

Victoria University

Bachelor of Information Technology

NIT3213

Goraksha Pratap Shahi (s8121610)

Android Application Development

Final Android App Assignment

Project Title: NIT3213App – Android App with
API Integration and Unit Testing

Table of Contents

1. Introduction
2. App Structure and Features
3. ViewModel, Repository, and API Integration
4. Testing (Unit Test + Login Test)
5. GitHub Integration
6. Challenges and Resolutions
7. Conclusion
8. References

1) Introduction

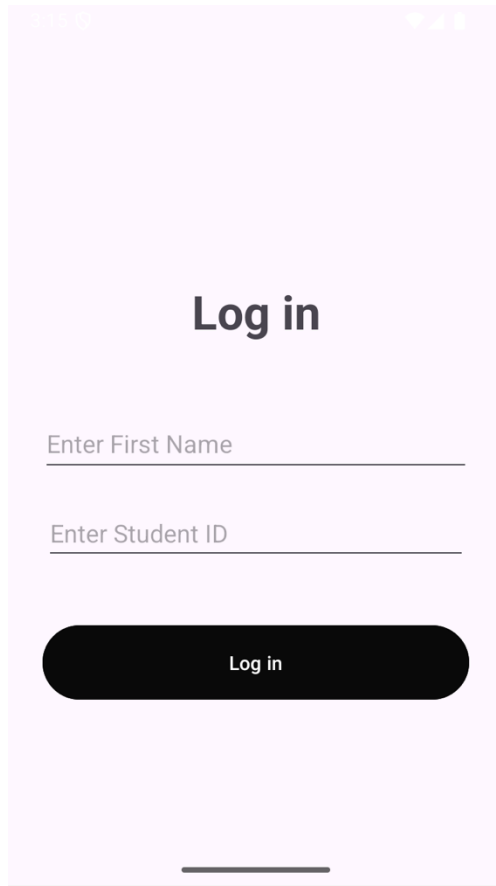
This Android application was created as part of the final assessment for the NIT3213 unit. The goal was to build a simple yet functional mobile app that demonstrates key skills in Android development, including working with APIs, displaying dynamic data, and writing unit tests.

The app includes three main screens: a Login screen, a Dashboard that loads data from an external API, and a Details screen to show more information about selected items. Technologies like Retrofit were used for API communication, and Hilt was used to manage dependencies efficiently.

Throughout the project, I focused on keeping the code clean and well-organized while following good software development practices. The entire project is also connected to GitHub for version control. This assignment helped me apply what I've learned in class to a real Android app, from designing the features to testing and debugging the code.

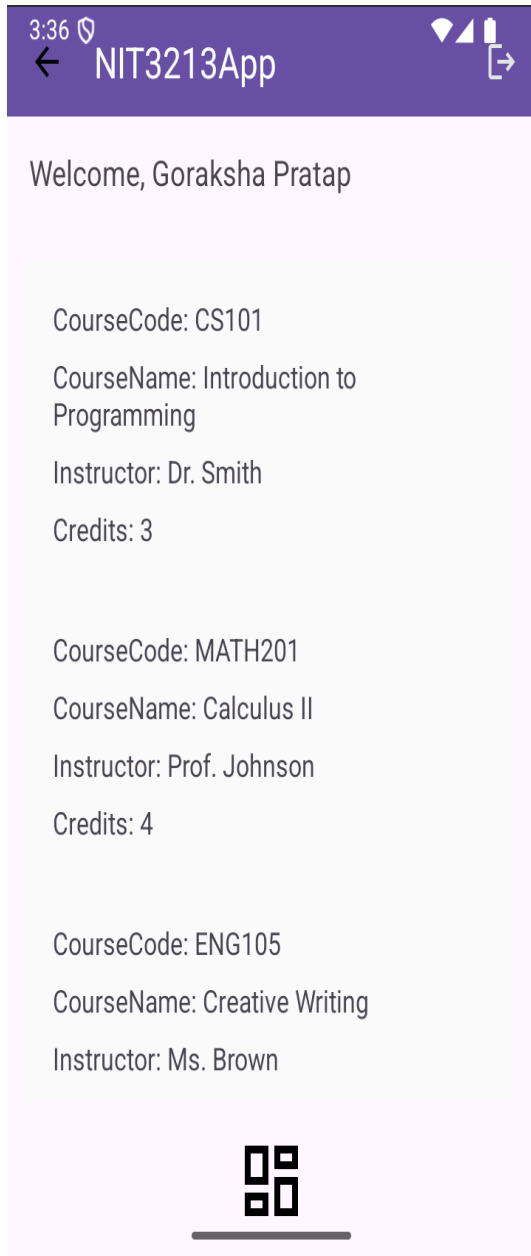
2) App Structure and Features:

Login Screen:

A mobile app login screen with a light pink background. At the top right, there are three small circular icons: a heart, a square, and a circle. In the center, the text "Log in" is displayed in a bold, dark font. Below this, there are two input fields. The first is labeled "Enter First Name" and the second is labeled "Enter Student ID", both in a light gray font. Below the input fields is a large, black, rounded rectangular button with the text "Log in" in white. At the bottom of the screen, there is a thin horizontal line representing the home indicator bar.

The login screen allows users to enter their credentials, which are sent to the server for verification. Upon successful login, the API returns a keypass, which is used to load user-specific data on the Dashboard.

Dashboard Screen:



This screen displays a dynamic list of items using RecyclerView. It fetches data from an API endpoint based on the keypass, allowing each user to see different content (such as books, courses, or subjects).

Logout Button:

On the Dashboard screen, there's a back/logout button that takes the user back to the Login screen. This helps users exit their session safely.

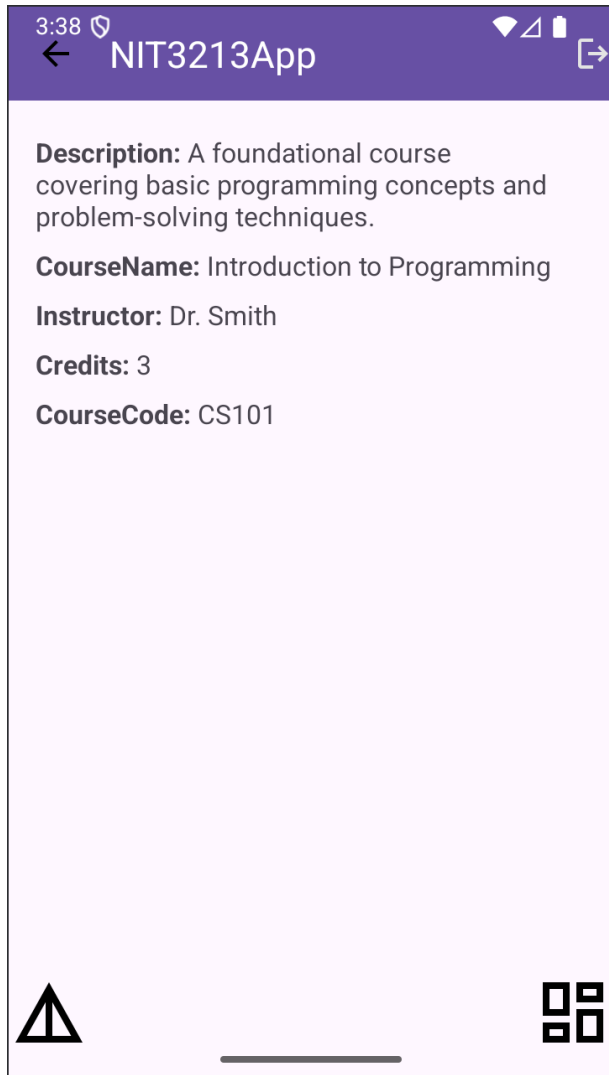
Dashboard Navigation:

After logging in, users are automatically taken to the Dashboard where all items are displayed. The navigation is handled using Intents.

Back Button:

The user can return to the login screen by using the back button on the top bar.

Details Screen:



Tapping an item on the Dashboard takes the user to a screen with detailed information about that item.

Dashboard Logo Functionality:

The Details screen has a Dashboard icon at the bottom. When tapped, it quickly takes the user back to the Dashboard. This gives users a fast and easy way to return without using the back button.

3. ViewModel, Repository, and API Integration:

This app follows a clean and simple structure using the MVVM (Model-View-ViewModel) pattern. The main goal here was to separate the UI logic from the data logic, making the code easier to manage and test.

ViewModel:

The DashboardViewModel handles the app's logic for retrieving and storing dashboard data. It uses LiveData to keep the UI updated when new data is available or when an error occurs.

Repository:

The DashboardRepository acts as a middle layer between the ViewModel and the API. It takes care of making network requests and deciding what data should be passed to the ViewModel. This helps keep the ViewModel clean and focused on logic rather than data fetching.

API Integration:

Retrofit is used to send and receive data from the API. When the user logs in, the app sends the username and password to the /auth endpoint and receives a keypass. This key is then used to fetch dashboard data from another endpoint.

Using this approach made the app easier to build and scale, and it also helped make parts of the app reusable and testable.

4. Testing

Testing was a key part of this project to make sure that the app works as expected and handles data properly. I wrote a few simple unit tests focusing on critical components like API parsing and model behavior.

What Was Tested:

LoginRequest Test:

This test checks that the LoginRequest data class correctly holds the username and password entered by the user.

DashboardItem Parsing Test

This test ensures that the app can correctly read and convert a JSON response into a DashboardItem object.

ApiService Test

This test uses a mock response to check if the app handles API data as expected when requesting dashboard items.

All tests were placed in the `src/test/java/...` folder and can be run directly from Android Studio. These tests helped me catch any early bugs and gave me confidence that the app would behave reliably.

5. GitHub Integration:

To manage the project version and keep track of changes, I used Git and pushed the code to a GitHub repository. This helped me back up my work, switch between versions, and collaborate more easily if needed.

<https://github.com/DrKarmafr/NIT3213App>

Git Features Used:

- Created a new repository on GitHub
- Initialized Git in Android Studio
- Committed changes with meaningful commit messages
- Pushed the entire project to GitHub
- Regularly committed changes to track progress

Using Git helped me stay organized throughout the project and allowed for easy submission and sharing with others.

6. Challenges and Resolutions:

While building the app, I faced a few technical and configuration challenges. Each one taught me something new and helped improve the project overall.

Challenge 1: Dependency Injection with Hilt:

Issue:

Setting up Hilt correctly was tricky, especially when updating Gradle files and using the right annotations.

Solution:

I followed official documentation step-by-step and made sure to update both build.gradle files and annotate the Application class properly with `@HiltAndroidApp`.

Challenge 2: Parsing Dynamic API Data:

Issue:

The API response changed depending on the user's login credentials (keypass), which meant I needed to build a generic structure to handle it.

Solution:

I created a flexible `DashboardItem` model with optional fields and used Gson to parse whatever structure the API returned.

Challenge 4: Unit Test Errors:

Issue:

Some unit tests failed due to unresolved references and missing imports.

Solution:

I restructured the test files, double-checked dependencies, and removed unnecessary test classes that caused build issues.

These challenges helped strengthen my understanding of real-world Android development.

7. Conclusion:

This assignment gave me a great opportunity to apply what I've learned in Android development. From building the UI and connecting to an API, to writing unit tests and using GitHub, I was able to experience the full process of creating a working app.

The project helped me understand how important good structure is — especially when using patterns like MVVM and tools like Hilt for dependency injection. It also gave me confidence in handling real-time data, writing clean code, and troubleshooting when things went wrong.

Overall, I'm proud of what I've built and the challenges I overcame. This app shows my progress as a developer and gives me a solid foundation to build on in future projects