QR CODE SCANNER

What is our GOAL for this MODULE?

We learned how to perform QR Code/Barcode scanning in a React Native application. We also learned to compare how books and ID cards are scanned by barcode scanning devices.

What did we ACHIEVE in the class TODAY?

- Got permission to use the Camera in the application.
- Used BarCodeScanner component to scan a QR code.
- Displayed data from QR code inside a text component.

Which CONCEPTS/ CODING BLOCKS did we cover today?

- Permissions
- BarCodeScanner Component
- TextInput

What did we REVISE today?

- User stories for Wireless Library Management system
- Tab Navigation
- Creating bottom tab navigation in our application.

How did we DO the activities?

1. Create a Button (using **TouchableOpacity** here) which will trigger the QR code scanner and display the scanned output as text.

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

export default class TransactionScreen extends Component {
  render() {

   return (
     <View style={styles.container}>
       <TouchableOpacity>
         <Text>Scan QR Code</Text>
       </TouchableOpacity>
     </View>
   );
 }
}
```

2. Add some styling to our **TouchableOpacity** and **Text** using StyleSheet.

```
export default class TransactionScreen extends Component {
  render() {

   return (
     <View style={styles.container}>

       <TouchableOpacity
         style={styles.button}
       >
         <Text style={styles.buttonText}>Scan QR Code</Text>
```

```
        </TouchableOpacity>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#5653D4"
  },
  text: {
    color: "#ffff",
    fontSize: 15
  },
  button: {
    width: "43%",
    height: 55,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F48D20",
    borderRadius: 15
  },
  buttonText: {
    fontSize: 24,
    color: "#FFFFFF"
  }
});
```

3. Use an expo package (library) which will help us build a QR code scanner in our application.

```
C:\Users\ADMIN\e-library>expo install expo-barcode-scanner

   There is a new version of expo-cli available (4.6.0).
   You are currently using expo-cli 4.4.3
   Install expo-cli globally using the package manager of your choice;
   for example: `npm install -g expo-cli` to get the latest version

Installing 1 SDK 41.0.0 compatible native module using npm.
```

4. Import **Barcode** scanner component and permissions.

```
C:\Users\ADMIN\e-library>expo install expo-permissions

   There is a new version of expo-cli available (4.6.0).
   You are currently using expo-cli 4.4.3
   Install expo-cli globally using the package manager of your choice;
```

```
import React, { Component } from "react";
import { View, Text, StyleSheet, TouchableOpacity } from
"react-native";
import * as Permissions from "expo-permissions";
import { BarCodeScanner } from "expo-barcode-scanner";


export default class TransactionScreen extends Component {
  render() {


   return (
     <View style={styles.container}>


       <TouchableOpacity
         style={styles.button}
```

```
        >
          <Text style={styles.buttonText}>Scan QR Code</Text>
        </TouchableOpacity>
      </View>
    );
  }
}
```

5. Define three states in our application:
    - **hasCameraPermissions: null** - This will tell if the user has granted camera permission to the application.
    - **scanned: false** - This will tell if the scanning has completed or not.
    - **scannedData: ''** - This will hold the scanned data that we get after scanning.

```
import React, { Component } from "react";
import { View, Text, StyleSheet, TouchableOpacity } from
"react-native";
import * as Permissions from "expo-permissions";
import { BarCodeScanner } from "expo-barcode-scanner";


export default class TransactionScreen extends Component {
 constructor(props){
   super(props);
   this.state = {
     domState : "normal",
     hasCameraPermissions: null,
     scanned: false,
     scannedData: ""
   }
 }
```

6. Enable the camera permissions when the Scan QR Code Button is pressed in our application:
    - Write a function - **getCameraPermission** - which can request for camera permission.
        - Note that this function needs to be asynchronous because it takes time for the user to give camera permission to the application.
    - The **Permission** component which is imported has a predefined function called **.askAsync()** which can request for various permissions.
        - Use the **.askAsync()** to request for camera permission inside the function and change the state of **hasCameraPermissions**.
    - Note that **.askAsync()** returns an object with a 'status' key containing the status of the permission granted by the user. If the user grants permission, status changes to 'granted'
    - *Note: **{status}** automatically extracts the value from the object with key 'status'.

```
import React, { Component } from "react";
import { View, Text, StyleSheet, TouchableOpacity } from
"react-native";
import * as Permissions from "expo-permissions";
import { BarCodeScanner } from "expo-barcode-scanner";


export default class TransactionScreen extends Component {
 constructor(props){
   super(props);
   this.state = {
     domState : "normal",
     hasCameraPermissions: null,
     scanned: false,
     scannedData: ""
   }
 }


 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
   });
 };
```

7. Test the code.
    - The camera permission is asked only once. If you grant the permission, the application will remember it.

```
render() {


   return (
     <View style={styles.container}>


       <TouchableOpacity
         style={[styles.button, { marginTop: 25 }]}
         onPress={() => this.getCameraPermissions("scanner")}
       >
         <Text style={styles.buttonText}>Scan QR Code</Text>
       </TouchableOpacity>
     </View>
   );
 }
```

8. Write code in our **render()** function.
    - Note that JSX can only be written in the return function. If JavaScript code is to be written it is done inside the curly {} brackets.
    - Display a text "Request for camera permission" if **hasCameraPermissions** is false or null.
    - If **hasCameraPermissions** is 'true', Display whatever text is inside the **scannedData** state.
    - Currently **scannedData** is an empty string.

```
render() {
   const { domState, hasCameraPermissions, scannedData,
scanned } = this.state;

   return (
     <View style={styles.container}>
       <Text style={styles.text}>
         {hasCameraPermissions ? scannedData : "Request for
Camera Permission"}
       </Text>
       <TouchableOpacity
         style={[styles.button, { marginTop: 25 }]}
         onPress={() => this.getCameraPermissions("scanner")}
       >
         <Text style={styles.buttonText}>Scan QR Code</Text>
       </TouchableOpacity>
     </View>
   );
 }
}
```

9. Display **BarCodeScanner** when Scan Button is clicked:

- Create a new state called **buttonState** which keeps track if the button has been clicked.
    - **domState** will be **'normal'** when the application starts.
    - When the button is clicked to get camera permissions, **domState** should change to '**clicked**'.

```
constructor(props){
  super(props);
  this.state = {
    domState : "normal",
    hasCameraPermissions: null,
    scanned: false,
    scannedData: ""
  }
}
```

10. Return a **BarCodeScanner** component when the button is clicked and the user has given camera permissions.
    - **BarCodeScanner** component automatically starts scanning using the Camera.
    - It has a prop called **onBarCodeScanned** which can call a function to handle data received after scanning.
    - Call this function only when scanned is false.

11. Code to write a function called **handleBarCodeScanned()** which is called when the scan is completed.
    - This function automatically receives the type of barcode scanned and the data inside the barcode.
    - **scannedData** here can be set to equal to the data received after scanning.
    - Once the scan has been completed, we also want to set the scanned state to true.
    - Change the state for the button to make it back to normal when the scan is completed.

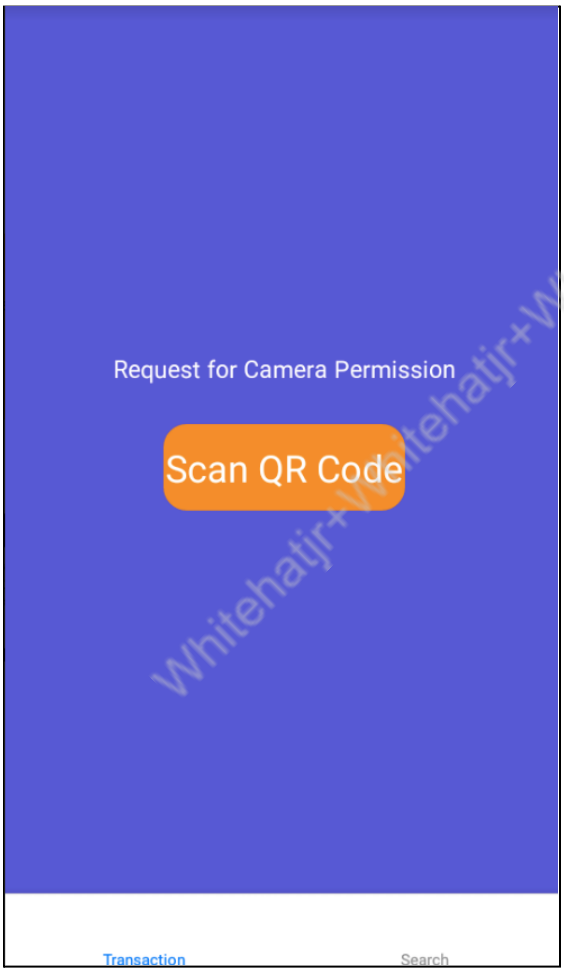
```
getCameraPermissions = async domState => {
  const { status } = await
Permissions.askAsync(Permissions.CAMERA);
```

```
    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
    });
  };

  handleBarCodeScanned = async ({ type, data }) => {
    this.setState({
      scannedData: data,
      domState: "normal",
      scanned: true
    });
  };

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

```
        );
    }
```

12. Render the text and the **TouchableOpacity** button, which was displayed earlier when the **buttonState** is normal.

13. To scan the QR code again when the button is clicked.
    - Make sure to set the scanned state to 'false' again. Otherwise, the **handleBarCodeScanned()** function will not be called.



What's NEXT?

In the next class, we will write code to automatically input information when book id and student ids are scanned.

EXTEND YOUR KNOWLEDGE

1. Expo BarcodeScanner: https://docs.expo.io/versions/v35.0.0/sdk/bar-code-scanner/