React Native

Part 3 – Example project, Recap & ContextProviders

Recap

- Components
- Props
- States
- Styling
- LifeCycle

Todays lecture

- Guide through the example app.
- Components from last time
- Make an HTTP call
- ContextProviders
- Exercise

Lists

To be able to loop through an array inside the render method you need to use the Carlsberg signs to tell react, that you want to use Native JavaScript.

```
const items = [
  {
    name: "hello",
  },
  {
    name: "hello again",
  },
  {
    name: "hello again and again",
  },
];

export default function ForLoops({ text }) {
  return (
    <View style={styles.container}>
      {items.forEach((item) => (
        <Text>{item.name}</Text>
      ))}
    </View>
  );
}
```

In this example we want to loop through all items in the ”items” array and display the name in a text element

Core componetns

- ActivityIndicator
- Button
- Image
- ScrollView
- Flatlist
- SectionList

ActivityIndicator

On an app, sometimes you want to display a loading indicator.
This could be when you are awaiting for a response from a webserver or something else, and want to notify the user, that data is being transported

```
import { ActivityIndicator, StyleSheet, Text, View } from "react-native";

export default function App() {
  return (
    <View style={styles.container}>
      <ActivityIndicator />
      <ActivityIndicator size="large" />
      <ActivityIndicator size="small" color="#0000ff" />
      <ActivityIndicator size="large" color="#00ff00" />
    </View>
  );
}
```

Button

When the user can perform some action by pressing something, a button is a great visual tool to display, that an user action is possible

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

export default function App() {
  return (
    <View style={styles.container}>
      <Button
        title="Press me"
        onPress={() => Alert.alert("Simple Button pressed")}
      />
    </View>
  );
}
```

Image

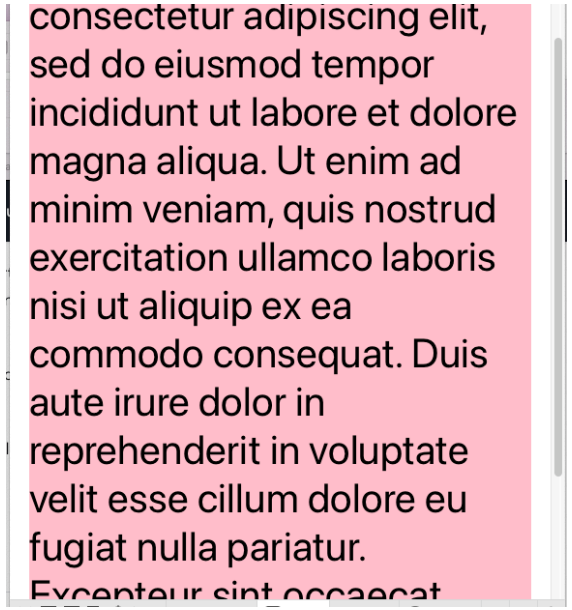
On an app, sometimes you want to display an image. This can be done by utilizing the Image component. Either download an image or use an locale address.

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

const DisplayAnImage = () => {
  return (
    <View style={styles.container}>
      <Image
        style={styles.tinyLogo}
        source={require("@expo/snack-static/react-native-logo.png")}
      />
      <Image
        style={styles.tinyLogo}
        source={{
          uri: "https://reactnative.dev/img/tiny_logo.png",
        }}
      />
    </View>
  );
};
```

ScrollView

Sometimes we can have a large section of information, that we rather would like contained inside a scrolling container. This is where you would is the ScrollView.

```
import {
  StyleSheet,
  Text,
  SafeAreaView,
  ScrollView,
  StatusBar,
} from "react-native";

const MySCrollView = () => {
  return (
    <SafeAreaView style={styles.container}>
      <ScrollView style={styles.scrollView}>
        <Text style={styles.text}>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
          eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
          minim veniam, quis nostrud exercitation ullamco laboris nisi ut
          aliquip ex ea commodo consequat. Duis aute irure dolor in
          reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
          pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
          culpa qui officia deserunt mollit anim id est laborum.
        </Text>
      </ScrollView>
    </SafeAreaView>
  );
};
```



FlatList

FlatList allows you to have a scrollable list, that only renders what is within the screen. Lets say you have 100 items, only the items within the screen will be rendered, and once you begin to scroll the other items begins to render.

```
const Item = ({ title }) => (
  <View style={styles.item}>
    <Text style={styles.title}>{title}</Text>
  </View>
);

const App = () => {
  const renderItem = ({ item }) => <Item title={item.title} />;

  return (
    <SafeAreaView style={styles.container}>
      <FlatList
        data={DATA}
        renderItem={renderItem}
        keyExtractor={(item) => item.id}
      />
    </SafeAreaView>
  );
};
```

```
const DATA = [
  {
    id: "bd7acbea-c1b1-46c2-aed5-3ad53abb28ba",
    title: "First Item",
  },
  {
    id: "3ac68afc-c605-48d3-a4f8-fbd91aa97f63",
    title: "Second Item",
  },
  {
    id: "58694a0f-3da1-471f-bd96-145571e29d72",
    title: "Third Item",
  },
];
```

SectionList

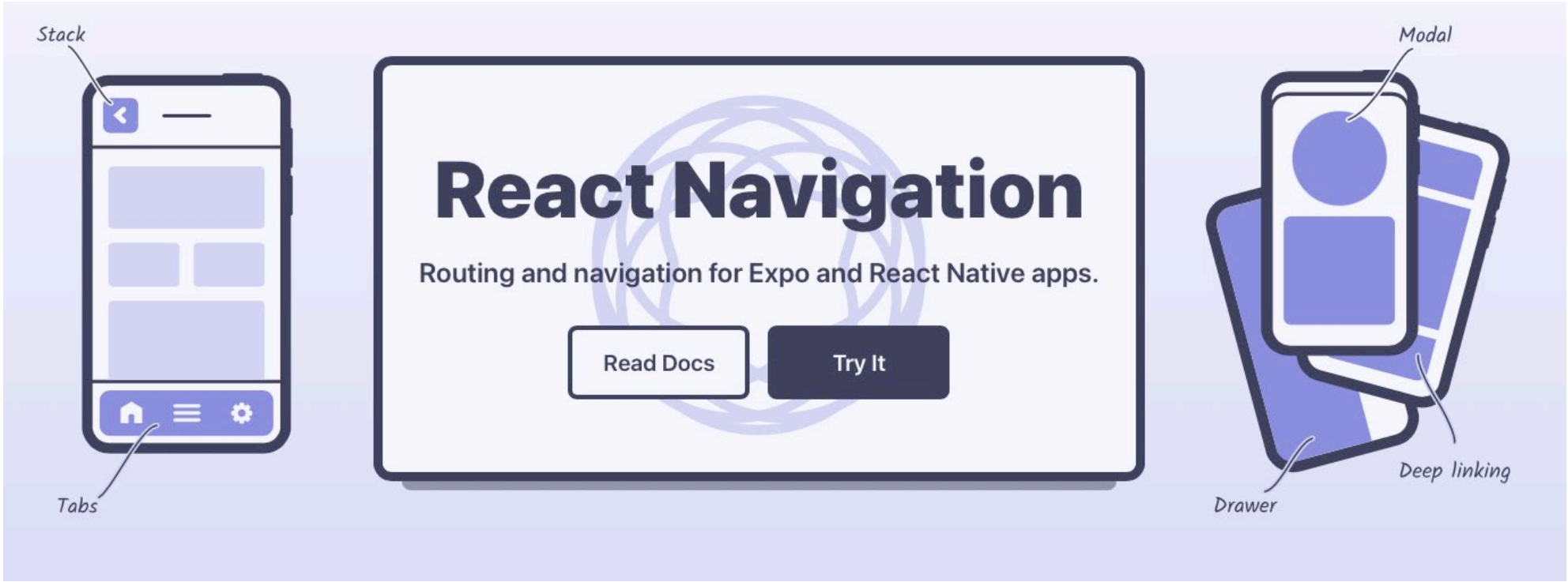
SectionList is very similar to FlatList, but have sections. If you dont need sections, you can just go with a flat list

```jsx
const Item = ({ title }) => (
  <View style={styles.item}>
    <Text style={styles.title}>{title}</Text>
  </View>
);

const App = () => (
  <SafeAreaView style={styles.container}>
    <SectionList
      sections={DATA}
      keyExtractor={(item, index) => item + index}
      renderItem={({ item }) => <Item title={item} />}
      renderSectionHeader={({ section: { title } }) => (
        <Text style={styles.header}>{title}</Text>
      )}
    />
  </SafeAreaView>
);
```

```jsx
const DATA = [
  {
    title: "Main dishes",
    data: ["Pizza", "Burger", "Risotto"],
  },
  {
    title: "Sides",
    data: ["French Fries", "Onion Rings", "Fried Shrimps"],
  },
  {
    title: "Drinks",
    data: ["Water", "Coke", "Beer"],
  },
  {
    title: "Desserts",
    data: ["Cheese Cake", "Ice Cream"],
  },
];
```

Navigation

A lot of apps have multiple pages, that you can navigate through. There is no core component for this, so therefor we have to fine a 3rd party library. Luckily expo itself have developed a nice library for us to use.



Install React Navigation

To utilize react navigation, go to your root directory and install the following libraries:

npm Yarn

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

```
expo install react-native-screens react-native-safe-area-context
```

```
npm install react-native-screens react-native-safe-area-context
```

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

Using the navigation components part 1

Import the components, and use them as shown:

```
import { NavigationContainer } from "@react-navigation/native";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
```

```
const Stack = createNativeStackNavigator();

export default function MyNavigation() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="SecondScreen" component={SecondScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Using the navigation components part 2

Import the components, and use them as shown:

```
function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: "center", justifyContent: "center" }}>
      <Text onPress={() => navigation.navigate("SecondScreen")}>
        Home screen
      </Text>
    </View>
  );
}

function SecondScreen() {
  return (
    <View style={{ flex: 1, alignItems: "center", justifyContent: "center" }}>
      <Text>Second Screen</Text>
    </View>
  );
}
```

Fetch data from API

```
// A state holding all the movie data.
const [data, setData] = useState([]);

// Fetch movie list when component is mounted
useEffect(() => {
  fetchMovies();
}, []);

// Function to fetch movie list
function fetchMovies() {
  fetch(
    `https://api.themoviedb.org/3/movie/popular?api_key=${process.env.API_KEY}&language=en-US&page=1`
  )
    .then((response) => response.json())
    .then((data) => setData(data.results));
}
```

15 - 20 min Exercise: Make an HTTP call.

1. Create a function that fetches data from an API
2. You can use this free API to fetch JSON data:
3. https://jsonplaceholder.typicode.com/todos/

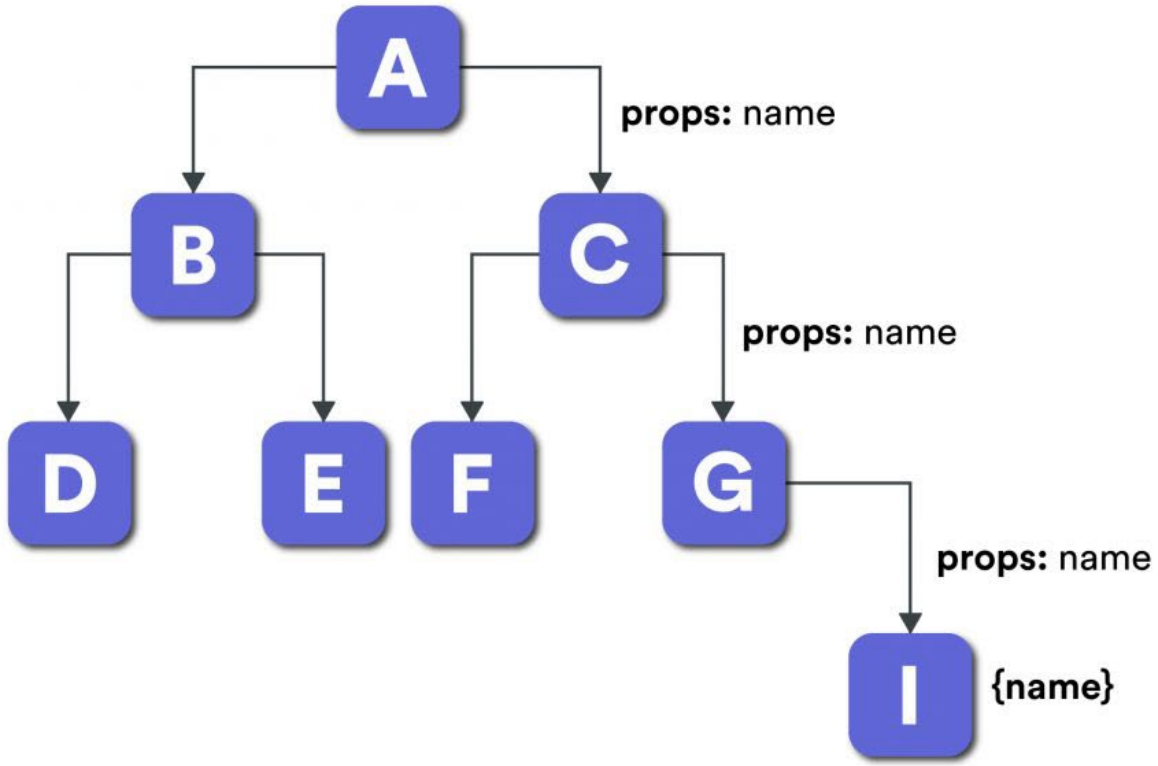
ContextProviders

Context provides a method to transport data across the component tree without having to send props down manually at every level," according to the React documentation.

The required hierarchy of sending props for each component in its component tree is therefore avoided thanks to Context API.

ContextProviders

The term "props-drilling" may be familiar to you. You can be in the process of drilling through numerous layers of components while you develop your application.



ContextProvider use cases

- Theming
- User authentication
- Removal of Prop-drilling issue

Implement Context Provider part 1

Create a MovieContext.js and implement the Following code. You can modify this to contain many more states.

```
import React, { Component, useEffect, useState } from "react";

const MovieContext = React.createContext({});

const MovieProvider = ({ children }) => {
  const [movies, setMovies] = useState([]);

  return (
    <MovieContext.Provider
      value={{
        movies,
        setMovies,
      }}
    >
      {children}
    </MovieContext.Provider>
  );
};

export default MovieContext;

export { MovieProvider };
```

Implement Context Provider part 2

Add the MovieProvider to App.js.
The MovieProvider should be added to App.js and now within all the child components, the useStates can now be called.

```
export default function App() {
  return (
    <NavigationContainer>
      <MovieProvider>
        <Stack.Navigator>
          <Stack.Screen name="Home" component={HomeScreen} />
          <Stack.Screen name="Details" component={DetailsScreen} />
        </Stack.Navigator>
      </MovieProvider>
    </NavigationContainer>
  );
}
```

Implement Context Provider part 3

Call the useContext, from the React library.
Using JavaScript object destruction, you can get that variables that you need.

```
const { setMovies, movies } = useContext(MovieContext);
```

25 - 30 min Exercise: Build your own ContextProvier

1. Create your own custom ContextProvider
2. Try to play around with adding multiple states and get familier with ContextProviders.