Swift Modular App - Coding Assignment

Objective:

The goal of this assignment is to develop a **modular, native Swift app** that connects to the **GitHub API** and displays public repositories of a specific user. The app should be **cleanly structured** using one of the following architectures:

- **Clean Swift (VIP)**
- **MVVM or MVVM-C**
- **VIPER**

The app should support **iOS 13 and later** and must be submitted as a **.zip file** upon completion along with README file with Instructions.

Kindly do not submit incomplete assignment. It will be rejected for review

Task 1: Modular Architecture & GitHub API Integration

Requirements:

1. **Modular Architecture**
    - Implement **separate frameworks** to organize different functionalities (e.g., Networking, Data, UI, and Features).
2. **GitHub API Integration**
    - Fetch public repositories from the GitHub API.
    - You may use your own account or this sample endpoint: **https://api.github.com/users/mralexgray/repos**
3. **TableView with Lazy Loading**
    - Display repositories in a **UITableView**.
    - Implement **lazy loading** (background API calls) to fetch repository details efficiently.
    - Only **fetch and display the latest 3 commits** for each repository **after the cell is expanded**.
4. **CollectionView Inside TableView (Circular Scrolling)**
    - The first **5 repositories** should be shown inside a **horizontally scrollable CollectionView** (inside TableView).
    - The CollectionView should have **circular scrolling**:
        - **Scrolling left from the first cell** should navigate to the **last cell**.
        - **Scrolling right from the last cell** should loop back to the **first cell**.
    - **No third-party libraries** should be used for circular scrolling.

5. **Commit Data Fetching via Lazy Loading**

    - **Do not fetch all commit data at once**. Implement **background API calls** to fetch commits of only **visible cells** and show latest **3 commits per repository if available**

6. **Detail Screen Overlay**

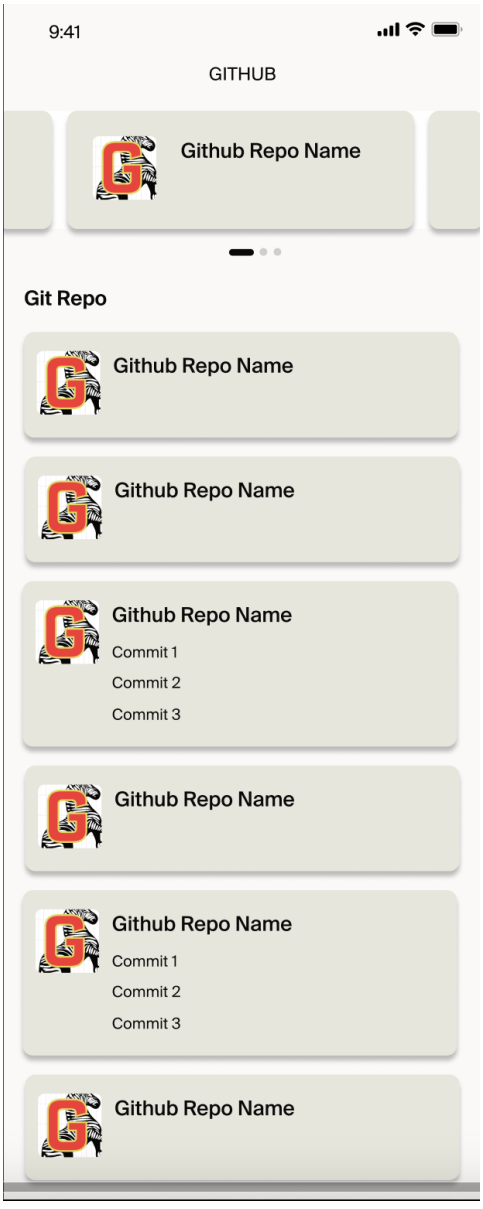
    - Implement a **detail screen overlay** (instead of navigating to a new screen).
    - The detail screen should open with a **drag gesture** and have a **close button**.

Figma Wireframe for Reference

- Use the **provided Figma design** to align the UI:
https://www.figma.com/proto/UDfGri4wc6qesEFhY3pDGi/Untitled?node-id=1-96&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=1%3A204

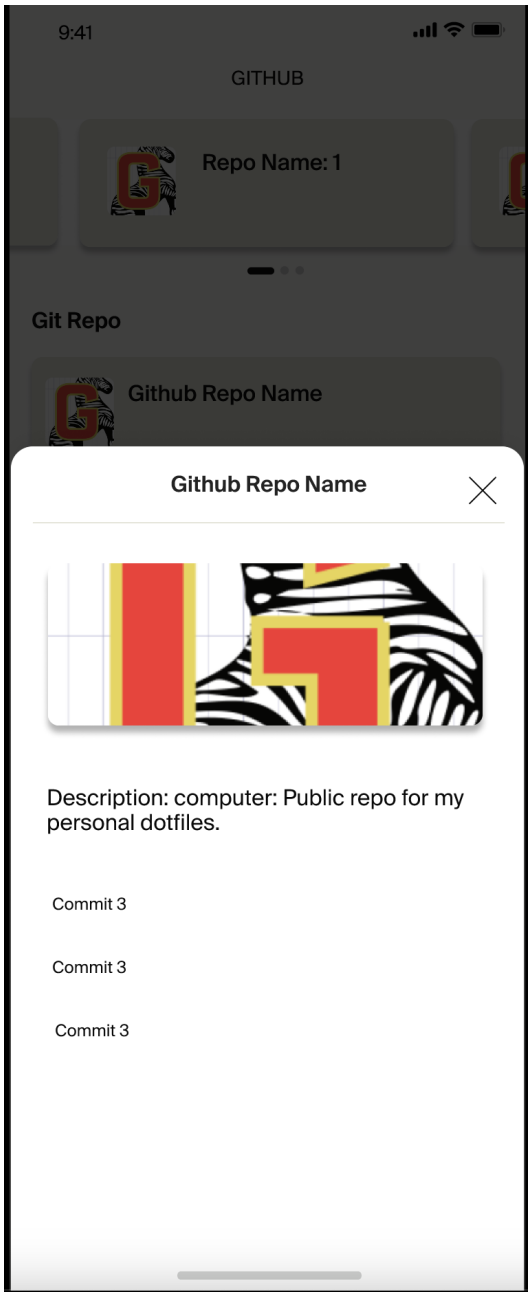
SCREEN 1:

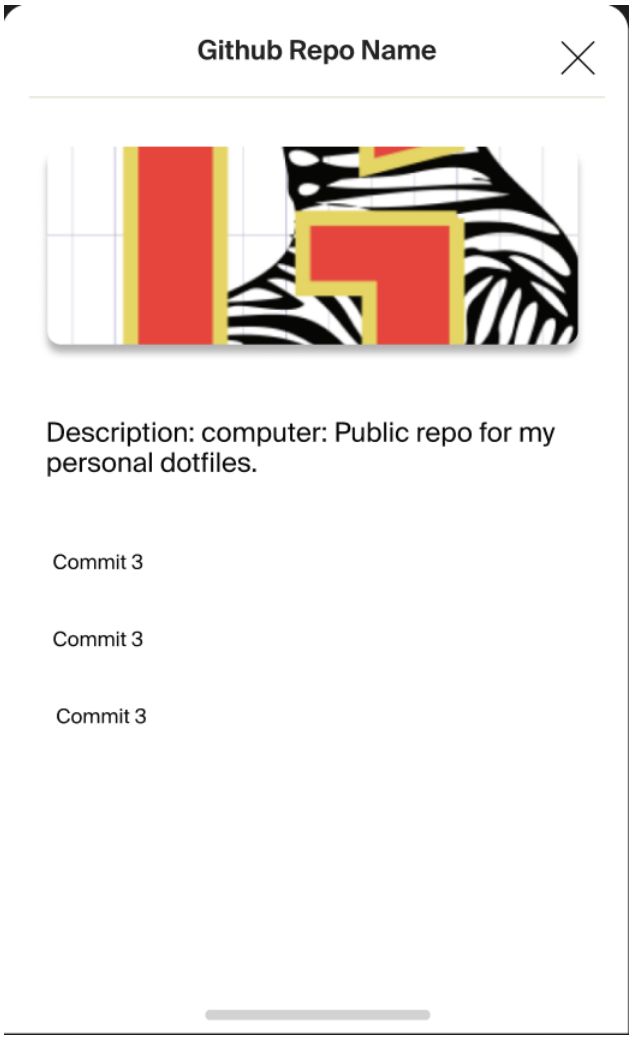
Scenario1:	*Scenario2:*
Repo fetched without commits	*After Commits Fetched with Lazy/Async API call. Expand the same cell once you get the commit list and show latest 3 commits*



SCREEN 2:

Detail Screen: Open the screen as overlay on the same list screen with drag gesture or close button to close the overlay





Task 2: Data Persistence with Core Data

Requirements:

1. **Implement a Core Data Persistence Layer**
    - Store repository data locally using **Core Data**.
2. **Modify Data Flow for Offline Mode**
    - **When the app starts:**
        - **First, load data from Core Data.**
        - Then, **fetch fresh data from GitHub API** and update the local storage.
    - **On app relaunch:**
        - **Display stored data first.**
        - Then, request fresh data from GitHub and update the UI accordingly.

What We Care About:

✅ **OOP (Object-Oriented) & POP (Protocol-Oriented) principles**

✅ **Clean Swift (VIP), MVVM-C, or VIPER architecture**

✅ **Unit Tests & Code Coverage (mandatory)**

✅ **No Auto Layout constraint warnings or code warnings**

✅ **Plus Point:** If you use the **Combine framework** for data handling