IS442

Object Oriented Programming

AY2019 – 2020, Term 2

Documentation

Section:	G1 – T10
Instructor:	Professor Lee Yeow Leong
Student Names and ID:	Chua Shao Shxuan Low Louis See Shao Jie, Glen Tan Yong Fan, Bryce

Table Of Content

Design Principles/Patterns:

In designing our project, we utilised the SOLID principle to help us. The SOLID principle is a combination of 5 basic designing principles,

1. Single Responsibility Principle (SRP)

Every of our class only had a single purpose. An example will be Wall.java is only responsible for the user's wall and nothing else.

2. Open/Closed Principle (OCP)

This principle states that software entities should be open for extension but closed for modification. What this means is that once a base class has been developed and tested (closed), the codes should only be changed to correct bugs. The only way that you are able to introduce new functionality would be to extend it and modify it from there.

3. Liscov Substitution Principle (LSP)

This principle states that functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

4. Interface Segregation Principle (ISP)

This principle states that Clients should not be forced to depend on interfaces which they do not use, and thus the number of members in the interface should be minimised.

5. Dependency Inversion Principle (DIP)

This principle states that 1) High Level modules should not depend upon low level modules and 2) Abstractions should not depend upon details. These helps us to develop loosely coupled codes by ensuring that high level modules depend on abstractions rather than concrete implementation of lower level modules.

Additionally, we also used the “Don’t Repeat Yourself (DRY)” principles in order to help us write scalable, maintainable and reusable codes. We achieved this by utilising the MVC architecture which we will elaborate below.

The last principle we also used is the “Keep it simple, Stupid (KISS)” principle whereby we tried to keep each tried to avoid unnecessary complexities by writing simpler and shorter codes.

One of the frameworks that we used was the Model-View-Controller framework in our application. It separates the application into three main logical components and each component is built to handle a specific development aspect of an application. The user will interact with a view and the view then alerts the controller of a particular event. Once this happens, the controller updates the model and the model would then update the view that the users see. In our project, the view are pages such as viewThread.java that just specifically handles the printing of information to the user, while the controller are pages like postDAO.java and responseDAO.java. The models are pages like Wall.java.

We utilised the MVC framework as it made code maintenance easy to extend and grow and each MVC Model component could be developed parallelly saving us a lot of time. It also offered us the best support for a test-driven environment. However, there also were setbacks from us using the MVC framework, which was that it was difficult for us to unit test the application. We also discovered that there were added complexities behind the application which might slow down the speed if the application was scaled up and also the inefficiency of data.

In conclusion, we only utilised half of the SOLID principles as we did not utilise inheritance in our project. In conjunction with half of the SOLID principles, we also heavily relied on the KISS and DRY principles as it helped us tremendously by keeping things simple and manageable.

Various approaches to tackle design:

As we studied the design of the application, we first came up with a Use Case model on the whiteboard to understand the actions that were occurring and who it came from. We classified each entity as a class and thought of the various data attributes it requires. We realised that there were a lot of cases whereby the CRUD of data was necessary a CSV file would be insufficient to handle all of it. As such, we decided on using a database to store our data and make it more convenient to manipulate the data.

While looking through the project requirements, we realised that aside from storing the data of each entity, there was a need to store data that connects between two different entities to be able to fulfill certain requirements (e.g. Keeping track of a user sending a gift to their friend). As such, we had to create additional tables to keep track of such.

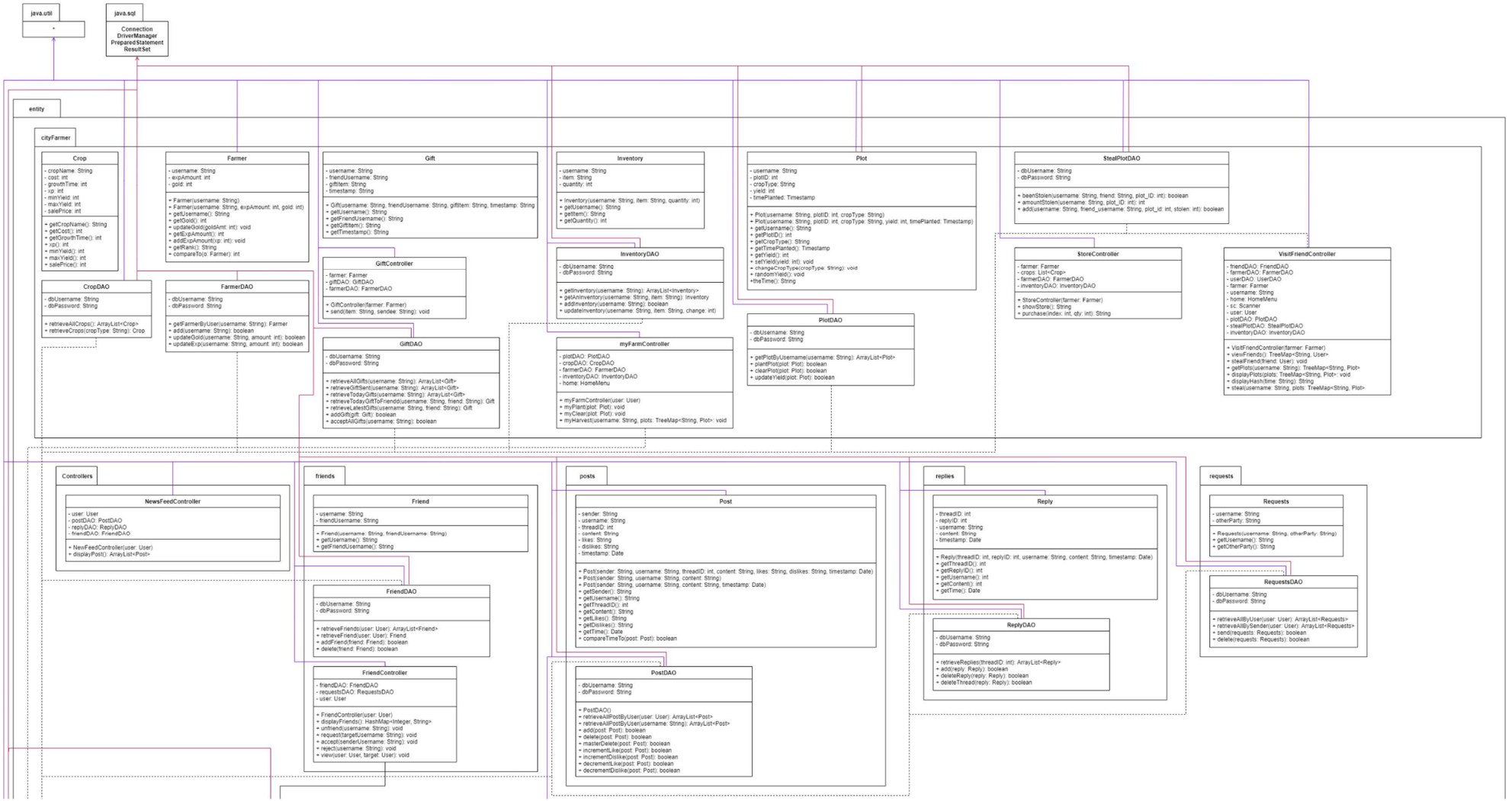
One approach we decided to make use of was the implementation of triggers within our database to handle some of the functionalities. By having triggers to help with some of the implementation, it reduced the complexity of our code and made it simpler and cleaner to view and understand.

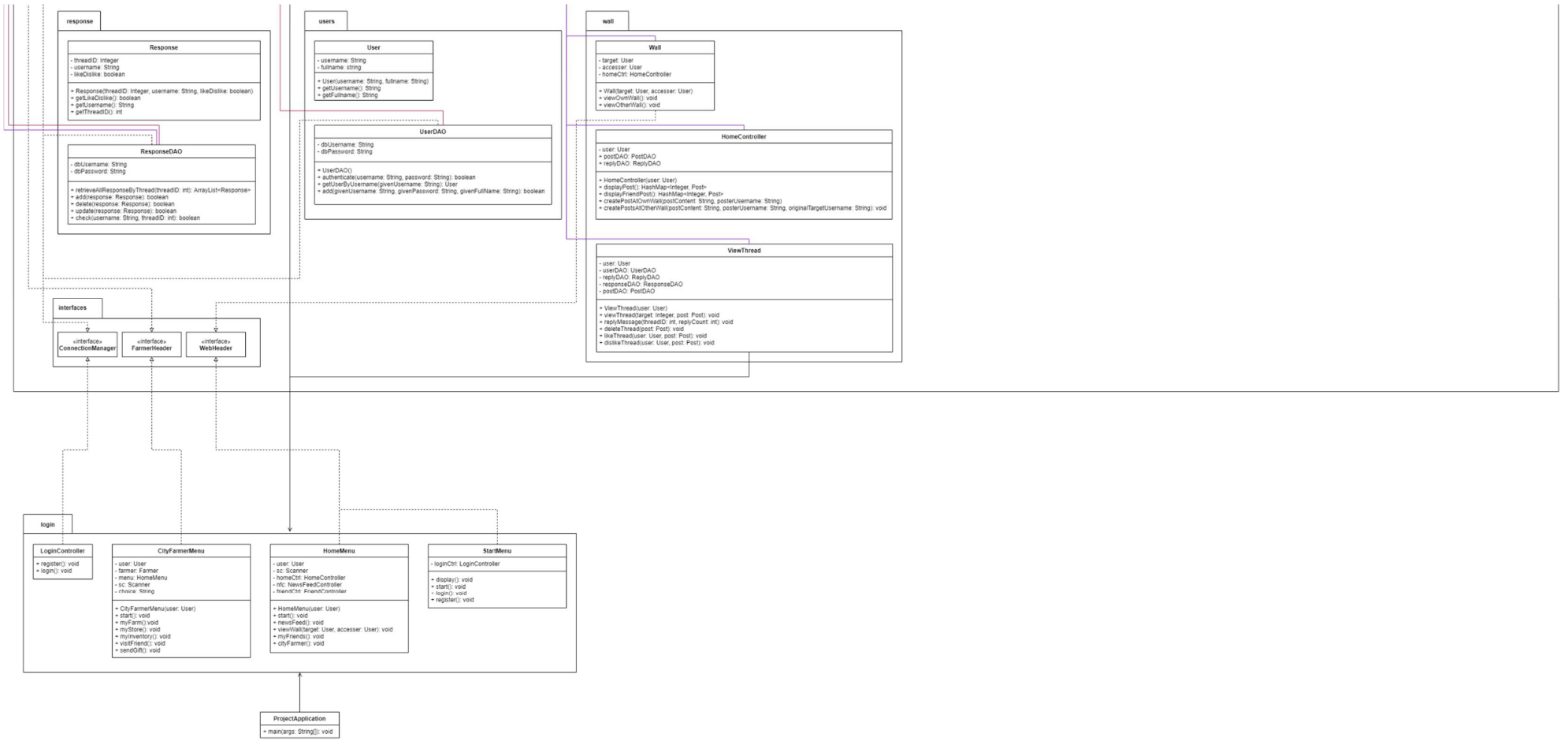
After having a grasp of the big picture, we realised that we could separate the application into two main parts, Social Magnet and Cityfarmer. With that, we decided on working solely on Social Magnet first since they are almost distinctively two separate systems. We did not tackle both problems at the same time as it would have been too overarching and daunting for us to do so, we wanted to start small and slowly chip at the project one step at a time.

From the appendix of the project requirements, we realise that there would be a decent chunk of repetition and as such we created functions and even interfaces to aid us in reducing the amount of repetition and in turn the amount of extra unnecessary work needed to complete the project.

As our team developed the code together, we created comments for each other to help each other to be capable of understanding our code better. It improved the efficiency of completing the project especially in light of COVID-19 in the past few weeks.

Design Class Diagram




response
Response
- threadID: Integer
- username: String
- likeDislike: boolean
+ Response(threadID: Integer, username: String, likeDislike: boolean)
+ getLikeDislike(): boolean
+ getUsername(): String
+ getThreadID(): int
ResponseDAO
- dbUsername: String
- dbPassword: String
+ retrieveAllResponseByThread(threadID: int): ArrayList<Response>
+ add(response: Response): boolean
+ delete(response: Response): boolean
+ update(response: Response): boolean
+ check(username: String, threadID: int): boolean
users
User
- username: String
- fullname: string
+ User(username: String, fullname: String)
+ getUsername(): String
+ getFullname(): String
UserDAO
- dbUsername: String
- dbPassword: String
+ UserDAO()
+ authenticate(username: String, password: String): boolean
+ getUserByUsername(givenUsername: String): User
+ add(givenUsername: String, givenPassword: String, givenFullName: String): boolean
wall
Wall
- target: User
- accesser: User
- homeCtrl: HomeController
+ Wall(target: User, accesser: User)
+ viewOwnWall(): void
+ viewOtherWall(): void
HomeController
- user: User
+ postDAO: PostDAO
+ replyDAO: ReplyDAO
+ HomeController(user: User)
+ displayPost(): HashMap<Integer, Post>
+ displayFriendPost(): HashMap<Integer, Post>
+ createPostAtOwnWall(postContent: String, posterUsername: String)
+ createPostsAtOtherWall(postContent: String, posterUsername: String, originalTargetUsername: String): void
ViewThread
- user: User
- userDAO: UserDAO
- replyDAO: ReplyDAO
- responseDAO: ResponseDAO
- postDAO: PostDAO
+ ViewThread(user: User)
+ viewThread(target: Integer, post: Post): void
+ replyMessage(threadID: int, replyCount: int): void
+ deleteThread(post: Post): void
+ likeThread(user: User, post: Post): void
+ dislikeThread(user: User, post: Post): void
interfaces
«interface» ConnectionManager
«interface» FarmerHeader
«interface» WebHeader
login
LoginController
+ register(): void
+ login(): void
CityFarmerMenu
- user: User
- farmer: Farmer
- menu: HomeMenu
- sc: Scanner
- choice: String
+ CityFarmerMenu(user: User)
+ start(): void
+ myFarm(): void
+ myStore(): void
+ myInventory(): void
+ visitFriend(): void
+ sendGift(): void
HomeMenu
- user: User
- sc: Scanner
- homeCtrl: HomeController
- nfc: NewsFeedController
- friendCtrl: FriendController
+ HomeMenu(user: User)
+ start(): void
+ newsFeed(): void
+ viewWall(target: User, accesser: User): void
+ myFriends(): void
+ cityFarmer(): void
StartMenu
- loginCtrl: LoginController
+ display(): void
+ start(): void
+ login(): void
+ register(): void
ProjectApplication
+ main(args: String[]): void

Logical Diagram