**Additional resources**

As we are getting closer to the end of this part, let's take a moment to look at some additional React Native related resources. [Awesome React Native](https://github.com/jondot/awesome-react-native) is an extremely encompassing curated list of React Native resources such as libraries, tutorials, and articles. Because the list is exhaustively long, let's have a closer look at few of its highlights

**React Native Paper**

Paper is a collection of customizable and production-ready components for React Native, following Google’s Material Design guidelines.

[React Native Paper](https://callstack.github.io/react-native-paper/) is for React Native what [Material-UI](https://material-ui.com/) is for React web applications. It offers a wide range of high-quality UI components, support for [custom themes](https://callstack.github.io/react-native-paper/theming.html) and a fairly simple [setup](https://callstack.github.io/react-native-paper/getting-started.html) for Expo based React Native applications.

**Styled-components**

Utilising tagged template literals (a recent addition to JavaScript) and the power of CSS, styled-components allows you to write actual CSS code to style your components. It also removes the mapping between components and styles – using components as a low-level styling construct could not be easier!

[Styled-components](https://styled-components.com/) is a library for styling React components using [CSS-in-JS](https://en.wikipedia.org/wiki/CSS-in-JS) technique. In React Native we are already used to defining component's styles as a JavaScript object, so CSS-in-JS is not so uncharted territory. However, the approach of styled-components is quite different from using the StyleSheet.create method and the style prop.

In styled-components components' styles are defined with the component using a feature called [tagged template literal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals#Tagged_templates) or a plain JavaScript object. Styled-components makes it possible to define new style properties for component based on its props at runtime. This brings many possibilities, such as seamlessly switching between a light and a dark theme. It also has a full [theming support](https://styled-components.com/docs/advanced#theming). Here is an example of creating a Text component with style variations based on props:

import styled from 'styled-components/native';

import { css } from 'styled-components';

const FancyText = styled.Text`

color: grey;

font-size: 14px;

${({ isBlue }) =>

isBlue &&

css`

color: blue;

`}

${({ isBig }) =>

isBig &&

css`

font-size: 24px;

font-weight: 700;

`}

`;

const Main = () => {

return (

<>

<FancyText>Simple text</FancyText>

<FancyText isBlue>Blue text</FancyText>

<FancyText isBig>Big text</FancyText>

<FancyText isBig isBlue>

Big blue text

</FancyText>

</>

);

};

Because styled-components processes the style definitions, it is possible to use CSS-like snake case syntax with the property names and units in property values. However, units don't have any effect because property values are internally unitless. For more information on styled-components, head out to the [documentation](https://styled-components.com/docs).

**React-spring**

react-spring is a spring-physics based animation library that should cover most of your UI related animation needs. It gives you tools flexible enough to confidently cast your ideas into moving interfaces.

[React-spring](https://www.react-spring.io/) is a library that provides a clean [API](https://react-spring.io/basics) for animating React Native components.

**React Navigation**

Routing and navigation for your React Native apps

[React Navigation](https://reactnavigation.org/) is a routing library for React Native. It shares some similarities with the React Router library we have been using during this and earlier parts. However, unlike React Router, React Navigation offers more native features such as native gestures and animations to transition between views.

**Closing words**

That's it, our application is ready. Good job! We have learned many new concepts during our journey such as setting up our React Native application using Expo, using React Native's core components and adding style to them, communicating with the server, and testing React Native applications. The final piece of the puzzle would be to deploy the application to the Apple App Store and Google Play Store.

Deploying the application is entirely *optional* and it isn't quite trivial, because you also need to fork and deploy the [rate-repository-api](https://github.com/fullstack-hy2020/rate-repository-api). For the React Native application itself, you first need to create either iOS or Android builds by following Expo's [documentation](https://docs.expo.io/distribution/building-standalone-apps/). Then you can upload these builds to either Apple App Store or Google Play Store. Expo has [documentation](https://docs.expo.io/distribution/uploading-apps/) for this as well.