
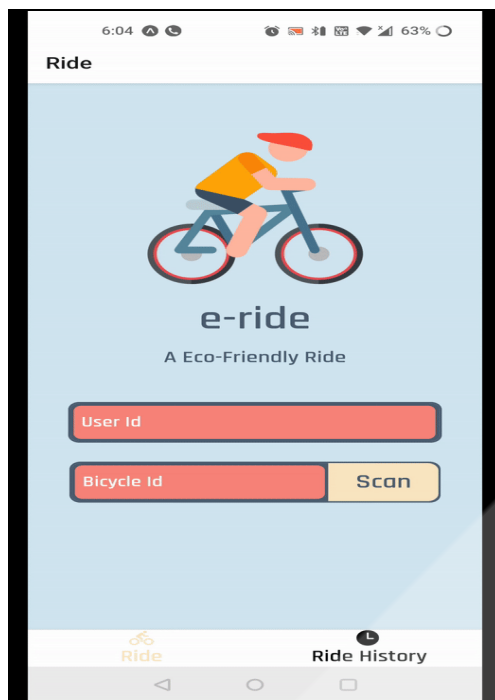


Topic	AUTOFILL TEXT INPUT	
Class Description	Students design the input form for the Transaction screen of the app. In addition, students write code to autofill the information when book id and student ids are scanned. They also learn to customize the appearance of tab navigation by adding icons to the tabs.	
Class	C70	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> • Customize the appearance of the bottom tabs by adding custom icons/images to it. • Design the input form for the issue/return screen of the app. • Write code to autofill the input box when book id and student id are scanned. 	
Resources Required	<ul style="list-style-type: none"> • Teacher Resources <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Android/iOS Smartphone with Expo App installed • Student Resources <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Android/iOS Smartphone with Expo App installed 	
Class structure	Warm-Up Teacher-led Activity Student-led Activity Wrap-Up	5 mins 15 mins 20 mins 5 mins
Credits	Ionicons is licensed under npm.js policies . QR Code generator provided by w69b GmbH,	

	Schwabstr. 41, 72108 Rottenburg, Germany	
Permissions	Expo permissions require the camera permission from the user to access their device's camera to scan the QR code.	
WARM-UP SESSION - 5 mins		
 <p>Teacher starts slideshow from slides 1 to 9</p> <p>Refer to speaker notes and follow the instructions on each slide.</p>		
Activity details		Solution/Guidelines
<p>Hi, how have you been? Are you excited to learn something new?</p> <p>Run the presentation from slide 1 to slide 3.</p> <p>The following are the warm-up session deliverables:</p> <ul style="list-style-type: none"> Reconnect with previous class topics. Warm-Up quiz session. 		<p>ESR: Varied Response.</p> <p>Click on the slide show tab and present the slides.</p>
QnA Session		
Question		Answer
<p>Choose the right block of code to render the TextInput component for bikeld.</p> <div data-bbox="256 1598 849 1885"> <pre> A. { <TextInput style={styles.textinput} placeholder={Bicycle Id} placeholderTextColor={"#FFFFFF"} value={bikeId} /> } </pre> </div>		C

<p>B. <pre>{/* <TextInput style={{styles.textinput}} placeholder="Bicycle Id" placeholderTextColor="#FFFFFF" value={bikeId} /> */}</pre></p> <p>C. <pre>{/* <TextInput style={styles.textinput} placeholder="Bicycle Id" placeholderTextColor="#FFFFFF" value={bikeId} /> */}</pre></p> <p>D. <pre>{/* <TextInput style={{styles.textinput}} placeholder={Bicycle Id} placeholderTextColor={#FFFFFF} value={bikeId} /> */}</pre></p>	
<p>Choose the right block of code to call the <code>getCameraPermissions()</code> when the button is clicked.</p>	<p>B</p>



- A.
B.
C.
D.

```
onPress={this.getCameraPermissions()}
onPress={() => this.getCameraPermissions()}
onPress={() => this.getCameraPermissions}
onPress={() => this.getCameraPermissions() }
```

Continue the warm-up session

Activity details

Run the presentation from slide 4 to slide 9 to set the problem statement.

The following are the warm-up session deliverables:

- **Preview the code from the previous class.**
- **Talk about customizing the appearance of bottom navigation.**

Solution/Guidelines

Narrate the story by using hand gestures and voice modulation methods to bring in more student interest.



Teacher ends slideshow

TEACHER-LED ACTIVITY - 15 mins

Teacher Initiates Screen Share

CHALLENGE

- Add icon/image to both the tabs in bottom navigation.

Teacher Action	Student Action
<p><i>Teacher opens the code from the previous class.</i></p> <p><i>Teacher can clone Teacher Activity 1 and install all the dependencies needed for the project.</i></p> <p>steps to clone the project:- <code>git clone <projectURL></code> <code>cd <projectFolder></code> <code>npm install</code></p>	
<p>Before we can go ahead and style our tabs, would you like to go through the code from the previous class and explain what is happening in our code so far?</p>	<p><i>Student goes through the code and explains:</i></p> <ul style="list-style-type: none"> • How tab navigation is created in the app. • Different states in the app and what they mean. • How the <code>handleCameraPermissions</code> is called when the button is pressed. • How the <code>handleBarcodeScanned</code> function gets the

	data and displays it inside a text.
<p>Awesome! It is always good to keep up with your code.</p> <p>Now, we want to customize the appearance of our tab navigation as well as add a custom font to add a special touch to the app.</p> <p>How to go about that?</p>	ESR: Varied.
<p>Whenever we get stuck at something, looking at the documentation is a good start.</p> <p>Let's first look at the expo-font and expo-google-fonts/rajdhani documentation.</p> <p>Using google fonts we can use any fonts that are available on Google. To use the font, write the name of the font after the ' '.</p> <p>example :- @expo-google-fonts/roboto</p> <p><i>Teacher opens the Teacher Activity 2 and Teacher Activity 3</i></p> <p>To install these fonts we'll use the commands</p> <ul style="list-style-type: none"> • expo install @expo-google-fonts/rajdhani • expo install expo-font@~8.4.0 <p>Now we'll import the above libraries in the App.js file.</p>	
importing libraries to App.js file	

```
import React, { Component } from "react";
import { Rajdhani_600SemiBold } from "@expo-google-fonts/rajdhani";
import * as Font from "expo-font";
```

Now let's create the function called **loadFonts()** which would be an async function and we'll call this function inside the **componentDidMount()** function of the App.js file and create a variable called fontLoaded inside the state. The initial value of this variable would be false.

Creating loadFont() and calling it in componentDidMount()

```
import React, { Component } from "react";
import { Rajdhani_600SemiBold } from "@expo-google-fonts/rajdhani";
import * as Font from "expo-font";

import BottomTabNavigator from "../components/BottomTabNavigator";

export default class App extends Component {
  constructor() {
    super();
    this.state = {
      fontLoaded: false
    };
  }

  async loadFonts() {}

  componentDidMount() {
    this.loadFonts();
  }
}
```

Now let's write the code for **loadFont()**.

In this function, we'll wait for the font to load and once it's loaded we'll set the fontLoaded state to true.

code for loadFont()

```

async loadFonts() {
  await Font.loadAsync({
    Rajdhani_600SemiBold: Rajdhani_600SemiBold
  });
  this.setState({ fontLoaded: true });
}

```

To make sure the fonts are loaded before rendering anything on the screen, let's write one condition to check that.

loading fonts before rendering the BottomTabNavigator

```

render() {
  const { fontLoaded } = this.state;
  if (fontLoaded) {
    return <BottomTabNavigator />;
  }
  return null;
}

```

Awesome, we are set with the fonts. Now, let's begin customizing the tab navigator.

Let's start by looking at the documentation for Tab Navigation and see if we can find something there.

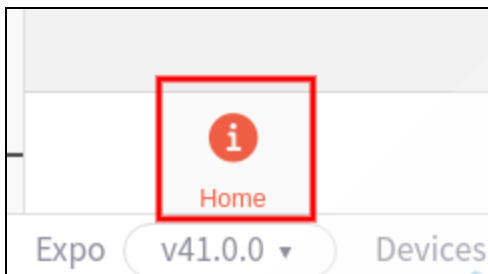
Teacher opens [Teacher Activity 4](#)

<p><i>Teacher opens the code snippet mentioned in the documentation.</i></p> <p>You can see that we have something here which talks about customizing the appearance of tab navigation.</p> <p>The code might look complex but there is also a code snippet which we can run on snack. Let's click on this and try to understand the code by experimenting with it.</p> <p><i>Refer to the “Customizing the appearance” part of the document.</i></p> <p><i>Click on the “Try this example on Snack” to open the code snippet.</i></p>	<p><i>Student opens Student Activity 1 linked to Tab Navigation documentation.</i></p> <p><i>The Student opens the code snippet mentioned in the documentation.</i></p>
<p>What do you observe in the code?</p>	<p><i>Student gives his/her own interpretation of the code.</i></p>
<p>You can see that in this code, the BottomTabNavigator is created directly inside the App.js file. We had created the BottomTabNavigator separately and then inserted it in the App.js file.</p> <p>Apart from this, you can also see that there are two additional options passed down in createBottomTabNavigator - screenOptions and tabBarOptions.</p> <p>tabBarOptions take the colors which you want in the bottom tabs to appear when they are active or inactive.</p> <p>Let's experiment with this by changing the colors.</p> <p><i>Teacher changes the colors in tabBarOptions and shows the output.</i></p>	<p><i>Students can also experiment with tabBarOptions inside snack.</i></p>

code from the snack to check the tabBarOptions.

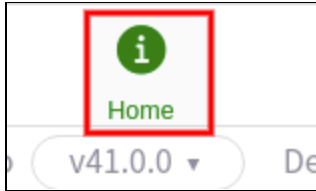
```
tabBarOptions={{
  activeTintColor: 'tomato',
  inactiveTintColor: 'gray',
}}
>
<Tab.Screen name="Home" component={HomeScreen} />
<Tab.Screen name="Settings" component={SettingsScreen} />
</Tab.Navigator>
</NavigationContainer>
);
```

OUTPUT:



```
tabBarOptions={{
  activeTintColor: 'green',
  inactiveTintColor: 'gray',
}}
>
<Tab.Screen name="Home" component={HomeScreen} />
<Tab.Screen name="Settings" component={SettingsScreen} />
</Tab.Navigator>
</NavigationContainer>
);
```

OUTPUT



screenOptions field returns some navigation options which are used in the app.

One of the navigation options is tabBarIcon.

tabBarIcon returns components which you want to use as an icon in your bottom tab navigation.

Let's experiment with this in our own code to understand better. we'll make these changes In our **BottomTabNavigator** file.

The student listens and asks questions.

create a const for bottom tab navigator

```
const Tab = createBottomTabNavigator();
```

create class BottomTabNavigator

```
export default class BottomTabNavigator extends Component {
  render() {
    return (
      <NavigationContainer>
        <Tab.Navigator
          screenOptions={({ route }) => ({
            tabBarIcon: ({ }) => {
              }
            }
          )>
          <Tab.Screen name="Transaction" component={TransactionScreen} />
          <Tab.Screen name="Search" component={SearchScreen} />
        </Tab.Navigator>
      </NavigationContainer>
    );
  }
}
```

Teacher opens the `BottomTabNavigator.js` file from the components folder.

Teacher imports all the required libraries.

Create a variable called `Tab`.

Inside this variable call the `createBottomTabNavigator()` function.

Inside the `Tab.Navigator` we'll use a predefined function called **screenOptions**.

This function takes the route object as default parameter.

The student looks at the code and asks questions.

Using the `screenOptions()` function

```
export default class BottomTabNavigator extends Component {
  render() {
    return (
      <NavigationContainer>
        <Tab.Navigator
          screenOptions={({ route }) => ({
            tabBarIcon: ({ }) => {
              // ...
            }
          })
        >
        <Tab.Screen name="Transaction" component={TransactionScreen} />
        <Tab.Screen name="Search" component={SearchScreen} />
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

Now, let's set the `tabBarIcon` field inside `screenOptions`. `tabBarIcon` should return a component (depending on the tab which is selected).

Teacher writes code for `tabBarIcon` which returns an empty object for now.

The student understands the code and asks questions to clarify.

<p>Note that the arguments for the function which will return the icons are kept empty for now since we don't plan to use focused, horizontal or tintColor in our code.</p> <p>All these objects hold the values of various properties:</p> <ul style="list-style-type: none"> ● focused: Returns true when the tab is tapped. Returns false otherwise. ● horizontal: Returns true when the device is in landscape mode. Returns false if the device is in portrait mode. ● tintColor: Returns the active set tintColor. <p>For more reference, please refer to Teacher Activity 5.</p>	
<p>Navigation object, which is passed down during function call, contains route.name property inside its state.</p> <p>route.name property changes depending on which screen is active in the app.</p> <ul style="list-style-type: none"> ● If the Transaction Screen is active, route.name equals "Transaction". ● If Search Screen is active, route.name equals "Search". <p>We need to return different components depending on which tab is active.</p> <p><i>Teacher writes code to return an empty Icon when each tab is active. Teacher also has to import the Ionicons component from react-native-icons/ionicons.</i></p> <p>Document for react-native-icons.</p>	<p><i>The student looks through the code and asks questions.</i></p>
<p>Code to return empty icon when the tab is active</p>	

```
export default class BottomTabNavigator extends Component {
  render() {
    return (
      <NavigationContainer>
        <Tab.Navigator
          screenOptions={({ route }) => ({
            tabBarIcon: ({ focused, color, size }) => {

              if (route.name === "Transaction") {

              } else if (route.name === "Search") {

              }

            }
          })
        >
      </Tab.Navigator>
    )
  }
}
```

Now the only thing that we need to do is to install the react-native-vector-icons/lonicons library using the:

npm install react-native vector-icons

We can use any icon which represents Transaction and Search.

Do you know any properties of the icons which can be used here?

ESR:

Name, size and color.

Teacher writes code to import the lonicons form react-native-vector-icons/lonicons and uses the book and search icons for transaction and search screen.

In the tabBarOptions sets the active and inactive colors along with the styles for the label.

The student observes the code written.

code to use ionicons

```
import React, { Component } from "react";
import { View } from "react-native";
import { NavigationContainer } from "@react-navigation/native";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
import Ionicons from "react-native-vector-icons/ionicons";

import TransactionScreen from "../screens/Transaction";
import SearchScreen from "../screens/Search";

const Tab = createBottomTabNavigator();

export default class BottomTabNavigator extends Component {
  render() {
    return (
      <NavigationContainer>
        <Tab.Navigator>
          screenOptions={({ route }) => ({
            tabBarIcon: ({ focused, color, size }) => {
              let iconName;

              if (route.name === "Transaction") {
                iconName = "book";
              } else if (route.name === "Search") {
                iconName = "search";
              }

              // You can return any component that you like here!
            }

            return (
              <Ionicons
                name={iconName}
                size={size}
                color={color}
              />
            );
          })
        </Tab.Navigator>
      </NavigationContainer>
    );
  }
}
```

```
// You can return any component that you like here!
```

```
    return (  
      <Icons  
        name={iconName}  
        size={size}  
        color={color}  
      />  
    );  
  }  
}}  
tabBarOptions={{  
  activeTintColor: "#FFFFFF",  
  inactiveTintColor: "black",  
  style: {  
    height: 130,  
    borderTopWidth: 0,  
    backgroundColor: "#5653d4"  
  },  
  labelStyle: {  
    fontSize: 20,  
    fontFamily: "Rajdhani_600SemiBold"  
  },  
  labelPosition: "beside-icon",  
  tabStyle: {  
    marginTop: 25,  
    marginLeft: 10,  
    marginRight: 10,  
    borderRadius: 30,  
    borderWidth: 2,  
    alignItems: "center",  
    justifyContent: "center",  
    backgroundColor: "#5653d4"  
  }  
}}  
>
```


Let's quickly test our app and see if the tab icon appears as the images we chose.



Teacher tests the code on her phone.

The student observes the outputs.



Awesome! Our app has a better look now with the icons.

The student explains how defaultNavigationOptions,

Can we now quickly go over the code and understand what we did?	<i>tabBarIcon and routeName were used in the code.</i>
Can you do this for your own app now? Also, another challenge. I want you to design the input form on the Transaction Screen of the App and autofill the book id field and student id field when QR code is scanned.	ESR: Yes!
Teacher Stops Screen Share	
	Now it's your turn. Please share your screen with me.
<div>  Teacher starts slideshow : Slide 12 - 18 </div>	
Run the presentation slide to set the student activity context.	
<div>  Teacher ends slideshow </div>	
STUDENT-LED ACTIVITY - 20 mins	
<ul style="list-style-type: none"> • Ask the student to press the ESC key to come back to the panel. • Guide the student to start Screen Share. • The teacher gets into Fullscreen. 	
<p align="center"><u>ACTIVITY</u></p> <ul style="list-style-type: none"> • Design the input screen used for book transactions. • autofill the text inputs when QR code is scanned. 	
Teacher Action	Student Action

<p><i>Guide the student to add the custom fonts to the app.</i></p> <p><i>Guide the student to add icons on the Bottom Navigation Bar in his/her app.</i></p>	<p><i>The student codes to add the custom fonts.</i></p> <p><i>The student writes code to add the icons.</i></p>
<p><i>Guide the student to make the input form on the Transaction Screen.</i></p> <ul style="list-style-type: none"> • The input form should contain two TextInput components - which will accept book id and student id respectively. • There should be two scan buttons next to the text input to allow the user to scan the QR code and autofill the text input fields. • There should be a submit button to submit the information entered. 	<ul style="list-style-type: none"> • The student imports TextInput and TouchableOpacity Components in the Transaction Screen. • The student writes code to add the input form to the transaction screen. • The student puts different sets of components in different Views (using Box Model - studied during flex box). • The student also styles the components to get the desired effect.
<p>code to Create an input form</p> <pre> return (<View style={styles.container}> <View style={styles.lowerContainer}> <View style={styles.textinputContainer}> </pre>	

```

    <TextInput
      style={styles.textinput}
      placeholder={"Book Id"}
      placeholderTextColor={"#FFFFFF"}
      value={bookId}
    />
    <TouchableOpacity
      style={styles.scanbutton}
    >
      <Text style={styles.scanbuttonText}>Scan</Text>
    </TouchableOpacity>
  </View>

  <View style={[styles.textinputContainer, { marginTop: 25 }]}>
    <TextInput
      style={styles.textinput}
      placeholder={"Student Id"}
      placeholderTextColor={"#FFFFFF"}
      value={studentId}
    />
    <TouchableOpacity
      style={styles.scanbutton}
    >
      <Text style={styles.scanbuttonText}>Scan</Text>
    </TouchableOpacity>
  </View>
</View>
</View>
);

```

```

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#FFFFFF"
  },
  lowerContainer: {
    flex: 0.5,

```

```
alignItems: "center"
},
textInputContainer: {
  borderWidth: 2,
  borderRadius: 10,
  flexDirection: "row",
  backgroundColor: "#9DFD24",
  borderColor: "#FFFFFF"
},
textInput: {
  width: "57%",
  height: 50,
  padding: 10,
  borderColor: "#FFFFFF",
  borderRadius: 10,
  borderWidth: 3,
  fontSize: 18,
  backgroundColor: "#5653D4",
  fontFamily: "Rajdhani_600SemiBold",
  color: "#FFFFFF"
},
scanbutton: {
  width: 100,
  height: 50,
  backgroundColor: "#9DFD24",
  borderTopRightRadius: 10,
  borderBottomRightRadius: 10,
  justifyContent: "center",
  alignItems: "center"
},
scanbuttonText: {
  fontSize: 24,
  color: "#0A0101",
  fontFamily: "Rajdhani_600SemiBold"
}
});
```

The teacher helps the student download assets from [Student Activity 3](#).

Let's also add a logo, background image and give our app some name.

To add the Background image and other images, we'll first need to import the ImageBackground and the image component from react native.

- *The student downloads the assets from [student activity 3](#) and adds to the folder.*
- *The student adds a background Image , app Name and app Icon.*
- *He/She also adds style to the text and image.*

Code to add the icons and image background to the app.

```
import {
  View,
  StyleSheet,
  TextInput,
  TouchableOpacity,
  Text,
  ImageBackground,
  Image
} from "react-native";
```

```
const bgImage = require("../assets/background2.png");
const appIcon = require("../assets/appIcon.png");
const appName = require("../assets/appName.png");
```

```
return (
  <View style={styles.container}>
    <ImageBackground source={bgImage} style={styles.bgImage}>
      <View style={styles.upperContainer}>
        <Image source={appIcon} style={styles.appIcon} />
        <Image source={appName} style={styles.appName} />
      </View>

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

```

<View style={styles.textinputContainer}>
  <TextInput
    style={styles.textinput}
    placeholder={"Book Id"}
    placeholderTextColor={"#FFFFFF"}
    value={bookId}
  />
  <TouchableOpacity
    style={styles.scanbutton}
    onPress={() => this.getCameraPermissions("bookId")}
  >
    <Text style={styles.scanbuttonText}>Scan</Text>
  </TouchableOpacity>
</View>
<View style={[styles.textinputContainer, { marginTop: 25 }]}>
  <TextInput
    style={styles.textinput}
    placeholder={"Student Id"}
    placeholderTextColor={"#FFFFFF"}
    value={studentId}
  />
  <TouchableOpacity
    style={styles.scanbutton}
    onPress={() => this.getCameraPermissions("studentId")}
  >
    <Text style={styles.scanbuttonText}>Scan</Text>
  </TouchableOpacity>
</View>
</View>
</ImageBackground>
</View>
);

```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#FFFFFF"
  },
  bgImage: {
    flex: 1,
    resizeMode: "cover",
    justifyContent: "center"
  },
  upperContainer: {
    flex: 0.5,
    justifyContent: "center",
    alignItems: "center"
  },
  appIcon: {
    width: 200,
    height: 200,
    resizeMode: "contain",
    marginTop: 80
  },
  appName: {
    width: 80,
    height: 80,
    resizeMode: "contain"
  },
  lowerContainer: {
    flex: 0.5,
    alignItems: "center"
  },
});
```


<p>We already have all the functions written to scan and get the information from QR code. How can we use it to autofill the TextInput fields?</p> <p>Hint: It is similar to what we did in the last class.</p> <p>Scan buttons should trigger the getCameraPermissions() function. We should be able to detect which scan button has been pressed using "buttonState" and change the value prop of TextInput. How can we do that?</p>	<p>ESR:</p> <p>We will call the getCameraPermissions() function when the scan Buttons are clicked.</p> <p>ESR:</p> <p>We will change the state of the buttonState depending on which button is clicked. We will store the scanned data into different states.</p>
<p>Alright! Let's try to do this.</p> <p>Let us add two more states:</p> <ul style="list-style-type: none"> ● bookId: It will store the scanned book id data. ● studentId: It will store the scanned student id data. 	<p><i>The student adds the new states.</i></p>
<p>adding states to store the bookId and studentId</p> <pre>export default class TransactionScreen extends Component { constructor(props) { super(props); this.state = { bookId: "", studentId: "", domState: "normal", hasCameraPermissions: null, scanned: false }; } }</pre>	

Let's call the **getCameraPermissions()** function when any of the scan buttons are clicked. But this time, let's pass which button is clicked as an argument.

Inside the **getCameraPermissions()** function, we will set the domState depending on which button is clicked.

*The student writes code to pass the buttonId for whichever button is pressed and sets the state for buttonState inside **getCameraPermissions ()** function.*

code to set the domState depending on the button pressed.

```
getCameraPermissions = async domState => {  
  const { status } = await Permissions.askAsync(Permissions.CAMERA);  
  
  this.setState({  
    /*status === "granted" is true when user has granted permission  
     status === "granted" is false when user has not granted the permission  
    */  
    hasCameraPermissions: status === "granted",  
    domState: domState,  
    scanned: false  
  });  
};
```

```
return ([
  <View style={styles.container}>
    <View style={styles.lowerContainer}>
      <View style={styles.textinputContainer}>
        <TextInput
          style={styles.textinput}
          placeholder={"Book Id"}
          placeholderTextColor={"#FFFFFF"}
          value={bookId}
        />
        <TouchableOpacity
          style={styles.scanbutton}
          onPress={() => this.getCameraPermissions("bookId")}
        >
          <Text style={styles.scanbuttonText}>Scan</Text>
        </TouchableOpacity>
      </View>
      <View style={[styles.textinputContainer, { marginTop: 25 }]}>
        <TextInput
          style={styles.textinput}
          placeholder={"Student Id"}
          placeholderTextColor={"#FFFFFF"}
          value={studentId}
        />
        <TouchableOpacity
          style={styles.scanbutton}
          onPress={() => this.getCameraPermissions("studentId")}
        >
          <Text style={styles.scanbuttonText}>Scan</Text>
        </TouchableOpacity>
      </View>
    </View>
  </View>
]);
```

When the Barcode is scanned, we need to set the state of bookId and studentId depending on which button is pressed. Where will we do this?

Let's also set the value for text input to these states which we have scanned.

ESR:

Inside
handleBarCodeScanned
function

The student writes code to

*change the states for
studentId and bookId.*

Code to update the domState and set values to the text boxes.

```
handleBarcodeScanned = async ({ type, data }) => {  
  const { domState } = this.state;  
  
  if (domState === "bookId") {  
    this.setState({  
      bookId: data,  
      domState: "normal",  
      scanned: true  
    });  
  } else if (domState === "studentId") {  
    this.setState({  
      studentId: data,  
      domState: "normal",  
      scanned: true  
    });  
  }  
};
```

```
return ([
  <View style={styles.container}>
    <View style={styles.lowerContainer}>
      <View style={styles.textinputContainer}>
        <TextInput
          style={styles.textinput}
          placeholder={"Book Id"}
          placeholderTextColor={"#FFFFFF"}
          value={bookId}
        />
        <TouchableOpacity
          style={styles.scanbutton}
          onPress={() => this.getCameraPermissions("bookId")}
        >
          <Text style={styles.scanbuttonText}>Scan</Text>
        </TouchableOpacity>
      </View>
      <View style={[styles.textinputContainer, { marginTop: 25 }]}>
        <TextInput
          style={styles.textinput}
          placeholder={"Student Id"}
          placeholderTextColor={"#FFFFFF"}
          value={studentId}
        />
        <TouchableOpacity
          style={styles.scanbutton}
          onPress={() => this.getCameraPermissions("studentId")}
        >
          <Text style={styles.scanbuttonText}>Scan</Text>
        </TouchableOpacity>
      </View>
    </View>
  </View>
]);
```

We want to display the barcode scanner as soon as the button changes from "normal". We can change the condition to render the BarCodeScanner component accordingly.

Let's do that.

The student changes the condition to render the BarCodeScanner Component.

code to display the barcode scanner.


```
render() {  
  const { bookId, studentId, domState, scanned } = this.state;  
  if (domState !== "normal") {  
    return (  
      <BarCodeScanner  
        onBarCodeScanned={scanned ? undefined : this.handleBarCodeScanned}  
        style={StyleSheet.absoluteFillObject}  
      />  
    );  
  }  
}
```

Let's try to run our code and test if this works.




The student runs the code on their system using expo start.

He/She presses the scan buttons, scans any QR code and checks if the text input gets populated.

	
Great job! Now we can see that our Text Input boxes are getting filled when the QR code is scanned.	

Let's summarise what we did today.	
Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 5 Mins	
<div>  </div> Teacher starts slideshow from slide 19 to slide 30	
Activity details	Solution/Guidelines
Run the presentation from slide 11 to slide 24 Following are the warm up session deliverables: <ul style="list-style-type: none"> ● Explain the facts and trivias ● Next class challenge ● Project for the day ● Additional Activity 	Guide the student to develop the project and share with us.
Quiz time - Click on in-class quiz	
Question	Answer
Select the correct option wrt the route.name property of the Navigation object. A. if Transaction Screen is active, routeName === 'Transaction' B. if Search Screen is active, routeName === 'Search' C. Both A and B D. if Search Screen is inactive , routeName = 'Search'	C
Which props for images can help us add different sources for image components and give them width and height? A. Style and resizeMode props B. Source and resizeMode props C. resizeMode and resizeMode props D. Source and Style props	D

<p>Why do we use <code>buttonState !== "normal"</code> to display the barcode scanner?</p> <p>A. it is an alternate way to write <code>buttonState === "clicked"</code></p> <p>B. because the <code>buttonState</code> does not have a "clicked" value</p> <p>C. both "normal" and "clicked" values mean the same</p> <p>D. <code>!==</code> means the button is clicked</p>	<p>B</p>
<p>End the quiz panel</p>	
Teacher Action	Student Action
<p>Awesome! We did a lot of coding today. Can you capture what we learned in today's class?</p>	<p>ESR: We learned how to customize the appearance of tab navigation by adding images to the tabs. We also learned how to scan the QR codes and autofill the text input box .</p>
<p>Great.</p> <p>In the next class, we will be learning what to do with the data that we have just collected. We will learn more about databases and how to use them in our app.</p> <p>I am excited and looking forward to the next class.</p>	
<p><u>FEEDBACK</u></p> <ul style="list-style-type: none"> Encourage the student to think about how to store and process the information in the database. 	
Teacher Action	Student Action

<p>You get a “hats off”.</p> <p>Till next class then. See you. Bye!</p>	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div data-bbox="1019 394 1312 495">  +10 Creatively Solved Activities </div> <div data-bbox="1019 558 1312 659">  +10 Great Question </div> <div data-bbox="1019 722 1312 823">  +10 Strong Concentration </div>
<p><i>* This Project will take only 30 mins to complete. Motivate students to try and finish it immediately after the class.</i></p> <p>Project Overview E-RIDE STAGE 3 Goal of the Project:</p> <p>In class 70, you have learned how to design the input form for the app's issue/return screen. You also wrote code to automatically input information when book id and student ids are scanned.</p> <p>You also learn to customize the appearance of tab navigation by adding images/icons to the tabs. You have to use the same concepts to complete this project by customizing the tab navigation and accessing permission for the camera.</p> <p><i>* This is a continuation of Project-68 & 69; make sure you have completed and submitted that before attempting this one.</i></p> <p>Story:</p>	

The QR code functionality you added in the last project is very impressive. Vihaan will add a QR code on each bicycle. Apart from that he wants a place for users to enter their userID. You can also make your app attractive with beautiful images/icons.

I am very excited to see your project solution and I know you will do really well.

Bye Bye!

Teacher ends slideshow



Teacher Clicks

✕ End Class

ADDITIONAL ACTIVITIES

Additional Activities

Encourage the student to write reflection notes in their reflection journal using markdown.

Use these as guiding questions:

- What happened today?
 - Describe what happened
 - Code I wrote
- How did I feel after the class?
- What have I learned about programming and developing games?
- What aspects of the class helped me?
- What did I find difficult?

The student uses the markdown editor to write her/his reflection as a reflection journal.

Activity	Activity Name	Links
----------	---------------	-------

Teacher Activity 1	Code for previous class	https://github.com/whitehatjr/e-library-v2-PRO-C69
Teacher Activity 2	Expo-fonts documentation	https://docs.expo.io/versions/latest/sdk/font/
Teacher Activity 3	Expo-google-fonts rajdhani documentation	https://www.npmjs.com/package/@expo-google-fonts/rajdhani
Teacher Activity 4	Customising tab Navigation	https://reactnavigation.org/docs/en/tab-based-navigation.html
Teacher Activity 5	Bottom Tab navigation config	https://reactnavigation.org/docs/en/bottom-tab-navigator.html
Teacher Activity 6	Reference code	https://github.com/whitehatjr/e-library-v2-PRO-C70
Student Activity 1	Customising tab Navigation	https://reactnavigation.org/docs/en/tab-based-navigation.html
Student Activity 2	Bottom Tab navigation config	https://reactnavigation.org/docs/en/bottom-tab-navigator.html
Student Activity 3	Assets link	https://s3-whjr-curriculum-uploads.whjr.online/68478465-f230-4b76-a901-a9d33f9f008a.zip
Visual-Aid	Visual -Aid Links	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C70-withcues.html
In-Class Quiz	in-class quiz links	https://s3-whjr-curriculum-uploads.whjr.online/c03283ba-0c1e-4209-9804-8dd63aeef9ec.pdf
Project Solution	E-Ride-Stage-3	https://github.com/whitehatjr/PRO-C70-PROJECT

