HOMEWORK ASSIGNMENT 4

CSCI 571 – Spring 2025

Abstract

Java, Kotlin, JSON, Android Studio, Retrofit, Coil, Jetpack Compose, Android Lifecycle, Material Design, PersistentCookieJar and Artsy API

Marco Papa

papa@usc.edu

Contents

1. OBJECTIVES

- Become familiar with Java, Kotlin, JSON, Android Lifecycle and Android Studio for Android app development.
- Build a good-looking Android app.
- Learn the essentials of Google's Material design rules for designing Android apps
- Learn to use the Artsy APIs and the Android SDK.
- Get familiar with third party libraries like Coil, Retrofit and OkHttp.

The objective is to create an Android application as specified in the document below and in the reference video.

2. BACKGROUND

2.1 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android application development, based on IntelliJ IDEA - a powerful Java IDE. On top of the capabilities you expect from IntelliJ, Android Studio offers:

- Flexible Gradle - based build system.
- Build variants and multiple apk file generation.
- Code templates to help you build common app features.
- Rich layout editor with support for drag and drop theme editing.
- Lint tools to catch performance, usability, version compatibility, and other problems.
- ProGuard and app-signing capabilities.
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine.

More information about Android Studio can be found at:
http://developer.android.com/tools/studio/index.html

2.2 Android

Android is a mobile operating system initially developed by Android Inc. a firm purchased by Google in 2005. Android is based upon a modified version of the Linux kernel.

The Official Android home page is located at:
http://www.android.com/
The Official Android Developer home page is located at:
http://developer.android.com/

2.3 Google App Engine (GAE)

Google App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change. With App Engine, there are no servers to maintain. You simply upload your application and it's ready to go. App Engine applications automatically scale based on incoming traffic. Load balancing, micro services, authorization, SQL and NoSQL databases, memcache, traffic splitting, logging, search, versioning, roll out and roll backs, and security scanning are all supported natively and are highly customizable.

To learn more about GAE support for Node.js visit this page:

https://cloud.google.com/appengine/docs/nodejs

3. PREREQUISITES

This Assignment requires the use of the following components:

- Download and install Android Studio. Technically, you may use any IDE other than Android Studio such as Eclipse, but the latest SDKs may not be supported with Eclipse. We will not be providing any help on problems arising due to your choice of alternate IDEs.

- You must use the **emulator, preferably Pixel 8 Pro with API 34**. Everything should just work out of the box.
- If you are new to Android Development, Hints are going to be your best friends!

4. HIGH LEVEL DESIGN

This Assignment is a mobile app version of the front-end of Assignment 3.

In this assignment, you will develop an Android application, which allows users to search for different artists and look at the detailed information about them. Users can also log in to save their favorite artist to view them or track their artworks. The App contains 5 screens: Home screen, Search Result screen, Detailed Artist Information screen, User Login screen and User Register Screen. However, the App has multiple features on each of these screens.

This Assignment contains 5 API calls. There are the calls to the Artsy API Service for authentication, search, artist, artwork, gene (category). Each of these 5 API calls are the same ones used in Assignment #3 (4-2 section). Therefore, you can use the same Node.js backend as Assignment #3. In case you need to change something in NodeJS, make sure you do not break your Angular assignment (or deploy a separate copy) as the grading for Assignment may not have been completed.

Both Kotlin and Java are allowed. This document lists Kotlin components and libraries. **Jetpack Compose** is also supported and recommended for UI development.

Note: This app has been designed and implemented in a Pixel 8 Pro emulator by using SDK API 34. It is highly recommended that you use the same virtual device and API to ensure consistency.

The assignment must be run on an Android Studio emulator and recorded using Zoom video recording, as instructed. No personal devices will be allowed.

5. IMPLEMENTATION

Figure 1.1

App Icon

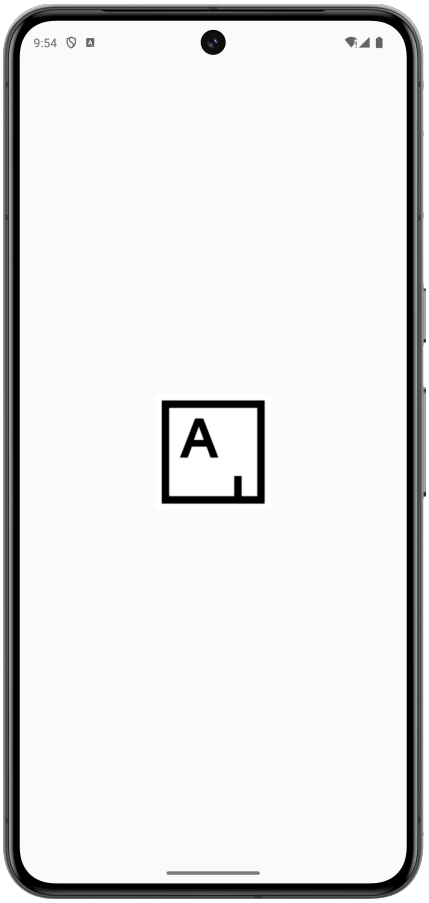


Figure 1.2

Splash Screen

5.1 App Icon and Splash Screen

To get the app icon/image, please see the **hints** section.

The app begins with a welcome screen (Figure 1.2) which displays the icon provided in the hint above.

This screen is called Splash Screen and can be implemented using many different methods. The simplest one is to create a resource file for the launcher screen and add it as a style to AppTheme.Launcher (see **hints**). Please refer to Figure 1.1 and Figure 1.2.

5.2 Home screen

When you open the app, the data is being fetched using Retrofit and Kotlin Coroutines. The home screen will have a toolbar on top left with title “Artist Search”, a **Search Icon** and **User Icon** (see **hints**) on top right. Below that, it will show the current date as shown in Figure 2.1.

There is 1 Section on the home screen called Favorites Section which is described below.

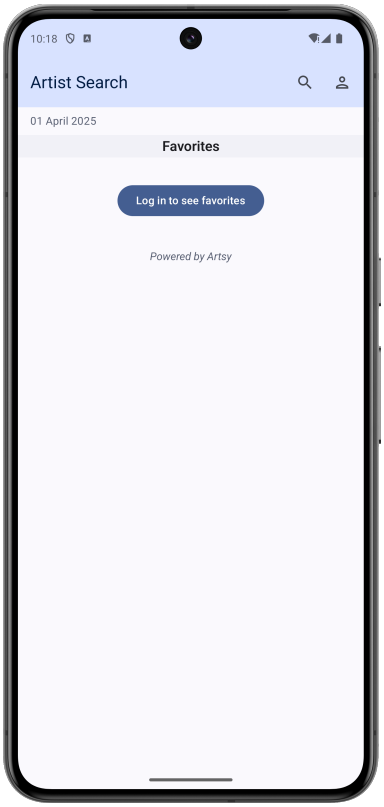


Figure 2.1

Home Screen (Log out)

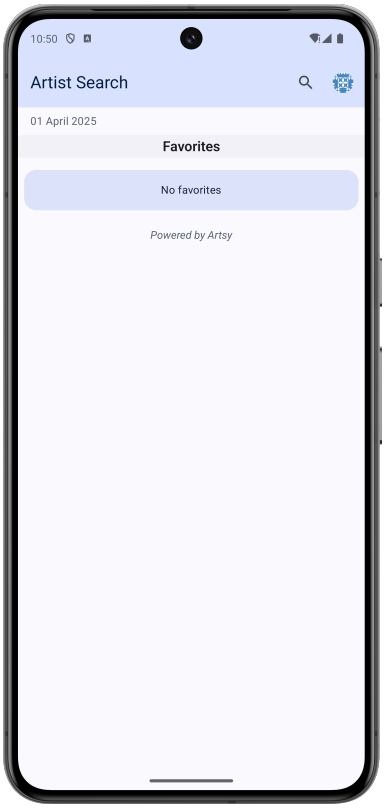


Figure 2.2

Home Screen (Log in)

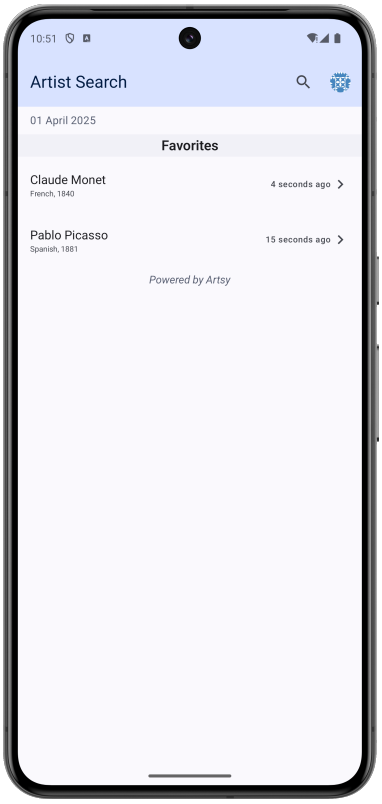


Figure 2.3

If the user starts the app with:

- **Log out State:**
  - A clickable button with the text "Log in to see favorites" will be displayed as shown in Figure 2.1.
  - Clicking the button navigates the user to the **Login Screen**
- **Log in State:**
  - If the user has no favorite artists, a Box with text “No favorites” will be displayed as shown in Figure 2.2.
  - Otherwise, the user’s favorite artists will be listed in the Favorites Section.

Favorites Section - This section will show all the artists that have been marked as “favorite” by the logged in user. For each favorited artist show (See Figure 2.3):

- Artist Name
- Nationality
- Birthday
- Added Time

Each artist listing also has a **Right Arrow** button (see **hints**) on the extreme right, next to the Added Time field. On clicking the button, the Detailed Artist Information screen will open for the selected artist.

At the bottom of the favorites sections, we have a ‘Powered by Artsy’ text in italics as shown in Figure 2.3. On clicking this text, the App should open the Artsy homepage in a browser. (Artsy URL: https://www.artsy.net/).

The home screen has been implemented using Jetpack Compose. The artist listings and sections are displayed using a **LazyColumn**. Each artist item is implemented as a composable function, such as **ArtistCard** or **FavoriteArtistListItem**. The UI dynamically updates based on the user's login and favorites status as discussed.

The **Search** button on the toolbar opens the search bar to type the artist’s name to search.

5.3 Search Functionality

- On the top right side, there will be a search button which opens a textbox where the user can enter a keyword to search for an artist name (See Figure 3.1).
- When the user enters more than 3 characters, the app automatically starts searching and calls the backend API to fetch results.
- For every additional character entered, the search results are updated dynamically by making another API call to the backend.

Implementing search functionality requires only searchable – see **hints**

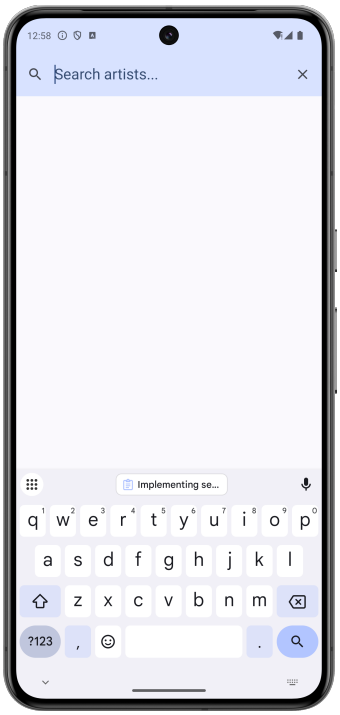


Figure 3.1
Search Bar

5.4 Search Result Screen

When the user performs a search from the home screen using the search bar, the app dynamically fetches results from the backend as the user types (starting from 3 characters). No explicit loading indicator is displayed during the search process. Instead:

- The search results are displayed in a **vertically scrollable list** implemented using Jetpack Compose's **LazyColumn**. (see **hints**)
- Each artist is displayed in a **Card View** with their name and image. (see **hints**)
- Images can be loaded asynchronously using **Coil**. (see **hints**)
    - Images are resized and fitted to ensure uniformity.
    - If the image takes longer to load, Coil handles the loading state internally.
- If the search results are empty, the screen remains blank.
- A **Close** icon (see **hints**) button is on extreme right to clear input

See the reference video and **Figures 4.*.**

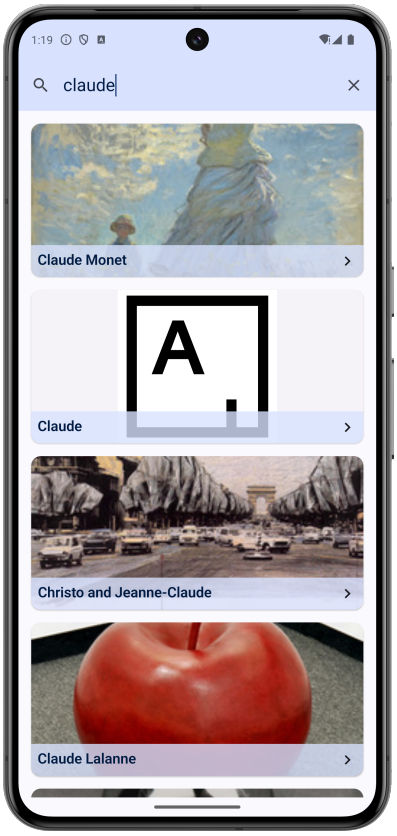


Figure 4.1

Search Result Screen

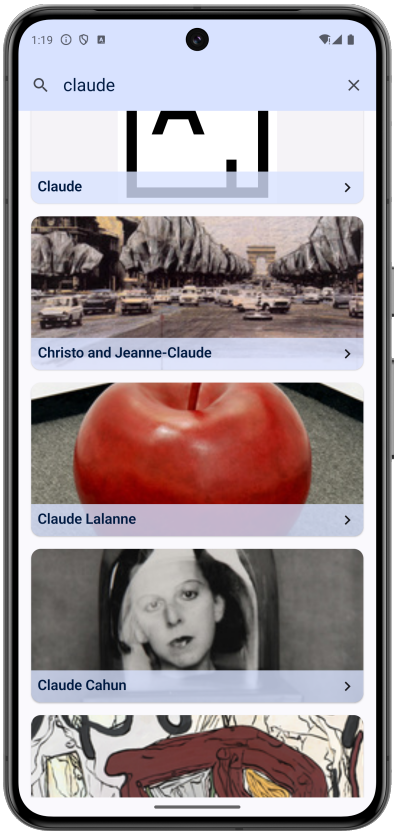


Figure 4.2

Vertically Scrollable

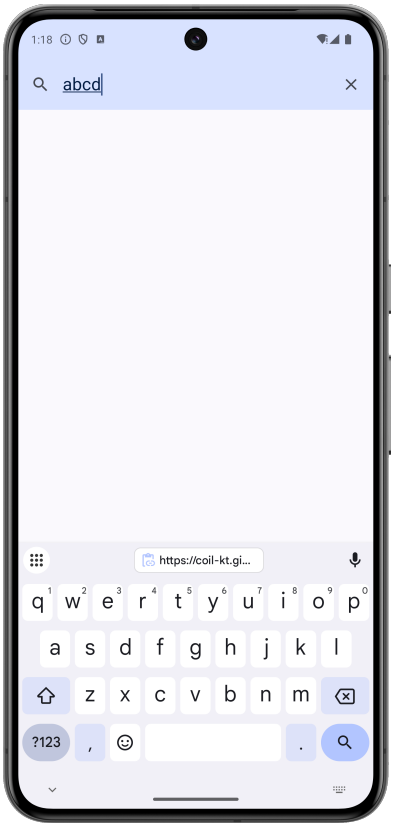


Figure 4.3

No Result Found

5.5 Detailed Artist Information Screen

On clicking the **Right Arrow** button in the favorites section on **Home screen** **or** clicking an artist **Card View** from the search result screen or from **Similar Tab** (when user logged in) on Detailed Artist Information Screen, a progress bar should be displayed while the artist details are being fetched. Once the data has been fetched the progress bar should disappear and a Tabbed Screen with Details Tab (Default) should be available to the user **(se Figure 5.2)**.

The top action bar must show the artist's name and the **Back Icon** (see **hints**) button to go back to the previous screen. The action bar should also contain a favorite icon on right top to add or remove the artist from favorites list (when user logged in). The favorite icon

will either be filled or bordered depending on whether the artist is marked as favorite or not. Adding/Removing the artist from favorites should also display a snackbar message (See hint) as shown in the video.

Below the Action bar, there should be 3 Tab Headings: **Details**, **Artworks** and **Similar** (when user logged in) with their respective icons (see **hints**) on it. By Default, the Details tab should be selected whenever this screen is opened. Each Tab should contain the following when selected.

Details Tab:

- The Details Tab should contain Artist Name, Artist Nationality, Artist Birthday, Artist Deathday and Artist Biography.
- The whole page should be vertically scrollable

If any of the above-mentioned information is null or missing, handle it properly by not displaying its entire part.

Artworks Tab:

- The tab displays a vertically scrollable list of artworks using a **LazyColumn**.
- Each artwork is displayed in a **Card View** containing:
    - Artwork Name: Displayed as a title
    - Artwork Image: Loaded asynchronously using Coil and displayed with a consistent size and aspect ratio.
    - A clickable 'View categories' button: showing a **Dialog** (see **hints**) when clicking on it.
        - The **AlertDialog** has 'Categoreis' shown as title and a confirm button 'Close' to close the dialog.
        - If the category data is still loading, a **Loading Indicator** (see **hints**) is displayed inside the dialog.
        - Once the data is loaded, the category details are displayed inside the dialog.
        - Category information are stored in a **Card View** and displayed in a **Carousel Loop** (see **hints**). Each Category Card contains category name, category image and category description.

If there is no artwork for the artist, 'No Artworks' should be displayed.

- The tab displays a vertically scrollable list of similar artists using a **LazyColumn**
- Each similar artist is displayed in a **Card View** with their name and image.

If there are no similar artists for the artist, 'No Similar Artists' should be displayed.

See reference video and **Figure 5.*.**

Figure 5.1

Initial Progress bar

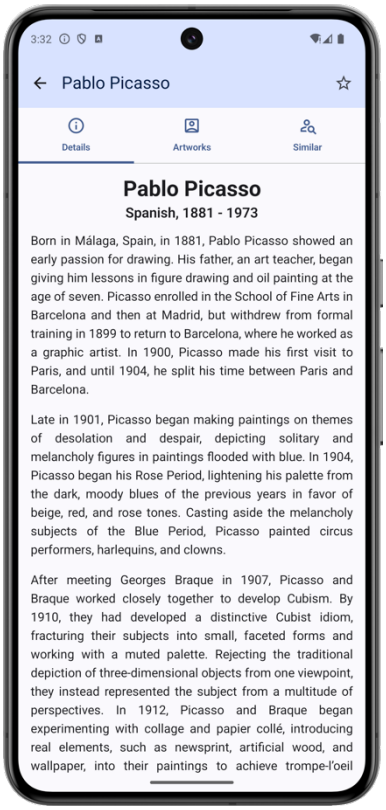


Figure 5.2

Default Details Tab

Figure 5.3

Artworks Tab

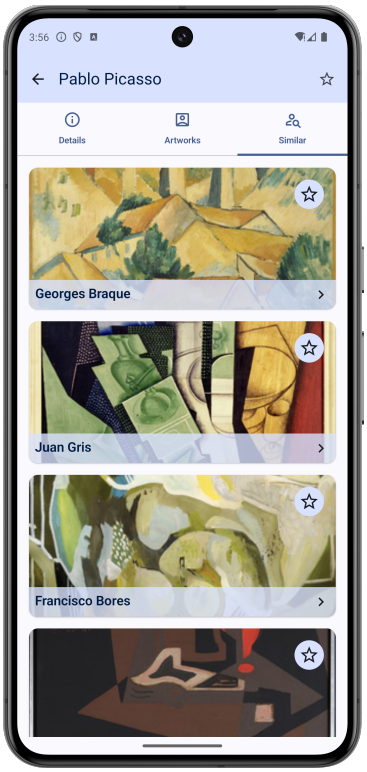


Figure 5.4

Similar Tab

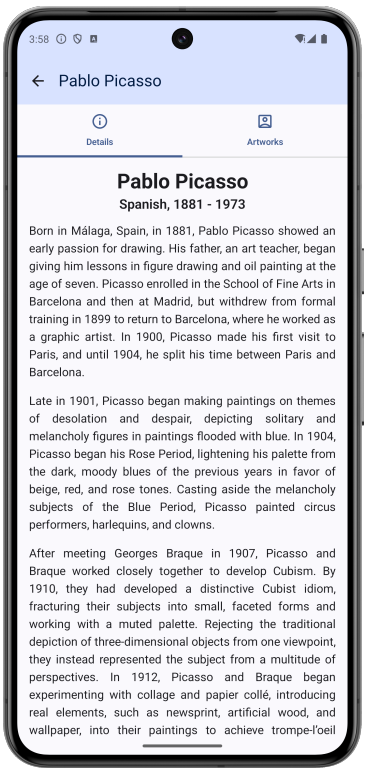


Figure 5.5

Default Details Tab

(User logged out)

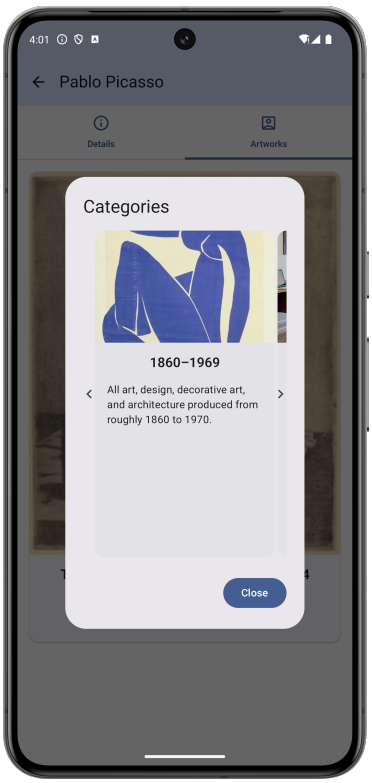


Figure 5.6

Category Dialog Box

Figure 5.7

No Artwork

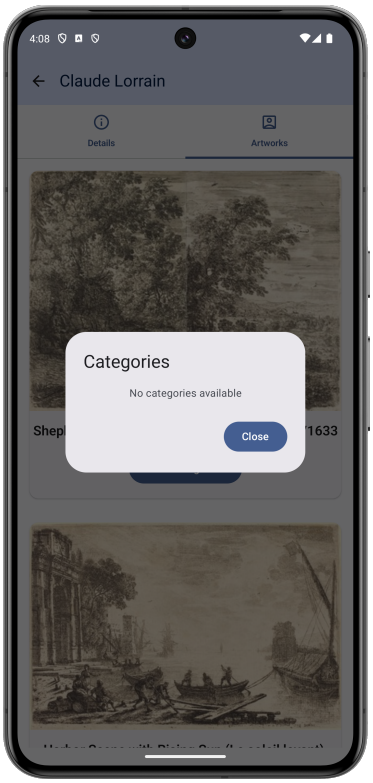


Figure 5.8

No Category

5.6 Login Screen

The Login Screen allows users to log in to their account by entering their email and password. It is implemented using **Jetpack Compose** and follows a modern, declarative UI approach. It contains the following parts:

Top Action Bar:

- A **Back Icon** on the left to navigate back to the previous screen.
- The title 'Login' displayed next to the button.

Login Form:

- Email Input Field:
  - An **OutlinedTextField** is used for entering the email address (see **hints**).
  - The field validates the email format when the user moves focus away.
  - If the email is invalid, an error message is displayed below the field.
- Password Input Field:
  - An **OutlinedTextField** is used for entering the password.
  - The password is hidden using **PasswordVisualTransformation** (see **hints**).
  - The field validates the password when the user moves focus away.
  - If the password is invalid, an error message is displayed below the field.
- Login Button:
  - A Button is used to submit the login form.
  - If the login process is in progress, a **CircularProgressIndicator** is displayed inside the button.
  - The button is **disabled** while the login process is ongoing.
- Error Message:
  - If the login fails (e.g., invalid credentials), an error message is displayed below the login button.
- Register Link:
  - A clickable 'Register' link is displayed below the form.
  - Clicking the link navigates the user to the **Register Screen**.

Snackbar:

- Upon successful login, a **Snackbar** (see **hints**) is displayed with the message 'Logged in successfully'.
- The user is then navigated to the **Home Screen**.

See the reference video and **Figure 6.*.**

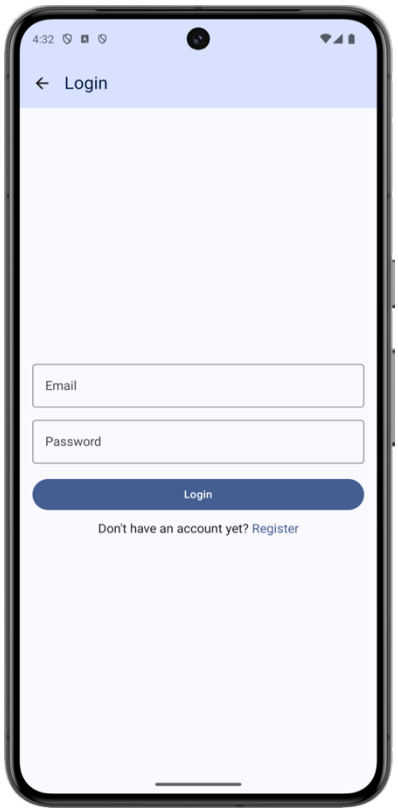


Figure 6.1

Login Screen

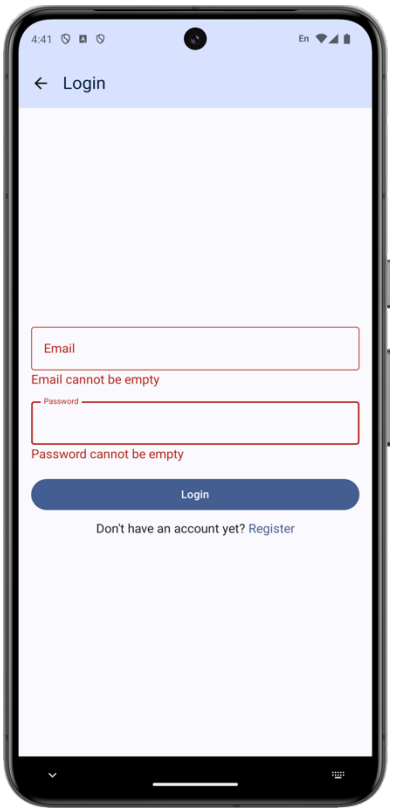


Figure 6.2

Input validation

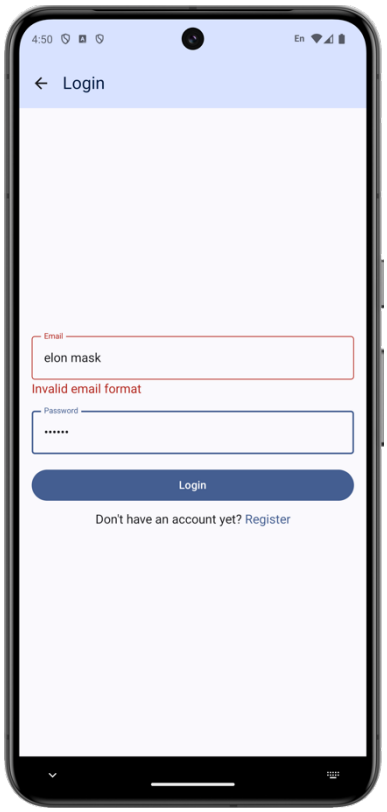


Figure 6.3

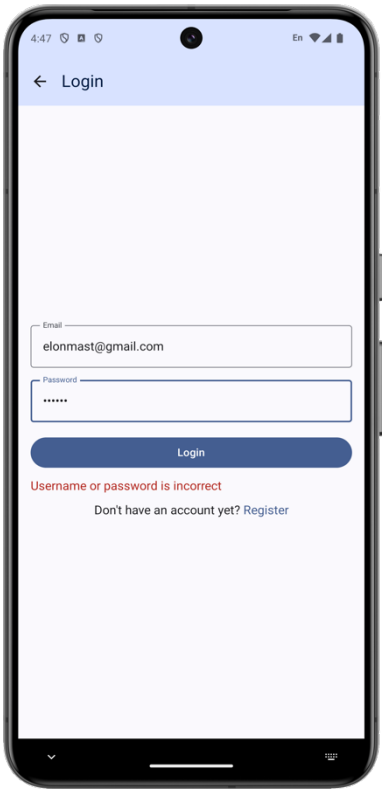


Figure 6.4

Error Message

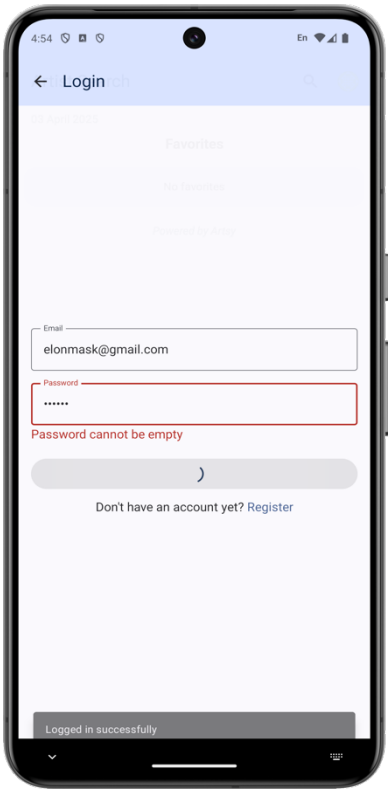


Figure 6.5

Login loading

& Snackbar

5.7 Register Screen

The Register Screen allows users to create a new account by entering their full name, email, and password. It is implemented using Jetpack Compose and follows a modern, declarative UI approach. It contains the following parts:

Top Action Bar:

- A **Back Icon** on the left to navigate back to the previous screen.
- The title ‘Register displayed next to the button.

Registration Form:

- Full Name Input Field:
  - An **OutlinedTextField** is used for entering the user's full name.
  - The field validates the input when the user moves focus away.
  - If the full name is invalid, an error message is displayed below the field.
- Email Input Field:
  - An **OutlinedTextField** is used for entering the email address.
  - The field validates the email format when the user moves focus away.
  - If the email is invalid, an error message is displayed below the field.
- Password Input Field:
  - An **OutlinedTextField** is used for entering the password.
  - The password is hidden using **PasswordVisualTransformation**.
  - The field validates the password when the user moves focus away.
  - If the password is invalid, an error message is displayed below the field.
- Register Button:
  - A Button is used to submit the registration form.
  - If the registration process is in progress, a **CircularProgressIndicator** is displayed inside the button.
  - The button is disabled while the registration process is ongoing.
- Error Message:
  - If the registration fails (e.g., invalid input or server error), an error message is displayed below the register button or corresponding input field.
- Login Link:
  - A clickable "Login" link is displayed below the form.
  - Clicking the link navigates the user to the Login Screen.

Snackbar:

- Upon successful registration:
  - A **Snackbar** is displayed with the message *"Registered successfully"*.
  - The user is then navigated to the **Home Screen**.

See the reference video and **Figure 7.*.**

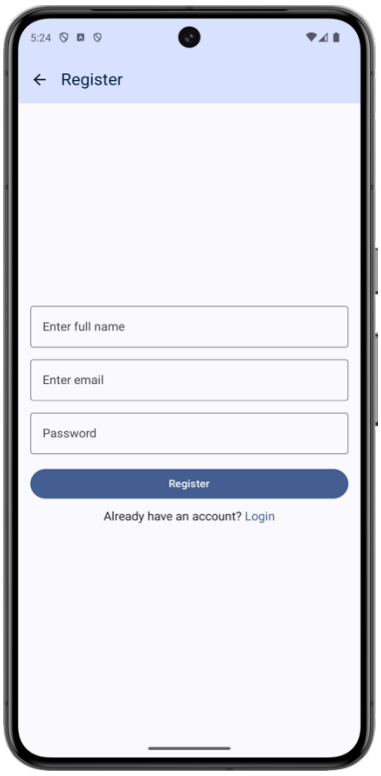


Figure 7.1

Register Screen

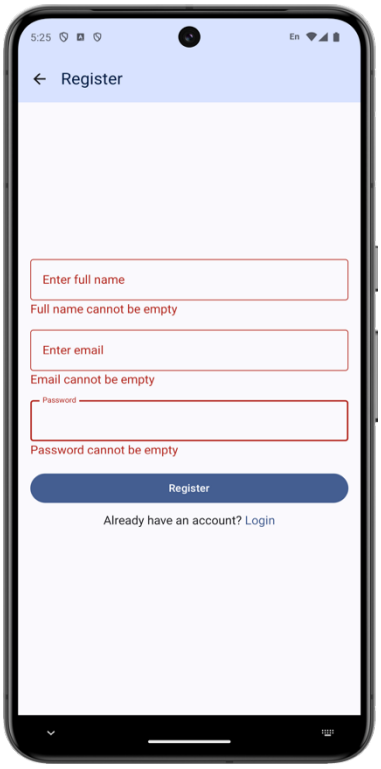


Figure 7.2

Input validation

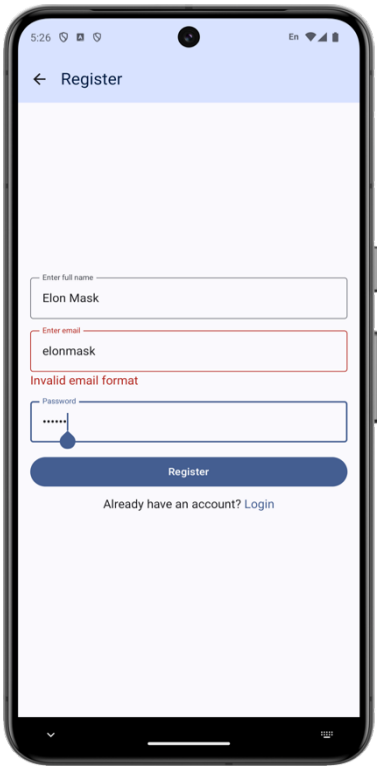
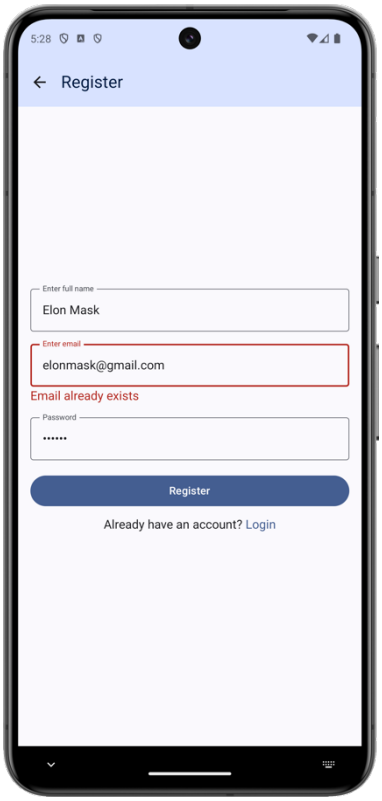


Figure 7.3

Figure 7.4

Error Message



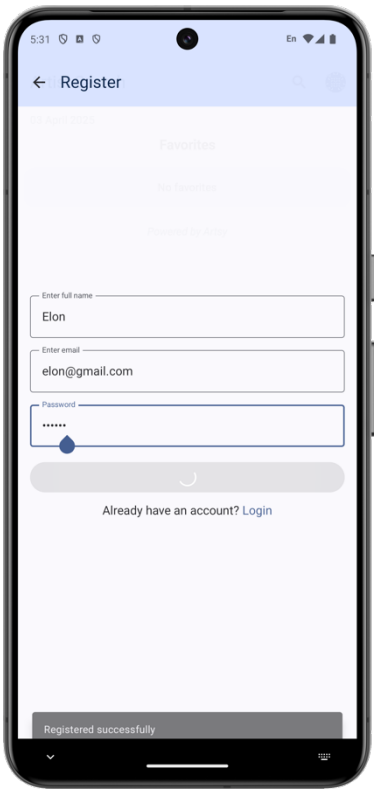


Figure 7.5

Register loading

& Snackbar

5.8 User Login

5.8.1 Persistent Login with Cookies and PersistentCookieJar

This app uses **OkHttp's CookieJar** (see **hints**) implementation to manage session cookies and ensure persistent login. The **PersistentCookieJar** class is responsible for storing cookies locally and reloading them when the app is reopened. Here's how it works:

Cookie Storage:

- Cookies are stored in **SharedPreferences** (see **hints**) using the **PersistentCookieJar** class.
- When cookies are saved, they are serialized into JSON format using **kotlinx.serialization** and stored in the **SharedPreferences**.

Loading Cookies:

- When the app is opened, the **PersistentCookieJar** loads cookies from **SharedPreferences** and deserializes them back into **Cookie** objects.
- Expired cookies are removed during this process.

Session Management:

- The **PersistentCookieJar** ensures that valid cookies are sent with every HTTP request, allowing the backend to recognize the user as logged in.
- If the session is still valid, the app fetches user data from the backend.

5.8.2 User Login Functionality

Login Process:

- When a user logs in, the app sends the email and password to the backend.
- If the login is successful:
    - The backend returns a session cookie, which is stored in the **PersistentCookieJar**.
    - The user's information (e.g., name, email, avatar) is fetched and stored in the app's state.
- If the login fails, an error message is displayed.

Persistent Login:

- When the app is reopened, the session is verified.
- If the session is valid, the user's information is fetched and stored in the app's state.

- If the session is invalid, the user is logged out automatically.

5.8.3 UI and Functional Features

Home Screen:

- Avatar in Top Action Bar:
    - Click the **Person Icon** to navigate to **Login Screen**. When successfully logged in and navigating back to the **Home** Screen, show a snackbar with message 'Logged in successfully'.
  - When logged in, the **Person Icon** in the top-right corner of the Home Screen is replaced with the user's **avatar**.
  - Clicking the avatar opens a **Dropdown Menu** (see **hints**) with the following options:
    - Log Out: Logs the user out and clears the session and show a snackbar with message 'Logged out successfully'.
    - Delete Account: Deletes the user's account and logs out, showing a snackbar with message 'Deleted user successfully'.

These functionalities are implemented as discussed in Assignment #3.

- Favorites Section:
  - When logged in, the **Favorites Section** is displayed on the **Home Screen**, showing the user's favorite artists.
  - This section dynamically updates based on the user's favorites stored in ***MongoDB Atlas***.

Artist Card View:

- Star Button (see **hints**):
  - When logged in, each artist card displays a **Star Button** in the top-right corner.
  - Clicking the button toggles the artist's favorite status (add/remove from the favorites list).
  - Logic:
    - If the artist is not in the favorites list, clicking the button will
      - Send a request to the backend to add the artist to the user's favorites in MongoDB Atlas.
      - Updates the local favorites.
      - Displays a Snackbar with the message 'Added to Favorites'.
    - If the artist is already in the favorites list, clicking the button will

- Send a request to the backend to remove the artist from the user's favorites in MongoDB Atlas.
- Updates the local favorites.
- Displays a Snackbar with the message ‘Removed from Favorites’.

Detailed Artist Information Screen:

- Star Icon in Top Action Bar:
  - When logged in, the top-right corner of the action bar displays a Star Icon Button (see **hints**).
  - Clicking the button toggles the artist's favorite status, similar to the Star Button in the Artist Card View.
- Similar Tab:
  - When logged in, the **Similar Tab** is available in the tabbed layout of the **Detailed Artist Information Screen**.
  - This tab displays a list of similar artists, as discussed in **Assignment 3**.

See reference video and **Figure 8.*.**

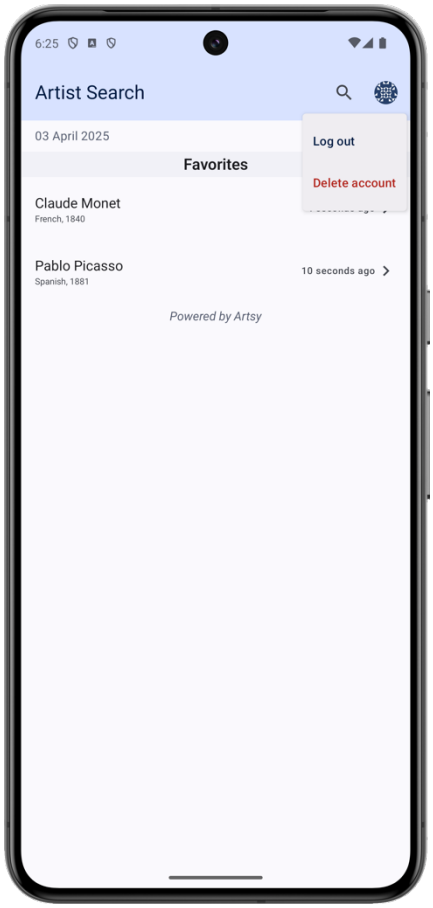


Figure 8.1

Home Screen (User logged in)

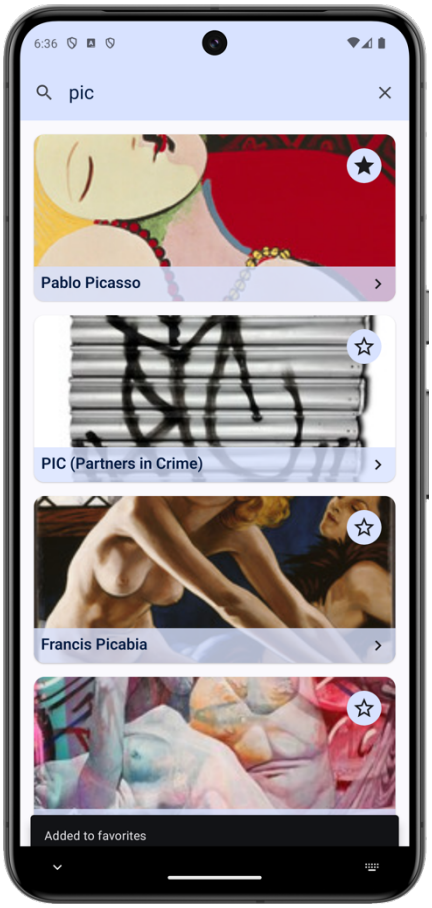


Figure 8.2

Toggle favorite

5.9 Error Handling

1. Make sure there are no conditions under which the app crashes
2. Make sure all icons and texts are correctly positioned as in the video/screenshots
3. Make sure the screens and snackbars are correctly displayed.
4. Make sure there is a progress bar when fetching data.
5. Make sure the styling for different features matches the video/screenshots.
6. When image from Artsy is missing, use the 'Artsy logo' instead.

5.10 Additional Info

For things not specified in the document, grading guidelines, the video, or in Piazza, you can make your own decisions. But keep in mind about the following points:

- Always display a proper message and do not crash if an error happens.
- All HTTP requests should be asynchronous and should not block the main UI thread. You can use third party libraries like Volley to achieve this in a simple manner.

6. IMPLEMENTATION HINTS

6.1 Icons

This project uses Material Icons from the **Jetpack Compose Material Design** library. You can refer to the following websites to search for additional icons

1. Material Icons Library: Official Material Design icons provided by Google
2. Jetpack Compose Material Icons Documentation: Documentation for using Material Icons in Jetpack Compose
    a.

Icon Description	**Icon name to search for**
Search Icon	Search
User Icon	Person
Right Arrow Icon	Keyboard Arrow Right

Close Icon	Close
Back Icon	Arrow Back
Details Tab Icon	Info
Artworks Tab Icon	Account Box
Similar Tab Icon	Person Search
Star Icon	Star / Star Border

The Artsy logo is at: https://commons.wikimedia.org/wiki/File:Artsy_logo.svg

6.2 Third party libraries

Sometimes using 3rd party libraries can make your implementation much easier and quicker. Some libraries you may have to use are:

6.2.1 Retrofit HTTP requests

Retrofit is used for making asynchronous HTTP requests to load data from the backend. Retrofit provides a clean and efficient way to handle API calls and parse responses.

OkHttp is used as the underlying HTTP client for Retrofit.

Kotlin Serialization Converter is used to serialize and deserialize JSON responses.

https://square.github.io/retrofit/

https://square.github.io/okhttp/

https://developer.android.com/codelabs/basic-android-kotlin-compose-getting-data-internet#0

6.2.2 Coil (Compose Image loading library)

Coil is a powerful image loading and caching library for Android.

https://coil-kt.github.io/coil/

6.3 Working with action bars and menus

Both **App Bar** and **Dropdown Menu** are part of the **Material Design 3** system, which provides modern UI components for Android apps.

https://developer.android.com/develop/ui/compose/components/app-bars

https://developer.android.com/develop/ui/compose/components/menu

6.4 Displaying Progress Bars

CircularProgressIndicator is part of the **Material 3** library.

https://developer.android.com/reference/kotlin/androidx/wear/protolayout/material/CircularProgressIndicator

6.5 Implementing Splash Screen

There are many ways to implement a splash screen. This blog highlights almost all of them with examples:

https://android.jlelse.eu/the-complete-android-splash-screen-guide-c7db82bce565

6.6 Adding the App Icon

https://dev.to/sfarias051/how-to-create-adaptive-icons-for-android-using-android-studio-459h

https://icon.kitchen/

6.7 Handle User Input

https://composables.com/material3/outlinedtextfield

https://developer.android.com/develop/ui/compose/text/user-input

6.8 Adding a button to ActionBar

https://developer.android.com/training/appbar/actions

https://stackoverflow.com/questions/12070744/add-back-button-to-action-bar

https://stackoverflow.com/questions/34110565/how-to-add-back-button-on-actionbar-in-android-studio

6.9 Implementing a LazyColumn in android

https://github.com/android/compose-samples

https://developer.android.com/develop/ui/compose/lists

https://developer.android.com/develop/ui/compose/documentation

6.10 Adding Snackbar

https://developer.android.com/reference/kotlin/androidx/compose/material/SnackbarHostState

6.11 Card View

https://developer.android.com/develop/ui/compose/components/card

6.12 Hiding User Input

https://developer.android.com/reference/kotlin/androidx/compose/ui/text/input/PasswordVisualTransformation

6.13 Open Link in browser

https://www.tutorialkart.com/kotlin-android/android-open-url-in-browser-activity/

6.14 Back press behavior on Back button

https://stackoverflow.com/a/27807976

6.15 Search Bar

To implement the search functionality, these pages will help:

https://developer.android.com/develop/ui/compose/components/search-bar

https://www.youtube.com/watch?v=9OWmnYPX1uc

https://developer.android.com/guide/topics/search/search-dialog

6.16 CookieJar

https://square.github.io/okhttp/3.x/okhttp/okhttp3/CookieJar.html

https://github.com/franmontiel/PersistentCookieJar

https://developer.android.com/reference/android/content/SharedPreferences

6.17 Implementing Dialogs

https://developer.android.com/develop/ui/compose/components/dialog

https://mkyong.com/android/android-custom-dialog-example/

https://androidexample.com/Custom_Dialog_-_Android_Example/index.php?view=article_discription&aid=88&aaid=111

https://medium.com/@suragch/creating-a-custom-alertdialog-bae919d2e

6.18 Implementing Carousel Loop in Android

https://iamjegul.medium.com/the-guide-to-infinite-loop-carousel-with-recyclerview-53b7986ab547

https://developer.android.com/develop/ui/views/animations/motionlayout/carousel

6.19 Implementing Dark Theme in Android

https://developer.android.com/develop/ui/views/theming/darktheme

7. FILES TO SUBMIT

You should Clean the folder of your object files, and then ZIP all your source code, from the top Android folder, and submit the resulting ZIP file.

You will have to submit a video of your assignment demo, which will be graded. Details for the script to use and how to create the video, are separately documented in D2L Brightspace.

The reference vide is at: https://www.youtube.com/watch?v=Si0QBggemRA

****IMPORTANT****

All videos are part of the Assignment description. All discussions and explanations on Piazza related to this Assignment are part of the Assignment description and will be accounted into grading. So please review all Piazza threads before finishing the assignment.