Final Report

TEAM #32:
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 customize 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

List of the features implemented:

➢ **User Requirements:**

User Requirement			
ID	**Description**	**User**	**Priority**
UR-01	As a new user, I should be able to create an account based on my role as user, food critic, restaurant owner, admin.	All Actors	Low
UR-02	As an existing user, I should be able to login into the system	All Actors	Low
UR-03	As an existing User or Critic, I should be able to comment on and rate restaurants.	User, Food critic	Low

UR-04	As an existing user who is administrator, I should be able to delete objectionable comments.	Administrator	Low
UR-07	As a Restaurant Owner, I should be able to create and add content for my own restaurant.	Restaurant Owner	Low
UR-08	As an administrator, I should be able to create and delete content for all restaurants.	Administrator	Low
UR-13	As a Restaurant Owner, I should be able to delete my own restaurant's content, replies and uploaded images.	Restaurant	Low
UR-14	As an Administrator, I should be able to create or delete any content across my application.	Administrator	Low
UR-15	All actors should be able to search restaurants based on different search filters in application	All actors	High
UR-17	All actors should be able to delete their accounts	All actors	Low

➢ **Business Requirements:**

Business Requirements		
ID	**Description**	**Topic Area**
BR-01	Project must be implemented in Java using Object Oriented Design practices.	Implementation
BR-02	Must be a standalone application	Implementation
BR-03	The project development should be able to be tracked using GitHub	Submission
BR-04	Application data to be backed up and tracked using relational database	Data Storage
BR-05	The code should be refactored to follow industry recognized design patterns	Refactoring
BR-06	The critic ratings should have a higher weightage than a user's rating	Rating

➢ **Functional Requirements:**

Functional Requirement			
ID	**Description**	**Topic Area**	**Priority**
FR-01	Password based authentication for all users while login	Authentication	Low
FR-02	All users should be able to reset passwords for their accounts	Authentication	Low
FR-03	The rating displayed is the weighted average of the critics and user ratings	Rating	High
FR-04	The user's search should be filtered according to name, cuisine type and location.	Search	High

➢ **Non - Functional Requirements:**

Non-Functional Requirement			
ID	**Description**	**Topic Area**	**Priority**
NFR-02	The system should be available to all users 24x7	Reliability /Performance	High
NFR-03	Application be able to support at least N users simultaneously using the application.	Performance	High

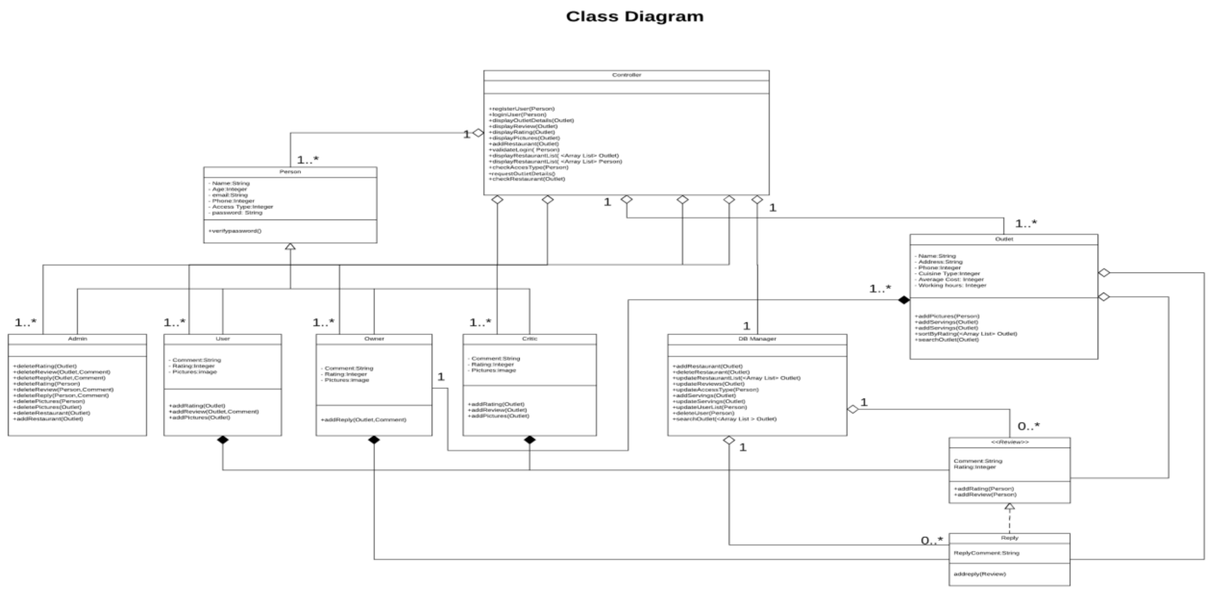
List of the features not implemented -

- **User Requirements:**

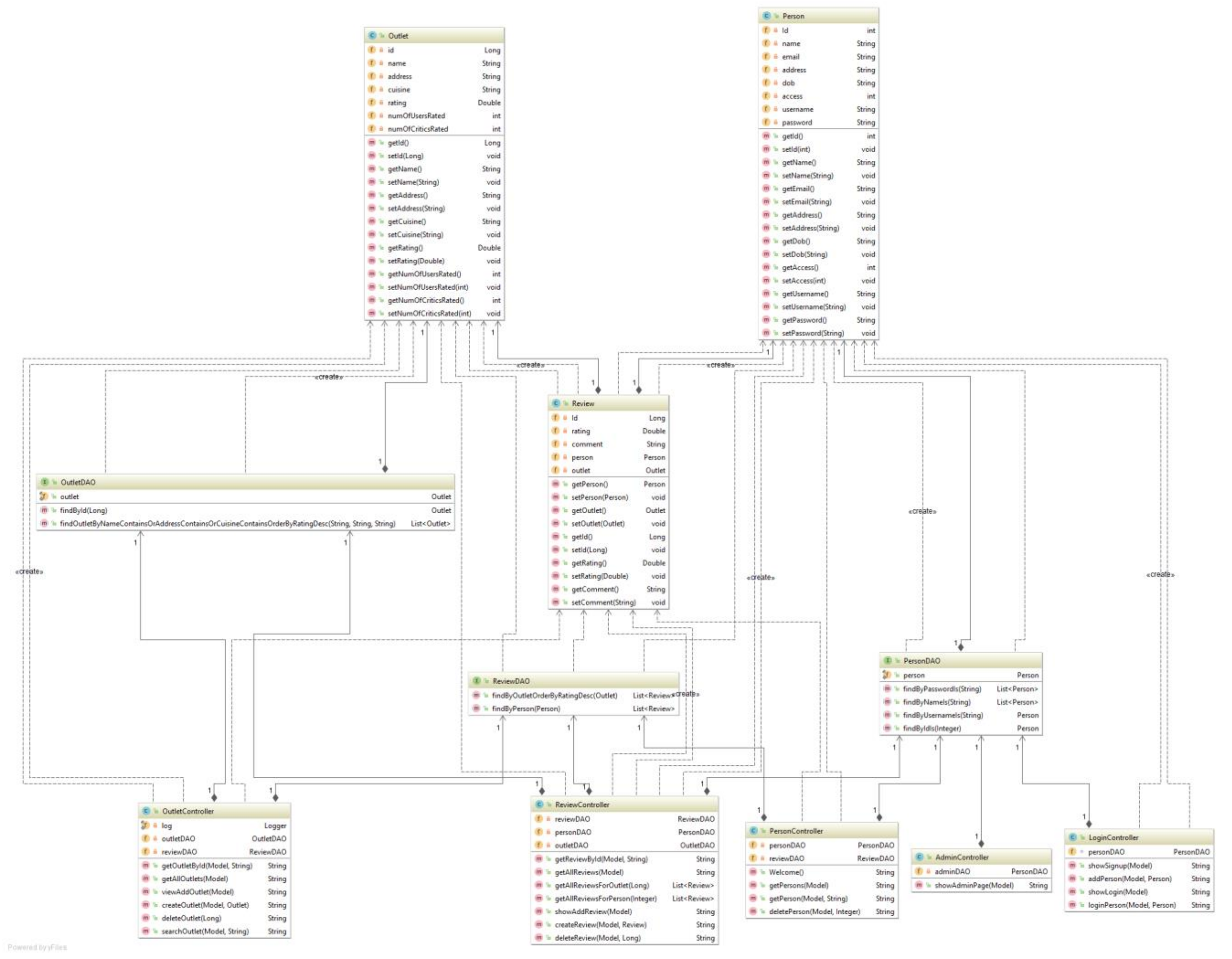
User Requirement			
ID	**Description**	**User**	**Priority**
UR-05	As a Food critic or Restaurant Owner, I should be able to reply to comments given by user.	Food critic, Restaurant Owner	Low
UR-06	As an administrator, I should be able to delete objectionable replies given by food critics or Restaurant Owner.	Administrator	Low
UR-09	As a User or Food critic, I should be able to suggest content for any restaurant	User, Food critic	Low
UR-10	As a Food critic or User, I should be able to upload images for all restaurants	User, Food critic	Low
UR-11	As a Restaurant Owner, I should be able to upload images for my own restaurant	Restaurant Owner	Low
UR-16	As an administrator, I should be able to approve the suggestions to add content by User or Critic or Restaurant Owner.	User, Food Critic, Restaurant Owner	Low

Difference between old class diagram and new class diagram. What changed? Why?

Old Class Diagram



Final Class Diagram



What changed ? Why?

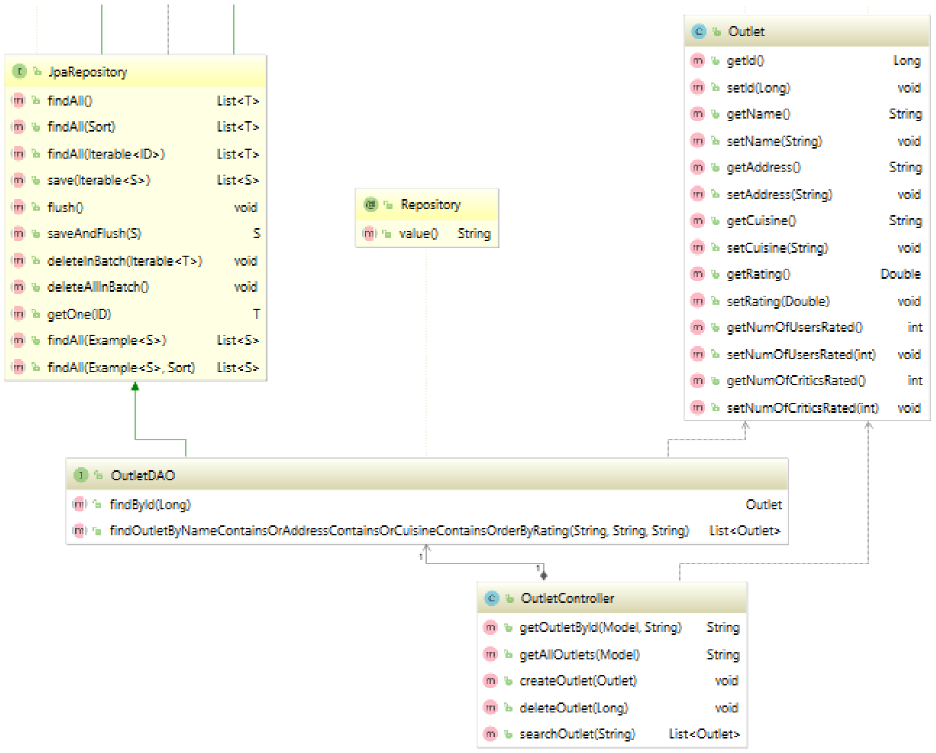
Yes. As seen in the previous design we had planned on having a **central controller** and **DB Proxy** to handle the functionality of the whole application centrally. This wasl not be a good way of implementing it because shared responsibility is desired for reliable operation an improve the modularity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

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**. We also realised that we shall need to build instances of the Review individually which shall complicate the constructor, Hence we implemented the Builder design pattern.

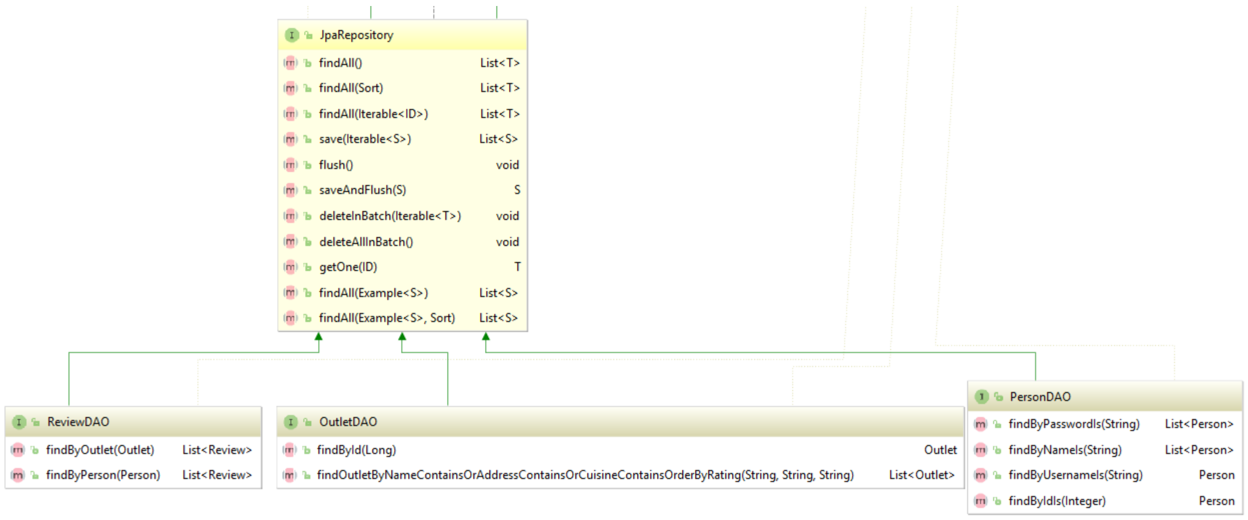
Design Patterns used and associated class diagrams -

DAO Design Pattern

We have implemented separate controllers for each entity of Outlet, Review and Person 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.

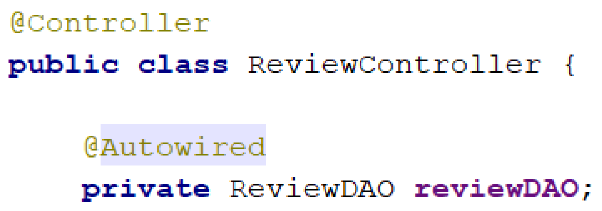


JpaRepository
findAll() List<T>
findAll(Sort) List<T>
findAll(Iterable<ID>) List<T>
save(Iterable<S>) List<S>
flush() void
saveAndFlush(S) S
deleteInBatch(Iterable<T>) void
deleteAllInBatch() void
getOne(ID) T
findAll(Example<S>) List<S>
findAll(Example<S>, Sort) List<S>
Repository
value() String
Outlet
getId() Long
setId(Long) void
getName() String
setName(String) void
getAddress() String
setAddress(String) void
getCuisine() String
setCuisine(String) void
getRating() Double
setRating(Double) void
getNumOfUsersRated() int
setNumOfUsersRated(int) void
getNumOfCriticsRated() int
setNumOfCriticsRated(int) void
OutletDAO
findById(Long) Outlet
findOutletByNameContainsOrAddressContainsOrCuisineContainsOrderByRating(String, String, String) List<Outlet>
OutletController
getOutletById(Model, String) String
getAllOutlets(Model) String
createOutlet(Outlet) void
deleteOutlet(Long) void
searchOutlet(String) List<Outlet>



JpaRepository
findAll() List<T>
findAll(Sort) List<T>
findAll(Iterable<ID>) List<T>
save(Iterable<S>) List<S>
flush() void
saveAndFlush(S) S
deleteInBatch(Iterable<T>) void
deleteAllInBatch() void
getOne(ID) T
findAll(Example<S>) List<S>
findAll(Example<S>, Sort) List<S>
ReviewDAO
findByOutlet(Outlet) List<Review>
findByPerson(Person) List<Review>
OutletDAO
findById(Long) Outlet
findOutletByNameContainsOrAddressContainsOrCuisineContainsOrderByRating(String, String, String) List<Outlet>
PersonDAO
findByPasswordIs(String) List<Person>
findByNameIs(String) List<Person>
findByUsernameIs(String) Person
findByIdIs(Integer) Person

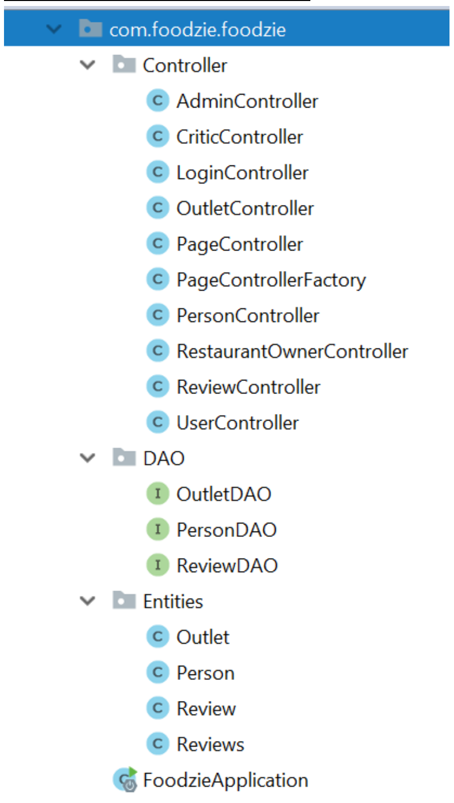
Singleton Design Pattern



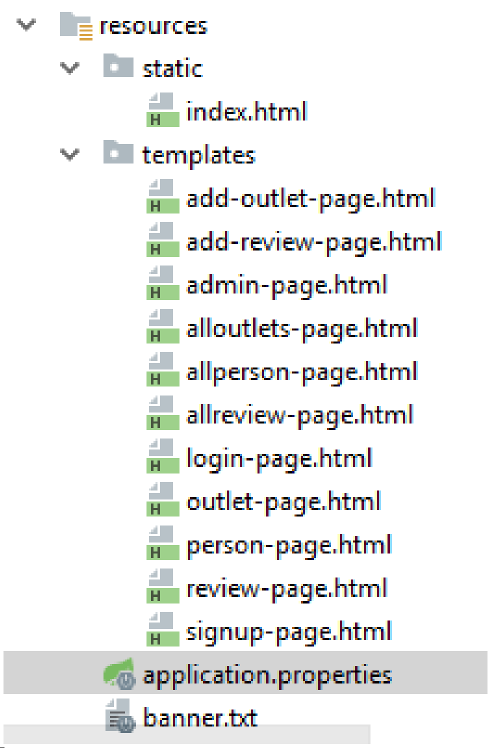
```
@Controller
public class ReviewController {

    @Autowired
    private ReviewDAO reviewDAO;
```

We are implementing Singleton design pattern in the sense that we are creating a single Data Access Object (DAO). Spring MVC will only create one instance of these each Person DAO, Outlet DAO and Review DAO. We are using these database objects to keep track of all operations with the database and all these operations will refer to these single created instances. Inside each controller of Spring MVC framework, by using **'@autowired'** for the DAO object we are ensuring that only the single instance of this database object gets created. By implementing this design pattern we ensure that, if we inject an object to Spring's DAO, it is wired only once and it never get refreshed. It is ideal to make all the controllers as the singleton and value objects as the prototype.

MVC Design Pattern



```
com.foodzie.foodzie
    Controller
        AdminController
        CriticController
        LoginController
        OutletController
        PageController
        PageControllerFactory
        PersonController
        RestaurantOwnerController
        ReviewController
        UserController
    DAO
        OutletDAO
        PersonDAO
        ReviewDAO
    Entities
        Outlet
        Person
        Review
        Reviews
    FoodzieApplication
```



```
resources
    static
        index.html
    templates
        add-outlet-page.html
        add-review-page.html
        admin-page.html
        alloutlets-page.html
        allperson-page.html
        allreview-page.html
        login-page.html
        outlet-page.html
        person-page.html
        review-page.html
        signup-page.html
    application.properties
    banner.txt
```

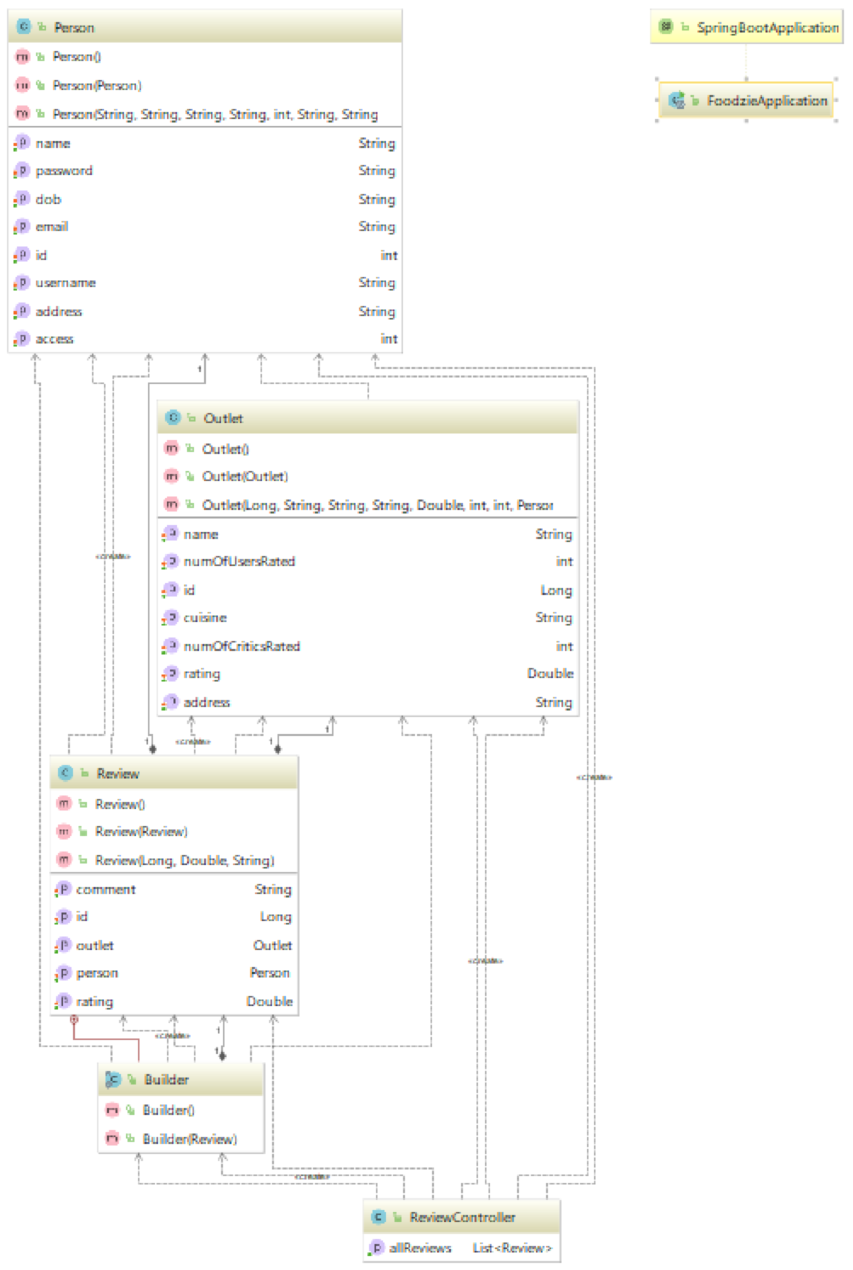
MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

- **Model** - Model represents an object carrying data. It can also have logic to update controller if its data changes.
- **View -** View represents the visualization of the data that model contains.
- **Controller -** Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

We are using Person, Outlet and Review Objects, acting as our model. In out veiw layer we have used the Thymeleaf Template Engine, that helps to map the underlying objects to our HTML pages. Each Person, Review and Outlet class have their own controllers which are responsible to store data into corresponding DAO object and upload the view corresponding to this instance.

Builder Design Pattern

We have also implemented the builder design pattern to generate instances of Review class. Review object is complex as it includes members of the Person and Outlet class. We implement Builder so that we can populate the Review object as per our needs. This is very flexible approach as compared to that of using a constructor. Class diagram for our implementation of builder is shown below:



<u>What have you learnt about the process of analysis and design now that you have stepped through the process to create, design and implement a system ?</u>

Through this project we got an hands on experience of the processes essential to development of a production grade application.

We understood that it is essential to have a detailed analysis of the classes and objects you will use, and the design pattern, software architecture diagram, before actually going ahead and implementing a large project and that if you are careful enough in design and creation phase you would not face a lot of problems going ahead during the implementation phase.

When creating the design architecture of the system, we started with an overview of which classes we planned to use, how we planned on implementing object oriented principles, what kind of design pattern would suit our use cases and goals.

Based in the valuable feedback from the instructing team, we paid attention to minute things that would seem wrong in the beginning but might create bigger side effects on our implementation down the line.

We learnt how to make use cases for your software system, how would you map these use cases across different user and functional requirements of the system, how different components of the system collaborate with each other though use of Class Diagrams, how does a particular use case run from start to end using Sequence Diagram. Guidance from professor and assisting instructing team on our UML diagrams made our implementation more efficient.

We learnt different Java application frameworks throughout the entire course of implementation of this project including - Spring Boot Framework, JPA Repository, Thymeleaf Java Template Engine, Hibernate ORM, HTML.

Link to Final Demo Video –

Link to Individual Submission Videos –

Shalin Shah –
Jeet Baru -
Vihanga Bare - https://youtu.be/8Wl4AOJimJ8