Filipe Lutz Mariano 25956

Mobile Apps 2

Assignment Eight

Submission Date 09/12/2024

Percent 10%

Include all workings and references

Section A.

1. How is Responsiveness achieved in JetPack Compose? (*3 marks*)

Responsiveness in Jetpack Compose is achieved through several core principles and features:

1. **Declarative UI Paradigm**: Jetpack Compose uses a declarative approach, where the UI is automatically updated when the underlying state changes. This ensures that the user interface always reflects the current state of the application.

2. **State Management**: Compose leverages state management tools, such as remember and MutableState, to monitor and react to changes. Any updates to the state trigger a recomposition of the affected UI elements, making the application responsive to user actions and data changes.

3. **Modifiers**: Modifiers in Compose allow developers to dynamically adapt layout, styling, and behavior to different screen sizes, orientations, and configurations, ensuring a responsive design.

4. **ConstraintLayout & Adaptive Design**: Compose supports layouts like ConstraintLayout and responsive design principles, enabling developers to create flexible UIs that adapt seamlessly to various screen sizes and resolutions.

5. **Animation and Gestures**: Jetpack Compose provides built-in support for animations and gestures, which enhances responsiveness by enabling smooth transitions and interactive experiences.

2. Give two real-world examples of where you would use DeepLinking (*2 marks*)

I. **E-commerce Application**:
DeepLinking can be used to direct users to a specific product page within the app when they click on a link in an email campaign or an advertisement. For example, clicking a link for a "50% off sale" would open the app directly at the relevant sale page.

II. **Social Media Sharing**:
DeepLinking is commonly used in social media apps to navigate directly to specific user profiles, posts, or messages when a user clicks on a shared link. For example, clicking on a tweet link opens the X (Twitter) app directly at the tweet's detailed view.

Section B.

1. Using an existing assignment, create and show the workings of a DeepLink (*3.5 marks*)
2. Create two buttons that will stop and start a specific sound on your app (*1.5 marks*)

Table of Contents

Report: Music Player with DeepLink Integration

1. Project Overview

The goal of this project was to create two buttons that will stop and start a specific sound and show the workings of a DeepLink functionality. This was achieved by integrating navigation, media playback controls, and establishing deep linking to allow users to open the Music Player application directly via a web link.

The final implementation supports:

- Navigating between screens within the app.
- Playback of a music list with play, pause, and navigation between tracks.
- DeepLink functionality that allows opening the app's Music Player screen from a web page.

2. Project Specifications

Specification 1: Using an existing assignment, create and show the workings of a DeepLink (3.5 marks)

How It Was Achieved:

To fulfill this requirement, I implemented DeepLink functionality in my app to allow users to open a specific music player screen from a web link. The key steps involved:

1. Creating a new GitHub Repository: I set up a repository at https://github.com/FilipeLutz/DeepLink.
2. Setting up the DeepLink Directory:
    - Created a ".well-known" folder in the repository root.
    - Added the "assetlinks.json" file within this folder to define app link settings provided by Android Studios.

This configuration ensures proper handling of the DeepLink by associating the domain https://filipelutz.github.io/DeepLink with the app.

Challenges Encountered:

Setting up DeepLinks was challenging as it involved:

- Creating the ".well-known" directory and the "assetlinks.json" file properly formatted and hosted correctly on GitHub Pages.
- Debugging the app-link association required testing the correct URL fingerprint matching.

Web Page with DeepLink Button:

I created a web page to provide a link that opens the Music Player screen via the DeepLink URL. This page can be found at the following link:

https://filipelutz.github.io/DeepLink

Specification 2: Create two buttons that will stop and start a specific sound on your app (1.5 marks)

How It Was Achieved:
To meet this requirement, I implemented playback controls for music within the app, allowing the user to control playback seamlessly. The Music Player screen features **three buttons** for controlling playback:

- **Previous Button:** Allows users to go back to the previous song in the playlist.
- **Play/Pause Button:** Allows users to toggle playback of the currently selected track (play or pause).
- **Next Button:** Allows users to skip to the next song in the playlist.

These three buttons are part of a user-friendly interface that lets users navigate between tracks and control playback states with ease.

Additionally, I downloaded **five songs** from https://kingdomsouthafrica.co.za/ and added them to a folder named raw in the res directory of the project. These songs are used in the Music Player screen for playback.

3. How to Copy and Use the Project from GitHub:

To set up and run the project locally, follow these steps:

1. **Clone the Repository:**
   First, copy the project repository from GitHub: **DeepLink**

2. **Open the Project in Android Studio:**
   After cloning the repository, open Android Studio. Click on **Open an Existing Project** and navigate to the location where the project was cloned. Select the project folder to open it.
3. **Set Up the Emulator:**
   In Android Studio, set up the Android Emulator to test the application. Go to **Tools > AVD Manager** and configure a device emulator that matches your target device or use a physical device for testing if connected.
4. **Build and Run the App:**
   Once the emulator is ready, build and run the app using the green **Play** button in Android Studio. This will install the application on the emulator for testing.
5. **Open the Web Page with the DeepLink Button:**
   To test the DeepLink functionality, open the web page with the DeepLink button at the following URL: https://filipelutz.github.io/DeepLink
   Click the **Open Music Player** on this web page, and it will trigger the application through the configured DeepLink mechanism.

Following these steps will allow you to interact with the application and test the DeepLink and playback functionalities locally.

4. Conclusion:

In conclusion, the project successfully meets the objectives by implementing DeepLink functionality and integrating music playback features. Using DeepLink, users can seamlessly open the application from external links, enhancing user experience and accessibility. Additionally, the music player features three buttons, Play/Pause, Previous, and Next allowing users to control playback effectively.

The development process involved configuring a DeepLink with a GitHub hosted web page and setting up an assetlinks.json file to ensure proper verification and navigation. Although the DeepLink setup required additional research and configuration, it was ultimately achieved, demonstrating a better understanding of app-linking protocols.

Overall, the project meets all the assignment requirements and provides a functional and user-friendly interface for music playback and DeepLink navigation.

5. References:

1. Layouts in Compose
2. Compose modifiers
3. State and Jetpack Compose
4. The FULL Deeplinking Guide With Jetpack Compose Deep Linking in Jetpack Compose
5. Create Deep Links to App Content
6. Create a deep link for a destination
7. Create a basic media player app using Media3 ExoPlayer
8. Supported media formats
9. How to Build a Simple Video Player With Jetpack Compose & ExoPlayer (Media3)
10. https://kingdomsouthafrica.co.za/