Topic	EXOPLANET CATALOG	
Class Description	The student will be creating a mobile app which will be a catalog for exo-planets based on all the data we curated.	
Class	PRO C137	
Class time	45 mins	
Goal	● Build 2 screens in Mobile app ● Integrate React Native with 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
Credit & Permissions:	Exoplanet Exploration by NASA	

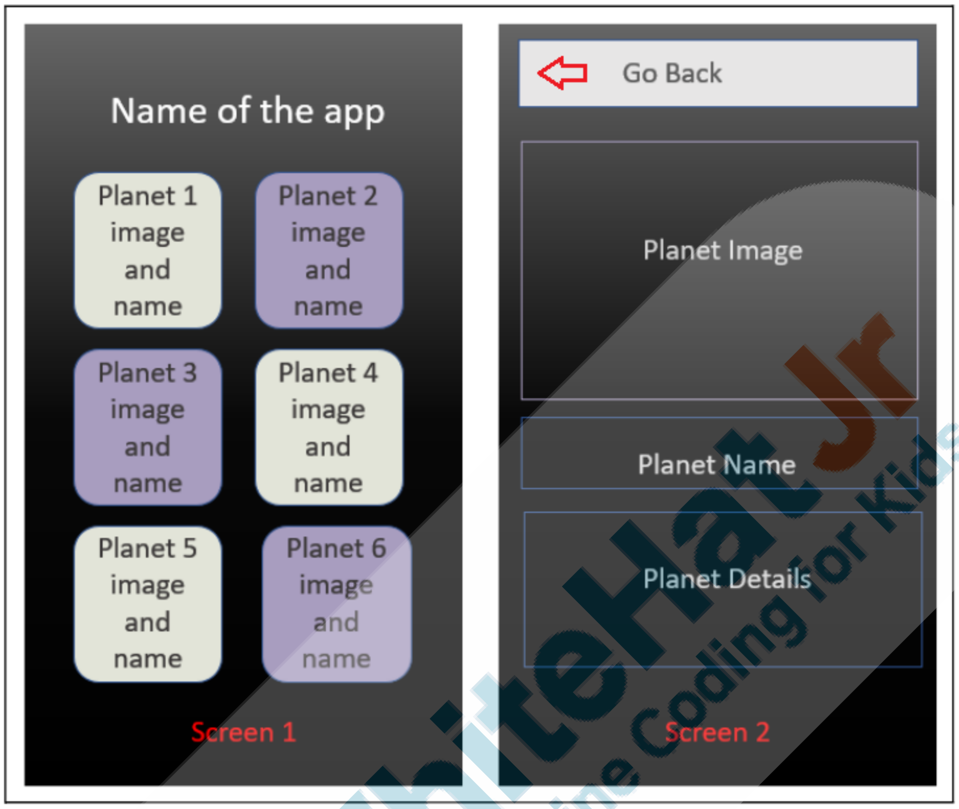
WARM-UP SESSION - 10 mins

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
Hi <Student Name>! We have finally created a Flask API with all the curated data! Now, it's time to create an Exoplanet Catalog using React Native. Can you tell me how many APIs we created in the last class and what were they?	**ESR:** We created 2 APIs in the last class: ● First to get data for all the exoplanets. ● Second, to get data for a particular exoplanet.
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 Action	**Student Action**
Great! Now in this class, you will create a react native app with 2 screens. The first one will have the name of all the planets and if you click on the planet, you go to the second screen that displays all the data about that planet! Are you excited?	**ESR:** Yes

Teacher Ends Slideshow	
TEACHER-LED ACTIVITY - 10 mins	
Teacher Initiates Screen Share	
ACTIVITY ● **To describe structure of React Native App**	
Teacher Action	**Student Action**
<In this class, the student will be doing most of the coding with the teacher's guidance> The teacher is required to 1. Help the student in completing: ● Screen 1 where the list of all the planet names is displayed. ● Screen 2 where we need to display the planet data of the planet that has been clicked. 2. Code for React Native App 3. Integration with Flask API Refer to Teacher Activity 1 for the sample catalog.	
App Design Layout:	

WhiteHat Jr
Live online Coding for Kids



Let's go through the boilerplate code once before we start. *<Teacher encourages the student to download the boiler-plate code and asks to share the student's screen>* Now, let's run the command "**npm install**" in this downloaded folder to install the **node_modules**. It will work in the background while we go through the given code.	*<Student downloads the boilerplate code>*
<Teacher asks the student to open the project folder with VS Code> Please go through the **App.js** file.	

Can you explain the code? If we run this code, which screen do you think will show up initially? Great!!	**ESR:** In **App.js**, there is a **stack navigator** which holds two screens **HomeScreen** and **DetailsScreen**. **ESR:** When we run this app, initially **HomeScreen** will be mounted. That is because the **initialRouteName** is set as **home.**
Let's look at the code of **Home.js** now. We can see that the base code is already given for this screen. Let's understand what we have to do in **Home.js**. In Home.js, we have to show the list of exoplanets using a **FlatList**. We will fetch this data from the flask API that we had created in the previous class.	
Now, we have another screen - **Details.js**. Why do you think we are adding this screen? How will the stack navigator help us in that case? Exactly! We will code it in a way that it shows the details of the planet we have clicked on.	**ESR:** We view the details about the planets here. **ESR:** In home.js, when we click on a particular planet name, the stack navigator will navigate us to the details screen.
Teacher Stops Screen Share	
Now it's your turn. Please share your screen with me.	

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 ● **To build a React Native App** ● **To integrate React Native App with Flask API**	
Teacher Action	**Student Action**
Our screens are already added in the **stack navigator**. One of the major goals of today's class is understanding how to integrate **React Native** with the **Flask API**. Let's code in our **Home.js** file to make a query on our **API** and get the list of all the planets and show it using **FlatList**.	

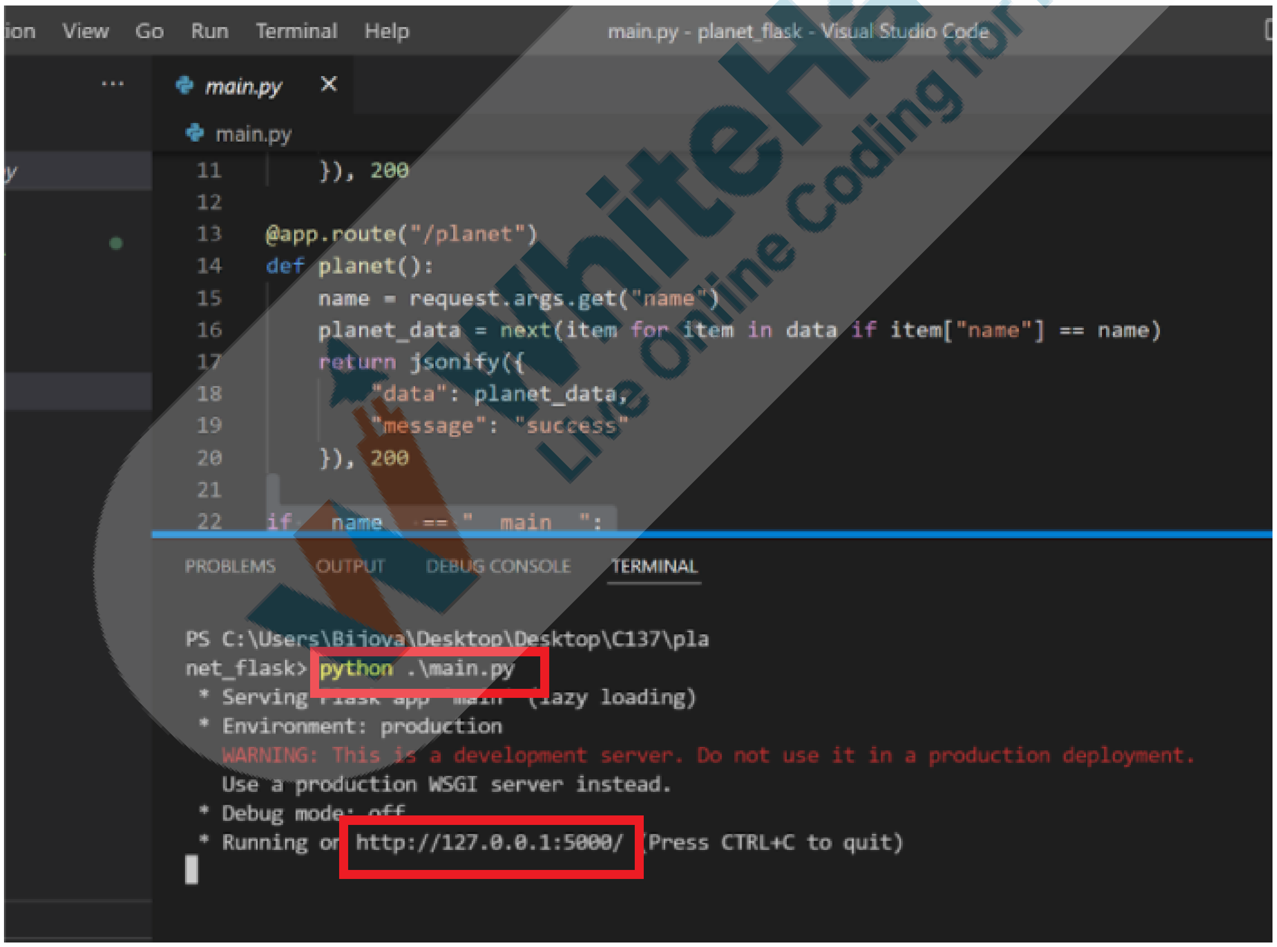
To make a **GET** request on an **API** we'll be using **axios**. To install **axios,** we use the command **npm install axios**.

Open Student Activity1 for the boilerplate code.

Note: *You can host the Flask API on localhost.*

To do that, open the code from the last class and run it on the terminal by running the command "**python file_name**"/ "**py file_name**".

<Student install axios module in the project folder>

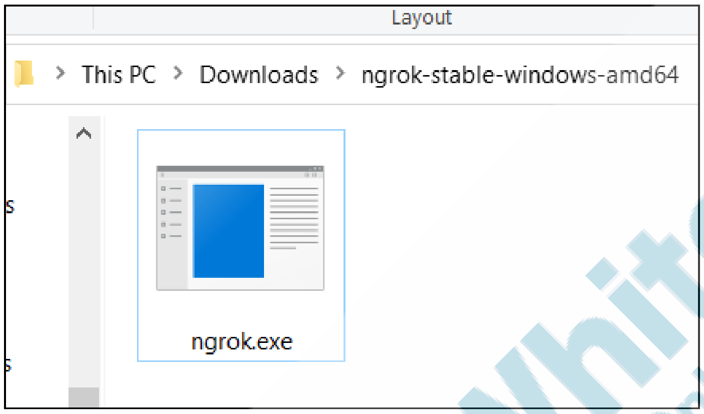
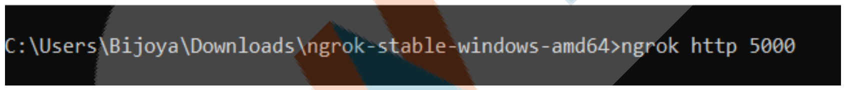


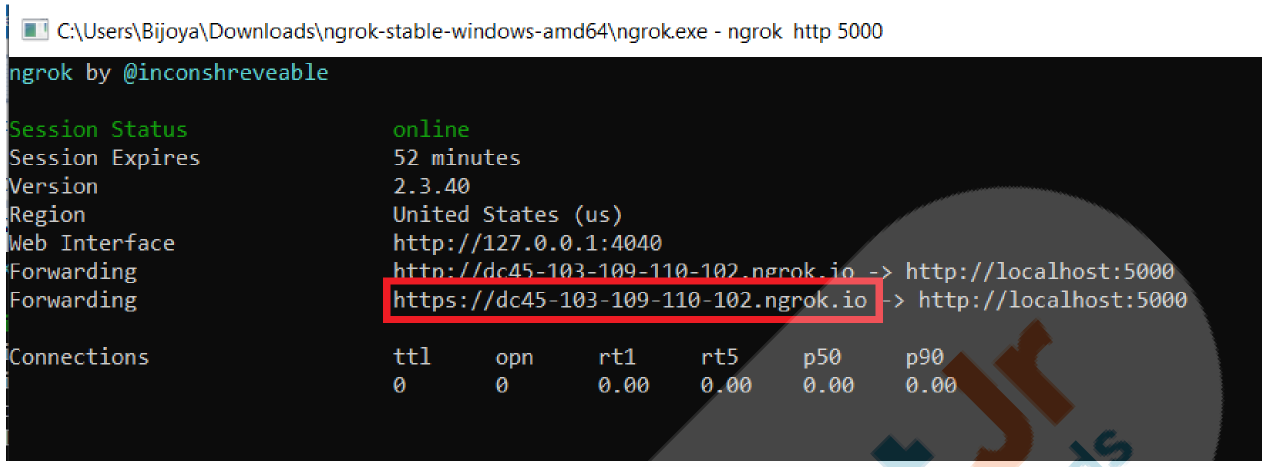
ion View Go Run Terminal Help
main.py - planet_flask - Visual Studio Code
main.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
    }), 200

@app.route("/planet")
def planet():
    name = request.args.get("name")
    planet_data = next(item for item in data if item["name"] == name)
    return jsonify({
        "data": planet_data,
        "message": "success"
    }), 200

if __name__ == "__main__":
```

```
PS C:\Users\Bijova\Desktop\Desktop\C137\pla
net_flask> 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: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

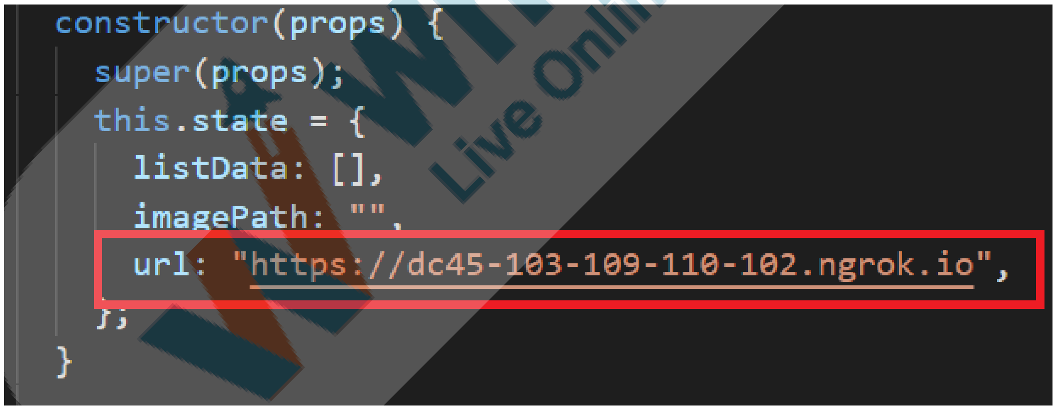
Now, we will have the flask API running on our local server.

We need to expose our local webserver running on your local machine to the internet. Ngrok helps us with that. We had already downloaded it on our system in a previous class. ***Note:** If it is not there in the student's system, download it from **https://dashboard.ngrok.com/get-started/setup**. After successfully downloading ngrok. Open **ngrok.exe** file.*  Layout This PC > Downloads > ngrok-stable-windows-amd64 ngrok.exe **ngrok http <port>**  `C:\Users\Bijoya\Downloads\ngrok-stable-windows-amd64>ngrok http 5000` **Note:** Flask API by default run on port 5000	



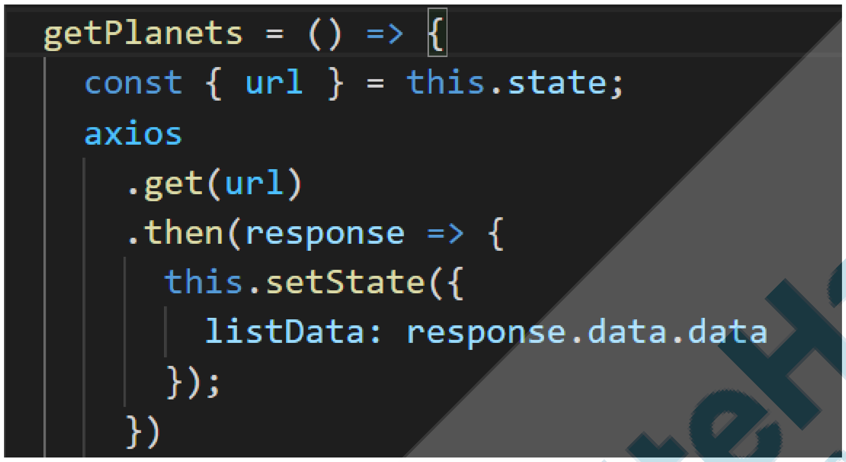
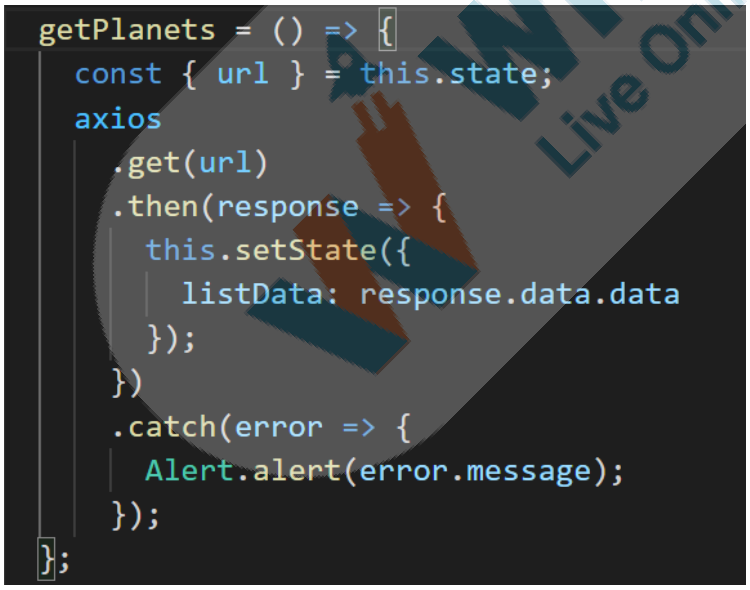
```
C:\Users\Bijoya\Downloads\ngrok-stable-windows-amd64\ngrok.exe - ngrok  http 5000
ngrok by @inconshreveable

Session Status                online
Session Expires               52 minutes
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://dc45-103-109-110-102.ngrok.io -> http://localhost:5000
Forwarding                    https://dc45-103-109-110-102.ngrok.io -> http://localhost:5000

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

We will use this link to fetch the data. *<Teacher guides the student to copy this link>* In **Home.js**, we have three states named **listData, imagePath,** and **url.** We will set the value of the **url** as the ngrok link.	

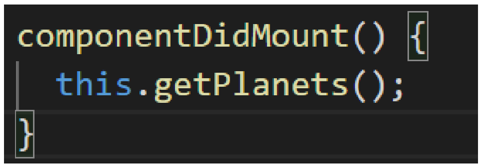


```
constructor(props) {
  super(props);
  this.state = {
    listData: [],
    imagePath: "",
    url: "https://dc45-103-109-110-102.ngrok.io",
  };
}
```

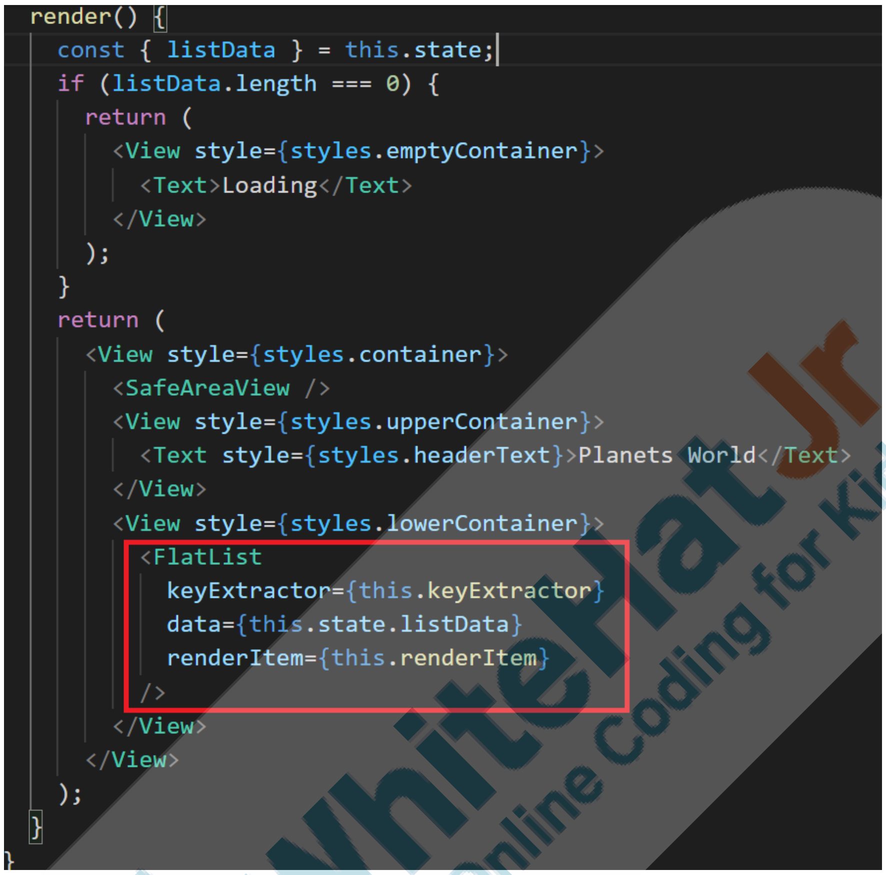
We have already installed the **axios** module. We will use it to fetch the data. Let's define a function named **getPlanets()** and here we will write the code to get the data from the **URL.** First, we will call the **axios.get()** function and we will pass the state named **URL** through it.	

We want to wait till we get the data from the API, so we use the **.then()** function after this.
 	
Once we have fetched the data, we store it in the **listData** state.
 	

`getPlanets = () => {`
`  const { url } = this.state;`
`  axios`
`    .get(url)`
`    .then(response => {`
`      this.setState({`
`        listData: response.data.data`
`      });`
`    })`
 	
We will also add the **.catch()** function after this, in case there is an error.
 	

`getPlanets = () => {`
`  const { url } = this.state;`
`  axios`
`    .get(url)`
`    .then(response => {`
`      this.setState({`
`        listData: response.data.data`
`      });`
`    })`
`    .catch(error => {`
`      Alert.alert(error.message);`
`    });`
`};`	

As we want to fetch this data as soon as the screen mounts, we will call the **getPlanets()** function in	

componentDidMount().

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

We have the data stored in a state now. What should be the next step? Exactly! Let's make a **FlatList** to showcase the planet names. We had learned that a Flatlist has three important props- **keyExtractor**, **data** and **renderItem**. We will call **this.keyExtractor()** function for the keyExtractor props and **this.renderItem()** function for renderItem props. The **keyExtractor()** function is already defined for us. We have the **renderItem()** function partially defined. We will assign the data as **this.state.listData.**	**ESR:** We should focus on rendering the data on the screen.

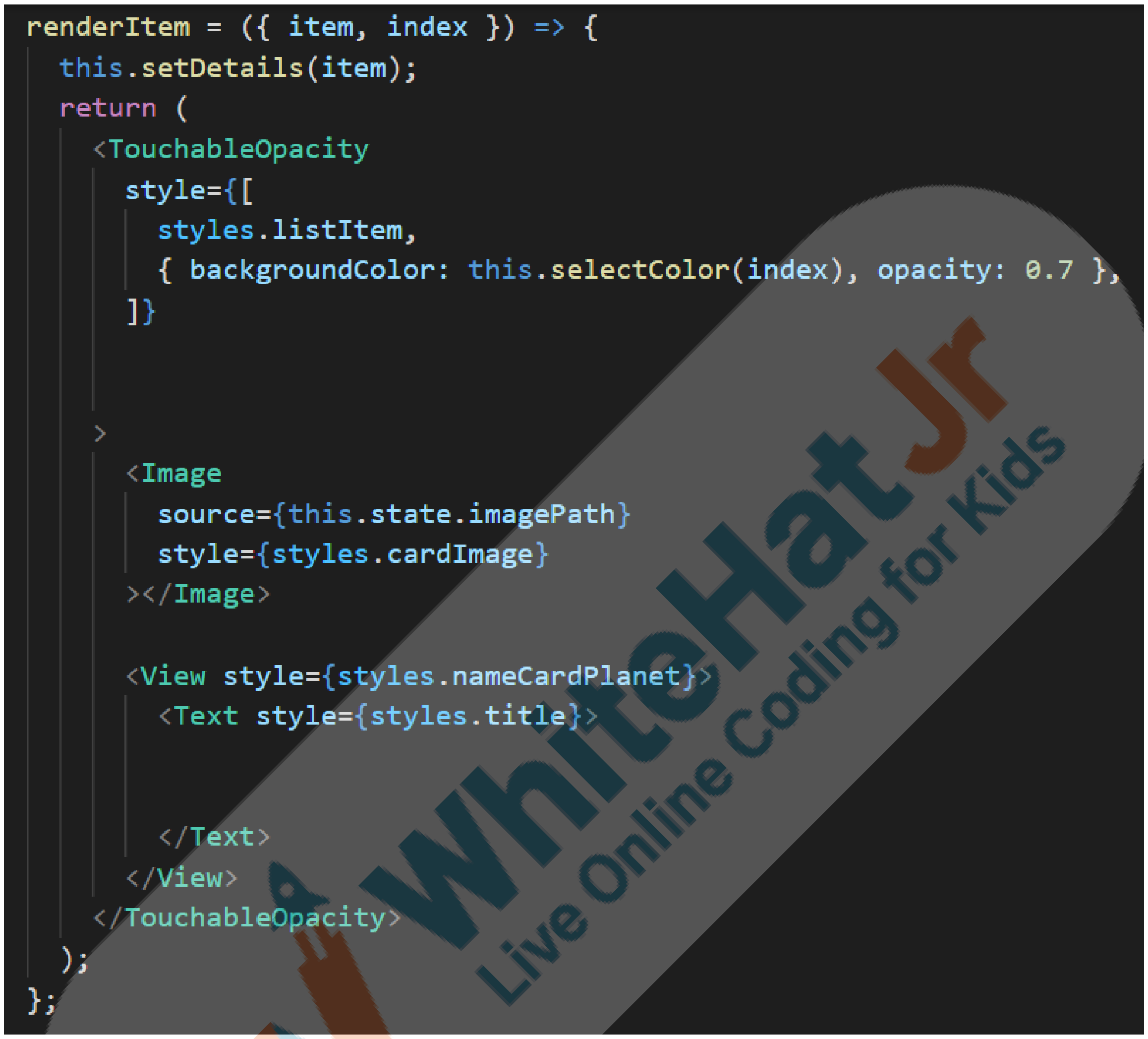


```
render() {
  const { listData } = this.state;
  if (listData.length === 0) {
    return (
      <View style={styles.emptyContainer}>
        <Text>Loading</Text>
      </View>
    );
  }
  return (
    <View style={styles.container}>
      <SafeAreaView />
      <View style={styles.upperContainer}>
        <Text style={styles.headerText}>Planets World</Text>
      </View>
      <View style={styles.lowerContainer}>
        <FlatList
          keyExtractor={this.keyExtractor}
          data={this.state.listData}
          renderItem={this.renderItem}
        />
      </View>
    </View>
  );
}
}
```

We also need to complete the **renderItem()** function now. We will render each item in the list using **TouchableOpacity**. Let's locate the function and start working on it.

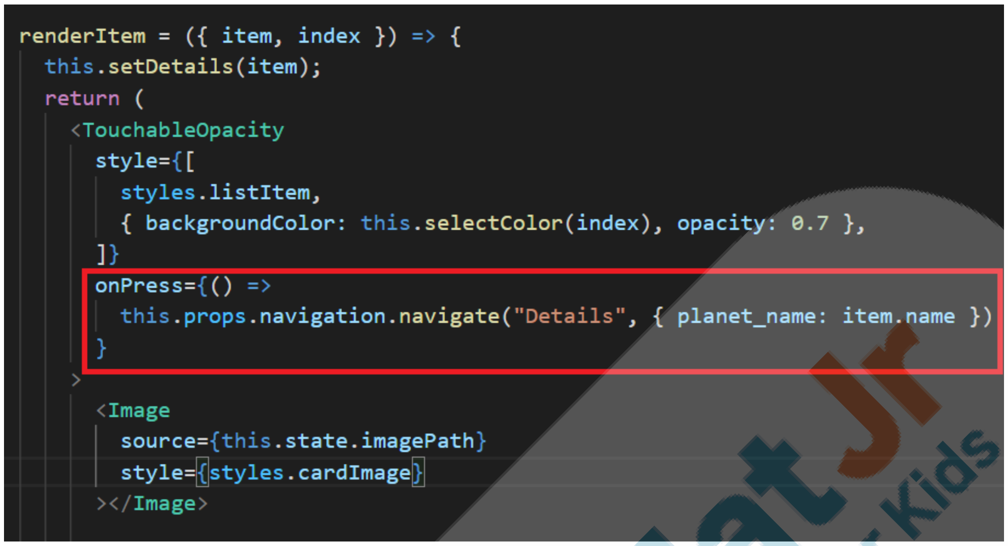
In the given code, we call the **setDetails()** function. This function was already defined for us in the boilerplate code. This function will determine an image for each planet depending on the **planet_type.**

Each item from the **listItem state** is rendered using a **TouchableOpacity.** An image is added depending on the **planet_type**.



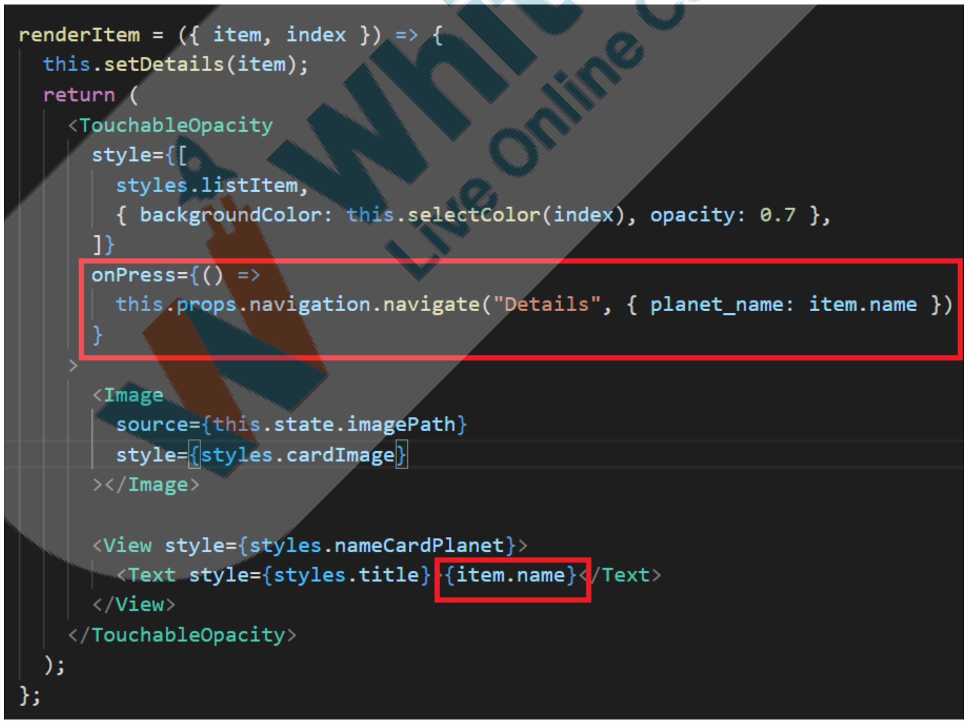
```
renderItem = ({ item, index }) => {
  this.setDetails(item);
  return (
    <TouchableOpacity
      style={[
        styles.listItem,
        { backgroundColor: this.selectColor(index), opacity: 0.7 },
      ]}


    >

      <Image
        source={this.state.imagePath}
        style={styles.cardImage}
      ></Image>

      <View style={styles.nameCardPlanet}>
        <Text style={styles.title}>


        </Text>
      </View>
    </TouchableOpacity>
  );
};
```

What do you think is left?	**ESR:** We need to add **onPress** props to the **TouchableOpacity** so that it can navigate to the details screen.
Great! Also, when we navigate to the details screen, we would want to pass the planet name which we have clicked on. Can you write the code for it?	**ESR:** Sure!



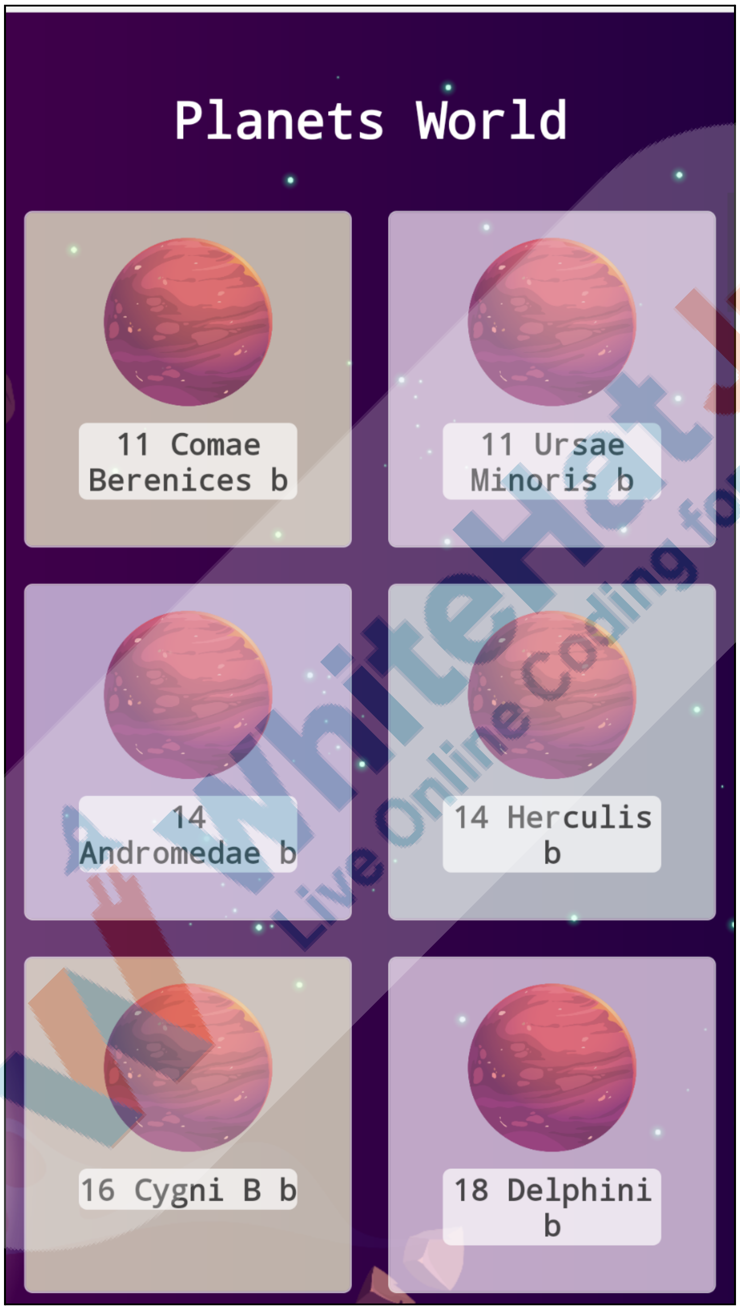
```
renderItem = ({ item, index }) => {
  this.setDetails(item);
  return (
    <TouchableOpacity
      style={[
        styles.listItem,
        { backgroundColor: this.selectColor(index), opacity: 0.7 },
      ]}
      onPress={() =>
        this.props.navigation.navigate("Details", { planet_name: item.name })
      }
    >
      <Image
        source={this.state.imagePath}
        style={styles.cardImage}
      ></Image>
```

We will also need to add the name of the planet to each item.



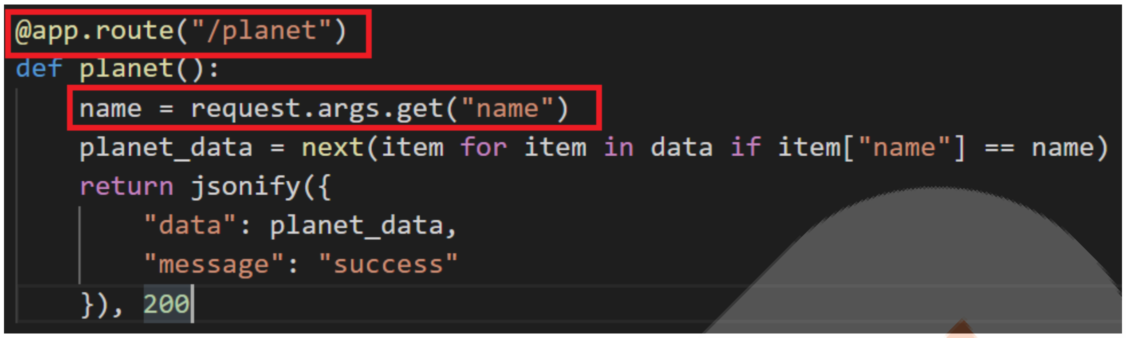
```
renderItem = ({ item, index }) => {
  this.setDetails(item);
  return (
    <TouchableOpacity
      style={[
        styles.listItem,
        { backgroundColor: this.selectColor(index), opacity: 0.7 },
      ]}
      onPress={() =>
        this.props.navigation.navigate("Details", { planet_name: item.name })
      }
    >
      <Image
        source={this.state.imagePath}
        style={styles.cardImage}
      ></Image>

      <View style={styles.nameCardPlanet}>
        <Text style={styles.title}>{item.name}</Text>
      </View>
    </TouchableOpacity>
  );
};
```

If you run this code now, the **HomeScreen** should look like this,



When someone clicks these planet names, they should be able to see the empty **detailsScreen**.

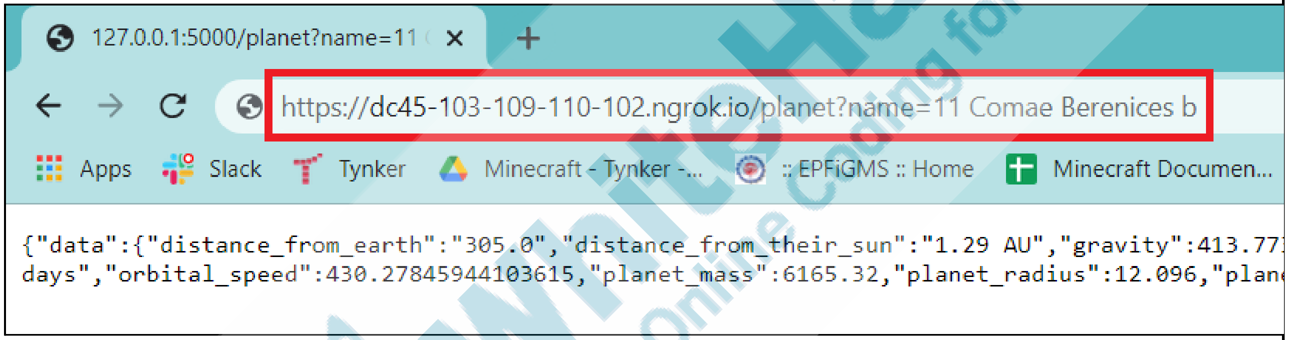
Now, let's write code for the details screen. What do we want to show here? And where do you think we can get the details of the selected planet? Exactly.	**ESR:** We want to show the planet details of the planet which was clicked on. **ESR:** We can get the details from the **Flask API** again.
Here, we need to know two things. 1. How to get the data that was passed from the previous screen. 2. How to access a particular data from an API. How do you think we can access the data from the previous screen? Great! We know the first step. Let's see how to fetch particular data from an API. If you look at our flask API code from the previous class, we can see that we had defined an **@app.route()** named **planet**. Here, we had also added **request.args.get("name")**. So, we can request that data by writing, "Localhost url or tunneled url / **planet?name=** *write the planet name here*".	**ESR:** We can use the **this.props.navigation.getParam()** function.



```
@app.route("/planet")
def planet():
    name = request.args.get("name")
    planet_data = next(item for item in data if item["name"] == name)
    return jsonify({
        "data": planet_data,
        "message": "success"
    }), 200
```

Run this to check the output in the browser.

Example-

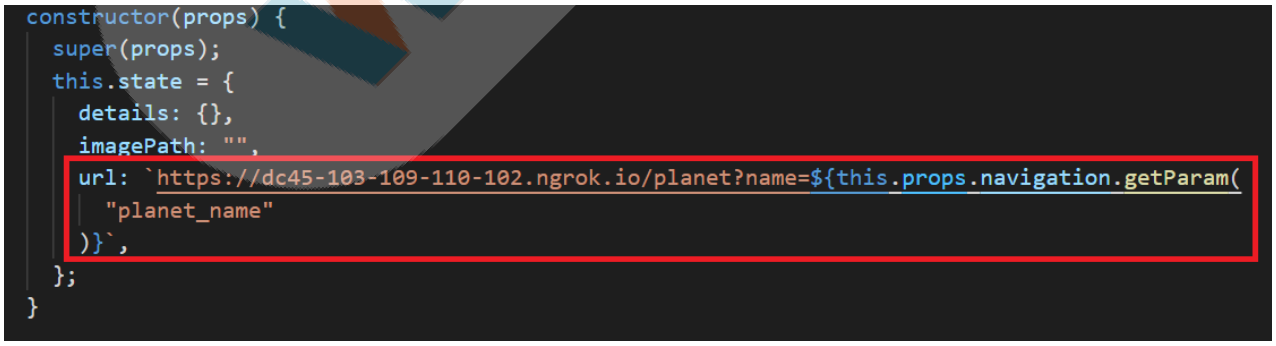


127.0.0.1:5000/planet?name=11
https://dc45-103-109-110-102.ngrok.io/planet?name=11 Comae Berenices b
Apps
Slack
Tynker
Minecraft - Tynker -...
:: EPFiGMS :: Home
Minecraft Documen...

```
{"data":{"distance_from_earth":"305.0","distance_from_their_sun":"1.29 AU","gravity":413.77
days","orbital_speed":430.27845944103615,"planet_mass":6165.32,"planet_radius":12.096,"plan
```

Let’s go back to VS Code now and let’s complete the code.

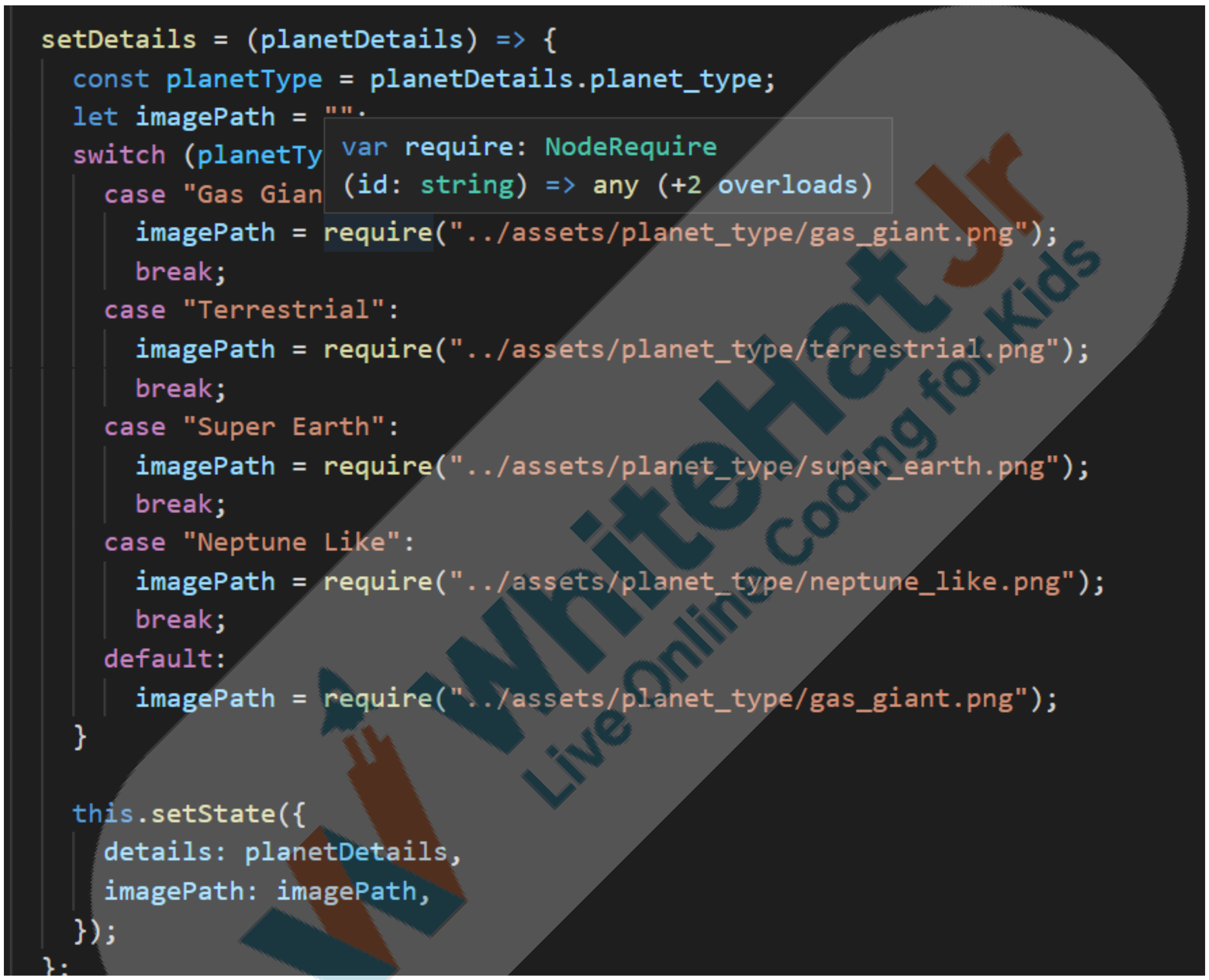
In **details.js**, we already have three states - **details**, **imagePath,** and **url**. In the **url** state, we will add the path to fetch data.



```
constructor(props) {
  super(props);
  this.state = {
    details: {},
    imagePath: "",
    url: `https://dc45-103-109-110-102.ngrok.io/planet?name=${this.props.navigation.getParam(
      "planet_name"
    )}`,
  };
}
```

Just like **Home.js**, **setDetails()** function was already defined for us in the **Details.js**. This function will determine an image for each planet depending on the **planet_type.**

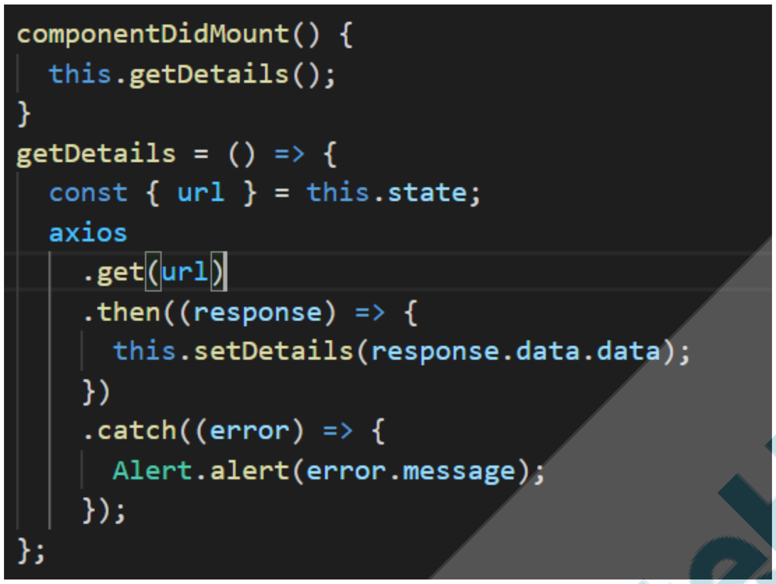
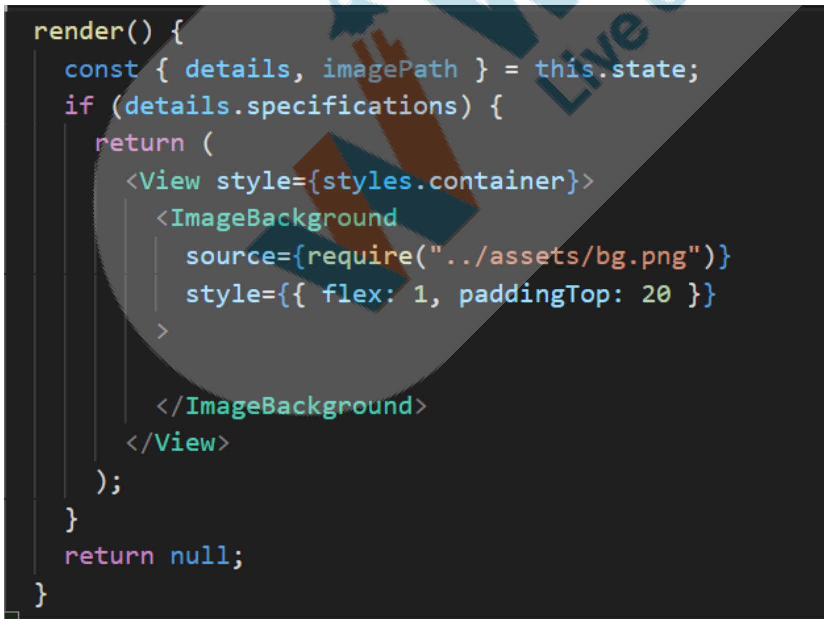
<Teacher shows this function. It is already provided in the boilerplate code. Don't need to write it>



```
setDetails = (planetDetails) => {
  const planetType = planetDetails.planet_type;
  let imagePath = "";
  switch (planetTy
    case "Gas Gian
      imagePath = require("../assets/planet_type/gas_giant.png");
      break;
    case "Terrestrial":
      imagePath = require("../assets/planet_type/terrestrial.png");
      break;
    case "Super Earth":
      imagePath = require("../assets/planet_type/super_earth.png");
      break;
    case "Neptune Like":
      imagePath = require("../assets/planet_type/neptune_like.png");
      break;
    default:
      imagePath = require("../assets/planet_type/gas_giant.png");
  }

  this.setState({
    details: planetDetails,
    imagePath: imagePath,
  });
};
```

```
var require: NodeRequire
(id: string) => any (+2 overloads)
```

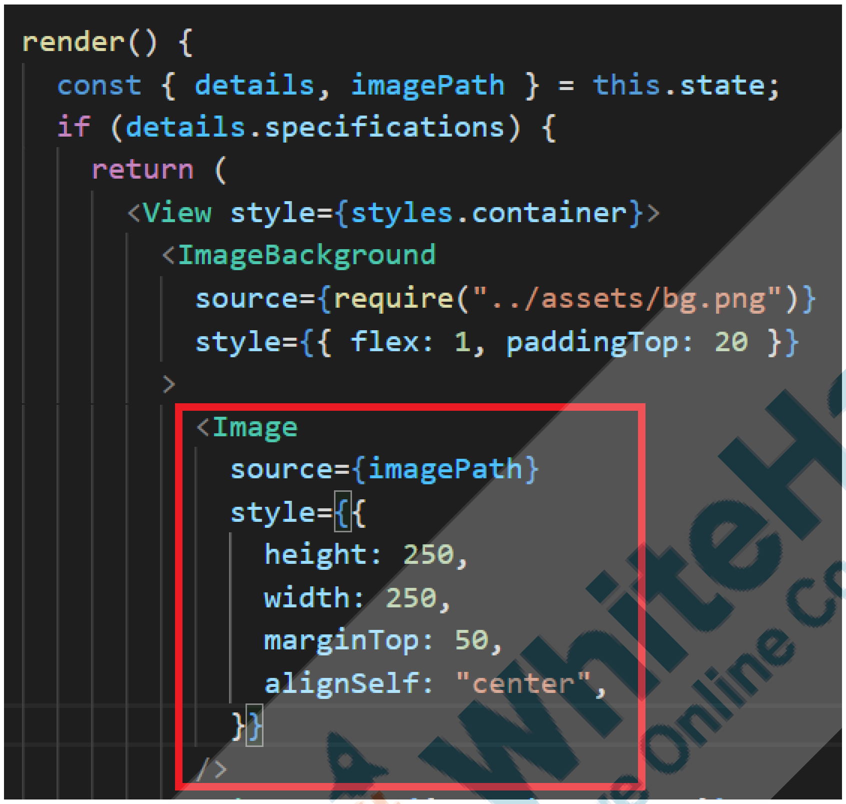
Let's define the **getDetails()** function now. It will be similar to the **getPlanets()** function in **Home.js**. **getDetails()** function which will make a **GET** request on the given URL and get the data. We will call the **setDetails()** function here and pass this data to this **setDetails()** function. We will call the **getDetails()** function in	

componentDidMount() function. This is because we want to get this data as soon as **detailScreen** is loaded.  ``componentDidMount() { this.getDetails(); } getDetails = () => { const { url } = this.state; axios .get(url) .then((response) => { this.setDetails(response.data.data); }) .catch((error) => { Alert.alert(error.message); }); };``	
Now, we have the code to get our data. It's time to render this data. We will add the background image of the screen. Similar to the **Home.js**.  ``render() { const { details, imagePath } = this.state; if (details.specifications) { return ( <View style={styles.container}> <ImageBackground source={require("../assets/bg.png")} style={{ flex: 1, paddingTop: 20 }} > </ImageBackground> </View> ); } return null; }``	

```
componentDidMount() {
  this.getDetails();
}
getDetails = () => {
  const { url } = this.state;
  axios
    .get(url)
    .then((response) => {
      this.setDetails(response.data.data);
    })
    .catch((error) => {
      Alert.alert(error.message);
    });
};
```

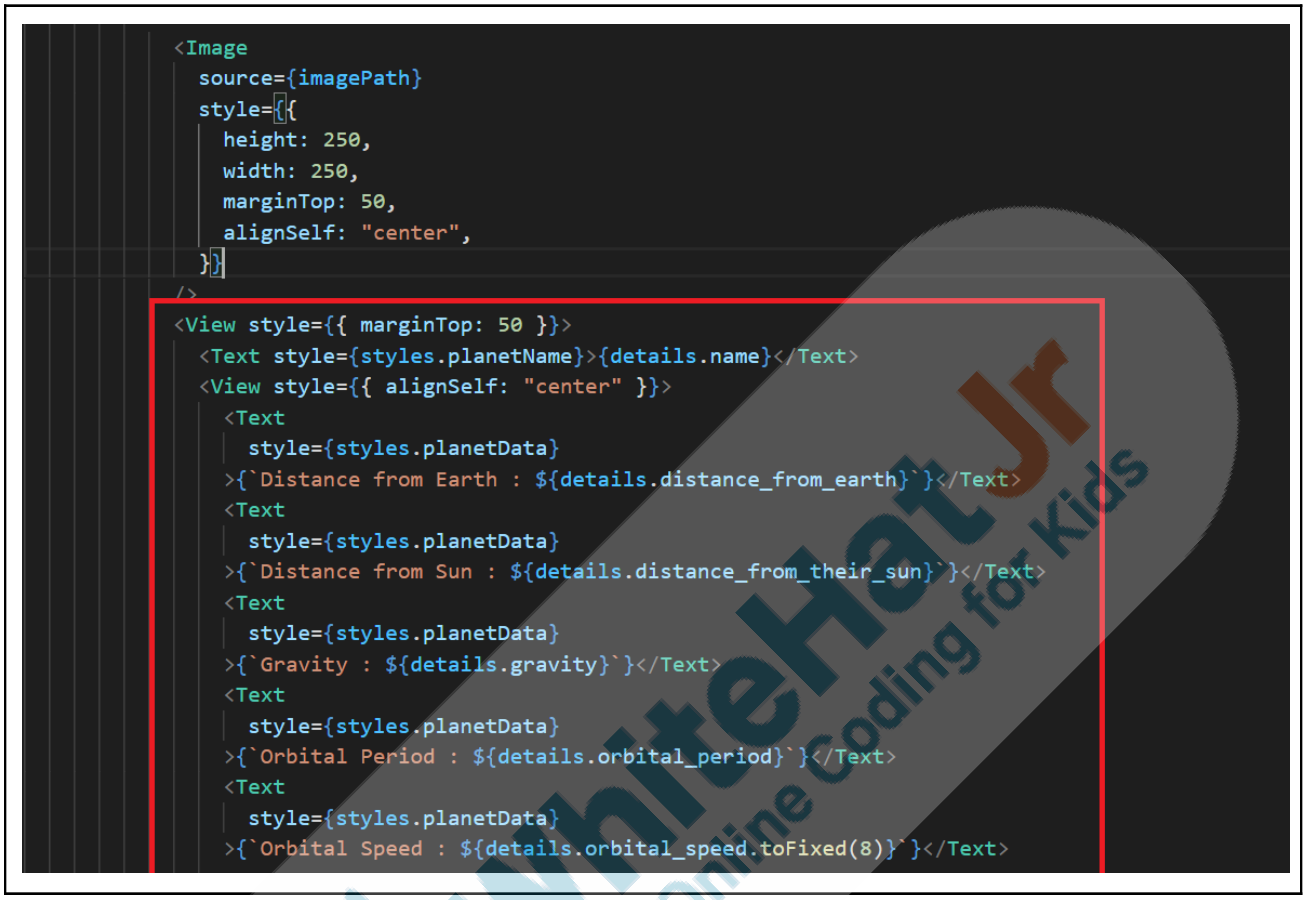
```
render() {
  const { details, imagePath } = this.state;
  if (details.specifications) {
    return (
      <View style={styles.container}>
        <ImageBackground
          source={require("../assets/bg.png")}
          style={{ flex: 1, paddingTop: 20 }}
        >

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

Now, let’s render each data about the planet one by one. We will start with the planet image.

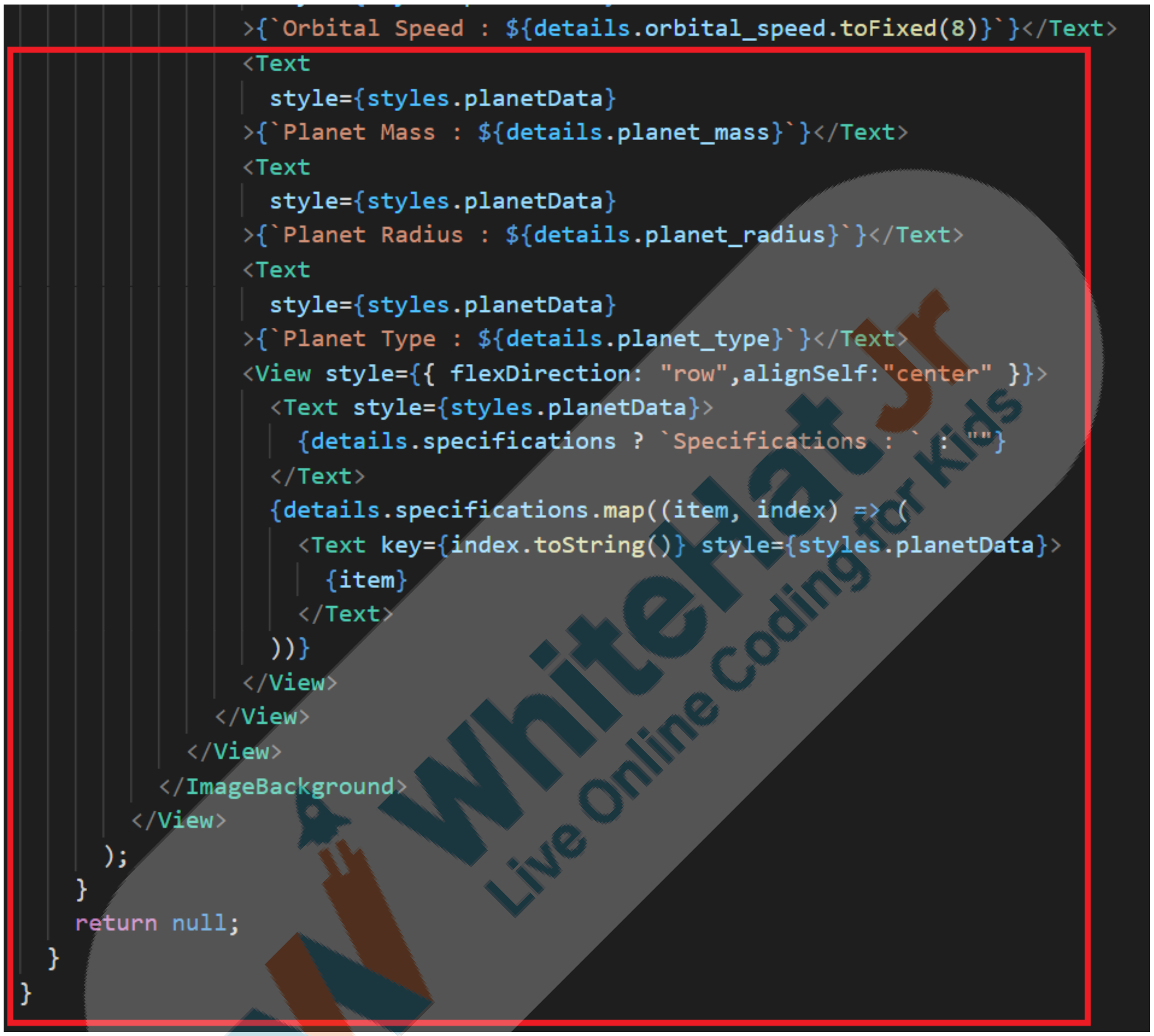


```
render() {
  const { details, imagePath } = this.state;
  if (details.specifications) {
    return (
      <View style={styles.container}>
        <ImageBackground
          source={require("../assets/bg.png")}
          style={{ flex: 1, paddingTop: 20 }}
        >
          <Image
            source={imagePath}
            style={{
              height: 250,
              width: 250,
              marginTop: 50,
              alignSelf: "center",
            }}
          />
```

Now, we will use **<View>** and **<Text>** components to render the rest of the data on the screen.



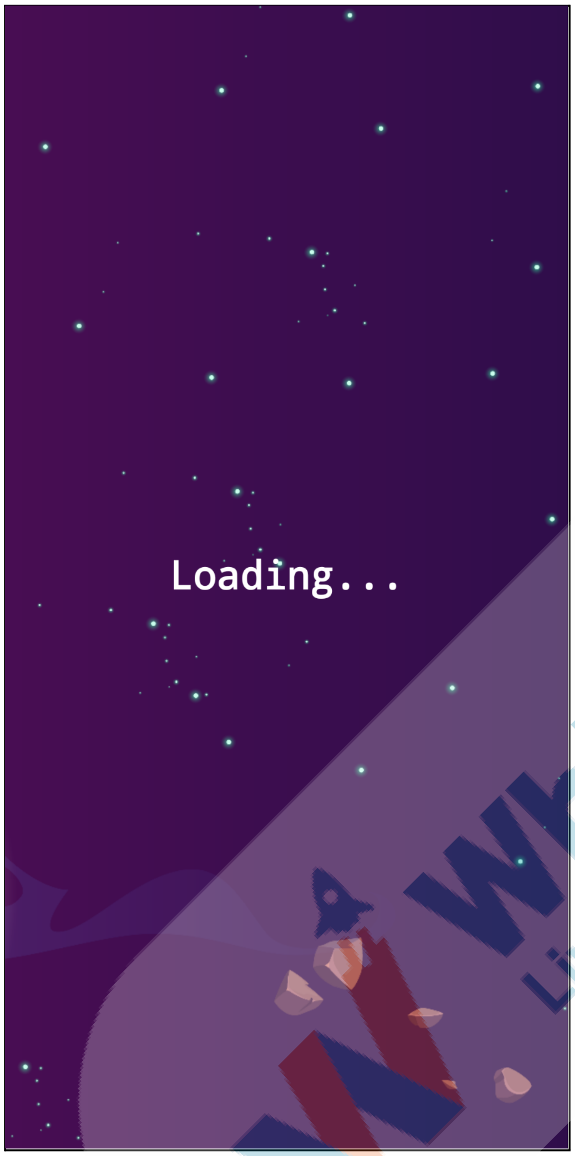
```
<Image
  source={imagePath}
  style={{
    height: 250,
    width: 250,
    marginTop: 50,
    alignSelf: "center",
  }}
/>
<View style={{ marginTop: 50 }}>
  <Text style={styles.planetName}>{details.name}</Text>
  <View style={{ alignSelf: "center" }}>
    <Text
      style={styles.planetData}
    >{`Distance from Earth : ${details.distance_from_earth}`}</Text>
    <Text
      style={styles.planetData}
    >{`Distance from Sun : ${details.distance_from_their_sun}`}</Text>
    <Text
      style={styles.planetData}
    >{`Gravity : ${details.gravity}`}</Text>
    <Text
      style={styles.planetData}
    >{`Orbital Period : ${details.orbital_period}`}</Text>
    <Text
      style={styles.planetData}
    >{`Orbital Speed : ${details.orbital_speed.toFixed(8)}`}</Text>
```

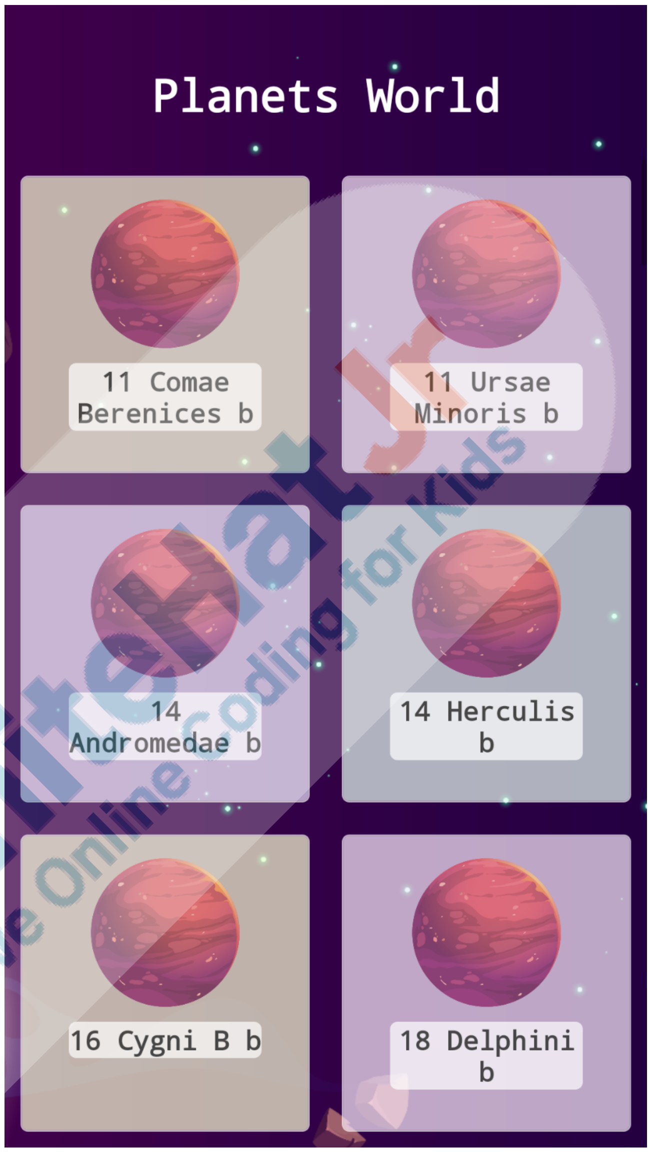


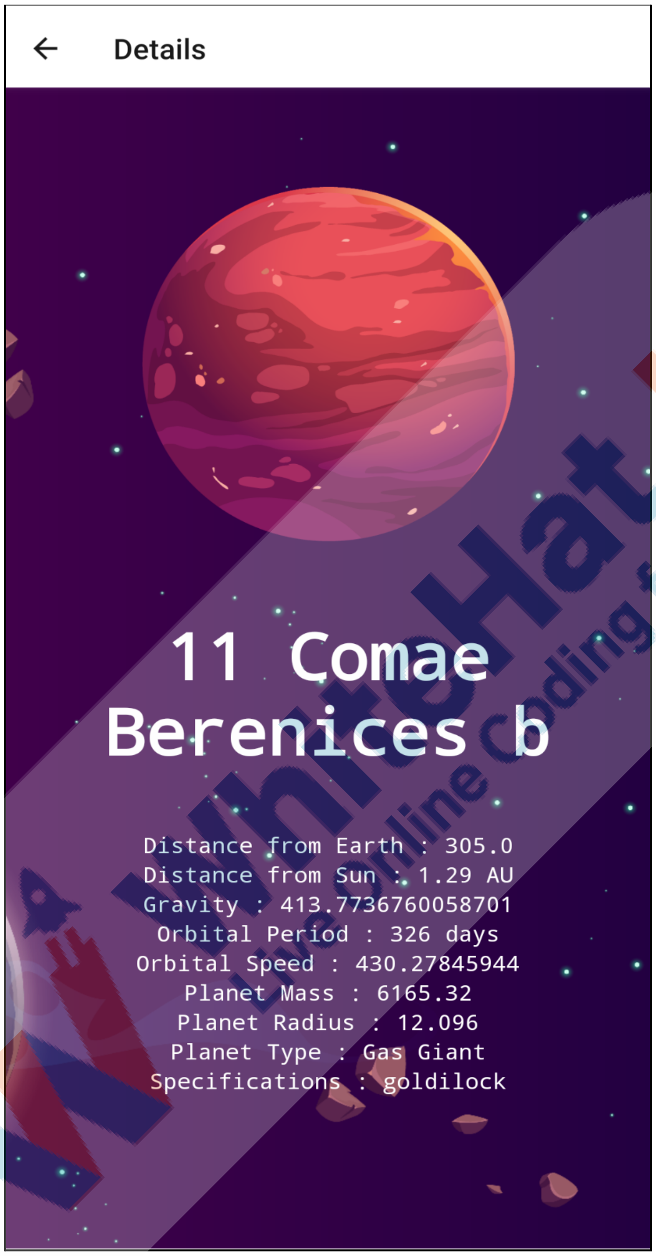
```
                >{`Orbital Speed : ${details.orbital_speed.toFixed(8)}`}</Text>
                <Text
                  style={styles.planetData}
                >{`Planet Mass : ${details.planet_mass}`}</Text>
                <Text
                  style={styles.planetData}
                >{`Planet Radius : ${details.planet_radius}`}</Text>
                <Text
                  style={styles.planetData}
                >{`Planet Type : ${details.planet_type}`}</Text>
                <View style={{ flexDirection: "row",alignSelf:"center" }}>
                  <Text style={styles.planetData}>
                    {details.specifications ? `Specifications : ` : ""}
                  </Text>
                  {details.specifications.map((item, index) => (
                    <Text key={index.toString()} style={styles.planetData}>
                      {item}
                    </Text>
                  ))}
                </View>
              </View>
            </View>
          </ImageBackground>
        </View>
      );
    }
    return null;
  }
}
```

Now, our details screen is ready. Let’s run the code.	

Finally our App will look like this-


Loading...


Planets World
11 Comae Berenices b
11 Ursae Minoris b
14 Andromedae b
14 Herculis b
16 Cygni B b
18 Delphini b



11 Comae Berenices b

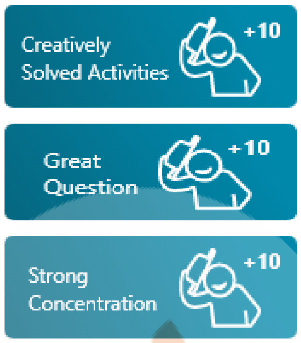
```
Distance from Earth : 305.0
Distance from Sun : 1.29 AU
Gravity : 413.7736760058701
Orbital Period : 326 days
Orbital Speed : 430.27845944
Planet Mass : 6165.32
Planet Radius : 12.096
Planet Type : Gas Giant
Specifications : goldilock
```

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 class for:*

In the next class, we'll be working on the Movie recommendation system.	 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	Sample output of Exoplanet Catalog	https://s3-whjr-curriculum-uploads.whjr.online/6b24a6bb-3dce-4b9c-9f74-6d23406cc411.gif
Teacher Activity 2	Previous Class Code	https://colab.research.google.com/drive/1g3ZFIwBxw9tQXsEXSJPvvOVOsS_hSROZ?usp=sharing
Teacher Activity 3	Boilerplate Code	https://github.com/procodingclass/PRO-C137-Student-Boilerplate
Teacher Activity 4	Reference Code	https://github.com/procodingclass/PRO-C137-Reference-Code
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/6b625d12-1f55-47cf-ba2c-461ac6d7da1e.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C137-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/66ebbb1c-f0c9-4083-b0a6-ab91361d1b6a.pdf
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C137-Student-Boilerplate