

React Philosophy



What we did:

- Compared HTML (declarative language) and Javascript (Imperative language).
- Introduction to React philosophy.
- Created a simple React Native component for their React Native app in expo.

How we did it:

Engineers at Facebook, while working on designing the different complex Facebook apps, realized that the declarative style of coding would make their app organization easy.

They developed a new programming language called JSX and a new mobile development framework called React Native to help build powerful and complex apps in an easy way.

JSX combines both HTML and Javascript style of coding. JSX is declarative - we tell what are the different components we want in the application.

We wrote JSX and React Native code in an online editor called 'snack'.

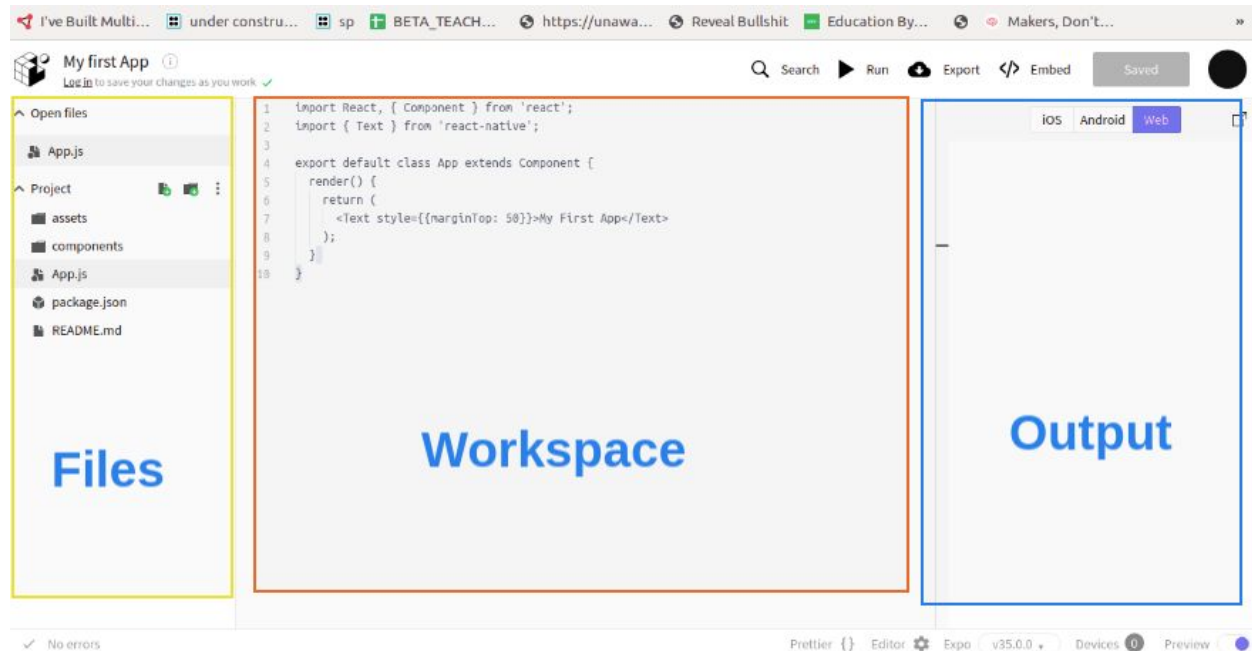
'Snack' is an online editor written by Expo, which allows us to write and compile React Native code in an online environment. We can also preview the output of the code using Snack.

On the left side, you see all the files in the project. We will look into these files later.

At the centre, you see the workspace where we will write our code.

On the right side, you can see the output of your code.

You can see the output for either Android, iOS or Web.

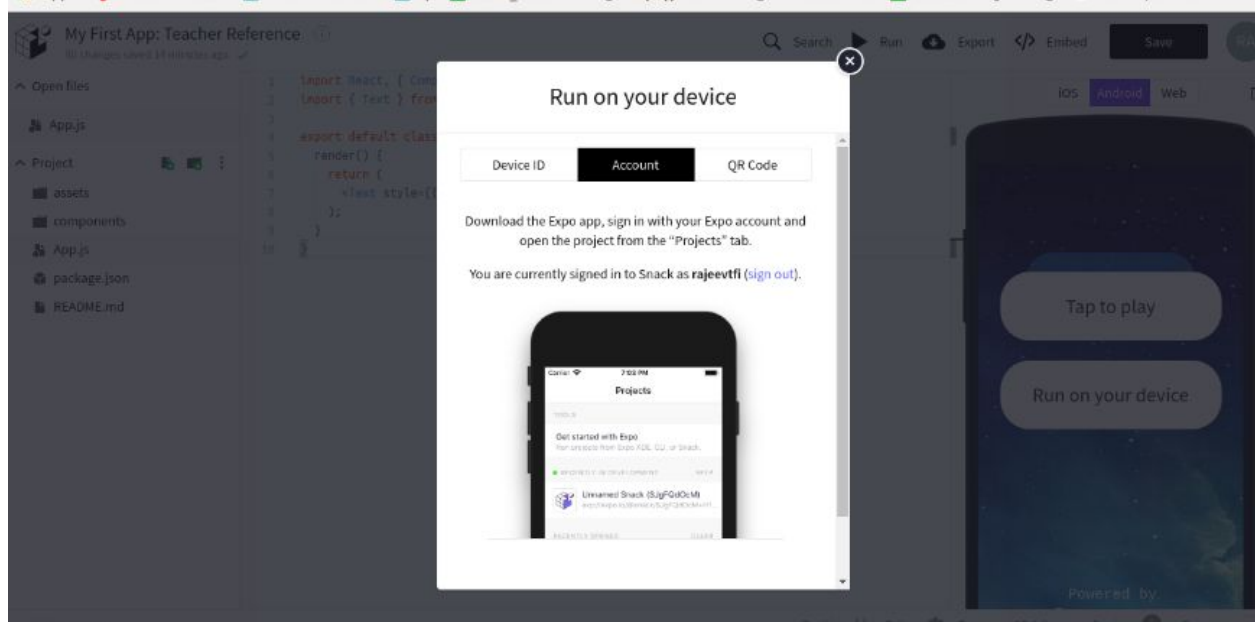
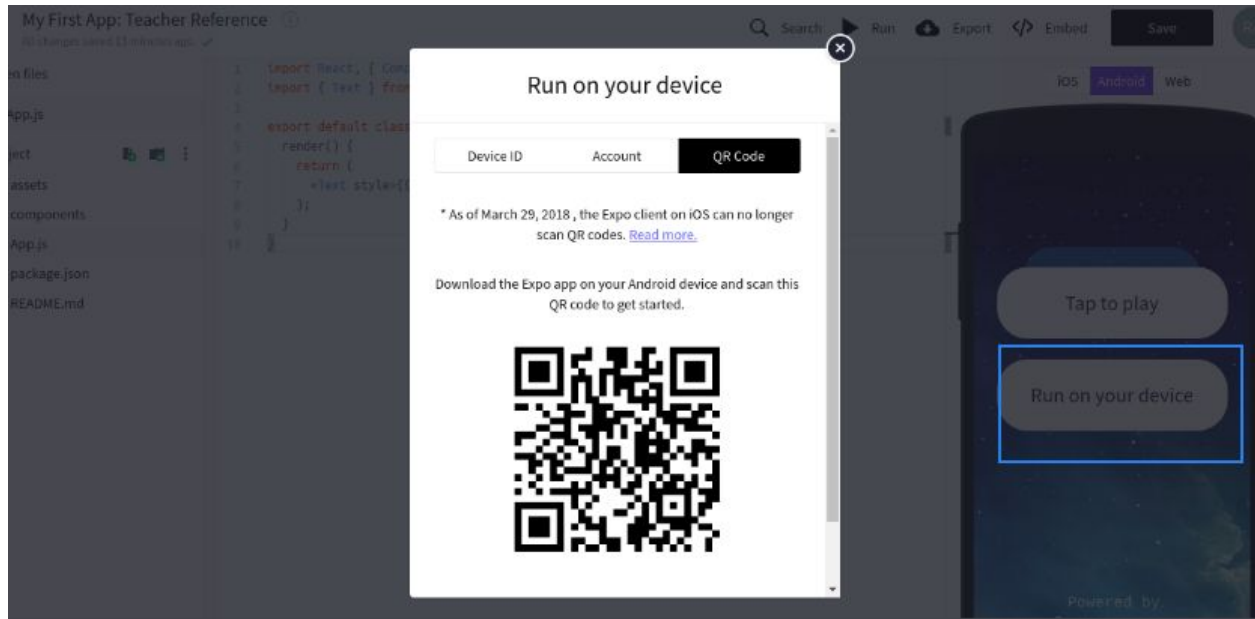


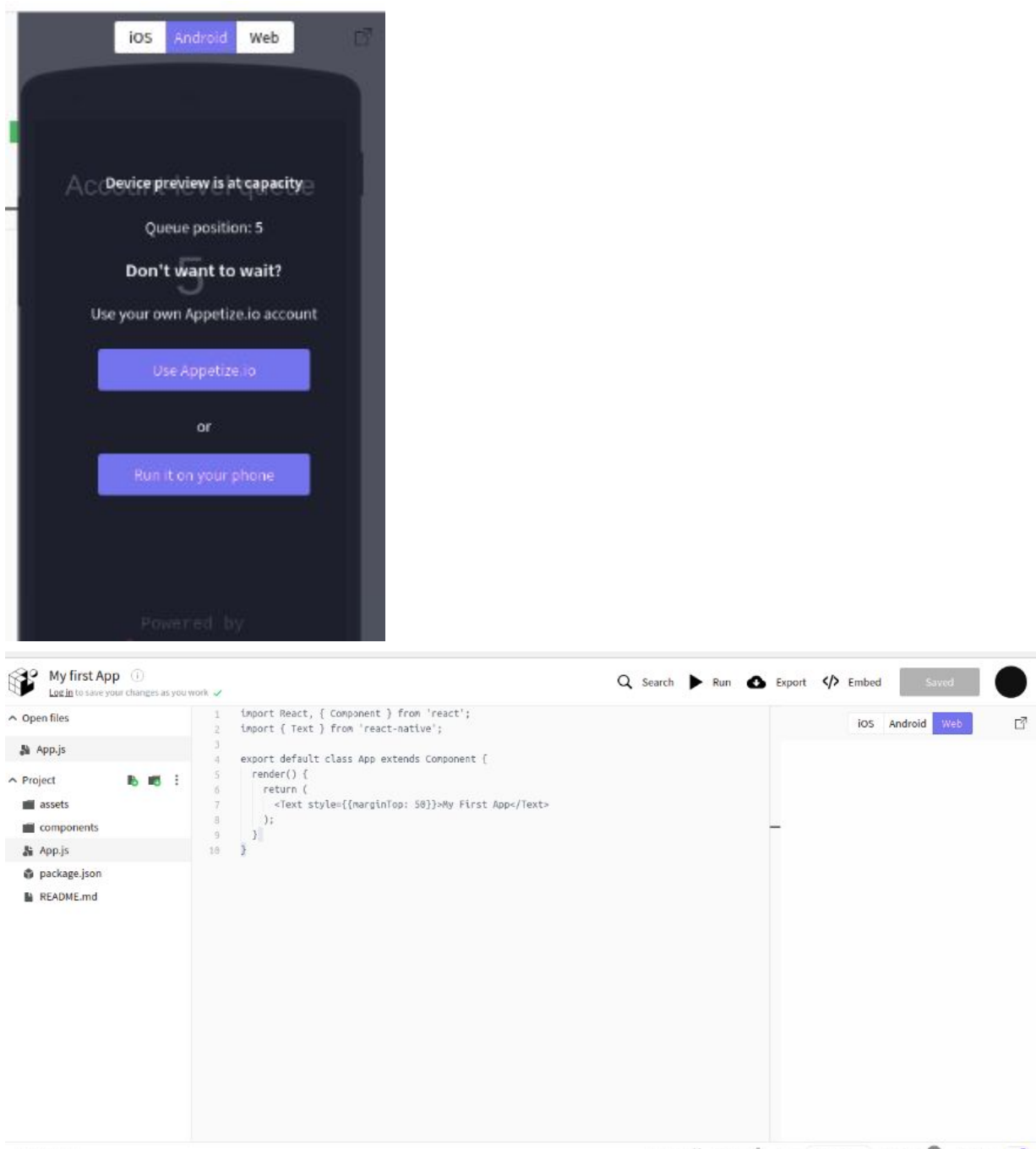
For Android/iOS, you can see the output either on your phone or on the android emulator in the browser. (By clicking Run on your device.)

For Android/iOS, you can install Expo Client, sign in with the same account and open the project from under Projects Tab.

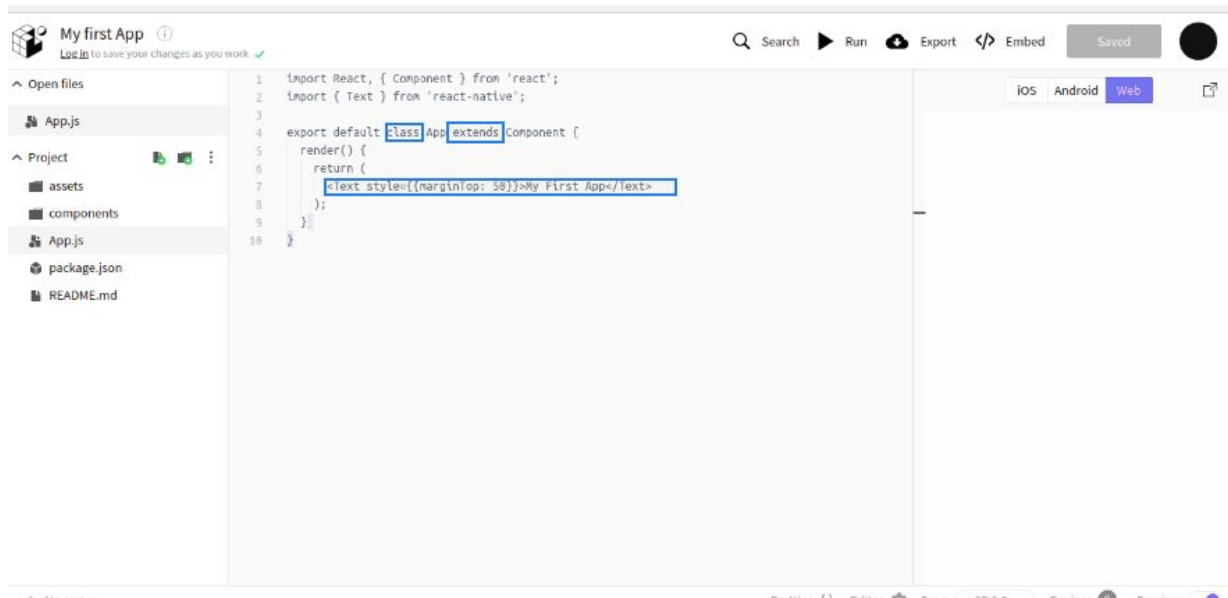
For Android, you can also scan the QR Code.
(Encourage the student to scan the QR code from their Expo App and check the output.)

For seeing the output on the device emulator in the browser, you have to press "Tap to play". You might have to wait in a queue as there are many systems online using a common emulator.

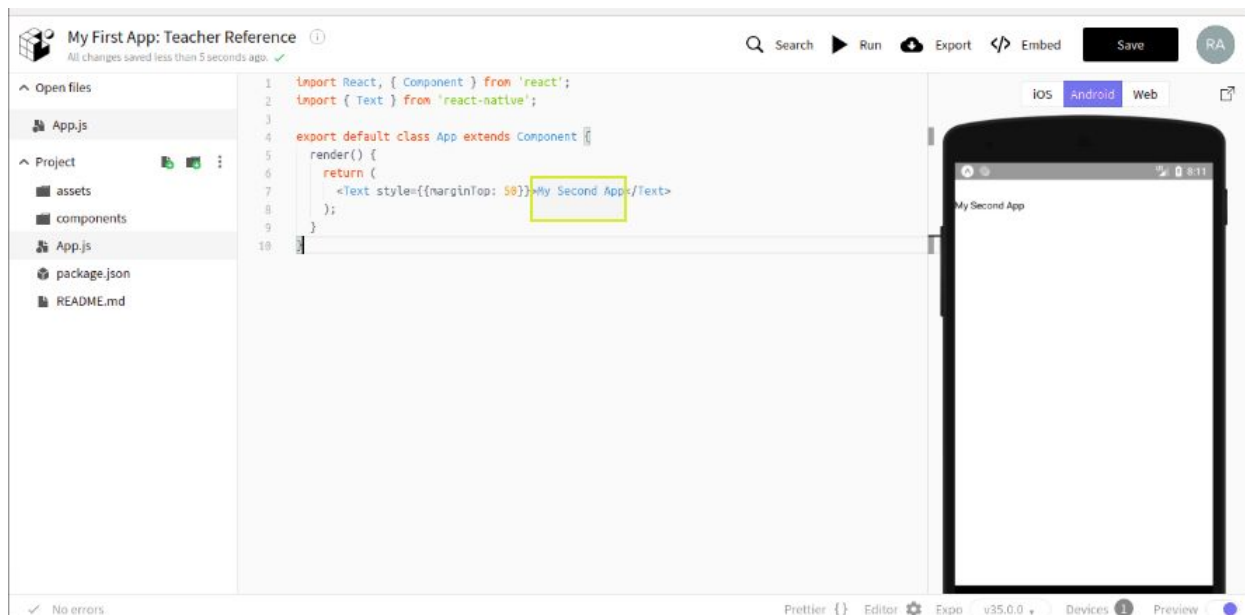




Code:



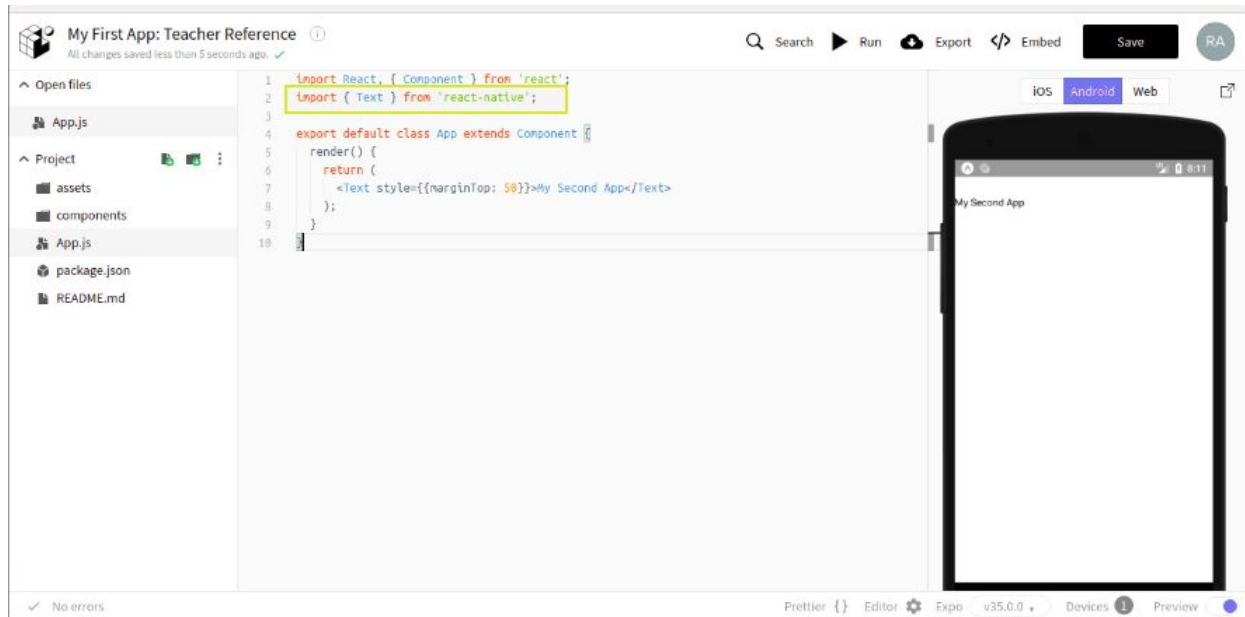
`<Text></Text>` tag which you see is such a React component. This component allows you to write text on the screen.



Text is a component which is already defined in a library called React Native.

In line 2, you can see how we are importing the component Text from React Native Library.

Note: When compared to HTML tags which start with lowercase letters, React native components start with uppercase.

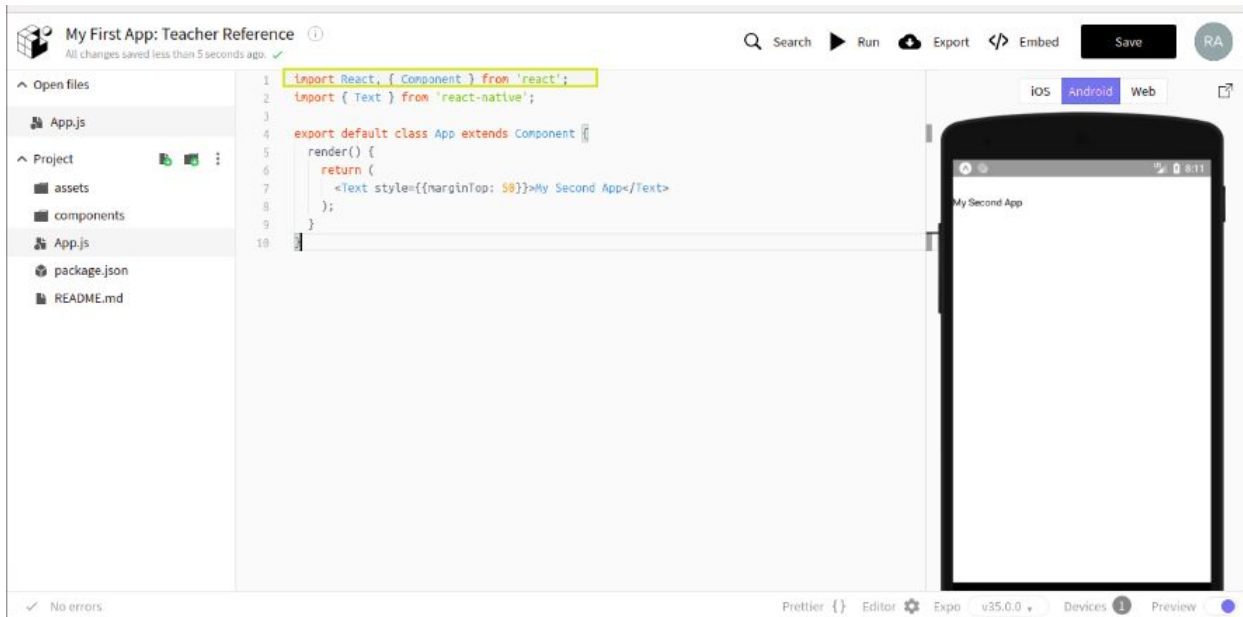


A Component is a Base class (Parent class) from which all components are inherited from.

Remember, Baseclass in Angry Birds? and how each of the classes - Bird, Pig, Box - extended the Base Class and inherited all its properties and functions?

A Component is a baseclass and all React components are inherited from this baseclass Component.

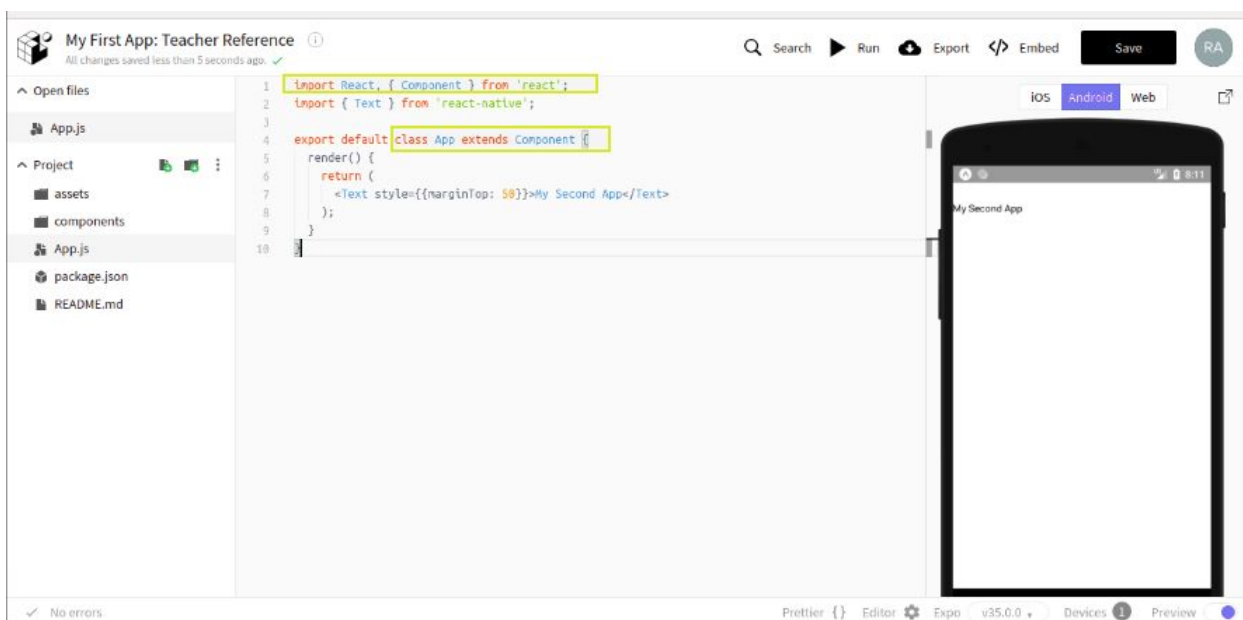
Component class is defined in the React library. You can see how React and Component library is imported from the React library in line 1.



In line 4 you can see how a new Class called App extends the Component class. (Ignore export default for now. We will come back to it sometime later.)

App itself is a React Native Component. All React Native components must live inside the App Component.

Things will become clearer soon when we practically start making an App.



Now you can see that there is only one defined function inside the App Class - it is called `render()`

'`render()`' function simply displays whatever components are returned by it.

Here, you can see that the `<Text>` component is returned. So, the `<Text>` component gets rendered or displayed on the screen.

Notice that all the code in the file is Javascript except the code inside `render`. This is JSX. It contains tags which correspond to React Components.

We can write Javascript inside JSX using `{ }` 'curly brackets'

You can also see that the `Text` component has a `style` 'property' defined on it. Just like html tags have some properties defined on them
(For example: `` tag has `src`, `width`, `height` etc.)

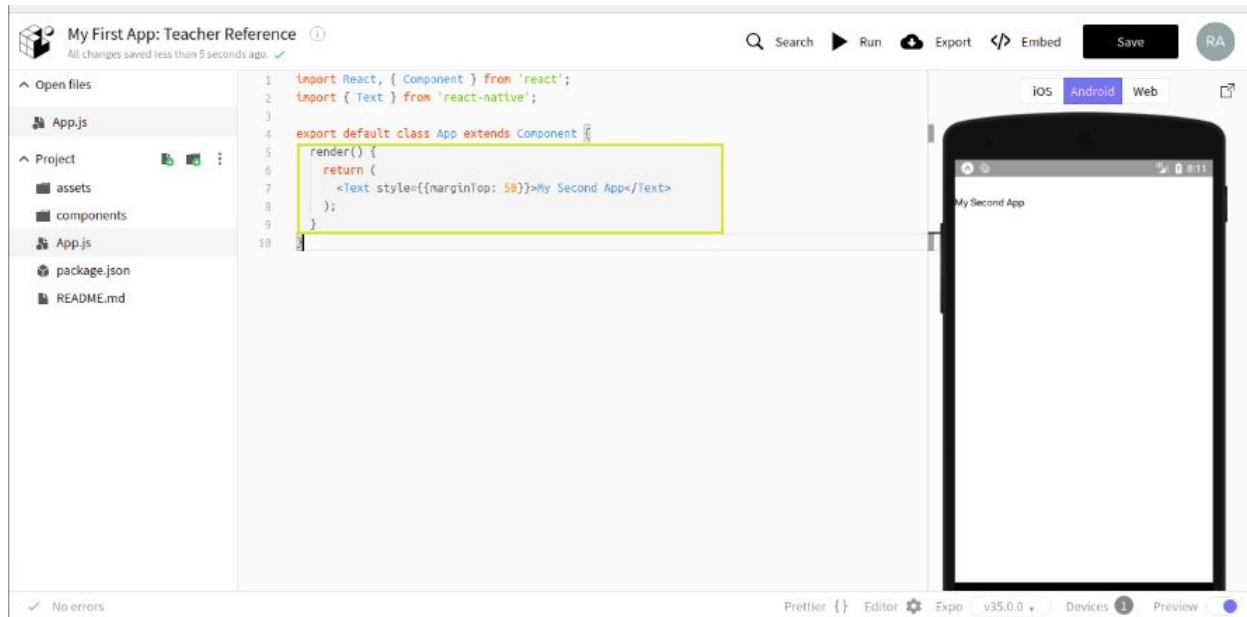
"`style`" property takes a json object `{}`.

When we are rendering components using JSX tags (`<Text>`), we can write/execute Javascript inside the `{}`.

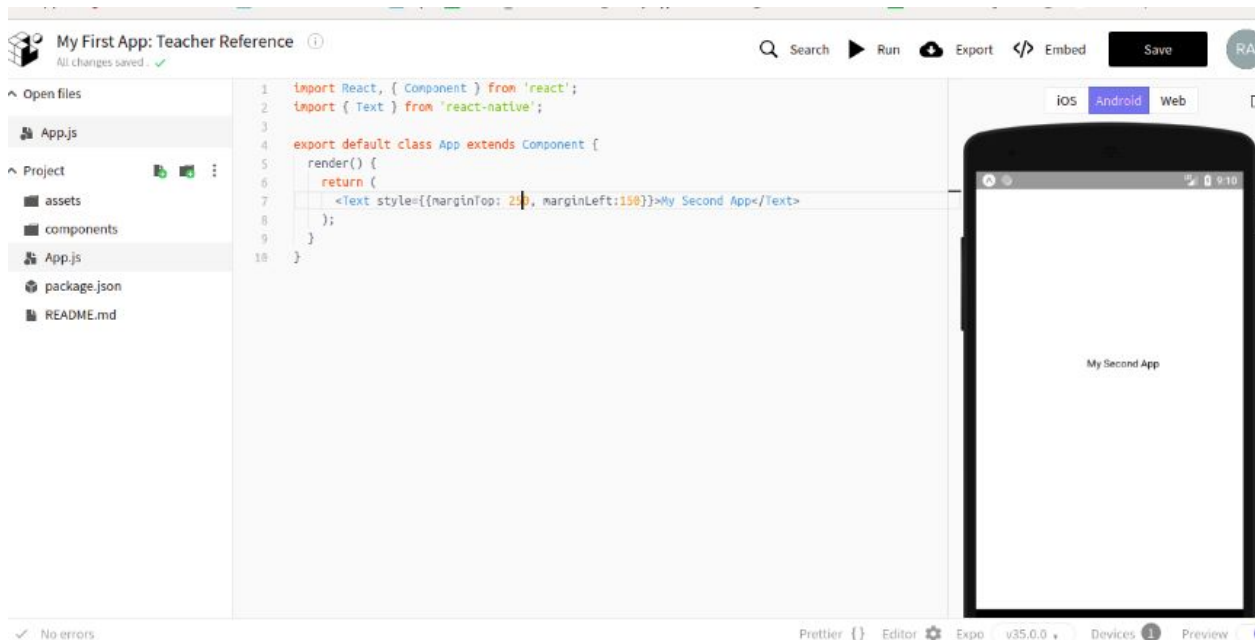
You see two `{{ }}` because one `{}` says that we are going to write javascript code. The other `{}` is for the json object.

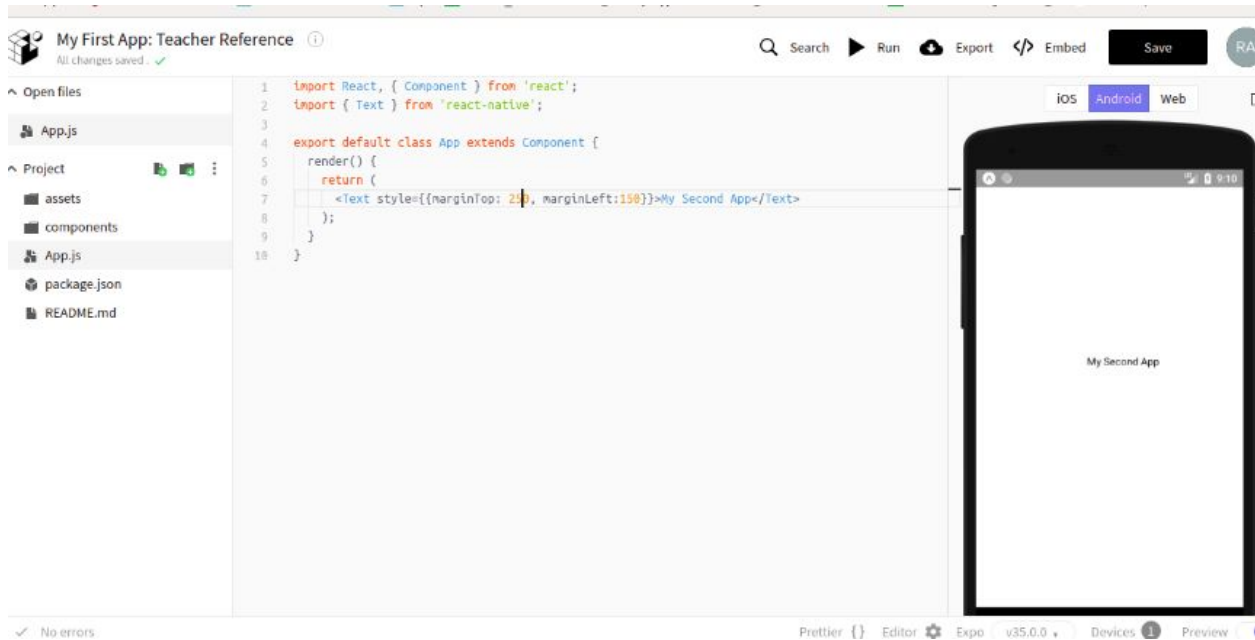
Also, we write CSS properties in React Native in camel case (`marginTop` instead of `margin-top`)

Again, things will become much clearer when we practically start doing things. But before we do that can you quickly recap what you can understand from the code?



Place the text somewhere in the centre.





The screenshot displays the WhiteHat Jr coding interface. On the left, a file explorer shows the project structure: 'App.js' under 'Open files', and 'assets', 'components', 'App.js', 'package.json', and 'README.md' under 'Project'. The main editor displays the following code in 'App.js':

```
1 import React, { Component } from 'react';
2 import { Text } from 'react-native';
3
4 export default class App extends Component {
5   render() {
6     return (
7       <Text style={{marginTop: 20, marginLeft: 150}}>My Second App</Text>
8     );
9   }
10 }
```

On the right, a preview window shows the app running on an Android device. The device screen displays the text 'My Second App'. The top of the preview window has tabs for 'iOS', 'Android', and 'Web', with 'Android' selected. The bottom status bar of the device shows the time as 9:10. At the bottom of the interface, there is a status bar with 'No errors', 'Prettier', 'Editor', 'Expo', 'v35.0.0', 'Devices', and 'Preview'.

Create an Expo account and login.

Hey friend! We are co-hosting a conference with [Software Mansion](#), [learn more](#).

Expo Search Expo

The fastest way to build an app

With Expo tools, services, and React, you can build, deploy, and quickly iterate on native Android, iOS, and web apps from the same JavaScript codebase.

- ✓ Access to device capabilities like camera, location, notifications, sensors, haptics, and [much more](#), all with universal APIs.
- ✓ Build service gives you app-store ready binaries and handles certificates, no need for you to touch Xcode or Android Studio.

Create your account

Create an account to discuss, publish, and manage all of your projects.

E-mail

Username

Password

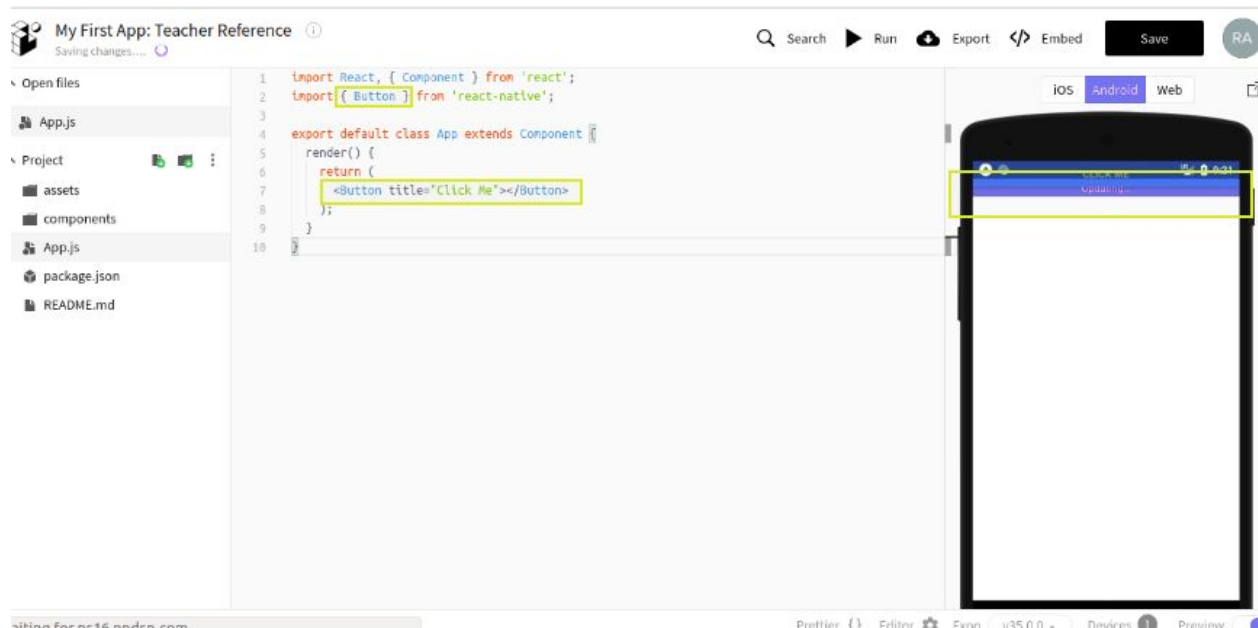
Confirm Password

☐ Enroll in [Expo Developer Services](#)

Let's import Button instead of Text from react native library.

We can use the `<Button></Button>` JSX tag to get a button component.

<Button> has a property called 'title' which can be used to display text inside the button.



You can see that there is a Button at the top of the app. It is half-hidden by the top notification bar of the phone.

Unfortunately, the button does not have a style property defined on it. Instead, we will use 'View Component' and place Button inside it.

We can then adjust the style of 'View Component'.

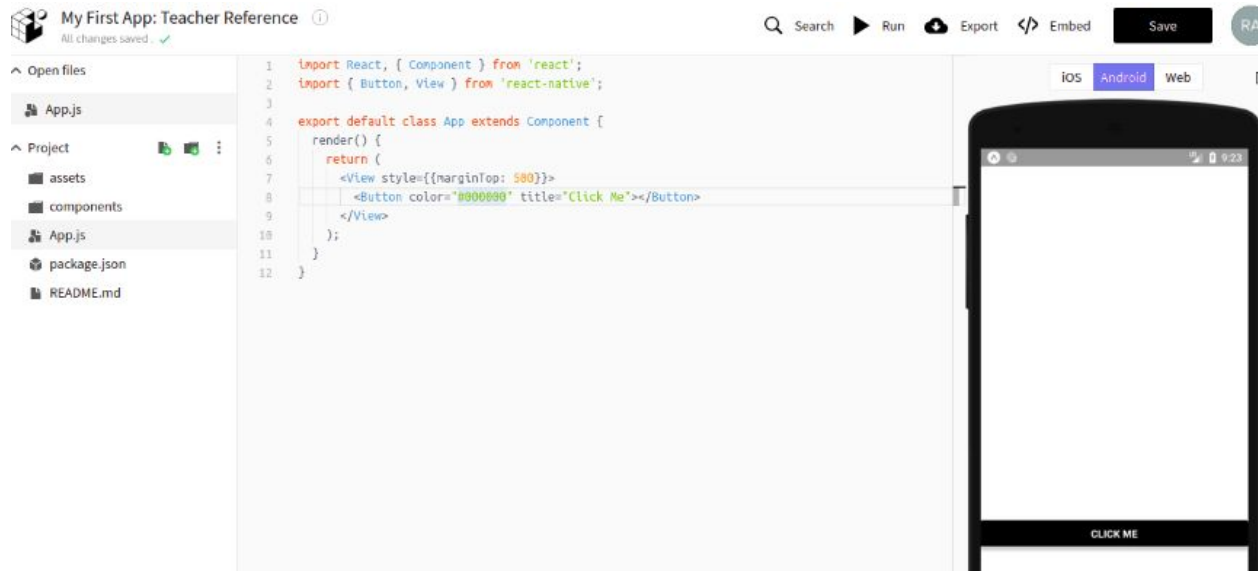
'View Component' is simply an empty container - just like 'div' in HTML. It has a style property defined on it. Let's import 'View Component'.

Let's place the button inside the 'View Component' and add style to it so that the button is at the bottom.



Button has another property called 'color' defined on it. You can use it to set a different color to the button. You can also use Hexadecimal numbers.





Create a simple React Native component called 'RedButton'. This React Native component will simply display a RedButton when rendered. We will need to create a class called RedButton which extends Component.



Write a render function with an empty return statement.

```

1  import React, { Component } from 'react';
2  import { Button, View } from 'react-native';
3
4  class RedButton extends Component{
5    render(){
6      return();
7    }
8  }
9
10
11 export default class App extends Component {
12   render() {
13     return (
14       <View style={{marginTop: 500}}>
15         <Button color="#000000" title="Click Me"></Button>
16       </View>
17     );
18   }
19 }

```

We want to render a Red Button. We can write that in the return statement.

Note: <Button/> is same as <Button></Button>

The former is a self-enclosing tag since there is nothing inside <Button></Button>

All tags in React Native must be closed.

```

1  import React, { Component } from 'react';
2  import { Button, View } from 'react-native';
3
4  class RedButton extends Component {
5    render() {
6      return <Button color="red" title="Click Me"/>;
7    }
8  }
9
10 export default class App extends Component {
11   render() {
12     return (
13       <View style={{ marginTop: 500 }}>
14
15       </View>
16     );
17   }
18 }
19

```



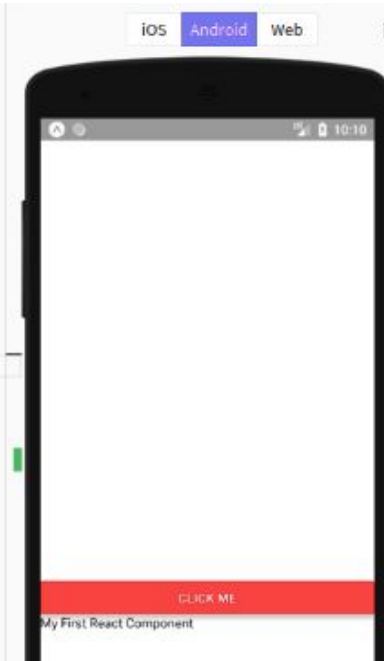
The screenshot shows a mobile application interface. At the top, there are three tabs: 'iOS', 'Android' (which is selected), and 'Web'. Below the tabs is a black header bar. The main content area is white and contains a single button with a red border and the text 'Click Me' in red. The button is centered on the screen.

Create the component by using the <RedButton/> inside the render function in App class.

```

1  import React, { Component } from 'react';
2  import { Button, View, Text } from 'react-native';
3
4  class RedButton extends Component {
5    render() {
6      return <Button color="red" title="Click Me"/>;
7    }
8  }
9
10 export default class App extends Component {
11   render() {
12     return (
13       <View style={{ marginTop: 500 }}>
14         <RedButton />
15         <Text>My First React Component</Text>
16       </View>
17     );
18   }
19 }
20

```




A render function can return only one React Component. If there are more than one React Components that need to be returned, they should be nested inside View.

```

1  import React, { Component } from 'react';
2  import { Button, View, Text } from 'react-native';
3
4  class RedButton extends Component {
5    render() {
6      return <Button color="red" title="Click Me"/>;
7    }
8  }
9
10 export default class App extends Component {
11   render() {
12     return (
13       <View style={{ marginTop: 500 }}>
14         <RedButton />
15         <Text>My First React Component</Text>
16       </View>
17     );
18   }
19 }
20

```



What's next?:

In the next class, you will learn how to add functions to a component. We will perform an action when a Button is pressed - like playing a sound.