Topic	Exoplanet Catalog	
Class Description	**In this class, we will be bundling a mobile app which will be a catalog for exo-planets based on all the data we curated.**	
Class	**C137**	
Class time	**45 Min**	
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 ● Student Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm Up** **Teacher-led Activity** **Student-led Activity** **Wrap up**	**5 Minutes** **5 Minutes** **30 Minutes** **5 Minutes**

CONTEXT

- **We will be building a React Native App**
- **We will integrate this React Native App with Flask API**

Class Steps	Teacher Action	Student Action
Step 1: Warm Up (5 Mins)	Hi <Student Name>! We have finally created a Flask API with all the curated data! Now, it's time to create an Exo-planet Catalog using React Native.	**ESR:** We created 2 APIs in the last class:

	Can you tell me how many APIs we created in the last class and what were they?	• First to get data for all the exo-planets • Second, to get data for a particular exo-planet
	I have an exciting quiz question for you! Are you ready to answer this question? Teacher click on the  button on the bottom right corner of your screen to start the In-Class Quiz. A quiz will be visible to both you and the student. Encourage the student to answer the quiz question. The student may choose the wrong option, help the student to think correctly about the question and then answer again. After the student selects the correct option, the  button will start appearing on your screen. Click the End quiz to close the quiz pop-up and continue the class.	
	Great! Now in this class, you will create a react native app with 2 screens. 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
	Let's start!	
Teacher Initiates Screen Share		
CHALLENGE • **Ask the student to set up the basic project** • **Guide the student while building the mobile app** • **Guide the student on integrating with Flask API**		
Step 2: Teacher-led Activity (5 Mins)	*<In this class, the student will be doing most of the coding with teacher's guidance if needed.>* ***The teacher is required to:*** 1. ***Help the student in creating:*** • ***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***	
Teacher Stops Screen Share		
	Now it's your turn. Please share your screen with me.	
• **Ask Student to press ESC key to come back to panel** • **Guide Student to start Screen Share**		

- **Teacher gets into Fullscreen**

ACTIVITY

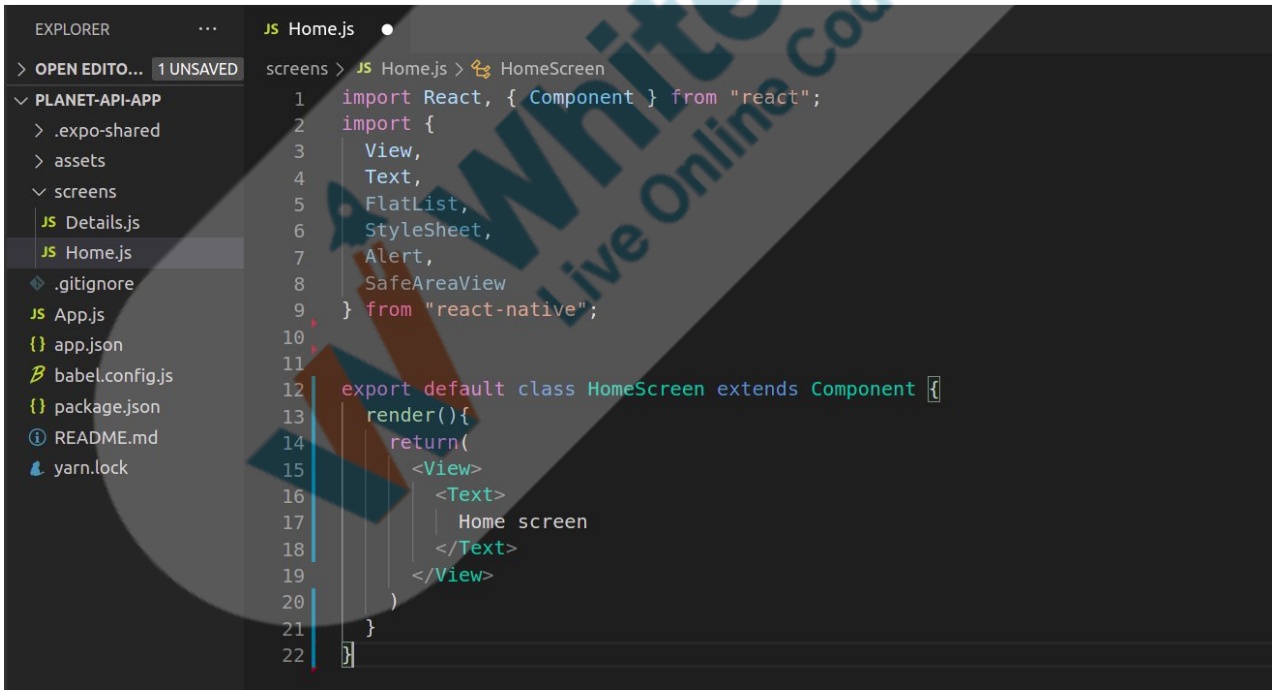
- **React Native Code**
- **Flask API Integration**

Step 3: Student-Led Activity (30 Minutes)	So you already know how to make an react native app. Let's start by creating a new expo project. *<Teacher helps student in creating a new expo project using "expo init Planet-App">*	*Student creates a new project using command "expo init Planet-App" and selects a blank screen.*

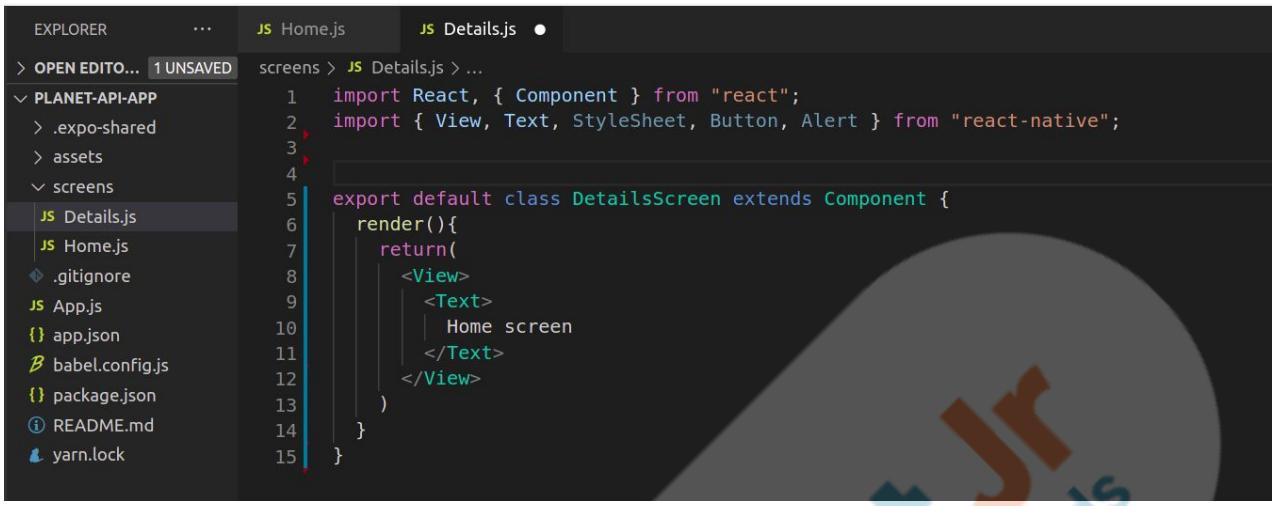
```
ashura@ashura-Lenovo-ThinkBook-14-IML:~/Desktop$ expo init Planet-App

   There is a new version of expo-cli available (3.28.1).
   You are currently using expo-cli 3.27.8
   Install expo-cli globally using the package manager of your choice;
   for example: `npm install -g expo-cli` to get the latest version

? Choose a template: (Use arrow keys)
  ----- Managed workflow -----
❯ blank                  a minimal app as clean as an empty canvas
  blank (TypeScript)     same as blank but with TypeScript configuration
  tabs (TypeScript)      several example screens and tabs using react-navigation and TypeScript
  ----- Bare workflow -----
  minimal                bare and minimal, just the essentials to get you started
  minimal (TypeScript)   same as minimal but with TypeScript configuration
```

	Now let's open the Project in our VS Code editor. So how many screens will our app have?	*The student opens the project in VS Code Editor.* **ESR:** Our app will have 2 screens. 1st screen will contain the list of the planets and the 2nd screen will contain the information of the planet chosen from the first screen.

	Perfect!.For now let's create 2 blank screens. Home.js will contain the list of all the planets and Details.js will contain the list of the selected planet. *<Teacher helps student to create Home.js and Details.js screen inside a* ***screens*** *folder.>* Can you tell me how can we do that?	*Student codes to create blank Home.js and Details.js screens inside a screens folder.* **ESR:** We'll create a HomeScreen component . Inside that component we'll add a plain text saying Home screen. Same for the details screen.



Details Screen



	Now our screens are ready. Let's add them to navigation so that we can navigate from one screen to another. To do this we'll be using Stack Navigation. To use stack navigation we'll need to install the library first to our project. So we need the **react-navigation** and **react-navigation-stack** library. We can install them using commands: **npm install react-navigation and npm install react-navigation-stack**	*Student codes to install the libraries using commands* ***npm install react-navigation and npm install react-navigation-stack***

```
npm install react-navigation
```

```
npm install react-navigation-stack
```

	Now let's import the createAppContainer from react-navigation and createStackNavigator from react-navigation-stack. Also let's import the Home.js screen and Details.js screen from the screens folder. Then we'll create a Stack Navigator and add the screens to it. Our initial route will be - home screen as we want our user to first see the home screen.	*Student codes to import createAppContainer from react-navigation and createStackNavigator from react-navigation-stack.* *And import Home.js screen and Details.js screen from the screens folder.*

```javascript
JS App.js > ...
import React from "react";
import { createAppContainer } from "react-navigation";
import { createStackNavigator } from "react-navigation-stack";
import HomeScreen from "./screens/Home";
import DetailsScreen from "./screens/Details";

export default function App() {
  return <AppContainer />;
}

const appStackNavigator = createStackNavigator(
  {
    Home: {
      screen: HomeScreen,
      navigationOptions: {
        headerShown: false
      }
    },
    Details: {
      screen: DetailsScreen
    }
  },
  {
    initialRouteName: "Home"
  }
);

const AppContainer = createAppContainer(appStackNavigator);

```

	Our screens are now added in the stack navigation. 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 the flat list. To make a GET request on an API we'll be using axios. To install axios we use command **npm install axios**. And to show different items we'll be using ListItems from 'react-native-elements'.	*Student codes to install axios and react-native-elements and then import them in the Home.js file.* *Student codes to create a constructor which contains a listData which will have the list of all the planets and an Url of the site on which the GET request is to be made.*

	We'll use the command 'npm install react-native-elements'.	*The student also codes to write a function called getPlanets which will make a GET request on the url and when it finds the data it will set it to the **listData** list or give an error if it doesn't find the data and call this function in the componentDidMount function.*

screens > JS Home.js > HomeScreen > constructor

```
1  import React, { Component } from "react";
2  import {
3    View,
4    Text,
5    FlatList,
6    StyleSheet,
7    Alert,
8    SafeAreaView
9  } from "react-native";
10 import { ListItem } from "react-native-elements";
11 import axios from "axios";
12
13 export default class HomeScreen extends Component {
14   constructor(props) {
15     super(props);
16     this.state = {
17       listData: [],
18       url: "http://localhost:5000/"
19     };
20   }
```

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

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

	Now using the Flat list we'll show the data. In the renderItem property of the flatlist we'll use the ListItem to show the title as planet's name and distance from earth as the subtitle. When the user presses an item we want the user to navigate to the Details screen where more details of that planet will be shown. So on press of an item **we'll navigate to the details screen and pass the planet name along with it.**	*Student codes to create a flatlist and uses a List item to show the title as planet's name and distance from earth as the subtitle.*

	How can we do that?	**ESR:** We'll use the flat list component to show the list of the planets . Flatlist has a prop called renderItems which helps us to render the elements of the list . Inside this renderItems we use ListItem component to add more details of the item such as title, subtitle. On the onPress property of the ListItem we'll navigate to the Details screen and also pass the planet name which the user clicks on,

```
renderItem = ({ item, index }) => (
  <ListItem
    key={index}
    title={`Planet : ${item.name}`}
    subtitle={`Distance from earth : ${item.distance_from_earth}`}
    titleStyle={styles.title}
    containerStyle={styles.listContainer}
    bottomDivider
    chevron
    onPress={() =>
      this.props.navigation.navigate("Details", { planet_name: item.name })
    }
  />
);

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

```
  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>
    );
  }
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#edc988"
  },
  upperContainer: {
    flex: 0.1,
    justifyContent: "center",
    alignItems: "center"
  },
  headerText: {
    fontSize: 30,
    fontWeight: "bold",
    color: "#132743"
  },
  lowerContainer: {
    flex: 0.9
  },
  emptyContainer: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center"
  },
  emptyContainerText: {
    fontSize: 20
  },
  title: {
    fontSize: 18,
    fontWeight: "bold",
    color: "#d7385e"
  },
  listContainer: {
    backgroundColor: "#eeecda"
  }
});
```

	Ok now our Home screen is ready. Now let's code for a Details screen where the details of that particular planet will be shown along with an image of its type.	*Student opens the Details.js screen.*
	First we'll create a state which will contain the details of the planet, imagePath of the planet and the url of the site. Here our url will contain the name parameter of the planet which we passed while navigating to the Details screen.	*Student codes to create a state which contains the details of the planet, imagePath of the planet and the url of the site.*

```
constructor(props) {
  super(props);
  this.state = {
    details: {},
    imagePath: "",
    url: `http://localhost:5000/planet?name=${this.props.navigation.getParam(
      "planet_name"
    )}`
  };
}
```

	We'll now write a **getDetails** function which will make a GET request on the given url and get the data and set it to **setDetails** function. This **setDetails** function will check the planet type from the given details and, using switch case, return the image path of the planet from the assets folder. And then set the details and image path to the state.	*Student downloads the images from the student Activity 1.* *The student adds the images to the assets folder.* *Student codes to write the **getDetails** function and **setDetails function.***

Home.js | Details.js | App.js

screens > Details.js > DetailsScreen > render

```
17  componentDidMount() {
18    this.getDetails();
19  }
20  getDetails = () => {
21    const { url } = this.state;
22    axios
23      .get(url)
24      .then(response => {
25        this.setDetails(response.data.data);
26      })
27      .catch(error => {
28        Alert.alert(error.message);
29      });
30  };
31
32  setDetails = planetDetails => {
33    const planetType = planetDetails.planet_type;
34    let imagePath = "";
35    switch (planetType) {
36      case "Gas Giant":
37        imagePath = require("../assets/planet_type/gas_giant.png");
38        break;
39      case "Terrestrial":
40        imagePath = require("../assets/planet_type/terrestrial.png");
41        break;
42      case "Super Earth":
43        imagePath = require("../assets/planet_type/super_earth.png");
44        break;
45      case "Neptune Like":
46        imagePath = require("../assets/planet_type/neptune_like.png");
47        break;
48      default:
49        imagePath = require("../assets/planet_type/gas_giant.png");
50    }
51
52    this.setState({
53      details: planetDetails,
54      imagePath: imagePath
55    });
```

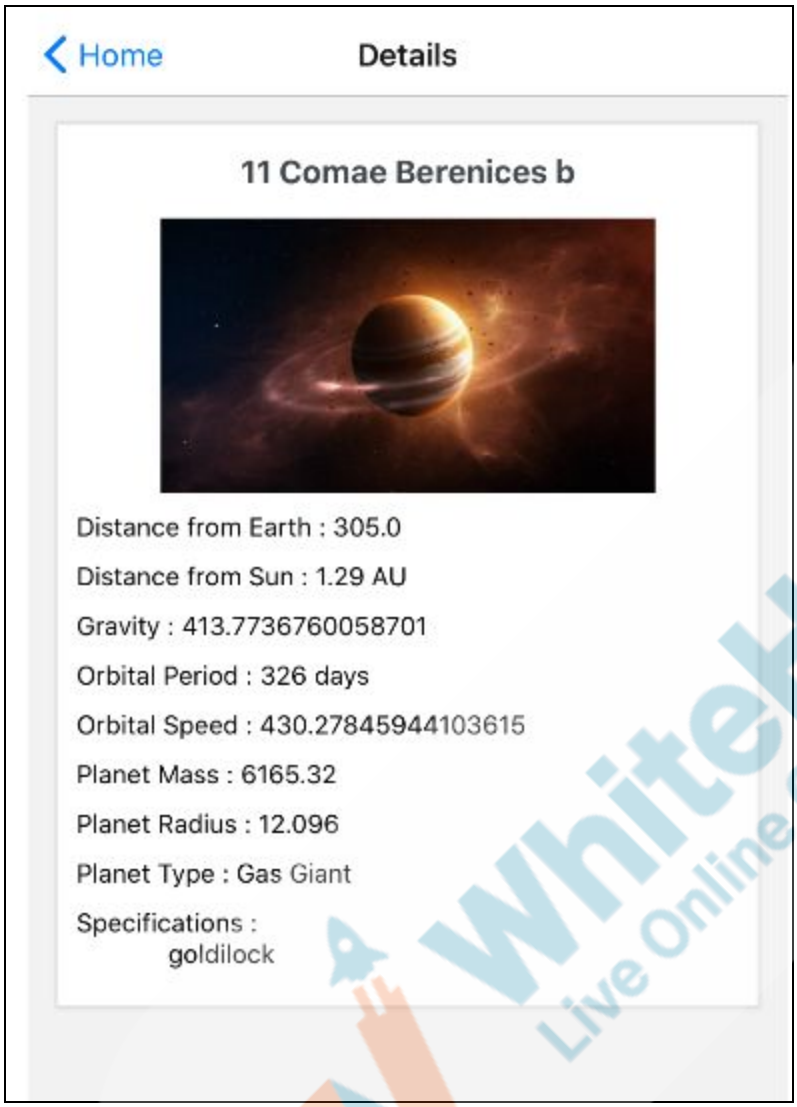
	To display all this information we'll use the cards from the react-native-elements. Remember we had used cards earlier in our book santa app. So can you tell me how can we do that?	*Student codes to import the cards from the react native elements and display the information on it.* **ESR:** We will import the **cards** from the **react native elements.**

	So first we'll import the cards from the react native elements and then use it to display the details.	In this card we'll display the name of the planet as title, image of the type of the planet in the main View component . In another view component we'll show the distance from earth, Distance from sun, Gravity, Orbital Period , Orbital Speed, Planet Mass, Planet Radius and Planet Type using the **Text** Component.

```
render() {
  const { details, imagePath } = this.state;
  if (details.specifications) {
    return (
      <View style={styles.container}>
        <Card
          title={details.name}
          image={imagePath}
          imageProps={{ resizeMode: "contain", width: "100%" }}
        >
          <View>
            <Text
              style={styles.cardItem}
            >{`Distance from Earth : ${details.distance_from_earth}`}</Text>
            <Text
              style={styles.cardItem}
            >{`Distance from Sun : ${details.distance_from_their_sun}`}</Text>
            <Text
              style={styles.cardItem}
            >{`Gravity : ${details.gravity}`}</Text>
            <Text
              style={styles.cardItem}
            >{`Orbital Period : ${details.orbital_period}`}</Text>
            <Text
              style={styles.cardItem}
            >{`Orbital Speed : ${details.orbital_speed}`}</Text>
            <Text
              style={styles.cardItem}
            >{`Planet Mass : ${details.planet_mass}`}</Text>
            <Text
              style={styles.cardItem}
            >{`Planet Radius : ${details.planet_radius}`}</Text>
            <Text
              style={styles.cardItem}
            >{`Planet Type : ${details.planet_type}`}</Text>
          </View>
          <View style={[styles.cardItem, { flexDirection: "column" }]}>
            <Text>{details.specifications ? `Specifications : ` : ""}</Text>
            {details.specifications.map((item, index) => (
```

```
            <View style={[styles.cardItem, { flexDirection: "column" }]}>
              <Text>{details.specifications ? `Specifications : ` : ""}</Text>
              {details.specifications.map((item, index) => (
                <Text key={index.toString()} style={{ marginLeft: 50 }}>
                  {item}
                </Text>
              ))}
            </View>
          </Card>
        </View>
      );
    }
    return null;
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1
  },
  cardItem: {
    marginBottom: 10
  }
});
```

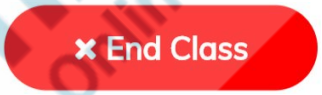
	So our Details screen is also ready now. Let's run and test the code.	*The student runs and tests the code.*



Teacher Guides Student to Stop Screen Share

FEEDBACK

- **Appreciate the student for their efforts**
- **Identify 2 strengths and 1 area of progress for the student**

Step 4: Wrap-Up (5 Min)	In this class, we completed the SpaceTech module where we performed a lot of analysis and calculations, built a Flask API and now a mobile app. How was your experience?	**ESR:** Varied
	Amazing! In the next class, we will get started with the Capstone Classes!	
Teacher Clicks		

Activity	Activity Name	Links
Teacher activity 1	Solution	https://github.com/whitehatjr/Planet-Api-App
Student Activity 1	Planet images	https://github.com/whitehatjr/Planet-Image-assets/tree/main/assets/planet_type