SWEN90007

Software Design & Architecture

Semester 2 - 2019

Submission 3

Software Architecture Document

Wednesday 16:15 PM Workshop

Savan Kanabar

Login: - skanabar@student.unimelb.edu.au

ID: - 965371

Umut Cem Soyulmaz

Login: - usoyulmaz@student.unimelb.edu.au

ID: - 989654

Table of Contents

Github: https://github.com/ucsoyulmaz/swen90007-Wednesday-16.15---Group1

Heroku: https://recipebook-swen9007.herokuapp.com

Overview of system

We are planning to build a system where users can upload and store their various cooking recipes which they want to share with other people. The user who uploads these recipes can also edit it or delete it entirely. These recipes will be stored using some keywords along with the appropriate category in which that recipe belongs to, for example; all the recipes for soups will be stored under same category. Along with this, the user will need to upload all the ingredients for that recipe and an estimate of cooking time and cost for the same.

We plan to have two types of users; one would be super user, and another would be regular user. Regular users can upload a recipe, edit it or delete it entirely as well as can read recipes posted by other users, comment on those posts and rate them. While super user is allowed to do all that stuff but with an increased scope where he/she can delete the comments made by all the users and delete the recipes

Feature A

Our primary feature is that the users can upload the recipes which they want to share on the website which they can later edit or delete entirely. Along with that they can also read the recipes posted by other users.

Feature B

Our second feature is that the super user can create categories for recipes, can delete the recipes posted by normal user and delete the comments made by normal user. We implemented security and now have sessions and locks which allows concurrency in our system without having deadlocks.

User levels

Currently we have two levels of user;

1. Normal user
2. Super user

A normal user is someone who can create an account and post the recipes that they want to share with the world, they can edit their recipes or delete them entirely if they want. Apart from this they can also comment on other user's recipes and can edit or delete their own comments.

A super user is someone who can do everything done by a normal user but apart from that they can delete the recipes and comments made by normal users.

Features provided as part of Feature A

1. A user can create a recipe.
2. A user can edit his own recipe.
3. A user can delete his own recipe.
4. All users can view the recipes posted by all users.
5. A user can comment on recipes posted by other users.
6. A user can comment on his own recipe post.
7. A user can edit comments made by him on his own recipe post.
8. A user can delete his own comment.
9. A user can delete comments by other users on his/her post.
10. All users can see all comments

Features provided as part of Feature B

1. We implemented Session
2. Implemented User Levels so now we have admin user and normal user
3. Implemented Patterns for Input Controllers and View
4. Implemented Locks to prevent Deadlocks and allow concurrent access
5. Implemented Remote Facade and DTO
6. Implemented Security

Why is this an Enterprise system

The reasons by which we can say that this system is an enterprise system are as follows:

1. The recipes can be posted simultaneously by multiple users and it will be on the same data source (Something which is done on Facebook in terms of posts).
2. The uploaded recipes can be viewed by multiple users.
3. The recipe which was uploaded can be edited by the owner only. The reason to design the system this way is because it does not make sense for someone else to edit my content. For example, if we make any kind of post on social media which is wrong somehow than others cannot edit that post, they can report it for sure, but editing is not up to others.
4. The comments can be made on a single recipe post by multiple users concurrently.
5. The comments can be viewed concurrently by multiple users.
6. The comments can be edited by the owner only. This is due to the same reason explained above.
7. The profile page of individual user can be seen concurrently by multiple users.

User Scenarios & Dummy Data

Login Details:

Normal User:

ID: either Savan or Umut or Ajay

Password: Savan or Umut or Ajay respectively

Admin User:

ID: either Eduardo or Maria or anyone you created

Password: Eduardo or Maria respectively or anyone you created

These are the accounts already in the database, alternatively you can create your own accounts as well as mentioned in the scenario below

Scenario 1: Open the link of Heroku and Login or Signup

1. You will be directed to Login page where you can use the credentials mentioned above to Login, alternatively you can also signup with your own credentials, this allows you to choose a user type as well, you can either sign up as an admin or a normal user.
2. Create a Recipe: can be used to create a recipe.

Scenario 2: User wants to create a Recipe

1. Click create a recipe button

2. Fill the form which is on the page loaded after clicking the button and click next

3. On top of this page you can see an ID field which will generate an ID for your recipe which you are about to create and become a masterchef!!

4. Select the necessary ingredients from the list and click submit.

5. You can see your profile page where all your recipes are shown in the form of timeline.

Scenario 3: User wants to view Recipes of other users

1. From homepage, click on Timeline

2. Here, all the recipes posted by all the users will be shown in the form of a table with some basic details like name of recipe, owner name and cooking time.

3. There is a view button next to each recipe, if some recipe generates enough interest from you and you want to see how it is made, click View.

4. Here you can see the description on how you can cook that delicious recipe and also the ingredients that you will require when you start cooking. (This also shows implementation of lazy load pattern of not loading all the data at once)

5. That's not it, you can also comment on recipes posted by other users and obviously on your own recipes as well which we will discuss in the next scenario.

Scenario 4: User wants to Comment on a Recipe posted by someone

1. For convenience it is assumed that you have already reached step no 5 of Scenario 3, if not then please follow those steps first.

2. Click on write a comment button

3. You can type your comment in the text area provided (remember to be excellent to your fellow community members.)

4. After writing comment, click the Post button.

5. You will be redirected to previous page where you can see your comment and comments made by other users.

Scenario 5: User wants to update a Recipe he created

1. For this it is necessary that you have created a recipe.

2. Click on My Recipes button at home page

3. From the list of your recipes, click the view button for the one you want to edit

4. Click on Edit the Recipe button

5. Make necessary changes and click Next

6. Make changes in ingredients if needed and click submit.

7. And that's how you edit recipes.

Scenario 6: User wants to delete a Recipe he created

1. For convenience we assume that you have performed until step 3 of Scenario 5 and are already viewing a specific recipe that you created.

2. Click on Delete the Recipe button.

3. You will be directed to your profile page and will be able to notice that the recipe is removed from there.

Scenario 7: User wants to view comments that he/she made on a Recipe posted by someone

1. Click on Timeline from home page

2. Click on the view button for the recipe on which you had previously made comments.

3. You will be directed to recipe details page of that particular recipe where you can see your comments.

Scenario 8: User wants to edit comments that he/she made on a Recipe posted by themselves.

1. Click on My Recipes button on homepage
2. Click on view recipe button for a specific recipe
3. Below the comments you can see an Edit comment button, click it.
4. Make necessary changes and click the post button
5. You have now edited your comment and can see the changes being reflected

Scenario 9: User wants to delete comments that he/she or other users made on a Recipe posted by themselves.

1. Perform until step 2 of Scenario 8
2. Click on delete comment button below the comment you want gone

User Scenario 10: Optimistic Offline Lock

1. Login as Umut or Savan
2. Password is Umut or Savan respectively
3. Go to My Recipes
4. Choose one Recipe
5. Click on Edit the Recipe button
6. **NOW OPEN ANOTHER BROWSER** without closing the first one and Login as an admin account: ID Eduardo/Maria, Password: Eduardo/Maria respectively or any user that you created as admin
7. Click on Timeline button.
8. Choose the same recipe that you selected in other account
9. Click on edit and make some edits
10. Click on Update button
11. Go back to previous user in the other browser, make some change and click on Update Button, the system won't allow it

User Scenario 11: Implicit Lock

1. Login as Umut or Savan
2. Click on My Recipes
3. Choose one of them
4. Click on the "Edit Comment" button of one of the comments
5. **NOW OPEN ANOTHER BROWSER** without closing the first one and Login as an admin

6. Click on Timeline button.
7. Choose the same recipe
8. Click on the same edit button, System won't allow it

User Scenario 12: See all recipes as Admin

1. Login as admin using admin credentials mentioned above or create your own admin account as mentioned in User Scenario 1.
2. Click on Timeline button.
3. Here you can see Recipes posted by all users.

User Scenario 13: Edit Recipes of other users

1. Complete until Step 2 of User Scenario 11.
2. Click View button on a specific recipe.
3. Click Edit Recipe button
4. Complete the form and Recipe is edited.

User Scenario 14: Delete Recipes of other users

1. Complete until Step 2 of User Scenario 13
2. Click on Delete Recipe button
3. Recipe will be Deleted

User Scenario 15: Edit Comments of other users.

1. Complete until Step 2 of User Scenario 13
2. Click on Edit Comment button
3. Click Post button after editing the comments

User Scenario 16: Delete Comments of other users.

1. Complete until step 2 of user scenario 13
2. Click Delete comment button

High Level Architecture

One of the things which makes a computer program better is separation of concerns which states that a program is divided into separate sections and each section deals with a separate functionality or a concern. A concern can be as simple as a hardware specification or any complex code which results in some kind of functionality in our program. It eliminates dependency and allows developers a higher level of freedom in terms of making any changes in the code.

In our code we are following the architectural pattern shown is the figure above, the presentation layer handles all the JSP which is responsible for rendering the UI, servlets are handled by service layer and DB is handled by data source layer. There is not a single servlet which communicates directly with DataSource, it has to pass through Domain layer. So, in addition to its duties related to the logic of the program, domain layer acts as an intermediate between service and data source layer. The service layer also allows us to connect any external application with our program.

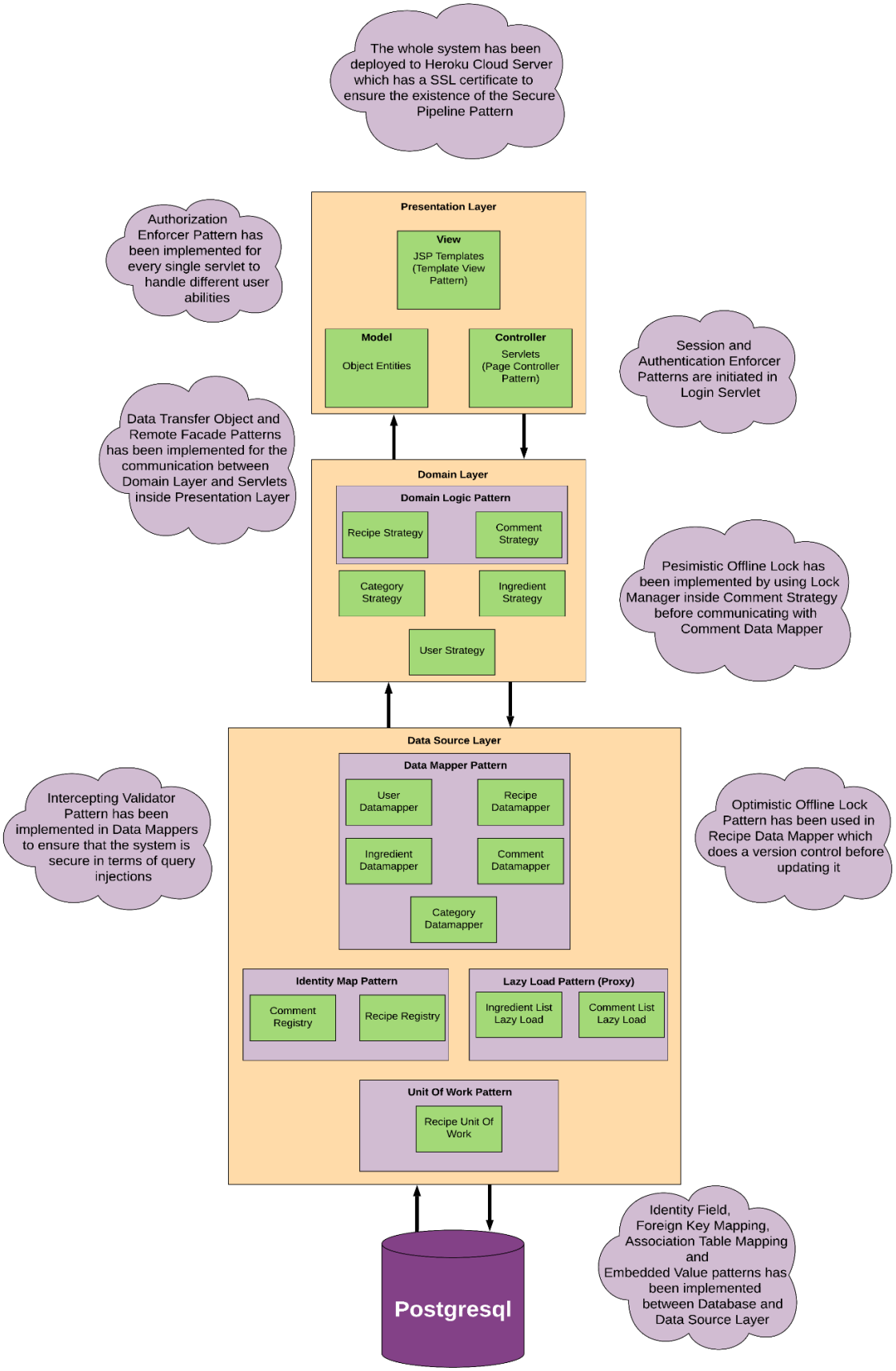

Figure 1High Level Architecture

Explanation of Patterns

Service Layer

For implementing the service layer, we aimed to make the domain layer and presentation layer separate from each other so that we can not only handle separation of concerns but also give external applications an opportunity to use our program starting from Domain Layer. Each servlet in our implementation is the member of Service Layer. The service layer is represented inside Presentation Layer as "Controller" which both handles model & view relations and communication between the Presentation Layer and Domain Layer at the same time.

In our implementation, we stored all the service layer elements which are called "servlets" under the folder controller and package servlets. Almost half of the servlets that are displayed below are the additional ones to handle the parameter transfers between JSP pages and our main servlets for the Deliverable 2. Thus, if the session feature of Java is used in the next Deliverable, we will not need these additional servlets anymore and our service layer will become more concrete.

Domain Model

During the implementation process of our project, we have used Domain Model for "Comment" and "Recipe" objects. Since basically both of these classes should have a very similar structure but different behaviours, we aimed to benefit from the advantages of this pattern. Although we established a very concrete common structure for our most important objects, their different behaviours remained the same due to the Domain Model design pattern.

The Domain Model implementation which is explained in the previous paragraph can be found under domainmodel package of controller folder in our project. To keep their structures very similar, we benefited from an interface called "Post Strategy" and make "Comment" and "Recipe" to implement it. Each common method in these two classes are responsible for completely different things. For example, addPost() function in Recipe class creates a recipe and addPost() function in Comment is responsible for adding a new comment to a given recipe.

Since our application will be complex in terms of implementation, it can be stated that using Transaction Script may not be enough to handle all the features efficiently. Additionally, as an enterprise application which can be considered as a social media for recipes, it can be concluded that using Table Module can create problems related to scalability in the future.

Data Mapper

In the implementation of our enterprise application, we benefited from Data Mapper pattern to separate the object models in Domain Layer from the database entities. For "Comment", "Recipe", "Category" and "Ingredient" models, we have separate data mappers which establish the connection between these models and the related database tables for them.

The Data Mapper implementation in our project can be found under the datamapper package of datasource folder. Our Data Mapper classes are:

- CommentDataMaper (Comment Class → comments table in database)
- RecipeDataMapper (Recipe Class → recipes table in database)
- CategoryDataMapper (Category Class → categories table in database)
- IngredientDataMapper (Ingredient Class → ingredientsOfRecipes table in database)

In our application, there is not a single Domain Layer component which interacts with database without communicating any Data Mapper classes.

Since we preferred to use Domain Model design pattern in Domain Layer, it can be stated that using Table Data Gateway could create some complexity related to communication with domain model

component. Additionally, for our application, it is aimed that we will have numerous users and numerous posts. Thus, using Table Data Gateway can be a problem about scalability.

Due to the fact that we will use Identity Map as well, it can be stated that using Data Mapper will be more convenient than using Row Data Gateway.

Lazy Load

In our implementation, we benefited from Lazy Load design pattern to make our system only willing to retrieve the data that it needs. Since our recipes include ingredients and comments which are not required to be displayed in the timeline table until the user clicks on "View" button, the list of comments and ingredients are loaded in the second step to not retrieve these data when our application does not really need.

The Lazy Load implementations can be found under lazyload package of datasource folder. We used Proxy implementation for both comments and ingredients. Since our implementation is very dependent on models, we were hesitant to modify our models with null values to work on other lazy load implementations. In contrast to other options, by using Proxy Implementation, we can easily activate or deactivate our lazy load behavior.

Identity Map

By implementing Identity Map in the implementation of our data source layer, we aimed to decrease the number of database interactions by not reading the data that we already retrieved before.

The Identity Map implementation artifacts can be found under the identity package of datasource folder. For "Comment" and "Recipe" objects we have used "CommentRegistry" and "RecipeRegistry" classes. By working together with Data Mappers, these two classes checks whether the given object has already been retrieved or not. If the answer is yes, the data is not retrieved again the database but if the answer is no, a query from the Data Mapper is sent to the database to retrieve this particular object.

Identity Field

Identity Field is the design pattern which we used most during our development process. In all of the models, we have benefited from primary keys of tables to track the objects by checking their unique ids.

The implementation of identity field pattern can be found in all the models and all the program flow is directly dependent on these unique model values.

Foreign Key Mapping

In our implementation, we benefited from Foreign Key Mapping design pattern for only the relation between Recipe and Category entities. Since categoryId attribute in Recipe object is originated from Category Object, the connection between these objects and their tables have been established by using Foreign Key Mapping design pattern.

Our Foreign Key Mapping design pattern implementation can be observed through Recipe &Category objects and recipes & categories database tables.

Association Table Mapping

Association Table Mapping design pattern provides us an option to retrieve all the many to many relations between ingredient & recipes tables and relatively Ingredient & Recipe objects. Owing to this design pattern, we are able to retrieve all the data related to ingredients of recipes with only one query.

Our Association Table Mapping design pattern can be observed through both data table "ingredientsOfRecipes" and "IngredientDataMapper" class. In "IngredientDataMapper" class, ingredient list data is retrieved data from ingredientsOfRecipes" and sent back to Recipe object.

Embedded Value

By using Embedded Value design pattern, the Recipe Object has been established as a composite structure which consists of ingredient list from "Ingredient" class and category from "Category" class.

To create a full Recipe Object, it is required to embed an ingredient list into Recipe constructor as a parameter. By embedding this value, we aimed to establish some connection between Recipe and Ingredient classes with an ownership relation.
In our implementation, this pattern can be found in Recipe class.

Input Controller

In order to make the communication between User Interface components and servlets as clear as possible, we have implemented Page Controller Pattern for each submit button in the jsp files. By benefiting from this pattern, we achieved a very traceable and easy-to-understand structure which decouples every process from each other. Therefore, we have preferred to choose Page Controller instead of the other Input Controller patterns.

In our implementation, each submit button which is located in a jsp file communicates with a particular servlet (each servlet name is reflecting which button has been clicked to initiate it). Based on the action that the servlet is responsible, the program flow is redirected to domain layer.

Our servlets can be found under controller.servlets package and our jsp files can be found under WebContent folder.

An example to show a part of this pattern in terms of class diagram can be found at **Figure 6**

View Controller

For structuring the View Components in the Presentation Layer, we have used Template View Pattern by preparing all the jsp file templates. By using Template View pattern, we aimed to achieve a structured presentation layer with a simple implementation process.

Each jsp file has its own markers which the servlets fill inside them based on the action performed. Since we do not have a very dynamic data to be displayed, generating the content of jsp files from servlets just like in the other View Controller Patterns was not a very necessary feature for our application. Additionally, these other patterns can decrease the planned simplicity of our presentation layer.

Our servlets can be found under controller.servlets package and our jsp files can be found under WebContent folder.

An example to show a part of this pattern in terms of class diagram can be found at **Figure 6**

Session Management

In our session management strategy, we have implemented Client Side Session Pattern by using the AppSession.java (works collaboratively with Shiro) along with the URL parameters. Owing to this strategy, we have tracked the state of the authenticated user along with the other required parameters easily.

The reason behind why we did not prefer using server side or database side session management is directly related to unreliable performance of the server and database deployment space that we are currently using. Since our enterprise system should not be affected by the lack of reliable performance, we aimed to give the responsibility of session management to client side.

The session configuration classes in our implementation can be found under config.session package and our session usage can be seen in all the jsp files and servlets.

Concurrency

Since Admin type and Normal type users are both able to access the same data source such as Recipe and Comment, the concurrency problem is an important one to be handled carefully.

For “Recipe Edit Use Case”, we have have benefited from Optimistic Offline Lock which lets both users to be able to reach the editing page for the same Recipe but allows only one of them to update. This action is controlled by putting a version column in the “Recipe” table and updating the model accordingly. In this case, if one of the users changes some data on the Recipe object, the other will not be able to change update the same recipe if s/he clicks on “Update” button. This use case can be tested by opening the application from two different browsers and try to edit the same Recipe at the same time as an Admin and as a Normal User.

The implementation of this pattern in our repo can be found in RecipeDataMapper,java → read function.

The class diagram to show this pattern can be found at **Figure 16**
Sequence Diagram at Figure 22

For “Comment Edit Use Case”, we have used Implicit Lock Pattern which is the combination of Pessimistic Offline Lock and Exclusive Read Lock. In this strategy, only the first user (admin or normal) is able to access the editing page for a particular comment object. If the second user wants to access the editing page for the same comment, s/he won’t be allowed to do it. This use case can be tested by opening the application from two different browsers and try to edit the same Recipe at the same time as an Admin and as a Normal User.

The implementation of this pattern can be found under controller.lock and CommentDataMapper.java → update function.

The class diagram to show this pattern can be found at **Figure 13**

By doing that, we have used all the major lock patterns in our implementation.

Security

In order to implement some Security patterns, we have used Apache Shiro as an external library which works collaboratively with session management of our application. The AppSession class which was configured based on the Shiro settings keep two attributes which are Username and User Role. If one of these attributes are missing and the user aims to reach an internal page in the app, s/he will be redirected to Login Page. By doing that we ensured the existence of authentication enforcer pattern. After each successful login request, the user is redirected to the pages which s/he has to see based on his/her User Role attribute in AppSession. By doing that we ensured the existence of authorization enforcer pattern.

The configuration files of these patterns can be found under config.security and config.session packages in our repo.

Owing to these patterns, it can be stated that there is not a single user who has an access scope which s/he is not supposed to have. The class diagram to show these patterns can be found at **Figure 11 & 12**

Since SQL Injection is one of the most critical problems of such enterprise systems, we have implemented Valid Interceptor Pattern to decrease the risk of being hacked by this type of attacks. For each query parameter, we have used "?" which only accepts valid inputs. The implementation of this pattern can be found in our Data Mapper Classes.

Another critical problem can be considered as the network tracking attacks which may end up with breaching the sensitive data such as login credentials. To avoid such kind of problems, we have chosen a deployment environment (Heroku) which supports Secure Pipeline Pattern. Since Heroku has a SSL Certificate, it can be stated that each request which goes to Heroku is encrypted.

By doing that, we have used all the major security patterns in our implementation.

Distributed Systems

Since our application is an enterprise system which will provide service for at least thousands of people, the deployment structure of distributed components and the relations between each other are some of the most critical points for the performance of the application. As the most basic distribution, we aimed to distribute our components between presentation layer and domain layer so that we can have 2 separate servers which are "Web Server" and "Application Server".

In this system, we planned that our servlets which are inside the Presentation Layer will communicate with Domain Layer through Remote Facade. The Remote Facade Pattern implementation can be found under distributedsystems.facade package in our project repo. In fact, we do not use it actively in our product because our system is not really distributed to different physical spaces. Therefore, Remote Facade Pattern is not being used in the submitted product but if the deployment issues can be solved, it is really easy to integrate that pattern with rest of the implementation in order to handle the communication between servlets and domain layer components.

The class diagram to show this pattern can be found at **Figure 31**

In order to make data transfer between servlets and Domain Layer components more efficient, we have implemented Data Transfer Object Pattern to decrease the total remote call time that we spend for communication between these two sides. The DTO Pattern implementation can be found under distributedsystems.dto package in our project repo. Although we do not use remote facade pattern, DTO objects are being used in the data transfer from Domain Layer to servlets.

The class diagram to show this pattern can be found at **Figure 32**

Domain Model Diagram

A domain model diagram for our program which shows relation between all the items.

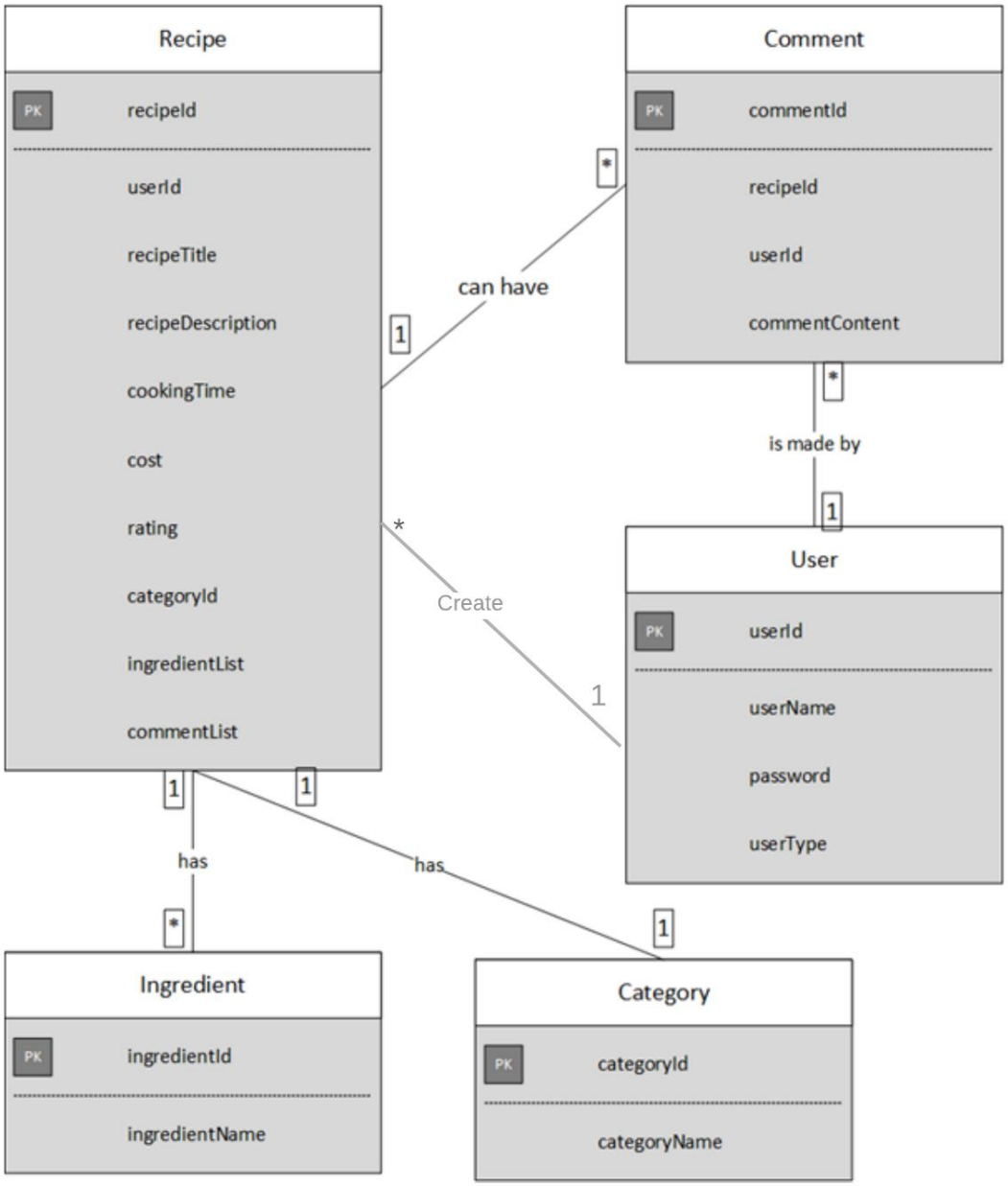

Figure 2 Domain Model

Class Diagrams

Following are the class diagrams of our product. We have separated them by their packages to make them more readable. In the next section, the connections between these components will be shown to explain the implementation flow of the application.

General Model

This shows class diagram of our general model of the system which contains all the different components

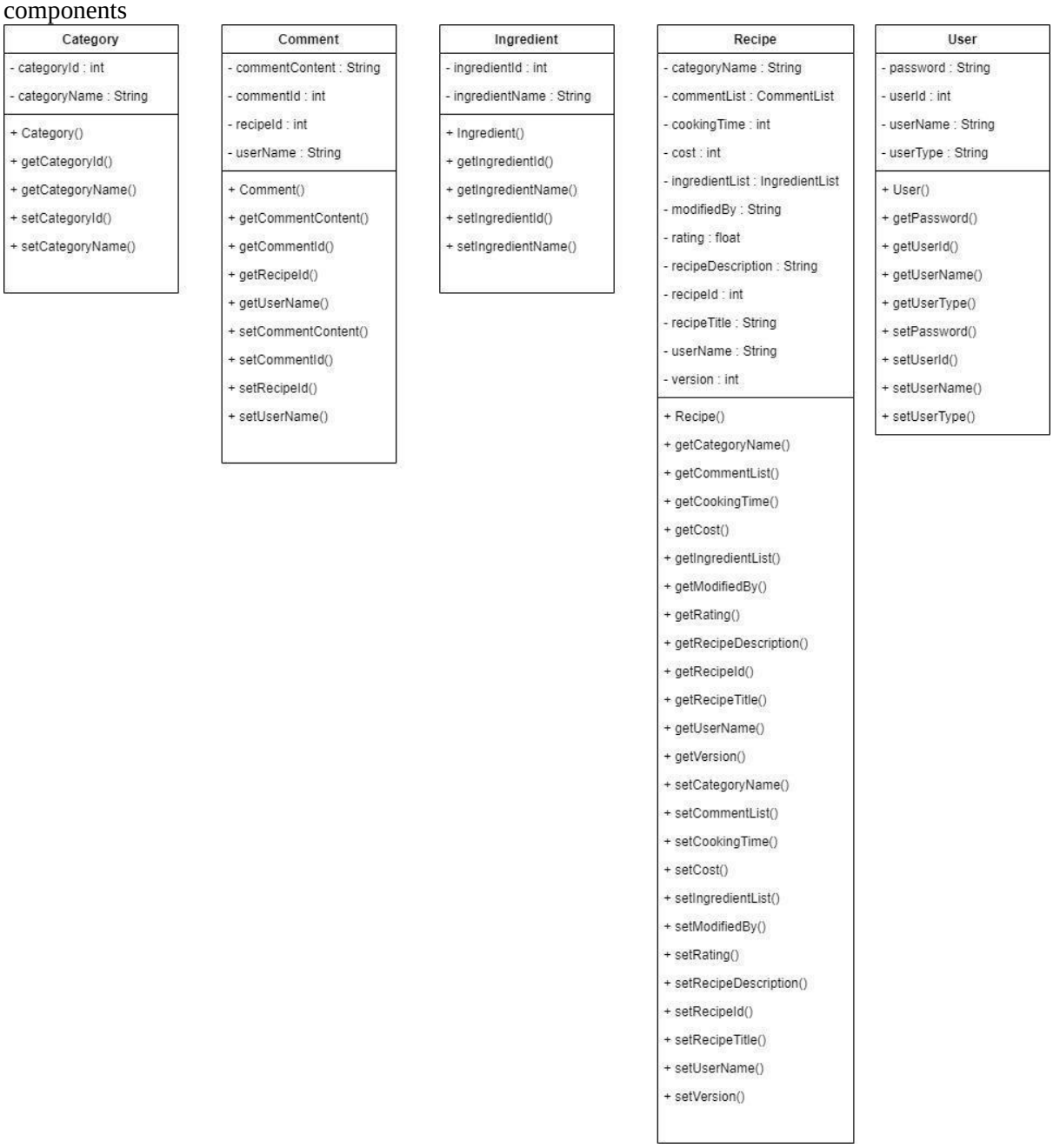

Figure 3 General Model

Servlets

This shows the class diagram for all the servlets that we have in our program

Version 1:

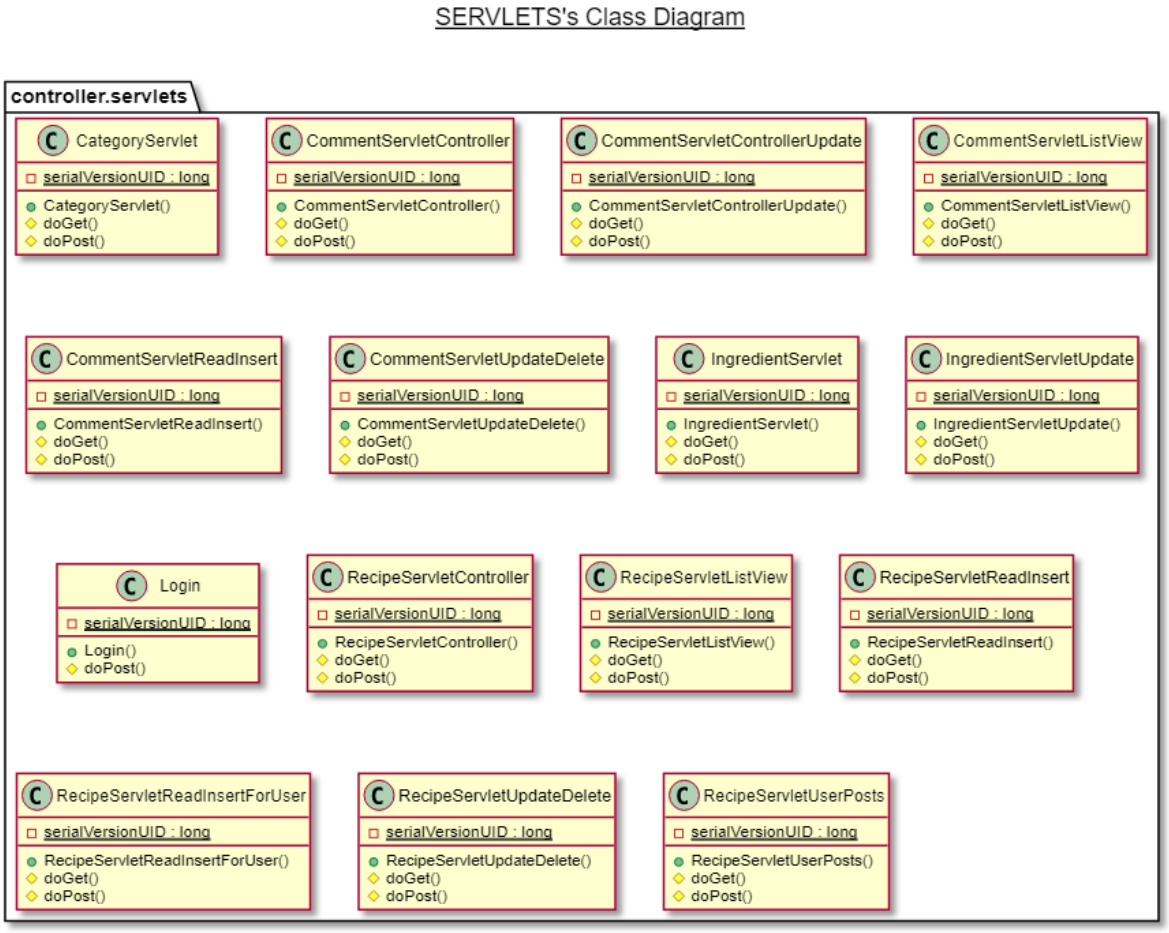

Figure 4 Old Servlets

Version 2

This diagram contains the latest reflection of our implementation of servlets in the system, each button is represented by a servlet. We have used the servlet names in other diagrams without showing the methods contained within to keep it simple.

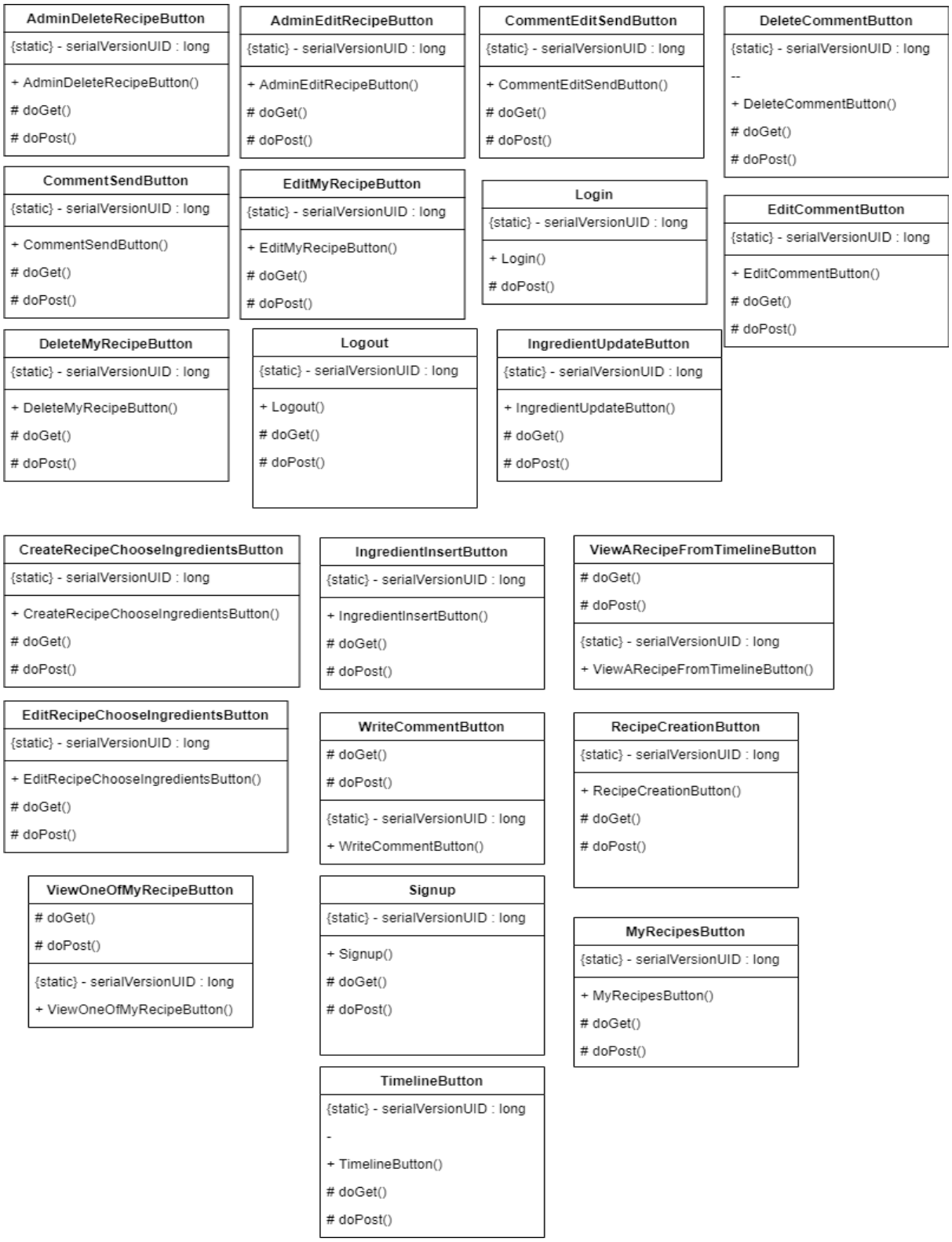

Figure 5 New Servlets

Domain Model & Bridge Combined (Controller)

This shows class diagram for Domain model and Bridge, both of which combined acts as a controller, and they use various strategies among them to get our system functional. Bridge folder is for the domain logic classes which are not being used in “Domain Model Pattern”. That is why they are separated and named as bridge because their only duty as a Domain Layer member is providing the communication between the Presentation Layer and Data Source Layer.

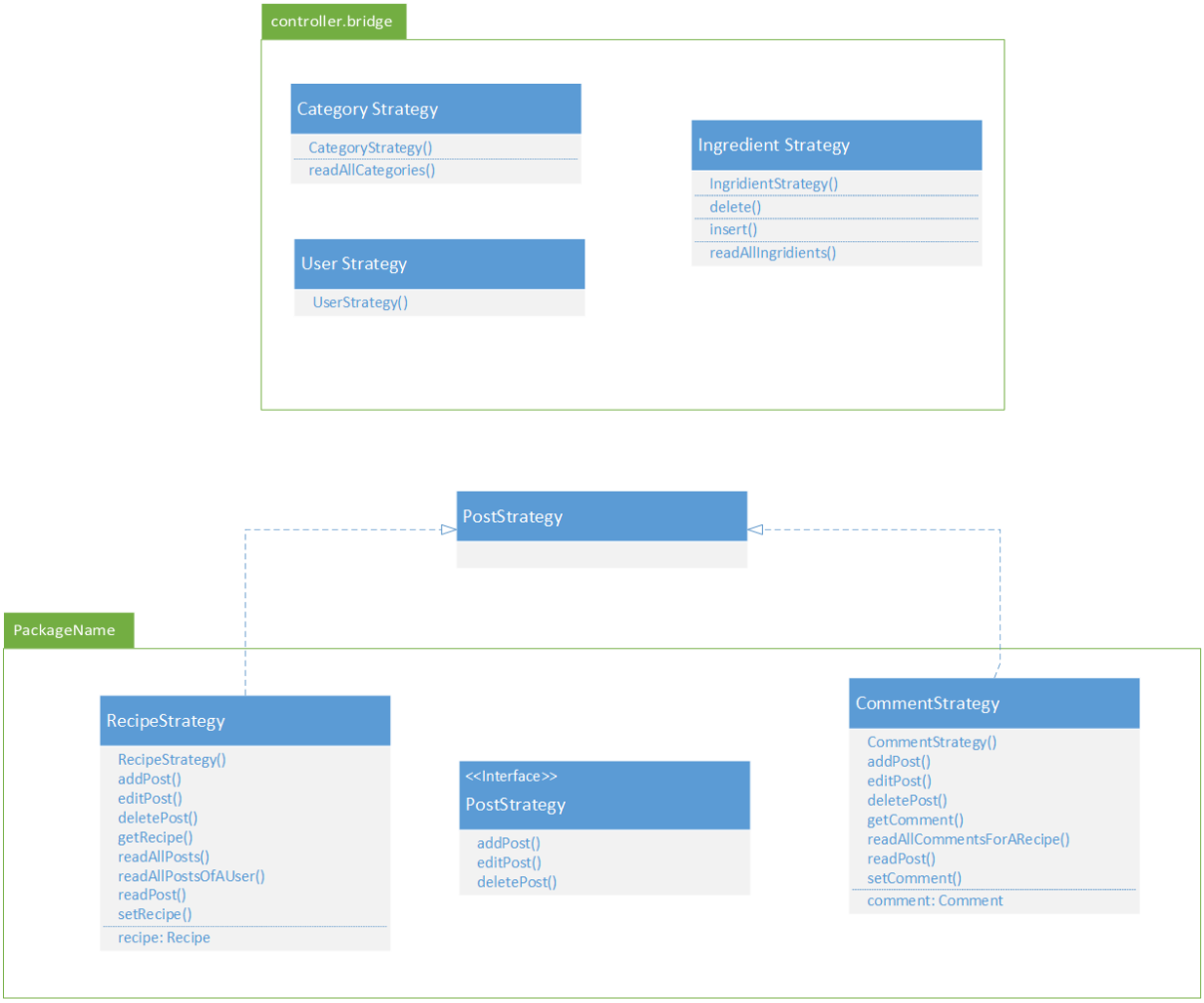

Figure 6 Controller

DataMapper

This shows class diagram for our datamapper.

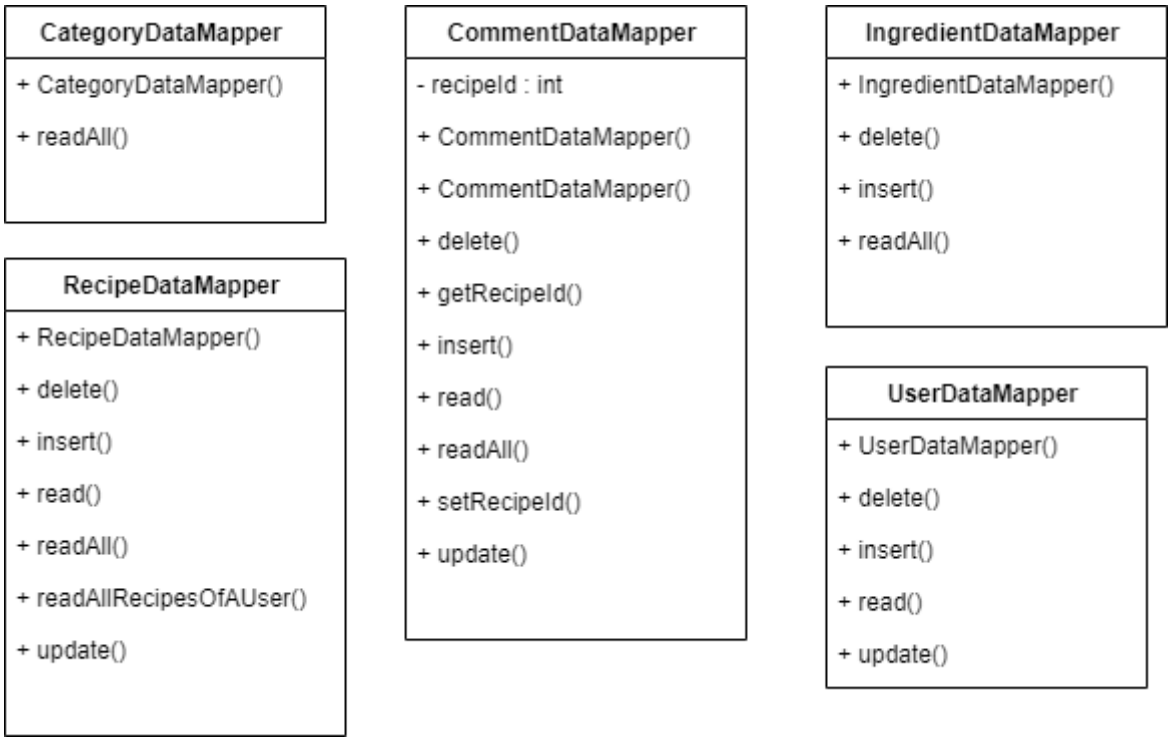

Figure 7 DataMapper

Lazy Load

This shows class diagram for our lazy load pattern. We have used Proxy Lazy Load Implementation for Ingredients and Comments of a Recipe.

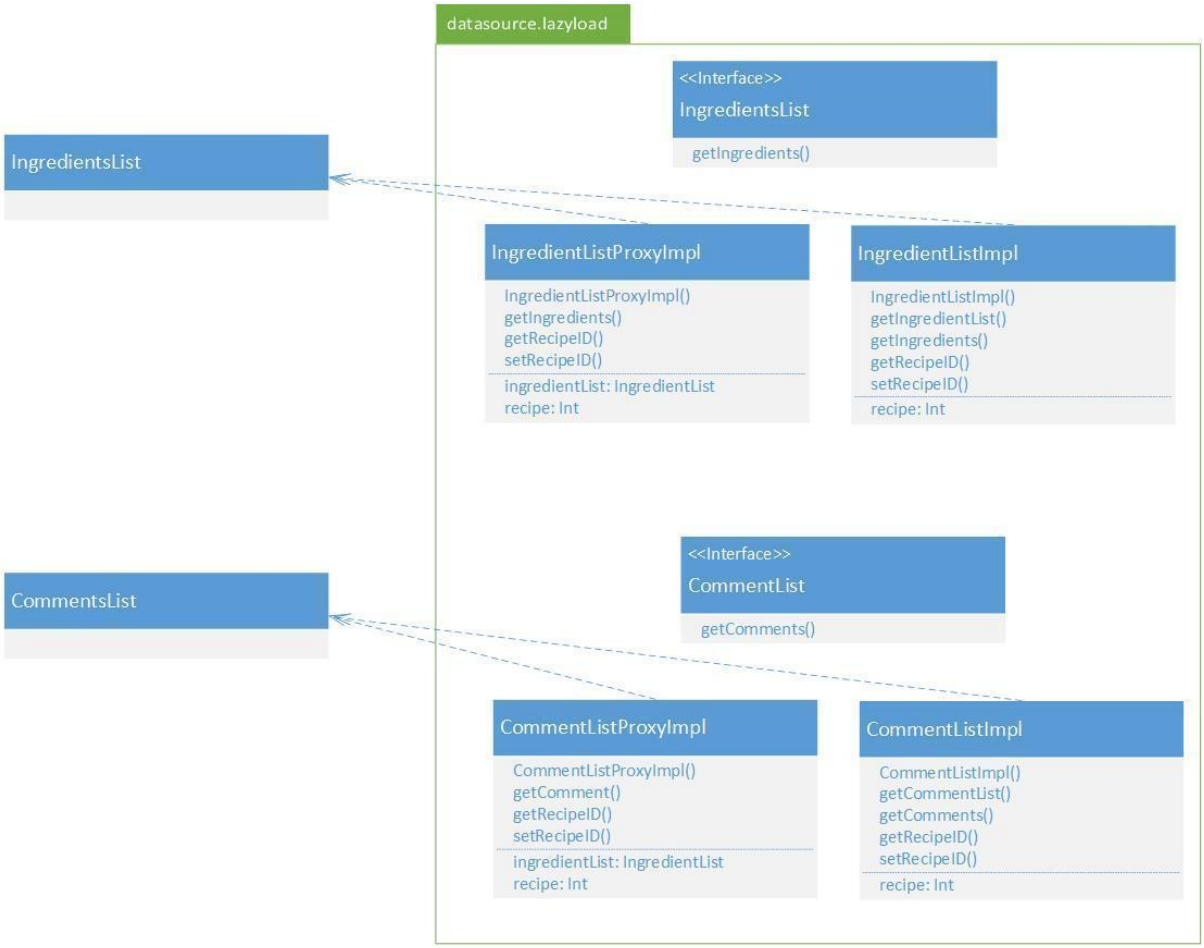

Figure 8 Lazy Load

Identity Map

This shows class diagram for Identity map. We have used identity map pattern for both Recipe and Comment objects in our implementation.

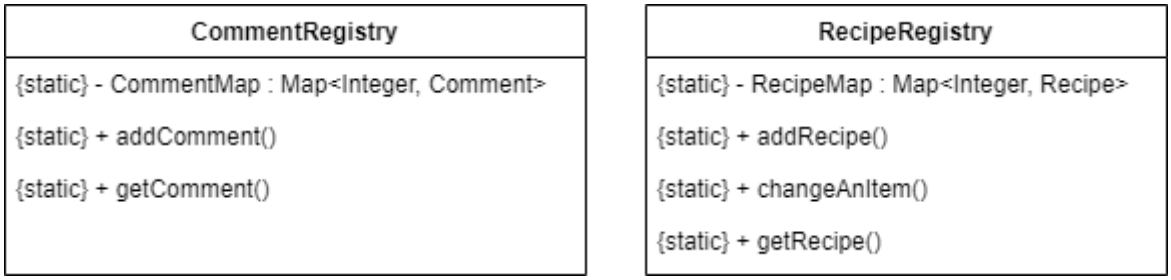

Figure 9 Identity Map

Unit of Work

This shows class diagram for Unit of Work

Figure 10 Unit of Work

Session

This diagram shows class diagram for Session

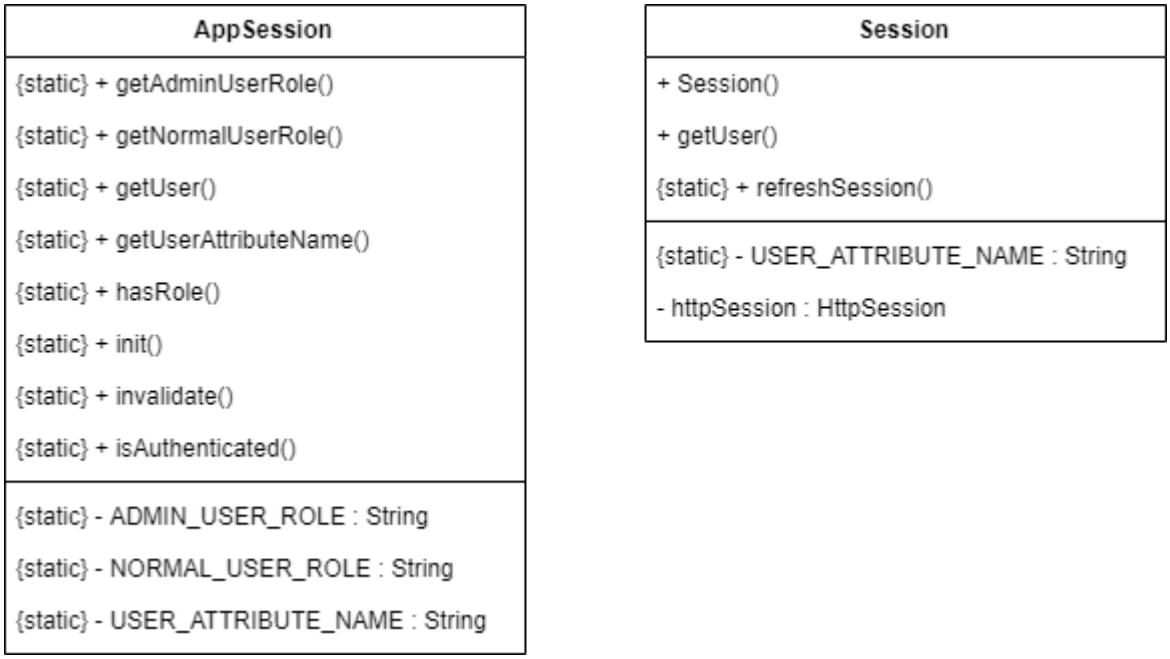

Figure 11 Session

Security

This diagram shows class of AppRealm which is for Security

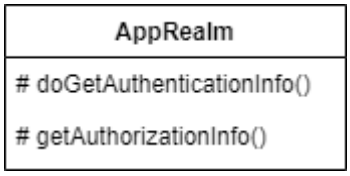

Figure 12 Security

Lock Manager

This is class diagram for our Lock which is part of Controller

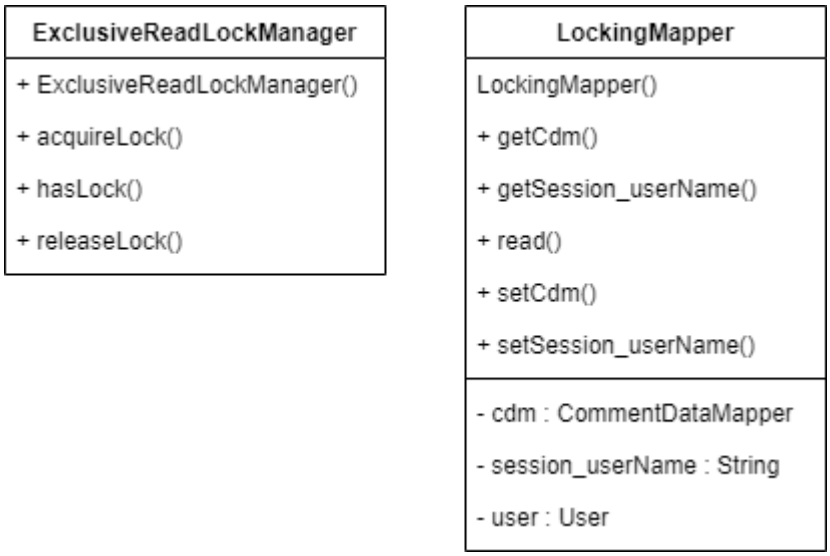

Figure 13 Lock Manager

Remote Façade

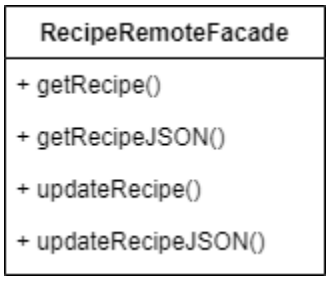

Figure 32 Remote Facade

DTO

Figure 33 DTO

Interaction of Comment Strategy with Servlets and Patterns

This diagram below shows how the actions related to comments are handled in our implementation. It also has a connection with Recipe methods because comments are associated with recipes. These servlets of Comment interacts with comment strategy and that has interactions with DataMapper, AppSession and LockingMapper.

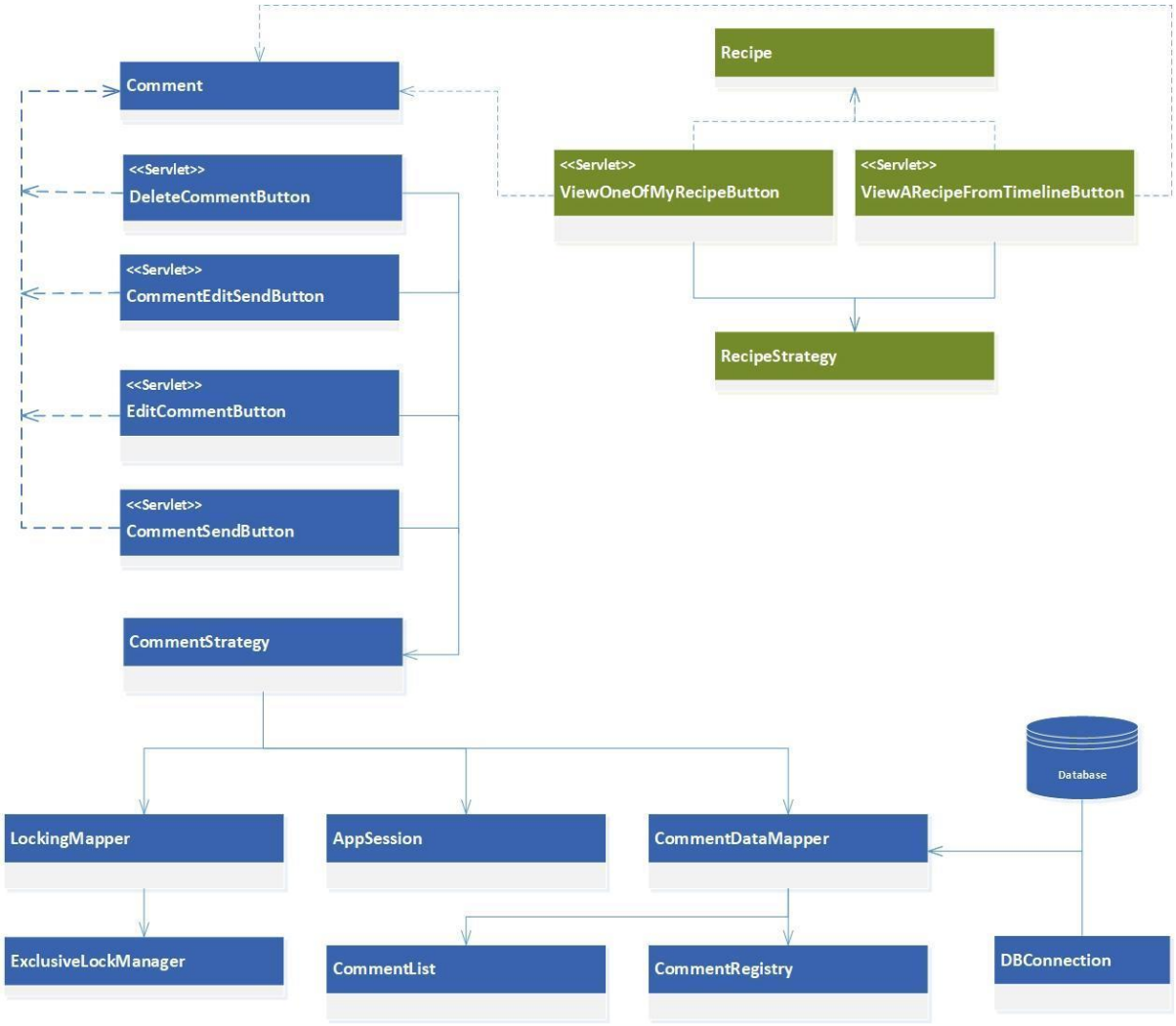

Figure 14 Comment Strategy

Interaction of Recipe Strategy with Servlets

Here we show the interaction that Recipe strategy has with various servlets, here Recipe uses all the following servlets shown in the figure, a detail on those servlets can be found in figure of servlets above

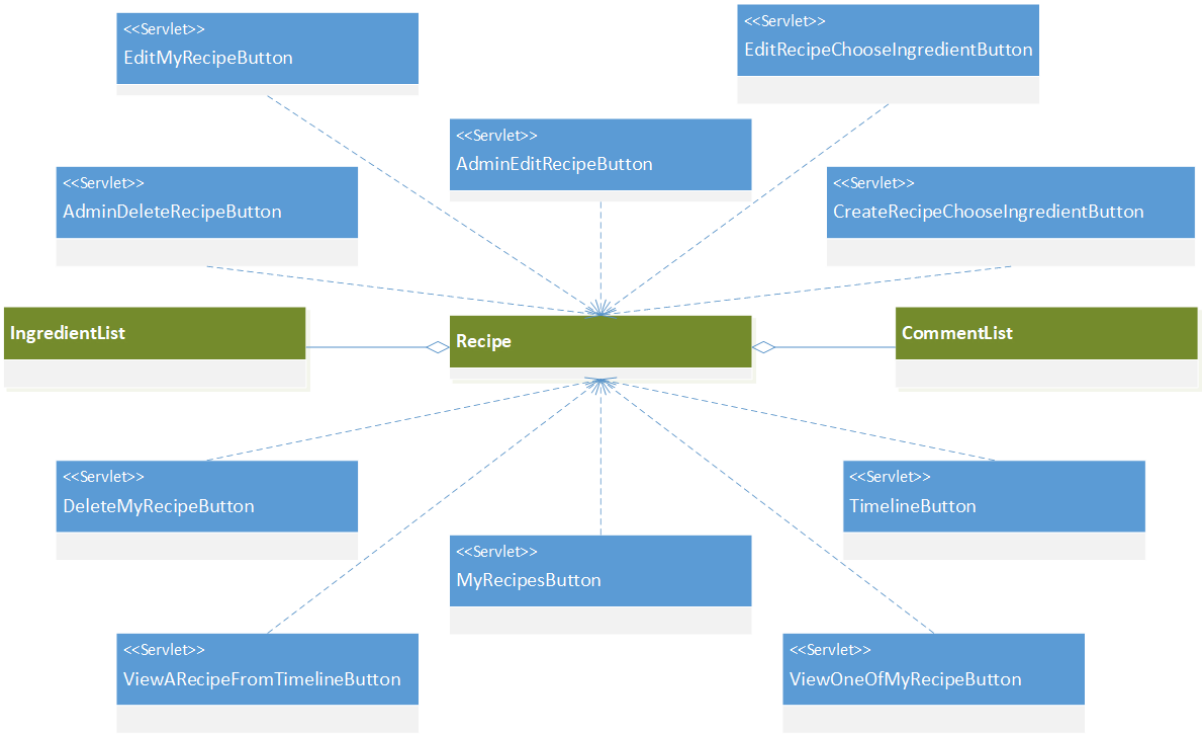

Figure 15 Recipe Strategy

Interaction of Recipe Strategy with Patterns

All the servlets in the above figure are associated with RecipeStrategy in this figure, we have kept the diagrams separate to keep figures concise. Here connection between RecipeStrategy is shown with other patterns such as DTO, Facade and DataMapper, the Datamapper is connected to a database and also uses a registry. It also has lazyload pattern which allows us to load only the specific required data on webpage.

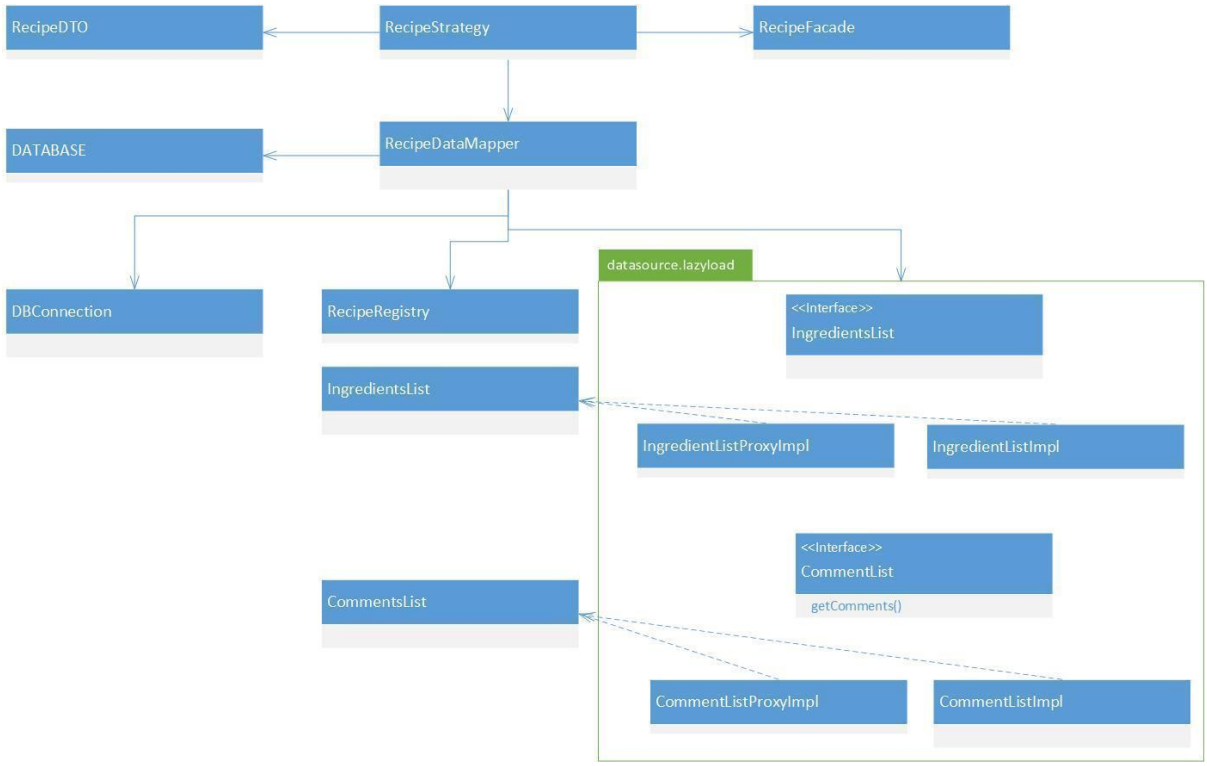

Figure 16 Recipe Strategy with Servlets

User Signup with Patterns

This figure shows how user can sign up or logout in our system. Our signup is linked to a UserStrategy which in turn is linked to a UserDataMapper and then a DBConnection. For Logout we maintain an AppSession

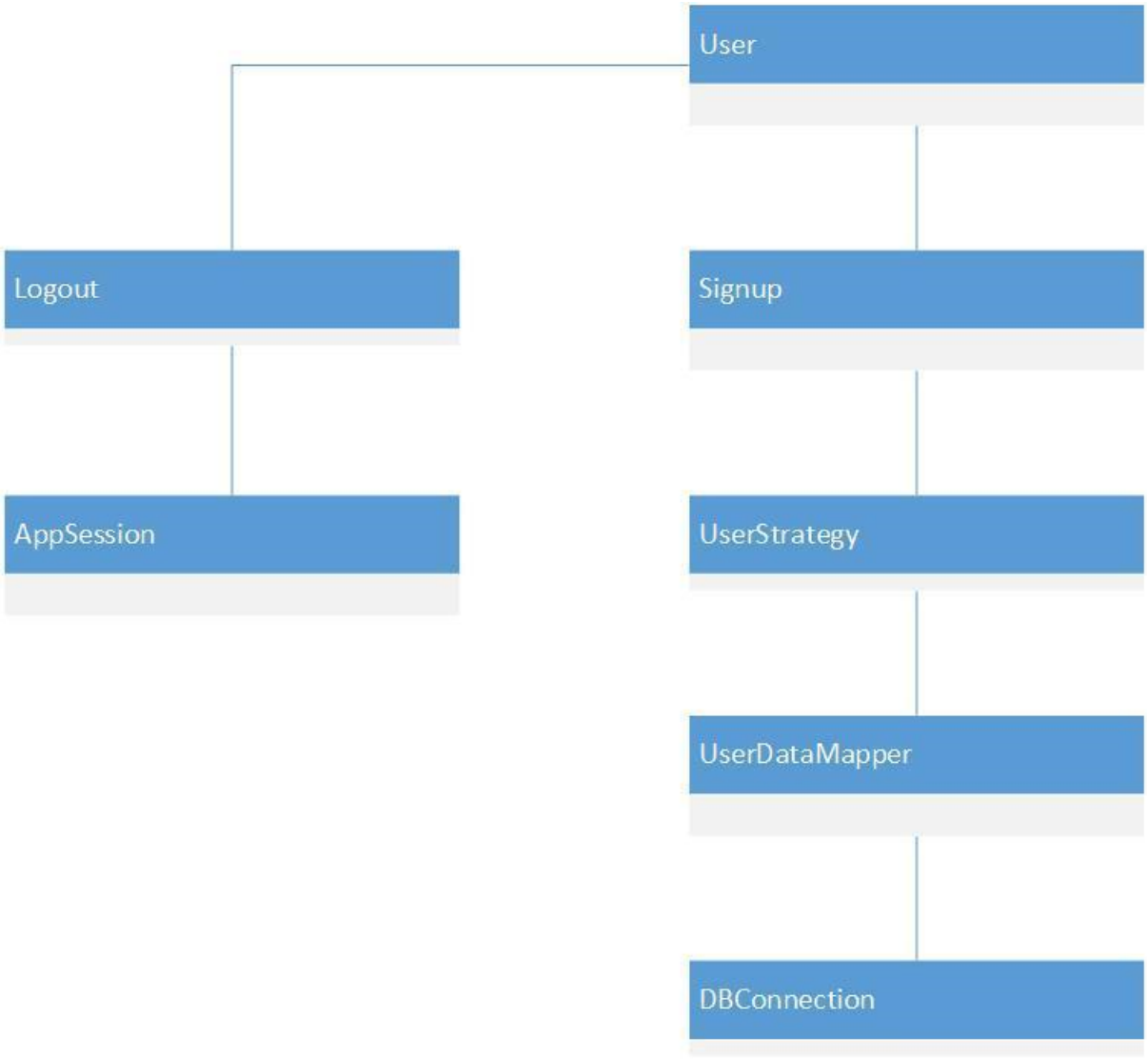

Figure 17 User Signup

User Login with Patterns

This class diagram is showing the interaction of Login with other strategies. Login has UserStrategy which has details of user, AppSession to maintain session and SecurityUtils for security package. All three are connected to UserDataMapper and that handles the communication with database.

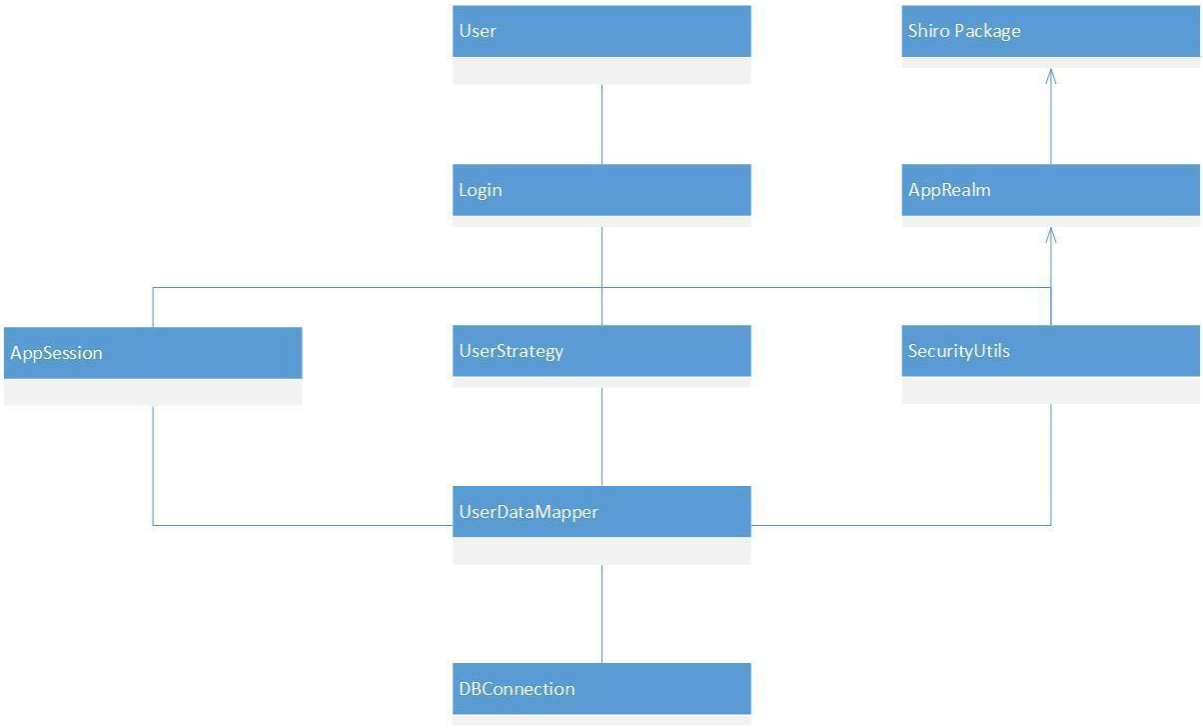

Figure 18 User Login

Component Diagram

This component diagram shows the relation between each component in our implementation. Since Presentation Layer consists of only jsp files and out of scope for this deliverable, it is not included. The Presentation Layer Components will be added in the next deliverable.

Version 1: Previous Submission

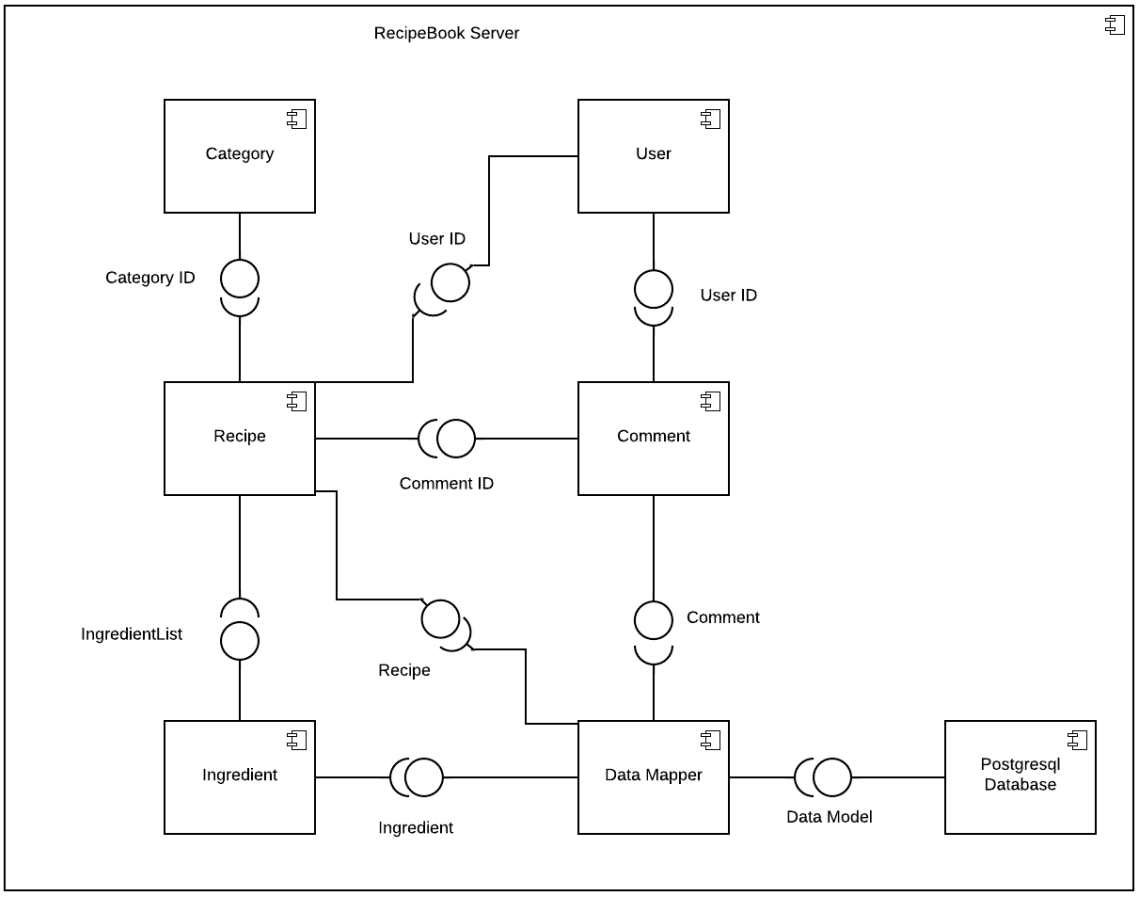

Figure 19 Old Component Diagram

Version 2: Current Submission

Following diagram shows the latest components in our system, it also shows connection between patterns.

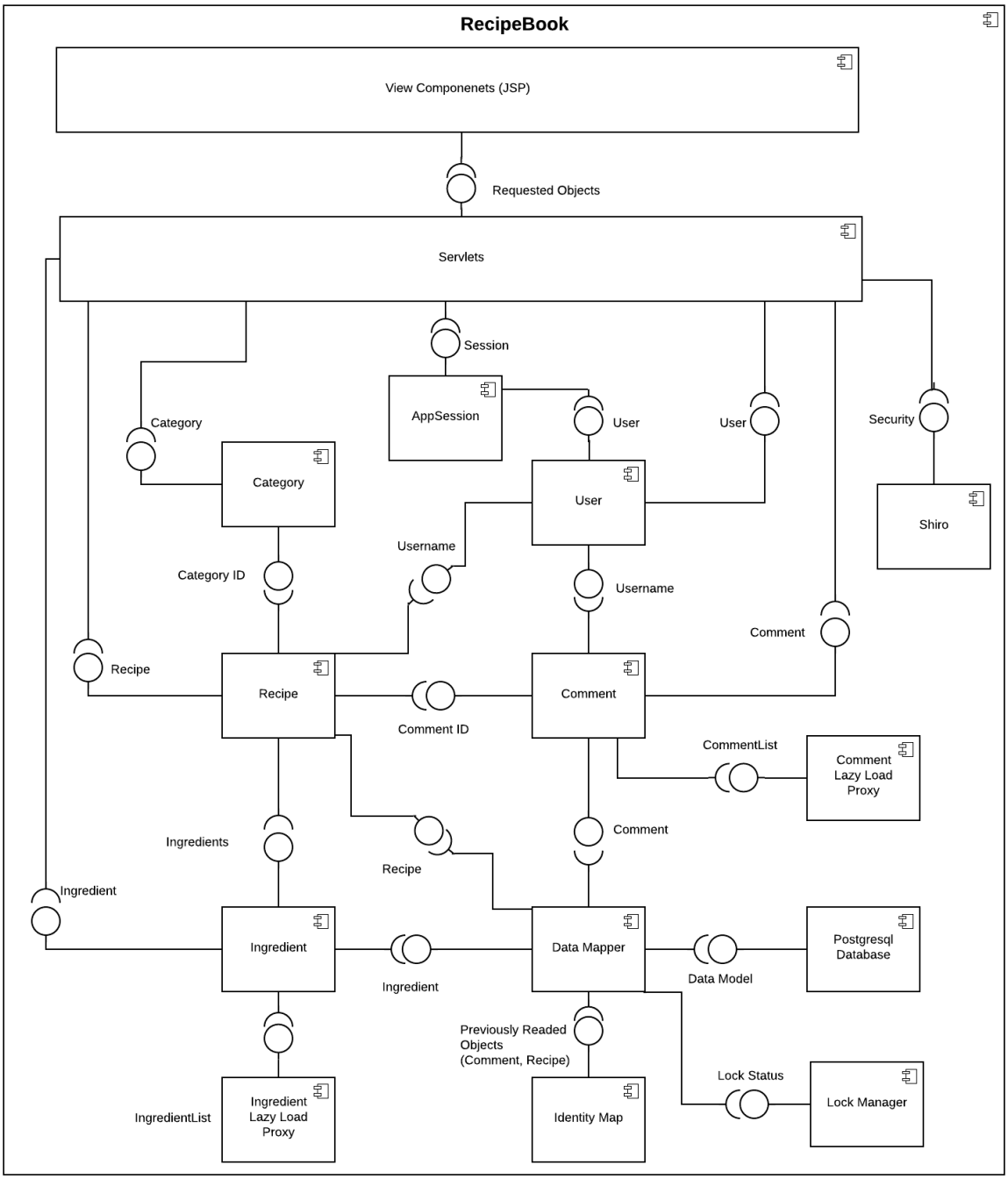

Figure 20 New Component Diagram

Interaction Diagrams

Exclusive Read Lock Manager Acquire Lock

This is ExclusiveReadLockManager for acquiring the lock, we use an object DB of DBConnection and that object interacts with methods preparedStatement and lastInsertedComment. As long as the lock is not acquired by anyone, a user can acquire it.

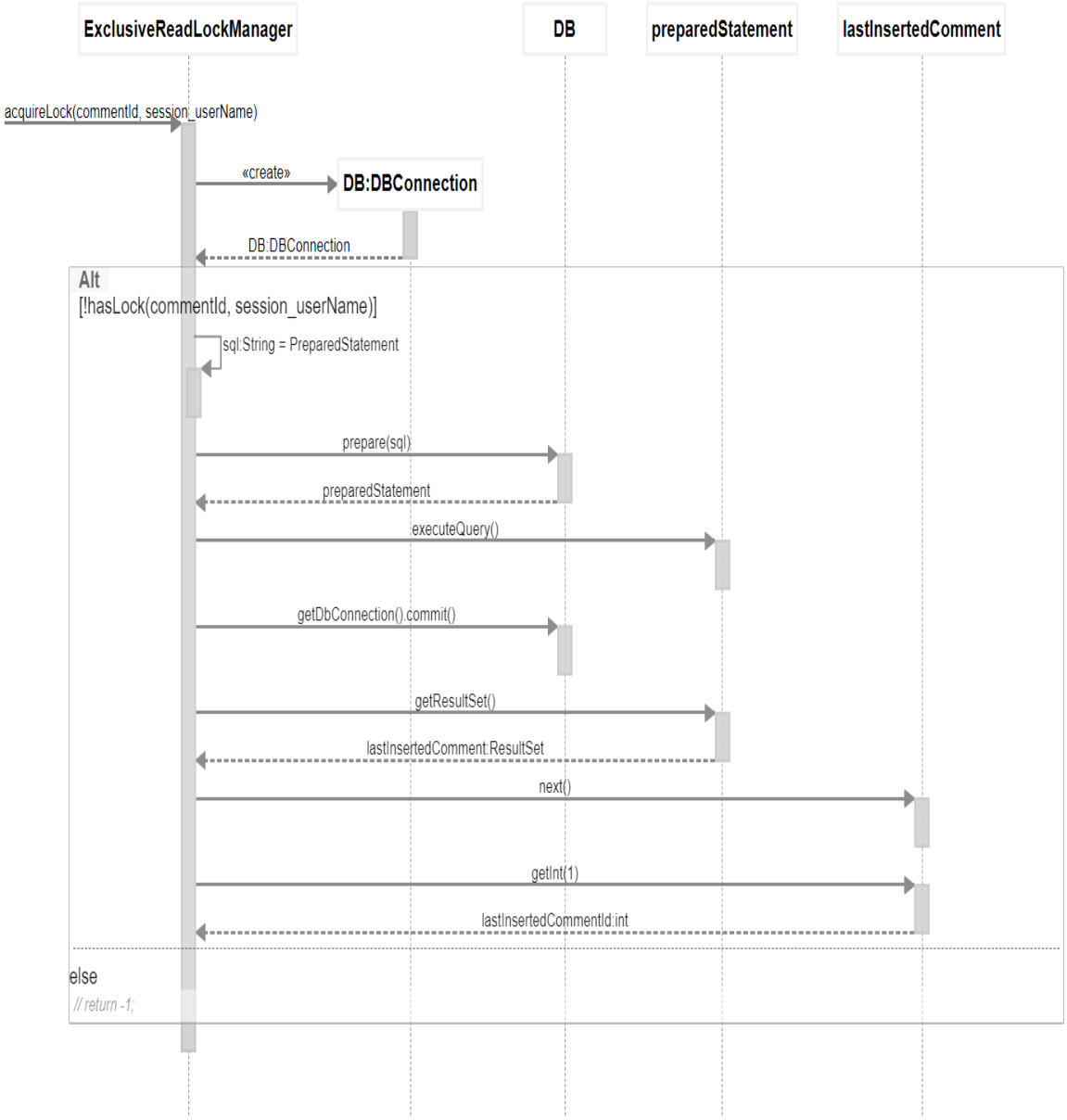

Figure 21 Exclusive Read Lock Manager Acquire Lock

Exclusive Read Lock Manager Release Lock

This is ExclusiveReadLockManager for releasing the lock, we use an object DB of DBConnection and that object interacts with methods preparedStatement and lastInsertedComment. As long as the lock is acquired by anyone, it can be released.

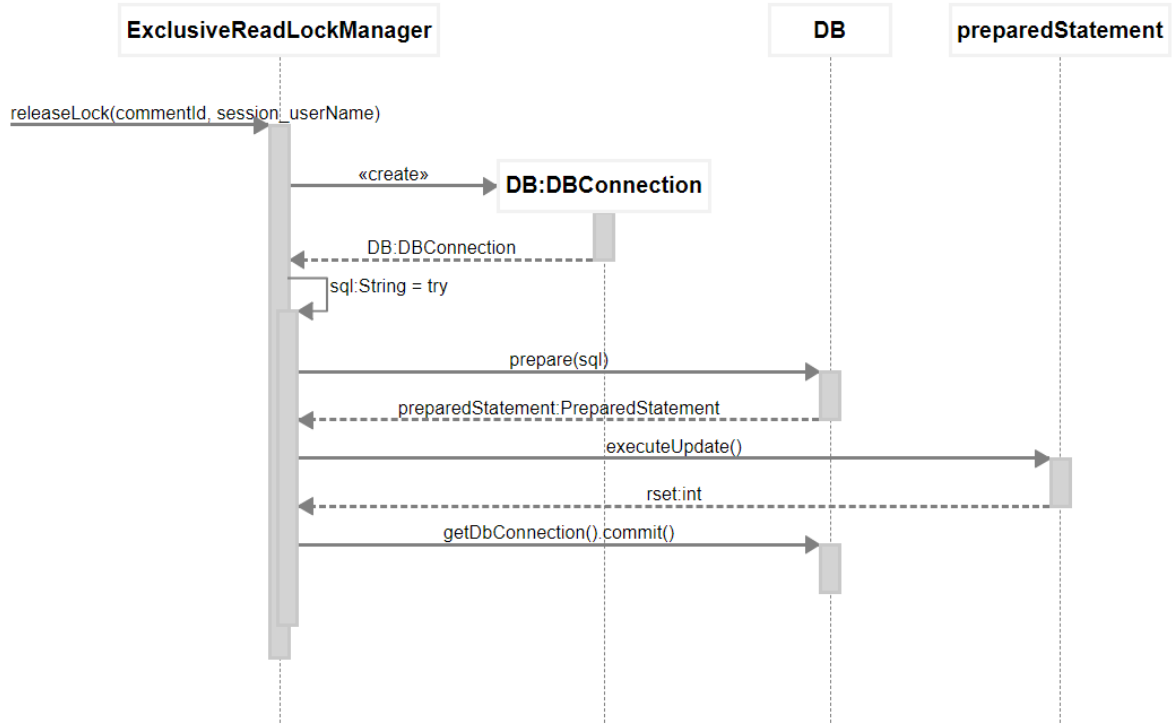

Figure 22 Exclusive Read Lock Manager release Lock

Security

Following is high level flow of Login Implementation:

login servlet —> user strategy—> user data mapper —> database —> user data mapper —> user strategy —> login servlet —> Shiro (AppRealm) —> login servlet —> AppSession —> login servlet

Until Shiro part, our strategy asks DB whether the Username and Password exists or not, if yes, then Shiro handles security and comes back to Login Servlet and AppRealm is inititated.

This shows the interaction diagram of our security implementation. Here we can see the generation of a user token and use of that token to along with a user object and user strategy.

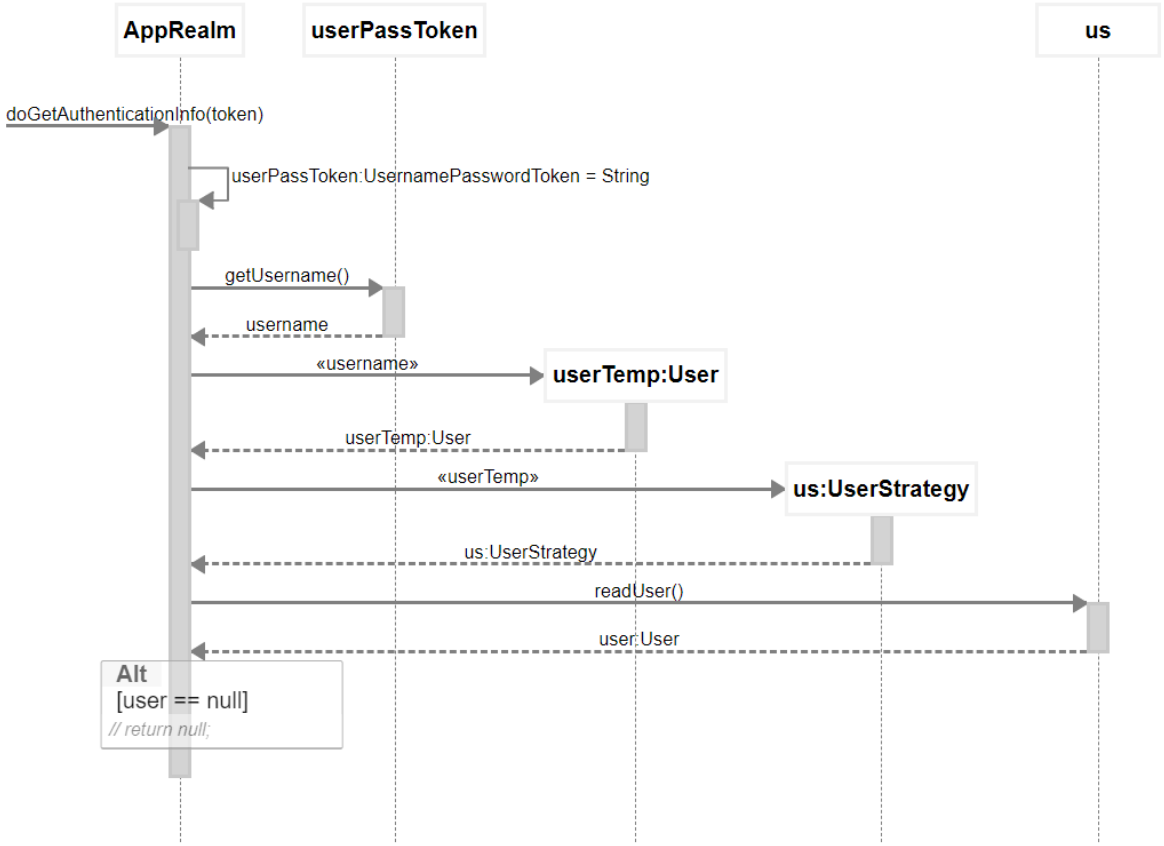

Figure 23 Security getToken

Here we see our security generating a role which is associated with a user and user strategy object called ‘us’. By using ‘us’ we try to find a user and then determine the user type which is shown by a conditional loop in diagram

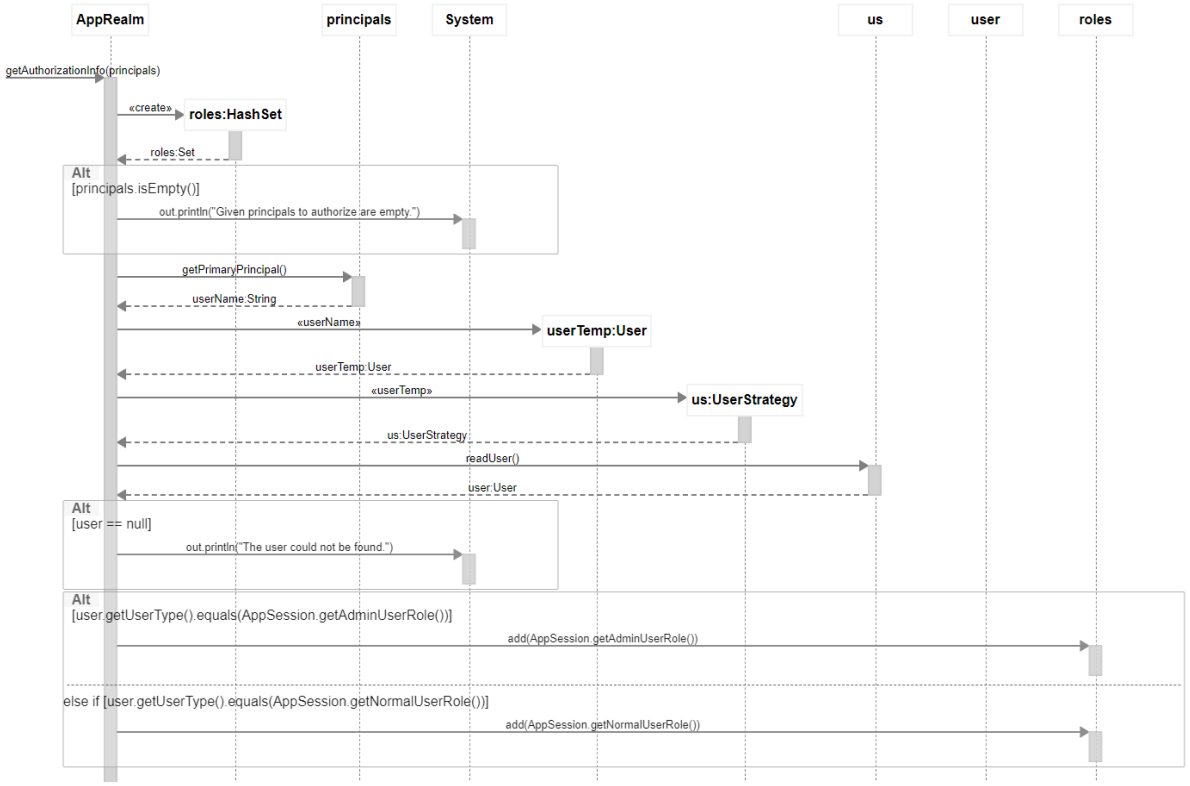

Figure 24 Security getPrinciple

Unit of Work Commit Recipe

This diagram shows implementation of unit of work with RecipeDataMapper. We create an object rdm for RecipeDataMapper which communicates with newRecipes to commit the recipes created

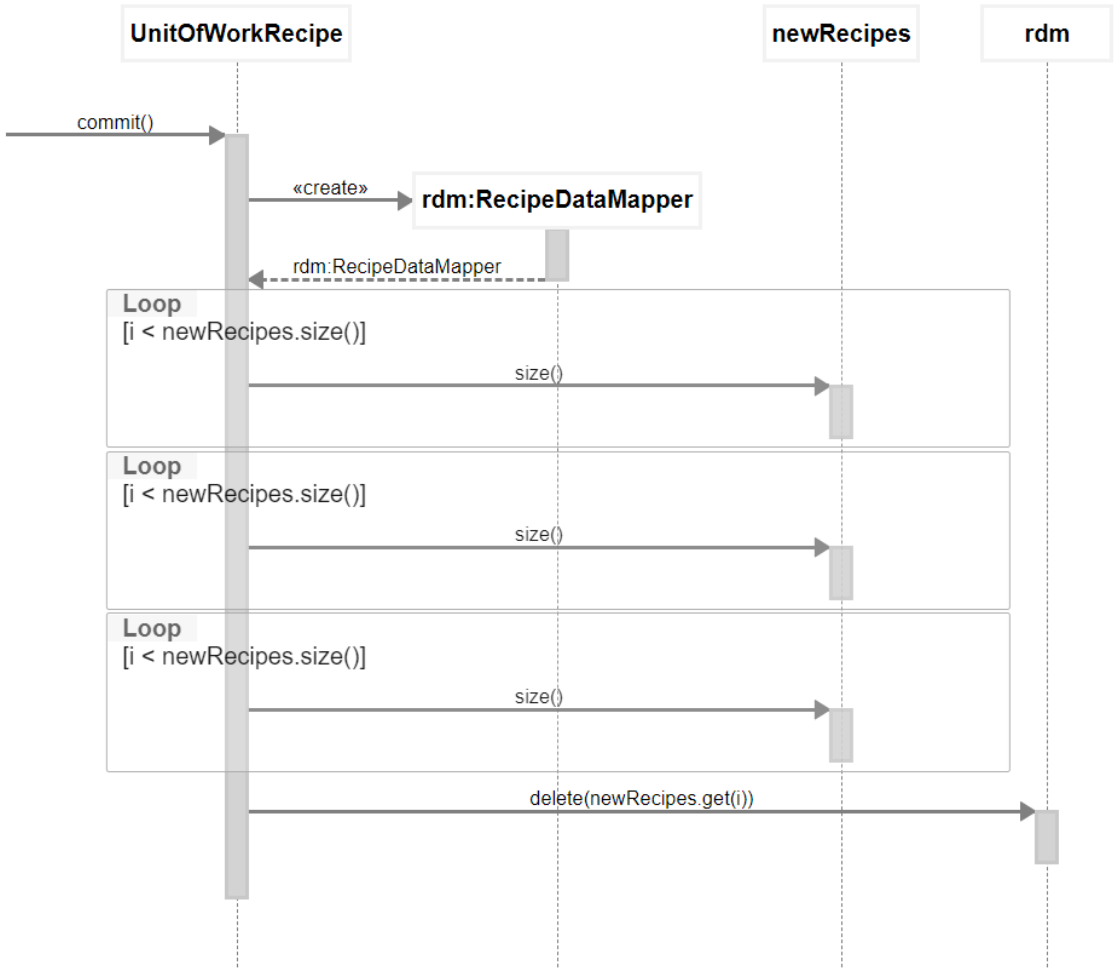

Figure 25 Unit of Work Commit

Unit of Work Delete Recipe

This diagram shows the implementation of Unit of Work for deleting the recipes.

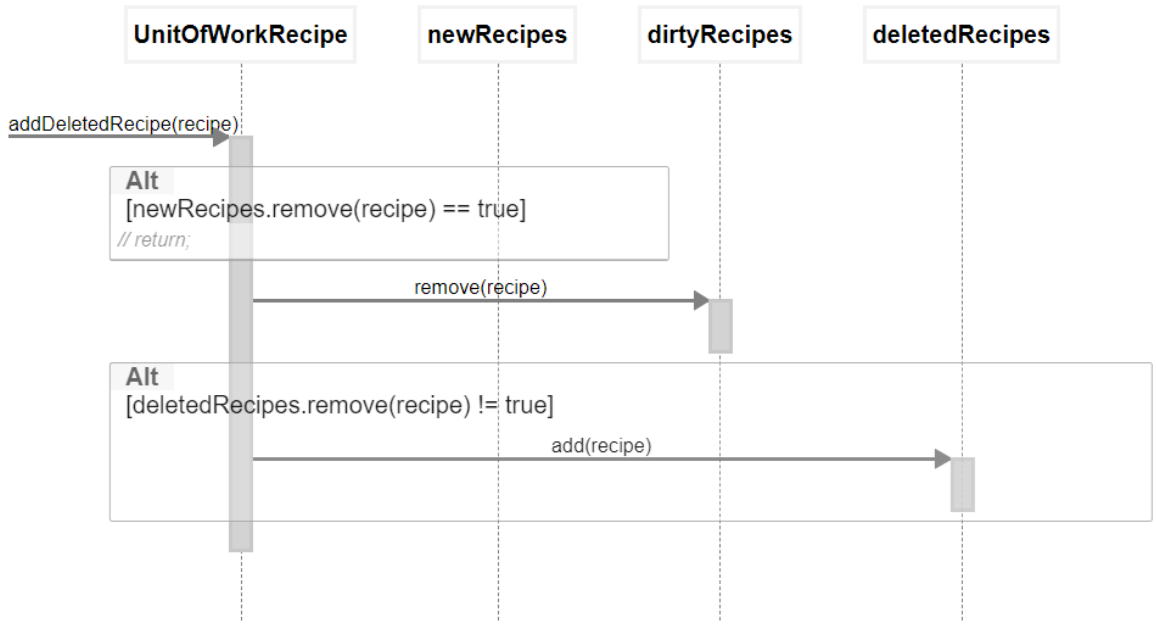

Figure 26 Unit of Work Delete

Interaction of RecipeStrategy with RecipeDataMapper

This diagram shows the interaction of RecipeStrategy with RecipeDataMapper, we have created an object rdm of RecipeDataMapper which is used throughout the figure. Our method inserting a recipe interacts with rdm and receives back an id. In the figures below, we will see the interaction that our rdm has with all other methods.

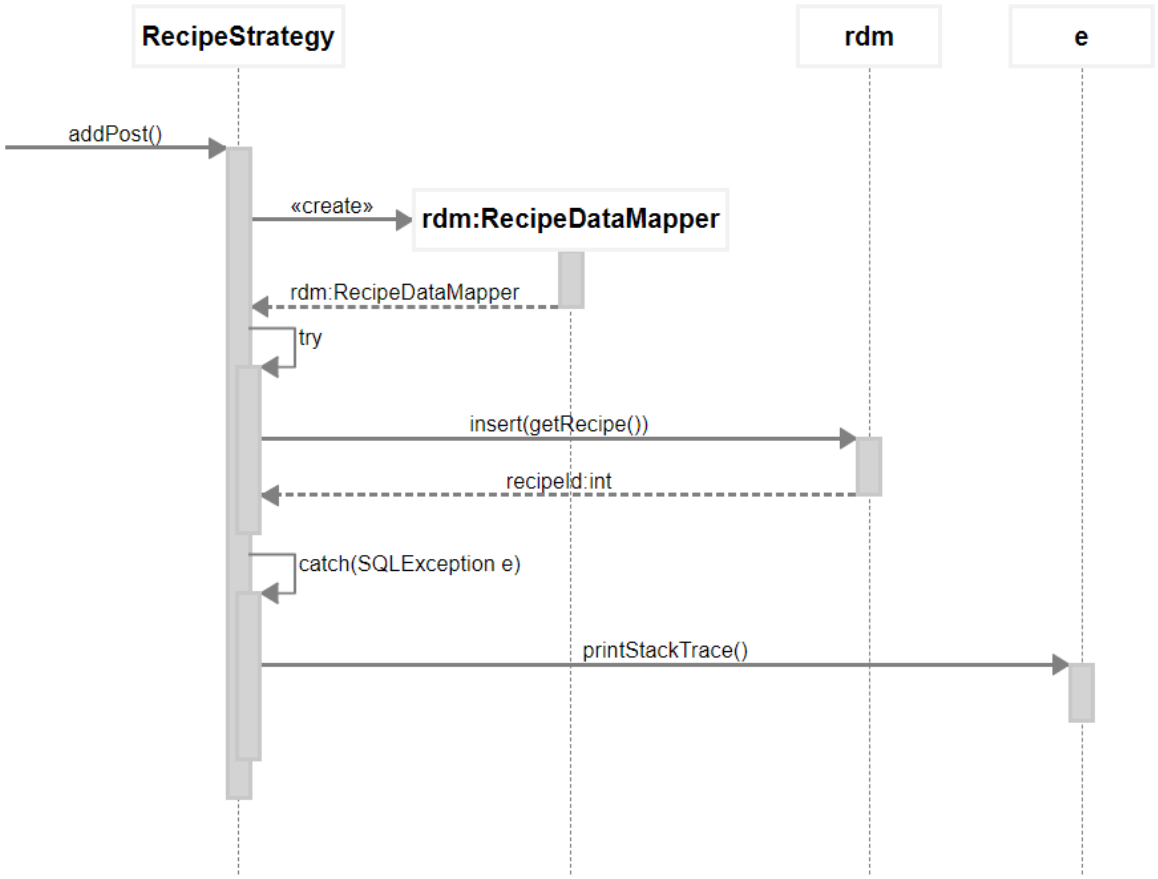

Figure 27 Recipe Strategy with RecipeDataMapper

Recipe Data Mapper Insert Recipe

This is RecipeDataMapper for inserting the recipe, here we create object DB for DBConnection and use it. The DB object interacts with methods preparedStatement and lastInsertedRecipe.

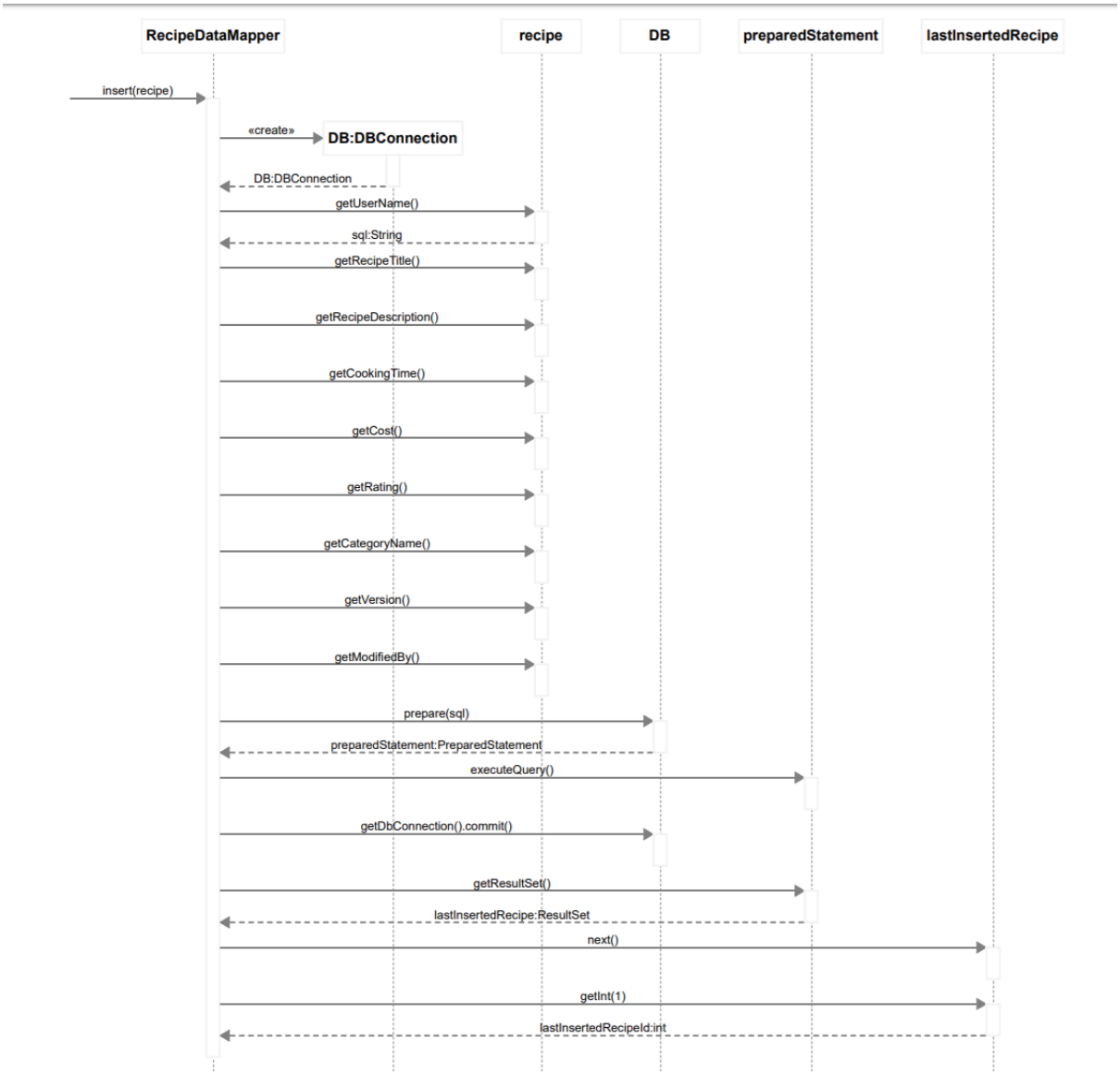

Figure 28 Insert Recipe

Recipe Data Mapper Read Recipe

Our RecipeDataMapper interacts with recipe object and RecipeRegistry to get ID of recipe and recipe itself respectively. Then it starts loading ingredients and comments related to that specific recipe. Here we can see use of Lazy Load pattern in and we show it in diagram by ingredientList and commentList

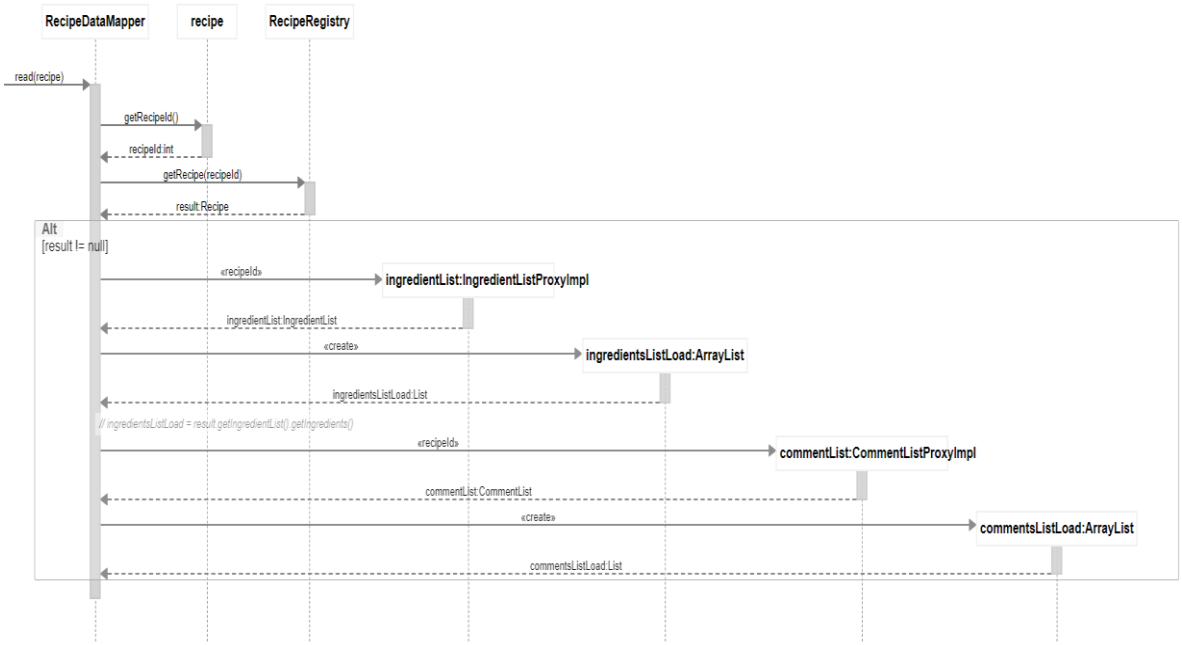

Figure 29 Read Recipe

Recipe Data Mapper Update Recipe

This diagram shows RecipeDataMapper for updating the recipes.

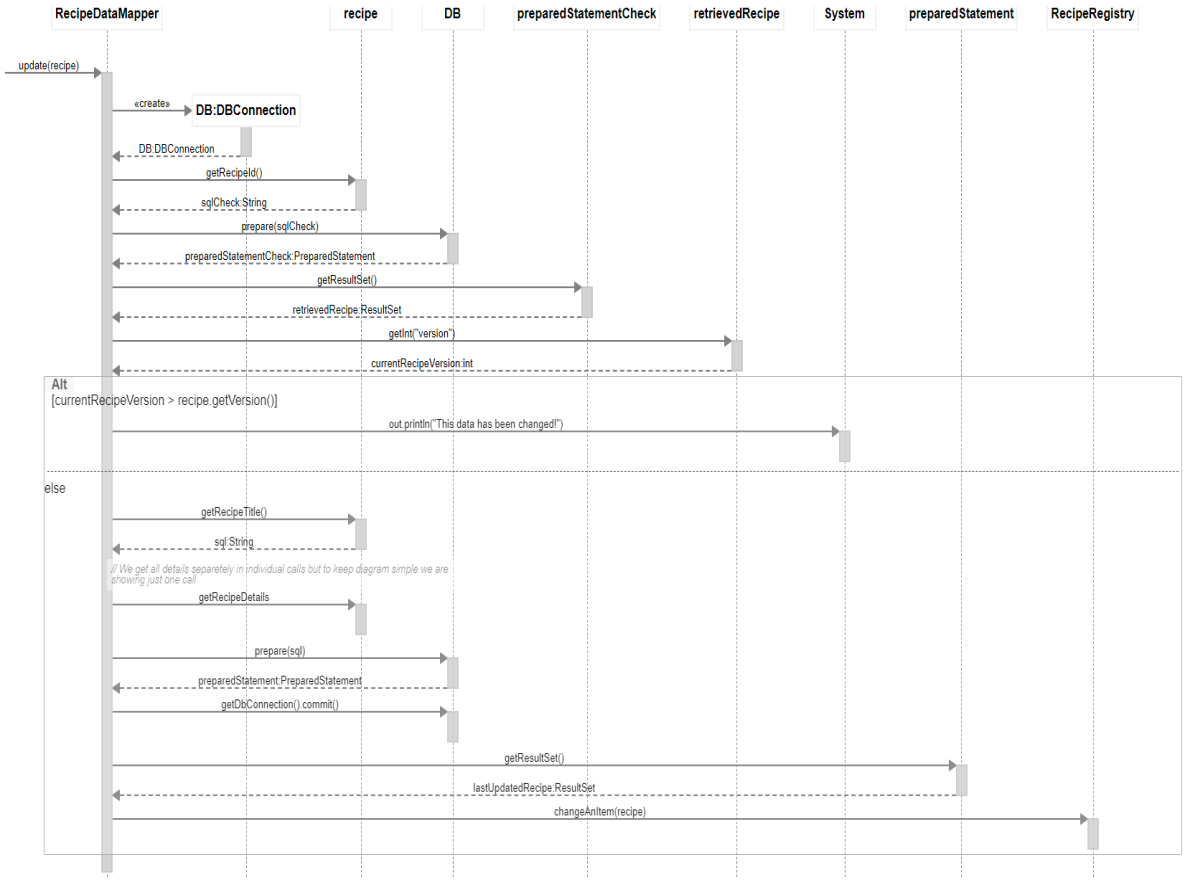

Figure 30 Update Recipe

Recipe Data mapper Delete Recipe

RecipeDataMapper to delete the Recipes, it interacts with recipe object and RecipeRegistry to get ID of recipe and creates a DBConnection object called DB. Then it communicates with preparedStatement method to execute the delete command and then DB commits the change.

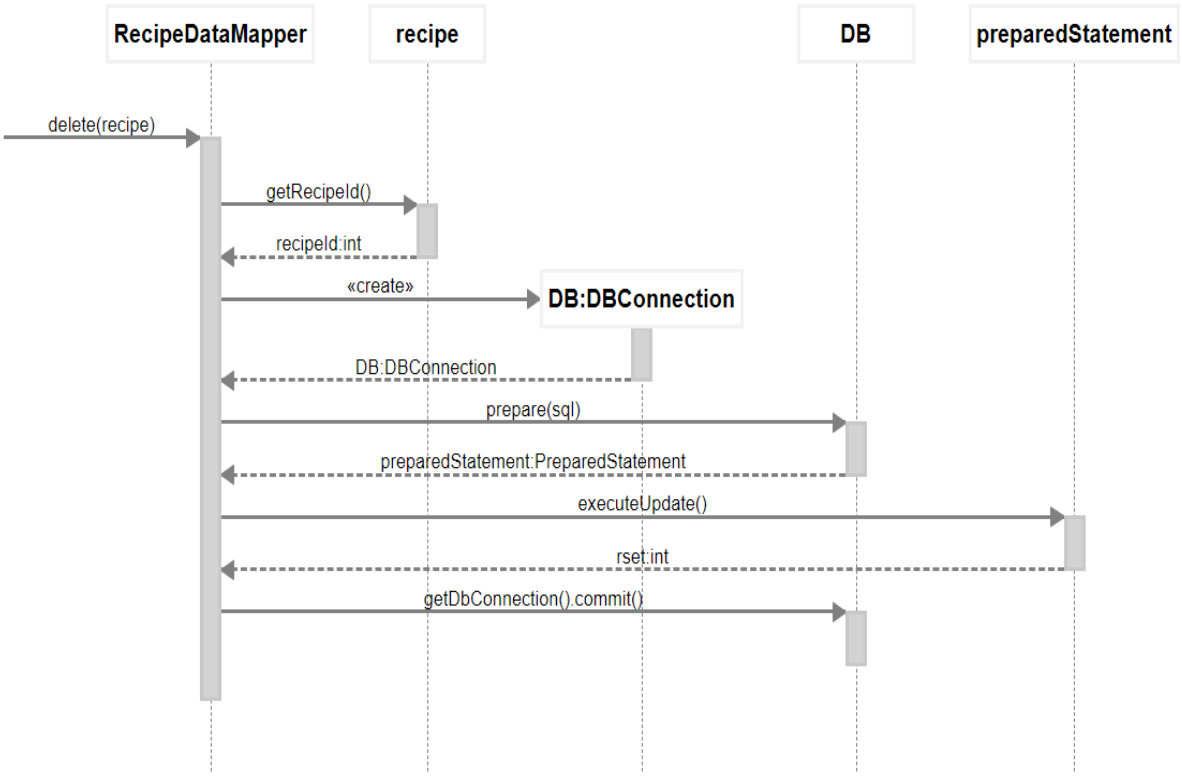

Figure 31 Delete Recipe