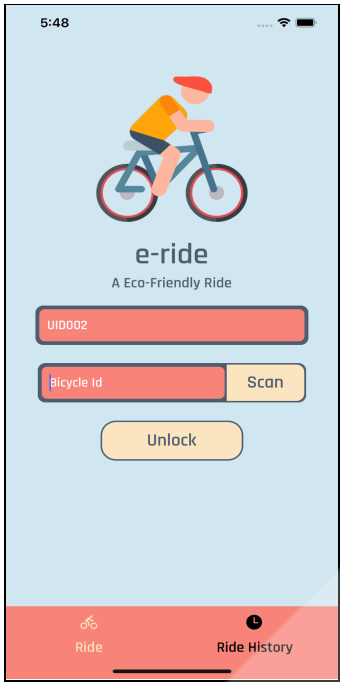
Topic	INTRODUCTION TO FIRESTORE	
Class Description	**Students learn how to design collections and documents in the firestore database. They design the database for the e-library app and program the submit button so that the book is issued or returned to a student as a library transaction.**	
Class	**C71**	
Class time	**45 mins**	
Goal	● Design e-library app database in firestore. ● Write code to issue/return a book by updating the database.	
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**
Credits	**QR Code by w69b GmbH, Schwabstr. 41, 72108 Rottenburg, Germany is licensed under the public domain of QR Code**	
WARM-UP SESSION - 5 mins		

CONTEXT
• **Talk about the functioning of the submit button and the way data would be stored for the e-library app.**
Teacher starts slideshow from slides 1 to 11 Refer to speaker notes and follow the instructions on each slide.

Activity details	Solution/Guidelines
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Run the presentation from slide 1 to slide 3** **Following are the WARM-UP session deliverables:** • **Greet the student.** • **Revision of previous class activities.** • **Quizzes**	**ESR**: Hi, thanks, Yes I am excited about it! Click on the slide show tab and present the slides

QnA Session	
Question	**Answer**
Choose the right block of code for importing the firestore library.	**A**



A.
```
import firebase from "firebase";
require ("@firebase/firestore");
```

B.
```
import firebase from "firebase";
require ["@firebase/firestore"];
```

C.
```
import firebase from "firebase";
require {"@firebase/firestore"};
```

D.
```
import firebase from "firebase";
require "@firebase/firestore";
```

Choose the right block of code for exporting firebase.firestore() from config.js file.

A.
```
export default firebase.firestore();
```

B.
```
export default firebase.firestore[];
```

C.
```
export default firebase.firestore{};
```

D.
```
export default firebase.firestore;
```

A

Continue the WARM-UP session

Activity details	Solution/Guidelines
Run the presentation from slide 4 to slide 11 to set the problem statement. **Following are the WARM-UP session deliverables:** • Appreciate the student. • Introduce Firestore	Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.
Teacher ends slideshow	
TEACHER-LED ACTIVITY - 15 mins	
Teacher Initiates Screen Share	
<u>CHALLENGE</u> • **Design the database for the e-library app in Firestore.**	
Teacher opens the code from the previous class. *OR* *Teacher can clone the code from Teacher Activity 1 and install all the dependencies for the project.* ***steps to clone the project:-*** ***git clone <projectURL>*** ***cd <projectFolder>*** ***npm install***	
Can we quickly go over the code we have written so far and review it so that we can understand what we have been doing so far? Let's quickly go over the code. *Teacher guides student to talk about:*	*The student goes through the code and explains how different features have been coded in the app so far.*

• How tab-navigation was used in the app • How tab-navigation was customized • How bar code scan was done • How we autofill the text component after scanning a QR code	
Alright. Let's quickly create a Submit Button which will be below our input fields. Can you help me with creating this Submit Button?	*The student guides the teacher to create the Submit button using Touchable Opacity and Text components.*

Code to create the submit button

```
<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>
      <TouchableOpacity
        style={[styles.button, { marginTop: 25 }]}
      >
        <Text style={styles.buttonText}>Submit</Text>
      </TouchableOpacity>
    </View>
  </ImageBackground>
</View>
```

```
);
```

Let's also style our Submit Button.	*The student guides the teacher in styling the Submit button and text inside it.*

```
scanbutton: {
  width: 100,
  height: 50,
  backgroundColor: "#9DFD24",
  borderTopRightRadius: 10,
  borderBottomRightRadius: 10,
  justifyContent: "center",
  alignItems: "center"
},
scanbuttonText: {
  fontSize: 24,
  color: "#0A0101",
  fontFamily: "Rajdhani_600SemiBold"
},
button: {
  width: "43%",
  height: 55,
  justifyContent: "center",
  alignItems: "center",
  backgroundColor: "#F48D20",
  borderRadius: 15
},
buttonText: {
  fontSize: 24,
  color: "#FFFFFF",
  fontFamily: "Rajdhani_600SemiBold"
}
});
```

What do we want to do when the Submit button is pressed? When do we want the book to be issued? When do we want the book to be returned?	**ESR:** It should issue or return the book to the student. **ESR:** We want the book to be issued when the book is available for issuing and hasn't been issued by someone else before. **ESR:** We want the book to be returned when the book has been issued to the student.
Awesome! Let's call a function **handleTransaction()** in the **onPress** prop for Submit Button. Note that we haven't yet written the function but it is helpful to think that this function is going to do everything that we want it to do. This is called abstraction in computer programming. An **abstraction** is a way to manage complexity. It's taking something that is inherently complex and making it simple to use and work with. We use abstraction repeatedly to simplify how we think and structure our code. We have been using abstraction earlier too.	*The student asks follow up questions about abstraction in programming.*

*Teacher calls **this.handleTransaction** inside **onPress** prop for the Submit button.* *Teacher also creates an empty **handleTransaction()** function.*	

creating empty handleTransaction() function and calling it when the submit button is pressed

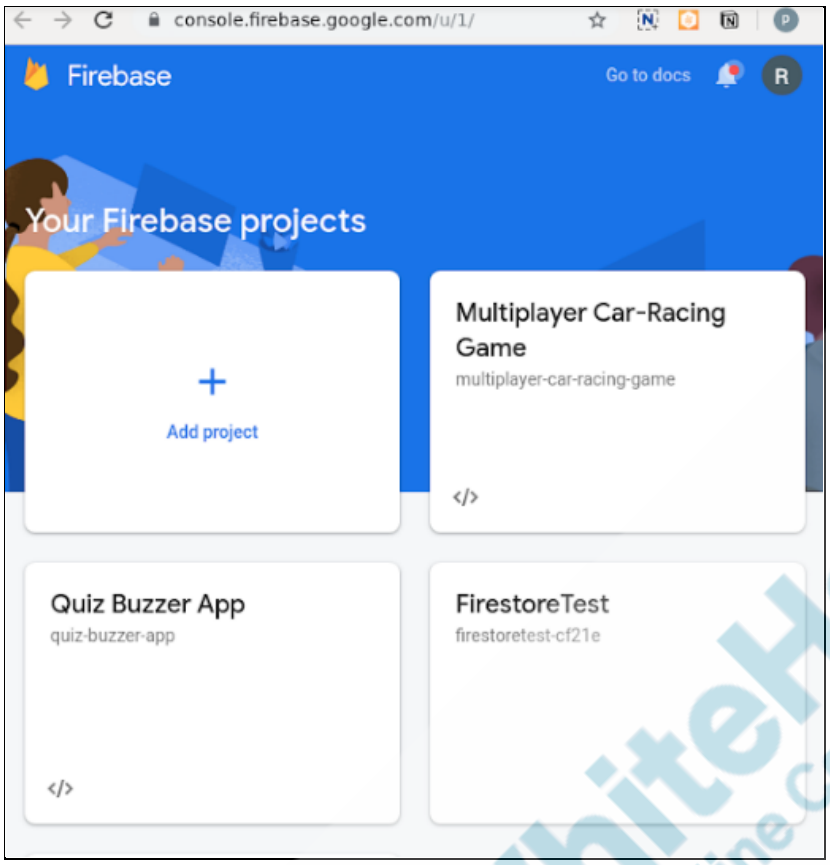
```
    if (domState === "bookId") {
      this.setState({
        bookId: data,
        domState: "normal",
        scanned: true
      });
    } else if (domState === "studentId") {
      this.setState({
        studentId: data,
        domState: "normal",
        scanned: true
      });
    }
  };

  handleTransaction = () => {

  };

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

```
<TouchableOpacity
  style={[styles.button, { marginTop: 25 }]}
  onPress={this.handleTransaction}
>
  <Text style={styles.buttonText}>Submit</Text>
</TouchableOpacity>
```

Before we can write code inside our **handleTransaction()** function, we need to design the database for our e-library App. Let's quickly create a Firestore Database. *Teacher creates a Firestore Database Project in test mode while explaining the steps to the student.* We will soon be learning more about Firestore security rules and then create a firestore database in production mode. Till then, let's learn how to use Firestore in test mode.	*The student observes how Firestore Database is created.*

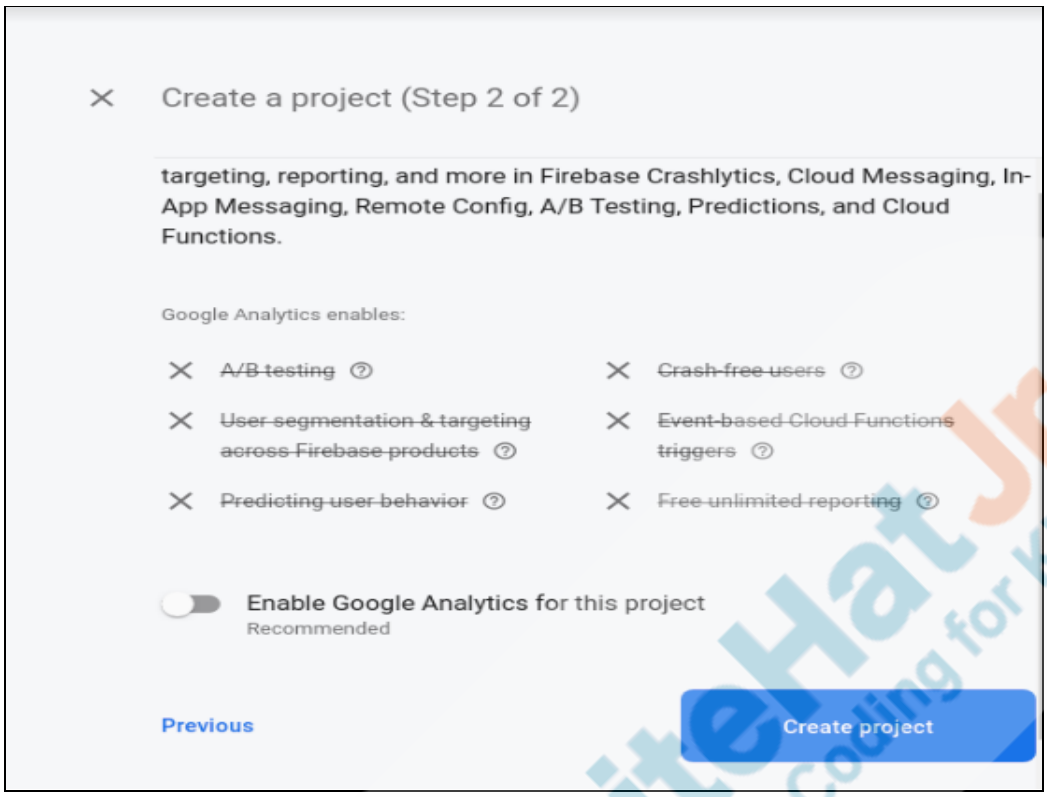
Steps to create a Firebase database.

1. **Click on add project**

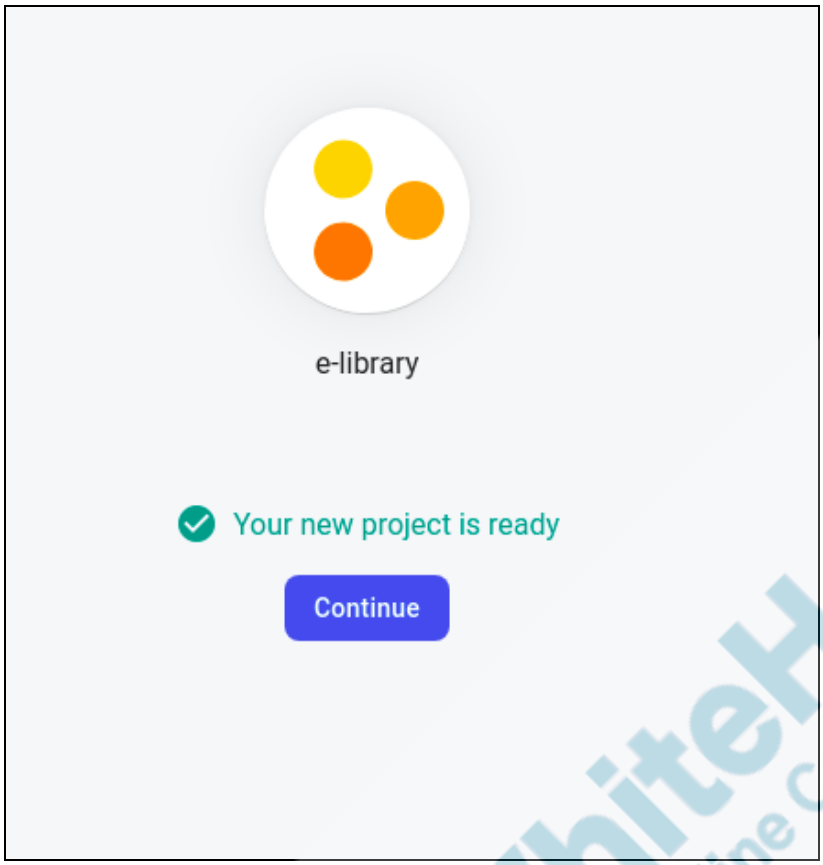


2. **Add the name for the project**

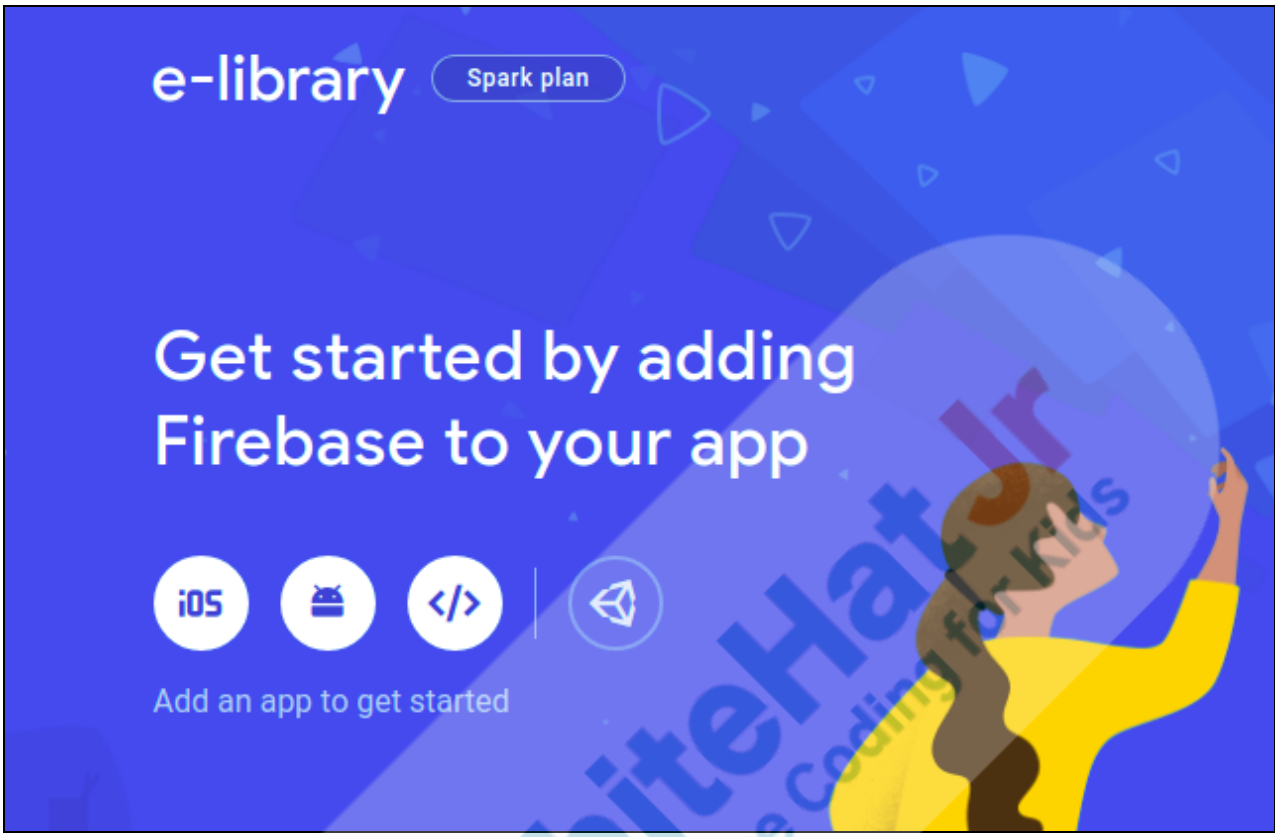
3. **Disable google analytics**



4. **Click on continue**



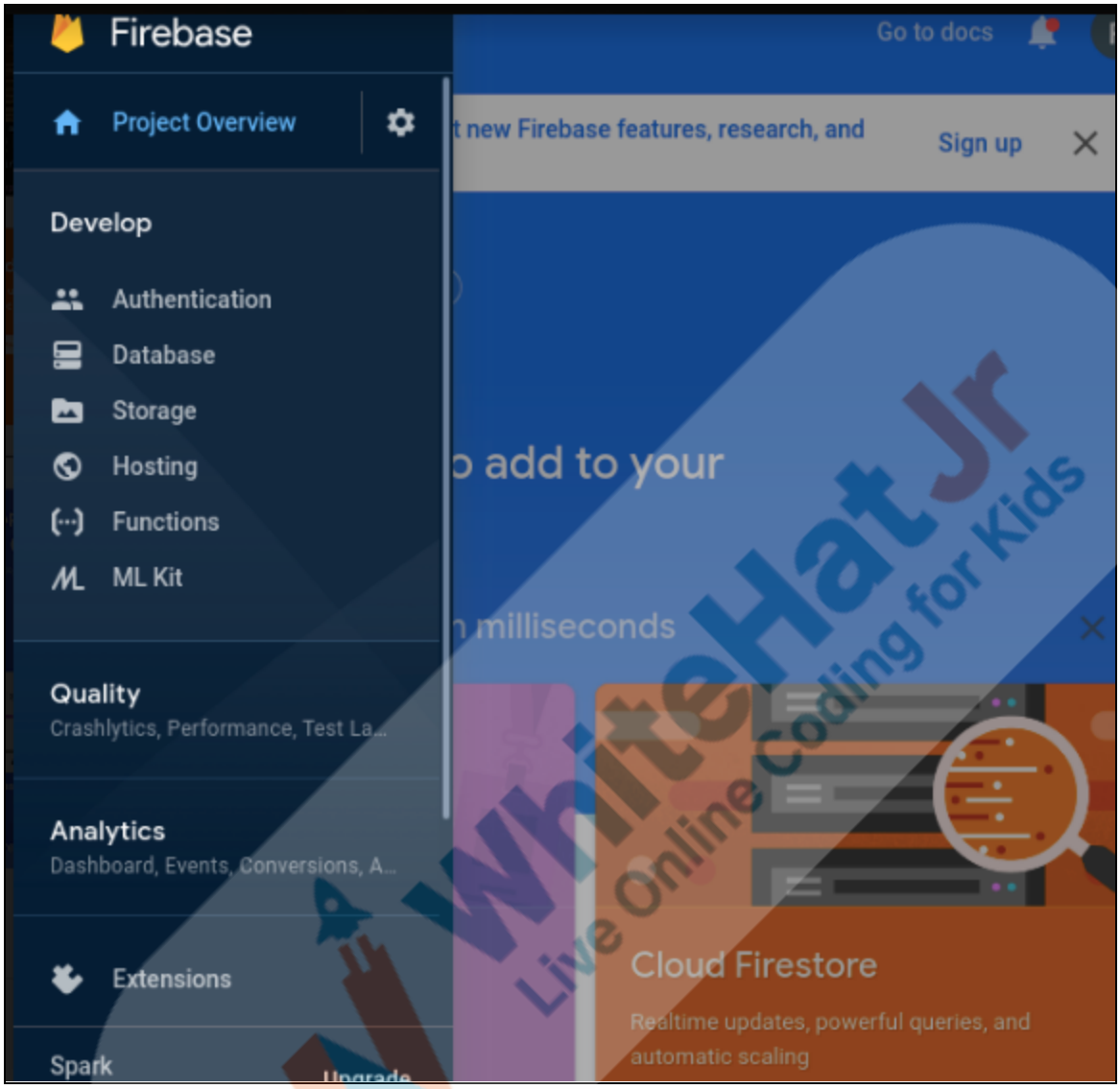
5. **Click on the code Icon.**



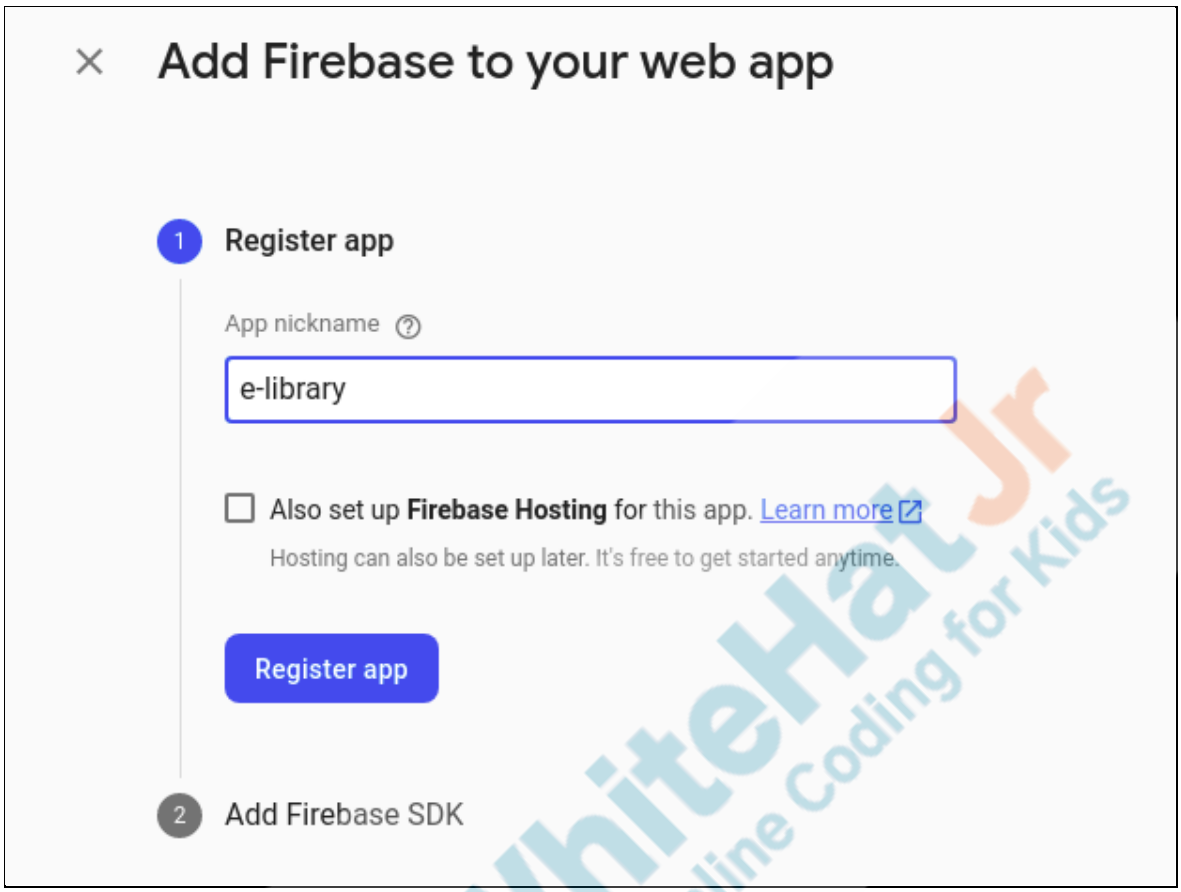
6. **Copy the credentials to add database to app.**

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-app.js"></script

<!-- TODO: Add SDKs for Firebase products that you want to use
     https://firebase.google.com/docs/web/setup#available-libraries -->

<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyB3dzLbgE8DPcHufZzeVKYwgmpXD0HFZYs",
    authDomain: "e-library-5fd21.firebaseapp.com",
    projectId: "e-library-5fd21",
    storageBucket: "e-library-5fd21.appspot.com",
    messagingSenderId: "665049297120",
    appId: "1:665049297120:web:7442fb17f050bde800e085"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
</script>
```

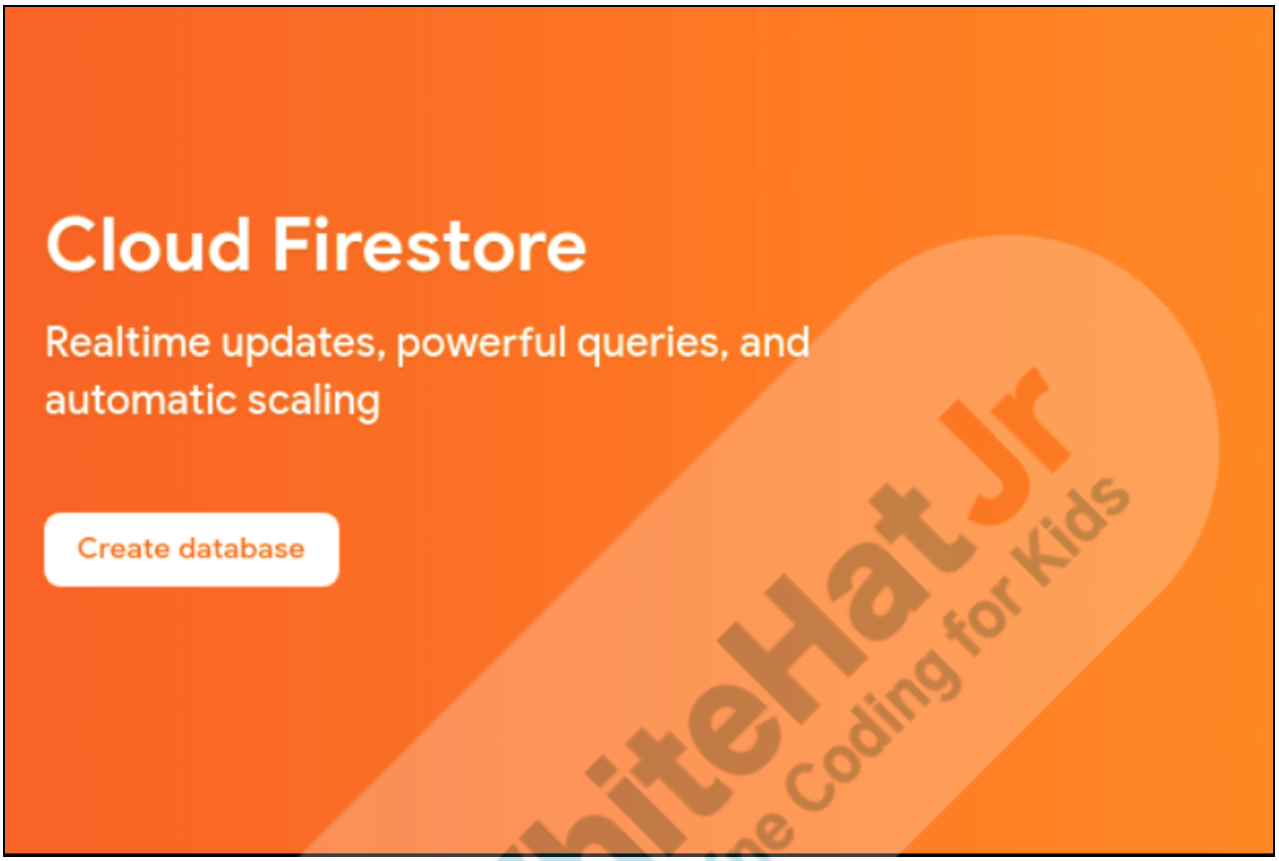
7. **Click on database in side panel**



8. **Add Firebase database to webapp.**

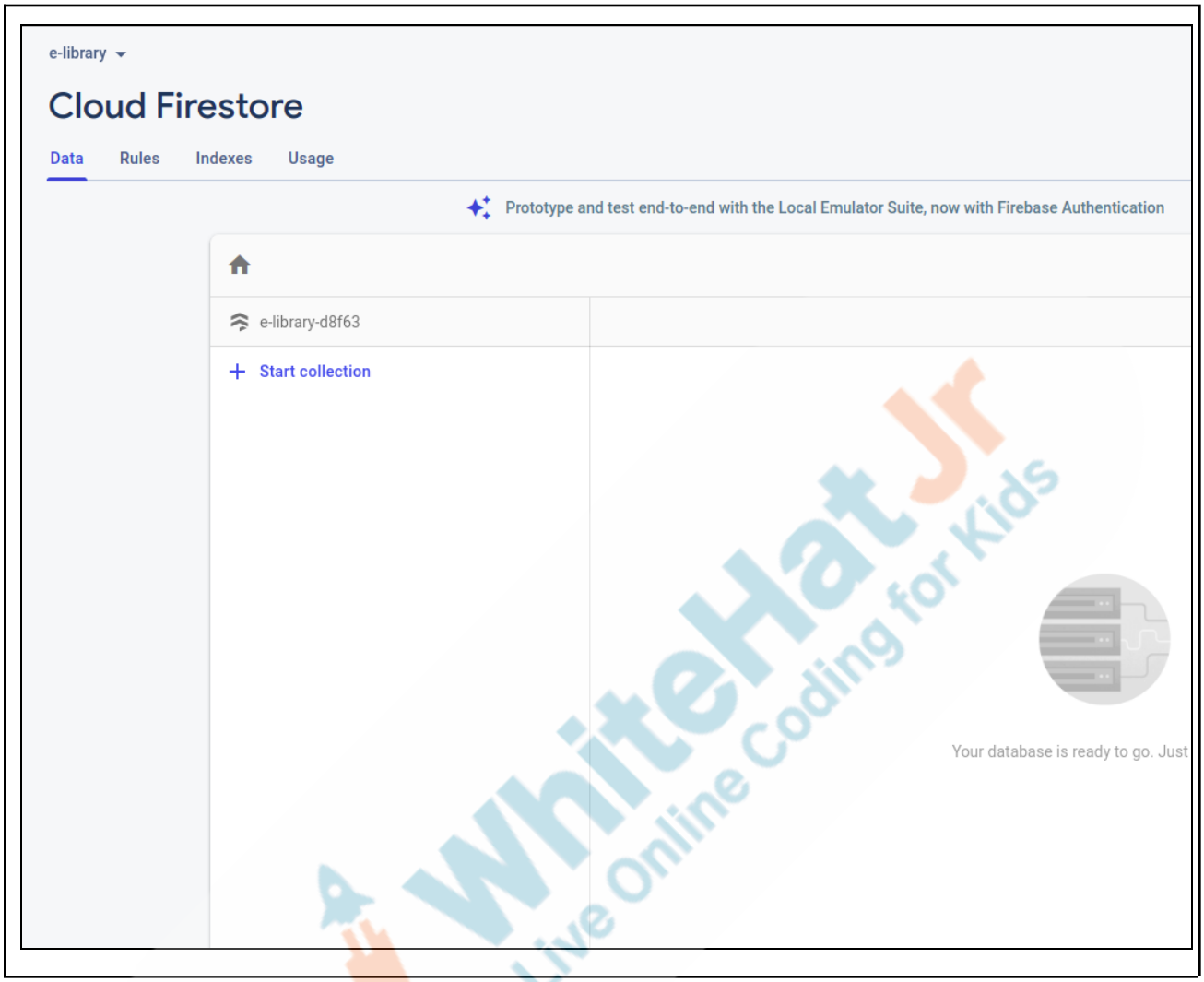


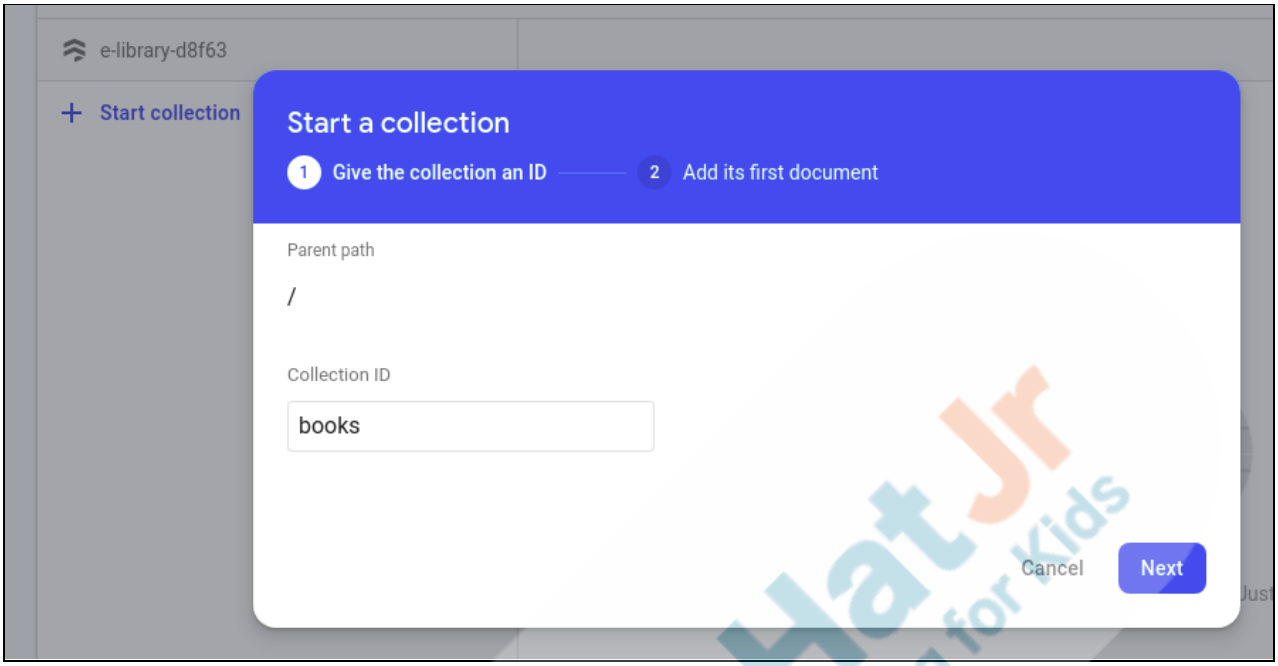
9. **Create the database.**



10. **Start in test mode.**

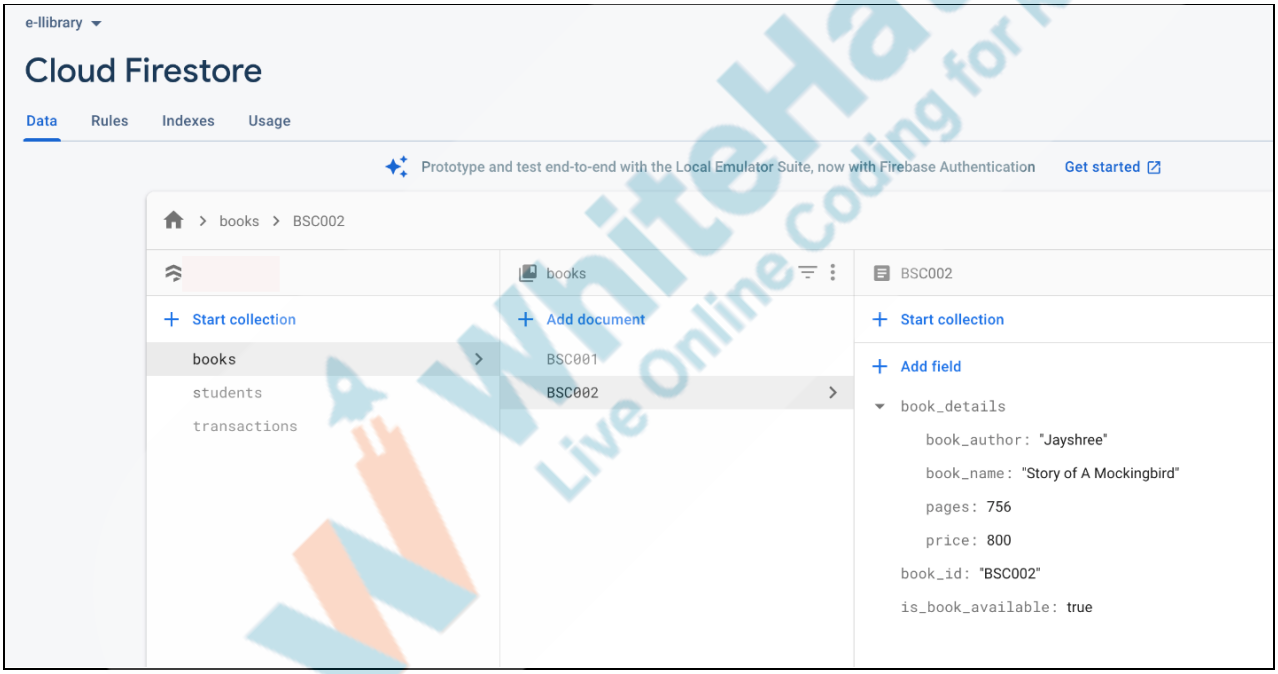
Firestore Databases organizes all data in terms of collections and documents. Collection is the name given to a group of documents holding some common properties. Documents are data stored inside collections as separate entities. We will quickly understand more when we create collections and documents for our own app.	*Student listens and asks questions.*



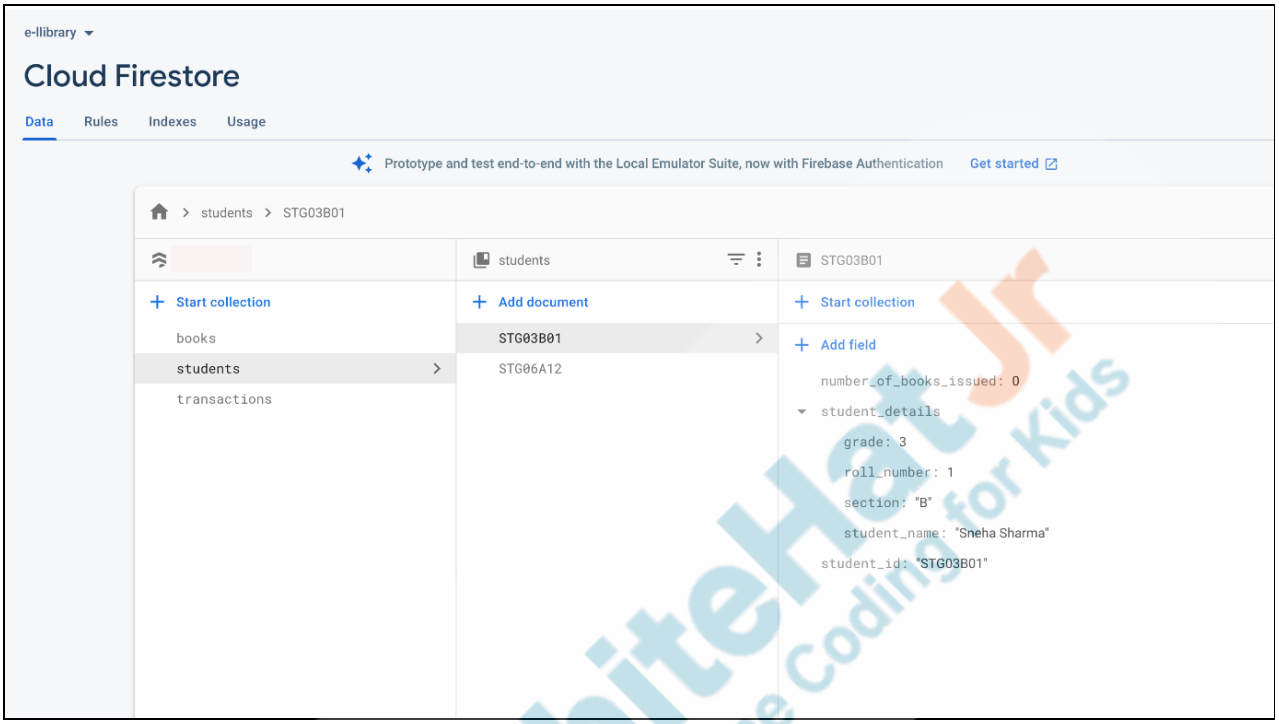


What are the two most important types of data that will be stored in our e-library app?	**ESR:** Books and Student info
Awesome! Let's create a collection called Books. *Teacher shows how to create a collection in Firestore.* Each document inside the collection will contain information about one book in the library. We will have as many documents as there are books in the library. What information about the book do we want to store?	**ESR:** book name, book author, number of pages, price, book availability, book description etc.

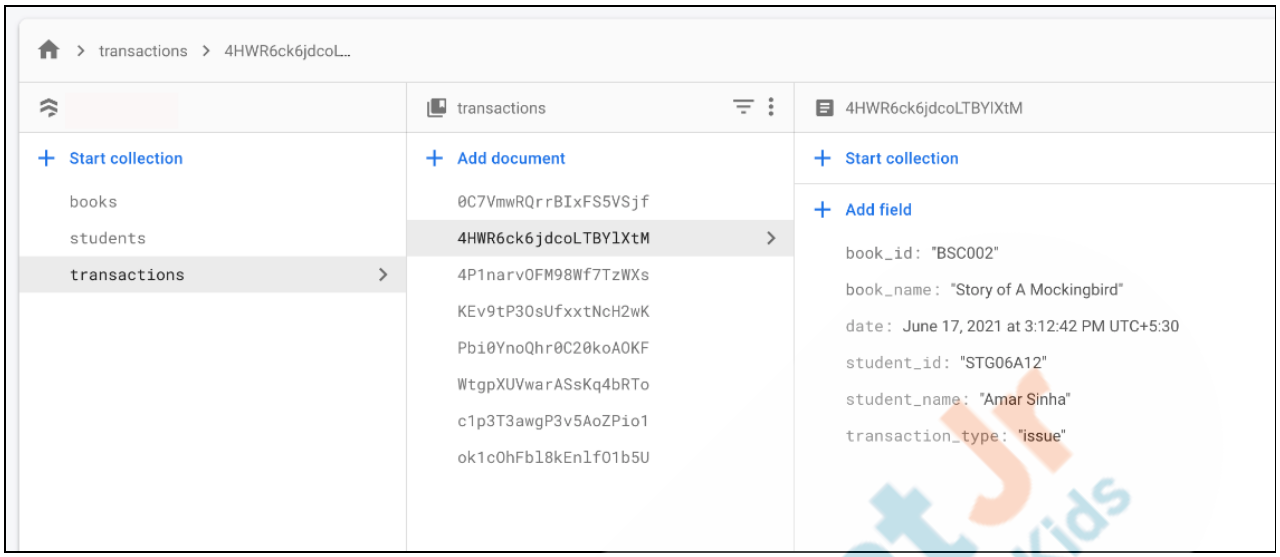
Awesome! We are going to create these fields inside every document in a book collection. Each document should also have a unique id. Let's call it a book id. If you don't give a unique id to your document, the firestore will generate an auto id for your document. *Teacher shows how to create a document with given fields. Teacher creates a sample book document with fields -* ***is_book_available, book_Id, book_details.***	*The student observes how to create a firestore document inside a collection.*



Now. Similarly let's create a student collection. What will the documents inside the student collection hold? *Teacher creates a Students Collection with a sample student document containing the fields -* ***student_Id, student_details, number_of_books_issued***	**ESR:** It will hold information about students like - student id, student name, grade, roll number, number of books issued etc.



Perfect! What other collections do we need in our e-library App Database?	**ESR:** *The student gives a thought and gives varied responses.*
Our e-library app will be used to issue or return books. Each issue or return is a transaction. Let's create a collection called **transactions**. It will contain the **book_id,** the **student_id**, the date of transaction and the type of transaction **(issue/return)**. *Teacher creates a transaction collection.* The document in our transaction collection will be created when the user presses the submit button.	*The student listens and asks questions.*



Let's get back to our code. What will we need to connect to our firestore database? You know where to get them. It is the same as a realtime database.	**ESR:** config keys for our database.
Let's create a config.js file in our app project where we initialize our firestore database and export firebase.firestore(). *Teacher shows how to create a config.js file: (similar to realtime database)* ● *Teacher imports the **firestore** library.* ● *Teacher exports the **firebase.firestore()** from **config.js** file.* ● *Teacher also installs firebase in the project using npm install firebase.*	*The student recollects how the config.js file was created in the Wireless Buzzer project.*

```js
JS config.js > ...
import firebase from "firebase";
require("@firebase/firestore");

const firebaseConfig = {
  apiKey: "AIzaSyAkp3gQdHbMPD5pCfHbBzIIgFpdCJXD5KM",
  authDomain: "e-library-v2.firebaseapp.com",
  projectId: "e-library-v2",
  storageBucket: "e-library-v2.appspot.com",
  messagingSenderId: "772559744213",
  appId: "1:772559744213:web:9417b879882f231175065c",
  measurementId: "G-H8Y352W6Z2"
};

firebase.initializeApp(firebaseConfig);

export default firebase.firestore();
```

```
C:\Users\ADMIN>npm install firebase
[..................] - rollbackFailedOptional: verb npm-session 734bf667a2aeba65
```

We are planning to use firestore in our Transaction Screen.

Let's import firestore as db here.

Teacher imports firebase.firestore() as db.

There are a number of functions pre-defined on db which we are going to use to update or create documents in our database.

```
import React, { Component } from "react";
import {
  View,
  StyleSheet,
  TextInput,
  TouchableOpacity,
  Text,
  ImageBackground,
  Image
} from "react-native";
import * as Permissions from "expo-permissions";
import { BarCodeScanner } from "expo-barcode-scanner";
import db from "../config";

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

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

Let's create two QR codes corresponding to the **studentId** and **bookId** we created in our database so that we can test the code we are going to write. *Teacher uses [Teacher Activity 2](#) to create two QR codes corresponding to the student id and book ids*	*The student helps in generating the QR code.*
What do we want to check once the book id and student ids are scanned?	**ESR:** • We want to check whether the book is available.

Which collection and document should we be looking into?	• We should be looking into the books collection with the document id same as the book id. • We should be checking the is_book_available field inside the document.
To get the data inside the document, there is a predefined function on db. It is invoked as: **db.collection(<collectionName>).doc(<docId>).get()** This function returns a promise - which means that it is an asynchronous function and will take time to get the document. The promise means that it is going to act as soon as the document is received. When the document is received, we can call **.then()** to do whatever we want to do with the document. **.then()** receives the document as a default argument. Let's quickly write the code to understand how to get the data from the document. *Teacher writes the code to get the data from the document with the same id as the bookId.* **doc.data()** is used to get all the information stored in the	*The student learns how to get data from the document using:* *db.collection(<collectionName>).doc(<docId>).get()* *The student asks questions to clarify doubts.*

document.

*Teacher can **console.log(doc.data())** to see the data getting printed on the TERMINAL WINDOW.*

Transaction.js

screens > Transaction.js > TransactionScreen

```
        this.setState({
          bookId: data,
          domState: "normal",
          scanned: true
        });
      } else if (domState === "studentId") {
        this.setState({
          studentId: data,
          domState: "normal",
          scanned: true
        });
      }
    };

    handleTransaction = () => {
      var { bookId } = this.state;
      db.collection("books")
        .doc(bookId)
        .get()
        .then(doc => {
          console.log(doc.data())
        });
    };

    |

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

```
Object {
  "book_details": Object {
    "book_author": "Abhijeet Holkar",
    "book_name": "Travel Magazine",
    "pages": 209,
    "price": 499,
  },
  "book_id": "BSC001",
  "is_book_available": false,
}
```

The returned data is a json object. We need to get the **is_book_available** field from the data. - If the book is available, we want to issue the book. - If the book is not available, we want to return the book. Let's write an if-else condition to do this. We will also use two abstract functions - **initiateBookIssue()** and **initiateBookReturn()** to write our code. Later we are going to write code for these functions. *Teacher writes code to call **initiateBookIssue()** if the book is available or **initiateBookReturn()** when the book is not available.*	*The student observes how abstract functions can be used to simplify thinking about the program.*

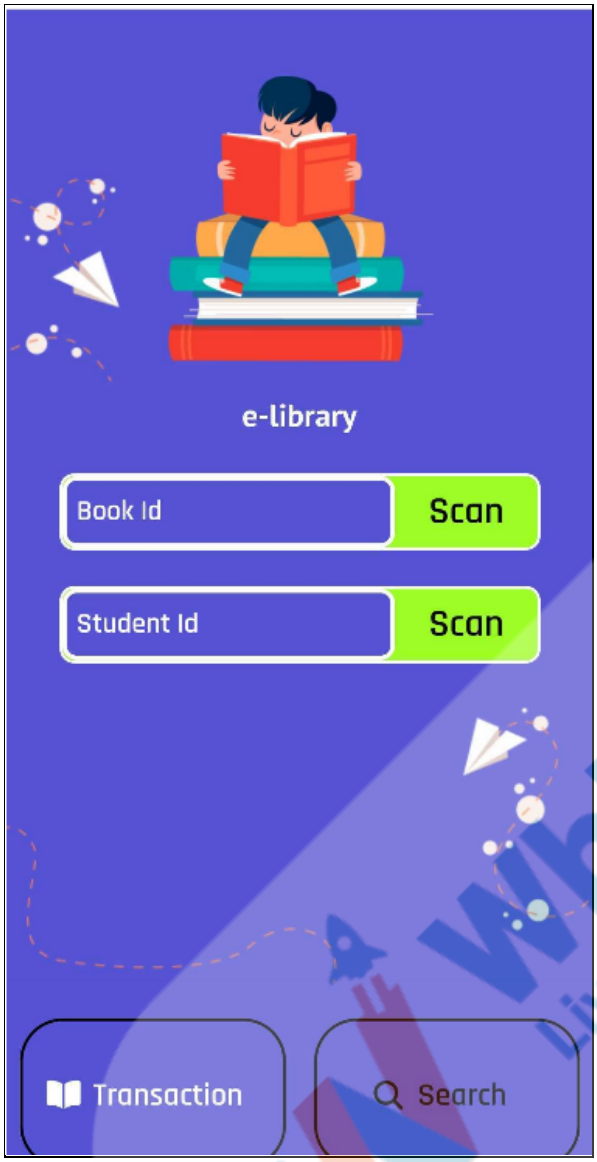
screens > Transaction.js > TransactionScreen

```
53      } else if (domState === "studentId") {
54        this.setState({
55          studentId: data,
56          domState: "normal",
57          scanned: true
58        });
59      }
60    };
61
62    handleTransaction = () => {
63      var { bookId } = this.state;
64      db.collection("books")
65        .doc(bookId)
66        .get()
67        .then(doc => {
68          var book = doc.data();
69          if (book.is_book_available) {
70            this.initiateBookIssue();
71          } else {
72            this.initiateBookReturn();
73          }
74        });
75    };
76
77    |
78
79    render() {
80      const { bookId, studentId, domState, scanned } = this.state;
81      if (domState !== "normal") {
82        return (
83          <BarCodeScanner
84            onBarCodeScanned={scanned ? undefined : this.handleBarCodeScanned}
85            style={StyleSheet.absoluteFillObject}
86          />
87        );
88      }
89      return (
90        <View style={styles.container}>
91          <ImageBackground source={bgImage} style={styles.bgImage}>
```

screens > Transaction.js > TransactionScreen

```
    } else if (domState === "studentId") {
      this.setState({
        studentId: data,
        domState: "normal",
        scanned: true
      });
    }
  };

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

  initiateBookIssue = () => {
    console.log("Book issued to the student!");
  };

  initiateBookReturn = () => {
    console.log("Book returned to the library!");
  };

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

Let's quickly test our code so far.



Wow! We have created the book issue / return functionality in our e-library app.

There is of course no message displayed to the user now for what has happened. We will be changing that in the next class.

For now, can you design the database and write code for your own app?

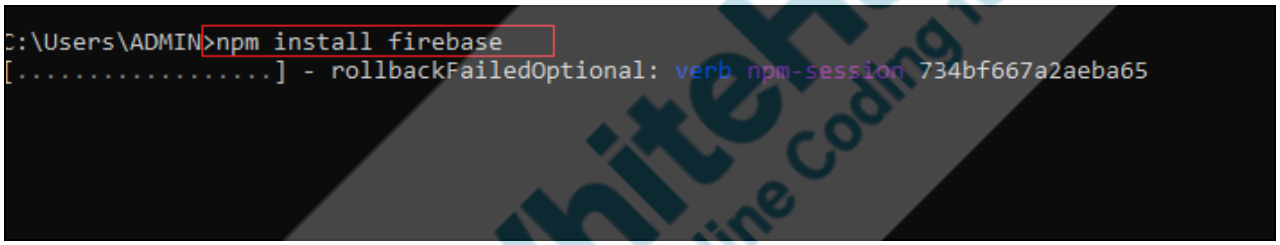
Teacher Stops Screen Share		
	Now it's your turn. Please share your screen with me.	
STUDENT-LED ACTIVITY - 25 mins		
• **Ask Student to press ESC key to come back to panel** • **Guide Student to start Screen Share** • **Teacher gets into Fullscreen**		
<u>ACTIVITY</u> • **Write a transaction function which handles the issue or return transaction for the student.**		
Teacher starts slideshow :Slide 12 to 13 Refer to speaker notes and follow the instructions on each slide.		
Teacher Action		**Student Action**
Help the student to open the previous class project.		*The student opens the previous class project.*
Create and style the submit button.		*The student creates the submit button and styles it.*

```
            <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>

            <TouchableOpacity
              style={[styles.button, { marginTop: 25 }]}
            >
              <Text style={styles.buttonText}>Submit</Text>
            </TouchableOpacity>

          </View>
        </ImageBackground>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#FFFFFF"
  },
  bgImage: {
    flex: 1,
    resizeMode: "cover",
    justifyContent: "center"
  },
```

screens > Transaction.js > TransactionScreen > render

```
187        borderRadius: 10,
188        borderWidth: 3,
189        fontSize: 18,
190        backgroundColor: "#5653D4",
191        fontFamily: "Rajdhani_600SemiBold",
192        color: "#FFFFFF"
193      },
194      scanbutton: {
195        width: 100,
196        height: 50,
197        backgroundColor: "#9DFD24",
198        borderTopRightRadius: 10,
199        borderBottomRightRadius: 10,
200        justifyContent: "center",
201        alignItems: "center"
202      },
203      scanbuttonText: {
204        fontSize: 24,
205        color: "#0A0101",
206        fontFamily: "Rajdhani_600SemiBold"
207      },
208      button: {
209        width: "43%",
210        height: 55,
211        justifyContent: "center",
212        alignItems: "center",
213        backgroundColor: "#F48D20",
214        borderRadius: 15
215      },
216      buttonText: {
217        fontSize: 24,
218        color: "#FFFFFF",
219        fontFamily: "Rajdhani_600SemiBold"
220      }
221    });
222
```

Create a firestore database with books, students and transaction collections. Create a test book and student document.	*The student creates the firestore database.* *He/She creates three collections - books,*

	students and transactions. *The student creates the test book and student documents with the appropriate fields.*
• install **firebase** on their project using **npm install**. • create a **config.js** file which exports **firebase.firestore()**	*The student installs the firebase library in their project.* *The student creates the config.js file and imports it in the TransactionScreen.*
	

```js
JS config.js > ...
import firebase from "firebase";
require("@firebase/firestore");

const firebaseConfig = {
  apiKey: "AIzaSyAkp3gQdHbMPD5pCfHbBzIIgFpdCJXD5KM",
  authDomain: "e-library-v2.firebaseapp.com",
  projectId: "e-library-v2",
  storageBucket: "e-library-v2.appspot.com",
  messagingSenderId: "772559744213",
  appId: "1:772559744213:web:9417b879882f231175065c",
  measurementId: "G-H8Y352W6Z2"
};

firebase.initializeApp(firebaseConfig);

export default firebase.firestore();
```

Create a function called **handleTransaction()** which is called when the submit button is pressed.	*The student creates the handleTransaction function which is called when the submit button is pressed.*

screens > Transaction.js > TransactionScreen

```
53      } else if (domState === "studentId") {
54        this.setState({
55          studentId: data,
56          domState: "normal",
57          scanned: true
58        });
59      }
60    };
61
62    handleTransaction = () => {
63      var { bookId } = this.state;
64      db.collection("books")
65        .doc(bookId)
66        .get()
67        .then(doc => {
68          var book = doc.data();
69          if (book.is_book_available) {
70            this.initiateBookIssue();
71          } else {
72            this.initiateBookReturn();
73          }
74        });
75    };
76
77
78
79    render() {
80      const { bookId, studentId, domState, scanned } = this.state;
81      if (domState !== "normal") {
82        return (
83          <BarCodeScanner
84            onBarCodeScanned={scanned ? undefined : this.handleBarCodeScanned}
85            style={StyleSheet.absoluteFillObject}
86          />
87        );
88      }
89      return (
90        <View style={styles.container}>
91          <ImageBackground source={bgImage} style={styles.bgImage}>
```

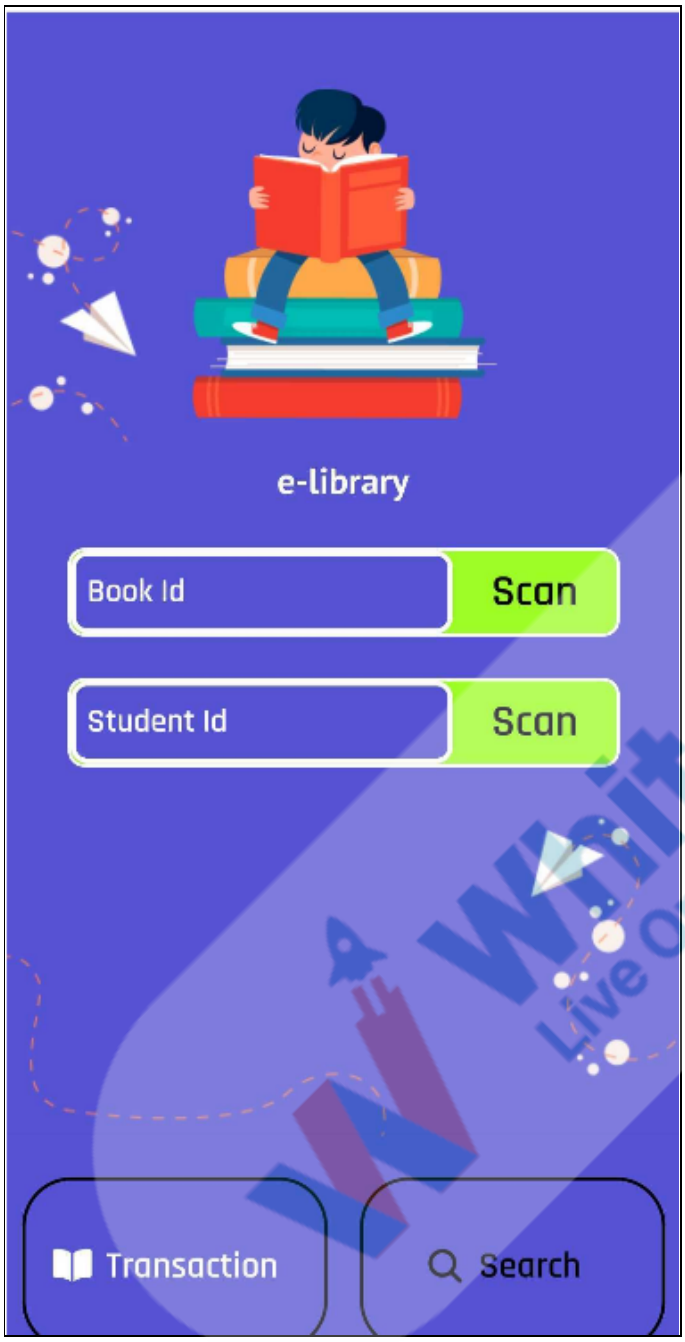
Get the book availability data from the scanned book id and call issue or return functions to complete the book transaction.	*The student gets the **is_book_available** from the book doc and calls the book issue or return function based on book availability.*

screens > Transaction.js > TransactionScreen

```
53      } else if (domState === "studentId") {
54        this.setState({
55          studentId: data,
56          domState: "normal",
57          scanned: true
58        });
59      }
60    };
61
62    handleTransaction = () => {
63      var { bookId } = this.state;
64      db.collection("books")
65        .doc(bookId)
66        .get()
67        .then(doc => {
68          var book = doc.data();
69          if (book.is_book_available) {
70            this.initiateBookIssue();
71          } else {
72            this.initiateBookReturn();
73          }
74        });
75    };
76
77    initiateBookIssue = () => {
78      console.log("Book issued to the student!");
79    };
80
81    initiateBookReturn = () => {
82      console.log("Book returned to the library!");
83    };
84
85    render() {
86      const { bookId, studentId, domState, scanned } = this.state;
87      if (domState !== "normal") {
88        return (
89          <BarCodeScanner
90            onBarCodeScanned={scanned ? undefined : this.handleBarCodeScanned}
91            style={StyleSheet.absoluteFillObject}
```

Write the book issue and return functions.	*The student writes the book issue and return function where:* ● *he/she writes the code to console log the message on the terminal depending on the request type(*

	either it is return or issue)
Test the code by running the app on the phone and look at the terminal for the messages which we consoled earlier. *Help the student debug the code.'* - *look for typos* - *check each code block's output by logging on the console etc.*	*The student tests the app by creating and scanning the QR codes for student id and book id.* *The student checks the terminal for the message that they consoled earlier.*



Awesome job today. So far you have learned to get the data from the db and issue or return the book based on the availability of the book.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 5 Mins

Teacher starts slideshow from slide 14 to slide 25	
Activity details	**Solution/Guidelines**
Run the presentation from slide 14 to slide 25 **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 Student will ask doubts related to the activities.
Quiz time - Click on in-class quiz	
Question	**Answer**
What is an abstraction? A. to inherit properties and functions from parent class to the child class B. it is the process of creating a class C. when the function is not written but it is helpful to think that this function is going to do everything that we want it to do D. it is the process of creating an object of the class	**D.**
Which of the following statements is incorrect? A. Collection is the name given to a group of documents holding some common properties. B. Documents are data stored inside collections as separate entities. C. Firestore Database organizes all data in terms of collections and documents. D. Cloud firestore databases are different from firestore databases.	**C.**
To get the data inside the document, which predefined function on db is used?	**A.**

A. .get() B. Student, Book and Transaction C. studentId, bookId, and transactionid D. Teacher, Student and Librarian	
End the quiz panel	
FEEDBACK • **Encourage the student to read more about functions/apis available in Google firestore.**	
Teacher Action	**Student Action**
There was a lot we learned in today's class. Let's quickly wrap up the class today. Can you quickly summarize what we learned today?	**ESR:** • Use of abstraction in writing our code. • Designing database in firestore. • Creating and updating a document in firestore.
Awesome. There is also the search functionality which we are yet to build. Think about how you would be doing that using a firestore. In the next class we will fix the issue of Keyboard Overlapping and show messages using Toasts. There are many issues still in our library app. Try to think about them. For example: Right now any student can return a book issued by someone else. We will be fixing them in the upcoming sessions.	

**** This Project will take only 30 mins to complete. Motivate students to try and finish it immediately after the class.***

E-RIDE STAGE 4

Goal of the Project:

In class 71, you have learned how to design collections and documents in the firestore database. You designed the database for the e-library app and programmed the submit button so that the book is issued or returned to a student as a library transaction.

You will use similar concepts to create a database for an e-ride App to store Bike information, User Information, and Transactions.

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

Story:

Your friend Vihaan is very impressed with the UI and QR code functionality. He has given you access to his firestore to prepare the database for saving all information and access it by app.

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-C70
Teacher Activity 2	Qr Code Generator	https://www.the-qrcode-generator.com/
Teacher Activity 3	Firebase doc	https://cloud.google.com/firestore/docs/manage-data/add-data
Teacher Activity 4	Final code	https://github.com/whitehatjr/e-library-PRO-C71
Student Activity 1	Qr code generator	https://www.the-qrcode-generator.com/
Student Activity 2	Firebase Doc	https://cloud.google.com/firestore/do

		cs/manage-data/add-data
Visual-Aid	Visual-Aid Link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C71-withcues.html
In-Class Quiz	In-class quiz link	https://s3-whjr-curriculum-uploads.whjr.online/388c7211-f89b-4244-aa45-26d52574bde5.pdf
Project Solution	E-Ride-Stage-4	https://github.com/whitehatjr/PRO-C71-PROJECT