Topic	ISS LOCATION	
Class Description	**The student codes to create an ISS Location screen. The student learns to use the React Native maps to use maps on the screen. The student also learns to use an API to fetch the coordinates of the ISS.**	
Class	**C78**	
Class time	**45 mins**	
Goal	• Use the React Native maps library to add the maps to the app. • Display the location of the ISS on the map. • Display the latitude, longitude, altitude, and velocity of the ISS on the screen.	

Resources Required	• Teacher Resources
	  ○ Visual Code Studio Editor
	  ○ laptop with internet connectivity
	  ○ earphones with mic
	  ○ notebook and pen
	• Student Resources
	  ○ Visual Code Studio Editor
	  ○ laptop with internet connectivity
	  ○ earphones with mic
	  ○ notebook and pen	

Class structure	**Warm-Up** **Teacher-led Activity** **Student-led Activity** **Wrap-Up**	**5 mins** **15 mins** **15 mins** **5 mins**
Credits	• **Open-source API for Getting the live location of the ISS Tracker offered by https://wheretheiss.at/**	

WARM-UP SESSION - 5 mins	
Teacher starts slideshow  **from slides 1 to 8** Refer to speaker notes and follow the instructions on each slide.	
Teacher Action	**Student Action**
Run the presentation from slide 1 to slide 3 to revise concepts. **The following are the warm-up session deliverables:** • Warm-Up Quiz Session • Wireframe of ISSTracker App	Click on the slide show tab and present the slides.
QnA Session - Click on in-class quiz	
Question	**Answer**
Select the correct code block that helps in giving style to the text in the title. A. `style={styles.titleText}` B. `style={{styles.titleText}}` C. `style=styles.titleText` D. `style={titleText}`	**A**

Choose the correct block of code which can be used to set the state whenever the text inside the textInput is changed.
A.
`// onChangeText={(text) => {`
`//  this.State({`
`//  longitude: text`
`//  })`
`// }}`
	D

B.

```
// onChangeText={(text) => {
//     this.setState(
//         longitude: text
//     )
// }}
```

C.

```
// onChangeText={() => {
//     this.setState({
//         longitude:
//     })
// }}
```

D.

```
// onChangeText={(text) => {
//     this.setState({
//         longitude: text
//     })
// }}
```

Continue the warm-up session	
Teacher Action	**Student Action**
Run the presentation from slide 4 to slide 8 to set the problem statement.	Narrate the slides by using hand gestures and voice modulation methods to bring in more student interest.
Teacher ends slideshow 	
TEACHER-LED ACTIVITY 1 -15 mins	
Teacher Initiates Screen Share	

ACTIVITY • **Add title and map to the ISS Location screen.** • **Mark the location of the ISS using an ISS Icon.**	
Teacher Action	**Student Action**
<The teacher opens the boilerplate code> *Teacher Activity 5*	
What are the things that we want to show on the screen?	**ESR:** - We want to show the title of the screen. - Then we want to show the map with the current location of the ISS. - We also want to show the coordinates of the ISS.
So first this is where we want to add the title to the screen and while we are at it, let's also add a background image to the screen as well. To do so, we'll need to: • Import the **ImageBackground** component to add/import the image to the background. • **SafeAreaView** component to add space for the status bar. • **StyleSheet** to add styles. • And import the **StatusBar** component from React Native to add the status bar. *(This is a horizontal bar, usually at the bottom of the screen or at the top showing information about a program running.)* *In screens/IssLocation.js in the boilerplate code -*	

First, we have a **View** container; inside this view we have a **SafeAreaView**; inside the **SafeAreaView** component we'll have an **ImageBackground** component.

And finally, inside the **ImageBackground** component, we have the **View** and **Text** components for the title.
We'll also add the styling to it.

<The teacher shows the output to the student.>

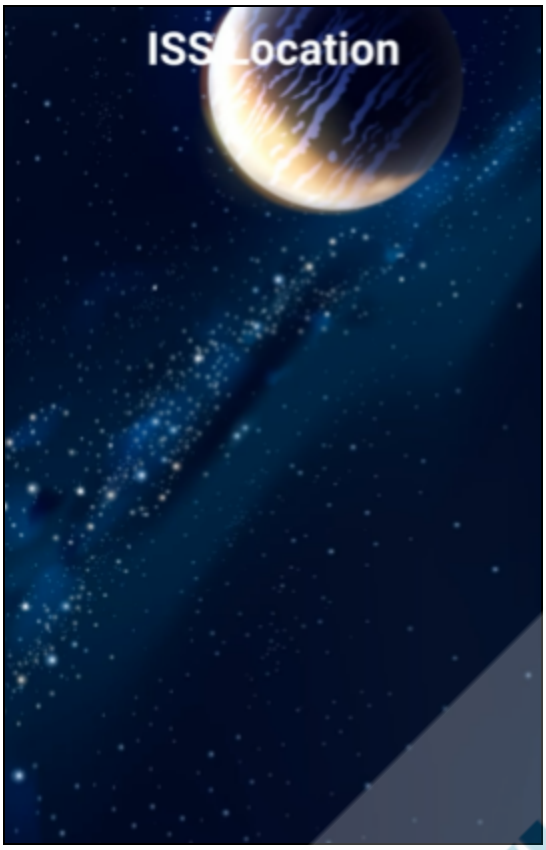
screens > IssLocation.js

```
import React, { Component } from 'react';
import { Text, View ,StyleSheet,ImageBackground,StatusBar, SafeAreaView} from 'react-native';

export default class IssLocationScreen extends Component {
    render() {
        return (
            <View style={styles.container}>
                <SafeAreaView styles={styles.droidSafeArea}/>
                <ImageBackground source={require('../assets/iss_bg.jpg')} style={styles.backGroundImage}>
                    <View style={styles.titleContainer}>
                        <Text style={styles.titleText}>ISS Location</Text>
                    </View>
                </ImageBackground>
            </View>
        )
    }
}

const styles = StyleSheet.create({
    container:{
        flex:1
    },
    droidSafeArea: {
        marginTop: Platform.OS === "android" ? StatusBar.currentHeight : 0
    },
    backGroundImage:{
        flex: 1,
        resizeMode: 'cover',
    },
    titleContainer: {
        flex: 0.1,
        justifyContent: "center",
        alignItems: "center"
    },
    titleText: {
        fontSize: 30,
        fontWeight: "bold",
        color: "white"
    },
```

Output:



What do we need next? Now that we have the title, let's add a Map to the screen. We'll install the **maps** library using the **expo** command. *<The teacher installs the **maps** library using the command **"expo install react-native-maps"**.>*	**ESR:** We'll be needing the map on which we'll be showing the current location of the ISS.

Use the command *expo install react-native-maps*

```
/ISS-Tracker$ expo install react-native-maps
```

In code, we'll import the **MapView** and **Marker** components from the **react-native-maps**.

- **MapView** will help us add the map.
- **Marker** will help us mark the location on the map.

*<Teacher refers to Teacher Activity 3 for **MapView's** reference as shown below>*

<Student refers to Student Activity 2 for MapView's reference>

```
<MapView
  initialRegion={{
    latitude: 37.78825,
    longitude: -122.4324,
    latitudeDelta: 0.0922,
    longitudeDelta: 0.0421,
  }}
/>
```

What is the other thing that we are missing here?

Correct, we are missing the coordinates of the ISS.

We have an API from where we can get the ISS's current location's data - https://api.wheretheiss.at/v1/satellites/25544.

ESR:
We are missing the location data.

Now you have worked with the APIs in the past. What's your understanding of APIs, and how did we use them earlier? *<The teacher opens the API and checks for the data.>*	**ESR:** We can write a function that will help us make a request on the API and get the data.


```
{"name":"iss","id":25544,"latitude":-37.264086390933,"longitude":-47.601405349805,"altitude"
:431.75316730831,"velocity":27548.608651074,"visibility":"eclipsed","footprint":4566.7813851
089,"timestamp":1612153955,"daynum":2459246.689294,"solar_lat":-17.041142488018,"solar_lon":
115.247552949,"units":"kilometers"}
```


How can we get the data from the API? We'll write a function called **getIssLocation()** in which using axios can be used when we want to fetch data from an external source. Here we are using axios to request the given API to fetch the data and set the data returned in a state called as **location**. Currently, we don't have axios installed so let's install it using commands: **"yarn add axios"** or **"npm add axios"** and then import it to the screen. *<The teacher installs axios using the commands **"yarn add axios"** or **"npm add axios"**.>*	**ESR:** Varied *<The student helps the teacher with the code.>*

```
ISS-Tracker$ npm add axios
```

Let's write the **getIssLocation()** function. In the function we are: ● Using the **axios.get()** function to get the data from the API and set the response to the **location** (declared in the **constructor()** as an array in the state using **.then**) ● Using **.catch** we'll catch any error that we get and show it as a message using the **Alert.alert()** function. ● We'll call this function in the **componentDidMount()** as we want the **getIssLocation()** to run when the screen is loaded. *<The teacher codes to write the **getIssLocation()** function and calls it in the **ComponentDidMount()** function.>*	-

```
export default class IssLocationScreen extends Component {
    constructor(props) {
        super(props);
        this.state = {
            location: {},
        };
    }

    componentDidMount() {
        this.getIssLocation()
    }

    getIssLocation = () => {
        axios
            .get("https://api.wheretheiss.at/v1/satellites/25544")
            .then(response => {
                this.setState({ location: response.data })
            })
            .catch(error => {
                Alert.alert(error.message)
            })
    }
```

We have the data now, let's add the map to the screen. To do so we'll use the **MapView** component. **MapView** has a property called as **region** using which we can set the map of the desired location. Here we'll use the **latitude** and the **longitude** that we get from the **getIssLocation()** function and set the map. Finally, also add some styling such as width and height for the map. *<The teacher codes to set the map using the **MapView** component and using the longitude and latitude data in the **region** property to set the map and add the height and width as the width.>*	*The student helps the teacher with the code.*

***Note: MapView** is not supported by the Web, therefore it is essential to only run the App on mobile phones using **Expo**.*

```
import MapView, { Marker } from 'react-native-maps';
```

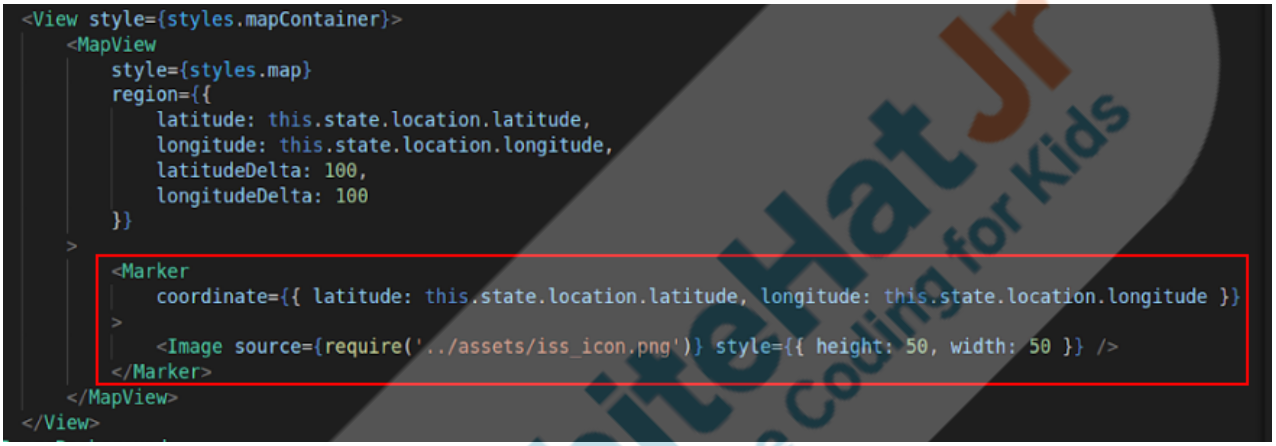
Using **MapView** in the **View** to display the map.

```
        return (
            <View style={styles.container}>
                <SafeAreaView style={styles.droidSafeArea} />
                <ImageBackground source={require('../assets/iss_bg.jpg')} style={styles.backgroundImage}>
                    <View style={styles.titleContainer}>
                        <Text style={styles.titleText}>ISS Location</Text>
                    </View>
                    <View style={styles.mapContainer}>
                        <MapView
                            style={styles.map}
                            region={{
                                latitude: this.state.location.latitude,
                                longitude: this.state.location.longitude,
                                latitudeDelta: 100,
                                longitudeDelta: 100
                            }}
                        >
                        </MapView>
                    </View>
                </ImageBackground>
            </View>
        );
    }
}
```

Adding styles to the Map as shown below:

```
const styles = StyleSheet.create({
    container: {
        flex: 1
    },
    droidSafeArea: {
        marginTop: Platform.OS === "android" ? StatusBar.currentHeight : 0
    },
    backgroundImage: {
        flex: 1,
        resizeMode: 'cover',
    },
    titleContainer: {
        flex: 0.1,
        justifyContent: "center",
        alignItems: "center"
    },
    titleText: {
        fontSize: 30,
        fontWeight: "bold",
        color: "white"
    },
    refeshContainer: {
        flex: 0.1,
        justifyContent: "center",
        alignItems: "center"
    },
    mapContainer: {
        flex: 0.6
    },
    map: {
        width: "100%",
        height: "100%"
    }
});
```

Now that we are able to fetch the ISS location, what is the next step?	**ESR:** We still have to add the location for the ISS on the map.
How can we do that?	**ESR:** We can use the **Marker** component.
Yes! The **Marker** component has a property called as **coordinate** which takes latitude and longitude to add the marker. Let's also add an ISS icon as the marker.	

Can you tell me how can we do that? *<The teacher codes to add the **Marker** using the **Marker** component* *And also use the **Image** component to add the ISS icon as the marker.>*	**ESR:** We can use the image component to add the ISS icon as the marker.
 	
As we open the ISS Location screen for the first time we won't have any content to show, so what can we do? Yes! Till the time our **getIssLocation()** function is called, we'll add a loading text which will appear as the function is called. *<The teacher codes to show the Loading text.>*	**ESR:** We can have a loading screen.

```
render() {
    if (Object.keys(this.state.location).length === 0) {
        return (
            <View
                style={{
                    flex: 1,
                    justifyContent: "center",
                    alignItems: "center"
                }}>
                <Text>Loading</Text>
            </View>
        )
    } else {
        return (
            <View style={styles.container}>
                <SafeAreaView style={styles.droidSafeArea} />
                <ImageBackground source={require('../assets/iss_bg.jpg')} style={styles.backgroundImage}>
                    <View style={styles.titleContainer}>
                        <Text style={styles.titleText}>ISS Location</Text>
                    </View>
                    <View style={styles.refeshContainer}>
                        <TouchableOpacity style={{ width: 100, height: "100%", alignItems: "center" }} onPress={() =>
                            this.setState({})
                        }>
                            <Image source={require("../assets/refresh_icon.jpg")} style={{ width: 50, height: 50 }}></Image>
                        </TouchableOpacity>
                    </View>
                    <View style={styles.mapContainer}>
                        <MapView
                            style={styles.map}
                            region={{
                                latitude: this.state.location.latitude,
                                longitude: this.state.location.longitude,
                                latitudeDelta: 100,
                                longitudeDelta: 100
                            }}
                        >
```

Here, the ***Object.Keys()*** is a widely used JavaScript method that returns an array of all the keys of an Object. It has the following syntax -

Object.Keys(<your object>)

Since we have an Object in our **state "location",** we are passing ***this.state.location*** inside our ***Object.keys()*** function and checking if the length of keys is **0** or **not**. If it is **0**, we display loading else we simply display the contents of the screen.

Now let's check the output. *<The teacher runs the code and checks the output.>*	

Currently, we can just see the position of the ISS and the data we have is not updating continuously so the ISS is fixed at one spot. What can we do to get the data updated continuously?	**ESR:** We can have the

<table>
<tr><td colspan="2"></td><td>user-defined function to get this data called after set intervals of time to get the continuous data and also show it on the screen.</td></tr>
<tr><td colspan="2">Awesome, would you like to give it a try? I'll guide you when you get stuck.</td><td>ESR:
Yes.</td></tr>
<tr><td colspan="2">Let's get you started then.</td><td></td></tr>
<tr><td colspan="3">Teacher Stops Screen Share</td></tr>
<tr><td></td><td>Now it's your turn. Please share your screen with me.</td><td></td></tr>
<tr><td colspan="3">STUDENT-LED ACTIVITY - 15 mins</td></tr>
<tr><td colspan="3">• 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 Fullscreen.</td></tr>
<tr><td colspan="3">Teacher starts slideshow  for slide 9 & 10
Refer to speaker notes and follow the instructions on each slide.</td></tr>
<tr><td colspan="3"><u>ACTIVITY</u>
• Create a new component screen.
• Write the function to get location data of the ISS and display it using a card.</td></tr>
<tr><td colspan="2">Teacher Action</td><td>Student Action</td></tr>
</table>

<The teacher guides the student to open code from Student activity 1.>	*<The student clones the code from Student activity 1 and opens it on his/her machine.>*
We just want to show the constantly updated information on the screen. If we look at the data we receive from the API; we can notice that we also receive - ● Velocity ● Altitude We can create an information box below the map to display the information of the ISS. We can also display its current latitude and longitude there. *<The teacher guides the student to code the view, texts and styles.>*	*<The student codes the view, texts and styles.>*

```
<View style={styles.infoContainer}>
    <Text style={styles.infoText}>Latitude: {this.state.location.latitude}</Text>
    <Text style={styles.infoText}>Longitude: {this.state.location.longitude}</Text>
    <Text style={styles.infoText}>Altitude (KM): {this.state.location.altitude}</Text>
    <Text style={styles.infoText}>Velocity (KM/H): {this.state.location.velocity}</Text>
</View>
```

Adding styles for **View** as shown below:

```
infoContainer: {
    flex: 0.2,
    backgroundColor: 'white',
    marginTop: -10,
    borderTopLeftRadius: 30,
    borderTopRightRadius: 30,
    padding: 30
},
infoText: {
    fontSize: 15,
    color: "black",
    fontWeight: "bold"
}
```

<The teacher guides the student to run the code and check the output.> **Output**:	*<The student runs the code to check the output.>*

We now created the ISS Location screen where we are showing the real-time location of the ISS. Isn't that amazing!

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 5 mins

Teacher can show slideshow  **from slides 11 to 22** Refer to speaker notes and follow the instructions on each slide.	
Teacher Action	**Student Action**
Run the presentation from slide 11 to slide 22. **<u>Following are the wrap-up session deliverables:</u>** • **Explain the facts and trivias.** • **Next class challenge.** • **Project for the day.** • **Additional Activity**	***Guide the student to develop the project and share it with us.***
QnA Session - Click on in-class quiz	
Question	**Answer**
Which of the following props of the ImageBackground component is used to give the location of the image? A. uri B. source C. style D. image	**B**
Why do we use Axios? A. Using axios we create a map in the app B. Using axios we get the latitude and the longitude of a location C. Using axios we make a request on the given API to get the data.	**C**

D. Using axios we can catch any error in the code	
What does componentDidMount() do? A. it executes each time the state is changed B. it executes only once when the screen is opened first C. it doesn't executes until we call it D. it executes only once when we are navigating to a different screen	**B**
End the quiz panel	
FEEDBACK • **Appreciate the student for their efforts in the class.** • **Ask the student to make notes for the reflection journal along with the code they wrote in today's class.**	
Teacher Action	**Student Action**
Amazing work today! You get a "hats-off". Amazing work today! In the next class, we will work on creating a fully functional Meteor screen for the app.	*Make sure you have given at least 2 Hats Off during the class for:* 
Project Overview	*** This Project will take only 30 mins to complete. Motivate students to try**

	and finish it immediately after the class. **Stellar Stage-3** **Goal of the Project:** In Class 78, we have designed the ISS Location screen to show the location of the ISS (International Space Station) using the map of the world. You will be using the same concepts to add a Star Map screen into Stellar-App. *This is continuation project of Project-76 & 77, make sure to complete that one before attempting this one **Story:** Jeff is happy with your work on the Stellar App so far. He wants you to add a screen showing live locations of constellations. **Constellations** are easily recognizable patterns that help people orient themselves using the night sky. Here's a fun fact - There are 88 such "official" **constellations**. I am very excited to see your project solution and I know you will do really well. Bye Bye!	
Teacher ends slideshow 		

Teacher Clicks	✖ End Class

ADDITIONAL ACTIVITY	
Encourage the student to write reflection notes in their reflection journal using Markdown. Use these as guiding questions: ● What happened today? ○ Describe what happened. ○ The code I wrote. ● How did I feel after the class? ● What have I learned about programming and developing games? ● What aspects of the class helped me? What did I find difficult?	*The student uses the Markdown editor to write their reflections in a reflection journal.*

Activity	Activity Name	Links
Teacher Activity 1	Previous class code	https://github.com/pro-whitehatjr/C77_ISSTracker_TeacherReferenceCode
Teacher Activity 2	Reference code	https://github.com/pro-whitehatjr/C78_ISSTracker_TeacherReferenceCode
Teacher Activity 3	Map View Documentation	https://www.npmjs.com/package/react-native-maps
Teacher Activity 4	Teacher Aid	https://drive.google.com/file/d/1WA1BQff4dmgv5BInU3f_imk4vlpvAyMa/view?usp=sharing
Teacher Activity 5	Boilerplate Code	https://github.com/pro-whitehatjr/C7

		8_ISSTracker_TeacherActivity
Student Activity 1	Boilerplate code	https://github.com/pro-whitehatjr/C78_ISSTracker_StudentActivity
Student Activity 2	Map View Documentation	https://www.npmjs.com/package/react-native-maps
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C78-+withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/66fef9d2-aee1-470d-af76-f2c1df0807fa.pdf
Project Solution	Stellar Stage-3	https://github.com/pro-whitehatjr/Stellar-Stage-3