

Project Milestone #2: Working UI with Dummy Data

Re-submit Assignment

Due Feb 21 by 11:59pm **Points** 65 **Submitting** a file upload

For this milestone you will be creating an Android user interface for your application. *This is to be the front-end of your application only* so do not make any requests to backend AWS services. Instead your front-end application should have realistic dummy data that will later (ie, in later milestones) be replaced with calls to such backend services.

Your user interface should include all views needed to meet the requirements documented in the "Requirements" section of the [Course Project](#) overview. After milestone 2, only minor refactoring and beautification would be needed for the front-end of your project. The deliverable at this stage requires a functioning front-end for your project, not just a wireframe or mockup.


Design

Your design should include:

- A **view** layer
- A **presenter** layer
- A **model** layer that includes *entity* and *application logic* classes. Entity classes represent the basic data items presented in your user interface. Application logic classes implement the client's functionality, such as logging in and sending statuses.
- A **lower** layer with one or more proxy (or facade) classes *that* handle calls to the backend services. For this milestone your proxy classes will simply return dummy data.
- AsyncTasks to avoid accessing network services or other slow running processes on the main UI thread.

You are to use the **observer pattern** to communicate from lower to higher level layers in your application.

Sample Code

A sample project that includes a partially functional application and demonstrates an architecture that satisfies the above requirements is available on github at [this URL](https://github.com/jerodw/tweeter-1) [.\(https://github.com/jerodw/tweeter-1\)](https://github.com/jerodw/tweeter-1). You are welcome to use this as a starter project for your own solution or to copy some or all of the code into your own project. A package diagram showing the architecture of the sample solution is available [here](#) .

Testing

At a minimum, you should include unit tests for all of your presenter classes.

Submission:

- Pass off your project with a TA by the due date at 5pm
- Submit to Canvas a zip file containing your project in its current form
- Submit to Canvas a PDF file containing your milestone report which includes the following:
 - A UML class diagram demonstrating your program's **model** layer. Typical *entity* classes would be things like: User, Status, Feed, Follower, etc. Put these classes in your diagram, including major relationships, attributes, and operations. By "major" we mean only include details that are important for helping the reader understand the overall structure of your design, and omit details that don't help very much and just clutter the diagram. This layer will also have some "application logic" classes that will eventually implement the client's functionality, such as logging in and sending statuses.
 - A UML class diagram demonstrating your program's **user interface** layer. The diagram should include major attributes, operations, and relationships on each class. This class diagram will consist mostly of *view* classes and *presenter* classes, and their major relationships, attributes, and operations. For example, what methods do your "views" call on your "presenters", and vice versa? What are the major attributes on your "view" and "view model" classes? Etc. Include enough details to give the reader an understanding of your design's overall structure, but do not include every detail. Including too much detail clutters the diagram and harms its readability. If we want more details, we can always look at your source code. Your user interface layer might also contain some support classes that help implement your user interface. For example, you might have a class named NavigationManager that implements navigation between the different views. If you have such classes, include them in your class diagram.
 - A UML sequence diagram demonstrating what happens when a user logs in.

Rubric

- [50 points] The spec defines 10 features you are to implement. Each feature will be graded out of 5 for a total of 50 feature points, as follows:
 - 1 point for use of Model-View-Presenter (MVP)
 - 1 point for use of AsyncTasks and using observer pattern to return data
 - 1 point for use of facade pattern (to access "backend" dummy data)
 - 1 point for correct functionality
 - 1 point for unit tests for presenter
- [15 points] For UML diagrams
 - 5 points for UML class diagram describing model classes

- 5 points for UML class diagram describing user interface
- 5 points for UML sequence diagram

The rubric below in the grades is just for TA convenience when grading. You should look at this one.

Milestone #2 Rubric (Just for TA convenience!)

Criteria	Ratings		Pts
Sign Up Feature - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Sign In Feature - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Sign Out Feature - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Post Status Feature - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
View Feed Feature - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
View User Story Feature - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts

Criteria	Ratings		Pts
View User Followers Feature <ul style="list-style-type: none"> - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests 	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
View User Following Feature <ul style="list-style-type: none"> - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests 	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Follow Feature <ul style="list-style-type: none"> - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests 	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Unfollow Feature <ul style="list-style-type: none"> - Uses MVP pattern - Uses AsyncTasks - Uses Facade pattern - Is functional - Has Unit Tests 	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
UML Class Diagram Describing Model Classes	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
UML Class Diagram Describing User Interface	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
UML Sequence Diagram For User Login	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts

Criteria	Ratings		Pts
Late Points -10% per day	0.0 pts Full Marks	0.0 pts No Marks	0.0 pts
Total Points: 65.0			