Topic	REACT NATIVE APP - 1	
Class Description	**The student will create a mobile application using react native. They will revise how to use the get() method to fetch information from the flask API. In addition to that, they will mark a movie either as a liked movie, disliked movie or as a movie which they haven't watched yet.**	
Class	**PRO C143**	
Class time	**45 mins**	
Goal	● Create a React Native app. ● Revise get() method and use it to fetch information from the flask API ● Mark movies in three categories - liked, disliked and did not watch.	
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**	**05 mins** **10 mins** **25 mins** **05 mins**

WARM-UP SESSION - 05 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? ***Note**: Encourage the student to give answers and be more involved in the discussion.* **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

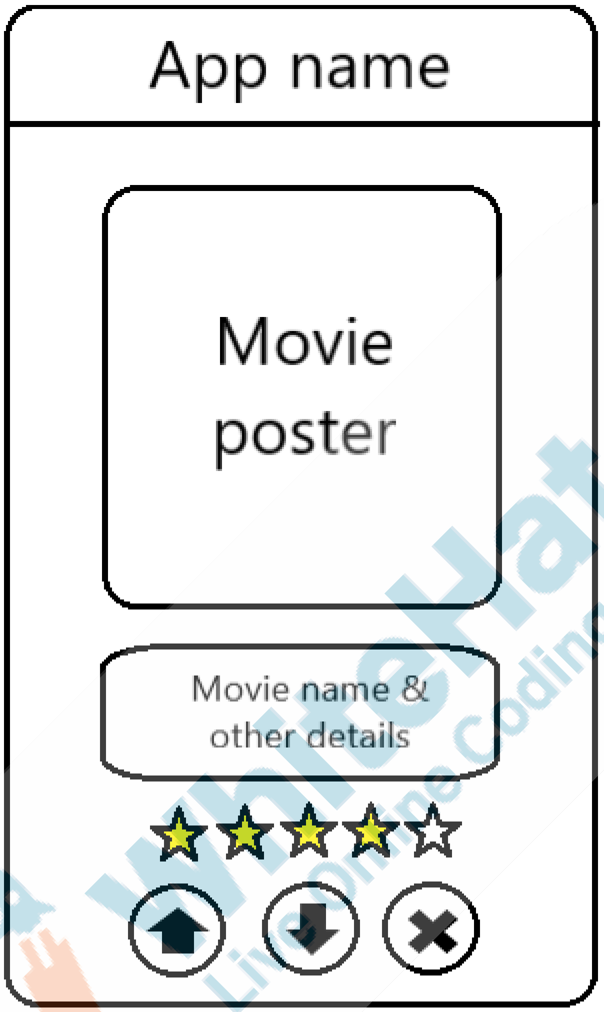
Following are the session deliverables:

- 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 ● **List down all the user actions and wireframe for the home screen.** ● **Discuss the logic or flow of the program to achieve these user behaviors.**	
Teacher Action	**Student Action**
In the last class we finished creating all the flask APIs required for the second screen of our app. Now we are in the final phase of our project, which is to create a **movie recommendation** mobile application using **React Native**. But before that let's revise what routes we created in the Flask API. Can you name the routes created in the **main.py**? Great! We will focus on the "**/movies**", "**/like**", "**/dislike**",	**ESR:** Yes. The routes were- 1. "**/movies**" 2. "**/like**" 3. "**/dislike**" 4. "**/did_not_watch**" 5. "**/liked**" 6. "**/popular_movies**" 7. "**/recommended_movies**"

"/did_not_watch" routes today. What was the “/movies” route returning? Perfect! What about the "/like", "/dislike", "/did_not_watch" routes? Exactly! The first movie from the list of movies will be added to the liked_movies list if the “/like” route is chosen. It will also remove this movie from the main movie list. This is similar for “/dislike” and “did_not_watch” routes. We will use these four routes today to create the first screen of our app.	**ESR:** It was returning the first movie from the list of movies. **ESR:** These routes will take the first movie from the list of movies and will add it to the liked_movies, did_not_like_movies or did_not_watch list depending on which route you have chosen. It will also remove this first movie from the entire movie list, so that you see the next movie’s information once a movie is marked.
In the first screen of our app, we want to show the movie details that are fetched by the "/movies" route.	**ESR:** *The student talks about the user*

Let’s discuss the wireframe of the first screen of our movie recommendation app. Can you tell me what should be there on the homescreen of your app? Yes! What Components do you think we will need for that? Very good! We will also add three buttons: ● Like button ● Dislike button ● Did not watch button These will make use of the "/like", "/dislike", "/did_not_watch" routes respectively.	*actions and app behavior for the home screen of the app.* **ESR:** It should have the name of the movie, the poster, ratings etc. **ESR:** We will need <Text>, <Image> components. Also, for ratings we will need a rating-related component.
It could look somewhat similar to the graphic shown below. Wireframe:	

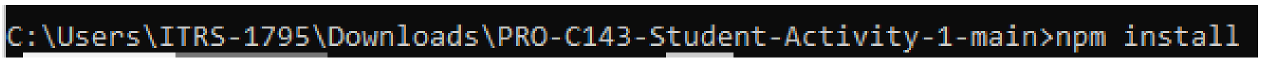


Click here to view the wireframe of the app.

This is also available in student activity 5. You can design your own UI as well.

Note: This is just a sample UI. You can design your own UI as well.

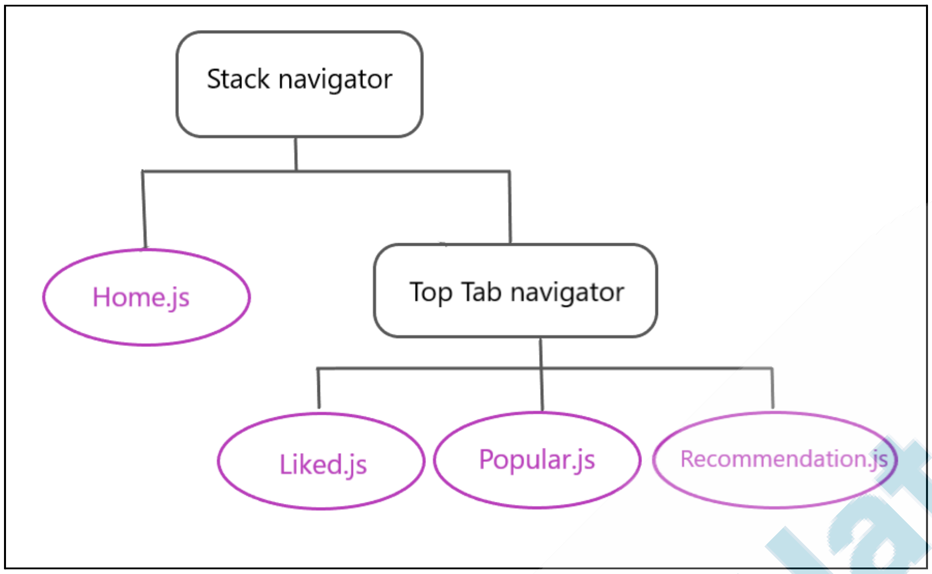
Teacher guides the student to share his screen.
Teacher guides the student to download the previous class code from Student Activity 1.
Teacher guides the student to download the boilerplate code from Student Activity 2.
Once the boilerplate code is downloaded, the student opens the command prompt and the "npm install" command in the project directory.



```
C:\Users\ITRS-1795\Downloads\PRO-C143-Student-Activity-1-main>npm install
```

Teacher can carry on with the next section of the class while it is being installed.

Let's observe the boilerplate code now. *Teacher guides the student to open the boilerplate code on the Visual Studio code editor.* In the boilerplate code, there are four screens - Home.js, Liked.js, Popular.js, Recommendations.js. We have the basic code structure already written for these screens. The modules are already installed. Also, the stylesheets are provided so that you can focus on the more important tasks. You can change the stylesheet if you want once our tasks for the class are completed and we have seen the output. Till then, we will use the given Stylesheet. We also have the navigators defined for our app. *Teacher asks the student to open the image from Student Activity 3.*	

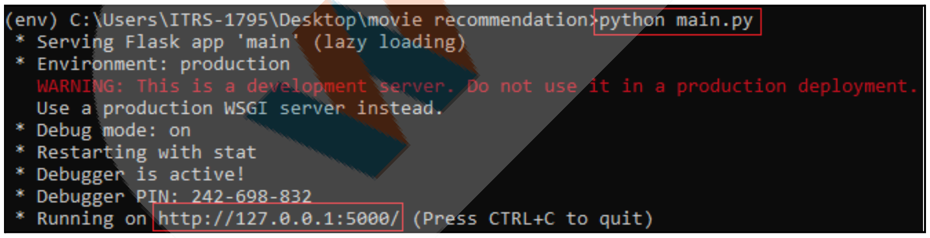


Note: Let the student think and help when required.

As we had discussed, today we will work on the **Home.js** only.

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

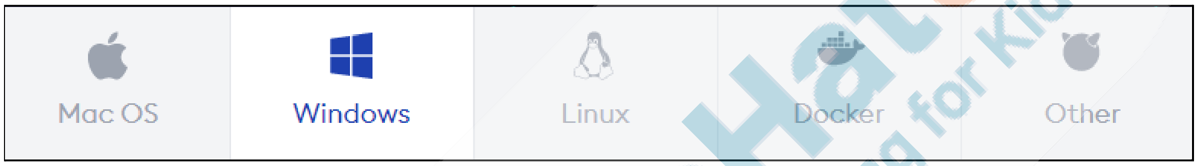
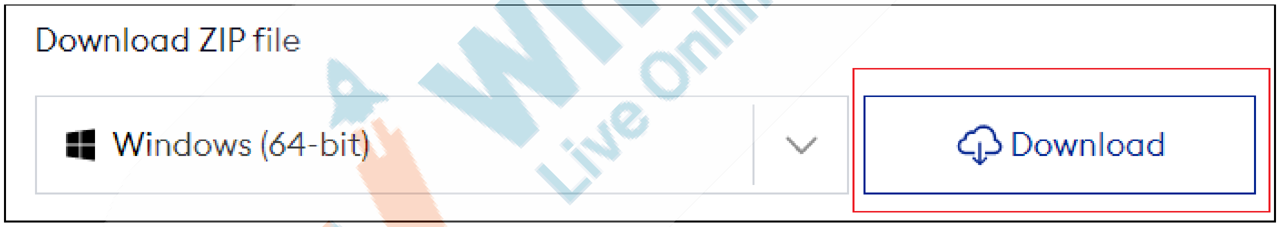
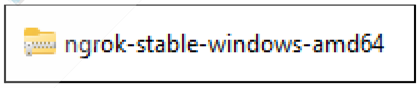
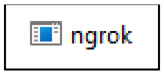
Student opens the previous class code. Then, student opens the terminal and activates the virtual environment. Then, student runs it on the terminal by using the command "python main.js".



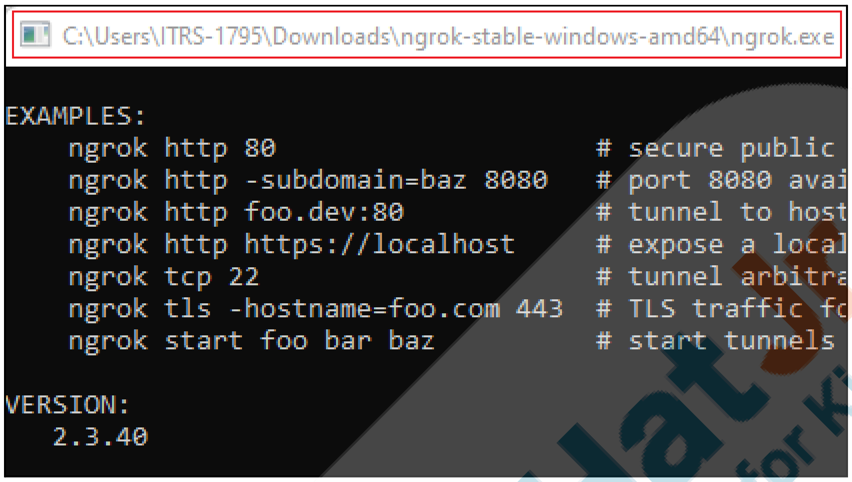
```
(env) C:\Users\ITRS-1795\Desktop\movie recommendation>python main.py
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with 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.

ESR: We can use

Can you tell me how we can overcome this challenge? Great, that is correct. **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 and select your operating system, if it's Windows, Mac, Linux etc.*	ngrok just like we did in the planet world app.
	
I have a **Windows** OS, so I will select that. Next, select if you have **32 bit Windows** or a **64 bit** one, and then click on the **Download** button.	
	
A **zipped file** will be downloaded. Extract it.	
	
When you open the uncompressed folder, you will see 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.



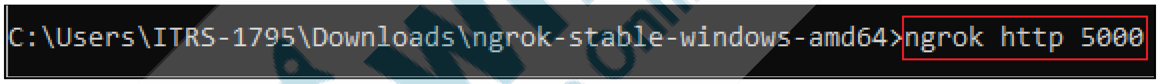
```
C:\Users\ITRS-1795\Downloads\ngrok-stable-windows-amd64\ngrok.exe

EXAMPLES:
    ngrok http 80                    # secure public
    ngrok http -subdomain=baz 8080   # port 8080 avai
    ngrok http foo.dev:80            # tunnel to host
    ngrok http https://localhost     # expose a local
    ngrok tcp 22                     # tunnel arbitra
    ngrok tls -hostname=foo.com 443  # TLS traffic fo
    ngrok start foo bar baz          # start tunnels

VERSION:
   2.3.40
```

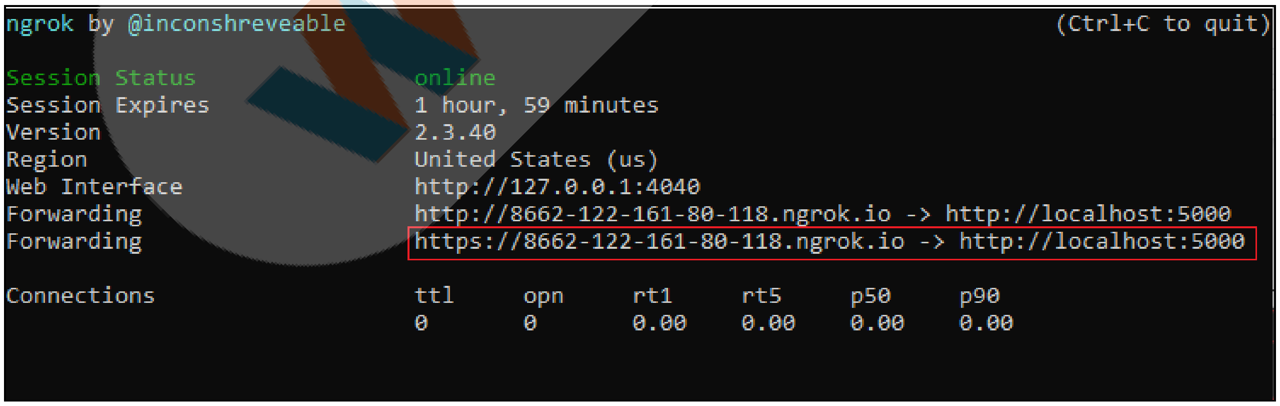
Remember, your flask server is running a port **5000.** To bring it online, use the command **ngrok http port_number,** or in our case,

ngrok http 5000



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

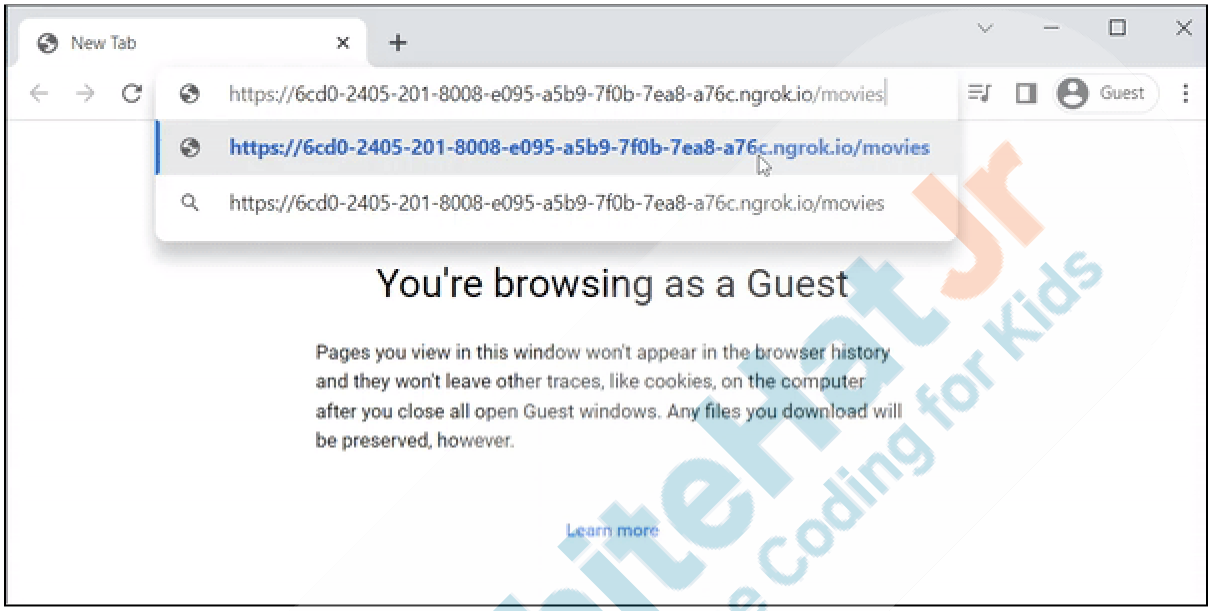
As soon as you run this command, you will see that your local server is now online and can be accessed using 2 links,



```
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.	
 New Tab 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.	
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 - 25 mins

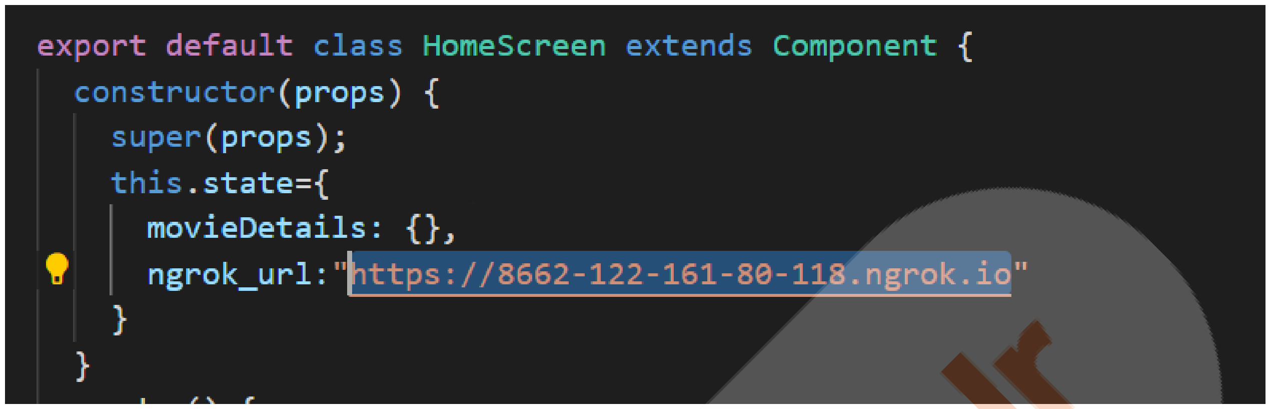
- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Full Screen.**

Student Initiates Screen Share

ACTIVITY

- **Write code to add the functionality in the app.**
- **Test and debug the code.**

Teacher Action	Student Action
Great, now your server is online. Next step is to start coding the app. Open the **Home.js** file listed under the **screens** folder in the downloaded boilerplate code. What do we have here? There is a **constructor** method in this class, which has 2 **states** named as **movieDetails** and **ngrok_url.** The **movieDetails state** will store all the data that we are going to receive from our flask API. We will paste our ngrok link in the **ngrok_url** state. *Note : You have to replace the url with your own url generated by the ngrok software.*	**ESR:** We have a prewritten **Homescreen** class



```
export default class HomeScreen extends Component {
  constructor(props) {
    super(props);
    this.state={
      movieDetails: {},
      ngrok_url:"https://8662-122-161-80-118.ngrok.io"
    }
  }
```

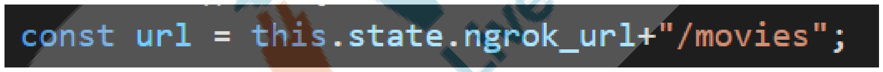
First, we need to fetch our movie data from the API.

Let's define a method named **getMovie(),** which will make a **GET** request to our server on the **'/movies' route** and fetch the first movie and all its details.

Once we get a response from the server, we can store it in the **movieDetails** state.

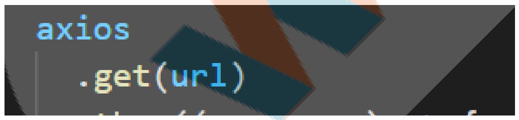
getMovie() function:

1. 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 + "/movies"**.



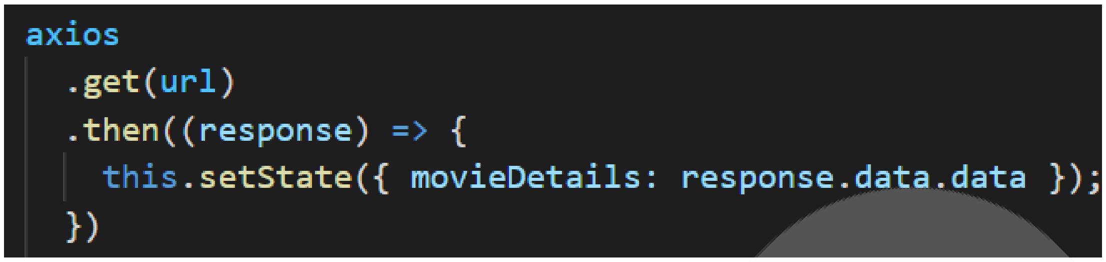
```
const url = this.state.ngrok_url+"/movies";
```

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



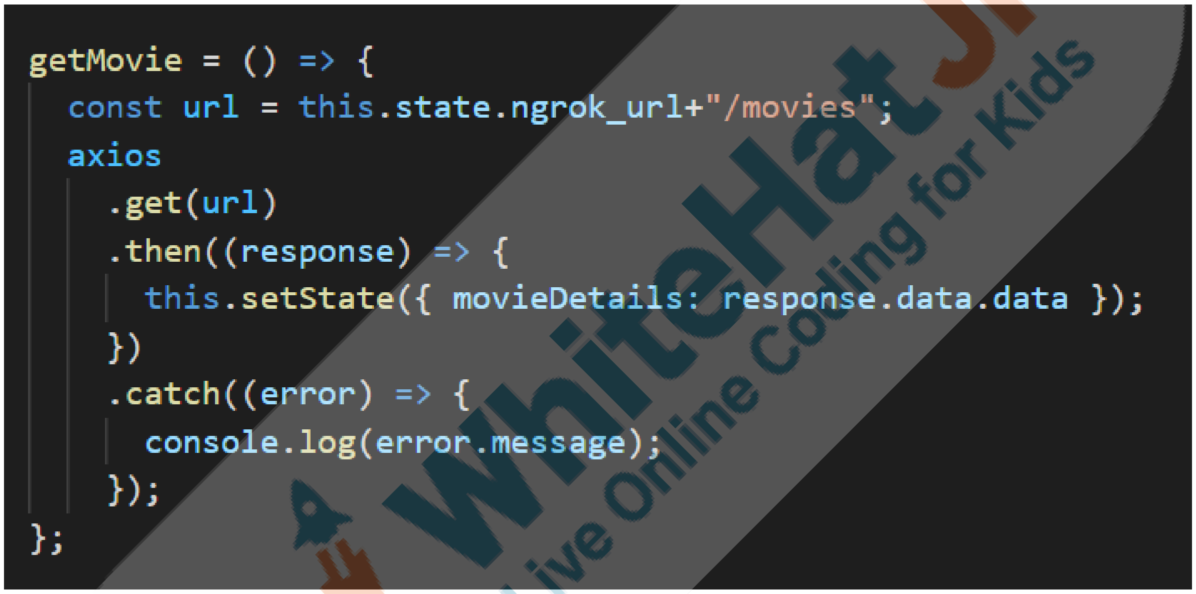
```
axios
  .get(url)
```

3. 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((response) => {
    this.setState({ movieDetails: response.data.data });
  })
```

4. Add a **.catch()** function in case any error arises. Here, simply **console.log()** the error message.

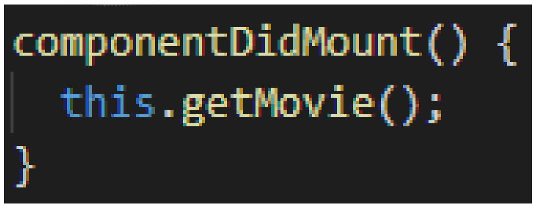


```
getMovie = () => {
  const url = this.state.ngrok_url+"/movies";
  axios
    .get(url)
    .then((response) => {
      this.setState({ movieDetails: response.data.data });
    })
    .catch((error) => {
      console.log(error.message);
    });
};
```

When should the **getMovie()** function be called?

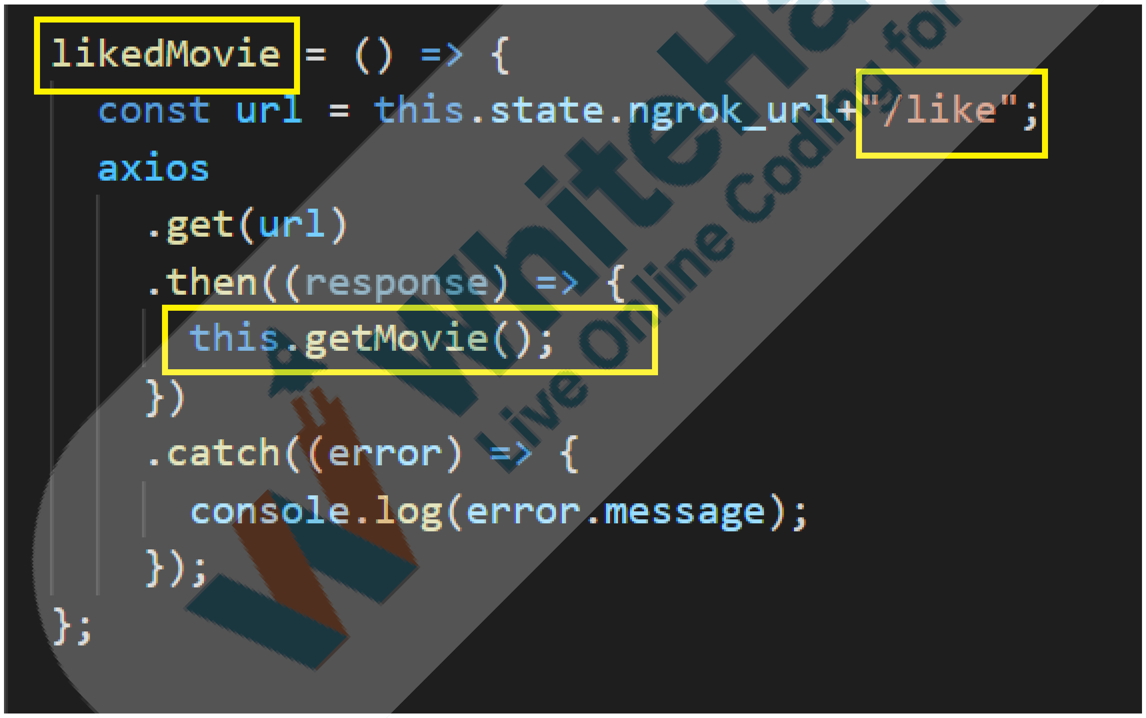
ESR: As soon as the component mounts i.e. the **HomeScreen** opens.

Exactly! So, let's call the **getMovie()** function in the **componentDidmount()** function.



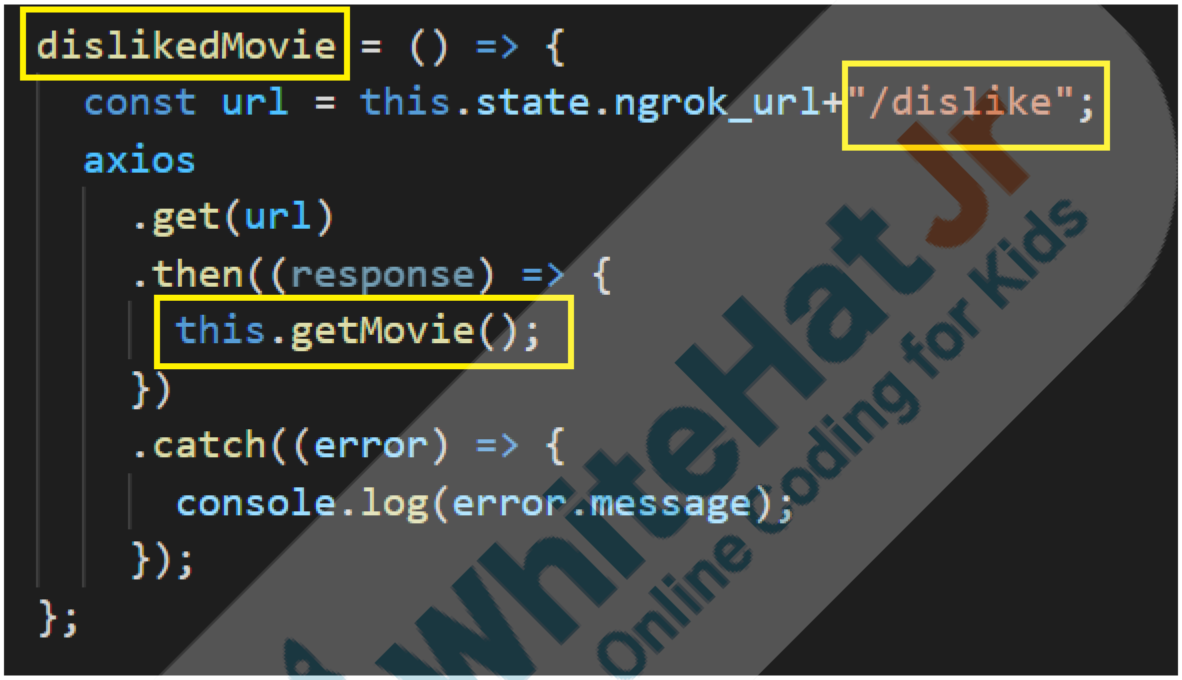
```
componentDidMount() {
  this.getMovie();
}
```

We can copy this **getMovie()** function and reuse it to define the next 3 functions. <u>**likedMovie()** function:</u> 1. Copy and paste the **getMovie()** function. 2. Rename the function to “**likedMovie**”. 3. Change the last portion of the url to “**/like**” instead of “/movies”. 4. In the **.then()** function call **this.getMovie()** function. This will fetch the data of the next movie once you like the current movie.	

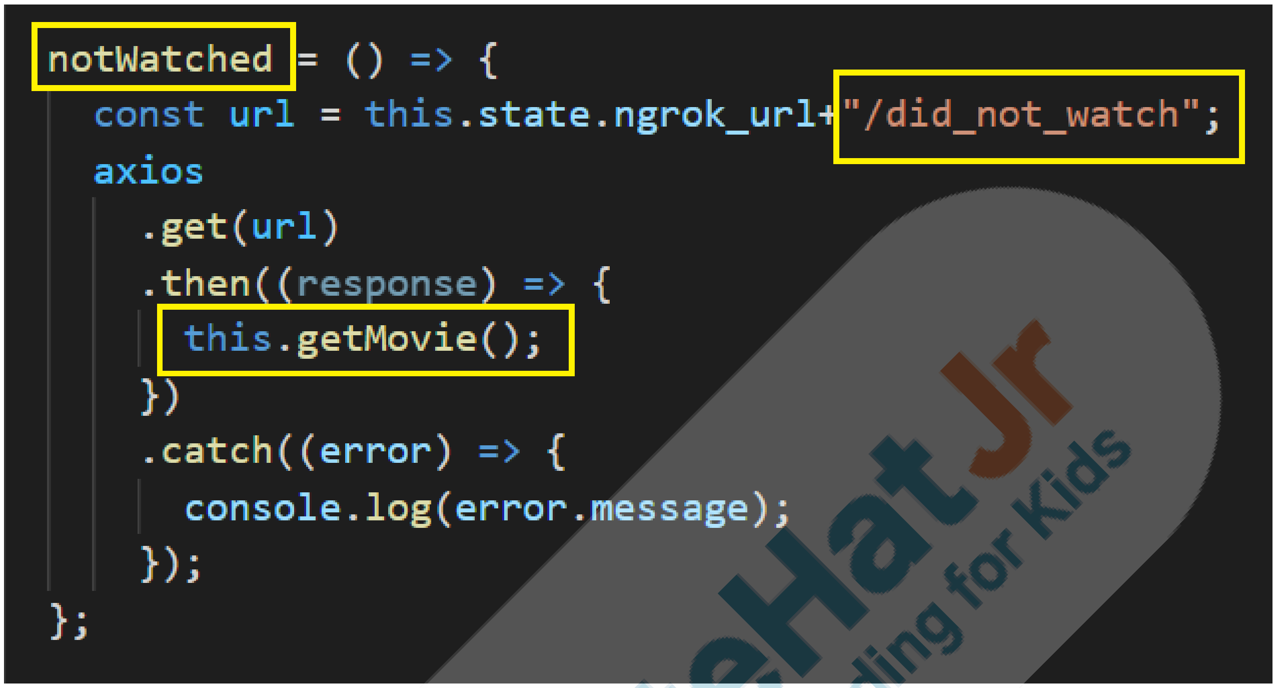


```
likedMovie = () => {
  const url = this.state.ngrok_url+"/like";
  axios
    .get(url)
    .then((response) => {
      this.getMovie();
    })
    .catch((error) => {
      console.log(error.message);
    });
};
```

<u>**dislikedMovie()** function:</u> 1. Copy and paste the **getMovie()** function. 2. Rename the function to “**dislikedMovie**”.	

3. Change the last portion of the url to “**/dislike**”. 4. In the **.then()** function call **this.getMovie()** function. This will fetch the data of the next movie once you like the current movie.	
	
notWatched() function: 1. Copy and paste the **getMovie()** function. 2. Rename the function to “**notWatched**”. 3. Change the last portion of the url to “**/did_not_watch**”. 4. In the **.then()** function call **this.getMovie()** function. This will fetch the data of the next movie once you like the current movie.	

```
dislikedMovie = () => {
  const url = this.state.ngrok_url+"/dislike";
  axios
    .get(url)
    .then((response) => {
      this.getMovie();
    })
    .catch((error) => {
      console.log(error.message);
    });
};
```


```
notWatched = () => {
  const url = this.state.ngrok_url+"/did_not_watch";
  axios
    .get(url)
    .then((response) => {
      this.getMovie();
    })
    .catch((error) => {
      console.log(error.message);
    });
};
```

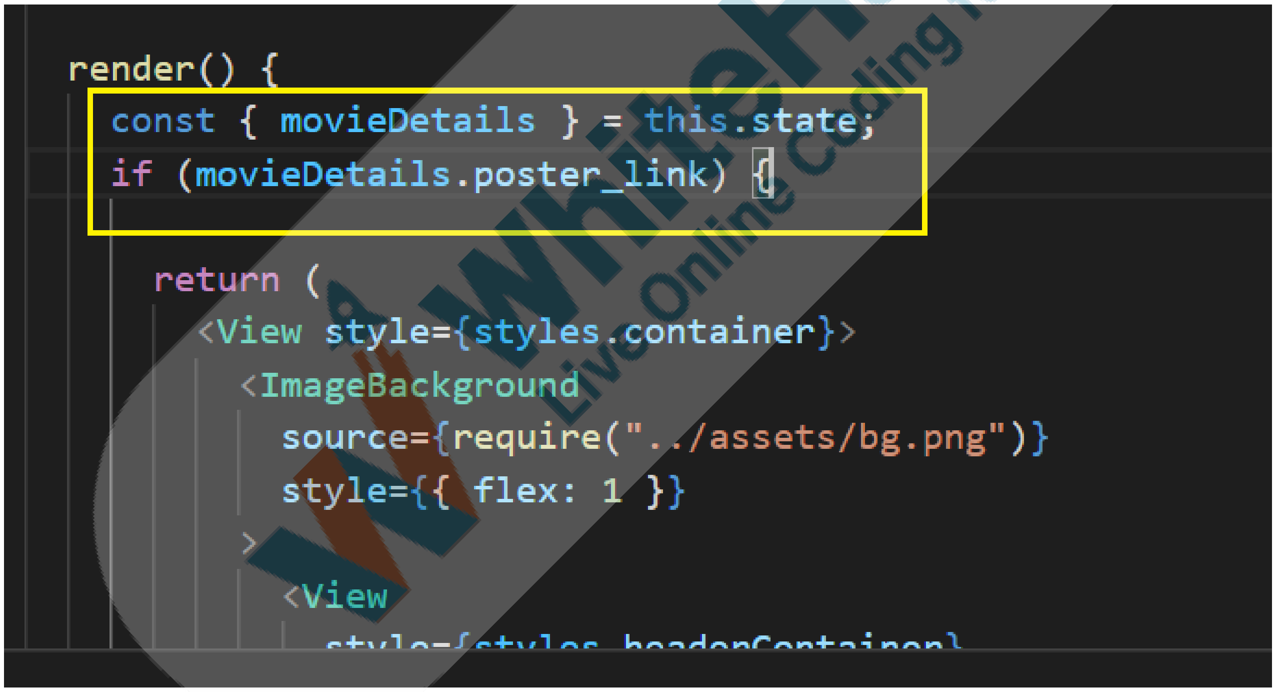
Now, we have all the API related functions defined which will make sure we have our movie data. Let's show this data on the HomeScreen now. As we are getting our data from an API, it might take some time to get the response. So, we need to make sure that we render our Components once we have fetched the data. Otherwise, what do you think will happen? Yes. To avoid this, what can we do?	**ESR:** Our app will crash. **ESR:** We can add an if-else condition and render the components only when the movieDetails state is having the data.

Perfect! We will also want to make sure that the **movieDetails** state has the poster_link too. We can check the **movieDetails.poster_link** only, this will make sure that poster_link is present and movieDetails is also not null.

1. At the beginning of the **render()** function, write **const {movieDetails} = this.state**

   This will make sure that you can access **this.state.movieDetails** just by writing **movieDetails** here onwards.

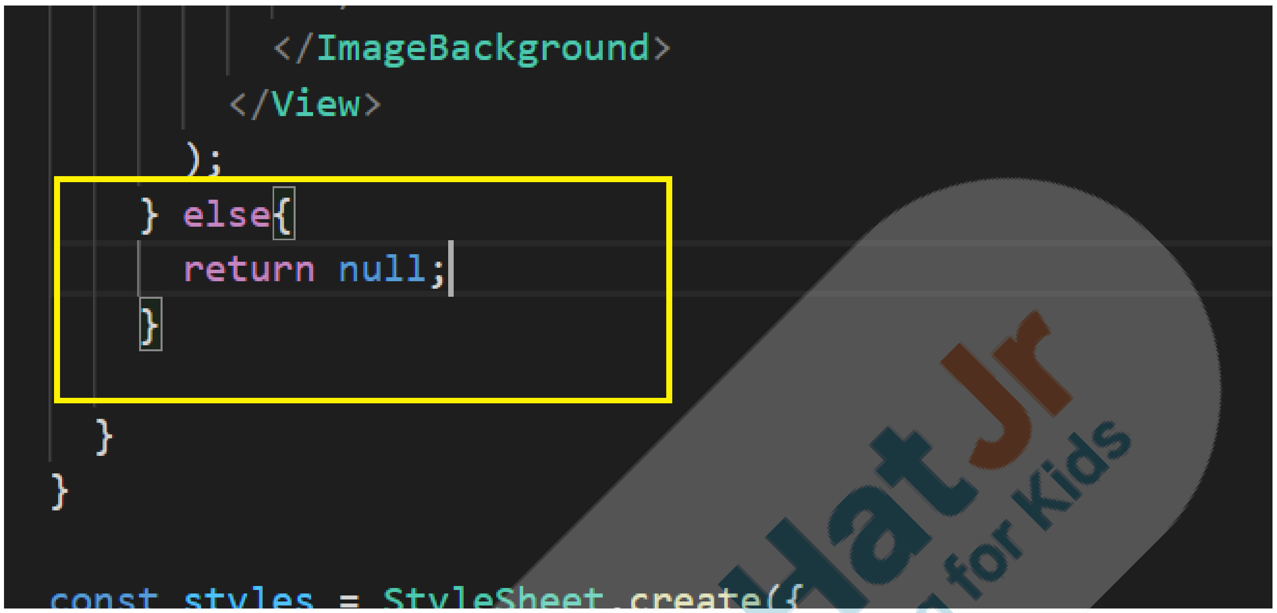
2. Add an **if** statement and check if the **movieDetails.poster_link** is null or not.



```
render() {
  const { movieDetails } = this.state;
  if (movieDetails.poster_link) {

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

3. At the end of the **render()** function, end the if condition and **return null** from else section. This will return nothing when you don't have the data yet and prevents the app from crashing.



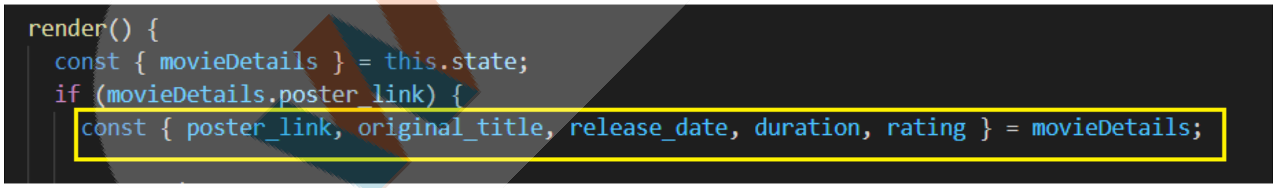
```
        </ImageBackground>
      </View>
    );
  } else{
    return null;
  }
 }
}

const styles = StyleSheet.create({
```

Now, we will add a line inside the **if** statement which will simplify the names of the data furthermore.

```
const { poster_link, original_title, release_date,
duration, rating } = movieDetails;
```

This will make sure that we can access **movieDetails.poster_link** by writing only **poster_link**.
Or, **movieDetails.title** by writing only **title** and so on.



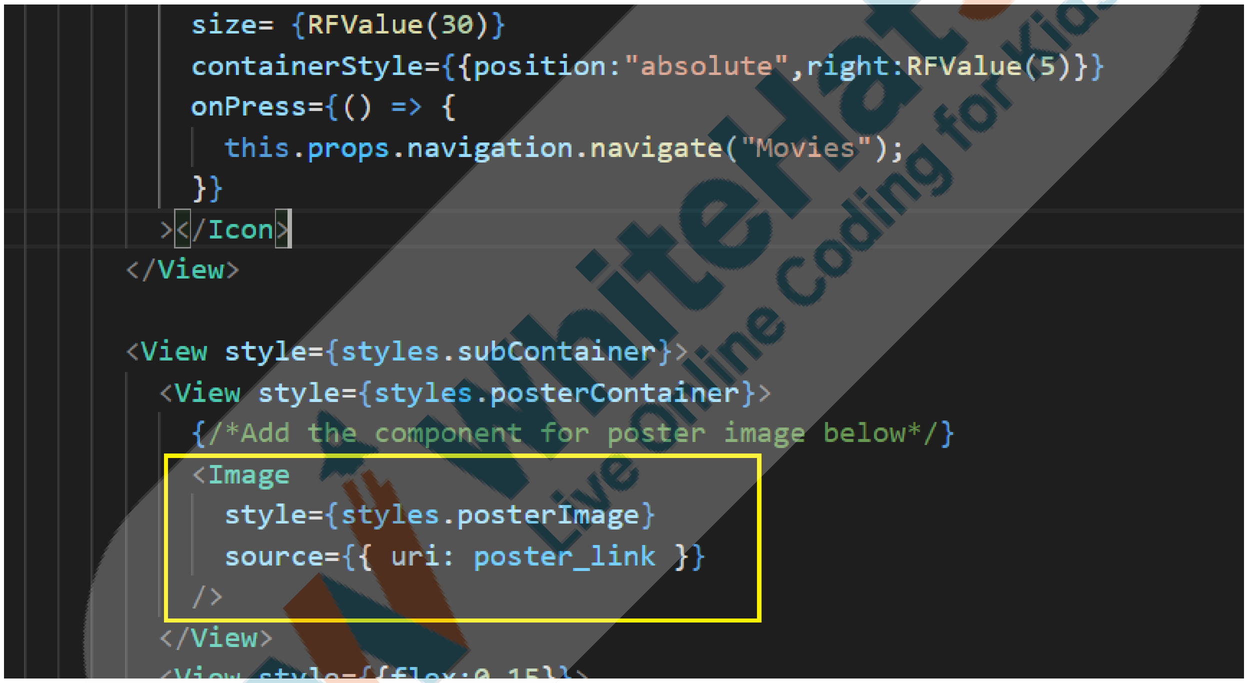
```
render() {
  const { movieDetails } = this.state;
  if (movieDetails.poster_link) {
    const { poster_link, original_title, release_date, duration, rating } = movieDetails;
```

Let’s work on the **return** function now.

We already have an **ImageBackground** component and a header defined.

So, let’s work on the poster first.

Render the poster: 1. Find the section where we want to add the poster. 2. We will be needing an **Image** component so that we can display the poster image for our movies. 3. Add the following props to this component: a. **style:** assign it as styles.posterImage. This style is already defined in the stylesheet b. **source:** assign it as {{uri: poster_link}}	

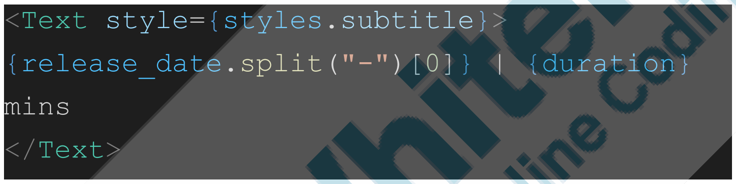


```
          size= {RFValue(30)}
          containerStyle={{position:"absolute",right:RFValue(5)}}
          onPress={() => {
            this.props.navigation.navigate("Movies");
          }}
        ></Icon>
      </View>

      <View style={styles.subContainer}>
        <View style={styles.posterContainer}>
          {/*Add the component for poster image below*/}
          <Image
            style={styles.posterImage}
            source={{ uri: poster_link }}
          />
        </View>
```

Now, we will show the movie name on the screen. Also, we will show some other details like — release date & duration of the movie. Render the Movie name & other details: 1. Find the section where we want to add the movie name & other details.	

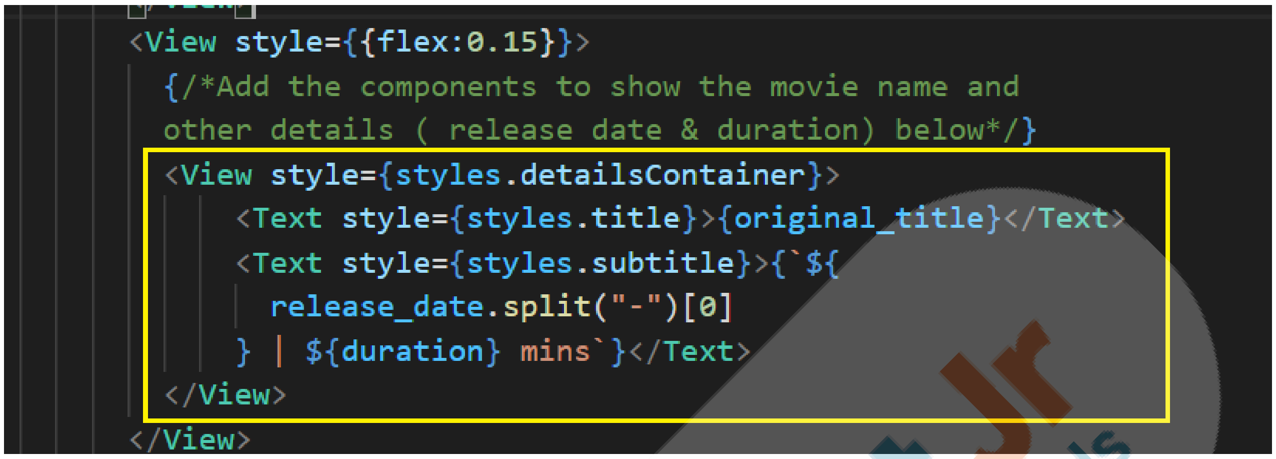
2. We will be needing a **<View>** component as a container.
    a. Add a **<View>** component.
    b. Add the style as **styles.detailsContainer**. This style is already defined in the stylesheet.
3. Add a **<Text>** component.
    a. In the first **<Text>** component, show the **original_title**.
    b. Add the style as **styles.title**. This style is already defined in the stylesheet.
4. Add another **<Text>** component.
    a. In the second **<Text>** component, show the release_date and duration of the movie.



```
<Text style={styles.subtitle}>
{release_date.split("-")[0]} | {duration}
mins
</Text>
```

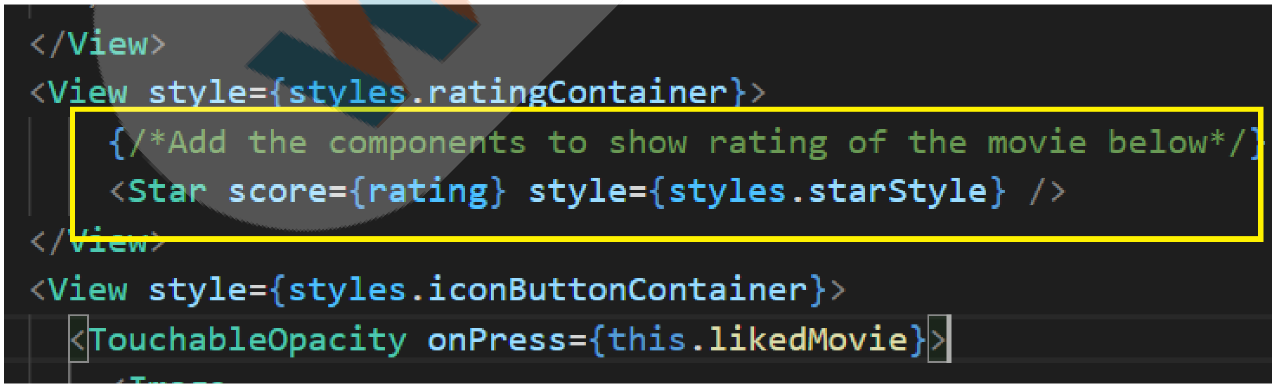
    b. Here, **release_date** is in **yyyy-mm-dd** format. We want to show the year of the release only. That's why we write **release_date.split("-")[0]**.

    The .split("-") function will return an array of [yyyy, mm, dd] format. The **[0]** returns the first element from this array.
    c. Show duration as well.
    d. Add the style as **styles.subtitle**. This style is already defined in the stylesheet.



```
<View style={{flex:0.15}}>
  {/*Add the components to show the movie name and
  other details ( release date & duration) below*/}
  <View style={styles.detailsContainer}>
      <Text style={styles.title}>{original_title}</Text>
      <Text style={styles.subtitle}>{`${
        release_date.split("-")[0]
      } | ${duration} mins`}</Text>
  </View>
</View>
```

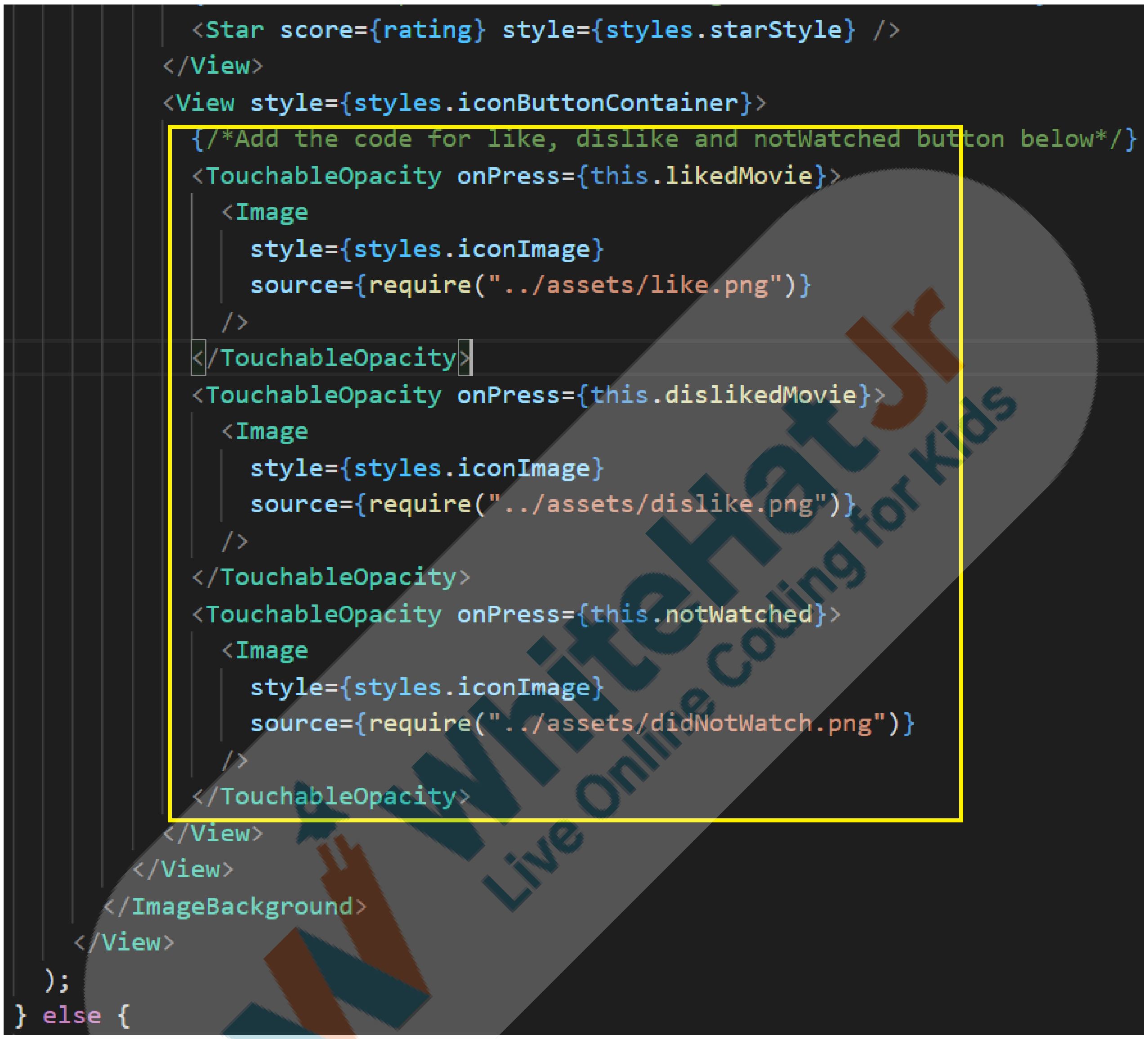
Now, what next? Great! Let's add that. Render movie rating: 1. Find the section where we want to add the movie rating. 2. We will be needing a **<Star>** component. 3. Add the following props to the **<Star>** component: a. **style:** assign it as styles.starStyle. This style is already defined in the stylesheet b. **score:** assign it as {rating}	**ESR:** Movie rating.



```
</View>
<View style={styles.ratingContainer}>
  {/*Add the components to show rating of the movie below*/}
  <Star score={rating} style={styles.starStyle} />
</View>
<View style={styles.iconButtonContainer}>
  <TouchableOpacity onPress={this.likedMovie}>
```

What's left now? Yes! Finally, we will use the **TouchableOpacity** and the **Image** components to create **like, dislike** and **notWatched buttons** which, when pressed, will call the **likedMovie, unlikedMovie** and **notWatched** movie functions. Render the **like**, **dislike** and **notWatched** buttons: 1. Find the section where we want to add the buttons. 2. Like button: a. Add a **<TouchableOpacity>** component for the like button. i. Add **onPress** props and call the **this.likedMovie** function. When you press this button, **likedMovie** function will be called. b. Add an <**Image**> component for the like image. Add the following props to this component. i. **style:** assign it as styles.iconImage. This style is already defined in the stylesheet ii. **source:** assign it as {require("add_*path_for_like_image_here*")} 3. Dislike button: a. Add a **<TouchableOpacity>** component for the dislike button. i. Add **onPress** props and call the **this.dislikedMovie** function. When you press this button, **dislikedMovie** function will be called. b. Add an <**Image**> component for the like image. Add the following props to this component.	**ESR:** The like, dislike and notWatched buttons.

i. **style:** assign it as styles.iconImage. This style is already defined in the stylesheet ii. **source:** assign it as {require("*add_path_for_dislike_image_here*")} 4. notWatched button: a. Add a **<TouchableOpacity>** component for the like button. i. Add **onPress** props and call the **this.notWatched** function. When you press this button, **notWatched** function will be called. b. Add an <**Image**> component for the like image. Add the following props to this component. i. **style:** assign it as styles.iconImage. This style is already defined in the stylesheet ii. **source:** assign it as {require("*add_path_for_didNotWatch_image_here*")}	

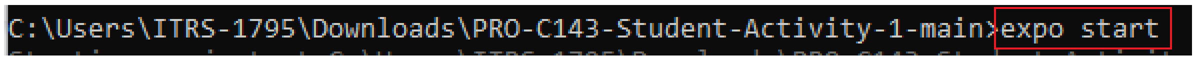


```
          <Star score={rating} style={styles.starStyle} />
        </View>
        <View style={styles.iconButtonContainer}>
          {/*Add the code for like, dislike and notWatched button below*/}
          <TouchableOpacity onPress={this.likedMovie}>
            <Image
              style={styles.iconImage}
              source={require("../assets/like.png")}
            />
          </TouchableOpacity>
          <TouchableOpacity onPress={this.dislikedMovie}>
            <Image
              style={styles.iconImage}
              source={require("../assets/dislike.png")}
            />
          </TouchableOpacity>
          <TouchableOpacity onPress={this.notWatched}>
            <Image
              style={styles.iconImage}
              source={require("../assets/didNotWatch.png")}
            />
          </TouchableOpacity>
        </View>
      </View>
    </ImageBackground>
  </View>
);
} else {
```

To run you code, save all the changes, and in your command prompt, write the command **expo start**.

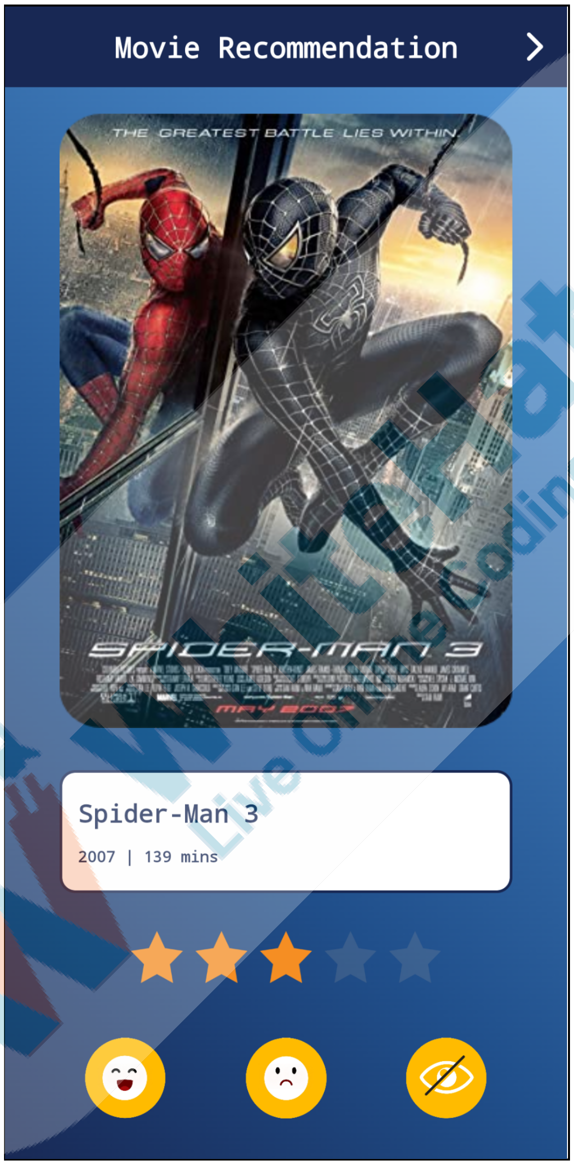
You will see a **QR code** generated. Scan it with the help of your mobile and you will be able to see the **Homescreen** of your app.

If you cannot see the output, recheck the ngrok and localhost url. Make sure these are working.



```
C:\Users\ITRS-1795\Downloads\PRO-C143-Student-Activity-1-main>expo start
```

Reference output:



Spider-Man 3

2007 | 139 mins

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)
FEEDBACK ● **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*

WhiteHat Jr
Live online Coding for Kids

In the next class, we will build the second screen of our movie recommendation app. We will show the lists for recommended movies, popular movies & liked movies in the second screen.	*hats-off during the 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 ✖ End Class	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Previous Class Code	https://github.com/procodingclass/PRO-C142-Reference-Code.git
Teacher Activity 2	Boilerplate code (React Native App)	https://github.com/procodingclass/PRO-C143-Student-Boilerplate
Teacher Activity 3	Reference Code	https://github.com/procodingclass/PRO-C143-Reference-Code
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/a296c926-2ee0-471a-8f1a-1ea2260fe714.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C143-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/948821d2-1c57-4783-bf25-40bb46b1ea2d.pdf
Student Activity 1	Previous class code (Flask API)	https://github.com/procodingclass/PRO-C142-Reference-Code.git
Student Activity 2	Boilerplate code (React Native App)	https://github.com/procodingclass/PRO-C143-Student-Boilerplate
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	ngrok download link	https://ngrok.com/download
Student Activity 5	Wireframe for screen 1	https://s3-whjr-curriculum-uploads.whjr.online/cba92d71-e7a9-4590-8ad6-04d75deb185c.png