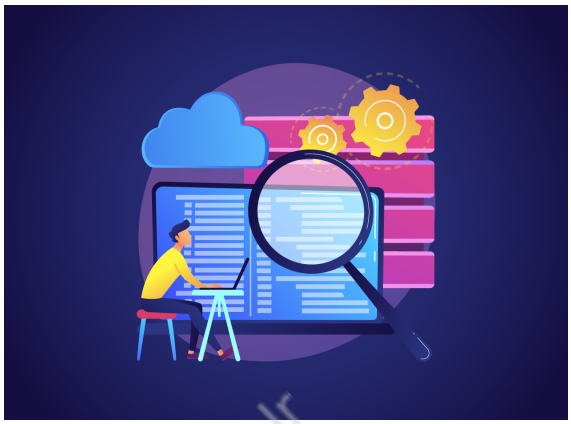
SAVE & READ STORIES USING FIREBASE



What is our GOAL for this MODULE?

In this class, we resolved the bug we were faced due to the cache memory of the device and integrated firebase to save and retrieve stories to the app.

What did we ACHIEVE in the class TODAY?

- Resolved the bug in which the Create Story Screen and the Feed Screen were not getting called every time due to caching.
- Integrated firebase to save stories to the database.
- Read stories from the database.

Which CONCEPTS/ CODING BLOCKS did we cover today?

- Making queries on firebase.
- Handling mobile phone caching issue.
- Unmounting components used in the Storytelling App.

How did we DO the activities?

1. Add the Submit button to **CreateStory.js**.

```
                    <View style={styles.fieldContainer}>
                        <TextInput
                            style={[this.state.light_theme ? styles.inputFontLight : styl
                            onChangeText={(moral) => this.setState({ moral })}
                            placeholder={"Moral of the story"}
                            multiline={true}
                            numberOfLines={4}
                            placeholderTextColor={this.state.light_theme ? "black" : "whi
                        />
                    </View>
                    <View style={styles.submitButton}>
                        <Button
                            onPress={() => this.addStory()}
                            title="Submit"
                            color="#841584"
                        />
                    </View>
                </ScrollView>
            </View>
```

2. Call the **addStory()** function when the **Submit** button is clicked by the user.

```
async addStory() {
    if (this.state.title && this.state.description && this.state.story &&
this.state.moral) {
      let storyData = {
        preview_image: this.state.previewImage,
        title: this.state.title,
        description: this.state.description,
        story: this.state.story,
        moral: this.state.moral,
        author: firebase.auth().currentUser.displayName,
        created_on: new Date(),
        author_uid: firebase.auth().currentUser.uid,
        likes: 0
      }
      await firebase
```

```
            .database()
            .ref("/posts/" + (Math.random().toString(36).slice(2)))
            .set(storyData)
            .then(function (snapshot) {


            })
        this.props.navigation.navigate("Feed")
    } else {
        Alert.alert(
            'Error',
            'All fields are required!',
            [
                { text: 'OK', onPress: () => console.log('OK Pressed') }
            ],
            { cancelable: false }
        );
    }
}
```

3. Create the **fetchStories()** function and call it in the **componentDidMount()** function.

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

fetchStories = () => {
    firebase
        .database()
        .ref("/posts/")
        .on("value", (snapshot) => {
            let stories = []
```

```
        if (snapshot.val()) {
            Object.keys(snapshot.val()).forEach(function (key) {
                stories.push({
                    key: key,
                    value: snapshot.val()[key]
                })
            });
        }
        this.setState({ stories: stories })
    }, function (errorObject) {
        console.log("The read failed: " + errorObject.code);
    })
}
```

4. Write the code to handle the condition if there are no stories in the app.

```
return (
        <View style={this.state.light_theme ? styles.containerLight :
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={this.state.light_theme ? styles.appTitleTextLight :
styles.appTitleText}>
                        Storytelling App
                    </Text>
                </View>
            </View>
            {
```

```
            !this.state.stories[0] ?
              <View style={styles.noStories}>
                <Text style={this.state.light_theme ? styles.noStoriesTextLight :
styles.noStoriesText}>No Stories Available</Text>
              </View>
              : <View style={styles.cardContainer}>
                <FlatList
                  keyExtractor={this.keyExtractor}
                  data={this.state.stories}
                  renderItem={this.renderItem}
                />
              </View>
          }
        </View>
)
```

5. Change the **constructor()** in our **StoryCard.js** to store the keys and values separately.

```
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
        };
    }
```

6. Create an object which maps the value of these keys with the path of their respective image.

```
render() {
    let story = this.state.story_data
    if (!this.state.fontsLoaded) {
        return <AppLoading />;
    } else {
        let images = {
            "image_1": require("../assets/story_image_1.png"),
            "image_2": require("../assets/story_image_2.png"),
            "image_3": require("../assets/story_image_3.png"),
            "image_4": require("../assets/story_image_4.png"),
            "image_5": require("../assets/story_image_5.png")
        }
        return (
```

7. Change the image source of the **<Image>** component that displays the image of the story.

```
<Image source={images[story.preview_image]}
```

8. Update the **render()** function.

```
render() {
  let story = this.state.story_data;
  if (!this.state.fontsLoaded) {
    return <AppLoading />;
  } else {
    let images = {
      image_1: require("../assets/story_image_1.png"),
      image_2: require("../assets/story_image_2.png"),
      image_3: require("../assets/story_image_3.png"),
      image_4: require("../assets/story_image_4.png"),
      image_5: require("../assets/story_image_5.png")
    };
    return (
      <TouchableOpacity
        style={styles.container}
```

```
    onPress={() =>
      this.props.navigation.navigate("StoryScreen", {
        story: this.props.story
      })
    }
  >
    <SafeAreaView style={styles.droidSafeArea} />
    <View
      style={
        this.state.light_theme
          ? styles.cardContainerLight
          : styles.cardContainer
      }
    >
      <Image
        source={images[story.preview_image]}
        style={styles.storyImage}
      ></Image>
      <View style={styles.titleContainer}>
        <View style={styles.titleTextContainer}>
          <Text
            style={
              this.state.light_theme
                ? styles.storyTitleTextLight
                : styles.storyTitleText
            }
          >
            {story.title}
          </Text>
          <Text
            style={
              this.state.light_theme
                ? styles.storyAuthorTextLight
                : styles.storyAuthorText
            }
```

```
        >
          {story.author}
        </Text>
        <Text
          style={
            this.state.light_theme
              ? styles.descriptionTextLight
              : styles.descriptionText
          }
        >
          {this.props.story.description}
        </Text>
      </View>
    </View>

    <View style={styles.actionContainer}>
      <View style={styles.likeButton}>
        <Ionicons
          name={"heart"}
          size={RFValue(30)}
          color={this.state.light_theme ? "black" : "white"}
        />
        <Text
          style={
            this.state.light_theme
              ? styles.likeTextLight
              : styles.likeText
          }
        >
          12k
        </Text>
      </View>
    </View>
  </View>
</TouchableOpacity>
```

```
  );
 }
}
```

At this stage, we observed a bug that, our **componentDidMount()** on the Create Story Screen and the Feed Screen is not getting called every time we come to those screens because our device is caching those screens in the RAM memory. We will fix this bug in the following steps:

9. Create the state in the **constructor()** named **isUpdated** and set it to **false**. This property will be updated from our screens of the app.

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

10. Write a function to update the state value of **isUpdated.** This will force the **Tab Navigator** to update itself, and it will end up updating the screens from the database instead of the cache memory.

```
changeUpdated = () => {
    this.setState({ isUpdated: true })
}

removeUpdated = () => {
    this.setState({ isUpdated: false })
}
```

11. Create a function **renderFeed()** which returns the components with **props** we need and then use those functions instead of the component in our **<Tab.Screen>**.

```
renderFeed = (props) => {
    return <Feed setUpdateToFalse={this.removeUpdated} {...props} />
```

```
}

renderStory = (props) => {
   return <CreateStory setUpdateToTrue={this.changeUpdated} {...props} />
}
```

```
<Tab.Screen name="Feed" component={this.renderFeed} />
<Tab.Screen name="Create Story" component={this.renderStory} />
```

12. In the **CreateStory.js**, use the **setUpdatedToTrue** function right before we are navigating to the Feed Screen after saving the story in the database.

```
this.props.setUpdateToTrue()
this.props.navigation.navigate("Feed")
```

13. The **FeedScreen.js** will use the **setUpdatedToFalse** right after we have successfully fetched the stories.

```
fetchStories = () => {
    firebase
        .database()
        .ref("/posts/")
        .on("value", (snapshot) => {
            let stories = []
            if (snapshot.val()) {
                Object.keys(snapshot.val()).forEach(function (key) {
                    stories.push({
                        key: key,
                        value: snapshot.val()[key]
                    })
                });
            }
            this.setState({ stories: stories })
            this.props.setUpdateToFalse()
        }, function (errorObject) {
            console.log("The read failed: " + errorObject.code);
        })
}
```

The second bug we fixed is in our Tab Navigator and Drawer Navigator, we can specify that we want to **unmount** a component as soon as a user goes away from a screen.

14. Change the code in **TabNavigator** and **Drawer Navigator** as shown below:

```
<Tab.Screen name="Feed" component={this.renderFeed} options={{
unmountOnBlur: true }} />
<Tab.Screen name="Create Story" component={this.renderStory} options={{
unmountOnBlur: true }} />

<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 }} />
```


What's NEXT?

In the next class, we will be working on custom-styled drawer navigation.

EXTEND YOUR KNOWLEDGE

Bookmark the following link to know learn about Firebase:
https://firebase.google.com/docs/guides