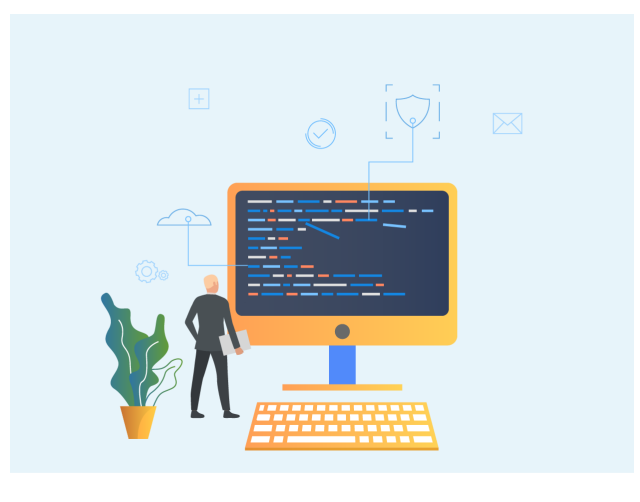


AVOIDING KEYBOARD OVERLAP AND TOASTS



What is our GOAL for this MODULE?

In this class, we learned to explore scenarios where how manually entering the text in a text box is beneficial. Also, we solve the issue of the overlapping keyboard on text. And finally, display a transaction message to the users using toasts or alerts.

What did we ACHIEVE in the class TODAY?

- Made the text box editable.
- Avoided keyboard layout overlap with the text box.
- Displayed a Transaction message when a transaction is completed.

Which CONCEPTS/CODING BLOCKS did we cover today?

- Avoiding Keyboard Overlap and Toasts.
- To make the Text box editable.

How did we DO the activities?

To issue a book to a student, you first needed to get the details of the student and the book from the database, after which you issue or return the book. Also, towards the end, you show the Alert/Toast message when the transaction is done.

1. Write the **getBookDetails()** function to get the book details.

```
getBookDetails = bookId => {  
  bookId = bookId.trim();  
  db.collection("books")  
    .where("book_id", "==", bookId)  
    .get()  
    .then(snapshot => {  
      snapshot.docs.map(doc => {  
        this.setState({  
          bookName: doc.data().book_details.book_name  
        });  
      });  
    });  
};
```

2. Write the **getStudentDetails()** to get the student details.

```
getStudentDetails = studentId => {  
  studentId = studentId.trim();  
  db.collection("students")  
    .where("student_id", "==", studentId)  
    .get()  
    .then(snapshot => {  
      snapshot.docs.map(doc => {  
        this.setState({  
          studentName: doc.data().student_details.student_name  
        });  
      });  
    });  
};
```

3. Call these functions inside the **handleTransaction()** function.

```
handleTransaction = async () => {
  var { bookId, studentId } = this.state;
  await this.getBookDetails(bookId);
  await this.getStudentDetails(studentId);
}
```

4. Write the **initiateBookIssue()** function. This function will take four parameters as follows:

- **StudentId,**
- **Student Name,**
- **bookId,**
- and **bookName.**

```
initiateBookIssue = async (bookId, studentId, bookName, studentName) => {
  //add a transaction
  db.collection("transactions").add({
    student_id: studentId,
    student_name: studentName,
    book_id: bookId,
    book_name: bookName,
    date: firebase.firestore.Timestamp.now().toDate(),
    transaction_type: "issue"
  });
  //change book status
  db.collection("books")
    .doc(bookId)
    .update({
      is_book_available: false
    });
  //change number of issued books for student
  db.collection("students")
    .doc(studentId)
    .update({
      number_of_books_issued: firebase.firestore.FieldValue.increment(1)
    });

  // Updating local state
  this.setState({
    bookId: "",
    studentId: ""
  });
};
```

5. Write the **initiateBookReturn()** function. Here, the **transaction_type** would be set as **return**. And **is_book_available** would be set as **true**. Update the number of books issued for a student, and update the **bookId** and **studentId**.

```
initiateBookReturn = async (bookId, studentId, bookName, studentName) => {  
  //add a transaction  
  db.collection("transactions").add({  
    student_id: studentId,  
    student_name: studentName,  
    book_id: bookId,  
    book_name: bookName,  
    date: firebase.firestore.Timestamp.now().toDate(),  
    transaction_type: "return"  
  });  
  //change book status  
  db.collection("books")  
    .doc(bookId)  
    .update({  
      is_book_available: true  
    });  
  //change number of issued books for student  
  db.collection("students")  
    .doc(studentId)  
    .update({  
      number_of_books_issued: firebase.firestore.FieldValue.increment(-1)  
    });  
  // Updating local state  
  this.setState({  
    bookId: "",  
    studentId: ""  
  });  
};
```

```
handleTransaction = async () => {
  var { bookId, studentId } = this.state;
  await this.getBookDetails(bookId);
  await this.getStudentDetails(studentId);

  db.collection("books")
    .doc(bookId)
    .get()
    .then(doc => {
      var book = doc.data();
      if (book.is_book_available) {
        var { bookName, studentName } = this.state;
        this.initiateBookIssue(bookId, studentId, bookName, studentName);
      } else {
        var { bookName, studentName } = this.state;
        this.initiateBookReturn(bookId, studentId, bookName, studentName);
      }
    })
  });
};
```

We have now written the code to get the details of the book and the student to issue or return the book to the student.

From here on ahead, we fix the issue of the overlapping keyboard on the text input boxes.

6. Use the **KeyboardAvoidingView** to solve the problem of the keyboard overlapping the **TextInput** boxes.

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

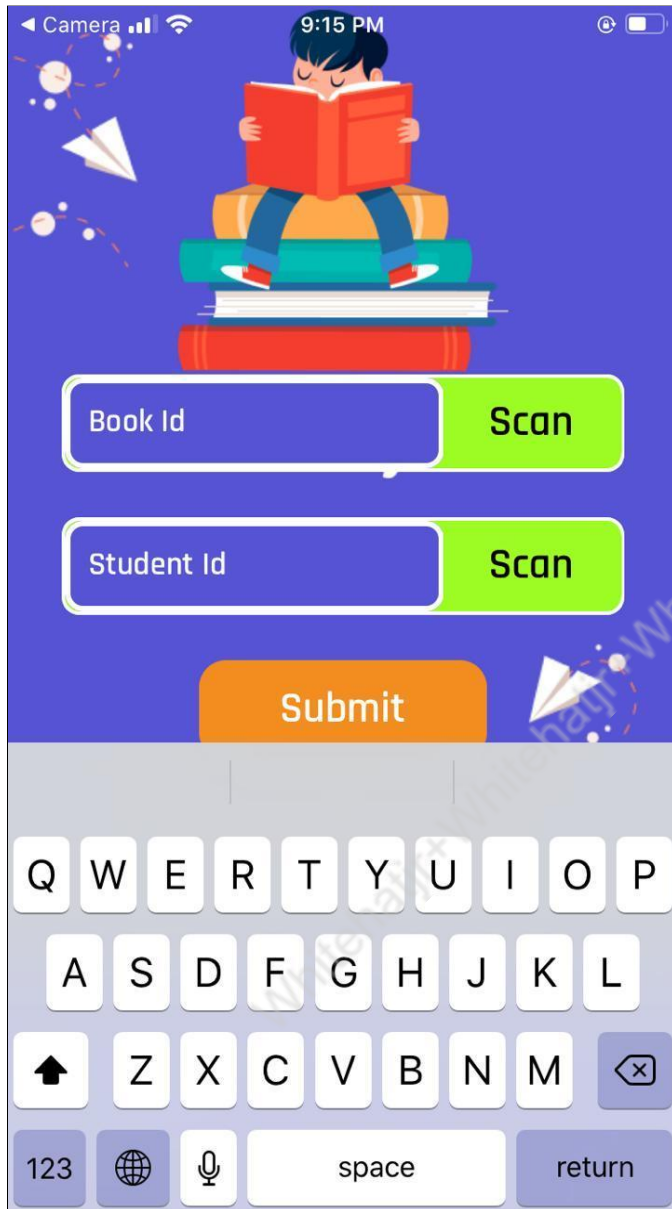
```
screens > JS Transaction.js > TransactionScreen > handleTransaction
190
197     };
198   }
199   return (
200     <KeyboardAvoidingView behavior="padding" style={styles.container}>
201       <ImageBackground source={bgImage} style={styles.bgImage}>
202         <View style={styles.upperContainer}>
203           <Image source={appIcon} style={styles.appIcon} />
204           <Image source={appName} style={styles.appName} />
205         </View>
206         <View style={styles.lowerContainer}>
207           <View style={styles.textinputContainer}>
208             <TextInput
209               style={styles.textinput}
210               placeholder={"Book Id"}
211               placeholderTextColor={"#FFFFFF"}
212               value={bookId}
213               onChangeText={text => this.setState({ bookId: text })}
214             />
215             <TouchableOpacity
216               style={styles.scanbutton}
217               onPress={() => this.getCameraPermissions("bookId")}
218             >
219               <Text style={styles.scanbuttonText}>Scan</Text>
220             </TouchableOpacity>
221           </View>
222           <View style={[styles.textinputContainer, { marginTop: 25 }]}>
223             <TextInput
224               style={styles.textinput}
225               placeholder={"Student Id"}
226               placeholderTextColor={"#FFFFFF"}
227               value={studentId}
228               onChangeText={text => this.setState({ studentId: text })}
229             />
230             <TouchableOpacity
231               style={styles.scanbutton}
232               onPress={() => this.getCameraPermissions("studentId")}
233             >
234               <Text style={styles.scanbuttonText}>Scan</Text>
235             </TouchableOpacity>
```

```

    </TouchableOpacity>
  </View>
</ImageBackground>
</KeyboardAvoidingView>
);

```

Output:

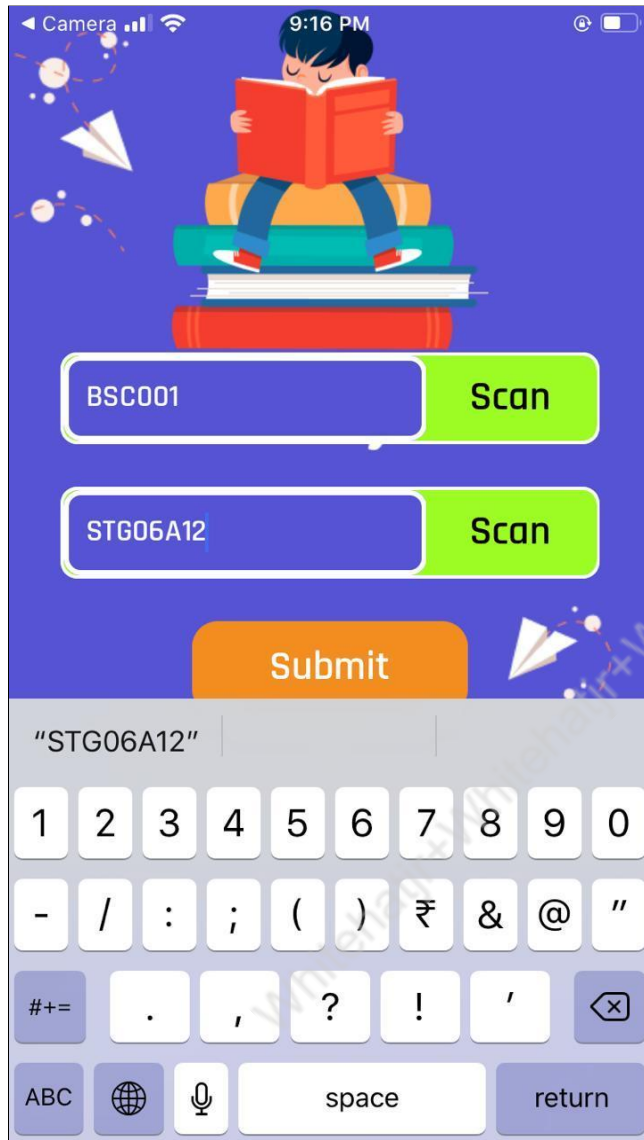


The screenshot shows a mobile application interface with a purple background. At the top, there is a status bar with 'Camera', signal strength, Wi-Fi, and the time '9:15 PM'. The main illustration features a cartoon boy with black hair, wearing an orange shirt and blue pants, sitting cross-legged on a stack of four books (red, green, white, and red from top to bottom). He is holding a large red book open and reading. To the left of the books are some decorative elements: a white paper airplane and several small white circles with dashed lines. Below the illustration, there are two input fields. The first is labeled 'Book Id' and is followed by a green button labeled 'Scan'. The second is labeled 'Student Id' and is also followed by a green button labeled 'Scan'. Below these is a large orange button labeled 'Submit'. At the bottom of the screen is a virtual keyboard with white keys on a grey background. The keyboard layout includes rows for QWERTY, ASDFGHJKL, ZXCVBNM, and a bottom row with '123', a globe icon, a microphone icon, a 'space' button, and a 'return' button.

- Get the text typed by the user in **TextInput** as the default argument, and use the **onChange** prop to set the values for student ID and book ID.

```
screens > JS Transaction.js > TransactionScreen > render
203 <Image source={appIcon} style={styles.appIcon} />
204 <Image source={appName} style={styles.appName} />
205 </View>
206 <View style={styles.lowerContainer}>
207   <View style={styles.textinputContainer}>
208     <TextInput
209       style={styles.textinput}
210       placeholder={"Book Id"}
211       placeholderTextColor={"#FFFFFF"}
212       value={bookId}
213       onChangeText={text => this.setState({ bookId: text })}
214     />
215     <TouchableOpacity
216       style={styles.scanbutton}
217       onPress={() => this.getCameraPermissions("bookId")}
218     >
219       <Text style={styles.scanbuttonText}>Scan</Text>
220     </TouchableOpacity>
221   </View>
222   <View style={styles.textinputContainer, { marginTop: 25 }}>
223     <TextInput
224       style={styles.textinput}
225       placeholder={"Student Id"}
226       placeholderTextColor={"#FFFFFF"}
227       value={studentId}
228       onChangeText={text => this.setState({ studentId: text })}
229     />
230     <TouchableOpacity
231       style={styles.scanbutton}
232       onPress={() => this.getCameraPermissions("studentId")}
233     >
234       <Text style={styles.scanbuttonText}>Scan</Text>
235     </TouchableOpacity>
236   </View>
237   <TouchableOpacity
238     style={[styles.button, { marginTop: 25 }]}
239     onPress={this.handleTransaction}
240   >
241     <Text style={styles.buttonText}>Submit</Text>
```


Output:



Now the text box is editable.

8. Display a message to the user when a transaction (issue or return) is completed, using the **ToastAndroid** Component to display a **Toast** Message.

Note: - **ToastAndroid** can only be used for android devices and not for iOS. For iOS users, use **Alert**.

To code using **Alert**:

```
handleTransaction = async () => {
  var { bookId, studentId } = this.state;
  await this.getBookDetails(bookId);
  await this.getStudentDetails(studentId);

  db.collection("books")
    .doc(bookId)
    .get()
    .then(doc => {
      var book = doc.data();
      if (book.is_book_available) {
        var { bookName, studentName } = this.state;
        this.initiateBookIssue(bookId, studentId, bookName, studentName);

        Alert.alert("Book issued to the student!");
      } else {
        var { bookName, studentName } = this.state;
        this.initiateBookReturn(bookId, studentId, bookName, studentName);

        Alert.alert("Book returned to the library!");
      }
    })
  });
};
```

To code using **ToastAndroid**:

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

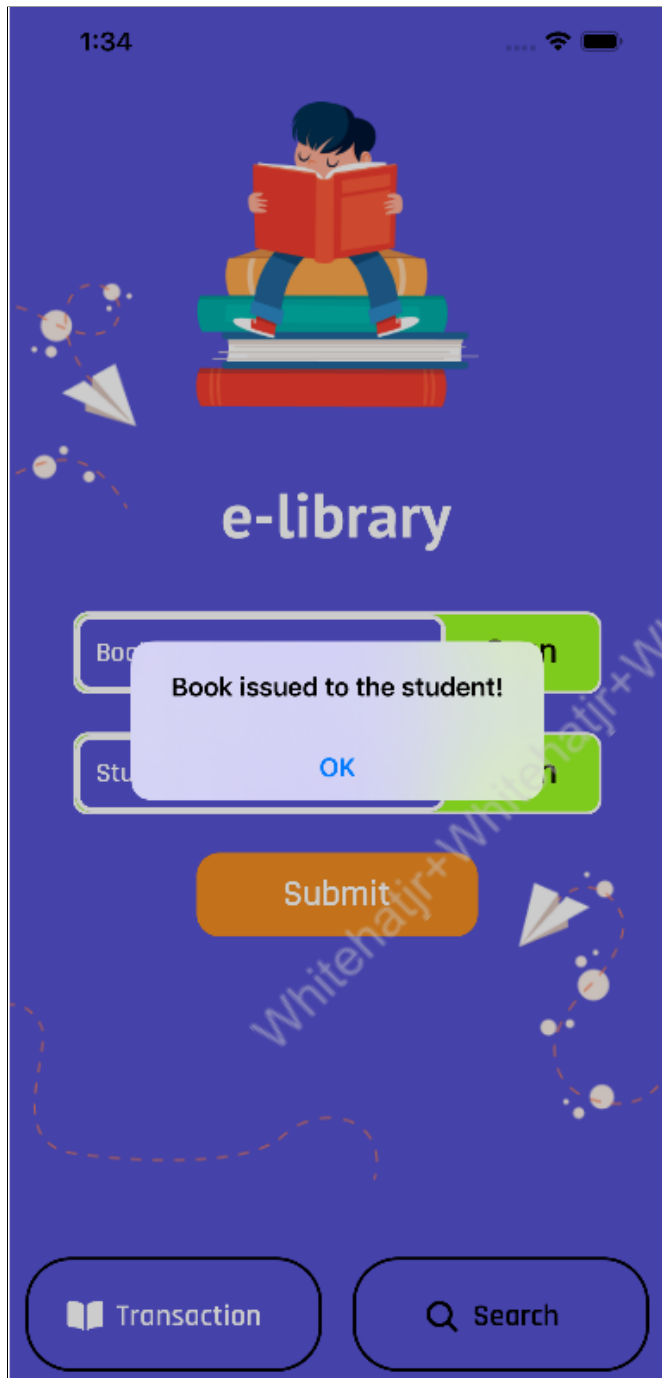
```
handleTransaction = async () => {
  var { bookId, studentId } = this.state;
  await this.getBookDetails(bookId);
  await this.getStudentDetails(studentId);

  db.collection("books")
    .doc(bookId)
    .get()
    .then(doc => {
      var book = doc.data();
      if (book.is_book_available) {
        var { bookName, studentName } = this.state;
        this.initiateBookIssue(bookId, studentId, bookName, studentName);

        // For Android users only
        ToastAndroid.show("Book issued to the student!", ToastAndroid.SHORT);
      } else {
        var { bookName, studentName } = this.state;
        this.initiateBookReturn(bookId, studentId, bookName, studentName);

        // For Android users only
        ToastAndroid.show(
          "Book returned to the library!",
          ToastAndroid.SHORT
        );
      }
    });
};
```

Output:



What's NEXT?

In the next class, we will learn how to make queries to a Firebase Database. Check for student eligibility and book eligibility before issuing/returning a book using firebase queries.

EXTEND YOUR KNOWLEDGE

1. Learn more about ToastAndroid: <https://reactnative.dev/docs/toastandroid>

Whitehatjr+Whitehatjr+Whitehatjr