

<b>Topic</b>	<b>FIRESTORE QUERIES</b>	
<b>Class Description</b>	The student learns to make queries to a firebase database. The student will check for student eligibility and book eligibility in the app before issuing/returning a book using firebase queries.	
<b>Class</b>	<b>C73</b>	
<b>Class time</b>	<b>40 mins</b>	
<b>Goal</b>	<ul style="list-style-type: none"> <li>• Make a firebase query to check if the book is eligible to be issued/returned.</li> <li>• Make a firebase query to check if the student is eligible for the transaction.</li> <li>• Display a message to the user for a successful/unsuccessful transaction using Toasts.</li> </ul>	
<b>Resources Required</b>	<ul style="list-style-type: none"> <li>• Teacher Resources               <ul style="list-style-type: none"> <li>○ Laptop with internet connectivity</li> <li>○ Earphones with mic</li> <li>○ Notebook and pen</li> <li>○ Android/iOS Smartphone with Expo App installed</li> </ul> </li> <li>• Student Resources               <ul style="list-style-type: none"> <li>○ Laptop with internet connectivity</li> <li>○ Earphones with mic</li> <li>○ Notebook and pen</li> <li>○ Android/iOS Smartphone with Expo App installed</li> </ul> </li> </ul>	
<b>Class structure</b>	<b>Warm-Up</b> <b>Teacher-led Activity</b> <b>Student-led Activity</b> <b>Wrap-Up</b>	<b>5 mins</b> <b>15 min</b> <b>15 min</b> <b>5 min</b>
<b>WARM-UP SESSION - 5 mins</b>		
<p style="text-align: center;"><b><u>CONTEXT</u></b></p> <ul style="list-style-type: none"> <li>• Talk about the different loopholes/bugs in our book issue/return logic.</li> </ul>		



**Teacher starts slideshow from slides 1 to 11**

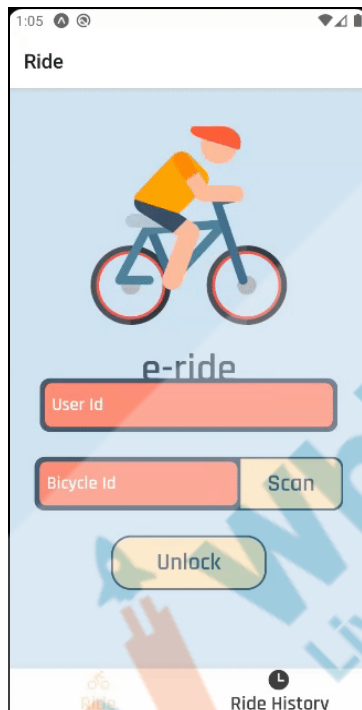
Refer to speaker notes and follow the instructions on each slide.

Activity details	Solution/Guidelines
<p>Hi, how have you been? Are you excited to learn something new?</p> <p><b>Run the presentation from slide 1 to slide 4.</b></p> <p><b>The following are the warm-up session deliverables:</b></p> <ul style="list-style-type: none"> <li>• Reconnect with previous class topics.</li> <li>• Warm-Up quiz session.</li> </ul>	<p><b>ESR:</b> 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 under the getUserDetails() function, to set the state with the values read from the firestore collection.</p> <div data-bbox="203 1260 852 1827"> <p>A. <pre>this.setState(   userName: doc.data().name,   userId: doc.data().id,   bikeAssigned: doc.data().bike_assigned );</pre></p> <p>B. <pre>this.setState(   userName: doc.data().name,   userId: doc.data().id,   bikeAssigned: doc.data().bike_assigned );</pre></p> <p>C. <pre>this.setState({   userName: doc.data().name   userId: doc.data().id   bikeAssigned: doc.data().bike_assigned });</pre></p> </div>	<p><b>D</b></p>

```
this.setState({
  userName: doc.data().name,
  userId: doc.data().id,
  bikeAssigned: doc.data().bike_assigned
});
```

D.

Choose the correct block of code under checkBikeAvailability() function to set transaction Type to “rented”, if bike is available, else set it to “return”.



```
/*transactionType = doc.data().is_bike_available ?
"rented" : "return";*/
```

A.

```
/*transactionType = doc.data().is_bike_available :
"rented" ? "return";*/
```

B.

```
/*transactionType === doc.data().is_bike_available ?
"rented" : "return";*/
```


C.

```
/*transactionType = doc.data().is_bike_available
"rented" "return";*/
```

D.

A

**Continue the warm-up session**

Activity details	Solution/Guidelines
<p><b>Run the presentation from slide 4 to slide 11 to set the problem statement.</b></p> <p><b>The following are the warm-up session deliverables:</b></p> <ul style="list-style-type: none"> <li>Discuss the different loopholes/bugs in our book issue/return logic.</li> </ul>	<p>Narrate the slides by using hand gestures and voice modulation methods to bring in more student interest.</p>
<p>Teacher ends slideshow </p>	
TEACHER-LED ACTIVITY - 15 mins	
Teacher Initiates Screen Share	
<p><b><u>CHALLENGE</u></b></p> <ul style="list-style-type: none"> <li>Make a firebase query to check if the book and student ID exist in the database.</li> <li>Write pseudocode and abstract functions to check for book and student eligibility.</li> </ul>	
Teacher Activity	Student Activity
<p><i>The teacher clones the boilerplate code from <a href="#">Teacher Activity 1</a> and installs all the project dependencies.</i></p> <p><b>Steps to clone the project:-</b></p> <pre>git clone &lt;projectURL&gt; cd &lt;projectFolder&gt; npm install</pre>	
<p>As usual, let's start by reviewing the code we have written so far.</p>	<p><i>The student reviews the code from the previous class.</i></p>

<p><b>*Note:</b> Focus on the <b>handleTransaction()</b> function and when the function is called.</p>	
<p>In our <b>handleTransaction()</b> function, we are currently issuing or returning the book based only on the <b>book availability status</b>.</p> <p>But as we discussed, what if the book doesn't exist in the database? Our app might not know what to do.</p> <p>Let's focus on our <b>handleTransaction()</b> function to check for book availability first. Is the book eligible for the transaction?</p> <p>We will also check for <b>studentEligibility</b> to issue and return a book.</p>	
<p><b>Note-</b> The <b>handleTransaction()</b> function code is already provided, the teacher needs to explain the code to the student.</p> <p>Do you recall the issues that we might face in the app, which we discussed?</p> <p>Yes! So let's pick these issues one by one and solve them.</p> <p>Let's first check for Book availability using the <b>checkBookAvailability()</b> function.</p>	<p><i>The student listens and asks questions.</i></p> <p><b>ESR:</b> The issue that we face might be:-</p> <ul style="list-style-type: none"> <li>• The book we are looking for is not available.</li> <li>• The studentId or the bookId doesn't exist in the DB.</li> <li>• Student can issue many books.</li> </ul>

As the name suggests, this function will check for the availability of the book in the database.

How can we do that?

Yes. We'll write a query on the books' collection to check if any of the **bookId** matches our given **bookId**.

If the query doesn't return anything, then we'll return the transaction type as **false**.

Otherwise, using the **ternary** operator, we'll check the **is\_book\_available** flag, to return **issue** if the book is present in the library or **return** if the book is not available in the library.

And then return the transaction type.

*The teacher writes the code.*

**ESR:**

We can query the books collections to check for the book.

```
checkBookAvailability = async bookId => {  
  const bookRef = await db  
    .collection("books")  
    .where("book_id", "==", bookId)  
    .get();  
  
  var transactionType = "";  
  if (bookRef.docs.length == 0) {  
    transactionType = false;  
  } else {  
    bookRef.docs.map(doc => {  
      //if the book is available then transaction type will be issue  
      // otherwise it will be return  
      transactionType = doc.data().is_book_available ? "issue" : "return";  
    });  
  }  
  
  return transactionType;  
};
```

Let's consider here that this function returns **false** if the book is not in the database, then using an alert we'll show a message that the book is not available and set **bookId** and **studentId** as empty in the state and show the alert which mentions the book doesn't exist in the library.

If the transaction type is **Issue** then we check if the student is eligible for issuing the book.

If the student is eligible, then we call the **initiateBookIssue()** function and give an alert that the book is issued to the student.

Similarly, if the transaction type is **return**, then we check if the student is eligible for returning the book.

If the student is eligible, then we call the **initiateBookReturn()** function and give an alert that the book has been returned.

*The teacher writes the code.*

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

  var transactionType = await this.checkBookAvailability(bookId);

  if (!transactionType) {
    this.setState({ bookId: "", studentId: "" });
    // For Android users only
    // ToastAndroid.show("The book doesn't exist in the library database!", ToastAndroid.SHORT);
    Alert.alert("The book doesn't exist in the library database!");
  } else if (transactionType === "issue") {

    var { bookName, studentName } = this.state;
    this.initiateBookIssue(bookId, studentId, bookName, studentName);

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

If the transaction type is "issue", let's check if the student is eligible for the book issue.

To be eligible for book issue, the student should exist in the database and the number of books issued by them **should be less than two**, we will be doing this ahead.

Let's assume that **checkStudentEligibilityForBookIssue()** does this. It returns **true** if the student is eligible and **false** if the student is not eligible.

If the student is eligible, let's call our function **initiateBookIssue()** and issue an alert that the book has been issued.



We can also **empty** the **TextInputs** after the books have been issued.

Awesome now can you guide me on what to do if the transaction type is **'return'**?

Awesome! Let's do that.

Let's use the **checkStudentEligibilityForReturn()** as an abstract function that returns true if the student is eligible to return and false if the student is not eligible.

If the student is eligible, let's call a function that returns the book.

Also, we can clear the **TextInput** boxes as we return the book.

**ESR:**

If the transaction type is **'return'**, we can call a function **checkStudentEligibilityForBookReturn()** which checks if the student is eligible for book returns.

```

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

  var transactionType = await this.checkBookAvailability(bookId);

  if (!transactionType) {
    this.setState({ bookId: "", studentId: "" });
    // For Android users only
    // ToastAndroid.show("The book doesn't exist in the library database!", ToastAndroid.SHORT);
    Alert.alert("The book doesn't exist in the library database!");
  } else if (transactionType === "issue") {
    var isEligible = await this.checkStudentEligibilityForBookIssue(
      studentId
    );

    if (isEligible) {
      var { bookName, studentName } = this.state;
      this.initiateBookIssue(bookId, studentId, bookName, studentName);
    }
    // For Android users only
    // ToastAndroid.show("Book issued to the student!", ToastAndroid.SHORT);
    Alert.alert("Book issued to the student!");
  } else {
    var isEligible = await this.checkStudentEligibilityForBookReturn(
      bookId,
      studentId
    );

    if (isEligible) {
      var { bookName, studentName } = this.state;
      this.initiateBookReturn(bookId, studentId, bookName, studentName);
    }
    // For Android users only
    // ToastAndroid.show("Book returned to the library!", ToastAndroid.SHORT);
    Alert.alert("Book returned to the library!");
  }
};


```

Awesome! We almost have the **handleTransaction()** function ready.

*The student observes and asks for any questions.*

We just need to detail the abstract functions which we have used.

Awesome! Now that you know how to make a firebase query, can you add detail to these abstract functions -

checkStudentEligibilityForBookIssue() and checkStudentEligibilityForReturn()?		ESR: Yes!
<b>Teacher Stops Screen Share</b>		
	Now it's your turn. Please share your screen with me.	
<b>STUDENT-LED ACTIVITY - 15 mins</b>		
<ul style="list-style-type: none"> <li>• Ask Student to press ESC key to come back to panel</li> <li>• Guide Student to start Screen Share</li> <li>• Teacher gets into Fullscreen</li> </ul>		
<p align="center"><b><u>ACTIVITY</u></b></p> <ul style="list-style-type: none"> <li>• Write code to check for book and student eligibility before completing a book transaction (issue/return).</li> <li>• Display a book transaction message to the user for both successful and unsuccessful transactions.</li> </ul>		
<p align="center">  <b>Teacher starts slideshow : Slide 12 to 14</b>        Refer to speaker notes and follow the instructions on each slide.     </p>		
<b>Teacher Action</b>		<b>Student Action</b>
<i>Guide the student to open the project from <a href="#">Student Activity 1</a>.</i>		<i>The student opens the code from <a href="#">Student Activity 1</a>.</i>
<p>The <b>handleTransactions()</b> function is almost ready. We just have to detail the abstract functions that we have used.</p> <p>Now you need to check if the student is eligible to issue or return a book.</p>		

<p>What do we need to do in the function <b>checkStudentEligibilityForBookIssue()</b>?</p>	<p><b>ESR:</b></p> <p>We need to check if the student ID exists in the database</p> <ul style="list-style-type: none"> <li>• If it exists, we need to check if the student has issued more than two books.</li> <li>• If not, we return true.</li> <li>• Else, we return false.</li> </ul>
<p><i>Help the student query the student collection to check if the student with the student ID exists.</i></p> <p><i>The teacher can refer to the documentation on how to write queries from <a href="#">Teacher activity 3</a> for reference.</i></p>	<p><i>The student can refer to the document from <a href="#">Student Activity 2</a>.</i></p> <p><i>The student writes the code to check for <b>checkStudentEligibilityForBookIssue()</b>.</i></p>

```
checkStudentEligibilityForBookIssue = async studentId => {
  const studentRef = await db
    .collection("students")
    .where("student_id", "==", studentId)
    .get();

  var isStudentEligible = "";
  if (studentRef.docs.length == 0) {
    this.setState({
      bookId: "",
      studentId: ""
    });
    isStudentEligible = false;
    Alert.alert("The student id doesn't exist in the database!");
  } else {
    studentRef.docs.map(doc => {
      if (doc.data().number_of_books_issued < 2) {
        isStudentEligible = true;
      } else {
        isStudentEligible = false;
        Alert.alert("The student has already issued 2 books!");
        this.setState({
          bookId: "",
          studentId: ""
        });
      }
    });
  }

  return isStudentEligible;
};
```

What do we need to do in this function  
**checkStudentEligibilityForReturn()**?

**ESR:**

We need to query the last transaction for the book.

We need to check if the last book transaction was performed by the same student.

**Note:** The **checkStudentEligibilityForReturn()** function

code is already provided, the teacher needs to explain the code to the student as mentioned below.

In this function :

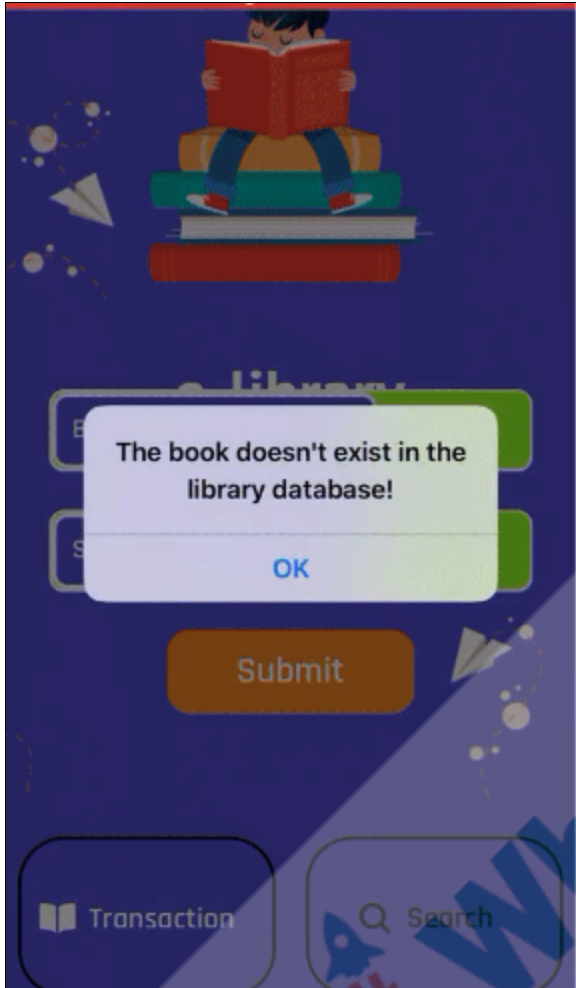
- 1) We'll make a query on transactions collection to get the last transaction of the book with the given id using **limit()**.
- 2) Using the data that we got from the earlier query, we'll check if the **studentId** matches our **studentId**.
- 3) If the **studentId** matches, then the student is eligible, else the student is not eligible and displays the message that the student is not eligible.
- 4) Finally, empty the text boxes.

```
checkStudentEligibilityForBookReturn = async (bookId, studentId) => {
  const transactionRef = await db
    .collection("transactions")
    .where("book_id", "==", bookId)
    .limit(1)
    .get();
  var isStudentEligible = "";
  transactionRef.docs.map(doc => {
    var lastBookTransaction = doc.data();
    if (lastBookTransaction.student_id === studentId) {
      isStudentEligible = true;
    } else {
      isStudentEligible = false;
      Alert.alert("The book wasn't issued by this student!");
      this.setState({
        bookId: "",
        studentId: ""
      });
    }
  });
  return isStudentEligible;
};
```

*Help the student test and check for typos.*

*The student runs the code on their phone to check the output.*

**Output:**



Awesome job today. So today we wrote the code to check if the student is eligible to issue or return the book. The code that we have written today is the backend of the app. So the only output you can see is the messages that are shown in toast or alert.

**Teacher Guides Student to Stop Screen Share**

**WRAP-UP SESSION - 5 Mins**

**Teacher starts slideshow**



**from slide 15 to slide 25**

**Activity details**

**Solution/Guidelines**

***Run the presentation from slide 15 to slide 25.***

Discuss with the student the

<b>Following are the WRAP-UP session deliverables:</b> <ul style="list-style-type: none"> <li>• <b>Appreciate the student.</b></li> <li>• <b>Revise the current class activities.</b></li> <li>• <b>Discuss the quizzes.</b></li> </ul>	current class activities and the student will ask doubts related to the activities.
<b>Quiz time - Click on in-class quiz</b>	
<b>Question</b>	<b>Answer</b>
In a database, a ____ is used to give us the exact document which we need.  A. Component B. Node C. Application D. Query	<b>D</b>
Which of the following functions do we use to specify the number of records to fetch from the database?  A. .limit() B. .get() C. .where() D. .which()	<b>A</b>
The query will return a list of documents in which format?  A. in an array. B. as different variables. C. as a separate documents. D. in a loop..	<b>A</b>
<b>Quiz time - End in-class quiz</b>	
<b>Teacher Action</b>	<b>Student Action</b>
Now that we have almost fully built the code for the book issue/return functionality in our app, we need to write code for the search functionality.	



What do you think search functionality does?	<b>ESR:</b> Search functionality is for the teacher (user) to search the transaction collection by book or username.
Correct! And we already know how to query the transactions' collection to do that.  In the next class, we will build the search functionality in our app and learn how to list the results in our app.	
<p style="text-align: center;"><b><u>FEEDBACK</u></b></p> <ul style="list-style-type: none"> <li>Encourage them to read more about the expo-barcode-scanner package from the documentation.</li> </ul>	
Teacher Action	Student Action
You get a “hats off” for your amazing performance today!	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div> <div>Creatively Solved Activities</div> <div>+10</div> </div> <div> <div>Great Question</div> <div>+10</div> </div> <div> <div>Strong Concentration</div> <div>+10</div> </div>
<p><b>* This Project will take only 30 mins to complete. Motivate students to try and finish it immediately after the class.</b></p> <p><b>Project Overview</b></p> <p><b>E-RIDE STAGE 6</b></p>	

### Goal of the Project:

In class 73, you modified the code to check for student eligibility and book eligibility before issuing/returning a book using firebase queries. In this project, you will apply the same concept to check user eligibility and bike eligibility before assigning and returning the bike.

\* This is a continuation of Project-68, 69, 70, 71, & 72 to make sure you have completed and submitted that before attempting this one.

### Story:

Your friend Vihaan has come across a situation where the same user is unlocking multiple bikes at the same time. He also noticed a case where the user who returned the bike was different from the one who was assigned that bikeID.

You need to modify the app code to refrain users from doing it. Vihaan thus wants to update the app to run certain checks on users as well as bikes.

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

Bye Bye!

Teacher ends slideshow



Teacher Clicks

✕ End Class

### ADDITIONAL ACTIVITIES

#### Additional Activities

*The student uses the Markdown editor to write*

*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?

*her/his reflection as a reflection journal.*

**Links:**

Activity	Activity Name	Links
Teacher Activity 1	Boilerplate code	<a href="https://github.com/pro-whitehatjr/e-library-c73-boiler-plate">https://github.com/pro-whitehatjr/e-library-c73-boiler-plate</a>
Teacher Activity 2	Reference Class code	<a href="https://github.com/whitehatjr/e-library-v2-PRO-C73">https://github.com/whitehatjr/e-library-v2-PRO-C73</a>
Teacher Activity 3	Documentation for query	<a href="https://firebase.google.com/docs/firestore/query-data/queries">https://firebase.google.com/docs/firestore/query-data/queries</a>
Student Activity 1	Boilerplate for student	<a href="https://github.com/pro-whitehatjr/e-library-c73-student-boilerplate">https://github.com/pro-whitehatjr/e-library-c73-student-boilerplate</a>
Student Activity 2	Documentation for query	<a href="https://firebase.google.com/docs/firestore/query-data/queries">https://firebase.google.com/docs/firestore/query-data/queries</a>
Visual Aid Link	VA Link	<a href="https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C73-withoutcues.html">https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C73-withoutcues.html</a>
In-Class Quiz	In-Class quiz doc	<a href="https://s3-whjr-curriculum-uploads.whjr.online/add214af-1d92-495e-95fb-44fd1271722c.pdf">https://s3-whjr-curriculum-uploads.whjr.online/add214af-1d92-495e-95fb-44fd1271722c.pdf</a>
Project Solution	E Ride Stage-6	<a href="https://github.com/whitehatjr/PRO-C73-PROJECT">https://github.com/whitehatjr/PRO-C73-PROJECT</a>