Topic	MATERIAL UI AND FEED SCREEN
Class Description	**In this class, the students will be exploring the material UI, with the help of which, we will be styling the bottom tab navigation and also, the feed screen.**
Class	**C82**
Class time	**45 mins**
Goal	● Styling the bottom tab navigation. ● Styling the feed screen.

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

WARM-UP SESSION - 5 mins

CHALLENGE
● **Design, style and create the feed screen.**
● **Use material UI design components in bottomTabNavigation.**

Teacher starts slideshow **from slides 1 to 11** Refer to speaker notes and follow the instructions on each slide.

Activity details	Solution/Guidelines
Hey <student name>. How are you? It's great to see you! Are you excited to learn something new today? **Run the presentation from slide 1 to slide 4.** **Following are the warm up session deliverables:** ● Connecting students to the previous class.	**ESR**: Hi, thanks, yes I am excited about it! Click on the slide show tab and present the slides.

QnA Session	
Question	**Answer**
Which component did we use to implement the drawer functionality? A. DrawerNavigation B. TabNavigation C. DrawerContainer D. DrawerNavigator	**D**
Which of the following screens are created so far in our storytelling app? A. Story Feed screen to display a list of all the stories. B. A story screen where the user can read a particular story. C. A profile screen where the user can set the theme of their choice. D. All of the above.	**D**

Continue the warm up session	
Activity details	**Solution/Guidelines**
Run the presentation from slide 5 to slide 11 to set the problem statement. **Following are the warm up session deliverables:** ● Explain the designing, styling and feed screen.	Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.

● TEACHER-LED ACTIVITY - 15 mins
Teacher Initiates Screen Share

CHALLENGE
● **Design, style and create the feed screen.** ● **Use material UI design components in bottomTabNavigation.**

Step 2: Teacher-led Activity (15 min)	Before we proceed with styling our screens, let's first take a look at what we are looking forward to building -	*Student observes.*

Teachers make sure to use Storytelling as one word and no space in between.

	Looks amazing, right?	ESR: Yes.
	Now in this, the first thing that you will notice is that our styling for bottom tab navigation is different.	
	Remember that we are going to build 2 themes for our app—light and dark.	

	The output we are currently seeing is for the dark theme, so we'll build the app with the dark theme first. We have a boilerplate code in which the styling for the bottom tab navigation is already done. Let's clone that! *Teacher refers to Teacher Activity 6 and clones the boilerplate code* Now to begin, let's first install the following dependencies - *yarn add @react-navigation/material-bottom-tabs react-native-paper* *yarn add react-native-responsive-fontsize* **Note -** *If the student and/or teacher is using the snack editor for these classes, please refer to the support document in Teacher Activity 7.* Can you tell me which file had the bottom tab navigation now?	**ESR:** The *TabNavigator.js* in the *navigation* folder.
	Great! Now previously, we were using **createBottomTabNavigator()** but here, we are using **createMaterialBottomTabNavigator(**	

	) which comes with exciting styling options. We are also using the stylesheet from react native. We can see that these are imported in our code as well -	

```javascript
JS TabNavigator.js ✕

navigation > JS TabNavigator.js > ⊘ BottomTabNavigator
1  import React from "react";
2  import { StyleSheet } from "react-native";
3  import { createMaterialBottomTabNavigator } from "@react-navigation/material-bottom-tabs";
4  import Ionicons from "react-native-vector-icons/Ionicons";
5  import { RFValue } from "react-native-responsive-fontsize";
6
7  import Feed from "../screens/Feed";
8  import CreateStory from "../screens/CreateStory";
9  const Tab = createMaterialBottomTabNavigator();
```

	Next, we can see some attributes added to our **<Tab.Navigator>** component. Material Bottom Tab Navigator uses a variety of different attributes to style itself. You can find out more about it from <_Student Activity 1_>. _(Teacher refers to the <Teacher Activity 1>)._ We have used some of them like -	

```
const BottomTabNavigator = () => {
  return (
    <Tab.Navigator
      labeled={false}
      barStyle={styles.bottomTabStyle}
      screenOptions={({ route }) => ({
        tabBarIcon: ({ focused, color, size }) => {
          let iconName;
          if (route.name === "Feed") {
            iconName = focused ? "home" : "home-outline";
          } else if (route.name === "Create Story") {
            iconName = focused ? "add-circle" : "add-circle-outline";
          }
          return (
            <Ionicons
              name={iconName}
              size={RFValue(25)}
              color={color}
              style={styles.icons}
            />
          );
        }
      })}
      activeColor={"#ee8249"}
      inactiveColor={"gray"}
    >
      <Tab.Screen name="Feed" component={Feed} />
      <Tab.Screen name="Create Story" component={CreateStory} />
    </Tab.Navigator>
  );
};
```

Teacher explains the code to the student

Here, first we have an attribute **labeled**, which is set to **false**. This means that there will be no labels and only icons will be displayed.

Next, we have **barStyle**, which can just be the styling of our bottom tab navigation. We haven't added the styles yet, but we will after walking through this.

Next, we have **screenOptions**, just like before. Do note that we have changed the icons this time.

Finally, we have our **activeColor** and **inactiveColor** for icons. These are the colors we want to have for active and inactive screens on our bottom tab navigation.

	Now, let's add the styling for this -	

```
const styles = StyleSheet.create({
  bottomTabStyle: {
    backgroundColor: "#2f345d",
    height: "8%",
    borderTopLeftRadius: 30,
    borderTopRightRadius: 30,
    overflow: "hidden",
    position: "absolute"
  },
  icons: {
    width: RFValue(30),
    height: RFValue(30)
  }
});
```

Teacher explains the code to the student

If we look at the styling here, we can see that we are using a **backgroundColor** to set the bluish color of our bottom tab navigator.

We are defining the **height** of our navigator, **8%** in this case.

We are then defining the **borderTopLeftRadius and borderTopRightRadius** to give curved edges at the top on both left and right sides. We don't want our bottom tab to have round edges or it would look bad.

We then set **overflow** as **hidden**. This means that we do not want this bottom tab to be scrollable. This means that if the contents of this bottom tab navigator do not fit in the screen, keep the **overflowed** content that's spilling out of the screen **hidden**.

Finally, we have **position** as **absolute**, so it stays at the bottom.

We also have defined **width** and **height** for our icons in our bottom tab navigation, but we are using a certain *RFValue()* here that we imported from **react-native-responsive-fontsize.**

RFValue() is used to make things responsive. There might be cases where we provide padding of 10 that might look good on small screens but would be of no use in bigger screens. To avoid such things, it's a good practice to use all your numbers while styling with *RFValue()*.

With this code added, our app looks something like this -
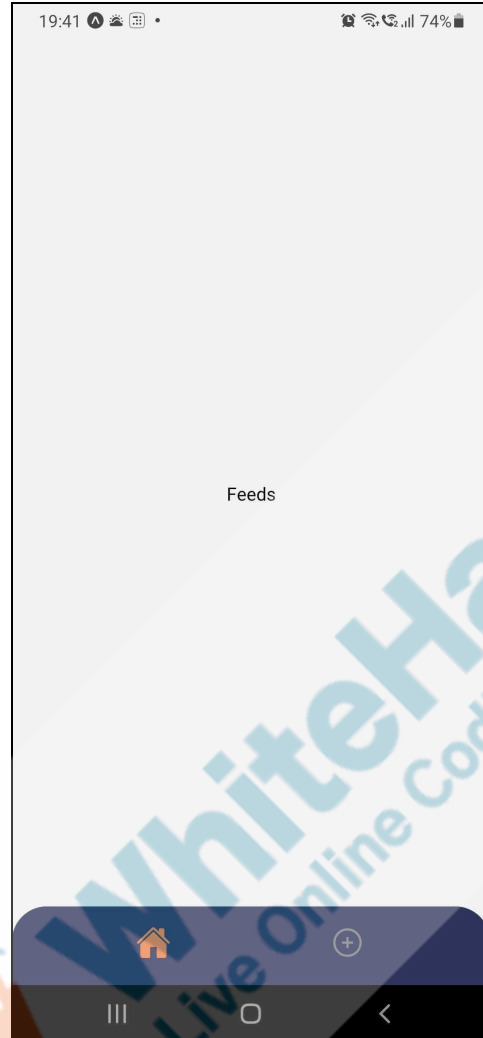
That looks good!

You are now equipped to create awesome bottom tab navigation for your apps.

Let's move forward and build our feed screen now.

In our app, we'll use a lot of assets. It would become hard for us to keep adding assets as we proceed;

	therefore we have an assets folder ready for you which contains all the assets for the app to the end. Don't worry about downloading them. They will already be there in your boilerplate code. *<Teacher refers to Teacher Activity 2 and get all the assets.>*	*<Student observes.>*
	Once done, the assets folder in our project should look something like this -	
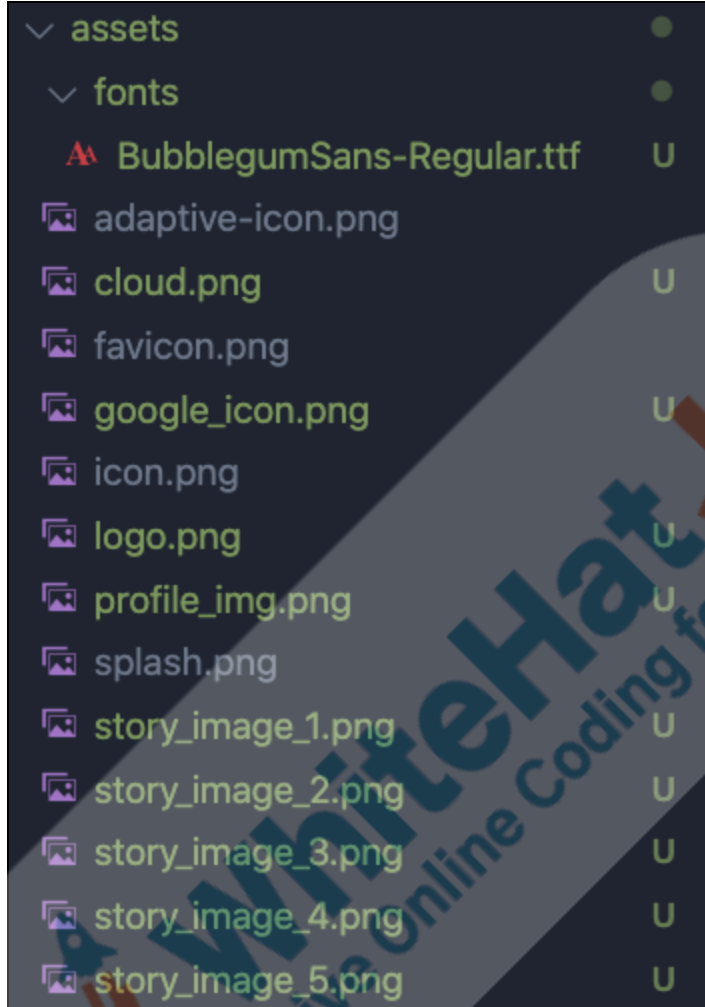
From the app screenshot that we saw above, we know we were not using the default fonts but rather different fonts.

For that, we specifically have a folder named **fonts** in our assets folder which contains the **BubblegumSans-Regular.ttf** file.

This file with the .ttf extension is a specific kind of file which stores the

	data about a particular font, but how do we use it?	**ESR:** Varied.
	To use this, we will have to install the following dependencies - *expo install expo-font* *expo install expo-splash-screen* We installed *expo-splash-screen.* As we are sourcing fonts from external resources, it might take time to render. First we want to load our fonts and while they are loading, we want to display "Loading…" on the screen and once they are loaded we want to display the Feed text in the imported fonts. To do this we'll write an if-else condition. Now, since we haven't built any functionalities of the app yet, it's obvious that we will not have any stories by default to show here. For that, we will create a temporary JSON file containing an array of objects with stored data. We can use this to create our UI. *<Teacher Refers to the Teacher Activity 3.>* *<Teacher creates a new file temp.json in the screens folder and*	*<Student observes.>*

	copies the contents from the file available at Teacher Activity 3.>	

```json
[
  {
    "title": "The Boy Who Cried Wolf",
    "author": "Apoorv Goyal",
    "created_on": "25th January, 2021",
    "description": "A story of a boy who lied and lost the trust of the people close to him.",
    "story": "Once upon a time, there lived a shepherd boy who was bored watching his flock of sheep on the hill. To amuse himself, he
    "moral": "Lying breaks trust. Nobody trusts a liar, even when s/he is telling the truth."
  },
  {
    "title": "The Midas Touch",
    "author": "Saurabh Aswani",
    "created_on": "14th February, 2021",
    "description": "A story of a king and his greed.",
    "story": "In ancient Greek, there was a king named Midas. He had a lot of gold and everything he needed. He also had a beautiful d
    "moral": "Do not get greedy. Be happy and content with what you have."
  }
]
```

	Alright, now we have our temporary data. Let's start working on the UI. The first thing that we want to do is to load our fonts in our app.	

```jsx
import React, { Component } from 'react';
import { Text, View } from 'react-native';

import * as Font from 'expo-font';
import * as SplashScreen from 'expo-splash-screen';

SplashScreen.preventAutoHideAsync();

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

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

```
        }

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

    componentDidMount() {
        this._loadFontsAsync();
    }

    render() {
        if (this.state.fontsLoaded) {
            SplashScreen.hideAsync();
            return (
                <View
                    style={{
                        flex: 1,
                        justifyContent: "center",
                        alignItems: "center"
                    }}>
                    <Text>Feeds</Text>
                </View>
            )
        }
    }
}
```

Teacher explains the code to the student

In our previously existing code, we imported SplashScreen and Font from the dependencies we installed. We have also called **SplashScreen.preventAutoHideAsync()** which makes the splash screen (It can be configured in app.json) remain visible until hideAsync is called.

[The image in the splash screen can be changed by **app.json** and changing the image link there.

```
{} app.json > {} expo
 1    {
 2      "expo": {
 3        "name": "ST-82-Boilerplate",
 4        "slug": "ST-82-Boilerplate",
 5        "version": "1.0.0",
 6        "orientation": "portrait",
 7        "icon": "./assets/icon.png",
 8        "userInterfaceStyle": "light",
 9        "splash": {
10          "image": "./assets/splash.png",
11          "resizeMode": "contain",
12          "backgroundColor": "#ffffff"
13        },
14        "updates": {
```
]

We then created a variable **customFont** and defined **"Bubblegum-Sans"** as the key and the path to the **.ttf** file as the value.

Inside our class component, we created a constructor with **fontsLoaded** state is set to **false**, because initially our fonts are not loaded.

We then created an async function **_loadFontAsync()** and called the **Font.loadAsync()** function to load the fonts and then set the state to **true**.

We are calling this function in our **componentDidMount()** method and in the **render()** method, we are checking if the fonts are loaded or not. If loaded, we are hiding **the splash screen** component.

	Now let's discuss the social media apps. Usually, the feed screen has cards for the posts or statuses which users add, so we want to display our stories in the form of cards as well. Can you guess how we can do this?	**ESR:** By using **<FlatList>**

| | Awesome! Let's begin with building the UI then.

Our return statement in the render function would look like - | |

```
return (
        <View style={styles.container}>
            <SafeAreaView style={styles.droidSafeArea} />
            <View style={styles.appTitle}>
                <View style={styles.appIcon}>
                    <Image source={require("../assets/logo.png")} style={{ width: 60,
height: 60, resizeMode: 'contain', marginLeft: 10 }}></Image>
                </View>
                <View style={styles.appTitleTextContainer}>
                    <Text style={styles.appTitleText}>
                        Storytelling App
                    </Text>
                </View>
            </View>
            <View style={styles.cardContainer}>
                <FlatList
                    keyExtractor={this.keyExtractor}
                    data={stories}
                    renderItem={this.renderItem}
                />
            </View>
        </View>
    )
```

Here, we first define a parent **<View>** with a style **container**.

Inside it, we again have the **<SafeAreaView/>** component.

Next, we want to display the logo and the app title at the top, so we have a **<View>** with style **appTitle**. Inside this, we have 2 views, one for our logo and the other for the title.

Next, below this top view, we have another view containing our **<FlatList>**. Don't forget to make the imports of things we are using here -

```
import {
  View,
  Text,
  StyleSheet,
  SafeAreaView,
  Platform,
  StatusBar,
  Image
} from "react-native";
import { FlatList } from "react-native-gesture-handler";
```

Our styling would look something like this -

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#15193c"
  },
  droidSafeArea: {
    marginTop: Platform.OS === "android" ? StatusBar.currentHeight : 0
  },
  appTitle: {
    flex: 0.07,
    flexDirection: "row",
    flexWrap: "wrap",
    padding: 5,
  },
  appIcon: {
    flex: 0.3
  },
  appTitleTextContainer: {
    justifyContent: "center",
    alignItems: "center"
  },
```

```
appTitleText: {
    color: "white",
    fontSize: 28,
    fontFamily: "Bubblegum-Sans",
    paddingLeft: 20
},
cardContainer: {
    flex: 0.85
  }
});
```

Note - Styles are expected to be copied and pasted from the document / reference code, with just the explanation of the styling given to the student.

Here, we are giving a **flex** of **1** and a **backgroundColor** to our main container.

[In case the student is unable to recall flex, Flex is designed to provide a consistent layout on different screen sizes.]

We then give a flex of **0.07** to our **appTitle** view. This means that this view should only have a height of 7% of the parent container. We have also specified **flexDirection** to be a row, which means that the views inside it should be placed in the same row. We **wrap** the views with the **flexWrap** style.

For our **appIcon**, which is the view containing our logo, we give a **flex** of 0.3. This means that it will take 30% of its parent container, which is 7% of the entire container. Also do note that since we mentioned **flexDirection** to **row** in its parent container, this container takes 30% in terms of width and not height.

In the **appTitleText**, we are mentioning the **fontFamily** as **Bubblegum-Sans**. Since this font is already loaded, it will use these fonts now for the title.

	You may also notice that for the **cardContainer,** we are using a **flex** of 0.85**,** which means 85%.	
		ESR:

	Since we've only used 7% for our title container, can you tell me why are we not using a flex of **0.93** here? That's right! Since our bottom tab navigator is using up to 8% of the height, then we are only left with 92% of height on the screen to display content. That's why after we give our title container 7% of the height, we are only left with 85%.	Because the bottom tab navigator has a height of 8%.
	Awesome! Now for our **\<FlatList\>**, we have again used: 1. keyExtractor 2. data 3. renderItem Let's create these.	

For our data, we have already created a file having stories. If we simply import it and store it in a variable called **stories** below our import statements, it will sort the **data** attribute for us -

```
let stories = require("./temp_stories.json");
```

Now for the renderItem and keyExtractor, we can create functions like -

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


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

Here, do note that for renderItem, instead of writing all the Components here, we have used a component **<StoryCard/>** in which we are passing the **story** as the props, and we haven't created this component yet.

We will simply import this at the top -

```
import StoryCard from "./StoryCard";
```

	Now, all we need to do is to create a file called **StoryCard.js** in our **screens** folder and make it return the components in its render function. That's what you will do.	

Teacher Stops Screen Share

	Now it's your turn. Please share your screen with me.	

STUDENT-LED ACTIVITY - 20 mins

Teacher starts slideshow **from slide 12 to slide 13**

- **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.**

ACTIVITY
- **Complete the StoryCard.js code so that cards can be displayed for stories in the feed screen.**

Step 3: Student-Led Activity (15 mins)	*Refer to the <Student Activity 4> and clone the code boilerplate code.*	*Student refers to <Student Activity 4> and clones the boilerplate code.*

	You will see a file called **StoryCard.js** in the boilerplate code. This is the file you will be working on. This code already has the fonts loaded and all the essential modules imported for you. You only have to work on the render function and add the styling. You can use the variable **this.props.story** to access the story object. The UI for the card that needs to be designed looks like the following -	

	Can you build this?	**ESR:** Yes!
	Let's start building it then!	
	<Teacher helps the student in writing the code in the render function.>	*<Student writes the code in the render function.>*

```
return (
        <View style={styles.container}>
           <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>
          </View>
      )
```

```
const styles = StyleSheet.create({
```

```
droidSafeArea: {
    marginTop: Platform.OS === "android" ? StatusBar.currentHeight : 0
},
cardContainer: {
    marginTop: -20,
    marginBottom: 20,
    marginLeft: 20,
    marginRight: 20,
    backgroundColor: "#2f345d",
    borderRadius: 20,
    height: undefined,
    padding: 10
},
titleContainer: {
    flexDirection: "row"
},
titleTextContainer: {
    flex: 1
},
storyTitleText: {
    fontFamily: "Bubblegum-Sans",
    fontSize: 25,
    color: "white"
},
storyAuthorText: {
    fontFamily: "Bubblegum-Sans",
    fontSize: 18,
    color: "white"
},
descriptionContainer: {
    marginTop: 5
},
descriptionText: {
    fontFamily: "Bubblegum-Sans",
    fontSize: 13,
    color: "white"
},
```

```
actionContainer: {
    marginTop: 10,
    justifyContent: "center",
    alignItems: "center"
},
likeButton: {
    backgroundColor: "#eb3948",
    borderRadius: 30,
    width: 160,
    height: 40,
    flexDirection: "row"
},
likeText: {
    color: "white",
    fontFamily: "Bubblegum-Sans",
    fontSize: 25,
    marginLeft: 25,
    marginTop: 6
  }
});
```

Note - Styles are expected to be copied and pasted from the document / reference code, with just the explanation of the styling given to the student.

	Teacher should help debug the errors, if any, on the student's side. *Teacher can prompt the student to talk about differences in output with and without fonts and images and how important UI is in an app.* You can stop sharing the screen. Let's have a quick revision.	
Teacher Guides Student to Stop Screen Share		

Teacher starts slideshow from slide 14 to slide 24	
Activity details	**Solution/Guidelines**
Run the presentation from slide 14 to slide 24 **Following are the warm up session deliverables:** ● **Explain the facts and trivia** ● **Next class challenge** ● **Project for the day** ● **Additional Activity**	Guide the student to develop the project and share with us.
Quiz time - Click on in-class quiz	
Question	**Answer**
Which of the following functions do we use to style the bottom tab navigator? A. createMaterialBottomTab() B. createMaterialBottomNavigator() C. createMaterialBottomTabNavigator() D. materialBottomTabNavigator()	**C**
Curved edges can be added at the top on both the left and right side by which of the following? A. borderTopLeftRadius and borderTopRightRadius B. overflow as hidden and position as absolute C. activeColor and inactiveColor D. TopLeftRadius and TopRightRadius	**A**
What does the following snippet of code do? ```let stories = require("./temp_stories.json");```	**B**

A. We are storing the stories in a JSON variable.
B. We are importing the file and storing it in a variable called **stories**.
C. We are creating temp_stories.json.
D. We are recovering the deleted files from the JSON variable.

FEEDBACK
- **Appreciate the student for their attentiveness in the class.**
- **Get them to play around with different ideas.**

	Amazing work today! You get a "hats-off". Alright. In the next class we will work on the CreateStory Screen in the next class and add functionality to submit the stories. See you in the next class.	*Make sure you have given at least 2 Hats Off during the class for:* Creatively Solved Activities +10 Great Question +10 Strong Concentration +10

Project Overview

Spectagram Stage 2

Goal of the Project:

In class 82, we used the material UI, with the help of which we will be styling the bottom tab navigation and also the feed screen.

In this project, you will practice the concepts learned in the class to style the bottom tab & design the feed screen for the Spectagram App.

*This is a continuation project of 81; 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 make the basic structure created in the last project more attractive by adding images and icons in Spectagram App.

Bye Bye!

Teacher Clicks	✖ End Class

| 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? | *The student uses the markdown editor to write their reflections in a reflection journal.* |

- What have I learned about programming and developing games?
- What aspects of the class helped me? What did I find difficult?

Activity	Activity Name	Links
Teacher Activity 1	Material Bottom Navigation	https://reactnavigation.org/docs/material-bottom-tab-navigator/
Teacher Activity 2	Assets	https://github.com/pro-whitehatjr/ST-82-Solution/tree/master/assets
Teacher Activity 3	Temporary Story Data	https://s3-whjr-curriculum-uploads.whjr.online/c268b21c-8199-436f-91d6-0a61252770db.json
Teacher Activity 4	Reference Code	https://github.com/pro-whitehatjr/ST-82-Solution
Teacher Activity 5	Teacher Aid	https://drive.google.com/file/d/1WA1BQff4dmgv5BInU3f_imk4vlpvAyMa/view?usp=sharing
Teacher Activity 6	Teacher Boilerplate Code	https://github.com/pro-whitehatjr/Story-Telling-App-82-TB
Teacher Activity 7	Snack Support Document	https://docs.google.com/document/d/11vq49uJQCfdxaUUzOoY7A65aau0kZqNMFhObZH-e71Y/edit?usp=sharing
Student Activity 1	Material Bottom Navigation	https://reactnavigation.org/docs/material-bottom-tab-navigator/
Student Activity 2	Assets	https://github.com/pro-whitehatjr/ST-82-Boilerplate/tree/master/assets

Student Activity 3	Temporary Story Data	https://s3-whjr-curriculum-uploads.whjr.online/c268b21c-8199-436f-91d6-0a61252770db.json
Student Activity 4	Boilerplate Code	https://github.com/pro-whitehatjr/ST-82-Boilerplate
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/PRO_V3_C82_LITE_withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/5786a463-0834-4213-a563-8bbd8ab69ff6.pdf