Topic	STACK NAVIGATION	
Class Description	**In this class, the students will be adding one more navigation to the project—Stack Navigation. Students will also implement text-to-speech.**	
Class	**C84**	
Class time	**45 mins**	
Goal	● Integrate Stack Navigation on the app. ● Create the story screen. ● Add text-to-speech.	
Resources Required	● Teacher Resources: ○ Visual Studio Code Editor ○ laptop with internet connectivity ○ earphones with mic ○ notebook and pen ● Student Resources: ○ Visual Studio Code Editor ○ laptop with internet connectivity ○ earphones with mic ○ notebook and pen	
Class structure	**Warm-Up**	**5 mins**

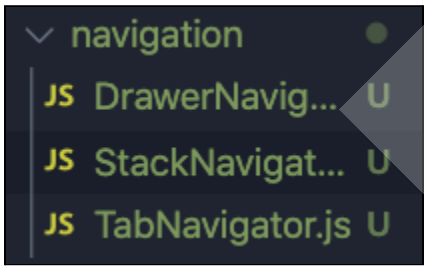
	Teacher-led Activity **Student-led Activity** **Wrap-Up**	**15 mins** **20 mins** **5 mins**

WARM-UP SESSION - 5 mins	
Teacher starts slideshow  **from slides 1 to 10** Refer to speaker notes and follow the instructions on each slide.	
Activity details	**Solution/Guidelines**
Hi, how have you been? Are you excited to learn something new? **Run the presentation from slide 1 to slide 3.** **The following are the warm-up session deliverables:** • Reconnect with previous class topics. • Warm-Up quiz session.	**ESR**: Varied Response. Click on the slide show tab and present the slides.
QnA Session	
Question	**Answer**

Which of the following fields have we added in our form? A. Image and Title B. Description and Story C. Moral D. All of the above	**D**
Which component is used to add a scrolling effect in the app? A. ViewScroll B. ScrollView C. scrollable D. fieldsContainer	**B**
Continue the warm-up session	
Activity details	**Solution/Guidelines**
Run the presentation from slide 4 to slide 10 to set the problem statement. **The following are the warm-up session deliverables:** ● Discuss about the flow of the app and create blueprints accordingly. ● Steps to write and run the code. ● Introduce the concepts of Teacher Led Activity.	Narrate the story by using hand gestures and voice modulation methods to bring in more student interest.

Teacher ends slideshow 		
TEACHER-LED ACTIVITY- 15 mins		
Teacher Initiates Screen Share		
CHALLENGE • **Integrate Stack Navigation.** • **Create a new story screen in our app.**		
Step 2: Teacher-led Activity (15 min)	*The teacher opens the code from the previous class or downloads the code from Teacher Activity 4* ***Note -*** *If the student and/or teacher is using the snack editor for these classes, please refer to the support document in Teacher Activity 5.* We have already implemented the Stack Navigation in the ISS Tracker app. In this app, we have implemented Drawer and Tab Navigation. However, we want to take the user to a Story	

	screen when they click on the story in the Feed Screen. For that, we will be using Stack Navigation.	
	Until now, we were using our Tab Navigation inside our Drawer Navigation, and we had our Drawer Navigation in App.js. To add Stack Navigation, we will change that. This time, we will use Tab Navigation inside a Stack Navigation and we will have Stack Navigation in the Drawer Navigation. It might seem confusing now, but you'll soon realize it's not. First let's install Stack Navigation with the following command - *yarn add @react-navigation/stack*	

	Now in the boilerplate code that we just cloned, we would notice that we already have a file called ***StoryScreen.js***. This is similar to the ***StoryCard*** component, but only getting displayed on the entire screen this time, instead of a small card. We will go over its code once, but first let's add Stack Navigation to our app so that this ***StoryScreen*** component can be accessed by clicking on the ***StoryCard***.	
	For that, let's create a new file - *StackNavigator.js* inside our *navigation* folder.	
 And the code inside this file would be -		

```
import React from "react";
import { createStackNavigator } from "@react-navigation/stack";
import TabNavigator from "./TabNavigator";
import StoryScreen from "../screens/StoryScreen";

const Stack = createStackNavigator();

const StackNavigator = () => {
  return (
    <Stack.Navigator initialRouteName="Home" screenOptions={{
      headerShown: false
    }}>
      <Stack.Screen name="Home" component={TabNavigator} />
      <Stack.Screen name="StoryScreen" component={StoryScreen} />
    </Stack.Navigator>
  );
};

export default StackNavigator;
```

Here, do note that we have **TabNavigator** as our default view. **If we add this Stack Navigator now in our DrawerNavigator, we will still see the TabNavigator by default, and have the ability to toggle to our StoryScreen.**

	Therefore, our *DrawerNavigator.js* becomes -	

```
import React from "react";
import { createDrawerNavigator } from "@react-navigation/drawer";
import StackNavigator from "./StackNavigator";
import Profile from "../screens/Profile";

const Drawer = createDrawerNavigator();

const DrawerNavigator = () => {
  return (
    <Drawer.Navigator>
      <Drawer.Screen name="Home" component={StackNavigator} />
      <Drawer.Screen name="Profile" component={Profile} />
    </Drawer.Navigator>
  );
};

export default DrawerNavigator;
```

where our Stack Navigator is the first view.

	Now, let's give it a thought. We want to access the ***StoryScreen*** from our	

Feed screen, but our cards are in ***StoryCard.js***.

Now we will have access to the navigation props in the Feed Screen, but we want the navigation to happen in the ***StoryCard***. Therefore, we will have to pass it to the component. In our ***Feed.js***, we can do that -

```
renderItem = ({ item: story }) => {
    return <StoryCard story={story} navigation={this.props.navigation} />
};
```

This way, we are passing our navigation props as **navigation** to our **<StoryCard>** component.

Now to use this **navigation** in our **<StoryCard>** component, we will have to use a **<TouchableOpacity>** component to wrap our card contents inside it and perform the navigation on the **onPress** event of our **<TouchableOpacity>** component.

Our ***StoryCard.js*** will therefore be -

```
return (
            <TouchableOpacity style={styles.container} onPress={() =>
this.props.navigation.navigate("StoryScreen", story = this.props.story)}>
                <SafeAreaView style={styles.droidSafeArea} />
                <View style={styles.cardContainer}>
                  <View style={styles.storyImage}>
                    <Image source={require("../assets/story_image_1.png")} style={{
resizeMode: 'contain', width: Dimensions.get('window').width - 60, height: 250,
borderRadius: 10 }}></Image>
                  </View>
                  <View style={styles.titleContainer}>
                    <View style={styles.titleTextContainer}>
                      <View style={styles.storyTitle}>
                        <Text
style={styles.storyTitleText}>{this.props.story.title}</Text>
                      </View>
                      <View style={styles.storyAuthor}>
                        <Text
style={styles.storyAuthorText}>{this.props.story.author}</Text>
                      </View>
                    </View>
                  </View>
                  <View style={styles.descriptionContainer}>
                    <Text style={styles.descriptionText}>
                      {this.props.story.description}
                    </Text>
                  </View>
```

```
            <View style={styles.actionContainer}>
              <View style={styles.likeButton}>
                <View style={styles.likeIcon}>
                  <Ionicons name={"heart"} size={30} color={"white"} style={{
width: 30, marginLeft: 20, marginTop: 5 }} />
                </View>
                <View>
                  <Text style={styles.likeText}>12k</Text>
                </View>
              </View>
            </View>
          </View>
        </TouchableOpacity>
      )
```

Here, instead of a view, we are using a **<TouchableOpacity>** for our container.

In our **onPress** of the ***TouchableOpacity***, we are calling **this.props.navigation.navigate()** to navigate to ***StoryScreen***, and we are passing our ***this.props.story*** to it as a story prop.

	Now, let's run the app and see if it works. (On clicking on any of the cards in the Feed screen, it should navigate to the Story Screen.)	

	Awesome! Now let's go over the code of this screen once. It is quite similar to what we had in the **StoryCard** component.	

```
import React, { Component } from "react";
import {
    View,
    Text,
    StyleSheet,
    SafeAreaView,
    Platform,
    StatusBar,
    Image,
    ScrollView,
    Dimensions
} from "react-native";
import Ionicons from "react-native-vector-icons/Ionicons";
import { RFValue } from "react-native-responsive-fontsize";

import AppLoading from "expo-app-loading";
import * as Font from "expo-font";
```

We first have the import statements for this screen. Most of the imports are similar to the ***StoryCard*** component, but we can notice that we have also imported ***Ionicons*** this time.

Next -

```
let customFonts = {
    "Bubblegum-Sans": require("../assets/fonts/BubblegumSans-Regular.ttf")
};

export default class StoryScreen extends Component {
    constructor(props) {
        super(props);
        this.state = {
            fontsLoaded: false,
            speakerColor: "gray",
            speakerIcon: "volume-high-outline"
        };
    }

    async _loadFontsAsync() {
        await Font.loadAsync(customFonts);
        this.setState({ fontsLoaded: true });
    }

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

We can see that we have defined the fonts, like in the previous screen.

We have then created our class component ***StoryScreen*** and inside it, we have added a constructor, a function to load our fonts and our componentDidMount() function. These

are the things that we have again, done previously in other screens.

One thing that's new is that we have 2 new states—**speakerColor** set to **gray** and **speakerIcon** set to **'volume-high-outline'**.

This is for the icon of the speaker that we have in our output.

Finally we have our render method, in which again, most of the things are similar to previous screens and the ***StoryCard*** component except for one thing -

```
<View style={styles.iconContainer}>
    <Ionicons
        name={this.state.speakerIcon}
        size={RFValue(30)}
        color={this.state.speakerColor}
        style={{ margin: RFValue(15) }}
    />
</View>
```

This code, that we have added for our speaker icon. After this, we have the styling too, added in the boilerplate. It is again very similar to the ***StoryCard*** component and other screens.

If we look at its output -

We will notice that there's a speaker icon next to the title of the story. This speaker icon is for our text-to-speech functionality that you'll be building.

	The idea is that as soon as someone clicks on the speaker icon, we want to change this icon's color to the one we have in the bottom tab navigator and we want to change the icon as well, so that it's just not an outline of a speaker. We will use *expo-speech* to implement our text-to-speech in this.	
Teacher Stops Screen Share		
	Now it's your turn. Please share your screen with me.	
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.** • **Teacher gets into fullscreen.**		
Teacher starts slideshow  **from slide 11 to slide 13**		
<u>ACTIVITY</u> • **Implement the text-to-speech functionality in the story screen.**		

Step 3: Student-Led Activity (20 mins)	Please refer to *Student Activity 1* to clone the Boilerplate code containing everything we've done in today's class.	*Student refers to Student Activity 1 to clone the repository.*
	Let's start by installing *expo-speech*. *expo install expo-speech* You can refer to *Student Activity 2 to read its documentation.* *Teacher refers to Teacher Activity 1.*	*Student installs the dependencies and refers to Student Activity 2.*
	Okay, so first, we need to wrap our icon within a ***TouchableOpacity*** to be able to add an **onPress** event to it. Now, for the text-to-speech, we want it to relay the: 1. title 2. name of the author 3. story 4. moral of the story	*Student writes the code.*

	For this, we can call an **initiateTTS()** function on the **onPress** event and pass these values to it.	

```
<TouchableOpacity
  onPress={() =>
    this.initiateTTS(
      this.props.route.params.story.title,
      this.props.route.params.story.author,
      this.props.route.params.story.story,
      this.props.route.params.story.moral
    )
  }
>
  <Ionicons
    name={this.state.speakerIcon}
    size={RFValue(30)}
    color={this.state.speakerColor}
    style={{ margin: RFValue(15) }}
  />
</TouchableOpacity>
```

Don’t forget to import the **TouchableOpacity** component.

	Now, we need to create the **initiateTTS()** function -	

```
async initiateTTS(title, author, story, moral) {
const current_color = this.state.speakerColor;
this.setState({
  speakerColor: current_color === "gray" ? "#ee8249" : "gray"
});   if (current_color === "gray") {

  Speech.speak(`${title} by ${author}`);
  Speech.speak(story);
  Speech.speak("The moral of the story is!");
  Speech.speak(moral);
} else {
  Speech.stop();
}
}
```

Here, since we want to change the color of our speaker icon, we are taking the current state of **speakerColor** in a constant **current_color** and based on its value, we are setting the state. This will immediately re-render the screen with a differently colored speaker icon.

Next, we are checking if the **current_color** is **gray**. Now, we are doing this because if the current color is gray, that means that the user has enabled text-to-speech. Note that the **current_color** here is the color of the icon before the user pressed it.

In this case, we are using the **Speech.speak()** function with the text inside it that we want it to speak.

If, however, the color wasn't gray, that means that the user wants to disable it and we are calling **Speech.stop()** so that if it still has something to speak, it can stop immediately.

We will also have to import Speech too -

```
import * as Speech from 'expo-speech';
```

	We successfully implemented text-to-speech.	

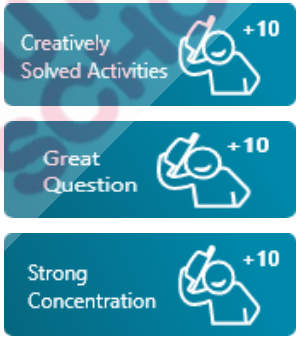
Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 5 Mins

Teacher starts slideshow  **from slide 13 to slide 24**

Activity details	Solution/Guidelines
Run the presentation from slide 15 to slide 24 **Following are the warm up session deliverables:** ● **Explain the facts and trivias** ● **Next class challenge** ● **Project for the day**	Guide the student to develop the project and share with us.

• **Additional Activity**	
Quiz time - Click on in-class quiz	
Question	**Answer**
_____ provides a way for your app to transition between screens, where each screen is placed on top of a stack. A. TabNavigator B. StackNavigator C. SwitchNavigator D. AppDrawerNavigator	**B**
Which of the following helps us to navigate from the current screen to a new screen (StoryScreen)? A. props.navigation.navigate B. this.props.navigation C. this.props.navigation.navigate D. this.props.navigate	**C**
To convert text-to-speech, which library is used? A. expo-speech B. Speech.speak() C. Speech.stop() D. text-speech	**A**

End the quiz panel		
FEEDBACK • **Appreciate the student for their attentiveness in class.** • **Get them to play around with different ideas.**		
Step 4: **Wrap-Up** **(5 min)**	Let's quickly wrap up today's class.	
	Amazing work today! You get a “hats-off”. In the next class, we will be implementing Google Authentication and integrating the app with Firebase.	*Make sure you have given at least 2 Hats Off during the class for:* 
Project Overview **Spectagram Stage - 4**		*The students engage with the teacher over the project.*

Goal of the Project:

In Class 84, we've integrated the Stack Navigation to the app. We also created the story screen and added text-to-speech.

In this project, you will practice the concepts learned in the class to create a Post screen and integrate the same in Stack Navigator.

*This is a continuation project of 81, 82 & 83; please make sure to finish that before attempting this one.

Story:

Jenny is a photographer. She wants to share pictures taken by her with others. At the same time, she wants to create a space for others to share their talent too. She has decided to create a social media app. She has asked for your help to create an app.

Guide Jenny to integrate Stack Navigation to the app and create a Posts screen.

Teacher ends slideshow 

Teacher Clicks 		
ADDITIONAL ACTIVITY		
Additional Activities	*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	Expo Speech Documentation	https://docs.expo.io/versions/latest/sdk/speech/
Teacher Activity 2	Reference Code	https://github.com/pro-whitehatjr/ST-84-Solution
Teacher Activity 3	Teacher Aid	https://drive.google.com/file/d/1WA1BQff4dmgv5BInU3f_imk4vlpvAyMa/view?usp=sharing
Teacher Activity 4	Teacher Boilerplate Code	https://github.com/pro-whitehatjr/Story-Telling-App-84-TB
Teacher Activity 5	Snack Support Document	https://docs.google.com/document/d/11vq49uJQCfdxaUUzOoY7A65aau0kZqNMFhObZH-e71Y/edit?usp=sharing
Student Activity 1	Boilerplate Code	https://github.com/pro-whitehatjr/ST-84-Boilerplate
Student Activity 2	Expo Speech Documentation	https://docs.expo.io/versions/latest/sdk/speech/
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/PRO_V3_C84_LITE_withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/b103b078-bff5-4110-a86c-f8e69bea7735.pdf