

Topic	Lazy Loading	
Class Description	Students learn how to display items in a list using the FlatList component. They build a search bar in the search screen to display the list of transactions queried by the user.	
Class	C74	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> • Display transactions in a Flatlist. • Build a search bar to query transactions collection to search for a field value. • Display the query results in a Flatlist. 	
Resources Required	<ul style="list-style-type: none"> • Teacher Resources <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Android/iOS Smartphone with Expo App installed • Student Resources <ul style="list-style-type: none"> ○ 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 15 min 5 min
<div style="text-align: center;"><u>CONTEXT</u></div> <ul style="list-style-type: none"> • Talk to the student about the logic for the search screen. • Talk about limitations of the ScrollView to display a list of items. 		

Class Steps	Teacher Action	Student Action
Step 1: Warm Up (5 mins)	<p>Hi <student name></p> <p>Last class we had completed designing and coding for the book transaction screen. Our user/ teacher can now help the students in issuing or returning a book from the library.</p> <p>What is still pending in our Wily App?</p>	<p>ESR: Search Screen</p>
	<p>What should our user/teacher be able to do with the search screen?</p>	<p>ESR: User should be able to search for a book id or student id and the screen should display the last transactions for the book id and student id</p>
	<p>Awesome! There might be a huge number of transactions for any student id or book id. How would we display them on the screen?</p>	<p>ESR: Using ScrollView. The user could scroll through the list of all the transactions.</p>
	<p>That is one way to do it.</p> <p>However, ScrollView takes and renders all the items in the list at once. It takes time for the javascript engine to load and render all the data. This will make the app less performant.</p>	<p><i>The student asks questions around ScrollView performance.</i></p>
	<p>There is another kind of React Component which lazily loads the list.</p>	

	<p>It loads only a list of items which can be displayed on one screen. When the user scrolls down, it fetches new data, adds them to the list and renders it on the screen.</p> <p>This react component is called a FlatList. Have you ever seen this kind of loading in any app?</p>	<p><i>The student talks about Facebook, Twitter and other kinds of apps where there is a newsfeed.</i></p>
Teacher Initiates Screen Share		
<p align="center"><u>CHALLENGE</u></p> <ul style="list-style-type: none"> • Display all the transactions in a FlatList. 		
Step 2: Teacher-led Activity (15 min)	<p><i>Teacher opens the activity from the previous class.</i></p> <p><i>Or Teacher clones previous class activity from Teacher Activity 1 and installs all the dependencies for the project.</i></p>	
	<p>As usual, before proceeding, let's review the code from the previous class.</p> <p>*Note: Get the student to focus on how firebase queries are done.</p>	<p><i>The student reviews the code from the previous class. He/She elaborates on how firebase queries were done in the app.</i></p>
	<p>Let's build our search screen now. Before we learn to use the FlatList component, let us display ALL the transaction documents from transaction collections in our search screen.</p> <p>How do you think we can do that?</p>	<p>ESR:</p> <p>We will create a state array which will hold all the transactions.</p> <p>When the screen mounts, we will query all the transactions and store them in the state array.</p>

		In the render function for the component, we will map over the state array and display each item.
	<p>Awesome! Let's try to do this.</p> <p><i>Teacher creates a state called allTransactions as an empty array.</i></p>	<p><i>The student guides the teacher in creating an empty state array called allTransactions inside the constructor.</i></p>
<pre> 1 import React from 'react'; 2 import { Text, View } from 'react-native'; 3 4 export default class Searchscreen extends React.Component { 5 constructor(props){ 6 super(props) 7 this.state = { 8 allTransactions: [] 9 } 10 } 11 render() { 12 return (13 <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}> 14 <Text>Search</Text> 15 </View> 16); 17 } </pre>		
	<p>Let's store the entire list of transactions inside this state when the component mounts.</p> <p><i>Teacher imports db from config.js and writes code to modify the state allTransactions inside componentDidMount function.</i></p>	<p><i>The student guides the teacher to get all the documents inside the transactions collection and store them in the allTransactions array.</i></p>

```
import React from 'react';
import { Text, View } from 'react-native';
import db from '../config'

export default class Searchscreen extends React.Component {
  constructor(props){
    super(props)
  }
}
```

```

screens > Searchscreen.js > Searchscreen > render
/
8   super(props)
9   this.state = {
10    allTransactions: []
11  }
12
13  componentDidMount = async ()=>{
14    const query = await db.collection("transactions").get()
15    query.docs.map((doc)=>{
16      this.setState({
17        allTransactions: [...this.state.allTransactions, doc.data()]
18      })
19    })
20  }
21  render() {
22    return (
23      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
```

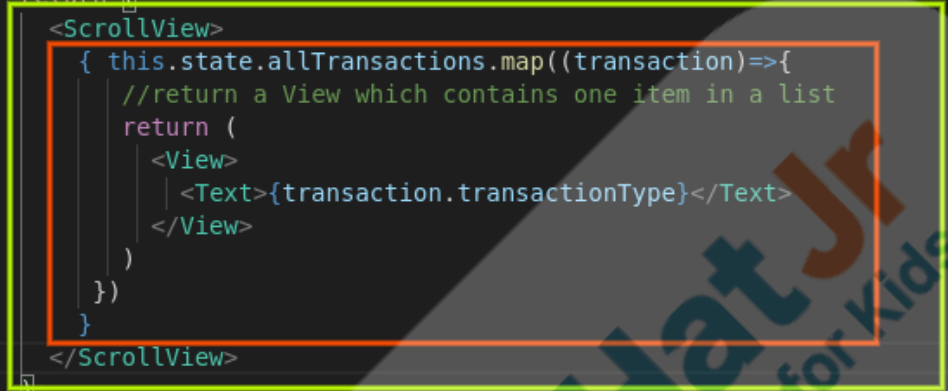
Let's render a ScrollView in our render function.

Inside ScrollView, we are going to write some javascript code. We will map over the allTransactions state array.

For each element in the array, we will return a View component containing information about that transaction using Text Components.

***Note:** Use curly brackets before writing Javascript code. toDate() function converts the firebase data value (in ms) to date string.

The student guides the teacher in mapping over the allTransactions state array and rendering a View component containing the transaction data for each item in the array.

	Teacher writes code with the guidance from the student.	
<pre> 22 }) 23 } 24 render() { 25 return (26 <ScrollView> 27 { this.state.allTransactions.map((transaction)=>{ 28 //return a View which contains one item in a list 29 return (30 <View> 31 <Text>{transaction.transactionType}</Text> 32 </View> 33) 34 }) 35 } 36 </ScrollView> 37), </pre>		
	<p>You might get a warning for giving a unique key prop for all the items in the list.</p> <p>A unique prop helps each list item to be identified and that helps in rendering the items.</p> <p>It is good practice to add a unique key to each component in the list. We will use the array index as the unique key here.</p> <p><i>Teacher shows how to add a unique key prop to each rendered View Component in the list.</i></p>	<p><i>The student observes and asks questions.</i></p>

```

render() {
  return (
    <ScrollView>
      { this.state.allTransactions.map((transaction, index)=>{
        //return a view which contains one item in a list
        return (
          <View style={{borderBottomWidth: 2}}>
            <Text>{"Book Id: " + transaction.bookId}</Text>
            <Text>{"Student Id: " + transaction.studentId}</Text>
            <Text>{"Transaction Type: " + transaction.transactionType}</Text>
            <Text>{"Date: " + transaction.date.toString()}</Text>
          </View>
        )
      })
    )
  )
}

```

```

}
render() {
  return (
    <ScrollView>
      { this.state.allTransactions.map((transaction, index)=>{
        //return a View which contains one item in a list
        return (
          <View key={index} style={{borderBottomWidth: 2}}>
            <Text>{"Book Id: " + transaction.bookId}</Text>
            <Text>{"Student Id: " + transaction.studentId}</Text>
            <Text>{"Transaction Type: " + transaction.transactionType}</Text>
            <Text>{"Date: " + transaction.date.toString()}</Text>
          </View>
        )
      })
    )
  )
}



```

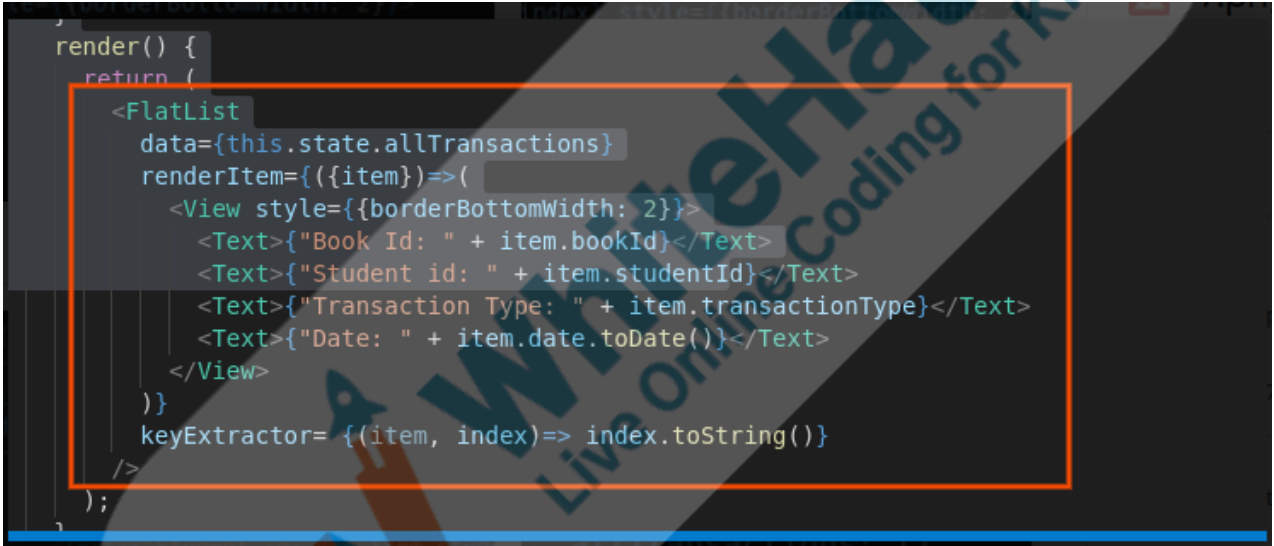
You can see that it takes a while for the items to be rendered inside scrollView.

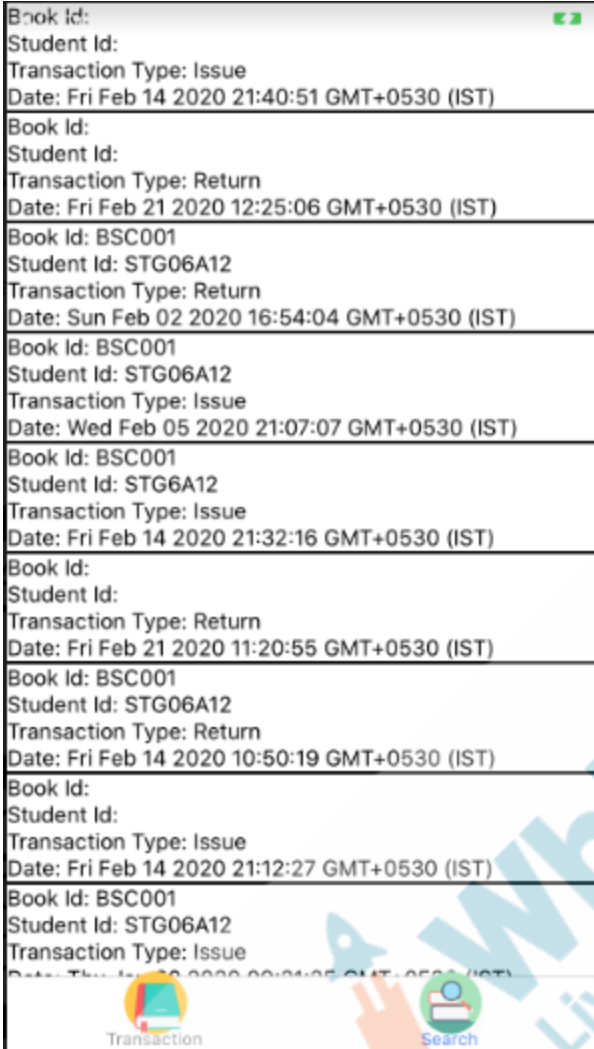
This is because, before rendering it waits to get all the items and only then renders them.

This will make our app less performant. To avoid that, we use FlatList.

The student and teacher go through the FlatList documentation.

	Teacher shows documentation for FlatList to the student.	
<div><div><div>Book Id:</div><div>Student Id:</div><div>Transaction Type: Issue</div><div>Date: Fri Feb 14 2020 21:40:51 GMT+0530 (IST)</div></div><div><div>Book Id:</div><div>Student Id:</div><div>Transaction Type: Return</div><div>Date: Fri Feb 21 2020 12:25:06 GMT+0530 (IST)</div></div><div><div>Book Id: BSC001</div><div>Student Id: STG06A12</div><div>Transaction Type: Return</div><div>Date: Sun Feb 02 2020 16:54:04 GMT+0530 (IST)</div></div><div><div>Book Id: BSC001</div><div>Student Id: STG06A12</div><div>Transaction Type: Issue</div><div>Date: Wed Feb 05 2020 21:07:07 GMT+0530 (IST)</div></div><div><div>Book Id: BSC001</div><div>Student Id: STG6A12</div><div>Transaction Type: Issue</div><div>Date: Fri Feb 14 2020 21:32:16 GMT+0530 (IST)</div></div><div><div>Book Id:</div><div>Student Id:</div><div>Transaction Type: Return</div><div>Date: Fri Feb 21 2020 11:20:55 GMT+0530 (IST)</div></div><div><div>Book Id: BSC001</div><div>Student Id: STG06A12</div><div>Transaction Type: Return</div><div>Date: Fri Feb 14 2020 10:50:19 GMT+0530 (IST)</div></div><div><div>Book Id:</div><div>Student Id:</div><div>Transaction Type: Issue</div><div>Date: Fri Feb 14 2020 21:12:27 GMT+0530 (IST)</div></div><div><div>Book Id: BSC001</div><div>Student Id: STG06A12</div><div>Transaction Type: Issue</div><div>Date: Thu Feb 13 2020 00:04:05 GMT+0530 (IST)</div></div></div> <div><div>Transaction</div><div>Search</div></div>		
<h3>Data in ScrollView</h3>		
	<p>Let's use FlatList to list down all the transactions.</p> <p>FlatList has 3 key props:</p> <ul style="list-style-type: none">● data: This contains all the data in the array which needs to be rendered.● renderItem: This takes each item from the data array and	<p>The student asks questions around the FlatList.</p>

	<p>renders it as described using JSX. This should return a JSX component.</p> <ul style="list-style-type: none"> • keyExtractor: It gives a unique key prop to each item in the list. The unique key prop should be a string. 	
	<p><i>Teacher codes to use FlatList for rendering the transactions using the 3 props described above.</i></p>	<p><i>The student observes the teacher write the code and asks questions for clarification.</i></p>
 <pre> render() { return (<FlatList data={this.state.allTransactions} renderItem={({item})=>(<View style={{borderBottomWidth: 2}}> <Text>{"Book Id: " + item.bookId}</Text> <Text>{"Student id: " + item.studentId}</Text> <Text>{"Transaction Type: " + item.transactionType}</Text> <Text>{"Date: " + item.date.toDate()}</Text> </View>)} keyExtractor= {(item, index)=> index.toString()} />); } </pre>		
	<p><i>Teacher runs the code and tests.</i></p> <p>You can see that the list is getting rendered the same as the ScrollView.</p> <p>Even in FlatLists, we are waiting for all the transaction documents in ComponentDidMount before rendering these items.</p> <p>We don't have to do that.</p>	<p><i>The student and teacher test the code.</i></p>

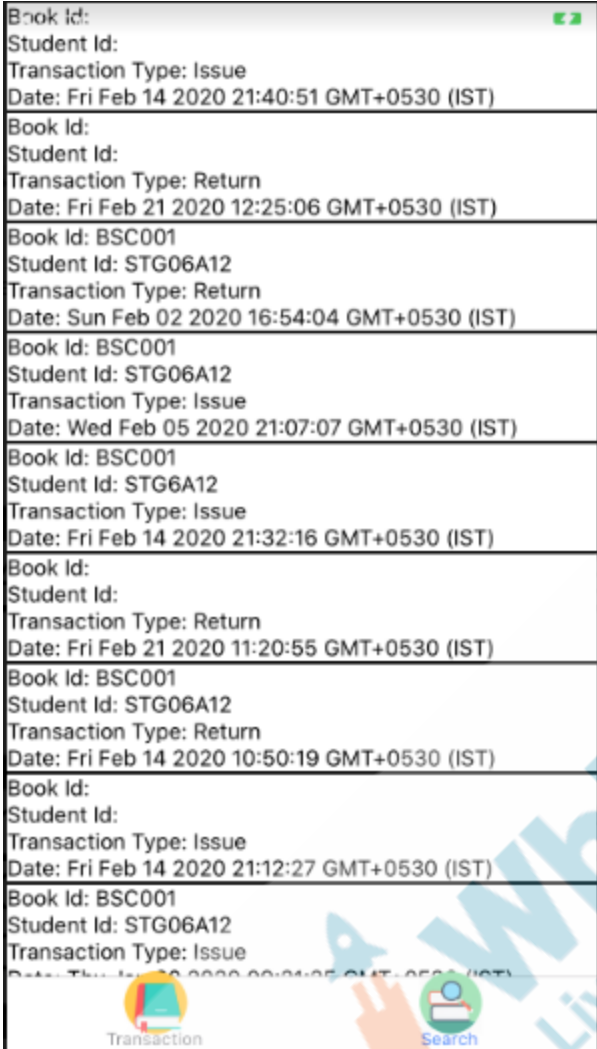
	<p>We can get only the first 10 transactions in componentDidMount. This will be quicker than getting all the transactions from the collection.</p> <p>Let's store the last transaction doc which we get inside another state called lastVisibleTransaction.</p> <p><i>Teacher writes the code.</i></p>		<p><i>The student guides the teacher and asks questions if needed.</i></p>
--	--	--	--

```
export default class Searchscreen extends React.Component {
  constructor(props){
    super(props)
    this.state = {
      allTransactions: [],
      lastVisibleTransaction: null
    }
  }

  fetchMoreTransactions = async ()=>{
```

	<p>FlatList has two more important props. onEndReached and onEndThreshold</p> <ul style="list-style-type: none"> • onEndReached can call a function to get more transaction documents after the last transaction document we fetched. • onEndThreshold defines when we want to call the function inside onEndReached prop. If onEndThreshold is 1, the function will be called when the user has completely scrolled through the list. If onEndThreshold is 0.5, the function will be called when the user is mid-way during scrolling the items. 	<p><i>The student listens and asks questions for clarification.</i></p>
	<p>Let's put the onEndReachedThreshold as 0.7 and write a function to fetch more</p>	<p><i>The student observes and helps the teacher write the code.</i></p>

	<p>transaction documents after the last transaction document we fetched.</p> <p><i>Teacher writes the code for <code>fetchMoreTransactions</code>.</i></p>	
	 <pre> return [<FlatList data={this.state.allTransactions} renderItem={({item})=>(<View style={{borderBottomWidth: 2}}> <Text>{"Book Id: " + item.bookId}</Text> <Text>{"Student id: " + item.studentId}</Text> <Text>{"Transaction Type: " + item.transactionType}</Text> <Text>{"Date: " + item.date.toString()}</Text> </View>)} keyExtractor= {(item, index)=> index.toString()} onEndReached={this.fetchMoreTransactions} onEndReachedThreshold={0.7} />]; </pre>	
	 <pre> fetchMoreTransactions = async ()=>{ const query = await db.collection("transactions").startAfter(this.state.lastVisibleTransaction).limit(10).get() query.docs.map((doc)=>{ this.setState({ allTransactions: [...this.state.allTransactions, doc.data()], lastVisibleTransaction: doc }) }) } </pre>	
	<p><i>Teacher runs and tests the code.</i></p> <p>You can see how lazy loading works. Everytime we scroll down, we see new items getting added to the list</p>	<p><i>The student observes the output.</i></p>

		
	<p>However, our user will have to scroll through all the transactions in the list to find what they are looking for.</p> <p>Do we want that?</p>	<p>ESR:</p> <p>No! We would rather have a search box where user can put the book id or student id and only the result transaction documents matching the query would be shown.</p>
	<p>Awesome. Can you take this as a challenge and do it now?</p>	<p><i>The student takes up the challenge.</i></p>
<p>Teacher Stops Screen Share</p>		

	Now it's your turn. Please share your screen with me.	
<ul style="list-style-type: none"> • Ask Student to press ESC key to come back to panel • Guide Student to start Screen Share • Teacher gets into Fullscreen 		
<p style="text-align: center;">ACTIVITY</p> <ul style="list-style-type: none"> • Build a search bar to query for book id or student id in transactions collection. • Display the results of the query in a FlatList. 		
Step 3: Student-Led Activity (15 min)		<i>The student opens the code from the previous class and sets up the project.</i>
	<i>Guide the student to use FlatList in their code to display transactions on Search Screen.</i>	<i>The student updates their code to display the transactions using FlatList on Search Screen.</i>
<pre> return [<FlatList data={this.state.allTransactions} renderItem={({item})=>{ <View style={{borderBottomWidth: 2}}> <Text>{"Book Id: " + item.bookId}</Text> <Text>{"Student id: " + item.studentId}</Text> <Text>{"Transaction Type: " + item.transactionType}</Text> <Text>{"Date: " + item.date.toString()}</Text> </View> }} keyExtractor={(item, index) => index.toString()} onEndReached={this.fetchMoreTransactions} onEndReachedThreshold={0.7} />]; </pre>		

	<p>How do you propose to start adding the search feature on the search screen?</p>	<p>ESR:</p> <p>We will have to create a search TextInput which takes book id or student id.</p> <p>After clicking the search button, a function should be called which takes the input entered by the user and queries the database.</p> <p>All the results of the query are added to the state array storing the list of items to be displayed.</p>
--	--	--

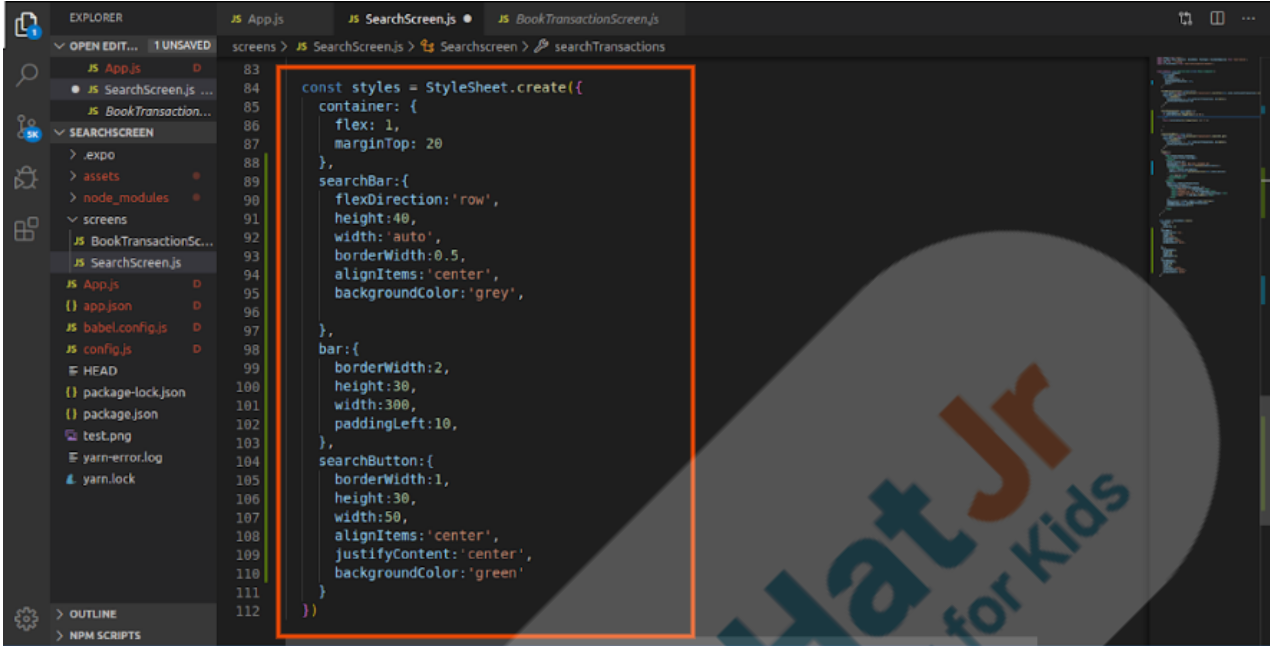


```

45  lastVisibleTransaction: doc
46  })
47  })
48  }
49  render() {
50    return (
51      <View style={styles.container}>
52        <View style={styles.searchBar}>
53          <TextInput
54            style={styles.bar}
55            placeholder="Enter Book Id or Student Id"
56            onChangeText={(text) => {this.setState({search:text})}}/>
57          <TouchableOpacity
58            style={styles.searchButton}
59            onPress={() => {this.searchTransactions(this.state.search)}}
60          >
61            <Text>Search</Text>
62          </TouchableOpacity>
63        </View>
64        <FlatList
65          data={this.state.allTransactions}
66          renderItem={({item}) => (
67            <View style={{borderBottomWidth: 2}}>
68              <Text>{"Book Id: " + item.bookId}</Text>
69              <Text>{"Student id: " + item.studentId}</Text>
70              <Text>{"Transaction Type: " + item.transactionType}</Text>
71              <Text>{"Date: " + item.date.toDate()}</Text>
72            </View>
73          )}
74          keyExtractor={({item, index}) => index.toString()}
75          onEndReached={this.fetchMoreTransactions}
76          onEndReachedThreshold={0.7}

```

	<p>Let's design the Search Bar. It will contain a TextInput and TouchableOpacity Button.</p> <p><i>Teacher helps in creating and styling the Search Bar for the student</i></p>	<p><i>The student creates the Search Bar using TextInput and TouchableOpacity Buttons.</i></p> <p><i>Student styles the Search Bar.</i></p>
--	---	---



```

const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 20
  },
  searchBar: {
    flexDirection: 'row',
    height: 40,
    width: 'auto',
    borderWidth: 0.5,
    alignItems: 'center',
    backgroundColor: 'grey',
  },
  bar: {
    borderWidth: 2,
    height: 30,
    width: 300,
    paddingLeft: 10,
  },
  searchButton: {
    borderWidth: 1,
    height: 30,
    width: 50,
    alignItems: 'center',
    justifyContent: 'center',
    backgroundColor: 'green'
  }
})

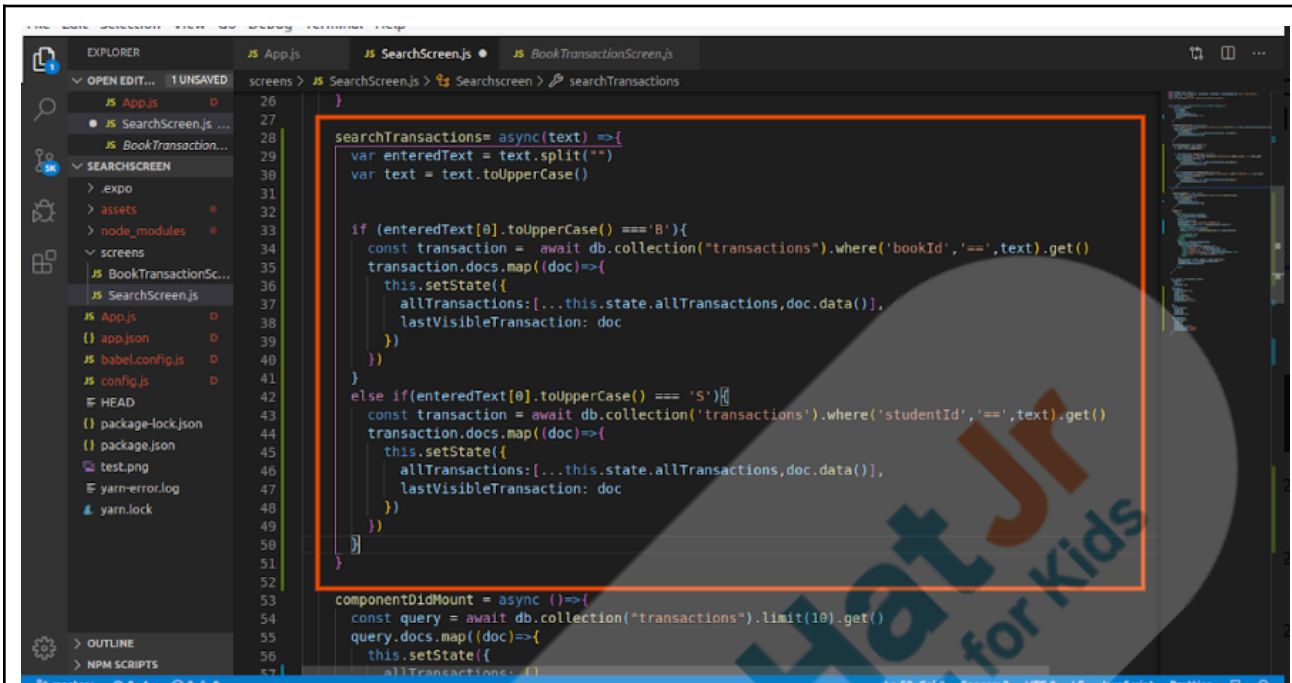
```

When search button is pressed, let's call a function `searchTransactions()`

`searchTransactions` will first check if the entered text is a book id or a student id (begins with B or S). Depending on that, the function will query the transactions collection matching the book id or student id and add them to the state `allTransactions` array.

Limit the fetched transactions to 10.

The student writes the function `searchTransactions()` which checks if the user has entered book id or student id and then fetches the documents matching the book id or student id.



```

26
27
28 searchTransactions= async(text) =>{
29   var enteredText = text.split("")
30   var text = text.toUpperCase()
31
32   if (enteredText[0].toUpperCase() === 'B'){
33     const transaction = await db.collection("transactions").where('bookId','==',text).get()
34     transaction.docs.map((doc)=>{
35       this.setState({
36         allTransactions:[...this.state.allTransactions,doc.data()],
37         lastVisibleTransaction: doc
38       })
39     })
40   }
41   else if(enteredText[0].toUpperCase() === 'S'){
42     const transaction = await db.collection('transactions').where('studentId','==',text).get()
43     transaction.docs.map((doc)=>{
44       this.setState({
45         allTransactions:[...this.state.allTransactions,doc.data()],
46         lastVisibleTransaction: doc
47       })
48     })
49   }
50 }
51
52
53 componentDidMount = async ()=>{
54   const query = await db.collection("transactions").limit(10).get()
55   query.docs.map((doc)=>{
56     this.setState({
57       allTransactions: []
58     })
59   })
60 }

```

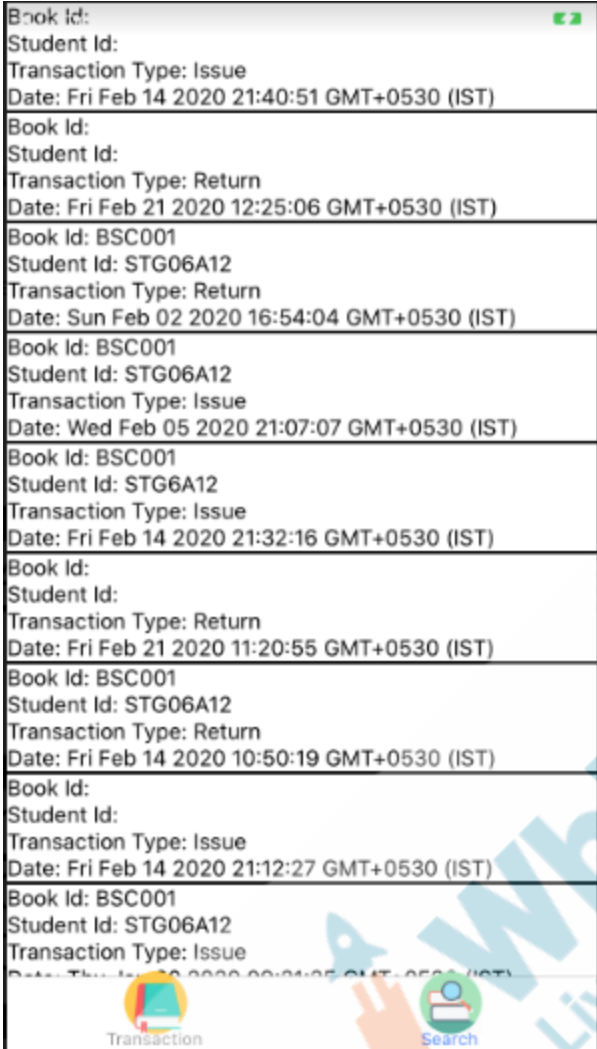
We will have to modify
fetchMoreTransactions to fetch more
transactions that match the student or
Book id.

*The student modifies the
code to fetch more
transactions through lazy
loading.*

```
screens > JS SearchScreen.js > Searchscreen
11  this.state = {
12    allTransactions: [],
13    lastVisibleTransaction: null,
14    search: ''
15  }
16
17
18  fetchMoreTransactions = async ()=>{
19    var text = this.state.search.toUpperCase()
20    var enteredText = text.split("")
21
22
23    if (enteredText[0].toUpperCase() === 'B'){
24      const query = await db.collection("transactions").where('bookId','==',text).startAfter(this.state.lastVisibleTransaction)
25      query.docs.map((doc)=>{
26        this.setState({
27          allTransactions: [...this.state.allTransactions, doc.data()],
28          lastVisibleTransaction: doc.data().bookId
29        })
30      })
31    }
32    else if(enteredText[0].toUpperCase() === 'S'){
33      const query = await db.collection("transactions").where('bookId','==',text).startAfter(this.state.lastVisibleTransaction)
34      query.docs.map((doc)=>{
35        this.setState({
36          allTransactions: [...this.state.allTransactions, doc.data()],
37          lastVisibleTransaction: doc.data().bookId
38        })
39      })
40    }
41  }
42 }
```

Let's test our code to see if we get the results we expect.

The student runs the code and checks the output.
The teacher helps in debugging the code.

		
<p style="text-align: center;"><u>FEEDBACK</u></p> <ul style="list-style-type: none"> Encourage the student to experiment with large lists using both FlatList and ScrollView. 		
<p>Step 4: Wrap-Up (5 min)</p>	<p>Let's wrap up today's class.</p> <p>What did we learn today?</p>	<p>ESR:</p> <p>We learned how to build lazy loading in our app using FlatList.</p> <p>We also learned how to build a search query box in our app.</p>

	<p>Awesome!</p> <p>We have the app almost ready now.</p> <p>In the next class we are going to work on authentication for our app so that only teachers who are authorized can use this app.</p> <p>Hope to see you in the next class then.</p> <p>Meanwhile, you can work on styling your app to make it look better.</p>	
	<p>Does the word 'Challenge' ring any bells? Well, we are talking about the upcoming Capstone challenge.</p> <p>In the next class, we will continue to build a complete library management system - Wily - which will allow teachers to issue or students to return books from a library by scanning QR codes.</p> <p>Please request your parents to join the class.</p>	
Project Overview	<p>*This Project will take only 30 mins to complete. Motivate students to try and finish it after class only.*</p> <p>BEDTIME STORIES - FLATLIST</p>	

	<p>Goal of the Project:</p> <p>In class 74, you have learnt how to display items in a list using a FlatList component. You have also learnt to build a search bar to display the list of the items queried by the user.</p> <p>In this project you have to practice and apply the concepts learned in the class and create a “BedTime Stories-FlatList” app.</p> <p>Story:</p> <p>Kareena wants to create a “BedTime Stories” App for her younger brother Joy who loves to read and listen to stories before going to sleep. Can you help her in creating this application?</p> <p>I am very excited to see your project solution and I know you will do really well.</p> <p>Bye Bye!</p>	
<div> <div>Teacher Clicks</div> <div>✕ End Class</div> </div>		
<p>Additional Activities</p>	<p>Encourage the student to write reflection notes in their reflection journal using markdown.</p> <p>Use these as guiding questions:</p> <ul style="list-style-type: none"> • What happened today? <ul style="list-style-type: none"> ○ Describe what happened ○ Code I wrote • How did I feel after the class? 	<p>The student uses the markdown editor to write her/his reflection as a reflection journal.</p>

	<ul style="list-style-type: none"> • What have I learned about programming and developing games? • What aspects of the class helped me? • What did I find difficult? 	
--	---	--

Activity	Activity Name	Links
Teacher Activity 1	Previous class code	https://github.com/whitehatjr/FirestoreQuery
Teacher Activity 2	Flatlist Documentation	https://facebook.github.io/react-native/docs/flatlist
Teacher Activity 3	Final Solution	https://github.com/whitehatjr/searchBar
Student Activity 1	Flatlist Documentation	https://facebook.github.io/react-native/docs/flatlist
Project Solution	Bedtime Stories- Flatlist	https://github.com/priyapandey2020/13d8a898ccf4a1ad158396970462275d