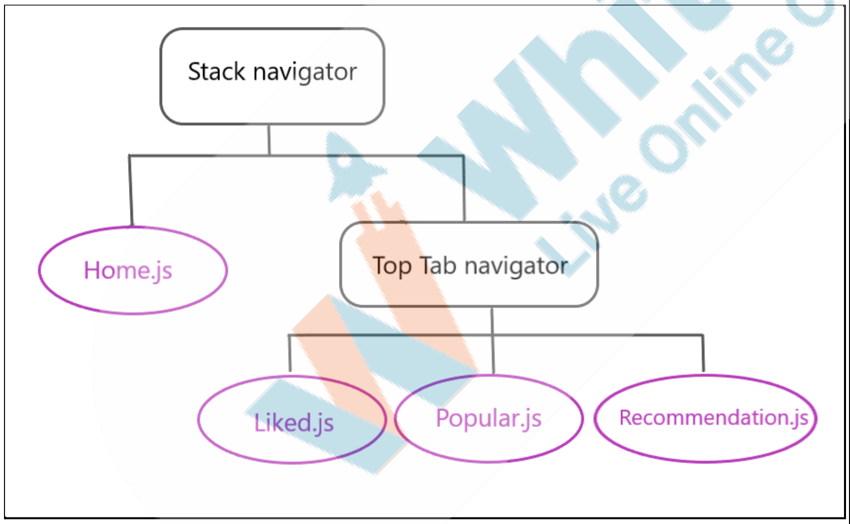
Topic	REACT NATIVE APP- 2	
Class Description	**The student will add another screen to the Mobile app where they can see the recommended and popular movies.**	
Class	**PRO C144**	
Class time	**45 mins**	
Goal	● Complete the movie recommendation app. ● Work on 3 screens to show the popular movies,the recommended movies & liked movies. ● Use the “/movies”, “/recommended” & “/liked” routes of the Flask API.	
Resources Required	● Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone ● Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up** **Teacher-Led Activity 1** **Student-Led Activity 1** **Wrap-Up**	**10 mins** **10 mins** **20 mins** **05 mins**
WARM-UP SESSION - 10 mins		
Teacher Starts Slideshow **Slide # to #** <**Note**: Only Applicable for Classes with VA>		

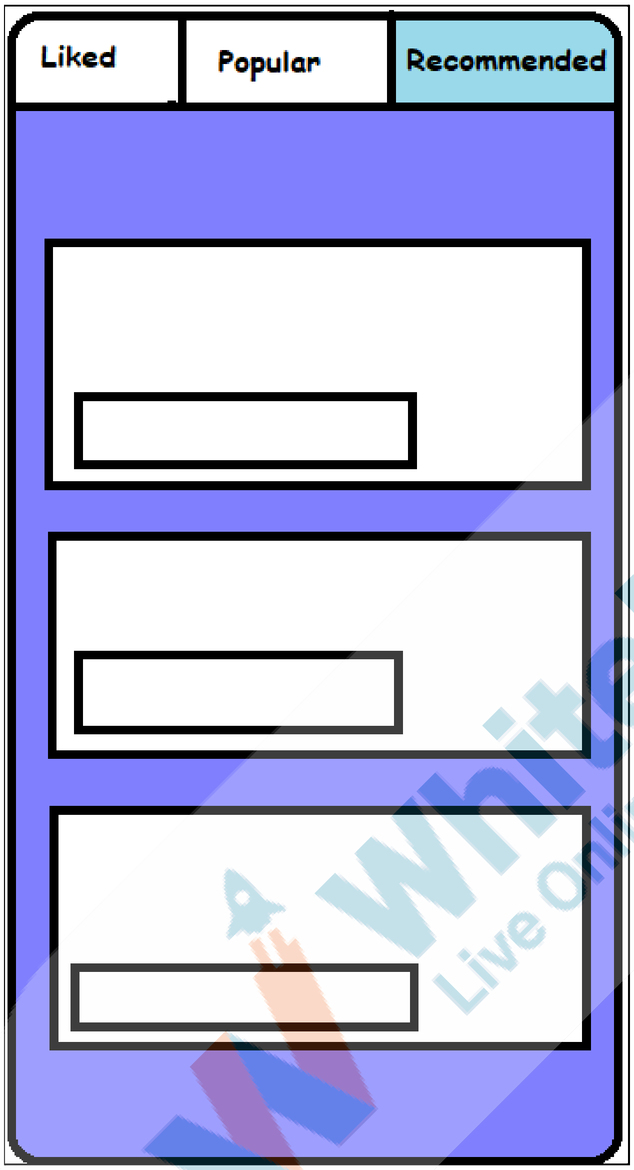
Refer to speaker notes and follow the instructions on each slide.	
Teacher Action	**Student Action**
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** ● Greet the student. ● Revision of previous class activities. ● Quizzes.	**ESR**: Hi, thanks! Yes, I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session **Slide # to #** <**Note**: Only Applicable for Classes with VA>	
Activity Details **Following are the session deliverables:** ● Appreciate the student. ● Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
Teacher Ends Slideshow	
TEACHER-LED ACTIVITY - 10 mins	
Teacher asks student to initiate Screen Share	
ACTIVITY ● **Talk about the other screens and the functionality of the app.**	

- **List down all the user actions and app behavior for the screen.**
- **Discuss the logic or flow of the program to achieve these user behaviors.**

Teacher Action	**Student Action**
Do you remember what we did in the last class?	**ESR:** Yes. We created the **home screen** for our movie recommendation mobile app.
Great! What did we add to this screen?	**ESR:** We rendered the information about the movie returned by the flask API. 1. Movie title 2. Poster 3. Duration & release date 4. Star rating 5. Like, dislike & notWatched button
Well done! Also, what routes we used from the Flask API in our first screen of the app?	**ESR:** Yes. The routes used in the first screen were- 1. **"/movies"** 2. **"/like"** 3. **"/dislike"** 4. **"/did_not_watch"**
Which routes we did not use yet?	**ESR:** Yes. The unused routes were- 1. **“/liked”** 2. **"/popular_movies"**

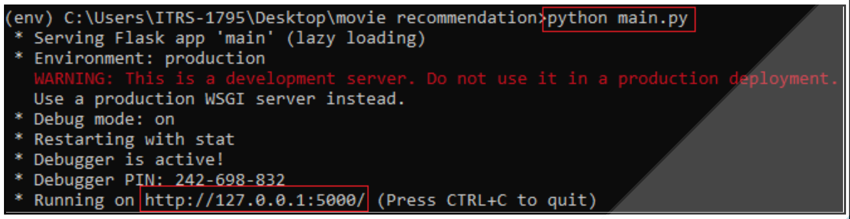
In this class, we will create three more screens which will make use of these 3 routes — **“/liked”, "/popular_movies", "/recommended_movies”.** One screen for displaying **popular movies**, one for displaying **recommended movies** and another for displaying the **liked movies.** These three screens will be part of the Tab navigator that we already have in our previous class code. *Teacher revises the code and makes sure the student understands it.* *Teacher can show this image to revise navigators inside the app.* *This is also available in Student Activity 3.*  Stack navigator Home.js Top Tab navigator Liked.js Popular.js Recommendation.js Let’s start by thinking about the features and components for our second screen.	3. **"/recommended_movies"**

How many tabs do we have in the second screen? Where will we get the data for these screens? Perfect! Let's see the wireframe of the second and then we will start building it. It can look somewhat like the graphic shown below. Wireframe:	**ESR:** 3 tabs which will display a list of **liked movies, popular movies** and **recommended movies** in the form of lists. **ESR:** We will get it from the flask API.

 Click here to view the wireframe of the app. *This is also available in student activity 4. You can design your own UI as well.* *Note: This is just a sample UI. You can design your own UI as well.*	

But before starting our flask API should be up and running.

Student opens the previous class code. Then, student opens the terminal and activates the virtual environment. Then, student runs it on the terminal by using the command "python main.js".



```
(env) C:\Users\ITRS-1795\Desktop\movie recommendation>python main.py
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 242-698-832
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now there is one challenge, this is running on the localhost, which is accessible only from your computer. But we might want to test our code on the phone as well.

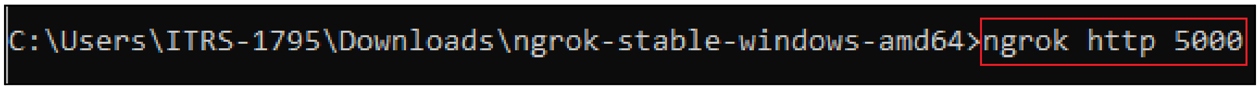
ngrok will help us to expose our local server to the internet with minimal effort.

Teacher guides the student to open ngrok from his system. If a student doesn't have ngrok downloaded, ask the student to click on this Student activity 4 to download.

Open the **ngrok** application. Once your server is up and running, open the **ngrok** application. You will see that your command prompt will be opened in that directory.

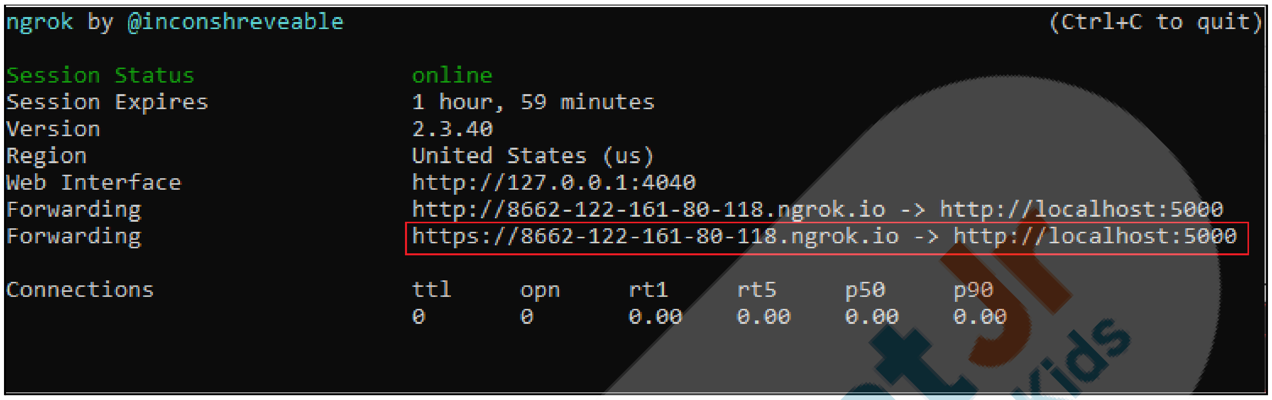
Remember, your flask server is running a port **5000.** To bring it online, use the command **ngrok http port_number,** or in our case,

ngrok http 5000



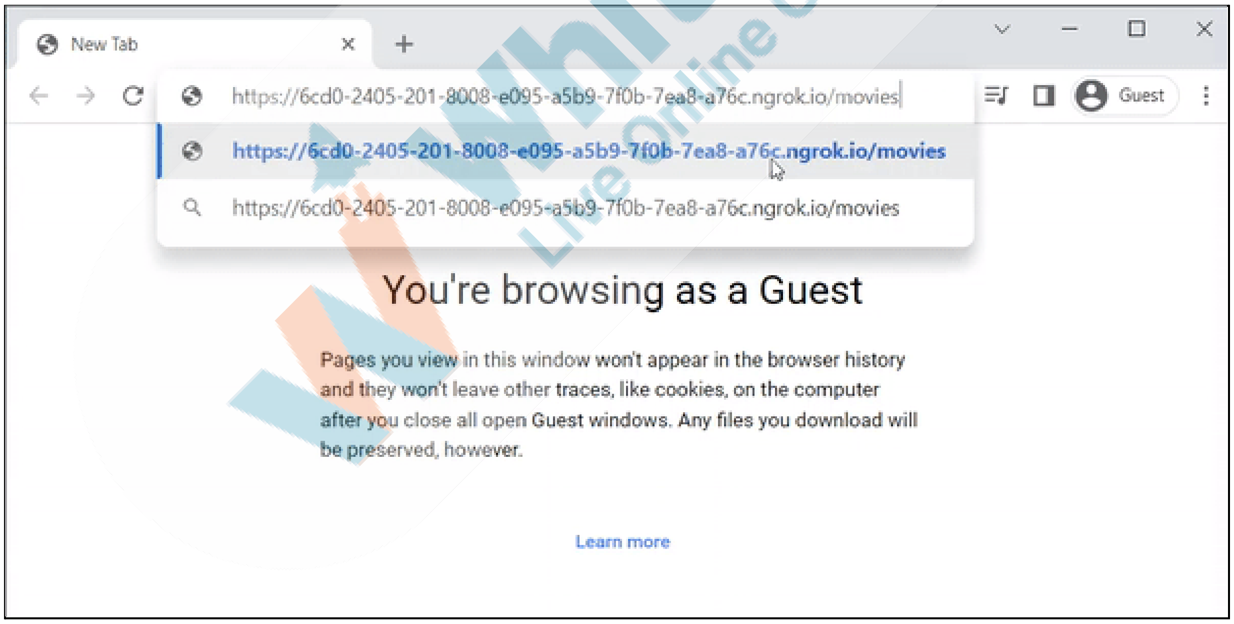
```
C:\Users\ITRS-1795\Downloads\ngrok-stable-windows-amd64>ngrok http 5000
```

As soon as you run this command, you will see that your local server is now online and can be accessed using 2 links.



```
ngrok by @inconshreveable                                                  (Ctrl+C to quit)

Session Status                online
Session Expires               1 hour, 59 minutes
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://8662-122-161-80-118.ngrok.io -> http://localhost:5000
Forwarding                    https://8662-122-161-80-118.ngrok.io -> http://localhost:5000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Let's verify the working of these links. Copy the **secure link** and in your web browser, call one of the APIs, using the command. This link changes every time you run the **ngrok http 5000** command.



```
https://6cd0-2405-201-8008-e095-a5b9-7f0b-7ea8-a76c.ngrok.io/movies
https://6cd0-2405-201-8008-e095-a5b9-7f0b-7ea8-a76c.ngrok.io/movies
https://6cd0-2405-201-8008-e095-a5b9-7f0b-7ea8-a76c.ngrok.io/movies
```

You're browsing as a Guest

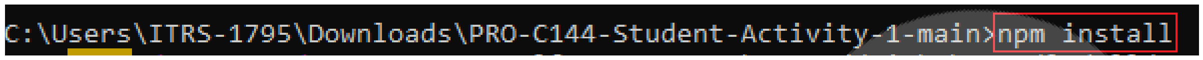
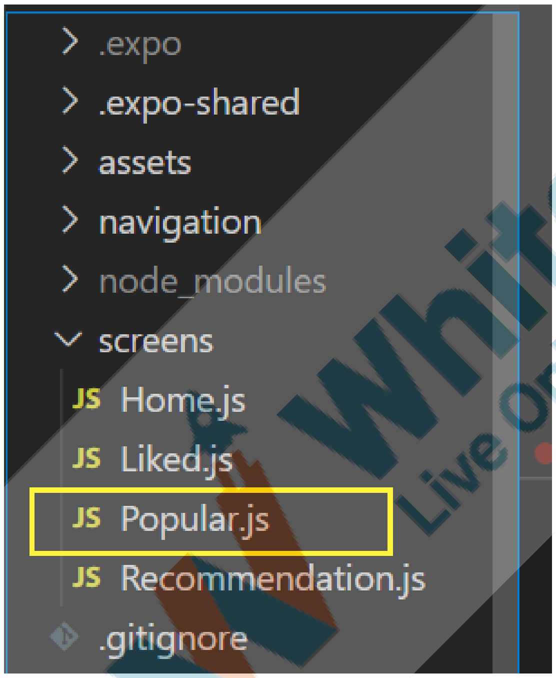
Pages you view in this window won't appear in the browser history and they won't leave other traces, like cookies, on the computer after you close all open Guest windows. Any files you download will be preserved, however.

Learn more

Click here to view the reference video.

WhiteHat Jr
Live online Coding for Kids

Let’s start coding.	
Student keeps sharing his Screen	
So now it’s your turn. Please share your screen with me.	
Teacher Starts Slideshow **Slide # to #** <**Note**: Only Applicable for Classes with VA> Refer to speaker notes and follow the instructions on each slide.	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	
Teacher Ends Slideshow	
STUDENT-LED ACTIVITY - 20 mins	
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start Screen Share.** • **The teacher gets into Full Screen.**	
Student Initiates Screen Share	
ACTIVITY • **Write code to add the functionality in the app.** • **Test and debug the code.**	
Teacher Action	**Student Action**
Teacher asks the student to open code from the previous class.	

This can also be cloned from [Student activity 1](). Make sure that the student installs the dependencies by running the command "npm install" if he has downloaded the code today.	
 `C:\Users\ITRS-1795\Downloads\PRO-C144-Student-Activity-1-main>npm install`	
Define Popular.js: 1. Open this folder with the **Visual Studio Code editor** and click on the **Popular.js** file listed under the **screens** folder. 	
2. Once you open this file, you will see some components that are already imported from different modules. Also in the **Popular.js** file, we have a prewritten **StyleSheet** object named **styles,** holding the styling data for different components.	
3. There is a prewritten **PopularMoviesScreen** class, with a **constructor** method, which has a **state object** named as **data (empty array)**. This **state object** will store all the data that we are going to	

receive from our flask API.

We will also include another state name **ngrok_url** and will add the ngrok url here

```
export default class PopularMoviesScreen extends Component {
  constructor(props) {
    super(props);
    this.state = {
      data: [],
      ngrok_url: ""
    };
  }
```

4. Let's define a method named **getData,** which will make a **GET** request to our server on the **'/popular_movies' route** and fetch the top 20 movies.

   Once we get a response from the server, we will update the **data** object.

   Define **getData()** function:

   a. Declare a **const** named **url** and assign the route from which we want to fetch data. In this case, it is **this.state.ngrok_url + "/popular_movies"**.

   ```
     const url = this.state.ngrok_url+"/popular_movies";
   ```

   b. Use **axios.get(url)** to fetch the data from the url.

   ```
   axios
   .get(url)
   ```

   c. Mention that **.then()** function after this. This function will define what to do when the data is fetched. Here, we will write the code to store the response data in the **movieDetails** state.

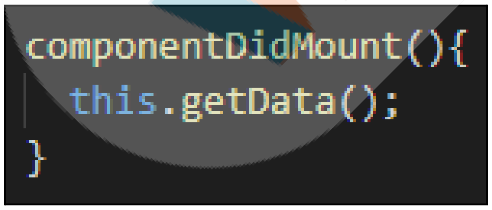
```
    axios
      .get(url)
```

```
      .then(async (response) => {
        this.setState({ data: response.data.data });
      })
```

d. We will also add a **.catch()** function in case any error arises. Here, we will simply **console.log()** the error message.

```
getData = () => {
  const url = this.state.ngrok_url+"/popular_movies";
  axios
    .get(url)
    .then(async (response) => {
      this.setState({ data: response.data.data });
    })
    .catch((error) => {
      console.log(error.message);
    });
};
```

5. Now this method should be called as soon as the **PopularMovieScreen** component is mounted on the screen.



```
componentDidMount(){
  this.getData();
}
```

As soon as the **getData** method is called, we will have the data for the top 20 popular movies. It will be stored as an array in the state named **data.** In order to render these movie data on our app, we will use the **Flatlist** component. **Flatlist** component has 3 important props, ● **data** : Data which is to be displayed using the **Flatlist** component. ● **keyExtractor :** This method is used to extract a unique key for a given item at the specified index. ● **renderItem** : This method takes an item from the data, and renders it to the list.	

6. Let's define the **keyExtractor()** function first.
   Through the **keyExtractor()** function , we will pass each item from the given array and its index. Then, we will convert this index into string and return it from the **keyExtractor().**

```
  keyExtractor = (item, index) => index.toString();
```

7. Now, think about the **renderItems()** function. Can you tell me how we should define it?	**ESR:** We will pass each item and its index through this function and return the components as we want to show it on the screen.

```
renderItems = ({ item, index }) => {
    return (
      <View style={styles.cardContainer}>
        <Image
          style={styles.posterImage}
```

```
            source={{ uri: item.poster_link }}
          ></Image>
          <View style={styles.movieTitleContainer}>
            <Text style={styles.title}>{item.original_title}</Text>
            <View style={{flexDirection:"row"}}>
              <Text style={styles.subtitle}>{item.duration} mins |
</Text>
              <Star score={item.rating} style={styles.starStyle}/>
            </View>
          </View>
        </View>
      );
    };
```

8. Finally it's time to complete the **render** method, which will **return** the styled components for our react native app.

   Within this **return** method, we already have the **ImageBackground** component. We will define the **Flatlist** component to render the list of top 20 popular movies.

   Here, we will use the following three props for the **FlatList** -
   a. **data** : use the state **data**.
   b. **keyExtractor** : call the **keyExtractor()** function here.
   c. **renderItem** : call the **renderItem()** function here.

```
render() {
    const { data } = this.state;
    return (
      <View style={styles.container}>
```

```
        <ImageBackground
          source={require("../assets/bg.png")}
          style={{ flex: 1 }}
        >
          <FlatList
            data={data}
            keyExtractor={this.keyExtractor}
            renderItem={this.renderItems}
          />
        </ImageBackground>
      </View>
    );
  }
```

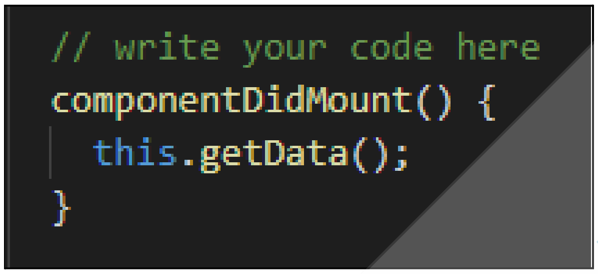
Define Recommendation.js:

a. Similar code has to be written for the **Recommendation.js** file. Go to the **Recommendation.js** file. You can reuse the code from **Popular.js** here.

b. Define the **getData()** function. It will be the same as the **getData()** function from the **Popular.js.** You only need to change the **url** (API route) to **"/recommended_movies"**

```
getData = () => {
   const url =this.state.ngrok_url+"/recommended_movies";
   axios
     .get(url)
     .then(async (response) => {
       this.setState({ data: response.data.data });
     })
     .catch((error) => {
        console.log(error.message);
    });
};
```

c. Also, call this method when the component is mounted.



```
// write your code here
componentDidMount() {
  this.getData();
}
```

d. Define the **keyExtractor** and the **renderItems** method for the **Flatlist** component (Same as **Popular.js**).

```
  keyExtractor = (item, index) => index.toString();

  renderItems = ({ item, index }) => {
    return (
      <View style={styles.cardContainer}>
        <Image
          style={styles.posterImage}
          source={{ uri: item.poster_link }}
        ></Image>
        <View style={styles.movieTitleContainer}>
          <Text style={styles.title}>{item.original_title}</Text>
          <View style={{flexDirection:"row"}}>
            <Text style={styles.subtitle}>{item.duration} mins |
</Text>
```

```
            <Star score={item.rating} style={styles.starStyle}/>
          </View>
        </View>
      </View>
    );
  };
```

e. Finally, let’s write the **render** method, which will **return** the styled components for our react native app. (Similar to **Popular.js**)

```
render() {
    const { data } = this.state;
      return (
        <View style={styles.container}>
          <ImageBackground
            source={require("../assets/bg.png")}
            style={{ flex: 1 }}
          >
            <FlatList
              data={data}
              keyExtractor={this.keyExtractor}
              renderItem={this.renderItems}
            />
          </ImageBackground>
        </View>
      );
  }
```

<u>Define Liked.js:</u>

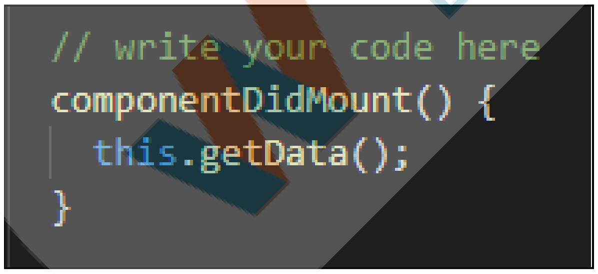
1. Similar code has to be written for the **Liked.js** file.

Go to the **Liked.js** file. You can reuse the code from **Popular.js** here.

2. Define the **getData()** function. It will be the same as the **getData()** function from the **Popular.js.** You only need to change the **url** (API route) to **“/liked”.**

```
getData = () => {
   const url =this.state.ngrok_url+"/liked";
   axios
     .get(url)
     .then(async (response) => {
       this.setState({ data: response.data.data });
     })
     .catch((error) => {
       console.log(error.message);
     });
 };
```

3. Also, call this method when the component is mounted.



```
// write your code here
componentDidMount() {
  this.getData();
}
```

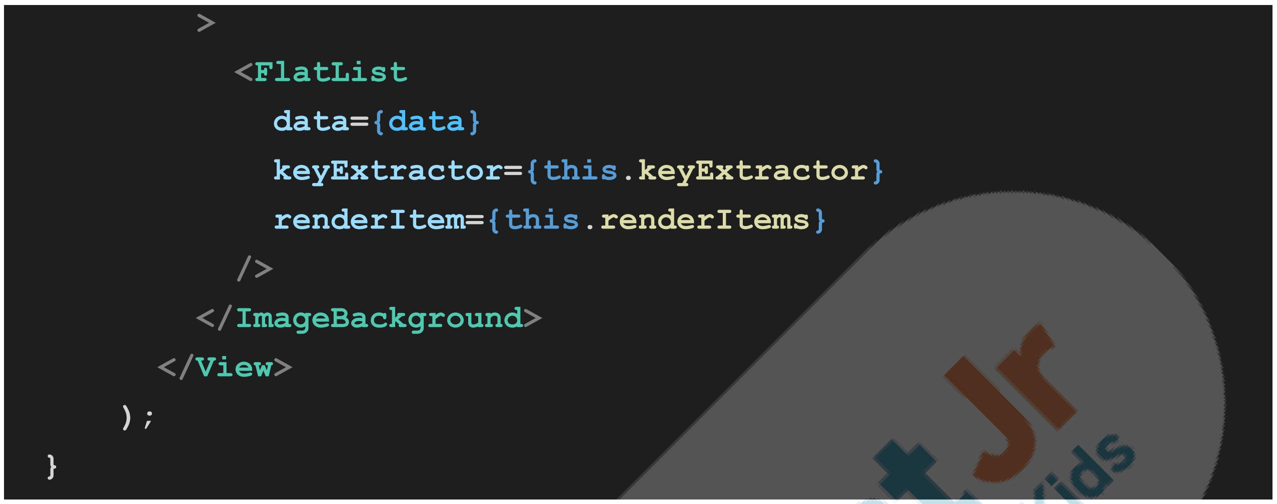
4. Define the **keyExtractor** and the **renderItems** method for the **Flatlist** component (Same as **Popular.js**).

```
 keyExtractor = (item, index) => index.toString();
```

```
renderItems = ({ item, index }) => {
  return (
    <View style={styles.cardContainer}>
      <Image
        style={styles.posterImage}
        source={{ uri: item.poster_link }}
      ></Image>
      <View style={styles.movieTitleContainer}>
        <Text style={styles.title}>{item.original_title}</Text>
        <View style={{flexDirection:"row"}}>
          <Text style={styles.subtitle}>{item.duration} mins |
</Text>
          <Star score={item.rating} style={styles.starStyle}/>
        </View>
      </View>
    </View>
  );
};
```

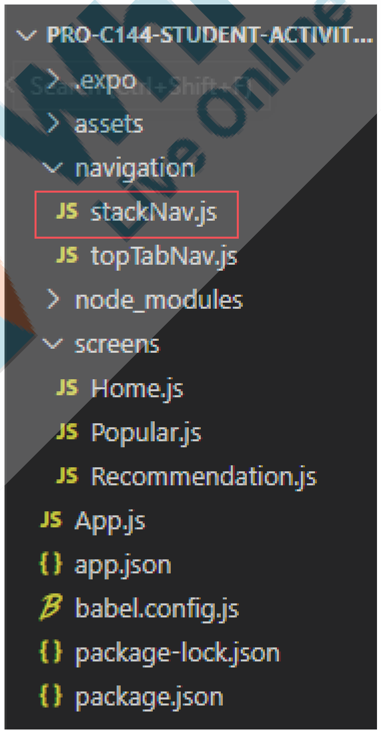
5. Finally, let's write the **render()** method, which will **return** the styled components for our react native app. (Similar to **Popular.js**)

```
render() {
   const { data } = this.state;
     return (
       <View style={styles.container}>
         <ImageBackground
           source={require("../assets/bg.png")}
           style={{ flex: 1 }}
```

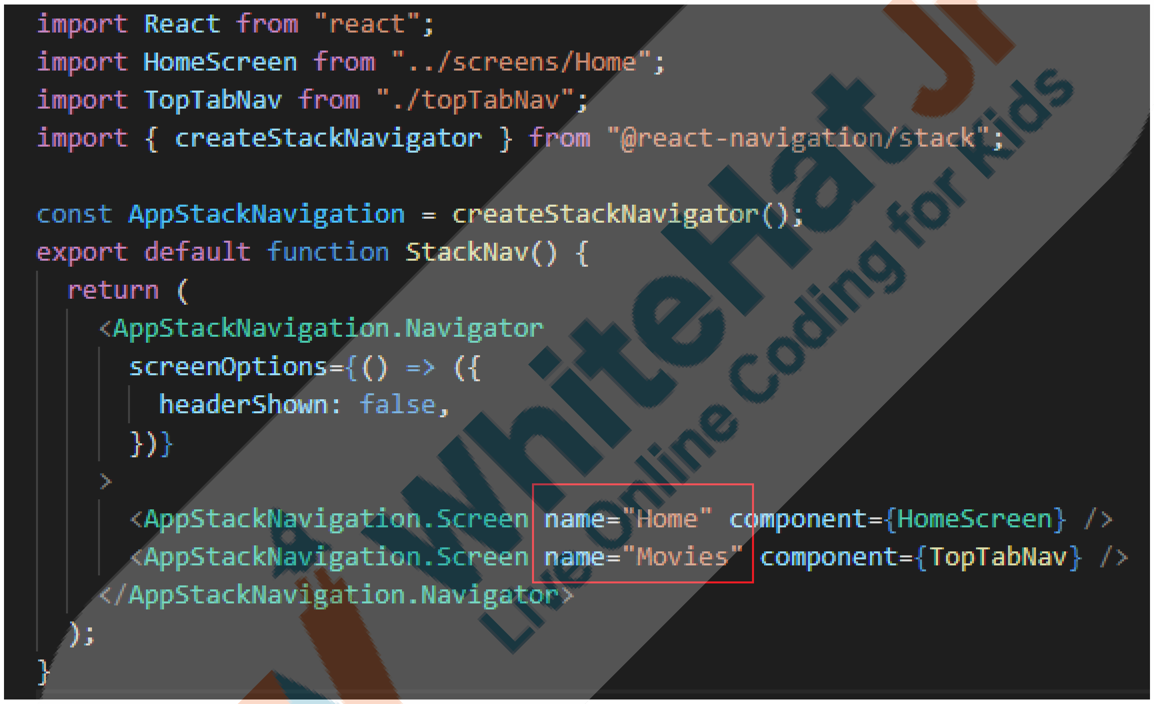

```
            >
              <FlatList
                data={data}
                keyExtractor={this.keyExtractor}
                renderItem={this.renderItems}
              />
            </ImageBackground>
          </View>
        );
    }
```

We have created all the files. Now we can navigate between these screens using the **top tab navigator**. The code for the navigation part is already pre-written.

Go to the **stackNav.js** file under the **navigation** folder.

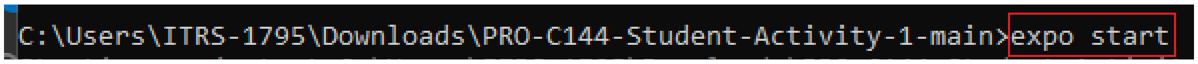


You will see we are navigating between 2 screens. One is the Homescreen, named as **“Home”,** and the other one is the **TopTabNav** screen, named as **“Movies”. TopTabNav** holds the **Popular.js**, **Recommended.js** and **Liked.js** screen. *Note: In class 143, we created an icon on our header component, which would take us to the “Movie” screen, when pressed.*	



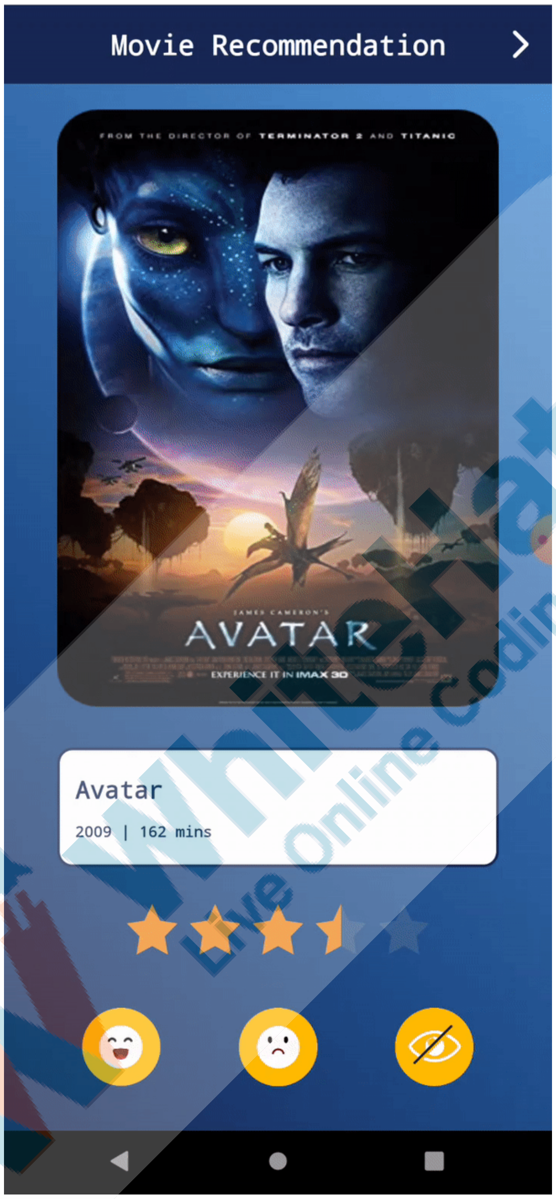
```
import React from "react";
import HomeScreen from "../screens/Home";
import TopTabNav from "./topTabNav";
import { createStackNavigator } from "@react-navigation/stack";

const AppStackNavigation = createStackNavigator();
export default function StackNav() {
  return (
    <AppStackNavigation.Navigator
      screenOptions={() => ({
        headerShown: false,
      })}
    >
      <AppStackNavigation.Screen name="Home" component={HomeScreen} />
      <AppStackNavigation.Screen name="Movies" component={TopTabNav} />
    </AppStackNavigation.Navigator>
  );
}
```

To run you code, save all the changes, and in your command prompt, write the command **expo start** You will see a **QR code** generated. Scan it with the help of your mobile and you will be able to see the **Homescreen** of your app.	



```
C:\Users\ITRS-1795\Downloads\PRO-C144-Student-Activity-1-main>expo start
```

Reference output:



Click here to view the output reference video.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins

Teacher Starts Slideshow
Slide # to #
<**Note**: Only Applicable for Classes with VA>

Activity details

Following are the WRAP-UP session deliverables:
- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ
Click on In-Class Quiz

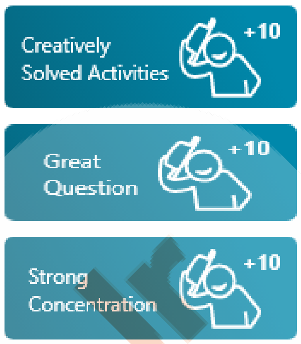
Continue WRAP-UP Session
Slide # to #
<**Note**: Only Applicable for Classes with VA>

Activity Details

Following are the session deliverables:
- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

<u>FEEDBACK</u>
- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

Teacher Action	**Student Action**
You get “hats-off” for your excellent work!	*Make sure you have given at least 2 hats-off during the*

From the next class onwards, we will start with the AR/VR module. In this module, we will learn about Augmented reality (AR) and virtual reality (VR). We will make different projects using AR/VR like- Solar system model, Shooting game etc. Till then keep exploring and try new projects!	*class for:*  Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Flask API code	https://github.com/procodingclass/PRO-C142-Reference-Code
Teacher Activity 2	Previous Class Code (React native app)	https://github.com/procodingclass/PRO-C144-Student-Boilerplate
Teacher Activity 3	Reference Code	https://github.com/procodingclass/PRO-C144-Reference-Code
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/d43f8351-7a20-4734-a606-9817dcec510f.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C144-Project_Solution
Teacher Reference 3	Visual-Aid	Will be added after VA creation
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/8580cf55-7bf0-4a34-918a-f311465d9db1.pdf
Student Activity 1	Previous Class Code (React native app)	https://github.com/procodingclass/PRO-C144-Student-Boilerplate
Student Activity 2	Flask API code	https://github.com/procodingclass/PRO-C142-Reference-Code
Student Activity 3	Structure of navigator in the app	https://s3-whjr-curriculum-uploads.whjr.online/df5845c6-2267-4e30-a970-f2629c6b6f79.png
Student Activity 4	Wireframe for screen 2	https://s3-whjr-curriculum-uploads.whjr.online/41d8193f-2b23-4f04-b0b3-d5077ab06d34.png