Topic	**AVOIDING KEYBOARD OVERLAP AND TOASTS**	
Class Description	**The student learns to make the text box editable and allows users to type the book ID and student ID if needed. The student learns to solve the issue of keyboard overlapping on the text input boxes and finally display transaction messages to the users using toasts or alerts.**	
Class	**C72**	
Class time	**45 mins**	
Goal	● Make the text box editable. ● Avoid keyboard layout overlap with the text box. ● Display a Transaction message when a transaction is completed.	
Resources Required	● Teacher Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Android/iOS Smartphone with Expo App installed ● Student Resources ○ 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 min** **20 min** **5 min**

WARM-UP SESSION - 5 mins

CONTEXT

- **Talk about a scenario where typing the student ID and book ID in the text box would be important.**

WhiteHat Jr
Live online Coding for Kids

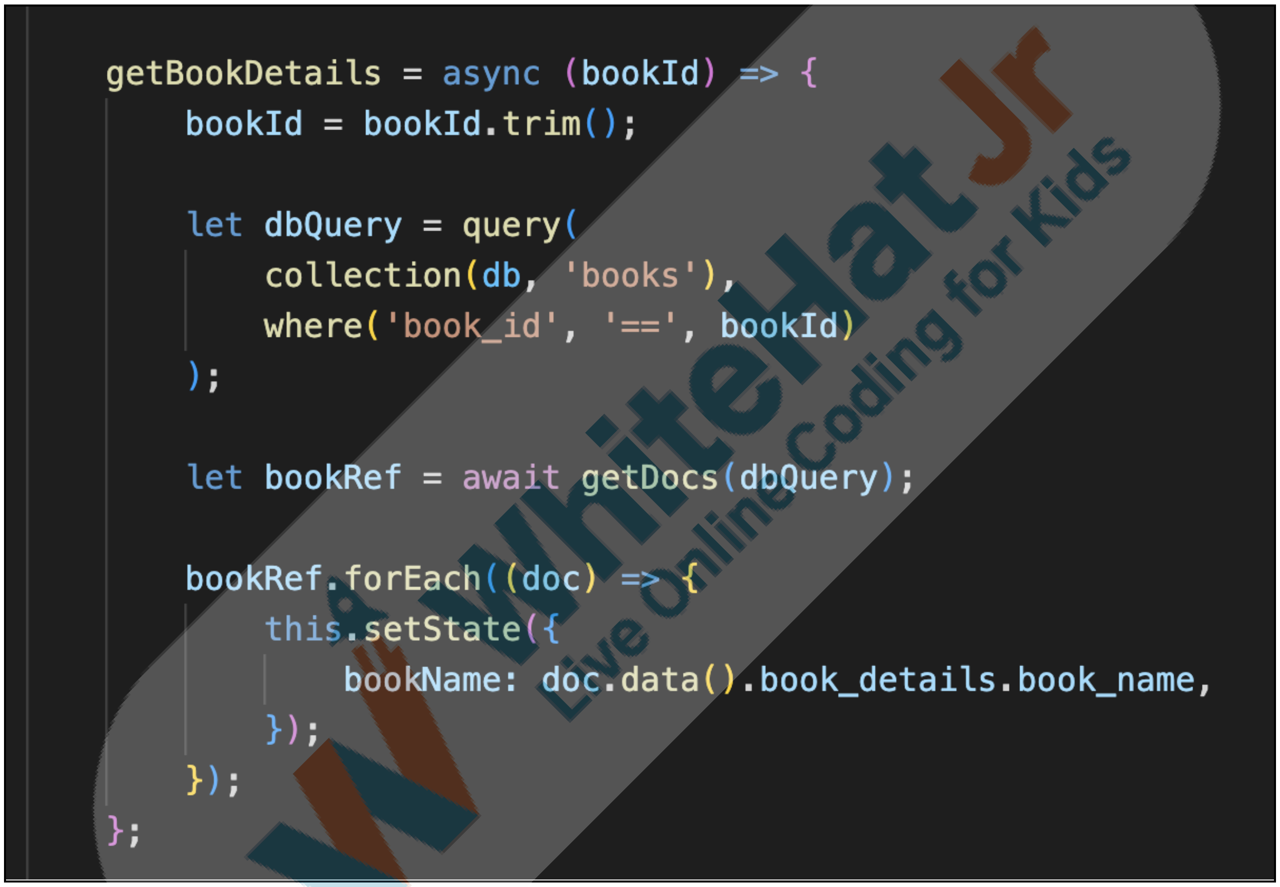
Teacher starts slideshow from slides 1 to 14 Refer to speaker notes and follow the instructions on each slide.	
Activity details	**Solution/Guidelines**
Hi, how have you been? Are you excited to learn something new? ***Run the presentation from slide 1 to slide 3.*** **The following are the warm-up session deliverables:** ● Reconnect with previous class topics. ● Warm-Up quiz session.	**ESR**: Varied Response. Click on the slide show tab and present the slides.
QnA Session	
Question	**Answer**
The collection 'transactions' contains _____. A. bookID B. studentID A. Date and type of transaction B. All of the above	**D**
await receives the _____ as a default argument. A. promise B. data A. document B. time	**C**
Continue the Warm-Up session	

Activity details	Solution/Guidelines
Run the presentation from slide 4 to slide 14 to set the problem statement. **The following are the warm-up session deliverables:** ● Discuss a scenario where typing the student ID and book ID in the text box would be important if there's an issue in scanning.	Narrate the slides 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 ● **Write code for the initiateBookIssue() and initiateBookReturn() functions.** ● **Make the Text box editable.**	
The teacher clones Teacher Activity 1 and installs all the dependencies and opens the code in VS Code. *Steps to clone the project:-* ***git clone <projectURL>*** ***cd <projectFolder>*** ***npm install***	
Before we start, let's go through the code from the previous class and review what we have done so far. The overview points are:	

● Tab Navigation ● Adding Icons to tab navigation. ● Bar code scanning. ● Auto-filling of text input when the barcode is scanned. ● Connecting to the firestore database.	
So in the last class, we just wrote abstract code for issuing and returning the book. What are the changes that we would want to make in the database when we are **issuing** a book to a student?	**ESR:** When we are issuing a book to a student, we would want to ● Create a new transaction and in this transaction, we'll add: 1.StudentId 2.Student Name 3.BookId 4.Book Name 5.Timestamp 6.Transaction type to issue. ● Change the book status/availability of the book to false. ● Change the number of books issued for the student. ● In the local state, update the **bookId** and **studentId**.

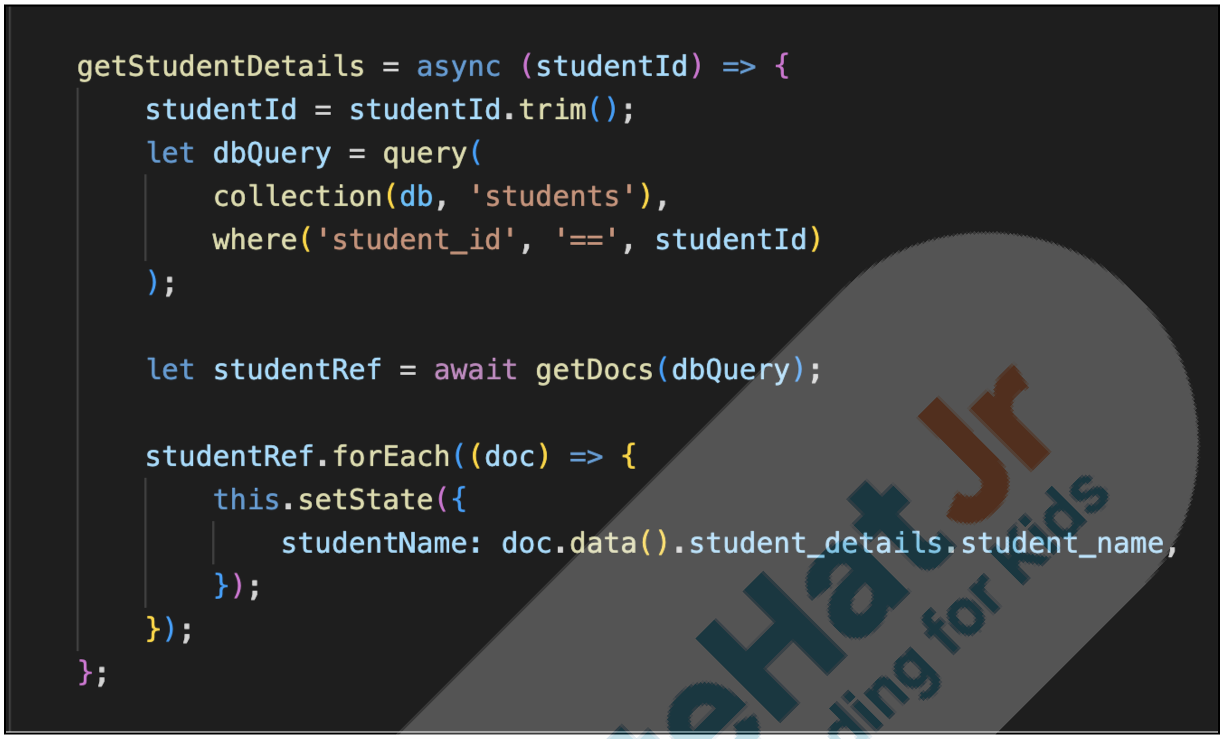
Yes. And what changes would we want to make to the database when the student is **returning** the book? But before we do it all, don't we need to get the details of the books and students to make the transactions?	**ESR:** ● We would make the same changes as we did while issuing the book just that the **transaction type** would be **return**, **book availability** will be **true**. ● Update the number of books issued by the student. ● And in the local state update the **bookId** and **StudentId**. **ESR:** Yes.
***Note**: The code for **getBookDetails** and **getStudentDetails** have already been added to the code. So make sure to update the code or download the boilerplate code from Teacher Activity 1.* To get the details of the book and the student, we'll write two different functions: 1. **getBookDetails()** to get books details. 2. And **getStudentDetails()** to get students details. How can we get a specific book's data from the database? Yes! To do this, we'll make a request to the books' collection, where we'll check if any book ID in the database matches our book ID.	**ESR:** We'll get the specific book by its ID.

In the constructor, we'll create a new state called as **bookName** and **studentName**. When we find the book, using a snapshot, we'll get the data of the book and set it to the **bookName** state. *The teacher explains the code to the student.*	

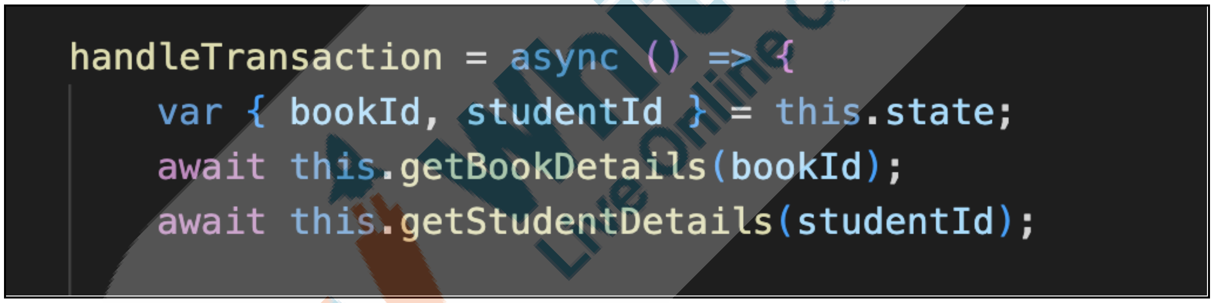


```
getBookDetails = async (bookId) => {
    bookId = bookId.trim();

    let dbQuery = query(
        collection(db, 'books'),
        where('book_id', '==', bookId)
    );

    let bookRef = await getDocs(dbQuery);

    bookRef.forEach((doc) => {
        this.setState({
            bookName: doc.data().book_details.book_name,
        });
    });
};
```

Similarly, we'll do the same for the student details. And then call these functions inside the **handleTransaction()** function.	



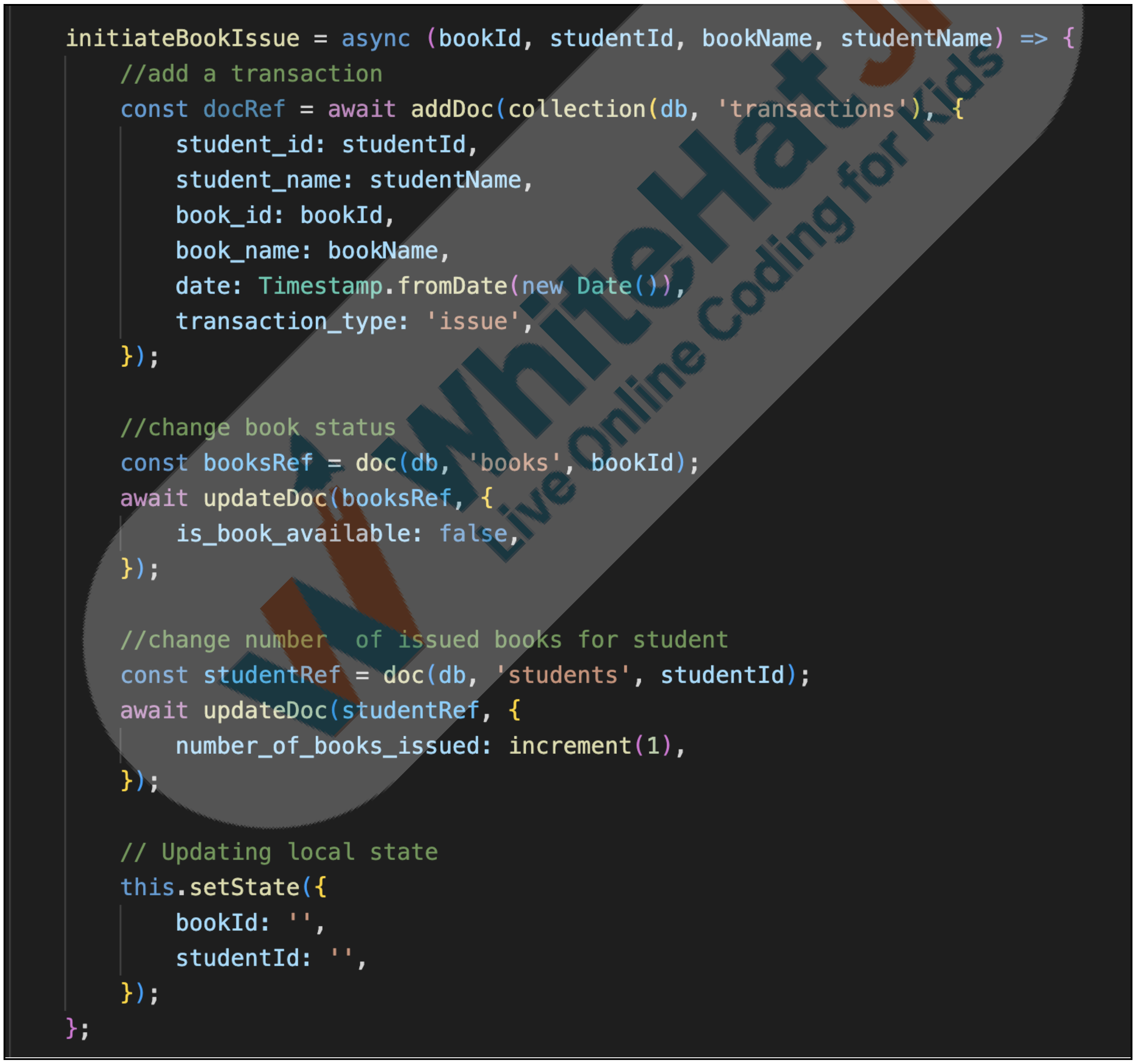
```
getStudentDetails = async (studentId) => {
    studentId = studentId.trim();
    let dbQuery = query(
        collection(db, 'students'),
        where('student_id', '==', studentId)
    );

    let studentRef = await getDocs(dbQuery);

    studentRef.forEach((doc) => {
        this.setState({
            studentName: doc.data().student_details.student_name,
        });
    });
};
```



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

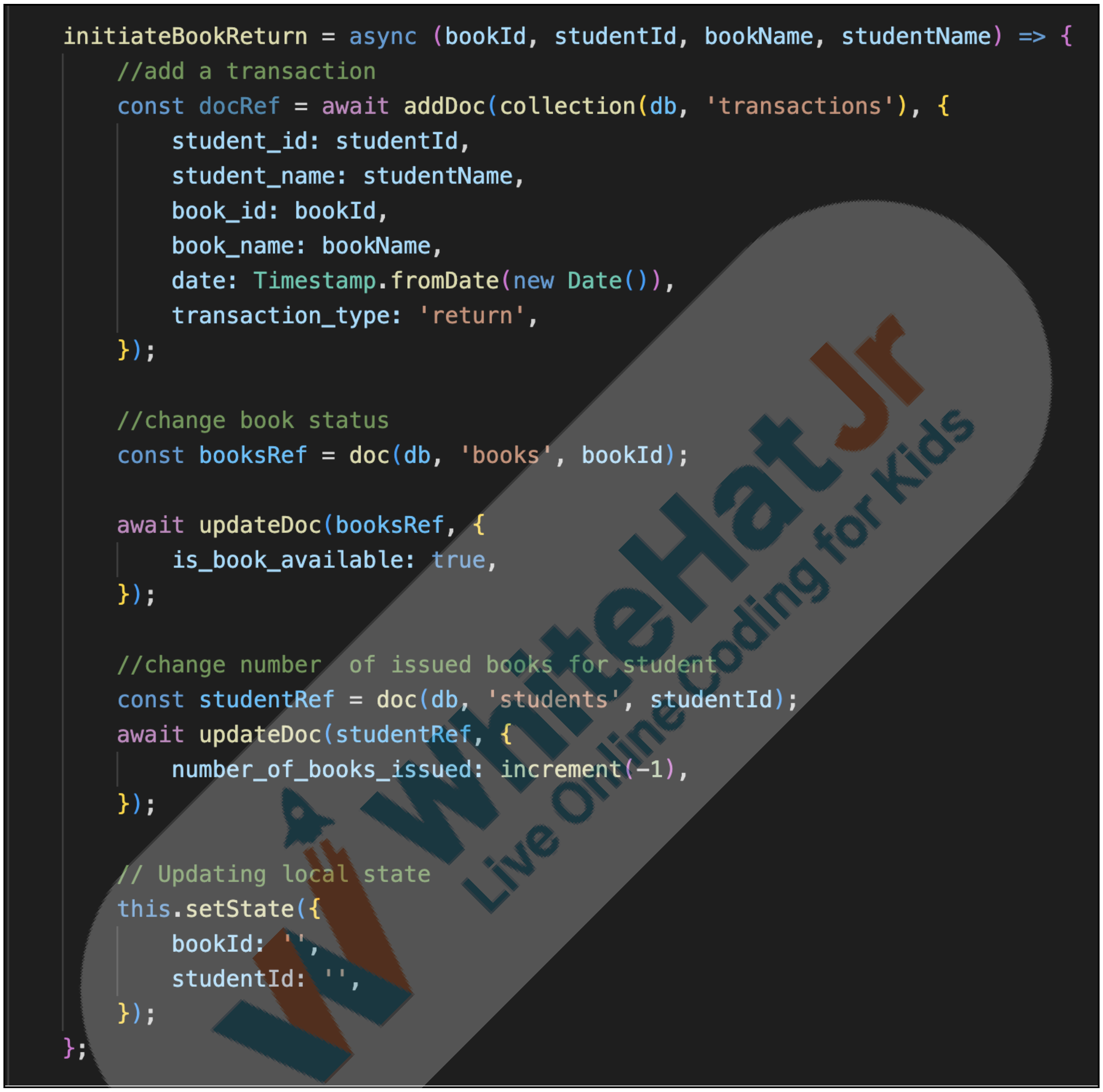
Now we have the data to issue or return the book. Let’s code to write the **initiateBookIssue()** function. This function will help us to issue the book to the student. This function will take four parameters: **StudentId, Student Name, bookId, and bookName.** Can you recall what changes we need to make in the database?	**ESR:** We would:

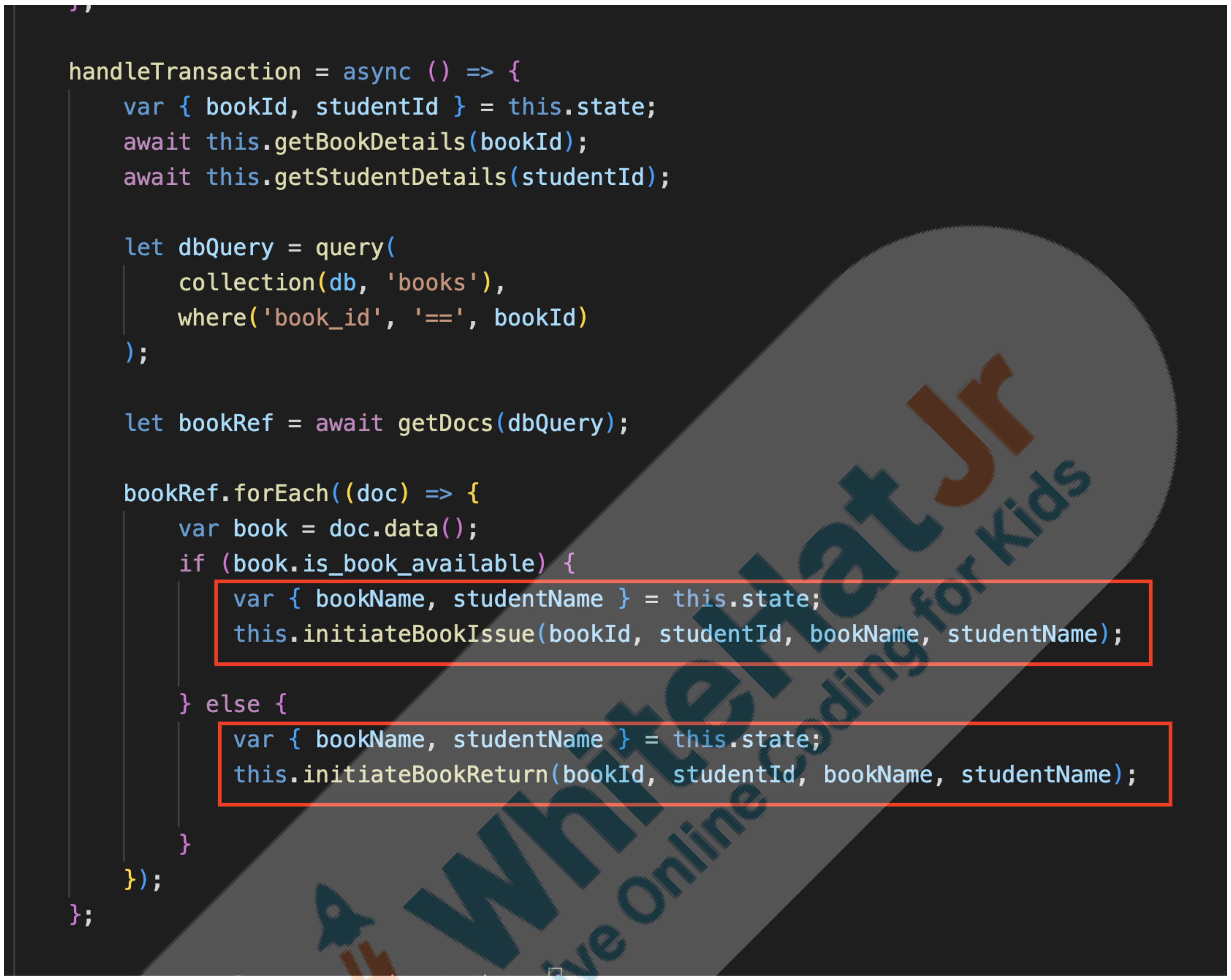
- Update the book status/availability of the book to **false**.
- Update the number of books issued for the student.
- In the local state update the **bookId** and **studentId**.



```javascript
initiateBookIssue = async (bookId, studentId, bookName, studentName) => {
    //add a transaction
    const docRef = await addDoc(collection(db, 'transactions'), {
        student_id: studentId,
        student_name: studentName,
        book_id: bookId,
        book_name: bookName,
        date: Timestamp.fromDate(new Date()),
        transaction_type: 'issue',
    });

    //change book status
    const booksRef = doc(db, 'books', bookId);
    await updateDoc(booksRef, {
        is_book_available: false,
    });

    //change number  of issued books for student
    const studentRef = doc(db, 'students', studentId);
    await updateDoc(studentRef, {
        number_of_books_issued: increment(1),
    });

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

We’ll do the same for the **issueBookReturn()** This function will help us to return the book. Here, the **transaction_type** would be **return**. And, **is_book_available** would be **true**. Change the number of books issued for a student, and update the **bookId** and **studentId**.	


```
initiateBookReturn = async (bookId, studentId, bookName, studentName) => {
    //add a transaction
    const docRef = await addDoc(collection(db, 'transactions'), {
        student_id: studentId,
        student_name: studentName,
        book_id: bookId,
        book_name: bookName,
        date: Timestamp.fromDate(new Date()),
        transaction_type: 'return',
    });

    //change book status
    const booksRef = doc(db, 'books', bookId);

    await updateDoc(booksRef, {
        is_book_available: true,
    });

    //change number  of issued books for student
    const studentRef = doc(db, 'students', studentId);
    await updateDoc(studentRef, {
        number_of_books_issued: increment(-1),
    });

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



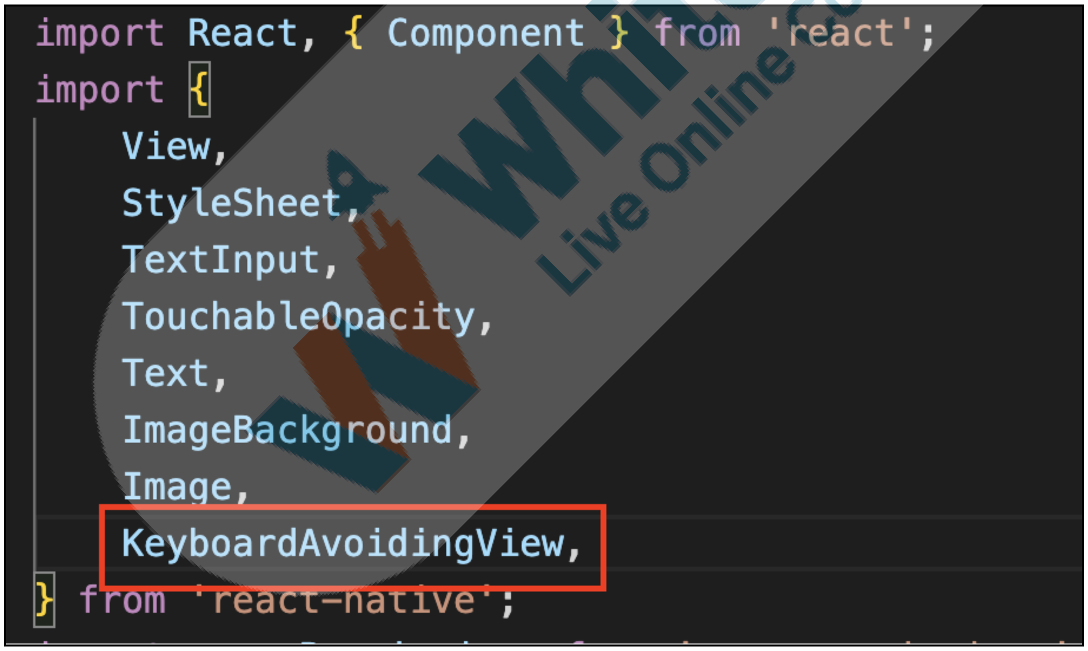
```
handleTransaction = async () => {
    var { bookId, studentId } = this.state;
    await this.getBookDetails(bookId);
    await this.getStudentDetails(studentId);

    let dbQuery = query(
        collection(db, 'books'),
        where('book_id', '==', bookId)
    );

    let bookRef = await getDocs(dbQuery);

    bookRef.forEach((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);

        }
    });
};
```

Awesome! Till now, we have written the functions to get the book and student details. We also wrote functions to issue or return the book to the student.

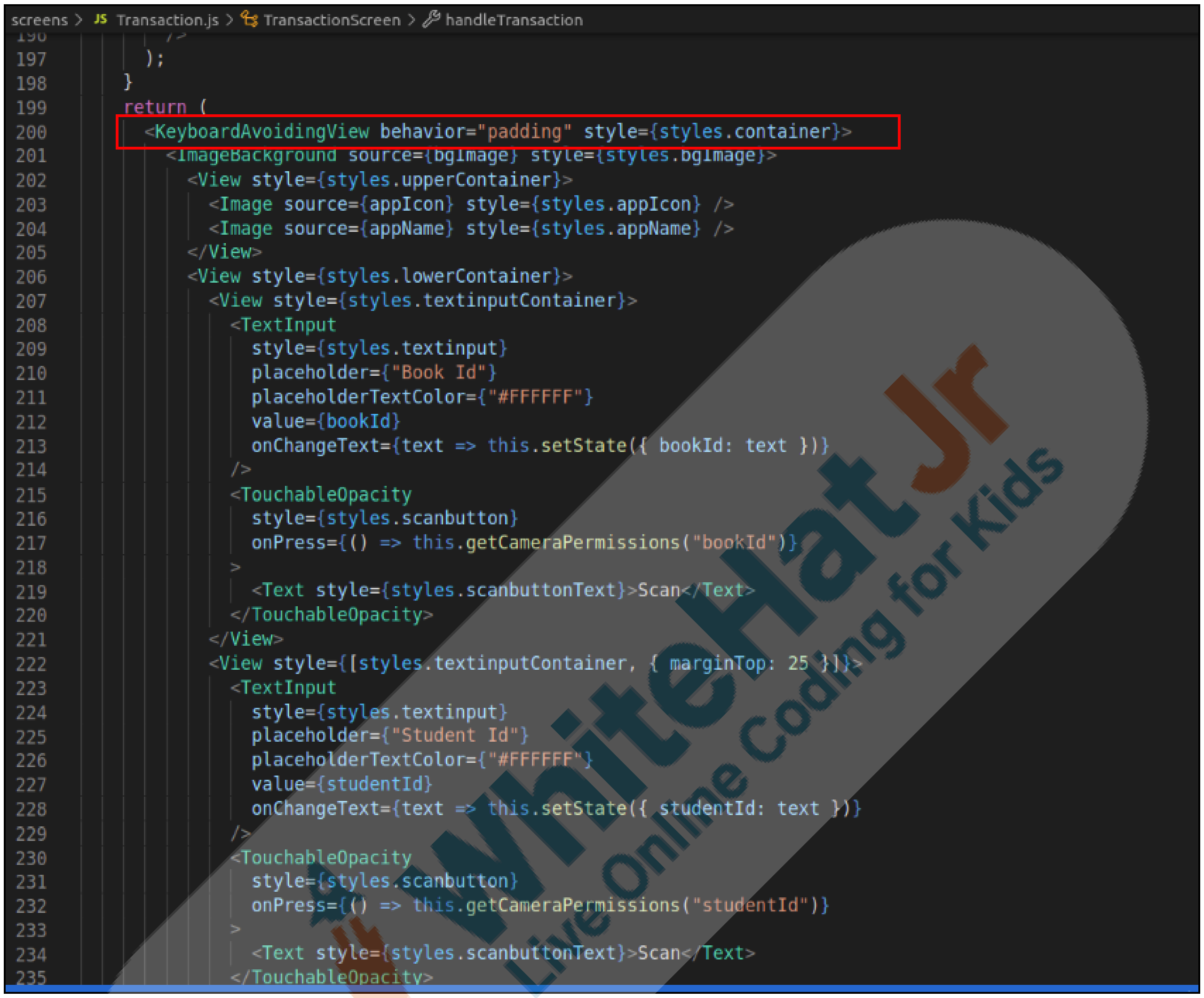
Now, let's start with the problem of avoiding the keyboard overlapping the Text Input boxes.

It turns out there is a very simple fix!

This is a very common problem faced by apps, and React has already thought about it. Instead of enclosing our **Input** form in the **View** component, we can enclose it in

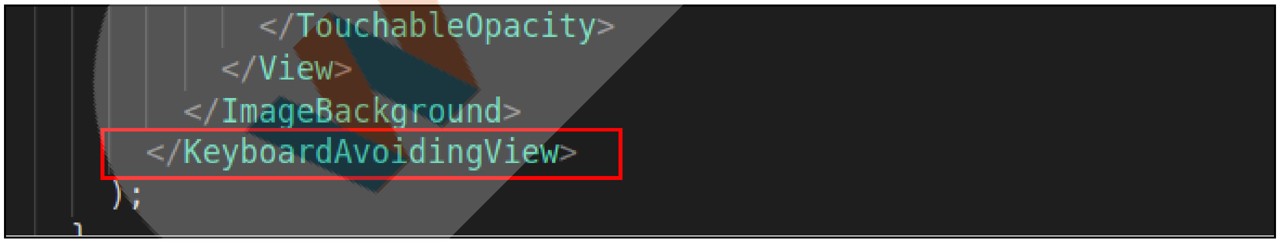
another component, which React Native has called **KeyboardAvoidingView**. The **KeyboardAvoidingView** has a prop called **behavior**. Using this, we can tell the program what to do when there is an overlap. It can either add padding, change the height or position of the enclosing components. Let's quickly look at the documentation for **KeyboardAvoidingView**. *The teacher opens the link from the Teacher Activity 2 and goes through the document along with the student.*	*The student looks at the documentation for* ***KeyboardAvoidingView****.* *Student Activity 2.*
Let us use this in our program. *The teacher takes input from the student to write the code.*	*The student guides the teacher on how to use* ***KeyboardAvoidingView****.*
	

```
import React, { Component } from 'react';
import {
    View,
    StyleSheet,
    TextInput,
    TouchableOpacity,
    Text,
    ImageBackground,
    Image,
    KeyboardAvoidingView,
} from 'react-native';
```



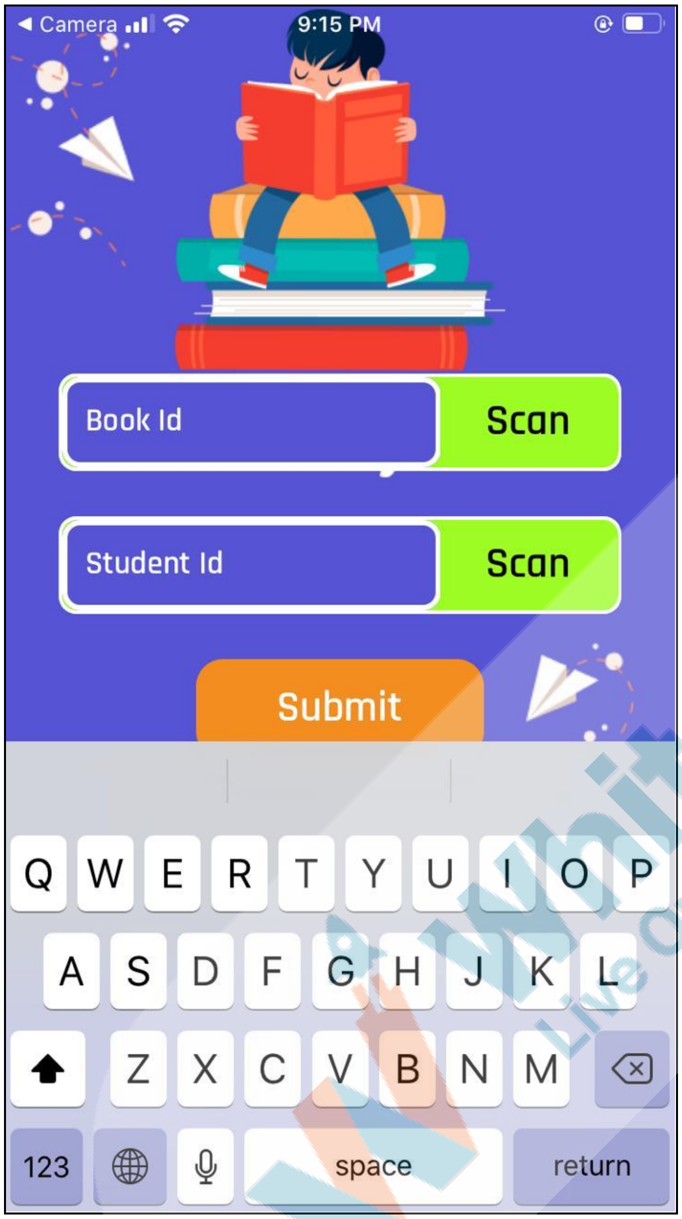
screens > Transaction.js > TransactionScreen > handleTransaction

```
        );
    }
    return (
      <KeyboardAvoidingView behavior="padding" 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}
                onChangeText={text => this.setState({ bookId: text })}
              />
              <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}
                onChangeText={text => this.setState({ studentId: text })}
              />
              <TouchableOpacity
                style={styles.scanbutton}
                onPress={() => this.getCameraPermissions("studentId")}
              >
                <Text style={styles.scanbuttonText}>Scan</Text>
              </TouchableOpacity>
```



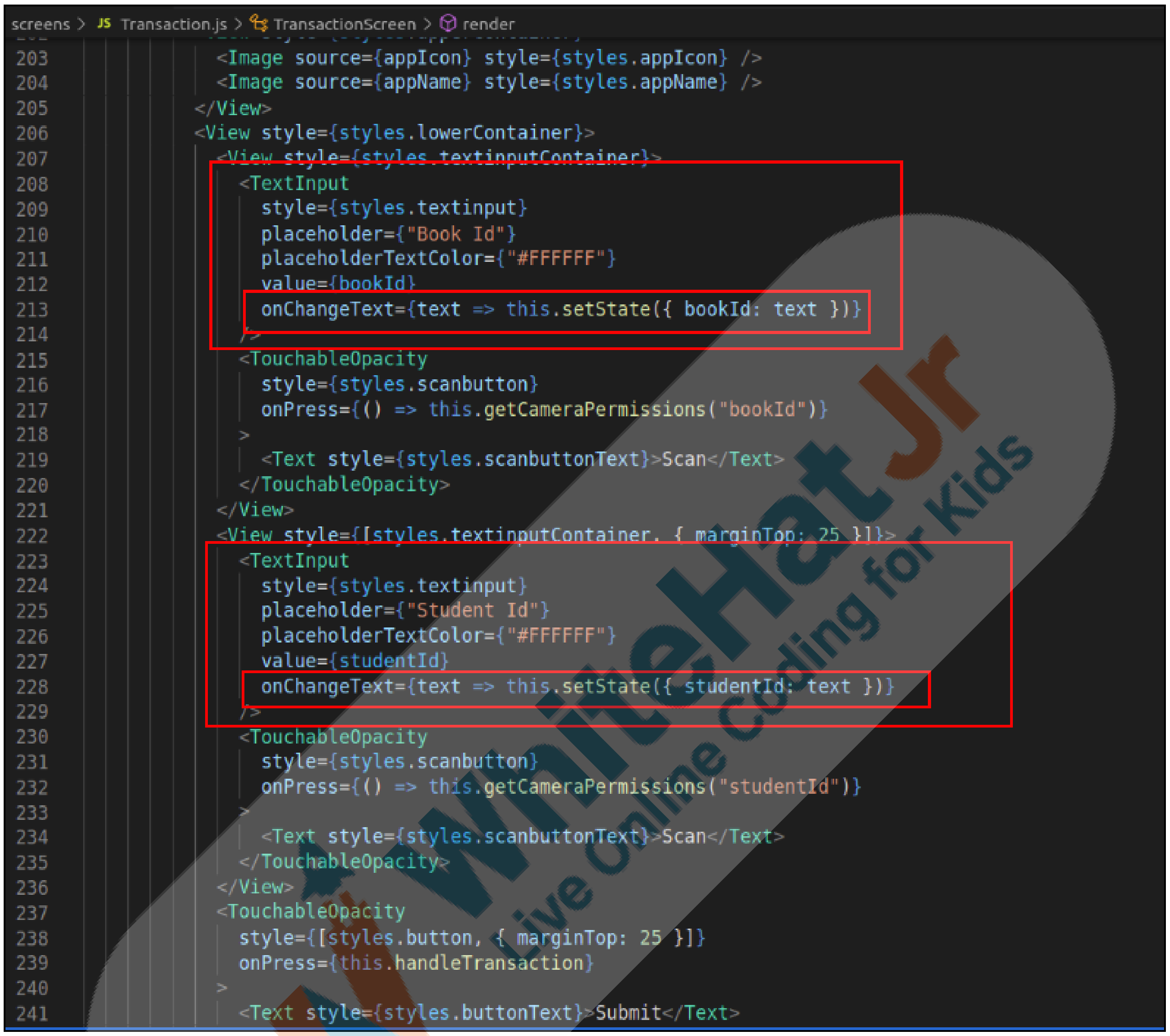
```
              </TouchableOpacity>
            </View>
          </ImageBackground>
        </KeyboardAvoidingView>
      );
```

Let's check our code and see if this works.	*The teacher runs and tests the code.*



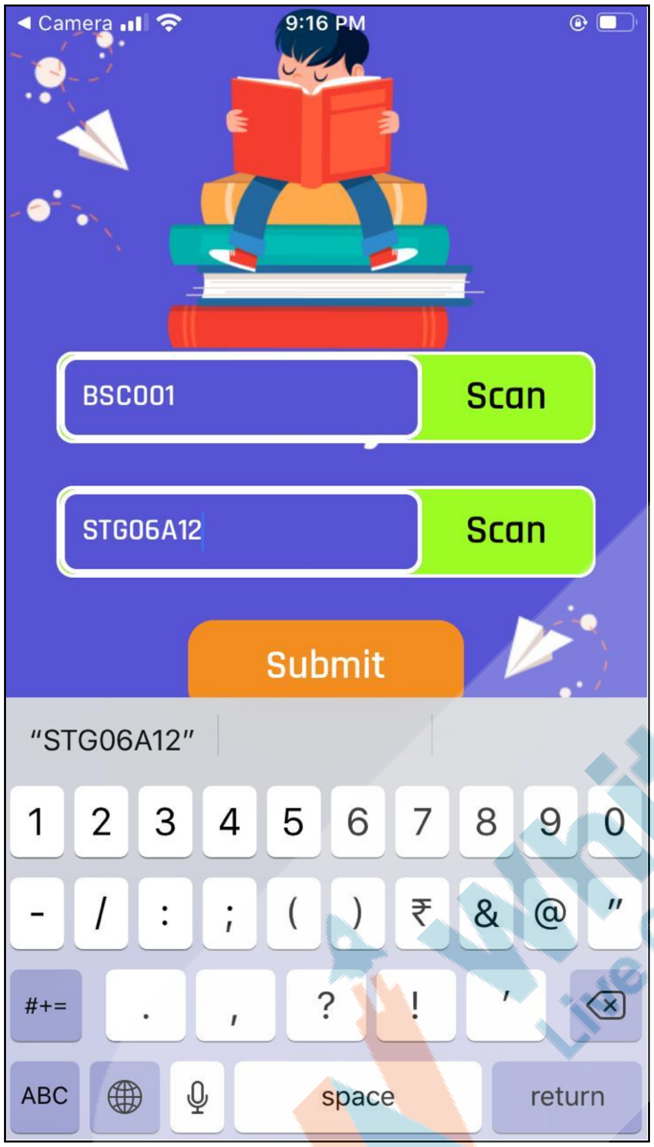
Amazing! We have done it! Now our entire form adds some padding above the keyboard layout so that the text input is always visible. However, as you can see, we are not able to type anything in the text input.	*The student observes the issue of Text Input not being editable.*

You have already learned how to display the text typed by the user in the **TextInput** component using the **onChangeText** prop in the Monkey-Chunky App.	*The student recollects how they use the **onChangeText** prop to display text in **TextInput** to the user.* The *student might also want to go back and look at the final code for the Monkey Chunky App.* *Student Activity 4*
The **onChangeText** prop gets the text typed by the user in **TextInput** as the default argument. We will use the **onChangeText** prop to set the values for **studentId** and **bookId**. *The teacher writes code to show the student how to use the **onChangeText** prop to set the value of **bookId** and **studentId** for the user.*	*The student gives his/her input to the teacher on how to get this done.*



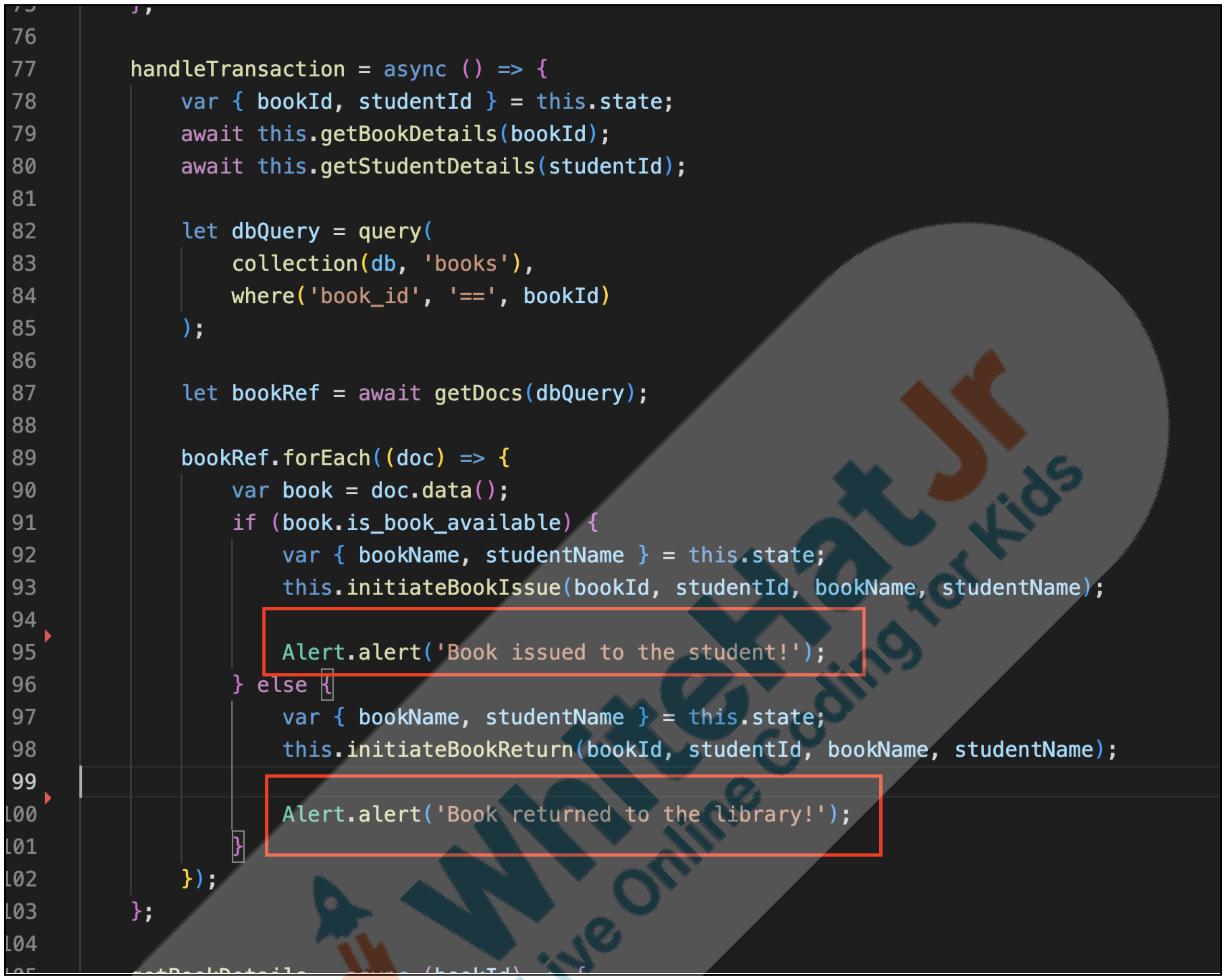
```
          <Image source={appIcon} style={styles.appIcon} />
          <Image source={appName} style={styles.appName} />
        </View>
        <View style={styles.lowerContainer}>
          <View style={styles.textinputContainer}>
            <TextInput
              style={styles.textinput}
              placeholder={"Book Id"}
              placeholderTextColor={"#FFFFFF"}
              value={bookId}
              onChangeText={text => this.setState({ bookId: text })}
            />
            <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}
              onChangeText={text => this.setState({ studentId: text })}
            />
            <TouchableOpacity
              style={styles.scanbutton}
              onPress={() => this.getCameraPermissions("studentId")}
            >
              <Text style={styles.scanbuttonText}>Scan</Text>
            </TouchableOpacity>
          </View>
          <TouchableOpacity
            style={[styles.button, { marginTop: 25 }]}
            onPress={this.handleTransaction}
          >
            <Text style={styles.buttonText}>Submit</Text>
```

Let's check the output and see if the user can now type text in the **TextInput**. *The teacher runs and tests the code to see if the text input box is editable.*	*The student observes the output.*



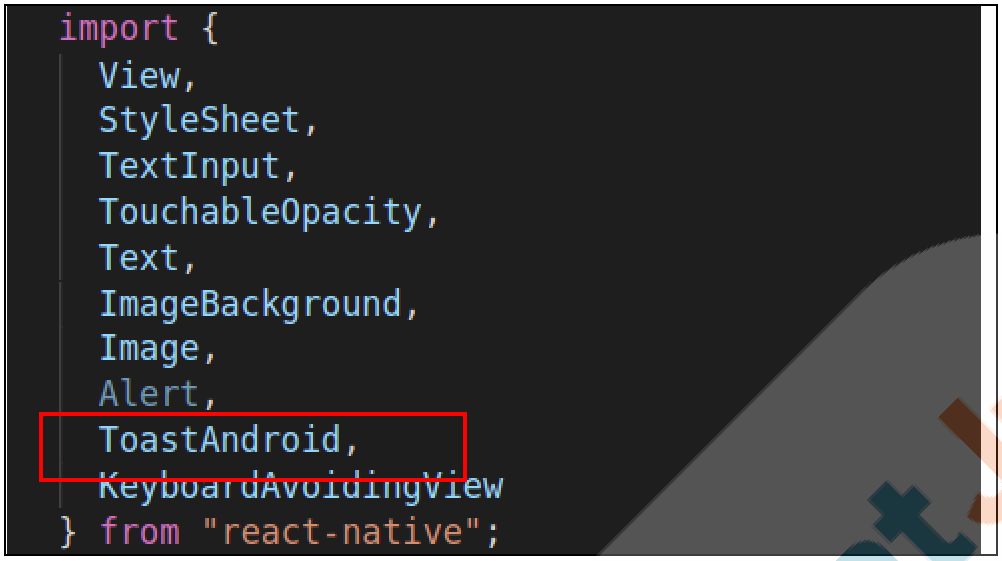
Perfect! Now we want to display a message to the user when a transaction (issue or return) is completed. You already know how to do this using Alerts. While you are doing it, I will also show you a new way of doing it through Toasts.	*The student takes up the challenge.*

Using Toast, you can set a duration for a message to be shown, and then have it disappeared. We use the keyword **SHORT** or **LONG** to set the duration of the Toast. Let's take a look at the documentation to understand better. *The teacher opens the doc from Teacher Activity 4* You might also want to empty the **TextInput** when the transaction is completed so that we are ready for another book transaction. *The teacher imports **ToastAndroid** from React-Native and replaces the Alert with Toast.* ***Note:- ToastAndroid only works for Android users. If you are an iOS user, please stick to using Alert. Both work the same.***	*The student opens the code from Student Activity 3*
If you are using **Alert** to display the message, refer to the following code:	

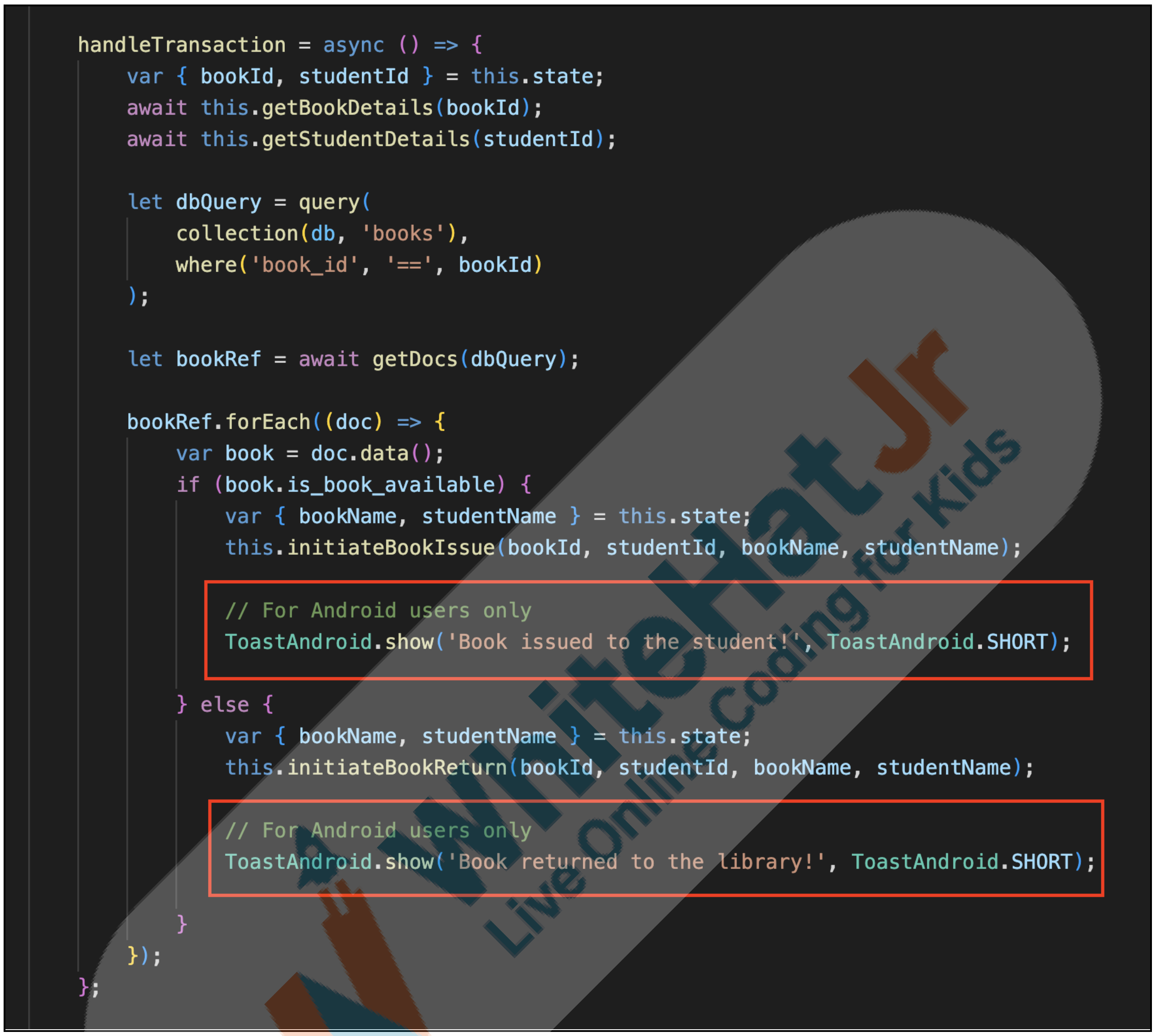


```
    handleTransaction = async () => {
        var { bookId, studentId } = this.state;
        await this.getBookDetails(bookId);
        await this.getStudentDetails(studentId);

        let dbQuery = query(
            collection(db, 'books'),
            where('book_id', '==', bookId)
        );

        let bookRef = await getDocs(dbQuery);

        bookRef.forEach((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!');
            }
        });
    };
```

If you are using **ToastAndroid** to display the message, refer to the following code:

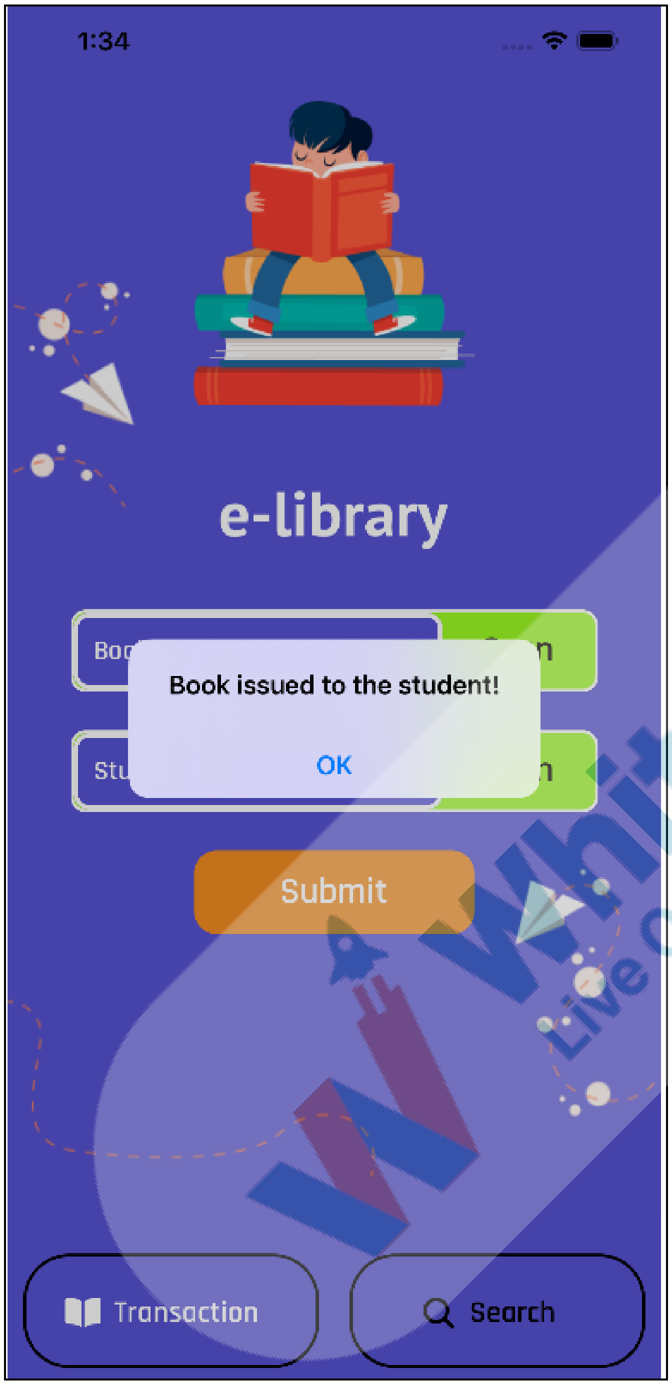


```
import {
  View,
  StyleSheet,
  TextInput,
  TouchableOpacity,
  Text,
  ImageBackground,
  Image,
  Alert,
  ToastAndroid,
  KeyboardAvoidingView
} from "react-native";
```



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

    let dbQuery = query(
        collection(db, 'books'),
        where('book_id', '==', bookId)
    );

    let bookRef = await getDocs(dbQuery);

    bookRef.forEach((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:



Finally, we are able to see the Toast message appearing when the transaction is complete.

Teacher Stops Screen Share		
	Now it's your turn. Please share your	

	screen with me.	

STUDENT-LED ACTIVITY - 20 mins

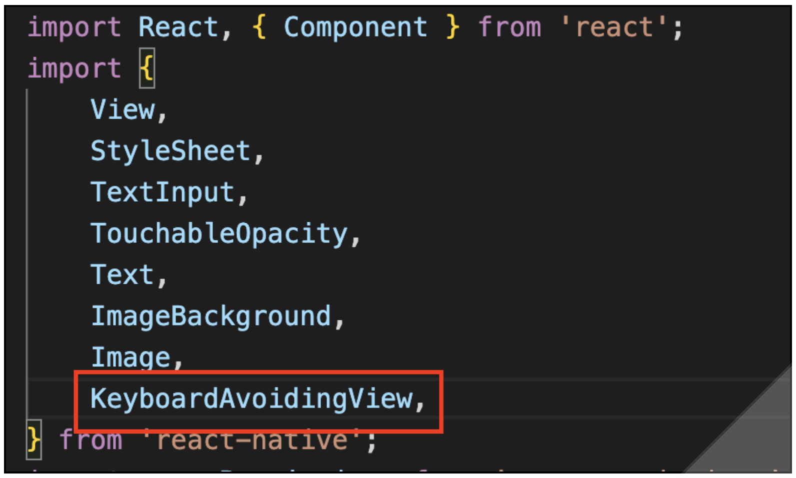
- **Ask Student to press ESC key to come back to panel**
- **Guide Student to start Screen Share**
- **Teacher gets into Fullscreen**

ACTIVITY

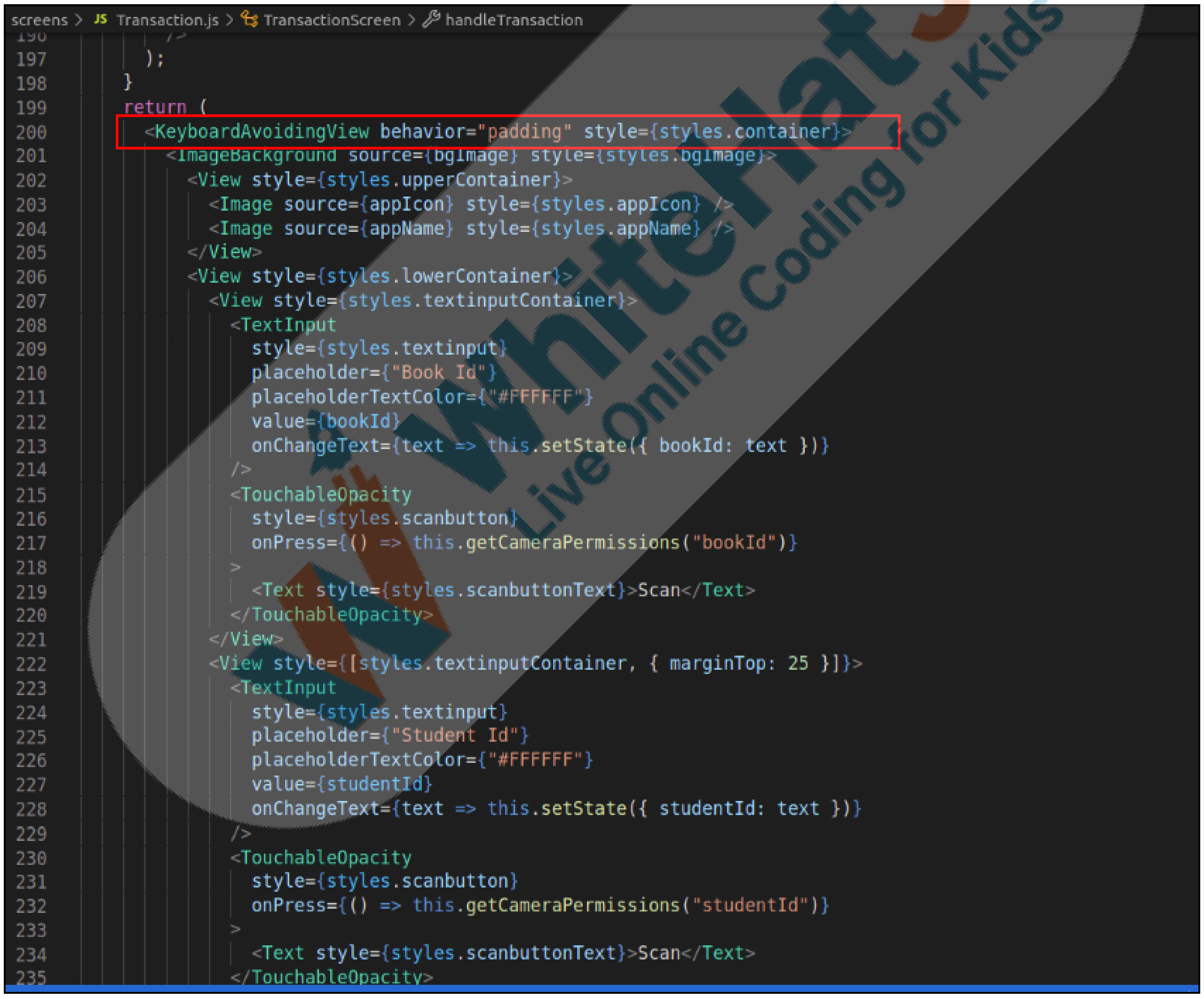
- **Avoiding keyboard overlap with the text box.**
- **Display a transaction message when a transaction is completed.**

Teacher starts slideshow : Slide 15 to 17
Refer to speaker notes and follow the instructions on each slide.

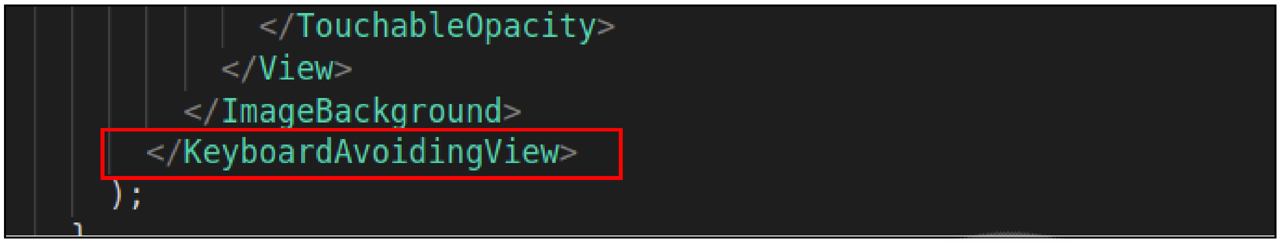
Teacher Action	*Student Action*
Guide the student to download the boilerplate code from Student Activity 1. *Note: The code for **getBookDetails** and **getStudentDetails** have already been added to the code. The student needs to work on fixing the overlapping issue and the Toast alert.*	*The student opens the code from Student Activity 1.*
Use **KeyboardAvoidingView** to avoid the **TextInput** overlap with the keyboard layout.	*The student uses the **KeyboardAvoidingView** to avoid the overlap of keyboard layout with **TextInput**.* *He/She runs the code and tests the app.*



```
import React, { Component } from 'react';
import {
    View,
    StyleSheet,
    TextInput,
    TouchableOpacity,
    Text,
    ImageBackground,
    Image,
    KeyboardAvoidingView,
} from 'react-native';
```

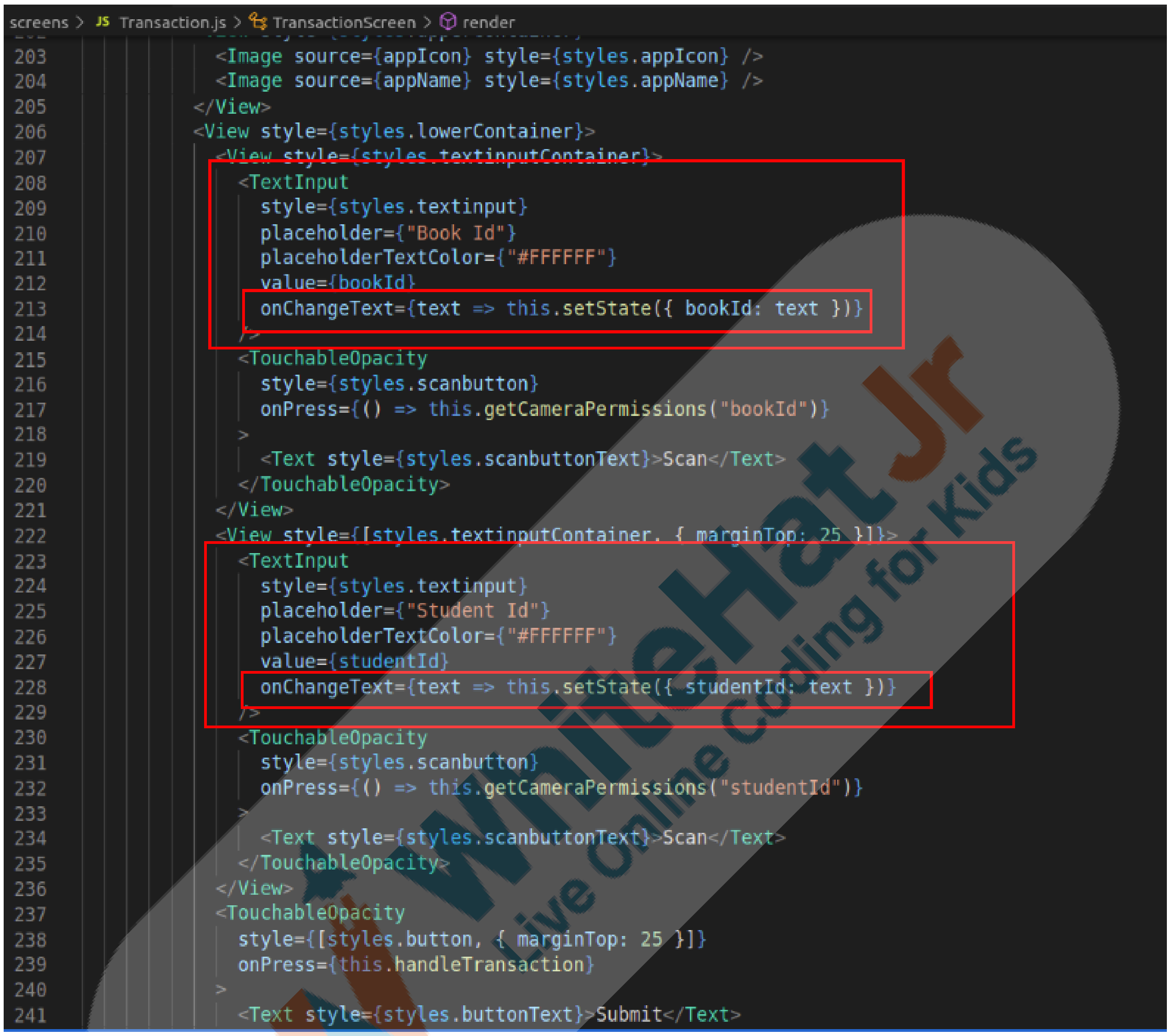

screens > Transaction.js > TransactionScreen > handleTransaction

```
          );
        }
        return (
          <KeyboardAvoidingView behavior="padding" 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}
                    onChangeText={text => this.setState({ bookId: text })}
                  />
                  <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}
                    onChangeText={text => this.setState({ studentId: text })}
                  />
                  <TouchableOpacity
                    style={styles.scanbutton}
                    onPress={() => this.getCameraPermissions("studentId")}
                  >
                    <Text style={styles.scanbuttonText}>Scan</Text>
                  </TouchableOpacity>
```



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

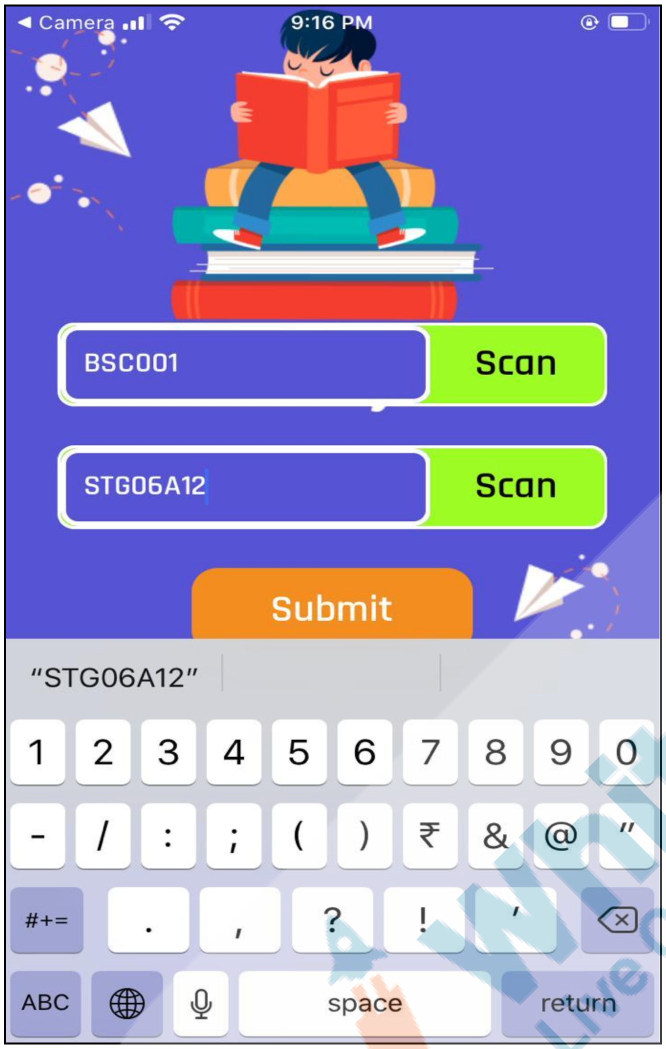
Now use the **onChangeText** prop of **TextInput** to make it editable by the user.	*The student makes the **TextInput** editable using the **onChangeText** Prop.*

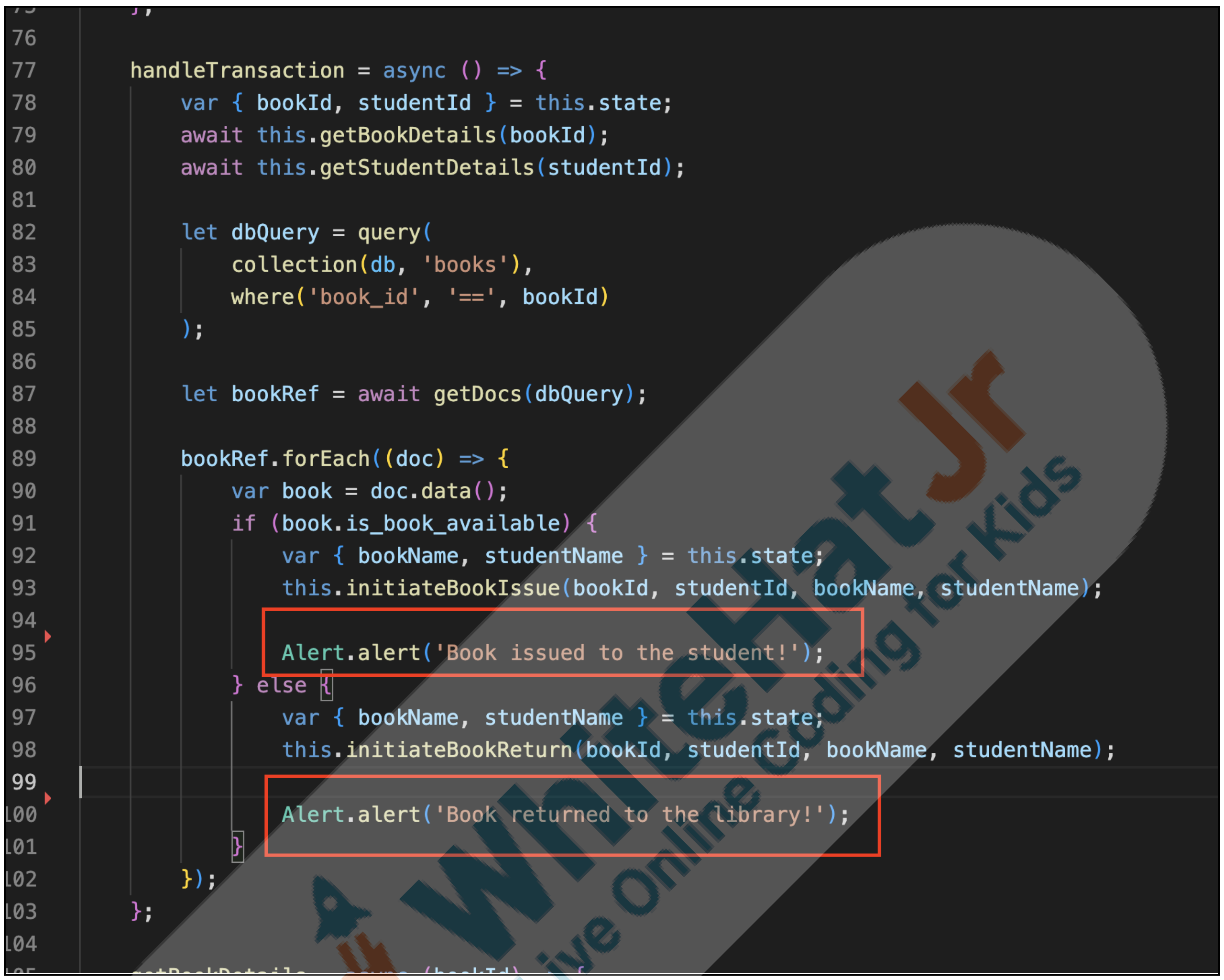


screens > JS Transaction.js > TransactionScreen > render

```
            <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}
                onChangeText={text => this.setState({ bookId: text })}
              />
              <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}
                onChangeText={text => this.setState({ studentId: text })}
              />
              <TouchableOpacity
                style={styles.scanbutton}
                onPress={() => this.getCameraPermissions("studentId")}
              >
                <Text style={styles.scanbuttonText}>Scan</Text>
              </TouchableOpacity>
            </View>
            <TouchableOpacity
              style={[styles.button, { marginTop: 25 }]}
              onPress={this.handleTransaction}
            >
              <Text style={styles.buttonText}>Submit</Text>
```

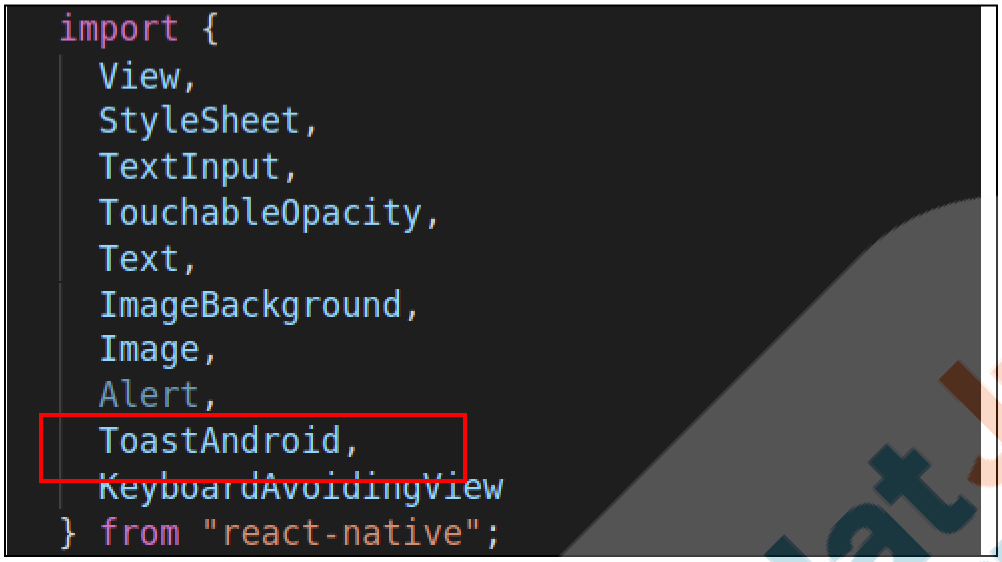
Finally, run and test the app.	*The student runs the code and tests his/her app to check if the **TextInput** is editable.*

 BSC001 Scan STG06A12 Scan Submit	
Use the **Alert** component to display a confirmation message when a transaction is successfully done.	*The student imports Alert and uses the **Alert** component to display an alert on the screen when a book Transaction is completed.*

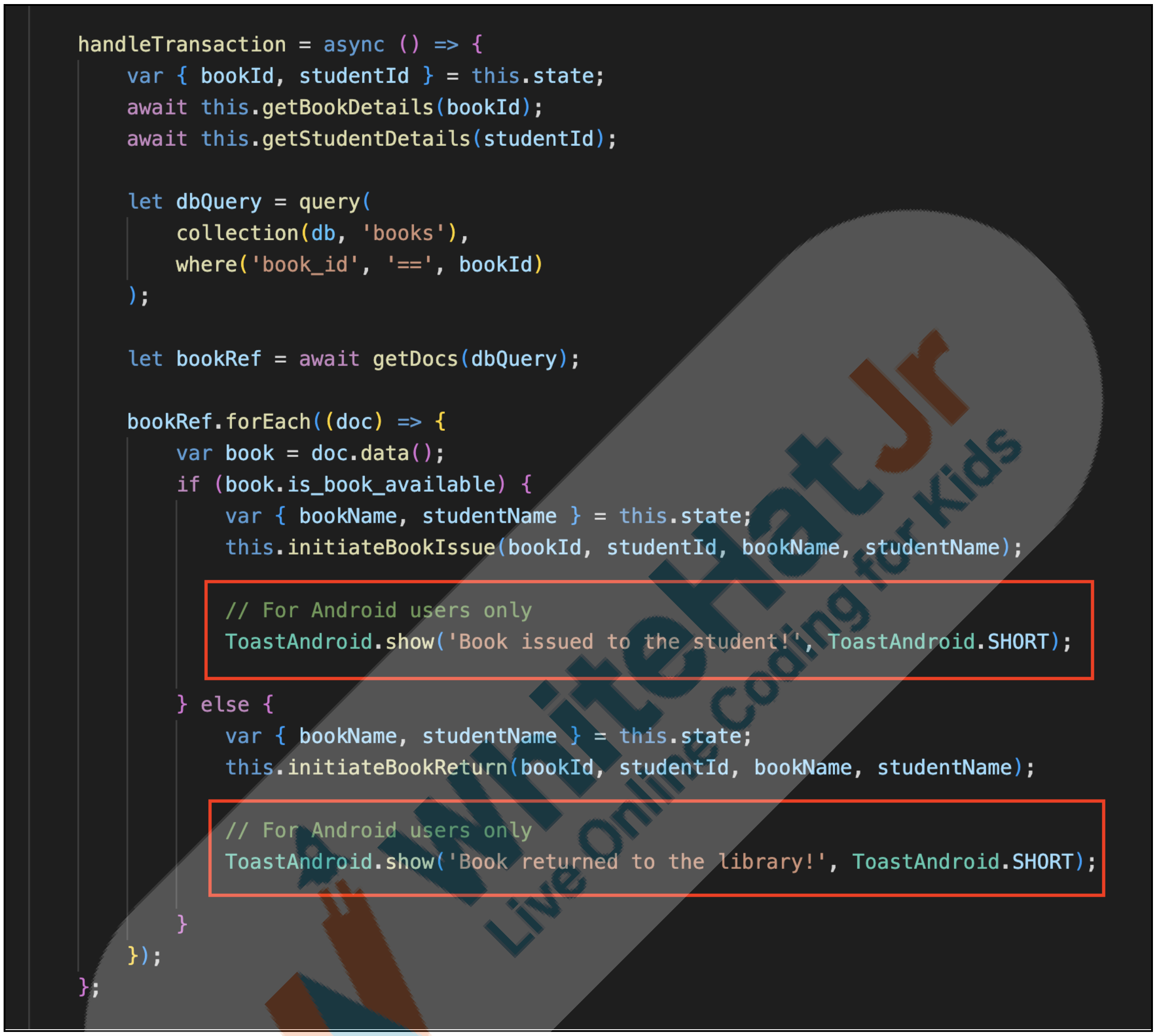


```
    };

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

        let dbQuery = query(
            collection(db, 'books'),
            where('book_id', '==', bookId)
        );

        let bookRef = await getDocs(dbQuery);

        bookRef.forEach((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!');
            }
        });
    };

    getBookDetails = async (bookId) {
```

Similarly, use the **ToastAndroid** Component to display a Toast Message. **Note**:- **ToastAndroid** can only be used for Android users and not for iOS. If the student has iOS, use **Alert** to show the messages.	*The student opens the document from Student Activity 3 to see the usage of **ToastAndroid**.* *The student imports **ToastAndroid** and uses it to display a Toast Message when a book issue or return is completed.*

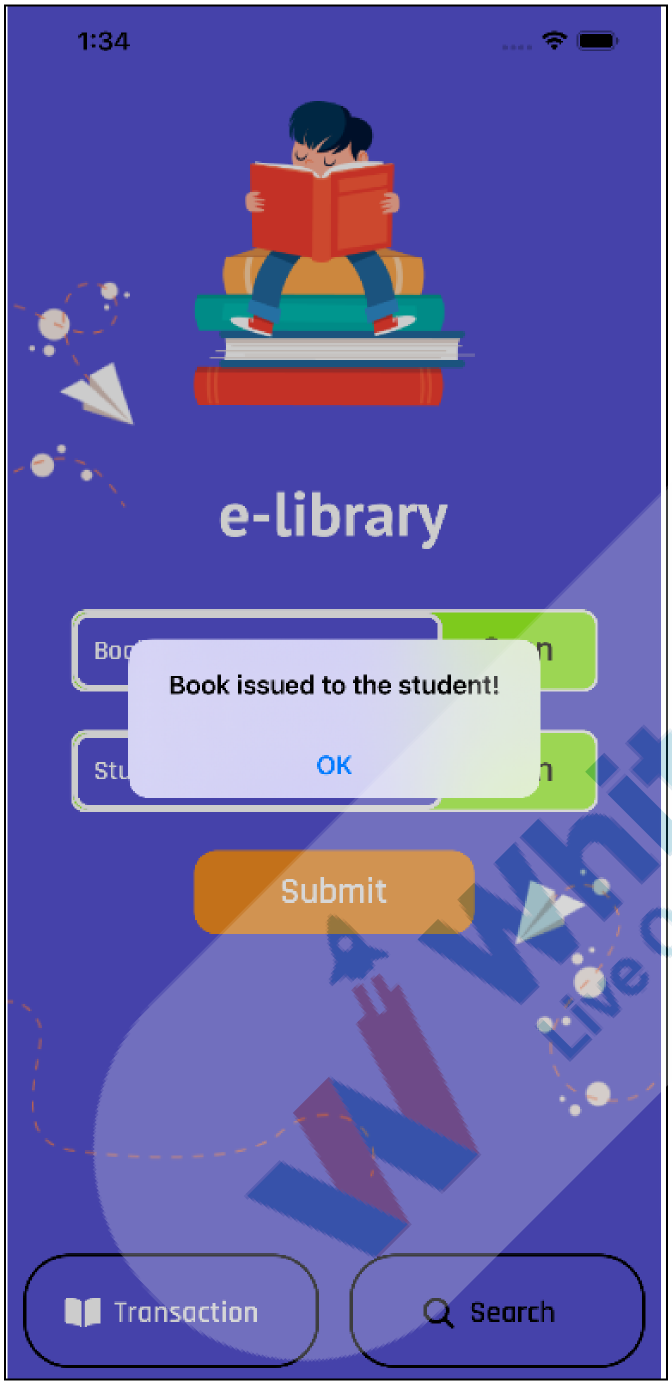


```
import {
  View,
  StyleSheet,
  TextInput,
  TouchableOpacity,
  Text,
  ImageBackground,
  Image,
  Alert,
  ToastAndroid,
  KeyboardAvoidingView
} from "react-native";
```



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

    let dbQuery = query(
        collection(db, 'books'),
        where('book_id', '==', bookId)
    );

    let bookRef = await getDocs(dbQuery);

    bookRef.forEach((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:



Now we can see the toast/Alert saying “**Book issued to the student!**” when the book is issued to the student. Similarly, we’ll see the toast when the book is returned by the student.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 5 Mins	
Teacher starts slideshow from slide 18 to slide 27	
Activity details	**Solution/Guidelines**
Run the presentation from slide 18 to slide 21. **Following are the WRAP-UP session deliverables:** ● **Appreciate the student.** ● **Revise the current class activities.** ● **Discuss the quizzes.**	Discuss with the student the current class activities, and the student will ask doubts related to the activities.
Quiz time - Click on in-class quiz	
Question	**Answer**
How did we solve the problem of the keyboard overlapping the text input boxes? A. Using KeyboardAssistingView component B. Using KeyboardOverlappingView component C. Using KeyboardAvoidingView component D. Using KeyboardRemovingView component	**C**
Which prop gets the text typed by the user in TextInput as the default argument? A. onAccept B. onInput C. onType D. onChange	**D**
Which component is used to display a Toast message? A. ToastiOS component B. ToastAndroid component C. Toast component	**B**

D. AndroidToast component	
Quiz time - End in-class quiz	
FEEDBACK ● **Encourage the student to display transaction messages to the users in different ways - alerts, toasts, and through Text Component on the screen.**	
Teacher Action	**Student Action**
There are so many scenarios that could happen, and our app isn't ready for it. For example: ● What if the student with the given QR code does not exist in the database. ● What if the book wasn't added to the database before the book was scanned? We can't predict how our app will behave in these circumstances since we haven't programmed for it. What are the other scenarios for which we haven't programmed in our application?	**ESR:** ● If the student who is returning the book is not the same as the student who issued the book (we are only checking book availability). ● If the student has issued more than the maximum number of books allowed.
We will program these using firebase in our next class!	

At the end of the next few classes, you can actually implement this app in your own school library!	
You get a “hats off” for your amazing performance today in class. Well Done! See you in the next class.	*Make sure you have given at least 2 Hats Off during the class for:* Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
Project Overview: E-RIDE STAGE 5 **Goal of the Project:** In class 72, you explored scenarios where typing the book ID and student ID would be important. Hence, you changed the text box editable. In this project, we will practice concepts of populating text inputs and the use of ToastAndoid to display alerts in your application. This is a continuation of Project-68, 69, 70 & 71 to make sure you have completed and submitted that before attempting this one. **Story:** The database structure you created in the last project is impressive. Your friend Vihaan is very excited to see	

WhiteHat Jr
Live online Coding for Kids

how you will map the user ID with the bicycle. Also, make changes to the unlock button so that the same user can only get a cycle once he has returned the previous one. 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	
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.*

Links:

Activity	Activity Name	Links

Teacher Activity 1	Boilerplate code	https://github.com/React-Native-Frontier/PRO-C72-E-Library-TA-boilerplate
Teacher Activity 2	KeyboardAvoidingView	https://facebook.github.io/react-native/docs/keyboardavoidingview
Teacher Activity 3	Final Reference code	https://github.com/React-Native-Frontier/PRO-C72-E-Library
Teacher Activity 4	ToastAndroid Documentation	https://facebook.github.io/react-native/docs/toastandroid#__docusaurus
Student Activity 1	Boilerplate code	https://github.com/React-Native-Frontier/PRO-C72-E-Library-SA-boilerplate
Student Activity 2	KeyboardAvoidingView	https://facebook.github.io/react-native/docs/keyboardavoidingview
Student Activity 3	ToastAndroid Documentation	https://facebook.github.io/react-native/docs/toastandroid#__docusaurus
Student Activity 4	Monkey chunky code	https://snack.expo.dev/@procodingclass/monkey-chunky-stage-3:-teacher-reference
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C72-withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/858da844-be77-4de6-a5f5-aa5539cf916e.pdf