PROJECT PART 6

Team: Omkar Reddy Seelam, Rishabh Berlia, Shivasankar Gunasekaran
Title: Rated – Online Movie Review System
Project Summary: An Online Movie Rating System in the form of a web application. It is a database of movies which allows the users to access movie reviews and ratings. The users can add/delete/update reviews. Reviews can also be up-voted and shared on social media.

1.List the features that were implemented (table with ID and title):

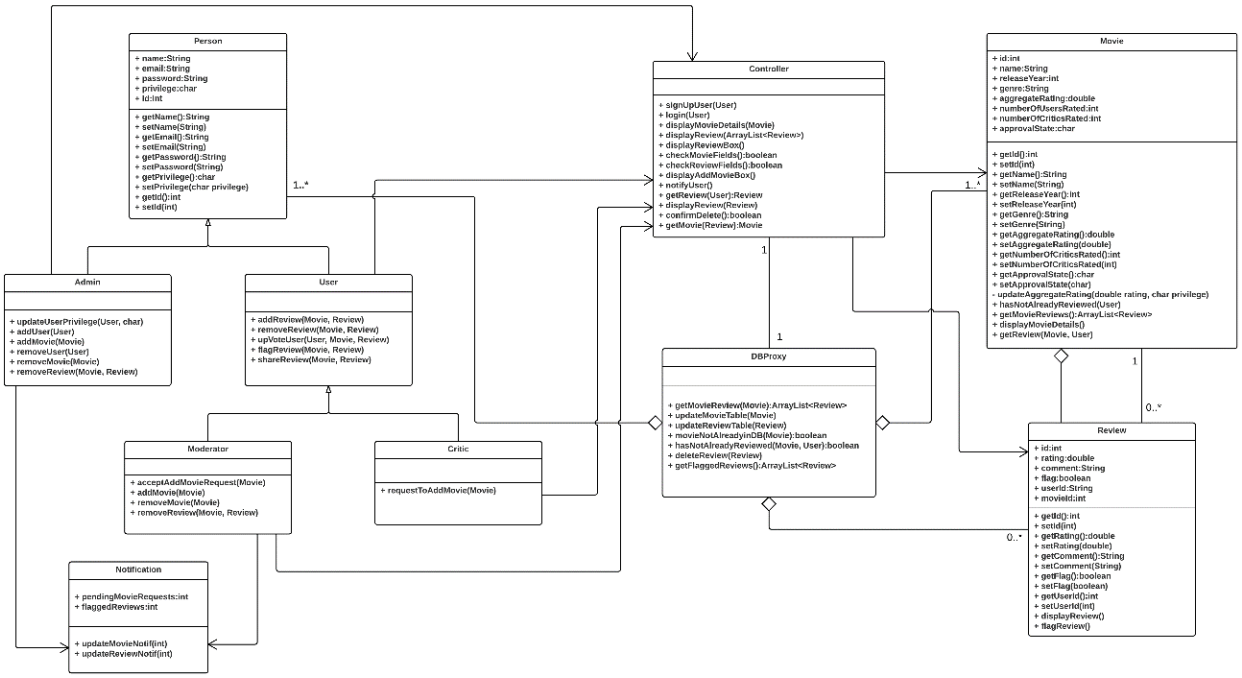
User Requirement		
ID	Description	User
UR-01	As a new user, I should be able create a role based account	All Actors
UR-02	As an existing user, I should be able to login to the system	All Actors
UR-03	As a User or a Critic, I should be able to view and give rating and review	User, Critic
UR-04	As a User or a Critic, I should be able to delete my rating and review	User, Critic
UR-05	As a user, I should be able to view movie list based on a filter	All Actors
UR-08	As an Admin or a Moderator, I should be able to add/remove movie in the list	Admin, Moderator
UR-09	As an Admin or a Moderator, I should be able monitor reviews	Admin, Moderator
UR-12	As a Moderator, I should be able to approve request to add movies by a Critic	Moderator
UR-13	As a Critic, I should be able to request to add a movie	Critic

2. List the features that were not implemented from Part 2 (table with ID and title):

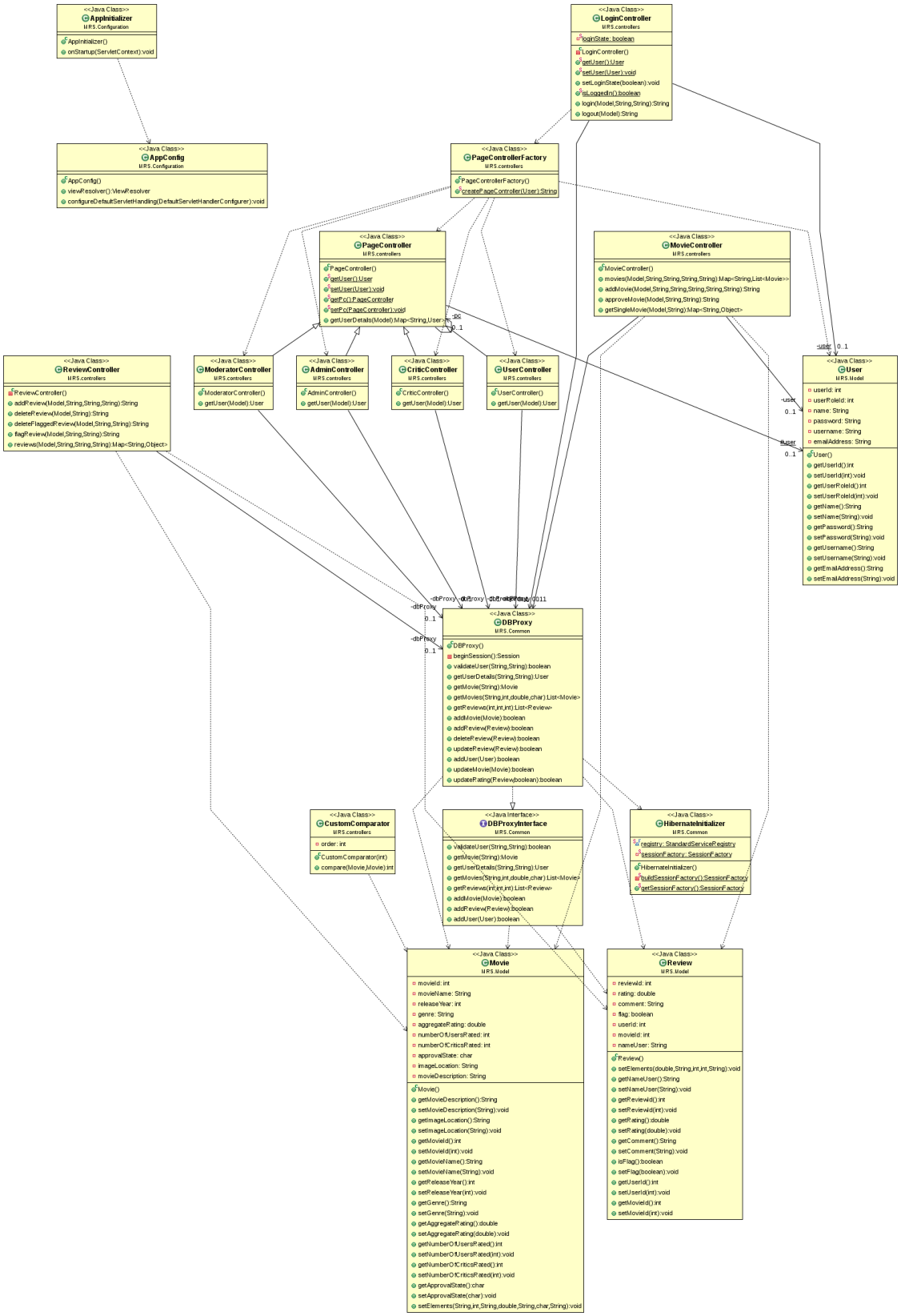
User Requirement		
ID	Description	User
UR-06	As a User or a Critic, I should be able to up-vote an existing review	User, Critic
UR-07	As a User or a Critic, I should be able to share any review on social media	User, Critic
UR-10	As an Admin, I should be able to promote Users to Critics and Critics to Moderators.	Admin
UR-11	As an Admin, I should be able to add/remove users	Admin

3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

a. Part 2 class diagram



b. Final class diagram



c. What changed and Why?

The final class diagram has a lot of changes from the class diagram from Part 2. The design of our application during Part 2 has a BLOB anti-pattern where in there was a single controller class which is responsible for all the user interface elements. The tasks have been broken down and more controller objects with tasks delegated appropriately have been created now. The Model View Controller (MVC) Architectural pattern has been implemented.

We implemented the Factory Method Design pattern to instantiate different type of user objects based on their roles. This allowed to encapsulate the object creation from object behavior, ensuring that if new type of user classes is added to the application in the future, only the object creation class has to be changed.

We also implemented the Singleton Design pattern to make sure only one instance of any user object is instantiated using the PageController class.

We created a new class User which is inherited by the PageController and LoginController for code reuse. We also re-used existing code for different use cases by using certain extra parameters for the methods.

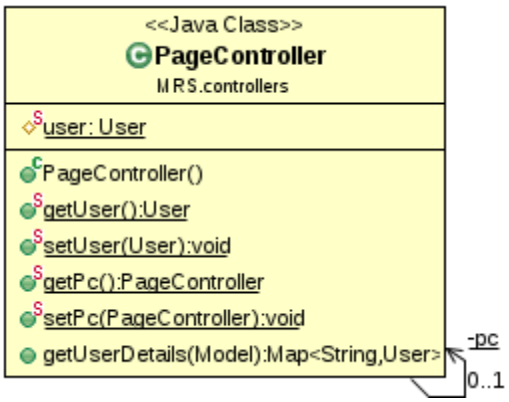
4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF). If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram:

Yes, we implemented the following design patterns: Factory Method, Singleton, Proxy and MVC architectural pattern.

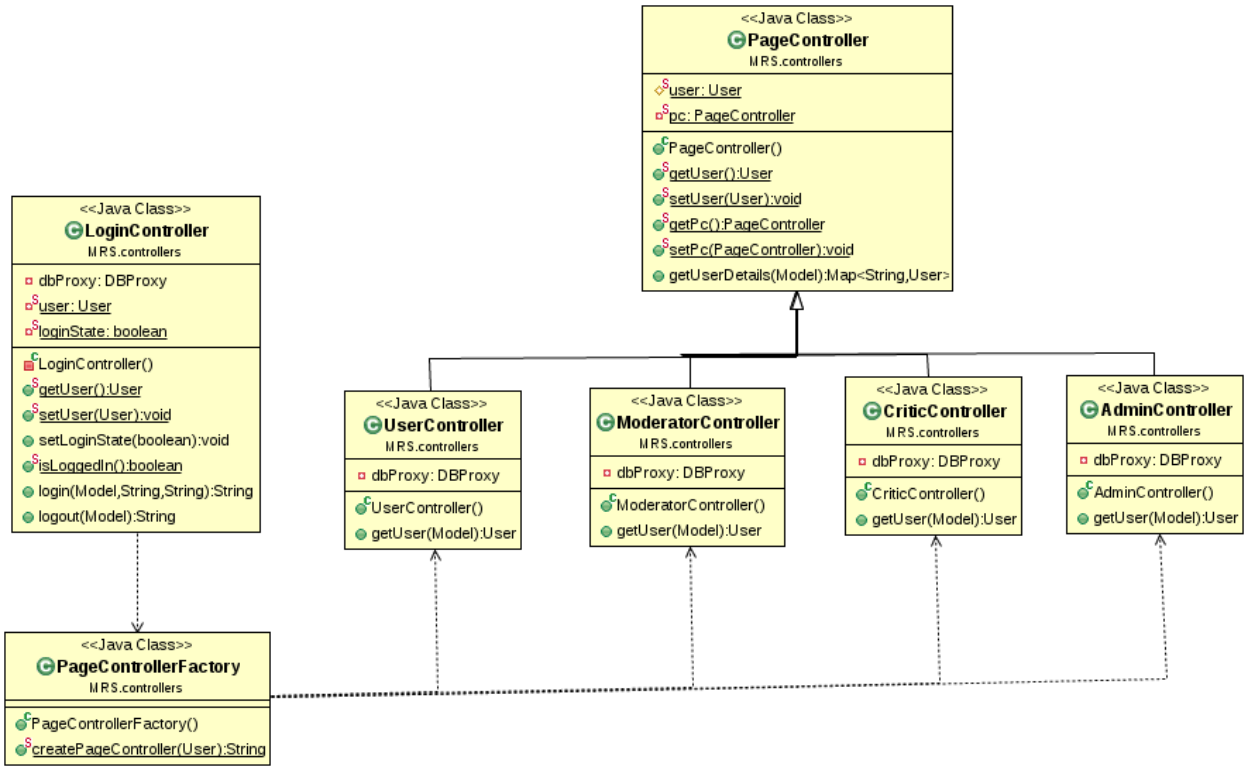
Apart from the design patterns, we implemented our application using the Model View Controller architectural pattern. The controller objects are the primary ones

that interact with the User Interface events and the Model classes. The controller updates the model based on the logic required and also updates the UI accordingly.

We implemented the Singleton pattern in the PageController class to make sure only one instance of the user is instantiated. Following is the Class diagram from representing the single instance of the Class.



Next, we used the Factory Method design pattern to instantiate different users based on their roles. All the user controllers inherit from the PageController class and the specific user type controller object is created using the PageControllerFactory class. The following is the class diagram depicting the class dependencies and relationships.



We also implemented the Proxy design pattern for the Database accesses. The DBProxy class is in the middle providing abstractions to the Database accesses.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

The process of analysis and design helped us to implement a better, reusable and maintainable code. The SDLC and the design patterns helped us get a good picture of how things should be implemented and avoid smelly code. Heavy use of object oriented principles and design patterns helped also taught us how responsibilities can be delegated and organized among different classes appropriately. The initial implementation had some anti-patterns but gradually these have been rectified and refactored to good design solutions. Further, we learned that user acceptance and continuous refactoring are a key to make sure that the application code is maintainable and reusable. This complete process of designing, creating and implementing with continuous changes to the design by refactoring taught us good object oriented programming and designing skills.