Project Part 3: Refactoring

Team: Jeet Baru
Shalin Shah
Vihanga Bare

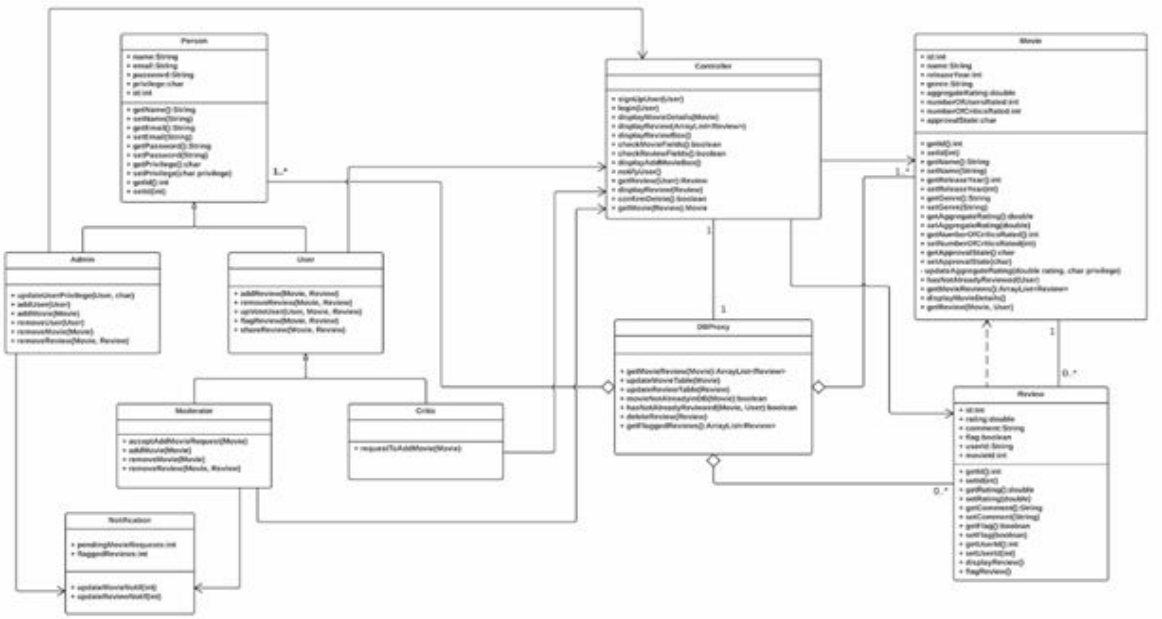
Title: Foodzie!

Description: Foodzie is an online restaurant discovery guide providing information on restaurant types,cuisines, reviews by critics and users, location and contact information. It is the one-stop-shop for all things food. Users will be able to search for outlets and customise results based on their liking. The application shall provide all-you-wish-to-know information about selected restaurant along with reviews by critics. Restaurant Owners get this platform to market their place by adding content to their restaurant's page.

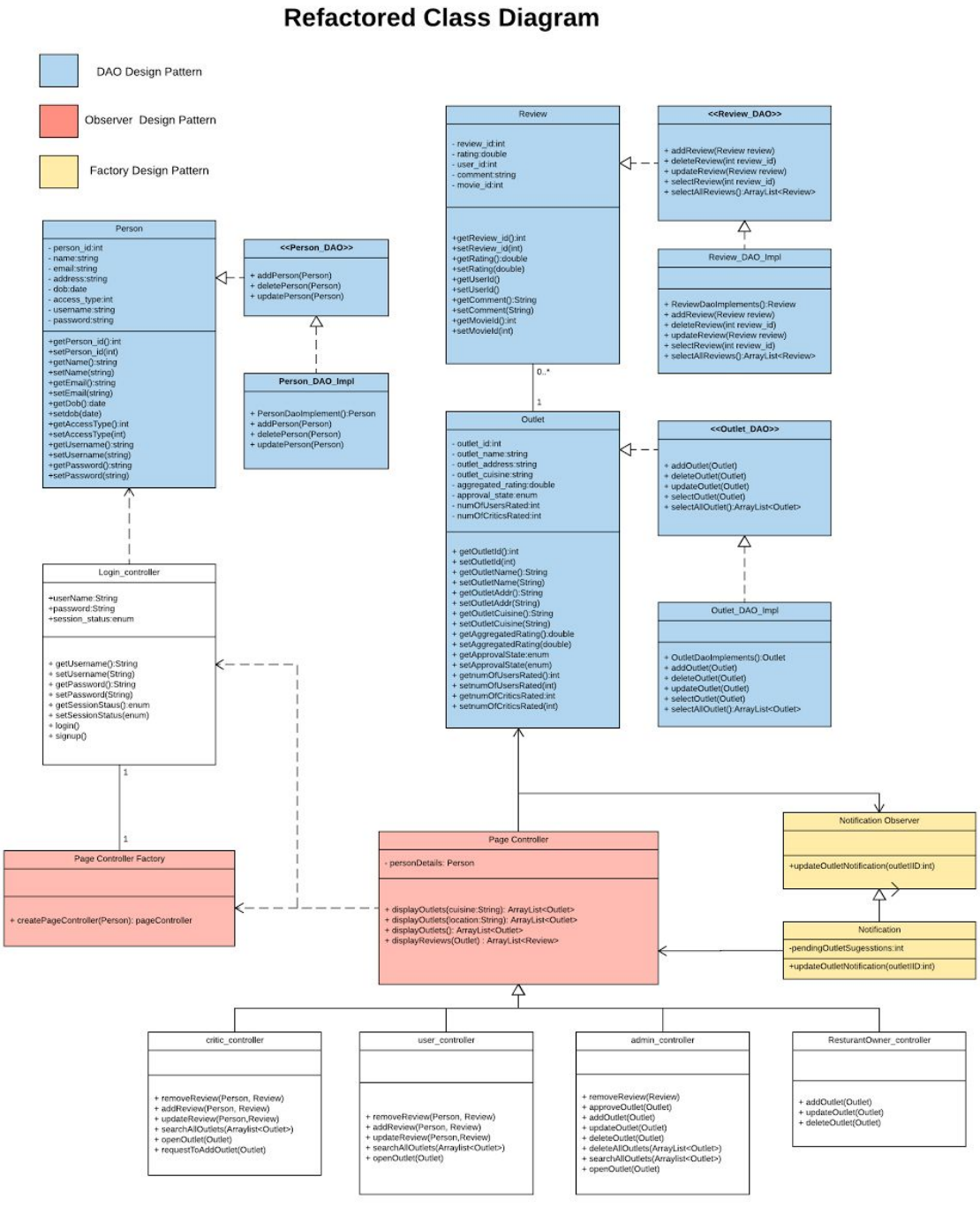
ACTORS: Admin, Restaurant owners, Users, Critics

In this part we intend to improve our design submitted in part 2 by refactoring and thereby applying design patterns we learnt in the course

Our class diagram from part 2 is as show below:

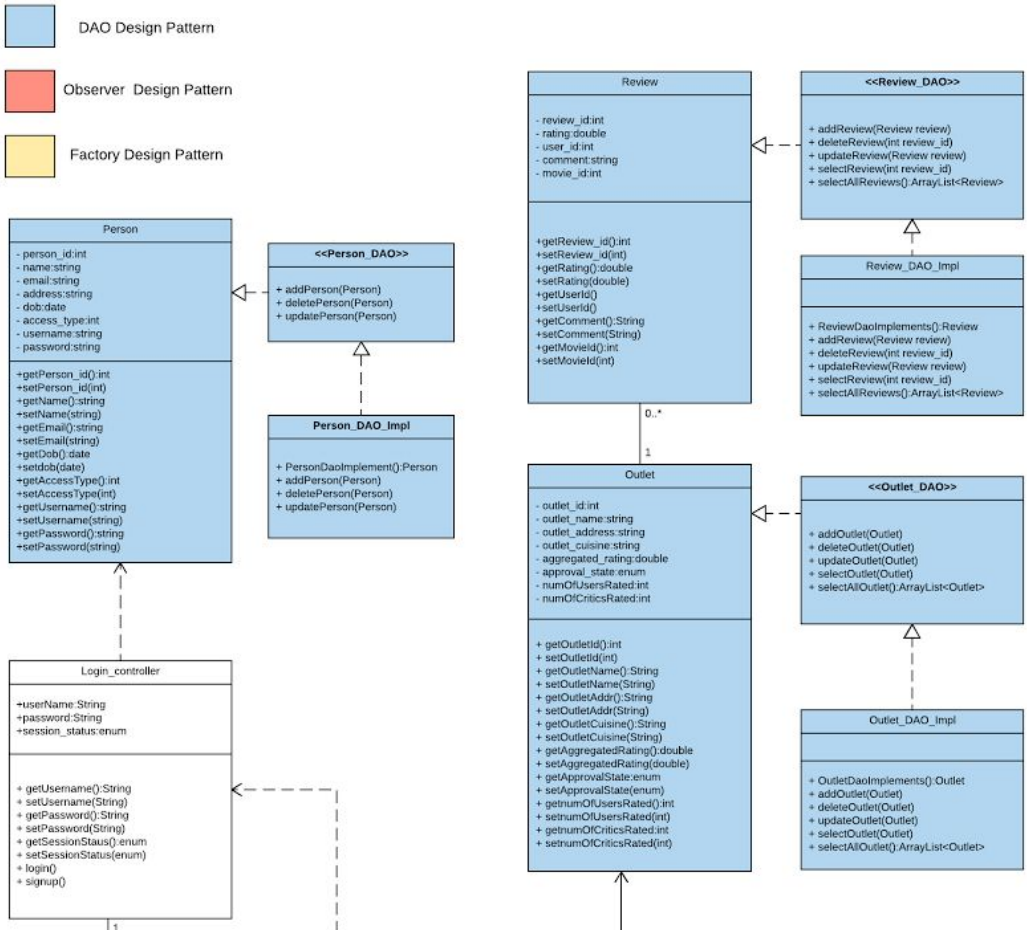


Refactored Class diagram for part 3:

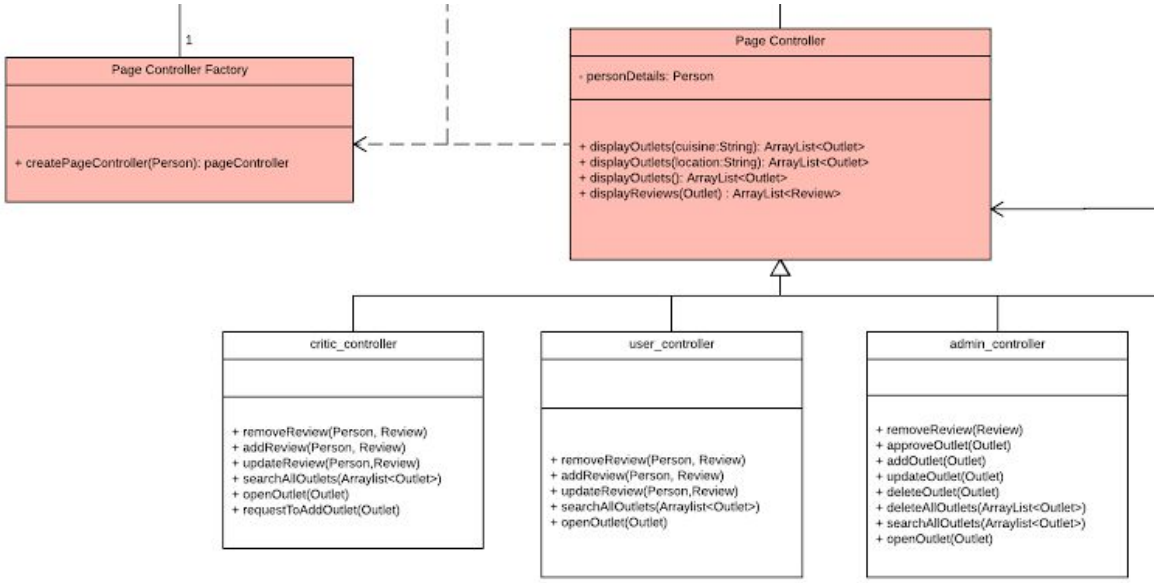


As seen in the previous design we had planned on having a **central controller** and **DB Proxy** to handle the functionality of the whole application centrally. This shall not be a good way of implementing it because shared responsibility is desired for reliable operation.

Hence, we decided to have separate controllers for each entity of Outlet, Review and Person. We plan to implement this using the **DAO (Data Access Object) Design Pattern**. This shall work as a layer of abstraction separating low level data accessing API from the high level business services also sharing responsibility between Person, Review and Outlet. Hence we plan to implement DAO pattern at three instances in our design.



We also plan to implement the **Factory design pattern** that will help us display views to user depending on the which user(User, Admin, Critic, Restaurant Owner) is logged in at run time. We have added a **Page Controller** class that would handle data to be displayed on the page depending on the user type. The **Page Controller** class together with **Page Controller Factory** class forms the **Factory design pattern**. **User Controller, Admin Controller, Owner Controller** and **Critic Controller** inherit from **Page Controller** and hence form the different products of the factory.



Also we intend to implement the **Observer design pattern** to notify the admin of addition of outlets recommended by User and Restaurant Owner and additions of these recommendations to the database. This shall be implemented by having the **Notification Observer** extending to a **Notification class** which updates the **Page Controller** only when the change occurs.