, ready to be distributed in the app store. You can deploy it to conclude the process without Expo managing it.

Note on build errors

If you are getting errors, in XCode, type Cmd+1, choose “Build Settings” -> “Linking” -> “Code stripping” -> “No”

This disables common errors produced by automatic code optimizations.

Publishing Android apps without Expo

Guides (Android) in the official React Native docs.

First thing first, we need to create a keystore key, which we will use to upload and update the app. It can be created from command line, using the command provided in the docs. Something like:

```
Maximilians-MBP:RNWithoutExpo mschwarzmueller$ keytool -genkeypair -v -keystore my-upload-key.keystore -alias my-key-alias -keyalg RSA -keysize 2048
```

My-Upload-key.keystore file is created, which can be used to sign your apps. Move it to android/ folder. Do NOT LOSE IT.

Some variables need to be set up in android.gradle now.

Setting up Gradle variables

1. Place the `my-upload-key.keystore` file under the `android/app` directory in your project folder.
2. Edit the file `~/.gradle/gradle.properties` or `android/gradle.properties`, and add the following (replace `*****` with the correct keystore password, alias and key password),

```
MYAPP_UPLOAD_STORE_FILE=my-upload-key.keystore
MYAPP_UPLOAD_KEY_ALIAS=my-key-alias
MYAPP_UPLOAD_STORE_PASSWORD=*****
MYAPP_UPLOAD_KEY_PASSWORD=*****
```

These are going to be global Gradle variables, which we can later use in our Gradle config to sign our app.

Note about security: If you are not keen on storing your passwords in plaintext, and you are running OSX, you can also store your credentials in the Keychain Access app. Then you can skip the two last rows in `~/.gradle/gradle.properties`.

Now, in ***android/build.gradle***, search for “**android**” node, and copy the signin configs stated:

Adding signing config to your app's Gradle config

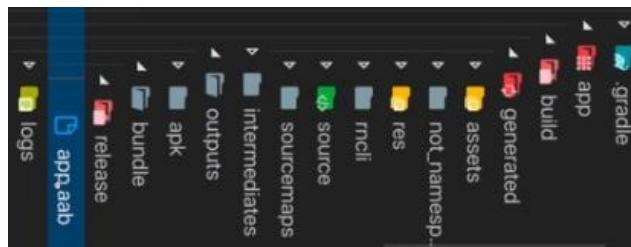
The last configuration step that needs to be done is to setup release builds to be signed using upload key. Edit the file ***android/app/build.gradle*** in your project folder, and add the signing config,

```
...  
    android {  
        ...  
        defaultConfig { ... }  
        signingConfigs {  
            release {  
                if (project.hasProperty('MYAPP_UPLOAD_STORE_FILE')) {  
                    storeFile file(MYAPP_UPLOAD_STORE_FILE)  
                    storePassword MYAPP_UPLOAD_STORE_PASSWORD  
                    keyAlias MYAPP_UPLOAD_KEY_ALIAS  
                    keyPassword MYAPP_UPLOAD_KEY_PASSWORD  
                }  
            }  
        }  
        buildTypes {  
            release {  
                ...  
                signingConfig signingConfigs.release  
            }  
        }  
    }  
    ...
```

Furhtermore, generate the APK by running:

```
$ cd android/ && ./gradlew bundleRelease
```

This builds your app and signs it for production. Once generated, you can locate it in ***app/build/outputs/bundle/release/app.aab***



We have the app ready for Google Playstore. Create a developer account (one-time fee), and log into Google Play Console.

Tap on ‘Create Application’, choose an app name, create.

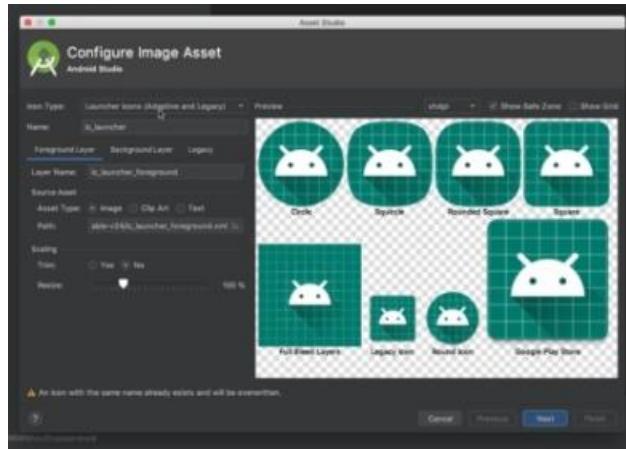
Under App Releases -> Production Track -> Create Release.

Configure the settings there (you can use defaults, but read on them), and upload your APK (.aab file)

Icons can be added with the help of Android Studio. Open the project there, navigate to `android/app/src/main/res`

An initialization load will start up, wait for its resolution.

Right click on `res/` -> `New` -> `image asset`. A screen will add where you can configure icons.



A foreground layer, background and legacy icons can also be configured. The splash screen can be configured using the docs.

Once this setup is finished, you re-run `./gradlew` command. Also, re-deploy the bundle in Google Console. **This ends the release steps for non-managed expo workflows.**

Extra notes

As shown earlier (when adding native modules to non-Expo apps), you can manage certain aspects of your Android app with the `AndroidManifest.xml` file.

There, you can configure three important things:

- The **App name** as it appears on the home screen:

<https://stackoverflow.com/questions/5443304/how-to-change-an-android-apps-name>

- The **bundle identifier & package name** of the app (also requires tweaking in other files):

<https://developer.android.com/studio/build/application-id>

- The **permissions** of the app:

<https://developer.android.com/guide/topics/manifest/intro#perms>

You should also set an **app version** and change it with every app update. This is done in the `build.gradle` file, see:

<https://developer.android.com/studio/publish/versioning>

Useful resources

Expo Deployment Docs:

<https://docs.expo.io/versions/v34.0.0/distribution/introduction/>

React Native - Android App Signing (Non-Expo) :

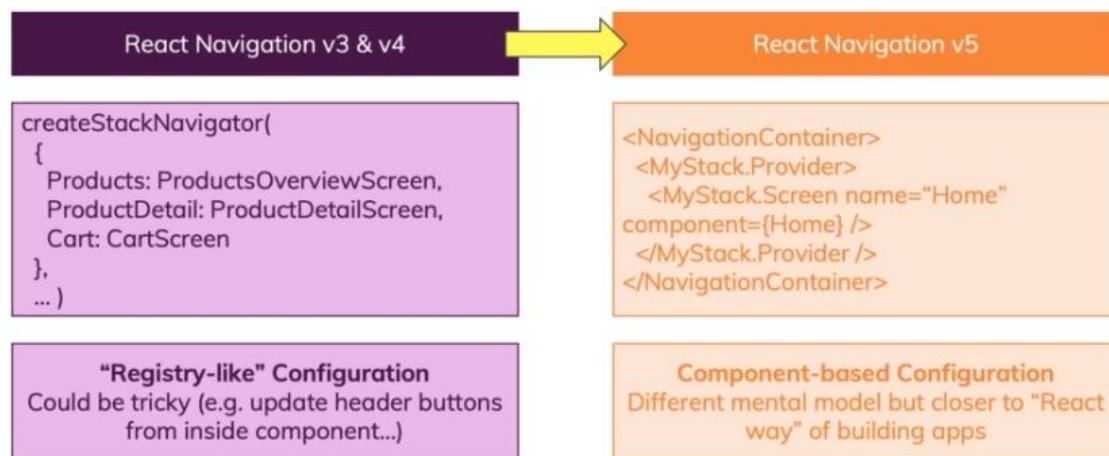
<https://facebook.github.io/react-native/docs/signed-apk-android>

React Native - Building the iOS App:

<https://facebook.github.io/react-native/docs/running-on-device#building-your-app-for-production>

Updating to React Navigation 5+

What changed?



Screens could be configured globally or in the local screen components, using `navigationOptions`. This works fine, but it can be tricky, for example, when trying to configure header buttons (we abuse of navigation params, not ideal, but the best way to do it).

With version 5, we do not set the configuration registry-like, but as a component tree (component-based). This allows us to pass data as props (hence, avoid overusing set/get params to handle parameters), and making the project look and feel like a React app at the same time.

The declarative syntax lets React and React-Navigation handle data and updates the way they consider more efficient, as the whole React philosophy states.

Note React versions 3 and 4 still exist in LOTS of projects out there. That's why we used it in our practice projects: to get a grasp of its workflow be ready if we happen to encounter them in the future.

Converting the project

We will convert **shopping-app/** to react navigation 5. First, let's install the latest version of Expo. Afterwards, the proper new version of react navigation.

```
$ expo upgrade && expo install @react-navigation/native
```

Everything broke now, on to fix one thing at a time, starting with Navigation container, which will need a new dummy stack for the time being. And while we are at it, the drawer navigation too. We will need it later.

```
$ expo install @react-navigation/stack @react-navigation/drawer
```

Navigation/AppNavigator.jsx

```
import React from "react"
import { useSelector } from "react-redux"
import { NavigationContainer } from "@react-navigation/native"
import { createStackNavigator } from "@react-navigation/stack"
import ProductsOverview from "../screens/shop/ProductsOverview"

const AppStack = createStackNavigator()

export default function AppNavigator(props) {
  const isAuthenticated = useSelector((state) => Boolean(state.auth.token))

  return <NavigationContainer>
    <AppStack.Navigator>
      <AppStack.Screen
        name='ProductsOverview' component={ProductsOverview} />
    </AppStack.Navigator>
  </NavigationContainer>
}
```

And this is the most basic setup we can configure for navigation using version 5. But we are far from done, as we must mimic all functionality we configured before.

Now, let's adjust replace imperative `navigationOptions` on each screen with an export, which will be used in **ShopNavigator**.

Screens/ProductOverview.jsx

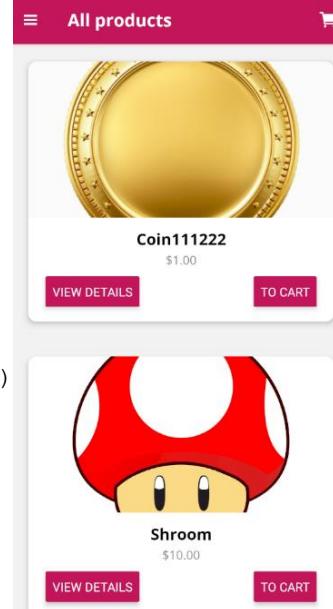
```
...
export default function ProductsOverview(props) {
  const [isLoading, setIsLoading] = useState(false)
  const [isRefreshing, setIsRefreshing] = useState(false)
  const [errorMsg, setErrorMsg] = useState("")
  const availableProducts = useSelector(
    (state) => state.products.availableProducts
  )
  const dispatch = useDispatch()

  const loadProducts = useCallback(async () => {
    setErrorMsg("")
    setIsRefreshing(true)
    try {
      await dispatch(productsActions.fetchProducts())
    } catch (err) {
      setErrorMsg(err.message)
    }
    setIsRefreshing(false)
  }, [setIsRefreshing, dispatch, setErrorMsg])

  useEffect(() => {
    setIsLoading(true)
    loadProducts().then(() => setIsLoading(false))
  }, [dispatch, loadProducts, setLoading])

  useEffect(() => {
    // no more 'willFocus' for screen events
    const unsubscribe =
      props.navigation.addListener("focus", loadProducts)
    return () => unsubscribe()
  }, [loadProducts])

  const handleSelectItem = (id, title) => {
    props.navigation.navigate("ProductDetails", {
      productId: id, productTitle: title
    })
  }
}
```



```

        return (
          <FetchViews
            errorMsg={errorMsg}
            errorButtonProps={{ title: "Refetch", onPress: loadProducts }}
            emptyResponseMsg="No products found" isLoading={isLoading}
            responseGate={availableProducts.length}
          >
            <FlatList
              data={availableProducts} refreshing={isRefreshing}
              onRefresh={loadProducts}
              renderItem={({ item }) => (
                <ProductItem image={item.imageUrl} title={item.title}
                  price={item.price}
                  onSelect={() => handleSelectItem(item.id, item.title)}
                >
                  <Button color={colors.PRIMARY} title="View details"
                    onPress={() => handleSelectItem(item.id, item.title)} />
                  <Button color={colors.PRIMARY} title="To cart"
                    onPress={() => dispatch(cartActions.addToCart(item))} />
                </ProductItem>
              ) }
            />
          </FetchViews>
        )
      }
    // ProductOverview.navigationOptions = ({ navigation }) => ({
    export const screenOptions = ({ navigation }) => ({
      headerTitle: "All products",
      headerRight: () => (
        // need `npm i --save @react-navigation/native`!
        <CustomHeaderButtons title="Cart" iconName="cart"
          onPress={() => navigation.navigate("Cart")}
        />
      ),
      headerLeft: () => (
        <CustomHeaderButtons title="Menu" iconName="menu"
          onPress={navigation.toggleDrawer}
        />
      )
    })
  )

```

Authentication logic handled in `Startup.jsx` (first screen to load), as well as reducer logic, also needs to be adjusted. This is such to avoid previously established `navigation.navigate` calls.

This is such because **react navigation v5+ does NOT have switch navigators**. You have to handle the logic to render screens in other ways, and redux will do the trick.

Store/actions/auth.js

```
import AsyncStorage from "@react-native-async-storage/async-storage"

const FB_API_KEY = `your_key`

export const AUTHENTICATE = "AUTHENTICATE"
export const LOGOUT = "LOGOUT"
export const SET_DID_TRY_AUTO_LOGIN = "SET_DID_TRY_AUTO_LOGIN"
...
export const setDidTryAutoLogin = () =>
  ({ type: SET_DID_TRY_AUTO_LOGIN })
...
...
```

Store/reducer/auth.js

```
import { AUTHENTICATE, LOGOUT, SET_DID_TRY_AUTO_LOGIN } from
"../actions/auth"

const initialState = {
  token: null, userId: null, didTryAutoLogin: false }

export default function authReducer(state = initialState, action) {
  switch (action.type) {
    case AUTHENTICATE:
      return { token: action.token, userId: action.userId,
        didTryAutoLogin: true }
    case LOGOUT: return { ...initialState, didTryAutoLogin: true }

    case SET_DID_TRY_AUTO_LOGIN:
      return { ...state, didTryAutoLogin: true }

    default: return state
  }
}
```

Screens/Startup.jsx

```
/**  
 * Screen to show at load, while checking if user is logged in using  
 * AsyncStorage.  
 */  
  
import AsyncStorage from "@react-native-async-storage/async-storage"  
...  
export default function Startup(props) {  
  const dispatch = useDispatch()  
  
  useEffect(() => {  
    const tryAuth = async () => {  
      const userData = await AsyncStorage.getItem("userData")  
  
      // if (!userData) return props.navigation.navigate("Auth")  
      if (!userData) return dispatch(authActions.setDidTryAutoLogin())  
  
      const { token, userId, expirationDate } = JSON.parse(userData)  
      // expiration date is an ISO string, we need a date object  
      const expiration = new Date(expirationDate)  
  
      // invalid auth if session expired, or no token or no registered user  
      if (expiration <= new Date() || !token || !userId) {  
        // return props.navigation.navigate("Auth")  
        return dispatch(authActions.setDidTryAutoLogin())  
      }  
  
      const expirationTime =  
        expiration.getTime() - new Date().getTime()  
  
      // props.navigation.navigate("Shop")  
      dispatch(  
        authActions.authenticate(userId, token, expirationTime)  
      )  
  
      tryAuth()  
    }, [])  
  
  return (  
    <View style={_styles.container}>
```

```

        <ActivityIndicator size="large" color={colors.PRIMARY} />
    </View>
)
}

const _styles = StyleSheet.create({
  container: sharedStyles.STRETCH_AND_CENTER
})

```

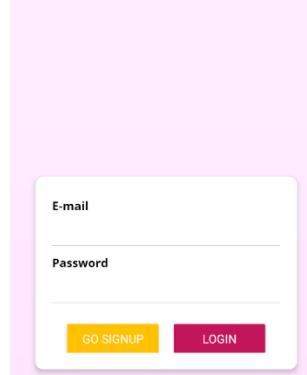
Note

At this point, all `navigation.navigate(<screenName>)` calls after screen actions' dispatch events have been removed across the app.

Navigation is fully controlled by `if` gates in `AppNavigator`. This is the intended behavior we tried to achieve with React Navigation v5 update.

Like in **Auth.jsx**

Authentication



Screens/Auth.jsx

```

...
const handleAuth = async () => {
  setIsLoading(true)
  setError("")
  const { email, password } = formState.values
  try {
    await dispatch(
      authActions[isSignup ? "signup" : "login"](email, password)
    )
    // props.navigation.navigate("Shop")
  } catch (err) { setError(err.message); setIsLoading(false) }
}
...

```

Screens/EdirProduct.jsx

```

...
export default function EditProduct(props) {
...
// const prodId = props.navigation.getParam("productId")
const prodId = props.route.params.productId ?? null // now `route`
...

```

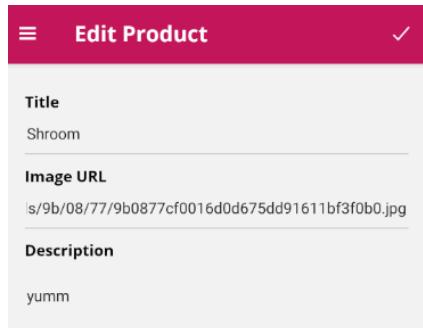
```

// no more `navigation.setParams({ handleSubmit })`, as we now
// can dynamically set navigation options from inside the component
useEffect(() => {
  props.navigation.setOptions({
    headerRight: () => (
      <CustomHeaderButtons title="Save" iconName="checkmark"
        onPress={handleSubmit ?? null} />
    )
  ), [handleSubmit])
}

return (...)

// headerRight is now in the component
export const screenOptions = ({ navigation, route }) => ({
  headerTitle:
    route.params?.productId ? "Edit Product" : "Add product",
  headerLeft: () => (
    <CustomHeaderButtons title="Menu" iconName="menu"
      onPress={navigation.toggleDrawer}
    />
  )
})

```



Navigation/AppNavigator.jsx

```

/**
 * NO more `SwitchNavigator` in React Navigation v5+, so we need to
 * handle conditional routing. It was removed due to now all logic
 * being handled with React components, so we can determine when to
 * load each navigator with `if` statements.
 */
import { NavigationContainer } from "@react-navigation/native"
import StartupScreen from "../screens/Startup"
import { ShopNavigator, AuthNavigator } from "./Shop"
...

export default function AppNavigator(props) {
  const isAuth = useSelector((state) => Boolean(state.auth.token))

```

```

const didTryAutoLogin =
  useSelector((state) => state.auth.didTryAutoLogin)

return (
  <NavigationContainer>
    {
      // render Shop screen if user is currently authenticated
      isAuth && <ShopNavigator />
    }
    {
      // render Auth screen if attempted auto-logged at app's startup
      // and user is not currently authenticated
      !isAuth && didTryAutoLogin && <AuthNavigator />
    }
    {!isAuth && !didTryAutoLogin && <StartupScreen />}
  </NavigationContainer>
)
}

```

Navigation/ShopNavigator.jsx

```

import React from "react"
import { Button, Platform, SafeAreaView, StatusBar, View } from
"react-native"
import { createStackNavigator } from "@react-navigation/stack"
import { createDrawerNavigator, DrawerItemList } from "@react-
navigation/drawer"
import { Ionicons } from "@expo/vector-icons"
import { useDispatch } from "react-redux"
import ProductsOverviewScreen, {
  screenOptions as productsOverviewOptions
} from "../screens/shop/ProductsOverview"
import ProductDetailsScreen, {
  screenOptions as productDetailsOptions
} from "../screens/shop/ProductDetails"
import OrdersScreen, {
  screenOptions as ordersOptions
} from "../screens/shop/Orders"
import UserProductsScreen, {
  screenOptions as userProductsOptions
} from "../screens/user/UserProducts"

```

```

import EditProductScreen, {
  screenOptions as editProductOptions
} from "../screens/user/EditProduct"

import CartScreen, { screenOptions as cartOptions } from
"../screens/shop/Cart"

import AuthScreen, { screenOptions as authOptions } from
"../screens/user/Auth"

import colors from "../constants/colors"
import * as authActions from "../store/actions/auth"

const sharedDefaultNavOptions = {

  headerStyle: {

    backgroundColor: Platform.OS === "android" ? colors.PRIMARY : ""

  },

  headerTitleStyle: { fontFamily: "open-sans-bold" },

  headerBackTitleStyle: { fontFamily: "open-sans" },

  headerTintColor:

    Platform.OS === "android" ? "white" : colors.PRIMARY

}

const ProductsStack = createStackNavigator()

export const ProductsNavigator = () => (
  <ProductsStack.Navigator screenOptions={sharedDefaultNavOptions}>
    <ProductsStack.Screen name="ProductsOverview"
      component={ProductsOverviewScreen}
      options={productsOverviewOptions}
    />
    <ProductsStack.Screen name="ProductDetails"
      component={ProductDetailsScreen} options={productDetailsOptions}
    />
    <ProductsStack.Screen name="Cart" component={CartScreen}
      options={cartOptions}
    />
  </ProductsStack.Navigator>
)

const OrdersStack = createStackNavigator()

export const OrdersNavigator = () => (
  <OrdersStack.Navigator screenOptions={sharedDefaultNavOptions}>

```

```

        <OrdersStack.Screen
            name="OrdersScreen" // avoids name clash with Drawer below
            component={OrdersScreen} options={ordersOptions} />
    </OrdersStack.Navigator>
)

const AdminStack = createStackNavigator()

export const AdminNavigator = () => (
    <AdminStack.Navigator screenOptions={sharedDefaultNavOptions}>
        <AdminStack.Screen name="UserProducts"
            component={UserProductsScreen} options={userProductsOptions} />
        <AdminStack.Screen name="EditProduct"
            component={EditProductScreen} options={editProductOptions} />
    </AdminStack.Navigator>
)

const AuthStack = createStackNavigator()

export const AuthNavigator = () => (
    <AuthStack.Navigator screenOptions={sharedDefaultNavOptions}>
        <AuthStack.Screen
            name="Auth" component={AuthScreen} options={authOptions} />
    </AuthStack.Navigator>
)

const getDrawerContent = (props, dispatch) => (
    <View style={{ flex: 1 }}>
        <SafeAreaView
            forceInset={{ top: "always", horizontal: "never" }}
            style={{ paddingTop: StatusBar.currentHeight }}>
        <!-- was `DrawerItems` on React Navigation v4-->
        <DrawerItemList {...props} />
        <Button title="logout" color={colors.PRIMARY}>
            onPress={() => dispatch(authActions.logout())}
        </Button>
    </SafeAreaView>
</View>
)

```

```

const getDrawerOptions = (iconName) => ({
  drawerIcon: (props) => (
    <Ionicons
      name={Platform.OS === "android" ?
        `md-${iconName}` : `ios-${iconName}`}
      size={23}
      color={props.color} // now "color" instead of "tintColor"
    />
  )
})
}

const ShopDrawer = createDrawerNavigator()

export const ShopNavigator = () => {
  const dispatch = useDispatch()

  return (
    <ShopDrawer.Navigator
      screenOptions={{
        drawerActiveTintColor: colors.PRIMARY,
        // hide default header. We render one on each component
        headerShown: false
      }}
      drawerContent={(props) => getDrawerContent(props, dispatch)}
    >
      <ShopDrawer.Screen
        name="Products" component={ProductsNavigator}
        options={getDrawerOptions("cart")} />
      <ShopDrawer.Screen
        name="Orders" component={OrdersNavigator}
        options={getDrawerOptions("list")} />
      <ShopDrawer.Screen
        name="Admin" component={AdminNavigator}
        options={getDrawerOptions("create")} />
    </ShopDrawer.Navigator>
  )
}

```

```
// ***** REACT NAVIGATION V4 *****
// const ProductsNavigator = createStackNavigator(
//   {
//     ProductsOverview: ProductsOverviewScreen,
//     ProductDetails: ProductDetailsScreen,
//     Cart: CartScreen
//   },
//   {
//     initialRouteName: "ProductsOverview",
//     navigationOptions: createSharedNavOptions("cart"),
//     defaultNavigationOptions: sharedDefaultNavOptions
//   }
// )
///////////
// const OrdersNavigator = createStackNavigator(
//   { Orders: OrdersScreen },
//   {
//     navigationOptions: createSharedNavOptions("list"),
//     defaultNavigationOptions: sharedDefaultNavOptions
//   }
// )

// const AdminNavigator = createStackNavigator(
//   {
//     UserProducts: UserProductsScreen,
//     EditProduct: EditProductScreen
//   },
//   {
//     navigationOptions: createSharedNavOptions("create"),
//     defaultNavigationOptions: sharedDefaultNavOptions
//   }
// )

// const AuthNavigator = createStackNavigator(
//   { Auth: AuthScreen },
//   { defaultNavigationOptions: sharedDefaultNavOptions }
// )

// const ShopNavigator = createDrawerNavigator(
```

```
//      Products: ProductsNavigator,
//      Orders: OrdersNavigator,
//      Admin: AdminNavigator
//    },
//  {
//    contentOptions: { activeTintColor: colors.PRIMARY },
//    contentComponent: (props) => {
//      const dispatch = useDispatch()

//      return (
//        <View style={{ flex: 1 }}>
//          <SafeAreaView
//            forceInset={{ top: "always", horizontal: "never" }}
//            style={{ paddingTop: StatusBar.currentHeight }}
//          >
//            {/* `DrawerNavigationItems` on latest React Navigation */}
//            <DrawerItems {...props} />
//            <Button
//              title="logout"
//              color={colors.PRIMARY}
//              onPress={() => {
//                dispatch(authActions.logout())
//                props.navigation.navigate("Auth")
//              }}
//            />
//          </SafeAreaView>
//        </View>
//      )
//    }
//  }
//)

// const MainNavigator = createSwitchNavigator({
//   Startup: StartupScreen,
//   Auth: AuthNavigator,
//   Shop: ShopNavigator
// })

// export default createAppContainer(MainNavigator)
```

React Navigation v5 summary

The biggest change is the way we configure the main navigator.

Before, we had a declarative “configurable” registry-like approach:

```
const OrdersNavigator = createStackNavigator(  
  { Orders: OrdersScreen },  
  {  
    navigationOptions: createSharedNavOptions("list"),  
    defaultNavigationOptions: sharedDefaultNavOptions  
  }  
)
```

Now, everything is component based, which is React's declarative paradigm logic:

```
const OrdersStack = createStackNavigator()  
  
export const OrdersNavigator = () => (  
  <OrdersStack.Navigator  
    screenOptions={sharedDefaultNavOptions}  
  >  
  <OrdersStack.Screen  
    name="Orders" component={OrdersScreen}  
    options={ordersOptions}  
  />  
  </OrdersStack.Navigator>  
)
```

Notice how we now have any type of navigator (stack, tab, drawer) configurable in the exact same way, all acting as wrappers to the screens they can navigate to.

Navigating to different screens is left untouched. The value of the first arg must always match the name of the screen you are moving to.

```
const handleEditProduct = (id) => {  
  props.navigation.navigate("EditProduct", { productId: id })  
}  
  
const goBack = () => props.navigation.goBack()
```

Some other navigation methods were added too, like in drawer navigation.

```
const jumpToPage = (page) => props.navigation.jumpTo(page)
```

How you nest navigators was also left untouched, but with configuration reworked under the new component-based approach:

```
<ShopDrawer.Navigator
  screenOptions={{
    drawerActiveTintColor: colors.PRIMARY,
    headerShown: false
  }}
  drawerContent={(props) => getDrawerContent(props, dispatch)}
>
  {/* these are all StackNavigators */}
  <ShopDrawer.Screen
    name="Products" component={ProductsNavigator}
    options={getDrawerOptions("cart")} />
  <ShopDrawer.Screen
    name="Orders" component={OrdersNavigator}
    options={getDrawerOptions("list")} />
  <ShopDrawer.Screen
    name="Admin" component={AdminNavigator}
    options={getDrawerOptions("create")} />
</ShopDrawer.Navigator>
```

Note how default options for all screens are set in the navigator component, while individual options for each screen, inside the screen components.

SwitchNavigator was completely removed, as with this new component-based approach, you can easily use route gates the traditional way.

```
<NavigationContainer>
  { isAuthenticated && <ShopNavigator /> }
  { !isAuthenticated && didTryAutoLogin && <AuthNavigator /> }
  { !isAuthenticated && !didTryAutoLogin && <StartupScreen /> }
</NavigationContainer>
```

Params are located inside `route.params` with the property named exactly the same as we set the params to. `route` being a new object made available by React Navigation, alongside `navigation`.

```
const productId = props.route.params.productId
```

Static params are still set when navigating to each route, in an object located on the second argument of `navigation.navigate`.

```
props.navigation.navigate("EditProduct", { productId: id })
```

And dynamic params (ones that change depending on the app's state), are no longer set, but directly configured inside the component with `navigation.setOptions`.

```
useEffect(() => {
  props.navigation.setOptions({
    headerRight: () => (
      <CustomHeaderButtons title="Save" iconName="checkmark"
        onPress={handleSubmit ?? null} />
    )
  })
}, [handleSubmit])
```

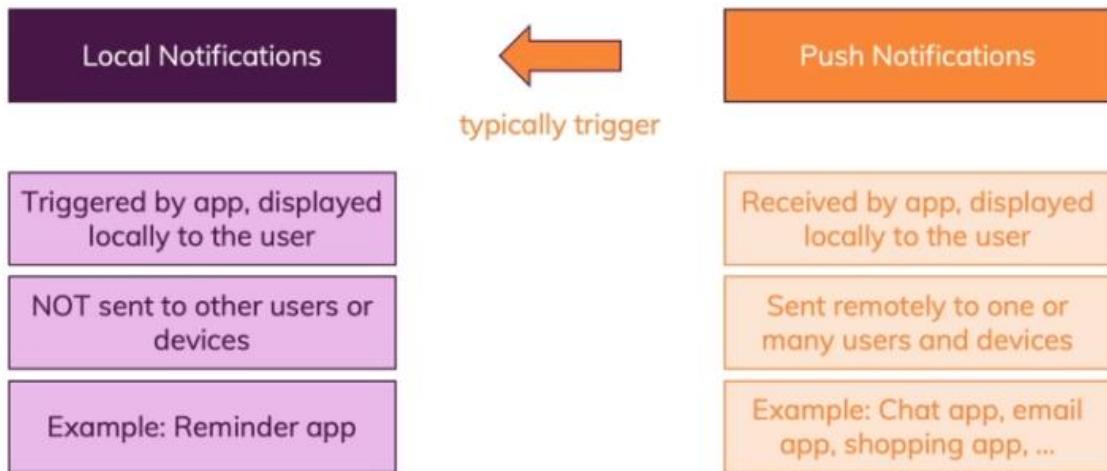
Old projects will certainly use React Navigation version 4 or below, while new ones have a higher chance of applying version 5 onwards. You need to know both. So many projects still use RN v4- that React Team decided to keep it up to date.

And as always, the official docs are your friend. READ THEM.

<https://reactnavigation.org/docs/4.x/getting-started/>

Push notifications

Local versus Push



Sending local notifications

As you would expect, notifications are way easier to be handled by expo, though you can also use them on native cli. We will see them with expo, on a new app.

```
$ expo install expo-notifications
```

First, background notifications, as foreground ones need a different configuration. Foreground notifs play when the app is in foreground, while background is when the app is minimized.

App.json

```
{  
  "expo": {  
    "name": "push-notifications",  
    ...  
    "android": {  
      "useNextNotificationsApi": true  
      ...  
    },  
  }  
}
```

App.js

```
import { StatusBar } from "expo-status-bar"
import React from "react"
import * as Notifications from "expo-notifications"
import { Button, StyleSheet, View } from "react-native"

export default function App() {
  const triggerNotification = () => {
    // create a local notifications.
    // There are a bunch of props!
    Notifications.scheduleNotificationAsync({
      content: {
        title: "First notif!",
        body: "Not an annoying notif"
      },
      trigger: { seconds: 3 }
    })
  }

  return (
    <View style={styles.container}>
      <Button title="Show notification"
        onPress={triggerNotification} />
      <StatusBar style="auto" />
    </View>
  )
}
```

SHOW NOTIFICATION



When you click on the notification, you are taken back to the app screen.

Permissions

On Android, permissions are handled by default. On iOS, you explicitly need to ask for them.

App.jsx

```
import React, { useEffect } from "react"
import * as Notifications from "expo-notifications"
import { Alert, Button, StyleSheet, View } from "react-native"
```

```

export default function App() {
  useEffect(() => {
    // this does nothing on Android
    Notifications.getPermissionsAsync()
      .then((res) => {
        if (res.status !== "granted") {
          return Notifications.requestPermissionsAsync()
        }
        return res
      })
      .then((res) => {
        if (res.status !== "granted") {
          return Alert.alert("Failed", "", [{ text: "ok" }])
        }
      })
    },
    []
)

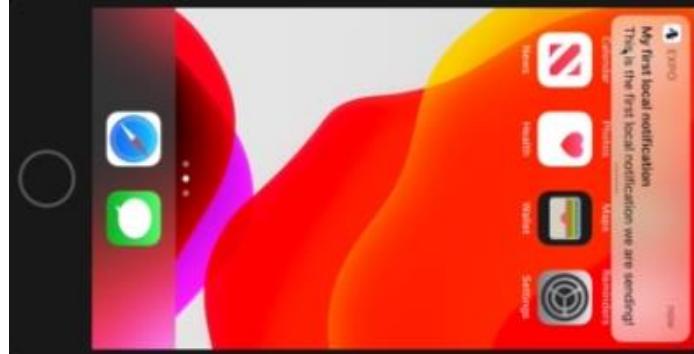
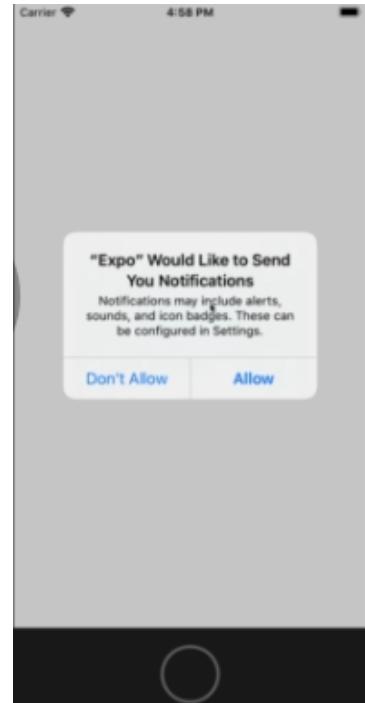
```

```

const triggerNotification = () => {
  Notifications.scheduleNotificationAsync({
    content: {
      title: "First notif!",
      body: "Not an annoying notif"
    },
    trigger: { seconds: 3 }
  })
}

return (
  <View style={styles.container}>
    <Button title="Show notification"
      onPress={triggerNotification} />
  </View>
)
}

```

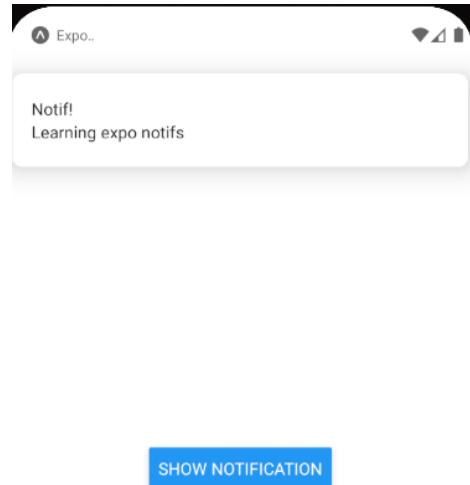


Controlling how notifications are displayed

While the app is running, the notification will be lost. To handle it, we set a listener.

App.jsx

```
...
// this executes when the notification arrives, before displaying it
// > It MUST return a promise, so just return an async function
// > The return of that promise must be an object with configs
Notifications.setNotificationHandler({
  handleNotification: async () => ({
    shouldShowAlert: true // display alert even on foreground
  })
})
export default function App() { ... }
```



Reacting to foreground notifications

You can react with a listener to notifications when they are received, as well as when the user taps on them.

There are two listeners for such task.

App.jsx

```
...
useEffect(() => {
  // triggers when app is running and notif is received
  const foregroundSubscription =
    Notifications.addNotificationReceivedListener(
      (notification) => { console.log(notification) }
    )
})
```

```

// triggers when app is running and notif is clicked
const backgroundSubscription =
  Notifications.addNotificationResponseReceivedListener(
    (response) => { console.log(response) }

return () => {
  foregroundSubscription.remove()
  backgroundSubscription.remove()
} , [])
}

const triggerNotification = () => {
  // create a local notifications. There are a bunch of props!
  Notifications.scheduleNotificationAsync({
    content: {
      title: "Notif!", body: "Learning expo notifs",
      data: { extra: ["extra", "data"] }
    },
    trigger: { seconds: 3 }
  })
}

...
// addNotificationReceivedListener (both devices)

Object {
  "date": 1630372679799,
  "request": Object {
    "content": Object {
      "autoDismiss": true,
      "badge": null,
      "body": "Learning expo notifs",
      "data": Object {
        "extra": Array [
          "extra",
          "data",
        ],
      },
      "sound": "default",
      "sticky": false,
      "subtitle": null,
      "title": "Notif!",
    },
    "identifier": "a197d8b4-9ad7-430b-",
    "trigger": Object {
      "channelId": null,
      "repeats": false,
      "seconds": 3,
      "type": "timeInterval",
    },
  },
}

```

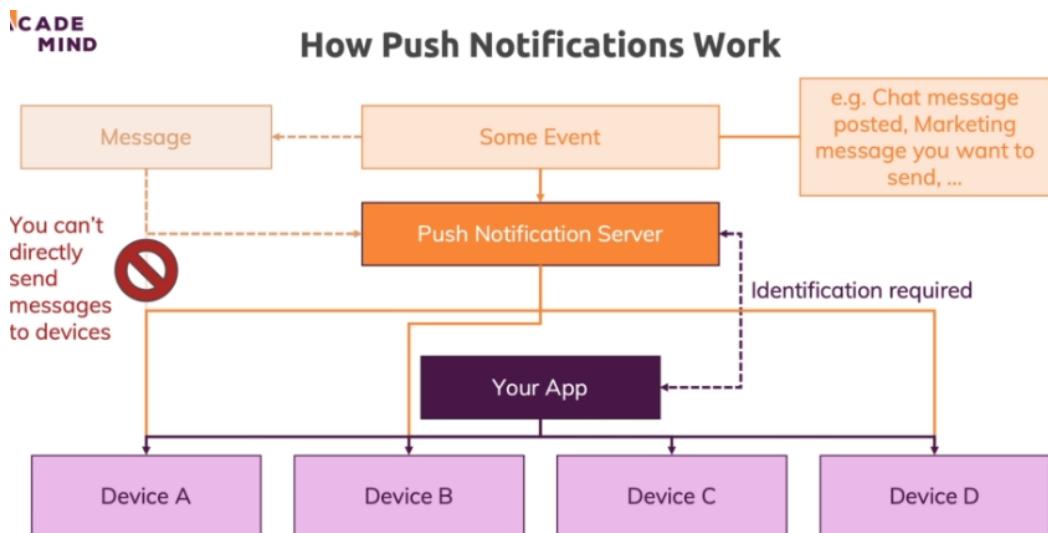
```
// addNotificationResponseReceivedListener (android / ios)

Object {
  "actionIdentifier": "expo.modules.notifications.actions.DEFAULT",
  "notification": Object {
    "date": 1630373469190,
    "request": Object {
      "content": Object {
        "autoDismiss": true,
        "badge": null,
        "body": "Learning expo notifs",
        "data": Object {
          "extra": Array [
            "extra",
            "data",
          ],
        },
        "sound": "default",
        "sticky": false,
        "subtitle": null,
        "title": "Notif!",
      },
      "identifier": "b285795d-293e-467a-a804-b634d64292ae",
      "trigger": Object {
        "channelId": null,
        "repeats": false,
        "seconds": 2,
        "type": "timeInterval",
      },
    },
  },
}
```

```
"actionIdentifier": "expo.modules.notifications.actions.DEFAULT",
"notification": Object {
  "date": 1594307664.899376,
  "request": Object {
    "content": Object {
      "attachments": Array [],
      "badge": null,
      "body": "This is the first local notification we are sending!",
      "categoryIdentifier": "",
      "data": Object {
        "experienceId": "@anonymous/react-native-push-0068f27a-c147-4",
        "mySpecialData": "Some text",
      },
      "launchImageName": "",
      "sound": null,
      "subtitle": null,
    },
  },
}
```

How push notifications work

Local notifications (the ones used up to now) are triggered by code inside the app. Push notifications, on the other side, are triggered from outside the app (server notification, developer pushed a notification, message arrived).



Expo and push notifications

Permissions are required for push notifications too (iOS only, Android enables on installation). Luckily, the ones we set up earlier work.

Push notifications trigger local ones, so the code to display and receive local notifications that is also written already, works.

The app needs an identification token for both Android and iOS, they are given by Google and Apple's servers.

Additionally, your app needs to be signed not only by the author, but also using that push token.

Expo's team has its own logic to sign itself up to get the tokens needed on their servers, and delivers those tokens to us. That's extremely convenient, as we only need to request for it with some lines of code. Manual signup is extremely tedious.

Note: Push notifications do NOT work on emulators, you need to test it on real devices.

Note: Push notifications NEED you to be logged in to your Expo developer account to work when in test mode.

```
> Create the account on Expo's website  
> $ expo login
```

You can test Expo notifications with Expo's push notification tool.

<https://expo.dev/notifications>

In reality, you would use Expo's server side push notifications SDKs to generate and send notifications, or directly from inside your app.

Note: Upon sending a push notification, the same code for receiving foreground/background notifications will be triggered.

App.jsx

```
import * as Notifications from "expo-notifications"  
  
...  
  
// this executes when the notification arrives, before displaying it  
// > It MUST return a promise, so just return an async function  
// > The return of that promise must be an object with configs  
Notifications.setNotificationHandler({  
  handleNotification: async () => ({  
    shouldShowAlert: true // display alert even on foreground  
  })  
})
```

```

export default function App() {
  useEffect(() => {
    // this does nothing on Android
    Notifications.getPermissionsAsync()
      .then((res) => {
        if (res.status !== "granted") {
          return Notifications.requestPermissionsAsync()
        }
        return res
      })
      .then((res) => {
        if (res.status !== "granted") {
          Alert.alert("permissions required", "", [{ text: "ok" }])
          // res.status can be "undetermined", which will still fall
          // to the next `then` block. We do not want that.
          throw new Error("Notification permissions required")
        }
      })
      .then(() => {
        // only scenario in this block is "res.status === 'granted'"
        return Notifications.getExpoPushTokenAsync()
      })
      .then((res) => { console.log(res.data) }) // res.data image below
      .catch((err) => console.log(err))
    }, [])
  }

  // runs also when a push notification is received
  useEffect(() => {
    const foregroundSubscription =
      Notifications.addNotificationReceivedListener((notification) => {
        console.log("notification", notification)
      })
    const backgroundSubscription =
      Notifications.addNotificationResponseReceivedListener((response) => {
        console.log("response", response)
      })
    return () => {
      foregroundSubscription.remove()
      backgroundSubscription.remove()
    }
  }, [])
}

```

```

const triggerNotification = () => {...}

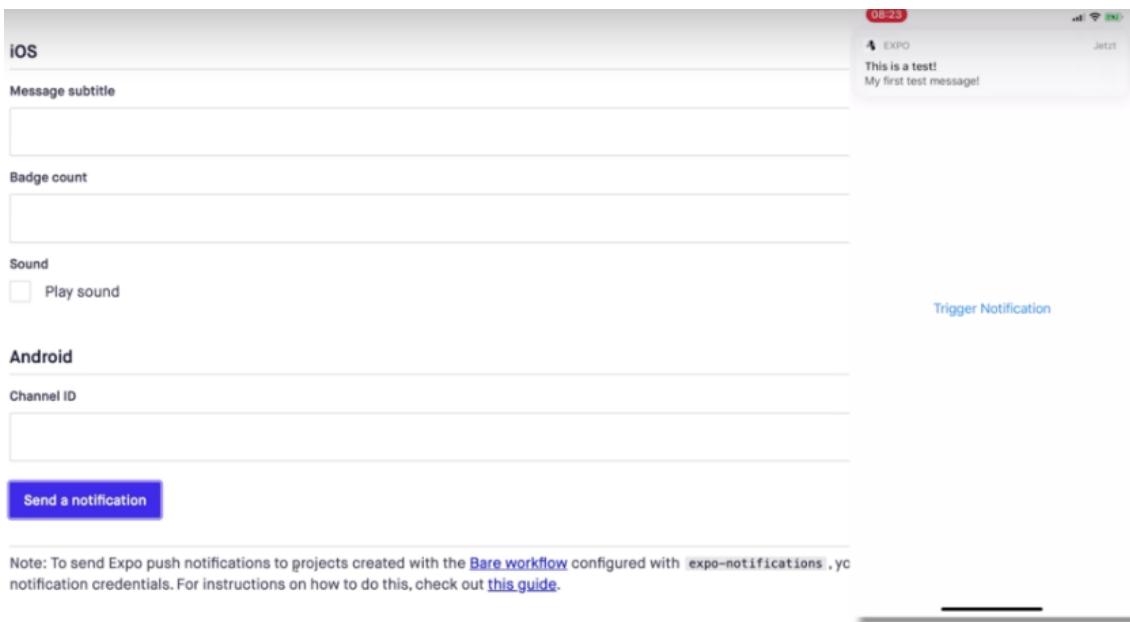
return ...
}

// token object retrieved by Notifications.getExpoPushTokenAsync()

Object {
  "data": "ExponentPushToken[REDACTED]", 
  "type": "expo", 
}

```

// on push notification sent by expo notification tool



Using expo's push server

An http request needs to be sent to Expo's push server wherever you wish to trigger a notification from inside the app.

In our case, it's the `triggerNotification` handler, and the fetch url is <https://exp.host/--/api/v2/push/send>.

A notification sent to a push token is received only by the device linked to that token (with the downloaded app hosting that token).

This means that up to now we only know one single push token, when push notifications are designed to be received by all devices running the app.

Then, we must handle it ourselves in our own backend servers. When we receive the token from `Notifications.getExpoPushTokenAsync()`, we also send an http POST request with that token to be stored in a database we manage.

Same goes for retrieving all tokens.

There are many ways to send a push notification with Expo's push API besides a fetch request. Read the docs!

<https://docs.expo.dev/push-notifications/sending-notifications/>

Notifications can be triggered from the server, as stated in the docs. Many languages are supported, like Node, Ruby, Python, GoLang, and so on.

Under the hood, those servers just do the same as we do inside the app, simply send a notification using expo's push servers.

App.jsx

```
import * as Notifications from "expo-notifications"

...

// this executes when the notification arrives, before displaying it
// > It MUST return a promise, so just return an async function
// > The return of that promise must be an object with configs
Notifications.setNotificationHandler({
  handleNotification: async () => ({
    shouldShowAlert: true // display alert even on foreground
  })
})

export default function App() {
  const [pushToken, setPushToken] = useState(null)

  useEffect(() => {
    // this does nothing on Android
    Notifications.getPermissionsAsync()
      .then((res) => {
        if (res.status !== "granted") {
          return Notifications.requestPermissionsAsync()
        }
        return res
      })
      .then((res) => {
        if (res.status !== "granted") {
          Alert.alert("Permissions required", "", [{ text: "ok" }])
          // res.status can be "undetermined", which will still fall
          // to the next `then` block. We do not want that.
          throw new Error("Notification permissions required")
        }
      })
  })
}
```

```

        }
    })
    .then(() => {
        // only scenario in this block is "res.status === 'granted'"
        return Notifications.getExpoPushTokenAsync()
    })
    .then((res) => {
        setPushToken(res.data) // `res.data` being the expo token
        // > store the token in your own managed database
        // fetch('https://your-server-to-store-token.com',
        // { method: "POST", ..., data: { token: res.data }})
    })
    .catch((err) => console.log(err))
}, [])

```

```

useEffect(() => {
    // triggers when app is running and notif is recieved
    const foregroundSubscription =
        Notifications.addNotificationReceivedListener((notification) => {
            console.log("notification", notification)
        })

    // triggers when app is running and notif is clicked
    const backgroundSubscription =
        Notifications.addNotificationResponseReceivedListener((response) => {
            console.log("response", response)
        })

```

```

    return () => {
        foregroundSubscription.remove()
        backgroundSubscription.remove()
    }
}, [])

```

```

const triggerNotification = () => {
    // create a local notifications. There are a bunch of props!
    // Notifications.scheduleNotificationAsync({
    //   content: {
    //     title: "Notif!",
    //     body: "Learning expo notifs",
    //     data: { extra: ["extra", "data"] }
}

```

```

        //      },
        //      trigger: { seconds: 1 }
        // })

// create a push notification. All of this in docs.
fetch("https://exp.host/--/api/v2/push/send", {
  method: "POST",
  headers: {
    Host: "exp.host",
    Accept: "application/json",
    "Accept-Encoding": "gzip, deflate",
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    to: pushToken,
    data: { extra: "some extra data" },
    title: "Sent by the app",
    body: "This push notification was sent by the app!"
  })
})

}

return (
  <View style={styles.container}>
    <Button title="Show notification" onPress={triggerNotification} />
  </View>
)
}

const styles = StyleSheet.create({
  container: {
    flex: 1, backgroundColor: "#fff",
    alignItems: "center", justifyContent: "center"
  }
})

```

Adding notifications to shopping app (1/3)

Let's apply all what we learned into the Shopping app. We will show a notification when a product is added.

For that, first we will add the push token to the added product's state in the reducer so that when we fetch the product from firebase, we can also later fetch the push token.

First goal: when a user adds a product, they will get notified. This is the first step.

```
$ expo install expo-notifications
```

App.json

```
...
"android": {
  "adaptiveIcon": {
    "foregroundImage": "./assets/adaptive-icon.png",
    "backgroundColor": "#FFFFFF"
  },
  "useNextNotificationsApi": true
},
...
...
```

Store/actions/products.js

```
import * as Notifications from "expo-notifications"
import { Alert } from "react-native"
import { Product } from "../../models/product"
...

export const createProduct = (title, description, imageUrl, price) =>
{
  return async (dispatch, getState) => {
    // get current permissions (iOS only, Android asks at
    // installation)
    let permission = await Notifications.getPermissionsAsync()
    let pushToken = null

    if (permission.status !== "granted") {
      // request for permissions if not already granted
      permission = await Notifications.requestPermissionsAsync()
    }
  }
}
```

```

if (permission.status !== "granted") {
    // if no permissions were granted upon request, nothing to do
    console.log("Permissions not granted for notifications.")
} else {
    // permissions granted. Save push token in `response.data`
    pushToken = (await Notifications.getExpoPushTokenAsync()).data
}

try {
    const { token, userId } = getState().auth

    const response = await fetch(
        `https://rnmcshoppingapp-default.firebaseio.com/
products.json?auth=${token}`,
        {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                title, description, imageUrl, price, userId,
                pushToken // user's push token (null or defined)
            })
        }
    )
}

const data = await response.json()

dispatch({
    type: CREATE_PRODUCT,
    // id is returned as "name" key in generated document
    payload: {
        id: data.name, title, description, imageUrl, price, userId
    }
} catch (err) {
    console.log(err)
}
}

```

↓

```

-MiTVgWA9I1h3SV6-aii
  description: "asdfasd"
  imageUrl: "\http://st.depositphotos.com/1000198/4574/i/450...
  price: 123
  pushToken: "ExponentPushToken[EQk5hpP5uLEoF4MowC6VRI]"
  title: "asdf"
  userId: "6N6tEuJ5z8hMmApk98KdPbxGWT1"

```

Adding notifications to shopping app (2/3)

Now, let's set up notifications to show when a user orders a product.

Notice we are working with Firebase, which indeed is a backend. This means we can also send push notifications using Expo SDK there.

This is done by sending an http request as we learned before, but from firebase. However, we do not own Firebase to write the code we wish. Instead, we can use its cloud functions. This is not free, though.

For this reason, we will do it from the app.

Models/cart.js

```
export default class CartItem {  
  constructor(quantity, price, title, pushToken, sum) {  
    this.quantity = quantity  
    this.price = price  
    this.title = title  
    this.pushToken = pushToken  
    this.sum = sum  
  }  
}
```

Store/actions/cart.js

```
...  
  
const initialState = { items: {}, total: 0 }  
  
const cartReducer = (state = initialState, action) => {  
  switch (action.type) {  
    case ADD_TO_CART:  
      const { id, price, title, pushToken } = action.product  
      let updatedOrNewCartItem = {}  
  
      if (state.items[id]) {  
        updatedOrNewCartItem = new CartItem(  
          state.items[id].quantity + 1, price, title,  
          pushToken, state.items[id].sum + price  
        )  
      } else {  
        updatedOrNewCartItem =  
          new CartItem(1, price, title, pushToken, price)  
      }  
      return {  
        ...state,  
        items: { ...state.items, [id]: updatedOrNewCartItem }  
      }  
    }  
  }  
  return state;  
}  
  
export default cartReducer
```

```
    ...state,  
    items: { ...state.items, [id]: updatedOrNewCartItem },  
    total: state.total + price  
}  
  
...  
  
...
```

Screens/shop/Cart.jsx

```
...  
  
export default function Cart(props) {  
  const [isLoading, setIsLoading] = useState(false)  
  
  const total = useSelector((state) => state.cart.total)  
  const dispatch = useDispatch()  
  
  const items = useSelector((state) => {  
    const transformedCartItems = []  
    for (const key in state.cart.items) {  
      const currentItem = state.cart.items[key]  
      transformedCartItems.push({  
        id: key,  
        title: currentItem.title,  
        price: currentItem.price,  
        quantity: currentItem.quantity,  
        sum: currentItem.sum,  
        pushToken: currentItem.pushToken  
      })  
    }  
    return transformedCartItems.sort((a, b) => (a.id > b.id ? 1 : -1))  
  })  
  
  const handleSendOrder = async () => {  
    setIsLoading(true)  
    await dispatch(orderActions.addOrder(items, total))  
    setIsLoading(false)  
  }  
  
  return (  
    <View style={_styles.container}>  
      <Card style={_styles.summary}>  
        <Text style={_styles.summaryText}>
```

```

        Total:{"  "}
        <Text style={_styles.amount}>
          ${Math.round(total.toFixed(2) * 100) / 100}
        </Text>
      </Text>
      {isLoading ? (
        <ActivityIndicator size="small" color={colors.PRIMARY} />
      ) : (
        <Button color={colors.SECONDARY} title="Order"
          disabled={items.length === 0} onPress={handleSendOrder} />
      )
    </Card>
    <FlatList data={items}
      renderItem={({ item: { id, quantity, title, sum } }) => (
        <CartItem
          quantity={quantity} title={title} amount={sum}
          onDelete={() => dispatch(cartActions.removeFromCart(id))}
        />
      )
    >
  </View>
)
}

export const screenOptions = { headerTitle: "Your cart" }

```

models/product.js

```

export class Product {
  constructor(id, userId, pushToken, title, imageUrl, description, price) {
    this.id = id
    this.userId = userId
    this.pushToken = pushToken
    this.title = title
    this.imageUrl = imageUrl
    this.description = description
    this.price = price
  }
}

```

Store/actions/product.js

```
...

export const fetchProducts = () => {
  return async (dispatch, getState) => {
    try {
      const response = await fetch(
        "https://rnmcshoppingapp-default.firebaseio.com/products.json"
      )

      if (!response.ok) throw new Error("Something went wrong")

      const userId = getState().auth.userId
      const data = await response.json()
      const products = []

      for (const key in data) {
        const values = data[key]
        products.push(
          new Product(
            key, values.userId,
            values.pushToken, // key extracted from firebase
            values.title, values.imageUrl,
            values.description, values.price
          )
        )
      }
    }

    dispatch({
      type: FETCH_PRODUCTS,
      payload: {
        availableProducts: products,
        userProducts: products.filter((p) => p.userId === userId)
      }
    })
  } catch (err) { console.log(err); throw err }
}

export const createProduct = (
  title, description, imageUrl, price) => {
```

```
return async (dispatch, getState) => {
  let permission = await Notifications.getPermissionsAsync()
  let pushToken = null

  if (permission.status !== "granted") {
    permission = await Notifications.requestPermissionsAsync()
  }

  if (permission.status !== "granted") {
    console.log("Permissions not granted for notifications.")
  } else {
    pushToken = (await Notifications.getExpoPushTokenAsync()).data
  }

  try {
    const { token, userId } = getState().auth
    const response = await fetch(
      `https://rnmcsShoppingApp-default.firebaseio.com/
        products.json?auth=${token}`,
      {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ title, description, imageUrl,
          price, userId, pushToken
        })
      }
    )

    const data = await response.json()

    dispatch({
      type: CREATE_PRODUCT,
      // id is returned as "name" key in generated document
      payload: { id: data.name, userId, pushToken,
        title, description, imageUrl, userId
      }
    })
  } catch (err) { console.log(err) }
}

}
```

Store/reducer/products.js

```
...
const initialState = { availableProducts: [], userProducts: [] }

const productsReducer = (state = initialState, action) => {
  switch (action.type) {
    case FETCH_PRODUCTS: ...
    case DELETE_PRODUCT: ...
    case CREATE_PRODUCT:
      const newProduct = new Product(
        action.payload.id, // created in firebase, as "name" field
        action.payload.userId, // created in firebase, when logging in
        action.payload.pushToken, // stored, when adding a product
        action.payload.title,
        action.payload.imageUrl,
        action.payload.description,
        action.payload.price
      )
      return {
        ...state,
        availableProducts: state.availableProducts.concat(newProduct),
        userProducts: state.userProducts.concat(newProduct)
      }
    case UPDATE_PRODUCT:
      const idxInUserProducts = state.userProducts.findIndex(
        (prod) => prod.id === action.payload.id
      )
      const idxInAvailableProducts = state.availableProducts.findIndex(
        (prod) => prod.id === action.payload.id
      )
      const updatedProduct = new Product(
        action.payload.id,
        state.userProducts[idxInUserProducts].userId,
        state.userProducts[idxInUserProducts].pushToken,
        action.payload.title,
        action.payload.imageUrl,
        action.payload.description,
        state.userProducts[idxInUserProducts].price
      )
      const updatedUserProducts = [...state.userProducts]
      const updatedAvailableProducts = [...state.availableProducts]
```

```

        updatedUserProducts[idxInUserProducts] = updatedProduct
        updatedAvailableProducts[idxInAvailableProducts] = updatedProduct
        return {
          ...state,
          userProducts: updatedUserProducts,
          availableProducts: updatedAvailableProducts
        }

      default: return state
    }
  }

export default productsReducer

```

store/actions/orders.js

```

...
export const addOrder = (items, total) => {
  return async (dispatch, getState) => {
    const date = new Date()

    try {
      const { token, userId } = getState().auth

      const response = await fetch(
        `https://rnmcshoppingapp-default-firebase.firebaseio.com/orders/
        ${userId}.json?auth=${token}`,
        {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ items, total, date: date.toISOString() })
        }
      )

      if (!response.ok) {
        throw new Error("Error in `addOrder` @ actions/orders.js ")
      }

      const data = await response.json()

      dispatch({
        type: ADD_ORDER, payload: { id: data.name, items, total, date }
      })
    }
  }
}
```

```

// we trigger notifications after an order is placed for all items.
// > This is sub-optimal! The best solution is handling it from the
// server. We do not do it there as Firestore is not free.

for (const item in items) {
  // firebase has cart items stored in values of objects where
  // keys are indexes starting from 0. So items[0] is the first,
  // then item[1], and so on.

  const { pushToken, title } = items[item]

  fetch("https://exp.host/--/api/v2/push/send"),
  {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Accept-Encoding": "gzip, deflate",
      "Content-Type": "application/json"
    },
    body: JSON.stringify({
      to: pushToken,
      // data: item, // <- extra data, if required
      title: "Order was placed!",
      body: title
    })
  }
}

} catch (err) {
  throw new Error("Error in `addOrder` @ actions/orders.js.", err)
}
}

```

Adding notifications to shopping app (3/3)

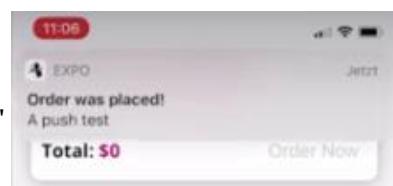
Notifications do not work, and that's simply because we did not set the proper handlers in App.js yet.

App.jsx

```

import * as Notifications from "expo-notifications"
...
Notifications.setNotificationHandler({
  handleNotification: async () => ({ shouldShowAlert: true })
}) ...

```



Push notifications in non-expo managed apps

You can still use Expo's powerful Push Notification services and features, even if you're NOT working in a managed Expo project.

Detailed setup instructions can be found here:

<https://github.com/expo/expo/tree/master/packages/expo-notifications#installation-in-bare-react-native-projects>

Once configured and set up, you can work with push notifications just as shown in this module.

Useful links

Expo Push Notification Guide:

<https://docs.expo.io/push-notifications/overview/>

Expo Push Notification API Docs:

<https://docs.expo.io/versions/latest/sdk/notifications/>

Next steps

Roundup

Practice, Build (Dummy) Apps!

Dive into Official Docs & Other Examples

Explore Native Modules & Third-Party Packages

Stay Updated with Github Repo & React Native +
Expo Blogs

Javascript and React refresher

Syntax

Weakly Typed Language

Object-Oriented Language

Versatile Language

No explicit type assignment

Data can be organized in logical objects

Runs in browser & directly on a PC/ server

Data types can be switched dynamically

Primitive and reference types

Can perform a broad variety of tasks