



Topic	CUSTOM STYLED DRAWER NAVIGATION	
Class Description	In this class, we will custom style our Drawer Navigation, and we will also integrate our like functionality on stories.	
Class	C89	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> • Custom styling our Drawer Navigation. • Like functionality for Stories. 	
Resources Required	<ul style="list-style-type: none"> • Teacher Resources <ul style="list-style-type: none"> ○ Visual Studio Code Editor ○ laptop with internet connectivity ○ earphones with mic ○ notebook and pen • Student Resources <ul style="list-style-type: none"> ○ Visual Studio Code 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 20 mins 5 mins
WARM-UP SESSION - 5 mins		
<p style="text-align: center;"><u>CONTEXT</u></p> <ul style="list-style-type: none"> • Discuss the journey of the App so far and set the context of customizing the Drawer Navigator. 		
<p style="text-align: center;"> Teacher starts slideshow  from slides 1 to 9 </p>		

Refer to speaker notes and follow the instructions on each slide.	
Activity details	Solution/Guidelines
<p><i>Hi, it's so good to see you after the trial class! How have you been? So this is going to be the first class of the first module. Are you excited to learn something new?</i></p> <p>Run the presentation from slide 1 to slide 4.</p> <p>The following are the warm-up session deliverables:</p> <ul style="list-style-type: none"> • Revision of previous class. • Warm-Up Quiz Session. 	<p>ESR: Varied Response.</p> <p>Click on the slide show tab and present the slides.</p>
QnA Session	
Question	Answer
<p>To accommodate the new change since the story is an object which has a key as the unique id of the story and value as the story's data, what do we need to change?</p> <p>A. No change is needed.</p> <p>B. We first change the constructor in our StoryCard.js to store the keys and values together.</p> <p>C. We change the constructor in our StoryCard.js to store the keys and values separately.</p> <p>D. None of the above.</p>	<p>C</p>
<p>What does the following piece of code do?</p> <pre><Image source={images[story.preview_image]}</pre> <p>A. Changes the source of the <Image> component that displays the image of the story accordingly.</p> <p>B. Changes the source of the <Image> component that displays a constant image.</p> <p>C. Adds a theme to the background.</p> <p>D. Displays an image based on the preferred theme.</p>	<p>A</p>

Continue the warm-up session		
Activity details		Solution/Guidelines
<p>Run the presentation from slide 5 to slide 9 to set the problem statement.</p> <p>The following are the warm-up session deliverables:</p> <ul style="list-style-type: none"> Discuss the Journey of the App so far and set the context of customizing Drawer Navigator. 		<p>Narrate the slides by using hand gestures and voice modulation methods to bring in more student interest.</p>
<p>Teacher ends slideshow</p>		
TEACHER-LED ACTIVITY 15 mins		
Teacher Initiates Screen Share		
<p><u>CHALLENGE</u></p> <ul style="list-style-type: none"> Custom styling for Drawer Navigation. Like functionality for the app. 		
<p>Step 2: Teacher-led Activity (15 min)</p>	<p>In the past few classes we worked on the UI of all the screens on our app, however, our side drawer is still plain white and does not match with the App.</p> <p><i>Teacher starts to code from the previous class; The code is provided as Teacher Activity 1.</i></p> <p>Let's start with the custom design for the Drawer Navigation.</p>	

To do this, we will have to create a custom UI and then use it in the Drawer Navigation.

Let's see how -

Note: Make changes in **DrawerNavigator.js** inside the **navigations** folder.

We will first import **firebase** and convert **const DrawerNavigator** into a **class** component from a **functional** component -

```
import { getAuth } from 'firebase/auth';
import { ref, onValue } from 'firebase/database';
import db from '../config';
```

```
export default class DrawerNavigator extends Component {
  constructor(props) {
    super(props);
    this.state = {
      light_theme: true
    };
  }
}
```

As we are adding **export default** here make sure to remove the **export** statement written as the last line from the previous class.

Now, similar to how we created the **componentDidMount()** in the Tab Navigation, we will do the same here as well -

```
componentDidMount() {  
  let theme;  
  const auth = getAuth();  
  const userId = auth.currentUser.uid;  
  
  onValue(ref(db, '/users/' + userId), (snapshot) => {  
    theme = snapshot.val().current_theme;  
    this.setState({  
      light_theme: theme === 'light' ? true : false,  
    });  
  });  
}
```

And we'll add our Drawer Navigator inside a **render()** function -

```
render() {  
  return (  
    <Drawer.Navigator>  
      <Drawer.Screen name="Home" component={StackNavigator} options={{ unmountOnBlur: true }} />  
      <Drawer.Screen name="Profile" component={Profile} options={{ unmountOnBlur: true }} />  
      <Drawer.Screen name="Logout" component={Logout} options={{ unmountOnBlur: true }} />  
    </Drawer.Navigator>  
  );  
}
```

Now, we'll add an attribute **drawerContentOptions** to our **<Drawer.Navigator>** component -

```
render() {
  return (
    <Drawer.Navigator
      drawerContentOptions={{
        activeTintColor: '#e91e63',
        inactiveTintColor: this.state.light_theme ? "black" : "white",
        itemStyle: { marginVertical: 5 },
      }}
    >
      <Drawer.Screen name="Home" component={StackNavigator} options={{ unmountOnBlur: true }} />
      <Drawer.Screen name="Profile" component={Profile} options={{ unmountOnBlur: true }} />
      <Drawer.Screen name="Logout" component={Logout} options={{ unmountOnBlur: true }} />
    </Drawer.Navigator>
  );
}
```

Next, we will add another attribute **drawerContent** in which we can define our custom menu for the Drawer -

```
render() {
  let props = this.props;
  return (
    <Drawer.Navigator
      drawerContentOptions={{
        activeTintColor: '#e91e63',
        inactiveTintColor: this.state.light_theme ? "black" : "white",
        itemStyle: { marginVertical: 5 },
      }}
      drawerContent={(props) => <CustomSidebarMenu {...props} />}
    >
      <Drawer.Screen name="Home" component={StackNavigator} options={{ unmountOnBlur: true }} />
      <Drawer.Screen name="Profile" component={Profile} options={{ unmountOnBlur: true }} />
      <Drawer.Screen name="Logout" component={Logout} options={{ unmountOnBlur: true }} />
    </Drawer.Navigator>
  );
}
```

Note that we have created a variable **props** as **this.props** is in our **render()** function above.

We will also have to import **CustomSidebarMenu**, a component we are yet to create.

Let's import it first -

```
import CustomSidebarMenu from "../screens/CustomSidebarMenu";
```

With the **drawerContentOptions** and the **drawerContent** attributes, we are explicitly telling **Drawer.Navigator** to use a component called **CustomSidebarMenu** and are setting relevant properties for it like **activeTintColor**, **inactiveTintColor**, and **itemStyle** (the style for the options in the menu).

Now let's create a file
CustomSidebarMenu.js in our
screens folder -

We will start by making all the imports in this file as shown below -

```
import React, { Component } from 'react';
import {
  SafeAreaView,
  StyleSheet,
  Image,
} from 'react-native';
import firebase from "firebase";

import {
  DrawerContentScrollView,
  DrawerItemList,
} from '@react-navigation/drawer';
```

Next, we will create our class component and export it -

```
export default class CustomSidebarMenu extends Component {
}
```

Inside this component, we will have our **constructor()** and **componentDidMount()** function -

```
constructor(props) {
  super(props);
  this.state = {
    light_theme: true,
  };
}

componentDidMount() {
  let theme;
  const auth = getAuth();
  const userId = auth.currentUser.uid;

  onValue(ref(db, '/users/' + userId), (snapshot) => {
    theme = snapshot.val().current_theme;
    this.setState({
      light_theme: theme === 'light' ? true : false,
    });
  });
}
```

Now let's create the **render()** function -

```
render() {
  let props = this.props;
  return (
    <SafeAreaView style={{ flex: 1, backgroundColor: this.state.light_theme ?
"white" : "#15193c" }}>
      <Image source={require("../assets/logo.png")}
style={styles.sideMenuProfileIcon}></Image>
      <DrawerContentScrollView {...props}>
        <DrawerItemList {...props} />
      </DrawerContentScrollView>
    </SafeAreaView>
  );
}
```

Here, we have a **<SafeAreaView>** component for our drawer in which, we have put the logo of our App in a **<Image>** component.

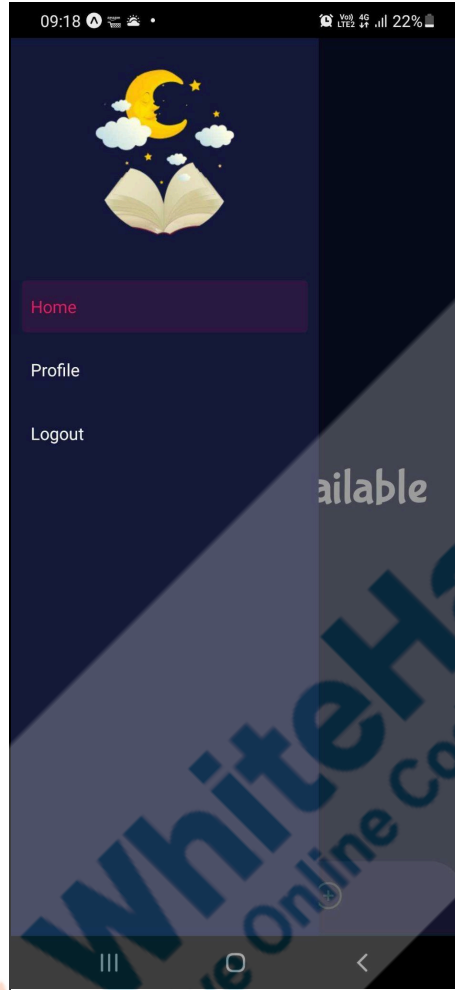
Next, we have our **<DrawerContentScrollView>** component in which we will place our

drawer's content and our drawer's content from our drawer navigator is available to us in the **<DrawerItemList>** component, which we have inside it.

The styles for this would look like this -

```
const styles = StyleSheet.create({
  sideMenuProfileIcon: {
    width: RFValue(140),
    height: RFValue(140),
    borderRadius: RFValue(70),
    alignSelf: "center",
    marginTop: RFValue(60),
    resizeMode: "contain"
  }
});
```

Our drawer now looks like -



That looks wonderful!

Now, most of our App is complete!
We are just left with the like button functionality.

We already have the like button in our **StoryCard** and **StoryScreen**.

Let's first do it in **StoryCard** -

We will start by adding two states in our constructor within **StoryCard.js** -

```
export default class StoryCard extends Component {
  constructor(props) {
    super(props);
    this.state = {
      fontsLoaded: false,
      light_theme: true,
      story_id: this.props.story.key,
      story_data: this.props.story.value,
      is_liked: false,
      likes: this.props.story.value.likes
    };
  }
}
```

The first state is, for if the story is liked, which is false by default. We also have stored the number of likes separately for now.

Next, let's add a **<TouchableOpacity>** component around our like button so that we can make it clickable -

```
<View style={styles.actionContainer}>
  <TouchableOpacity onPress={() => this.likeAction()}>
    <View style={styles.likeButton}>
      <View style={styles.likeIcon}>
        <Ionicons name={"heart"} size={30} color={this.state.light_the
      </View>
      <View>
        <Text style={this.state.light_theme ? styles.likeTextLight : s
      </View>
    </View>
  </TouchableOpacity>
</View>
```

Here, we have used a function **likeAction()** with the **onPress** event of our **<TouchableOpacity>**

Let's now change the styles of the **likeButton**. We want it styled differently for when it's

liked and when it's not liked -

Note: The changes are highlighted for reference purposes.

```
<TouchableOpacity onPress={() => this.likeAction()}>
  <View style={this.state.is_liked ? styles.likeButtonLiked
: styles.likeButtonDisliked}>

    <View style={styles.likeIcon}>
      <Icons name={"heart"} size={30}
color={this.state.light_theme ? "black" : "white"} style={{ width: 30, marginLeft: 20,
marginTop: 5 }} />
    </View>
    <View>
      <Text style={this.state.light_theme ?
styles.likeTextLight : styles.likeText}>{this.state.likes}</Text>
    </View>
  </View>
</TouchableOpacity>
```

Here, based on if the state `is_liked` is **true** or **false**, we are giving our **View** container for the like button different styles. Also, note that we are not using `this.state.likes` for our number of likes.

Let's add the styling for `likeButtonLiked` and `likeButtonDisliked` -

```
likeButtonLiked: {
  backgroundColor: "#eb3948",
  borderRadius: 30,
  width: 160,
  height: 40,
  flexDirection: "row"
},
likeButtonDisliked: {
  borderColor: "#eb3948",
  borderWidth: 2,
  borderRadius: 30,
  width: 160,
  height: 40,
```

```
flexDirection: "row"
},
```

Awesome! Now we still have to create our **likeAction()** function, so let's do that -

```
likeAction = () => {
  if (this.state.is_liked) {
    const dbRef = ref(db, `posts/${this.state.story_id}/`);
    update(dbRef, {
      likes: increment(-1),
    });
  } else {
    const dbRef = ref(db, `posts/${this.state.story_id}/`);
    update(dbRef, {
      likes: increment(1),
    });

    this.setState({ likes: (this.state.likes += 1), is_liked: true });
  }
};
```


Here, if the state **is_liked** is true, that means that the user has disliked the post. This is because we are not yet changing the state of the button and hence, it was previously liked.

If that's the case, we are reducing the number of likes by 1 (by using **increment(-1)**) and updating the states accordingly.

If however, the user has liked the post, we are increasing the number of likes by 1 and then updating the states accordingly.

Note that we had our unique ID for the story stored in our state, and we are using it to identify which story it is on which we want to perform the given operation.

	With this, our ability to like the post's functionality is complete as well.	
--	--	--

	<p>We can now recreate this and add the like functionality in the StoryScreen as well.</p> <p>Now it's your turn. Please share your screen with me.</p>	
Teacher Stops Screen Share		
STUDENT-LED ACTIVITY 1 - 20 mins		
<ul style="list-style-type: none"> • 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. 		
<p>ACTIVITY</p> <ul style="list-style-type: none"> • Adding the like functionality in StoryScreen and finishing the app. 		
<p>Teacher starts slideshow  for slide 10 to 11</p> <p>Refer to speaker notes and follow the instructions on each slide.</p>		
Step 3: Student-Led Activity (20 mins)	<p>Please refer to <i>Student Activity 1</i> to clone the boilerplate code.</p> <p>Don't forget to add the config.js with your credentials in it.</p> <p>You will also have to update the OAuth IDs in the login screen.</p>	<p><i>The student refers to Student Activity 1, clones the repo, and adds the config.js file.</i></p>
	<p>The boilerplate code contains the code we just did for the Drawer Navigation and also the like functionality in the StoryCard.</p> <p>Now it's your turn to add the same functionality in StoryScreen and completing the app.</p>	

The teacher helps the student in writing the code

The student writes the code and completes the app.

Again, we start by adding our two new states in the constructor, **is_liked** set to **false** when initially no users have clicked on the like button. The other state **likes** is using **this.props.route.params.story.story.likes**.

*//Remember that since we passed the story from one screen to another through navigation and not from the parent component to the child component directly, the story's data will not be accessible to us with **this.props.story**. Instead, we will be using - **this.props.route.params** since our route is a prop for our Navigation and the **story** has been passed as a parameter.//*

```
export default class StoryScreen extends Component {
  constructor(props) {
    super(props);
    this.state = {
      fontsLoaded: false,
      speakerColor: "gray",
      speakerIcon: 'volume-high-outline',
      light_theme: true,
      likes: this.props.route.params.story.story.likes,
      is_liked: false
    };
  }
}
```

We change the code for our Like Button to check if it has been clicked or not; we are using a **ternary operator** to change the style depending on the result of Like Button pressed and also calling **this.likeAction()** function written below:

```
<TouchableOpacity onPress={() => this.likeAction()} >
  <View style={this.state.is_liked ?
styles.likeButtonLiked : styles.likeButtonDisliked}>
    <View style={styles.likeIcon}>
      <Ionicons name={"heart"} size={30}
color={this.state.light_theme ? "black" : "white"} style={{ width: 30, marginLeft: 20,
marginTop: 5 }} />
    </View>
  </View>
```

```

<View>
  <Text style={this.state.light_theme ?
styles.likeTextLight : styles.likeText}>{this.state.likes}</Text>
</View>
</View>
</TouchableOpacity>

```

Add styles for it -

```

likeButtonLiked: {
  backgroundColor: "#eb3948",
  borderRadius: 30,
  width: 160,
  height: 40,
  flexDirection: "row"
},
likeButtonDisliked: {
  borderColor: "#eb3948",
  borderWidth: 2,
  borderRadius: 30,
  width: 160,
  height: 40,
  flexDirection: "row"
},

```

We are then creating our **likeAction()** function -

```

likeAction = () => {
  if (this.state.is_liked) {
    const dbRef = ref(db, `posts/${this.props.route.params.story.key}/`);
    update(dbRef, {
      likes: increment(-1),
    });
  } else {
    const dbRef = ref(db, `posts/${this.props.route.params.story.key}/`);
    update(dbRef, {
      likes: increment(1),
    });
  }
}

```



```
});

    this.setState({ likes: (this.state.likes += 1), is_liked: true });
  }
};
```

And voilà! Our Story Telling App is complete.

The student runs the code and clicks on the heart to add the number of likes.



Great Job! We have successfully completed the Storytelling app and in doing this, you have learned so many things.

Congratulations!

Teacher Guides Student to Stop Screen Share

FEEDBACK

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

WRAP-UP SESSION - 5 Mins



Teacher can show slideshow from slides 12 to 22
 Refer to speaker notes and follow the instructions on each slide.

Run the presentation from slide 12 to slide 22.

Following are the wrap-up session deliverables:

- **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

Question	Answer
<p>In which attribute of the Drawer.Navigator, can we define our custom menu for the Drawer?</p> <p>A. drawerContentOptions B. drawerOptions C. drawerContent D. contentOptions</p>	A
<p>In which component is the drawer's content from our drawer navigator available to us?</p> <p>A. DrawerItem component B. DrawerItem component C. DrawerItemList component D. None of the above.</p>	C
<p>What does the following piece of code do?</p>	A

```
update(dbRef, {
  likes: increment(1),
});
```

- A. If the user has liked the post we are increasing the number of likes by 1 (by using increment(1) in the database.
- B. If the user has disliked the post we are incrementing the number of likes by 1 in the database.
- C. If the user has liked the post we are reducing the number of likes by 1 (by using increment(1) in the application only.
- D. None of the above.

End the quiz panel

Amazing work today! You get a "hats-off".

Alright. See you in the next class, we will brainstorm some ideas to create your own app. So keep thinking!! I am looking forward to hearing some great ideas from you.

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 - 9

Goal of the Project:

In Class 89, We created the custom styling for Drawer Navigation, added the like functionality. In this project,

The students engage with the teacher over the project.

you will practice the concepts learned in the class to create a custom styling for Drawer Navigation.

**This is a continuation project of 81 to 88, 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 decided to create a social media app for her and all upcoming talents. She has asked for your help to create an App.

We have reached the final stage of the project. Guide Jenny to create a custom styling for Drawer Navigation.

Teacher ends slideshow



Teacher Clicks

✕ End Class

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?

The student uses the Markdown editor to write their reflections in a reflection journal.

	<ul style="list-style-type: none"> • 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	Previous Class Code	https://github.com/React-Native-Frontier/PRO-C88-Story-Teller-TA-Solution
Teacher Activity 2	Reference Code	https://github.com/React-Native-Frontier/PRO-C89-Story-Teller-TA-Solution
Teacher Activity 3	Teacher Aid	https://drive.google.com/file/d/1WA1BQff4dmgv5BlnU3f_imk4vlpvAyMa/view?usp=sharing
Student Activity 1	Boilerplate Code	https://github.com/React-Native-Frontier/PRO-C89-Story-Teller-SA-Boilerplate
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/slides-pro-c89_withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/3c9dd60a-0152-45d3-9221-bba44fb74765.pdf