CU Connect

Final Report

Team Number: 41

Team Members:

1. Richard Noronha
2. Jayakrishnan HJ

Vision and CUConnect is an online portal for CU students and past pupils to connect with each other. It is a social media prototype for such a portal.

Actors: User, Admin

Description: In this portal, the users can connect and share content such as status. The other users will have the ability to view the status. The backend of the system will be implemented using Java. We have tried to be as detailed as we can foresee as of now. As and when modifications are needed, the documents and diagrams will be updated. A database will be implemented too

List the features implemented:

1. **Project Requirements:**

User Requirements			
ID	**DESCRIPTION**	**USER**	**PRIORITY**
UR-01	As a new user I should be able to register and create an account	User	Low
UR-02	As an existing user I should be able to login to the system	All actors	Low
UR-05	As an existing user I should be able to add content to my account	All actors	Low
UR-07	As an existing user I should be able to search user based on filters	All actors	Low
UR-08	As an existing user I should be able to add other users as friends	User	Low
UR-09	All actors should be able to delete their accounts	All actors	Low
UR-10	As an existing user I should be able to delete another user's account	Admin	Low
UR-11	As an existing user I should be able to view another user's full content	User	Low
UR-13	As a user, I should be able to edit my content	User	Low

ID	DESCRIPTION	TOPIC AREA
BR-01	Project must be implemented in Java using Object Oriented Design practices	Implementation
BR-02	It should be a standalone application	Implementation

BR-03	It should have a central repository in Github and it should be versioned controlled through git	Implementation
BR-04	The code should be refactored to follow industry recognized design patterns	Implementation

Functional Requirements			
ID	**DESCRIPTION**	**TOPIC AREA**	**PRIORITY**
FR-01	Authentication of all users is performed using login and password	Authentication	Low

Non-Functional Requirements			
ID	**DESCRIPTION**	**USER**	**PRIORITY**
NFR-02	The system should be available all day	Reliability	High
NFR-03	The system should be able to support n users simultaneously	Performance	High

List Features not implemented:

1. **Project Requirements:**

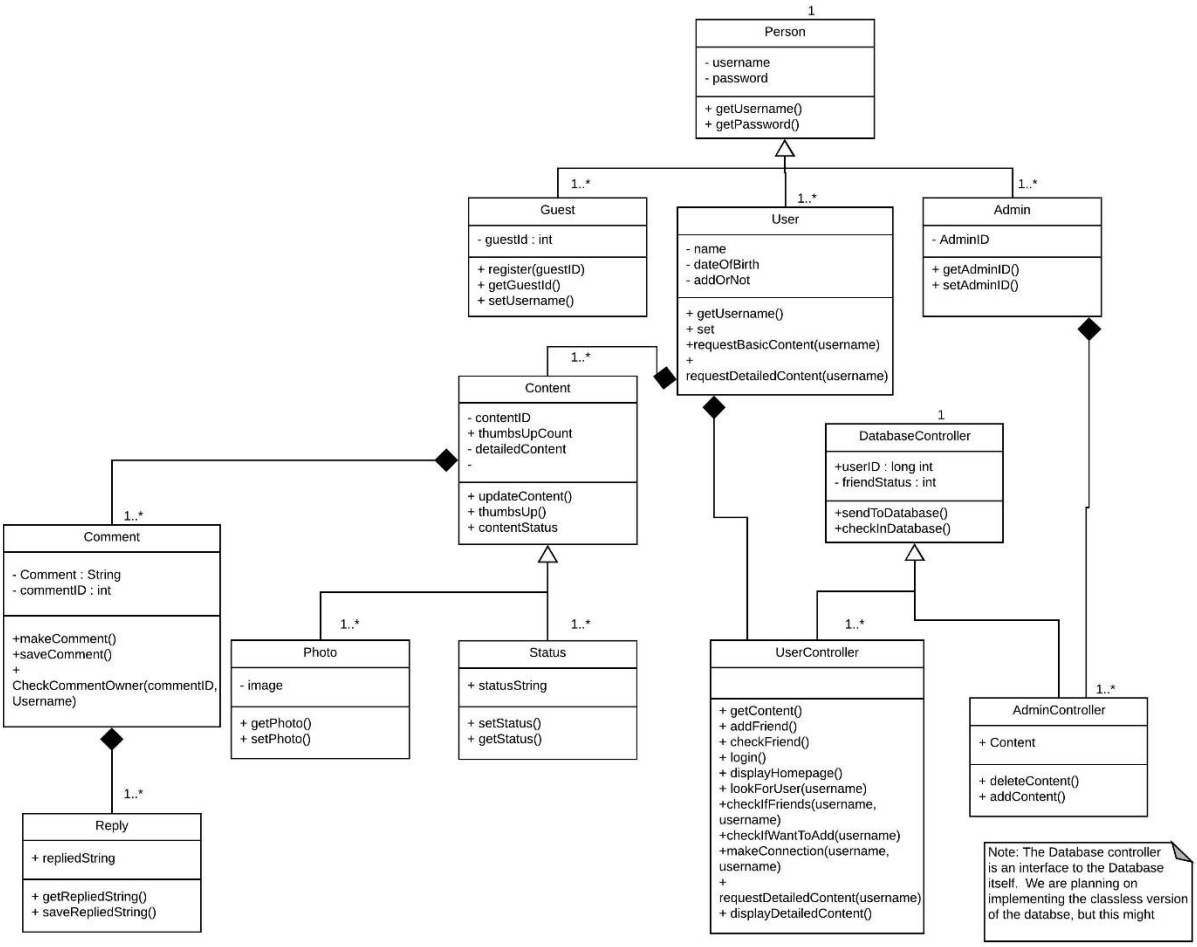
User Requirements			
ID	**DESCRIPTION**	**USER**	**PRIORITY**
UR-03	As an existing user I should be able to add comments on other user's profile contents	All actors	Low
UR-04	As an existing user who is an admin, I should be able to review and delete objectionable comments	Admin	Low
UR-06	As an existing user as an admin I should be able to delete any content of another user	Admin	Low
UR-12	As an existing user I should be able to thumbs up/down to another friend's content	User	Low

ID	DESCRIPTION	TOPIC AREA
BR-01	Project must be implemented in Java using Object Oriented Design practices	Implementation
BR-02	It should be a standalone application	Implementation
BR-03	It should have a central repository in Github and it should be versioned controlled through git	Implementation
BR-04	The code should be refactored to follow industry recognized design patterns	Implementation

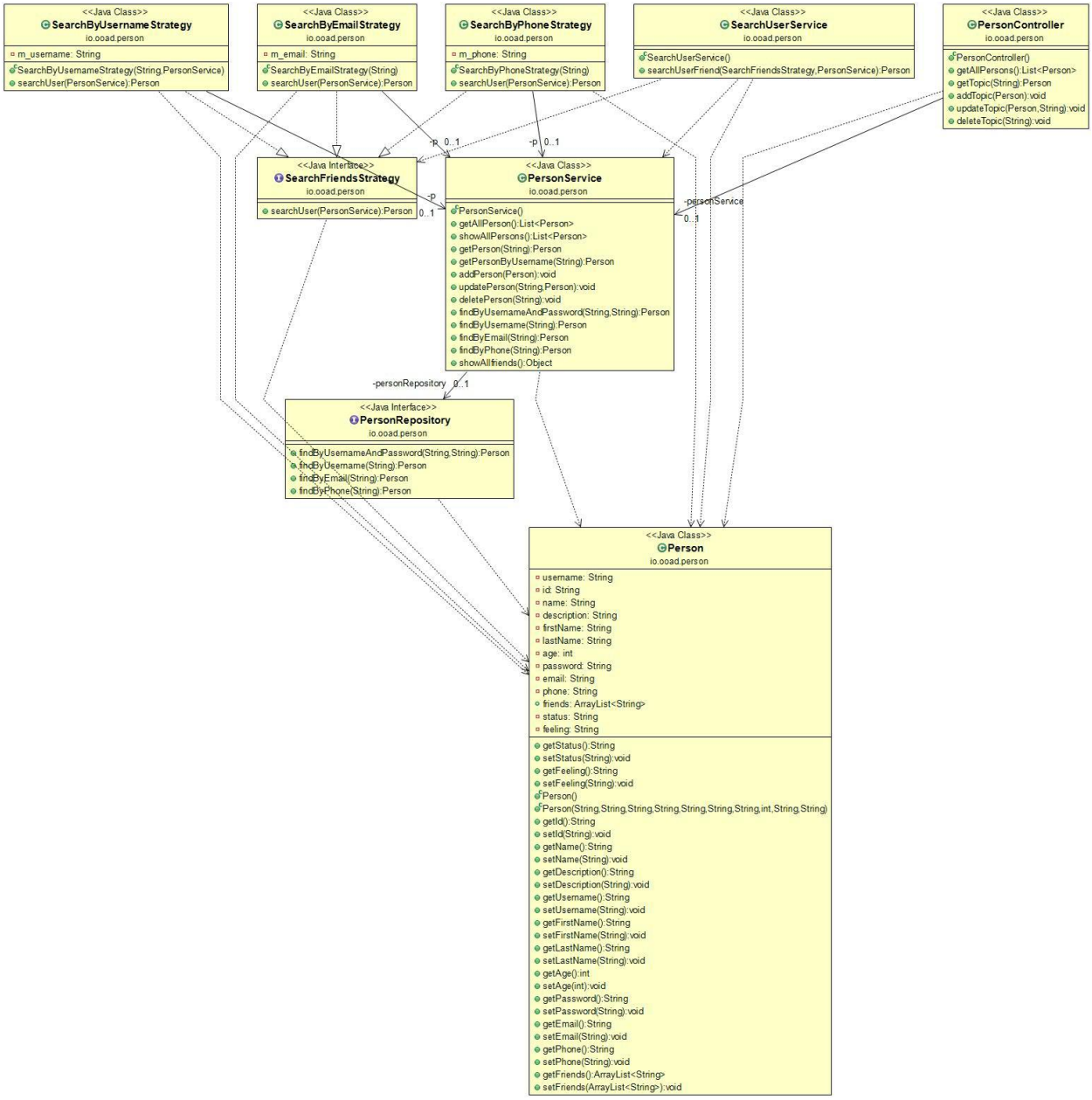
Functional Requirements			
ID	**DESCRIPTION**	**TOPIC AREA**	**PRIORITY**
FR-02	All users should be able to modify their privacy settings	Authentication	Low
FR-03	Only CU-Boulder students, staff and alumni are allowed to set up an account	Accounts	High
FR-04	Users should be able to search other users based on various filter settings such as graduation year, location and major	Search	High

Non-Functional Requirements			
ID	**DESCRIPTION**	**USER**	**PRIORITY**
NFR-01	Passwords can be stored after encrypting	Authentication	High
NFR-04	The system should generate periodic logs about new users and the amount of data added to the database	Stats	Medium

Part 2 class diagram



Final Class diagram:



What changed and why?

MVC:

The model interacts directly with the database doing the CRUD operations. The model here is the personService class. View is basically the user interface, in this project it is the JSP page that is implemented with HTML and CSS. The controller acts as the interface between the model and view and

performs the data exchange to display data on the JSP page and to get the data from JSP page to the model. The controller in this project is the TestController class.

Strategy Design Pattern:

The strategy design pattern was implemented to determine how to look for a person. The user can be looked for by using email, username or phone number. This is extendible and when new features are added to the system. This is in line with the OAD design that is needed. The strategy pattern proved useful as we needed to use different algorithms that are determined at runtime based on the input from the user.

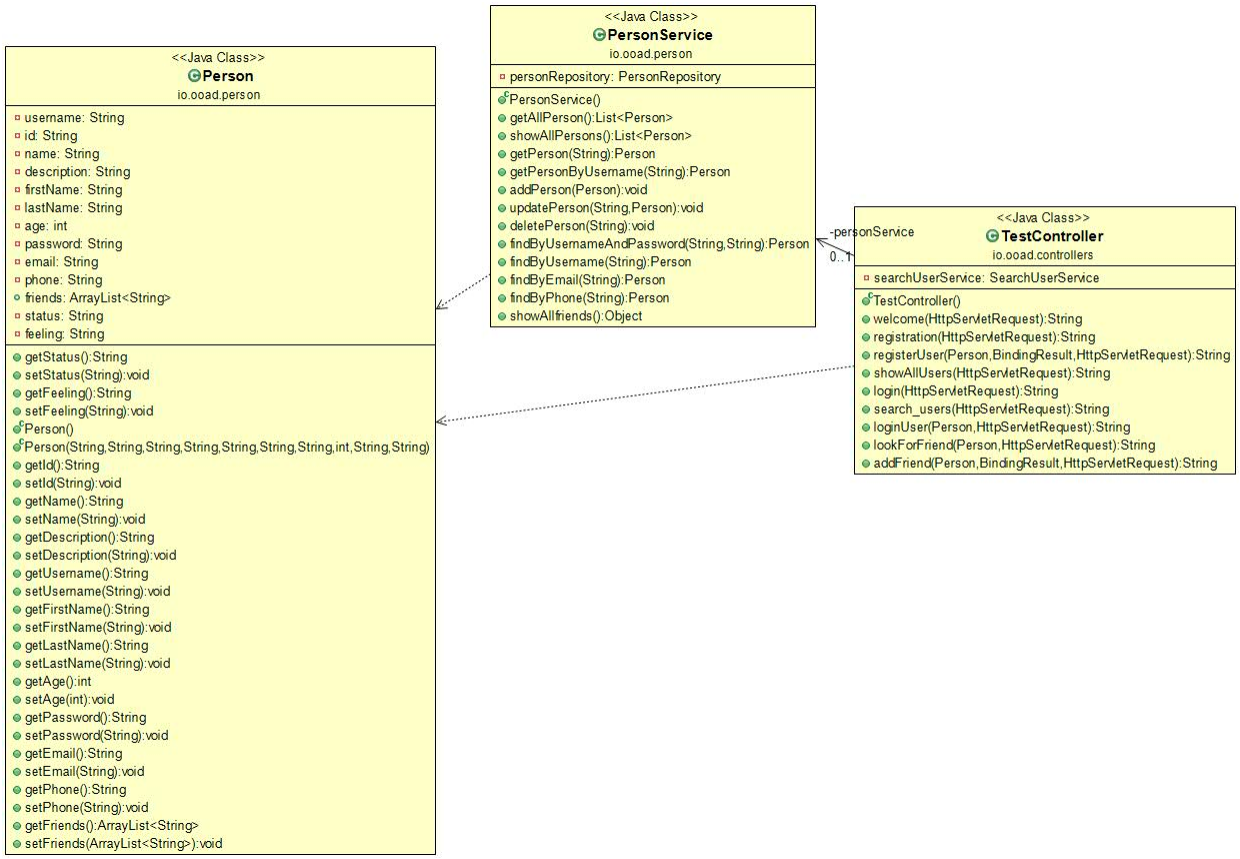
Composite Design Pattern:

We had planned to use the Composite Design Pattern but could not implement it finally. This was because of some issues arising with the framework. We could not resolve many issues in the final implementation.

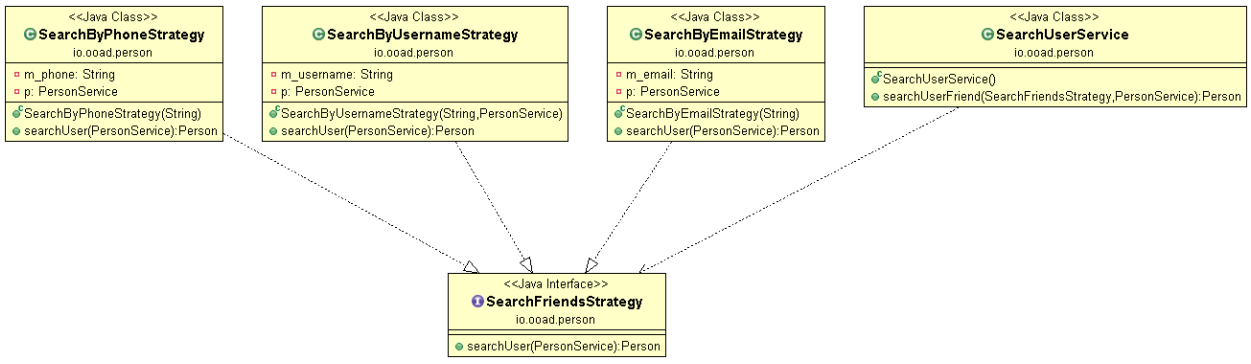
Overall, the design up front enabled us to see the entire project much before implementation. This helped with actually coding up the project. However, we did not see many of the challenges we faced during actual implementation. These mainly involved the framework and getting the interfaces correct. We implemented the web application using the Embedded Apace Derby Database instead of MySQL.

Classes from class diagram that implement each design pattern

MVC:



Strategy Design Pattern



What did you learn from the process:

This project we worked on gave us a lot of experience in working on an implementation of a system from the analysis, design, environment setup and finally the implementation of the project with version control.

The initial class diagram analysis helped in understanding how the system would look like up front. The other diagrams helped similarly. The refactoring to the design patterns gave us a perspective of how we can implement a design pattern into our design by tweaking a few elements of the architecture.

On final implementation, we realized that prior experience in setting up the interface between java and a web application would be useful. We learned this the hard way. Many parts of the system had to be dropped all together as it was too much to do in the given amount of time. We faced many challenges along the way. Many of the technologies we implemented were way beyond our comfort zone. We implemented the web application using the Embedded Apace Derby Database. We have never played with Spring, Hibernate, JSP, Databases, Glassfish vs Tomcat server and the various IDEs before. This project helped develop skills in these areas.

The valuable feedback from the instructing team helped in assessing our progress in the project.