Software Design and Architecture

Online Movie Ticket Booking System

Haoyang Cui 886794

Linyuan Zhao 947588

* Item marked LIKE THIS are the main new/update part for Submission3

Part 1

System Overview

The online movie ticket booking system is a web-based enterprise system that allows customers to book movie tickets and cinema manager to do order management. The system stores lots of information, including customer order history, movie detail information and movie schedule information. The system allows customers to buy movie tickets, view movie information and order history, and it also allows a cinema manager to do movie schedule management and customer order management. The system has numbers of typical enterprise application characteristics, such as concurrent access, multiple user interface screen and different use permissions, which will make it suitable to support multiple users accessing and be able to handle conflicts like the required number of tickets not available or overlapped movie schedule.

Features & Use Cases

Feature A: Movie Management
The online movie ticket booking system allows movie management, and different users have different permissions on the system. A cinema manager could add new movies, delete movies, update movie information and schedule and view movie information, while a customer could only view the movie information in this system. The detailed use cases are shown below.

Scenario 1: Add Movie
Description: This service allows cinema managers to add new movies into the system.
Actor: Cinema manager
Precondition: The cinema manager has already been logged-in and on the home page.
Main successfully scenarios:

1. Click on the “Movie Management” button, and then there will be a list of all existed movies.
2. Click on the “Add New Movie” button
3. Input valid detail information about the movie.
4. Click on “Submit” button
5. The system shows “success” information and the movie is added successfully.

Alternative scenarios
3a. Some input is invalid, e.g. the selected theatre is used by another movie and is invalid in the scheduled time

1. Click on the “Submit” button
2. The system shows “Some settings are invalid ” information and the movie is not added

3b. Some input field is empty

1. Click on the “Submit”button
2. The System shows “Some settings are needed” information and the movie is not added

Scenario 2: delete Movie

Description: This service allows cinema managers to delete existed movies from the system.

Actor: Cinema manager

Precondition: The cinema manager has already been logged-in and on the home page.

Main successfully scenarios:

1. Click on the “Movie Management” button, and then there will be a list of all existed movies.
2. Select a movie from the movie list
3. Click on “Delete” button
4. The system shows “success” information and the movie is deleted successfully.

Scenario 3: Update Movie Information

Description: This service allows cinema managers to update movies’ detailed information or schedule.

Actor: Cinema manager

Precondition: The cinema manager has already been logged-in and on the home page.

Main successfully scenarios:

1. Click on the “Movie Management” button, and then there will be a list of all existed movies.
2. Select a movie from the movie list
3. Click on the “Edit” button
4. Input valid detail information about the movie.
5. Click on “Submit” button
6. The system shows “success” information and the movie is added successfully.

Alternative scenarios

4a. Some input is invalid, e.g. the selected theatre is used by another movie and is invalid in the scheduled time

    1. Click on the “Submit” button
    2. The system shows “Some settings are invalid ” information and the movie is not added

4b. Some input field is empty

    1. Click on the “Submit”button
    2. The System shows “Some settings are needed” information and the movie is not added

Scenario 4: View Movie Information (Cinema Manager)

Description: This service allows cinema managers to view existed movies from the system.

Actor: Cinema manager

Precondition: The cinema manager has already been logged-in and on the home page.

Main successfully scenarios:

1. Click on the “Movie Management” button, and then there will be a list of all existed movies.
2. Select a movie from the movie list
3. Then the system will show all the detailed information and schedule of the selected movie.

Scenario 5: View Movie Information (Customer)

Description: This service allows customers to view existed movies from the system.

Actor: Customer

Precondition: The customer has already logged in and in the home page
Main successfully scenarios:

1. Click on the “Movies” button
2. Select a movie from the movie list
3. The system shows detailed information and schedule of the selected movie.

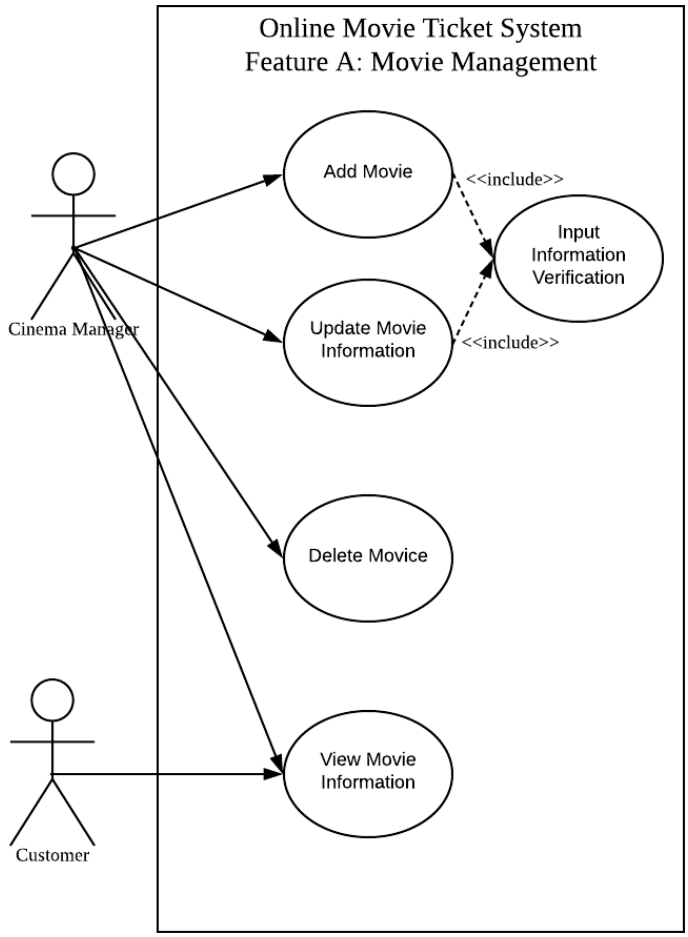


Figure 1 — Use case diagram of Feature A

Feature B: Order Management

The system allows order management, and different users have different permissions. A customer could create new orders and view all the existed orders (order history). A cinema manager could view all existed customer order information, cancel orders (with some limitation) and re-schedule order (with some limitation). The detailed use cases are shown below.

Scenario 6: Buy Movie Ticket

Description: This service allows customers to buy tickets of some specific movie session.

Actor: Customer

Precondition: The customer has already logged in and in the home page

Main successfully scenarios:

1. Click on the “Movies” button
2. Select a movie from the movie list
3. Click on “Buy Tickets” button
4. Select one time-schedule item from all the movie time list
5. Input some specific number of tickets
6. Click “Pay” to book the tickets
7. The system shows “Success” information and the number of tickets for that movie session are booked successfully.

Alternative scenarios

5a. Leave the field of number of tickets empty or input invalid information

1. Click the “Pay” button
2. The system shows “Please input a valid number” information and no ticket is booked

5b. The input number of tickets is larger than the number of available tickets

1. Click the “Pay” button
2. The system shows “No enough tickets left” information and no ticket is booked

6a. Click the “Cancel” button

1. The booking is cancelled.
2. The system jumps back to the home page

6b. The payment is unsuccessfully

1. The system shows “Payment failure” information and no ticketed is booked
2. The system jumps back to the home page

Scenario 7:View Order History (Customer)

Description: This service allows customer to view all his order history.

Actor: Customer

Precondition: The customer has already logged in and in the home page

Main successfully scenarios:

1. Click on “Order” button
2. The system shows all the order history and information in the form of list.

Scenario 8: View order History (Cinema Manager)

Description: This service allows cinema managers to view all customers’ order information.

Actor: Cinema manager

Precondition: The cinema manager has already been logged-in and on the home page.

Main successfully scenarios:

1. Click on the “Customer Management” button, and then there will be a new page with all existed customer accounts

2. Click on one specific account from the account list
3. Click on the “Order Management” button
4. The system shows an order list which is the order history of the selected customer

Scenario 9: Cancel order

Description: This service allows cinema managers to cancel customer orders.

Actor: Cinema manager

Precondition: The cinema manager has already been logged-in and on the home page.

Main successfully scenarios:

1. Click on the “Customer Management” button, and then there will be a new page with all existed customer accounts
2. Click on one specific account
3. Click on the “Order Management” button
4. The system shows an order list which is the order history of the selected customer
5. Click on “Cancel” button in a specific order item row, where the specific order is still valid to be cancelled.
6. The system shows “Success” information. The order is cancelled and the customer will be refunded later

Alternative Scenarios:

5a. Click on “Cancel” button where the selected order is invalid to cancel (too late or some other reasons)

1. The system will show “Cannot cancel it now” information. The order will not be cancelled and the customer won’t be refunded.

Scenario 10: Re-schedule order

Description: This service allows cinema managers to re-schedule customer orders.

Actor: Cinema manager

Precondition: The cinema manager has already been logged-in and on the home page.

Main successfully scenarios:

1. Click on the “Customer Management” button, and then there will be a new page with all existed customer accounts
2. Click on one specific account
3. Click on the “Order Management” button
4. The system shows an order list which is the order history of the selected customer
5. Click on “Re-schedule” button in a specific order item row, where the specific order is still valid to re-schedule.
6. Input valid re-schedule information, including movie, time and number of tickets.
7. Click on the “Submit” button
8. The system shows “Success” information. The order is re-scheduled and the customer will be refunded if necessary.

Alternative Scenarios:

5a. Click on “Re-schedule” button where the selected order is invalid to re-schedule (too late or some other reasons)

1. The system will show “Cannot re-schedule it now” information. The order will not be changed.

6a. Input some invalid information or leave some field empty

1. Click on the “Submit” button
2. The system shows “Please input valid information”, and the order will not be changed

6b. The inputed schedule is not available, which may because the number of tickets is not available or some other reasons.

1. Click on the “Submit” button
2. The system shows “The schedule is invalid”, and the order will not be changed

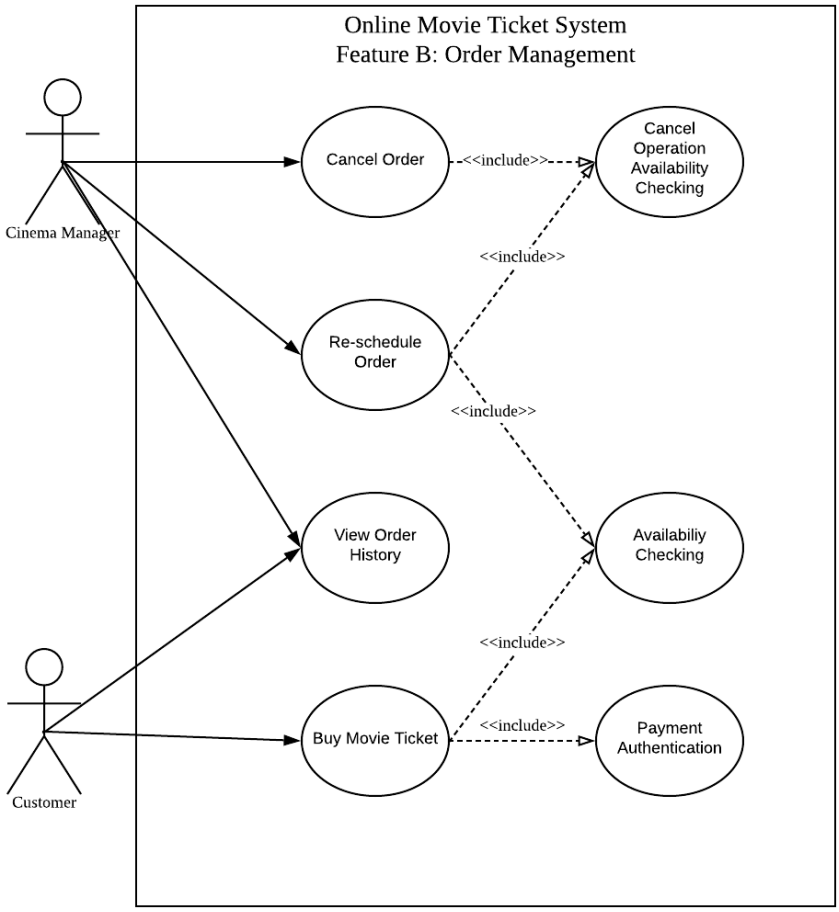


Figure 2 — Use Case diagram of Feature B

Some other necessary features & use cases

Scenario 11: Log in (Cinema Manager)

Description: This service allows users to log in as cinema managers with related permissions

Actor: Cinema Manager

Precondition: In the home page without any account already logged-in

Main Successful Scenarios:

1. The cinema manager click on the “Login” button
2. Input valid username and password.
3. Click on the “Login ” button
4. The system shows success information and the cinema manager logs in successfully.

Alternative Scenarios

2a. The cinema manager input invalid username or password

1. Click on the “Login” button
2. The system shows “invalid username or password” information and the cinema manager won’t be logged in.

2b. The cinema manager leaves username field or password field empty

1. Click on the “Login” button
2. The system shows “username or password needed” information and the cinema manager won’t be logged in.

Scenario 12: Logout (Cinema Manager)

Description: This service allows cinema managers to log out.

Actor: Cinema manager

Precondition: The cinema manager has already been logged-in and on the home page.

Main successfully scenarios:

1. Click on the “Logout” button
2. The system will show “success” information and the cinema manager logs out successfully.

Scenario 13: Register (Customer)

Description: This service allows users to create new customer account

Actor: Customer

Precondition: In the home page without any account already logged-in

Main successfully scenarios:

1. Click on the “Register” button
2. Input valid username and password
3. Click on the “Submit” button
4. The new account is created successfully

Alternative Scenarios:

2a. The username has already been used

1. Click on the “Submit” button
2. The system shows “Invalid username, input another please” information, and the new account is not created.

2b. The username or password field is empty

1. Click on the “Submit” button
2. The system shows “Some field still empty” information, and the new account is not created.

Scenario 14: Login (Customer)

Description: This service allows users to log in as customers with related permissions

Actor: Customer

Precondition: In the home page without any account already logged-in

Main successfully scenarios:

1. Click on the “Login” button
2. Input valid username and password
3. Click on the “Login” button
4. The user logs in as a customer successfully

Alternative Scenarios:

2a. Input invalid username or password or leaves username field or password field empty

1. Click on the “Login” button
2. The system shows “Please input valid username and password” information and the customer is not logged-in

Scenario 15: Logout (Customer)

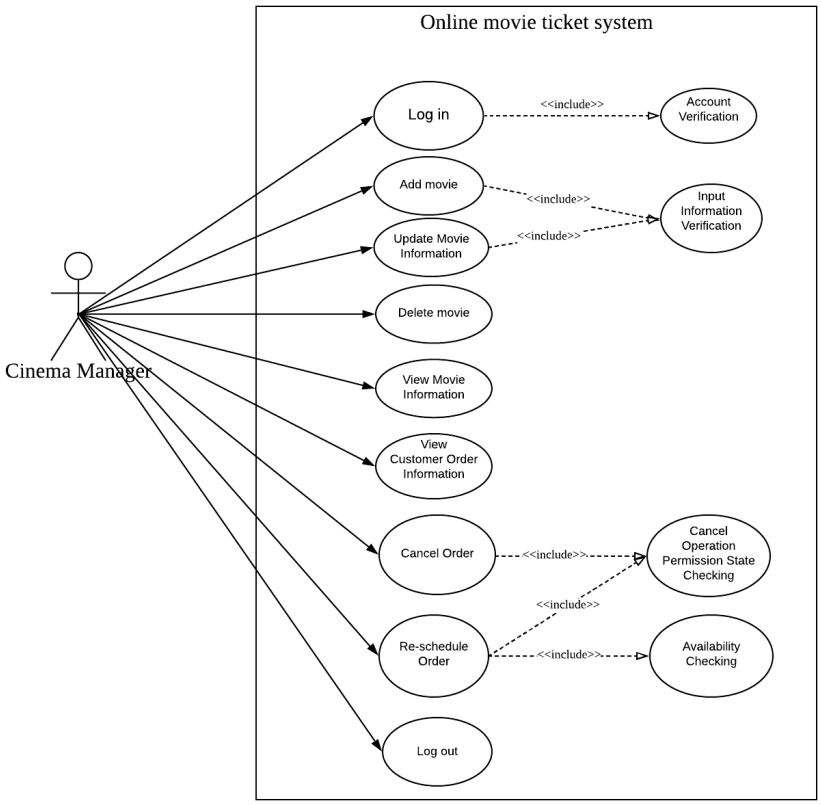
Description: This service allows customers to log out.

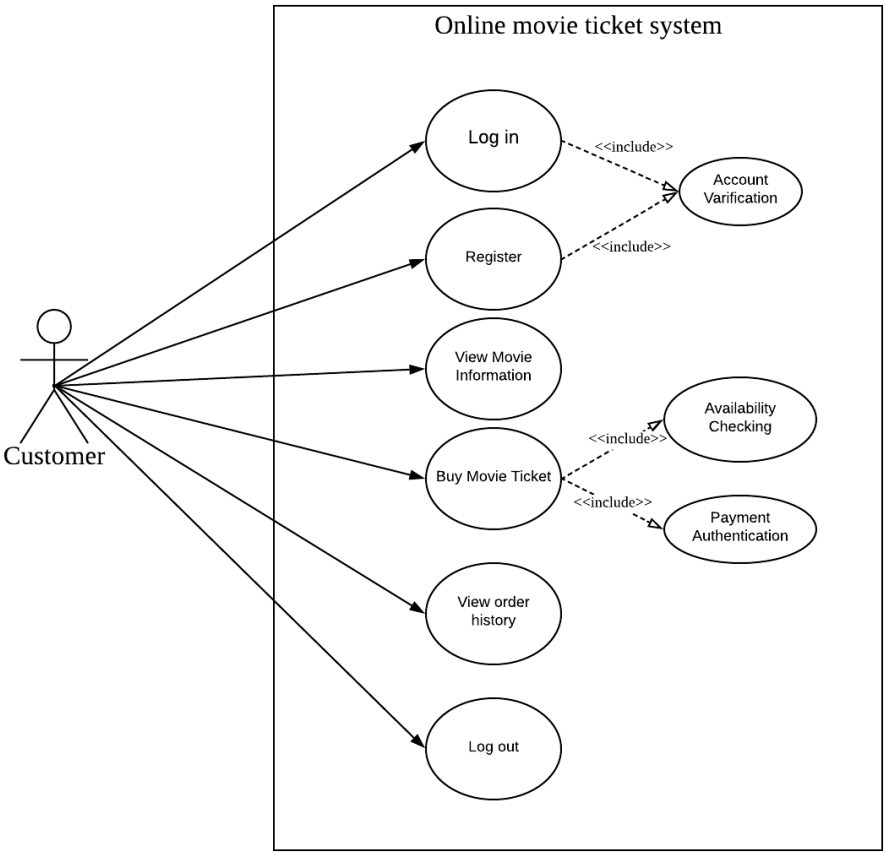
Actor: Customer

Precondition: The customer has already logged in and in the home page

Main successfully scenarios:

1. Click on the “Logout” button
2. The system will show “success” information and the customer logs out successfull


Online movie ticket system
Cinema Manager
Log in
<<include>>
Account Verification
Add movie
<<include>>
Input Information Verification
Update Movie Information
<<include>>
Delete movie
View Movie Information
View Customer Order Information
Cancel Order
<<include>>
Cancel Operation Permission State Checking
<<include>>
Re-schedule Order
<<include>>
Availability Checking
Log out


Online movie ticket system
Customer
Log in
<<include>>
Account Varification
Register
<<include>>
View Movie Information
Buy Movie Ticket
<<include>>
Availability Checking
<<include>>
Payment Authentication
View order history
Log out

Git repository: https://github.com/HaoyangCui0830/SWEN90007-Project-OnlineMovieTicketBookingSystem.git

Deployment URL: https://online-movie-ticket-booking.herokuapp.com/

Part 2 & 3

Feature A: Movie Management

Feature B: Order Management

1. Introduction

The online movie ticket booking system is a web-based enterprise system that allows customers to book movie tickets and cinema managers to do ticket management. In this submission, we completed the second feature in our proposal, which is ticket management system. Same as last submission, there are two roles in this feature, which are customer and cinema manager.

- **Customer**

Customer could view all movie information in the system. Customers could firstly get a list of all movies and they could view detailed information of one specific movie by clicking on the view button of that movie's row.

Customer could also buy tickets of one specific selected movie session by clicking the "buy ticket" button. In our system, users will need to login as customers firstly to purchase tickets. One unregistered user will need to register and get one account firstly. After buying some tickets successfully, a customer could also view all his order history.

- **Cinema Manager**

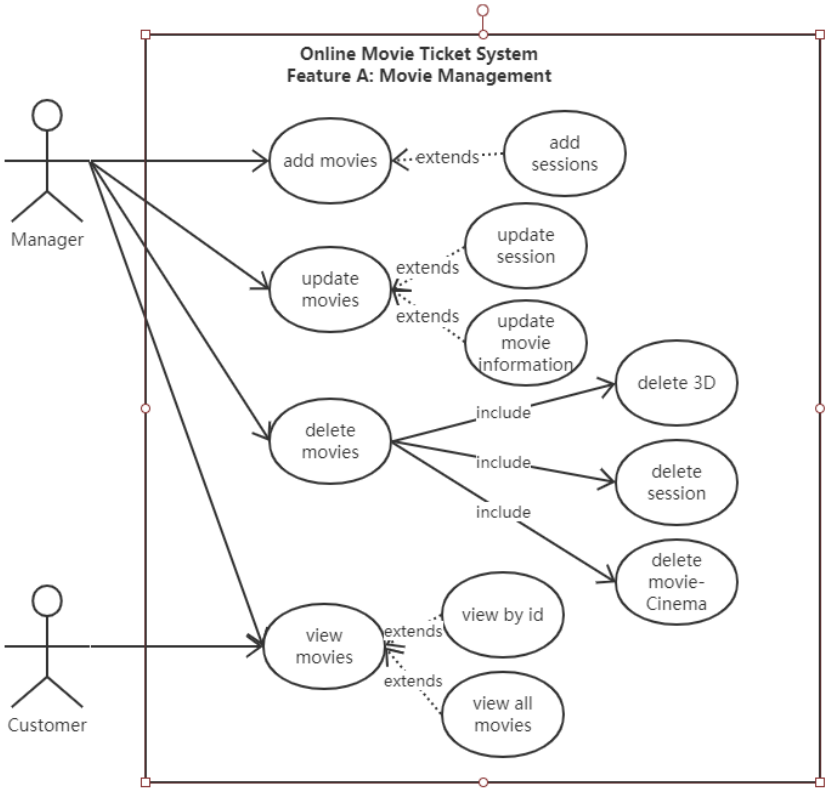
The cinema manager has the right to add, delete, edit and view movie information. Here, the movie information includes both the movie details and other movie related information, including cinemas, sessions and whether it's a 3D movie or not. The word 'session' represents the movie screening information, including when that session will start, when it will end and how many seats will be available.

Also, a cinema manager could edit, delete and view all users' ticket information. To achieve that, one should login as cinema manager firstly. Our system didn't provide register function for cinema manager. All manager account are set previously in our DB.

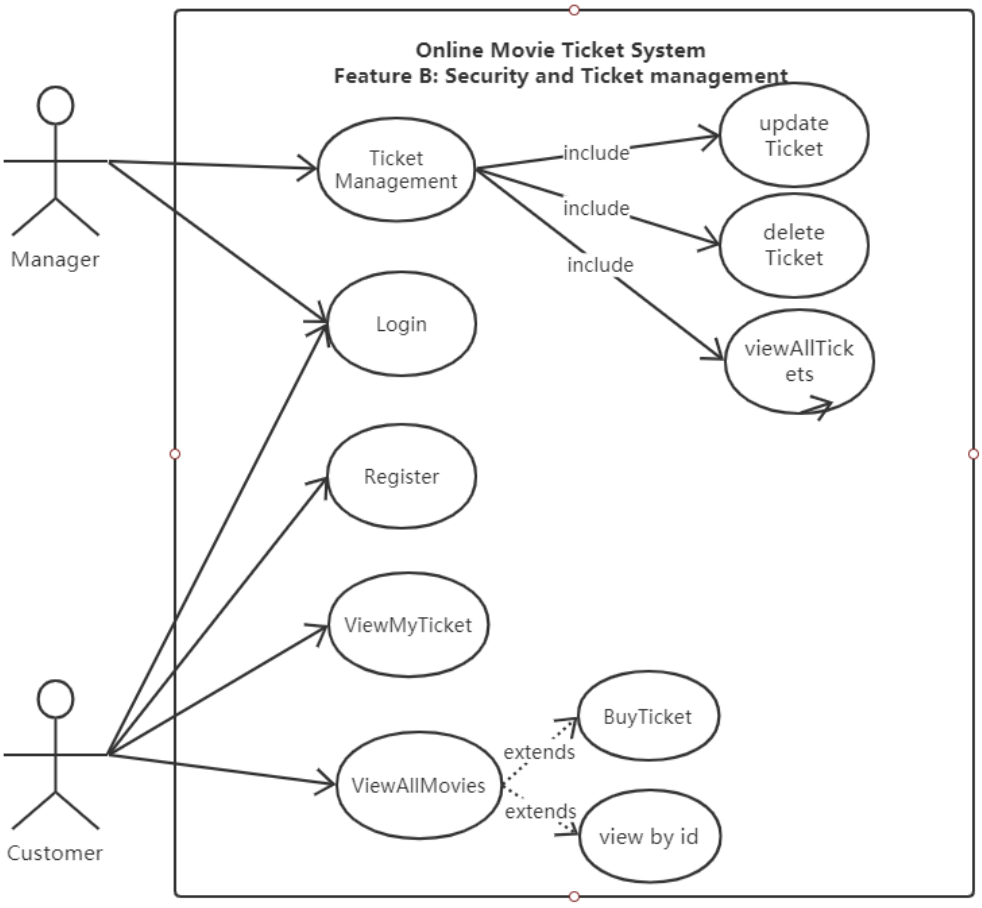
2. Use Case

Here is the use case diagrams of both features in our system

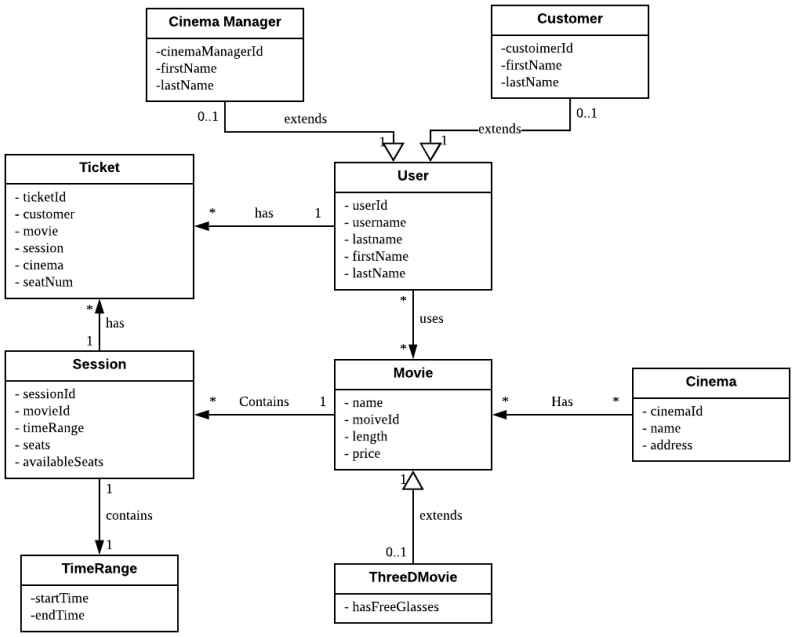
Feature A: Movie Management



Feature B: Ticket Management



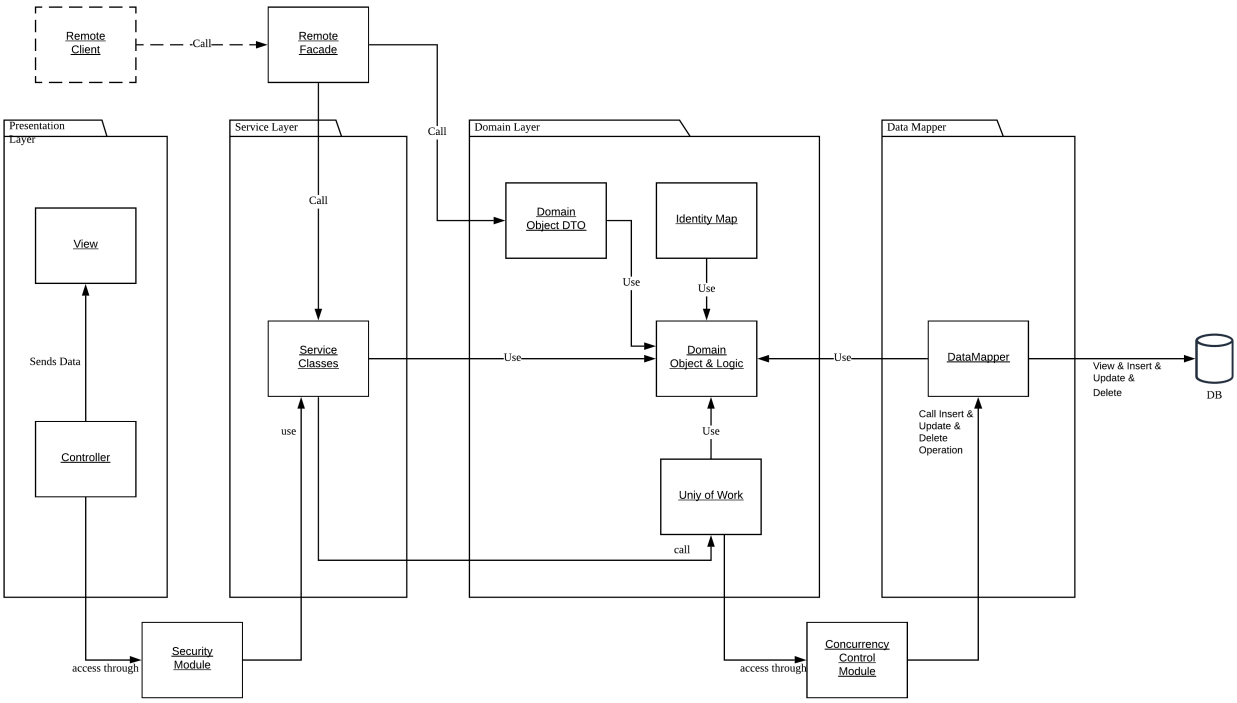
3. Class Diagram &. Explanation



- **User:** This class contains username, password and all other information of both Cinema Manager and Customer. It's used in many patterns and functions, including authentication, authorization and so on.
- **Ticket:** This class contains all information needed in ticket, including customer, specific movie and movie session, and also has the seatNum attribute which represents how many seats is ordered in one ticket.
- **Cinema:** This class contains the information of a cinema.
- **Movie**: This class contains all information of a movie itself.
- **ThreeDMovie:** This class contains extra information of a 3D movie. It extends the Movie class. Some Movie object may have related ThreeDMovie object while some others may not. The inheritance relationship is class table inheritance, which means each class has one table and they share one same primary key (movieId).
- **Customer:** This class contains all information of a customer.
- **Cinema Manager:** This class contains all information of a cinema manager.
- **Session:** This class contains all information of a session, and one session could only related with one movie object by the including movieId in attributes.
- **TimeRange:** This class contains all information of one time schedule, including the start time and end time. It is used by session object and one session object could only has one TimeRange object as one of its attributes.

4. High Level Architecture

The high level architecture diagram of the system is as below:

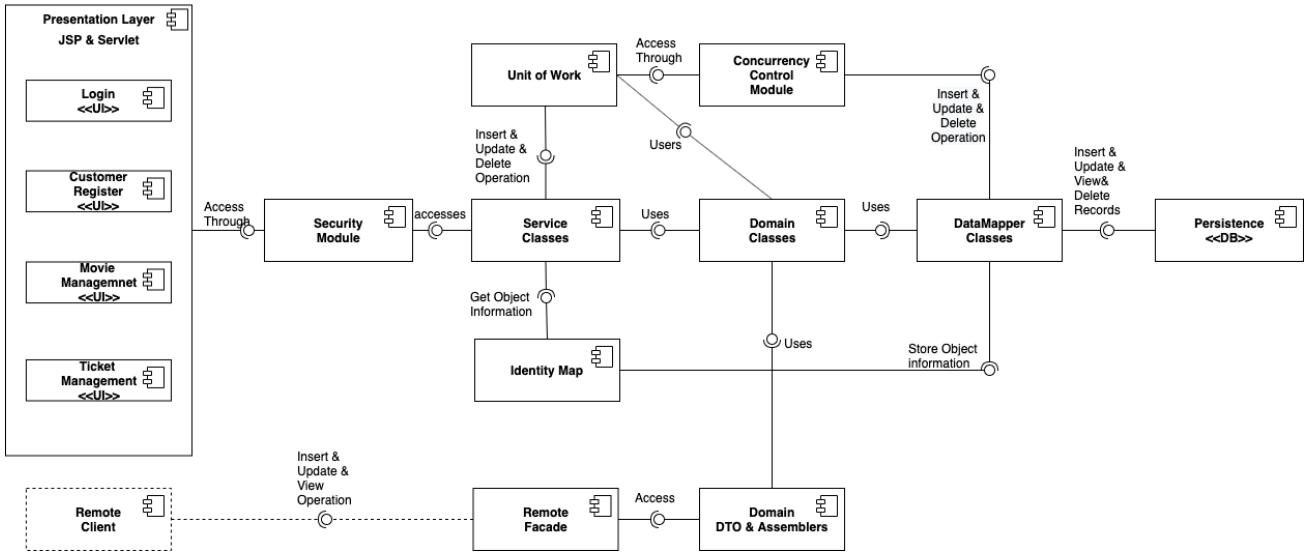


Roles and Responsibilities of All Layers

Layer	Roles and Responsibilities
Presentation Layer	• Display Views and Information generated by controllers • Collect user input information • Transmit all business calls with related data (in http session) to the service layer
Service Layer	• Contains application-specific logic and handle with all application requests • Involve multiple resources and actions and coordinate the application's response in each operation
Domain Logic Layer	• Contains the domain-specific logic which is independent of the system • Data access management • Provide DTO and Assembler classes for remote calls (work together with remote facade)
Data Mapper	• Provides the mapping between DataBase and Domain/Service Layer • Make the rest of application layers independent with the structure of DataBase • Also implement concurrency control component (implicit mapper) in this layer

(More detailed explanation about components in High Level diagram is listed after component diagram)

The **Component Diagram** of the system is show as below



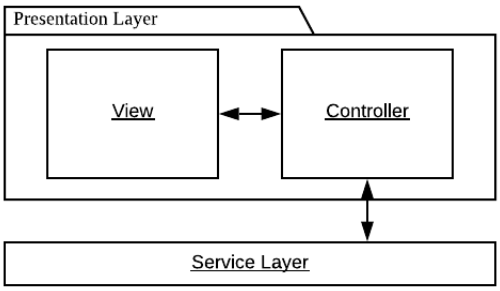
Roles and Responsibilities of some important Components:

Components	Roles and Responsibilities
Unit of Work	• Maintain a list of Domain Object, commit all to DataMapper to reduce the times connecting with DB • Used in concurrency control, call implicit mapper to make all services acquire locks automatically
Identity Map	• Ensure all object only be loaded once • Keep all loaded objects in local map
Security Module	• Includes Authentication enforcer, Authorization enforcer, Validator and Security Pipeline • All request from clients will pass through security module and be verified to be safe before reaching Service Layer
DTO & Assemblers	• DTO classes are the coarse-grained object based on relevant domain object, it helps to reduce the number of method calls • Assemblers will translates DTO to domain object and from domain object to DTO. It will be also in charge of providing View, Update and Add interface for remote facade.
Remote Facade	• Provide coarse-grained facade for remote clients to reduce the number of calls
Remote Client	• The distributed system which will call Remote Facade
Concurrency Control Module	• Exclusive write lock and Implicit lock are implemented in this module • Will solve the concurrent writing problem by prevent another callers to get some lock which is currently hold by one process.

5. Presentation Layer

5.1 Entire Presentation Layer Design

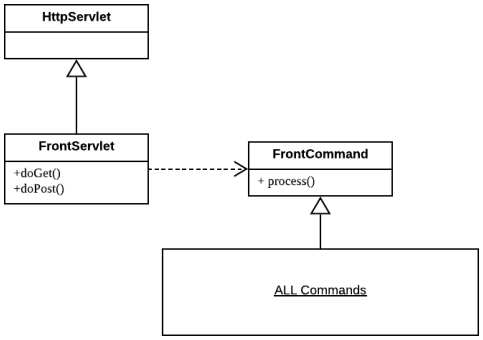
MVC pattern is applied in our presentation layer. For controller, we used **front controller**. And for view, **template view** is used in our model. The following diagram shows the general presentation layer pattern.



- The Service layer will provide data and handle request from clients.
- The controller will handle the interaction and data flow between view and domain.
- View will deploy information and collect input from user.
- All requests will be firstly verified by security module for authentication, authorization, validation. The data transmission uses SSL certificates.

5.2 Pattern for Controller

Front Controller pattern is implemented in our system. The abstract structure of the controller pattern is as follows.



The *FrontServlet* is the only servlet class in the controller. All url request will come through *FrontServlet* and then be parsed into related *command* class. All *command* class extends the *FrontCommand* abstract class and will implements all its methods. And each *command* class will handle their own issues.

The reason why we use *Front Controller* pattern are:

- With only one entry servlet, we could make the implementation easier and make the presentation layer structure more clear.
- By implements all the commands separately, we could make the system be much easier to extend, which means better scalability and flexibility.

Comparison with Page Controller pattern: compared with *Page Controller, Front Controller* could provide much better extensibility. And in *Page Controller,* one single controller has to handle all logic related with the webpage, which make it complex to implement.

5.3 Pattern for View

Template View pattern is implemented in our system for View. This pattern will use static JSP files as webpage template, together with data collected from controllers or users to build the total webpage view.

Comparison with other View Patterns: The reason why we select this pattern is simply because it's easy to design and implement. As our system is small-sized, the disadvantages of this pattern like low maintainability and hard to test are not big problems. Applying this pattern could take much less time than *Transform View* and *Two Step View.*

5.4 Actual Presentation layer Structure Diagram

The actually presentation layer structure is like below
As the diagram's detail is hard to be seen clearly on this documentation, we provide the *dropbox* link so that the structure could be viewed clearly.

Dropbox Link:
https://www.dropbox.com/s/dvaeicnqly5olxh/Controller.png?dl=0

5.5 Detail Responsibilities of all classes in Controller

Command	Responsibility
CustomerHomeCommand	Return to the Customer Home Page
CustomerViewAllMoviesCommand	Collect information to build View Movie List page for Customers
CustomerViewMovieCommand	Used after customer select one movie on the movie list and jump to the Movie detailed Page. Will collect all needed information of that specific movie
FrontCommand	The command to be extend by all the other commands, has one abstract method process().
FrontServlet	The Only handle for all requests from user interface. Also parse the input command name and then create and call corresponding commands

Command	Responsibility
ManagerAddMovieCommand	Used to jump to the add movie information page for manager
ManagerAddSessionCommand	Used to jump to the add session information page for manager
ManagerDeleteMovieCommand	Collect the delete movie command from user and call movie service to delete one specific movie
ManagerDeleteSessionCommand	Collect the delete session command from user and call session service to delete one specific session
ManagerEditMovieCommand	Jump to the edit movie information page for manager
ManagerEditSessionCommand	Jump to the edit session information page for manager
ManagerHomeCommand	Jump to the home page for manager
MangerSubmitNewMovieCommand	Used to collect the input parameters for a new movie and call movie service to add new movie
ManagerSubmitNewSessionCommand	Used to collect the input parameters for a new session and call the session service to add the new session
ManagerUpdateExistedMovieCommand	Used to collect the input parameters for an existed movie and call the movie service to submit the edit information for the movie
ManagerUpdateExistedSessionCommand	Used to collect the input parameter for an existed session and call the session service to submit the edit information for the session
ManagerViewAllMovie	Collect all information and build a movie list page for manager
ManagerViewMovieCommand	Used after manager select one movie on the movie list and jump to the Movie detailed Page. Will collect all needed information of that specific movie
ToHomePageCommand	Jump to the root Home Page
CustomerBuyTicketCommand	Jump to the customer purchase page for confirming the order information and buying the tickets
CustomerConfirmCommand	Customer check and confirm all the ticket information and the purchase it or cancel order
CustomerPaymentCommand	Be Called after customer click on the Confirm button in order confirm page. Will check whether the order is successful or not and will jump to error Page if the purchase failed.
CustomerViewTicketCommand	Customer could view all the order history
LoginCommand	Will make the brewer jump to the login JSP page
LogoutCommand	The logout command for both customer and cinema Manager
ManagerDeleteTicketCommand	Page for cinema manager to delete some specific order

Command	Responsibility
ManagerEditTicketCommand	Page for cinema manager to edit selected order information
ManagerUpdateTicketCommand	After cinema manager changing some ticket's information, the command will submit all the changes to the system.
ManagerViewAllTicketCommand	This Page will return all ticket information for cinema manager for viewing
ManagerViewTicketCommand	This page will give the information of one selected ticket to the cinema manager
RegisterCommand	This command will check and submit all this information inputed by a user to register as a customer.
ToRegisterCommand	This command will make the view jump to the register JSP webpage
UserLoginPage	The login Page for both customer and cinema Manager

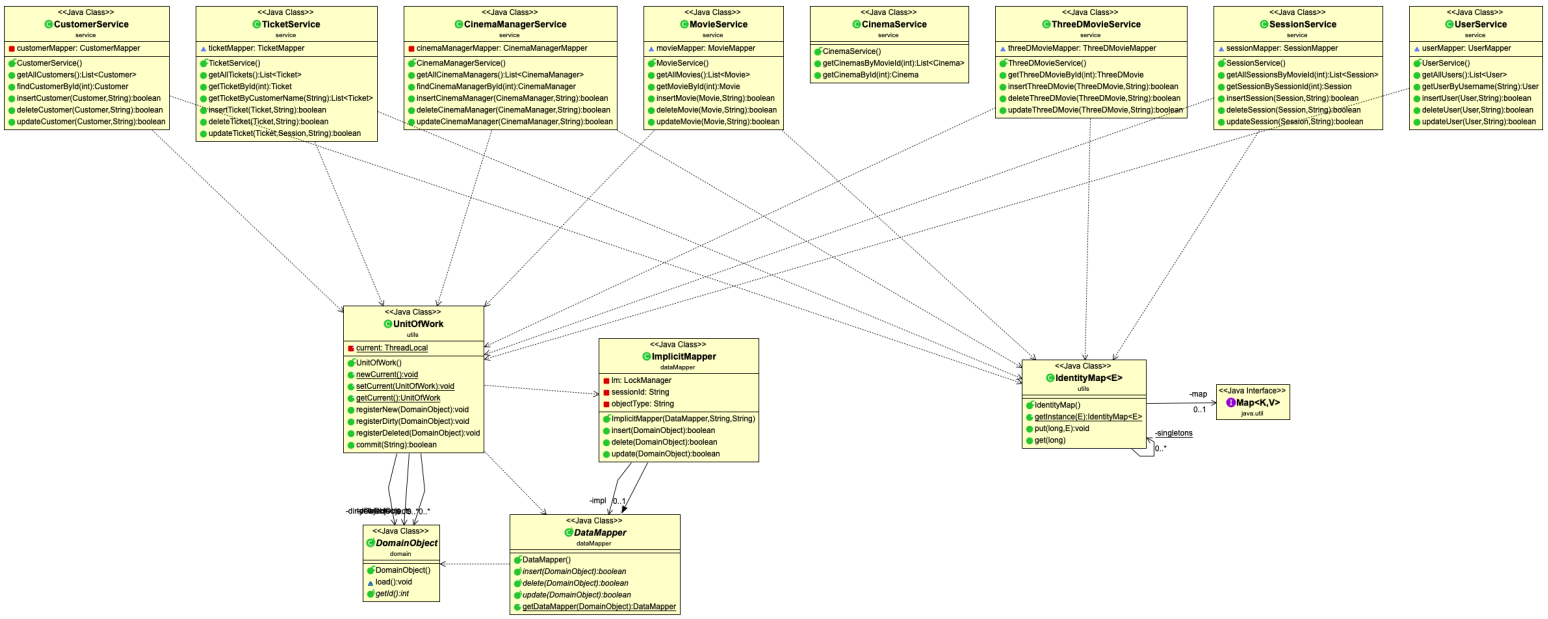
6. Service Layer

6.1 Service Layer General Design

The service layer is used to "Define an application's boundary" and "coordinates the application's response in each operation". **Operation Script** variation of Service Layer's Division methods is used to divide service layer with domain layer. According to this pattern's definition, domain layer will contain the domain-specific logic which is the domain itself and is independent of the system, like the type of Movie (3D or not). While the service layer contains the application-specific logic which will packages multiple domain classes, actions and resources and could be accessed by the presentation layer directly.

The reason why **Operation Script** pattern is applied is that it promotes re-use better than **Domain Facade** pattern. As the design and logic of the domain-specific logic could be reused by other systems and different actions could be implemented by simply replacing the application-specific logic into some others.

6.2 The Service Layer Class Diagram

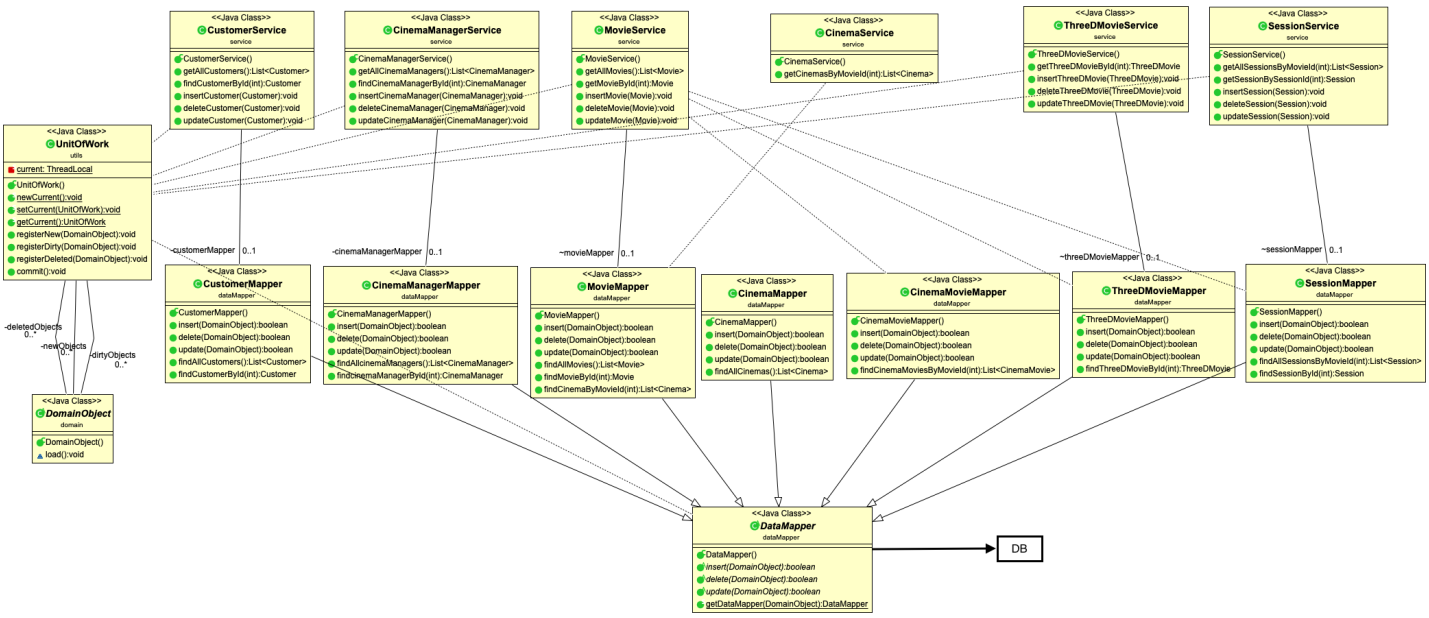


6.3 Responsibilities of all Services

Service	Responsibility
MovieService	CRUD service for movie. The movie objects could be viewed as a movie list or one specific movie object (after select one from the movie list)
SessionService	CRUD service for session. Could be view in two ways the same as MovieService.
ThreeDMovieService	CRUD service for ThreeDMovie. Could be view in two ways the same as MovieService.
CustomerService	CRUD service for Customer. Could be view in two ways the same as MovieService.
CinemaManagerService	CRUD service for CinameManager. Could be view in two ways the same as MovieService.
CinemaService	Viewing Cinema based on Movie Id. As we don't allow cinema manager to create, delete or edit cinema information.
UserService	CRUD service for User. Could get all user information or get one specific user based on the id.
TicketService	CRUD service for Ticket object, could get all tickets' information or check one specific ticket based on customer or ticket id.

6.4 Unit of Work

Unit of Work pattern is implemented and used by both Service Layer and Data Mapper Layer. The way it is used could be shown as below

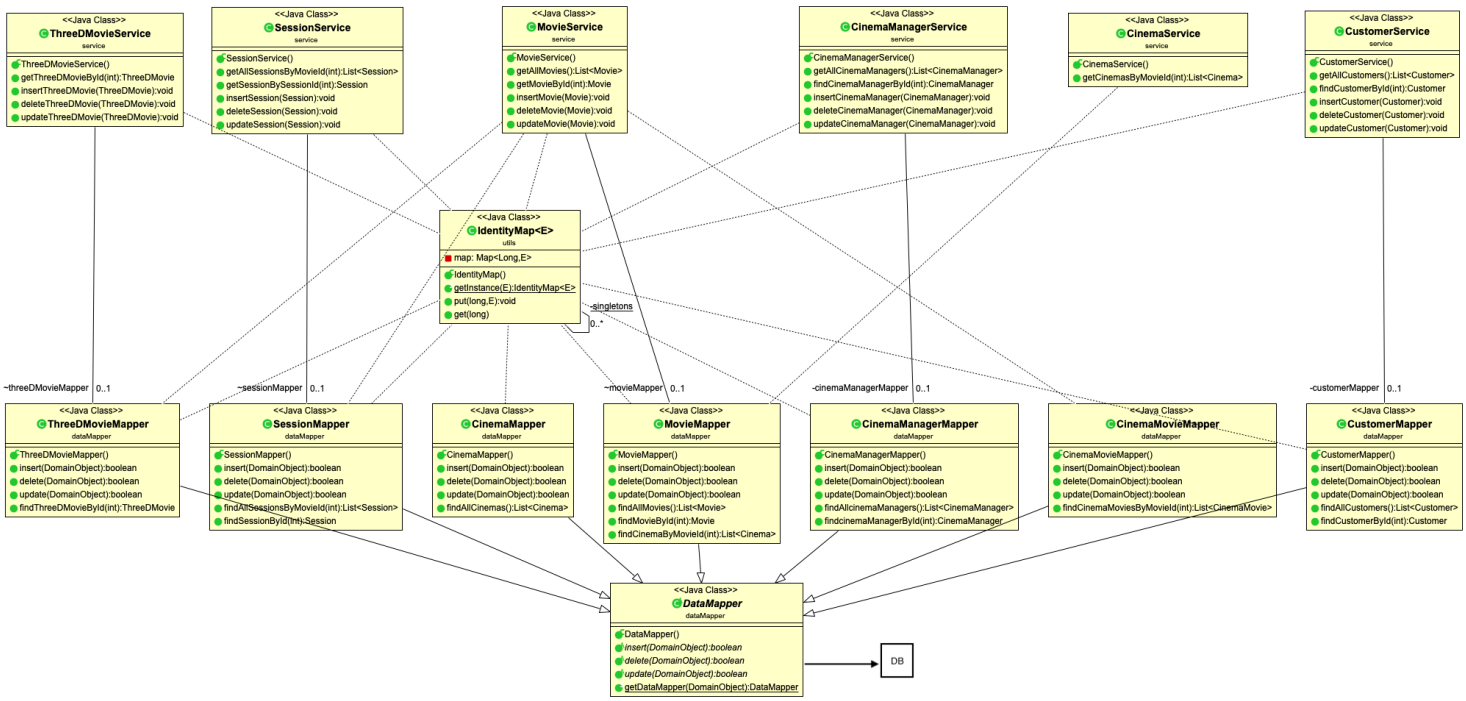


Explanations:
The unit of work pattern is used to maintain a list of objects that are affected by a business transaction. The reason why we applied it is that this pattern could make the DB connected operation more efficient, as it will tag all the objects changes in one business logic operation and commit all of them to the database together. For example, in the implementation of deleteMovie(Movie) function in MovieService Class, if the user want to delete one movie object, all related Session objects, Cinema_Movie objects and ThreeDMovie objects will be deleted as well. By using the Unit of Work pattern, the system will tag all related objects as “deleted” and commit all to the DB at the end of this method. It’s an efficient and simple way to implement the business logic of deleting movie.

The way in which UnitofWork achieves the commit operation is related with Data Mapper Layer and will be explained there. (The explanation of commit&DataMapper: UnitofWorkinDataMapperLayer)

6.5 Identity Map

Identity Map pattern is implemented and used by both Service Layer and Data Mapper Layer, the way it is used could be shown as below:



Explanations:
By keeping loaded objects in a map, identity map could make the loading of one object only happens once. The advantages are:

- By only one loading and maintaining only one same object, clash of different operations could be prevent as there is only one object across all the system. In this way, concurrent accessing’s safety could be guaranteed.
- Apart from that, it could also decrease the cost as duplicate objects loading will be prevented.

The way in which service classes use identity map is as follows:

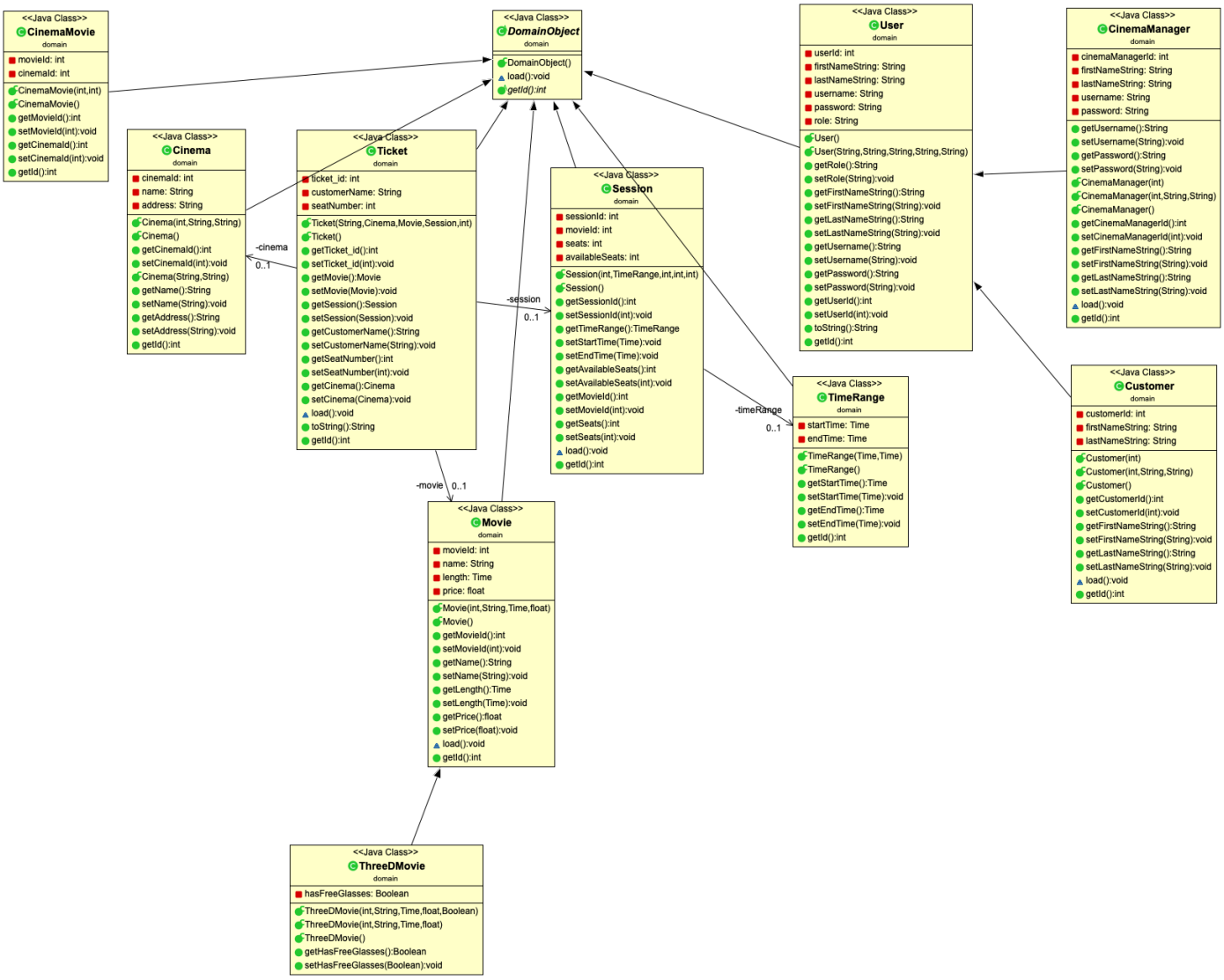
- Every time a view type of service is called, the service class will firstly check if the required object is already existed in identity.
- If so, it will load from the object from identity map and return it directly.
- If not, it will call related Data Mapper class and function, the function will load the object's information from database and assign the information to one object. That object will be put into identity map and then return. Thus, singleton is still maintained across all layers.

(As most identity map related description is listed here, there won't be any more duplicate explanation in the Data Mapper Layer section even it's also used by Data Mapper Layer).

7. Domain Layer

7.1 General Design

Multiple patterns are applied in the design of Domain Layer, including Lazy Load, Identity field, Foreign Key Mapping, Association Table Mapping, Embedded Value and Class Table Inheritance. The Domain Layer Class Diagram is as follows:



Explanations:

- All classes in Domain Layer extend DomainObject class, which could make the design more clear and clean, and it's also necessary for UnitofWork implementation.
- As Operation Script devision principle is used, the domain later only contains domain-specific logic which is independent with the system. All application-specific operations (like deleting, inserting and updating) will only be implemented in the Service Layer.

7.2 Responsibilities of All Classes:

Classes	Responsibility
DomainObject	The class that is extended by all the other classes in Domain Layer
Movie	Represent the movie information of Movie table in DB.
Session	Represent the session information of Session table in DB
Customer	Represent the customer information fo Customer table in DB
CinemaManager	Represent the CinemaManager information for Cinema Manager table in DB
CinemaMovie	Represent the CinemaMovie information for CinemaMovie table in DB
ThreeDMovie	Represent the ThreeDMovie information for ThreeDMovie table in DB
Cinema	Represent the Cinema Information for Cinema table in DB
TimeRange	Used to store the start time and end time information in Session table in DB
User	Use to represent user information, is needed by Shiro
Ticket	Represent the ticket information, includes other objects' information like session, movie and so on

7.3 Lazy Load:

Ghost pattern is used in the many domain layer classes, which is easy to implement and could decrease the times needed to access the database. The way in which it achieve the outcome is that every time instead of query only the required field at one time, the pattern will force the object to query all fields it has, and the next some other attributes of that object is required, there is no needs to query again as it has already been loaded. The implementation of Lazy Load increases the efficiency of accessing data and all related operations.

7.4 Object-to-relational structural design

Multiple patterns of object-to-relational structural design are implemented in the system, including Identity field, Foreign Key Mapping, Association Table Mapping, Embedded Value and Class Table Inheritance. Here are some details.

7.4.1 Identity Field

This pattern is used between many classes and database tables, for example, the movidId field in Movie Class in Domain Layer represent the primary key of Movie table in database. The advantages of this pattern is that it is simple to implement and every time we could easily identify the corresponding row of domain objects using the identity field.

7.4.2 Foreign Key Mapping

This pattern is used to Maps an association between objects to a foreign key reference between tables. It is implemented in our system as well. For example, the movieId field in Session class represents the foreign key relationship between Session table and Movie Table in database. The reason why we implement this pattern is that it's efficient for representing one-to-many relationship in database, and the mapping between the domain relationships and database tables could be done simply and mechanically.

7.4.3 Association Table Mapping

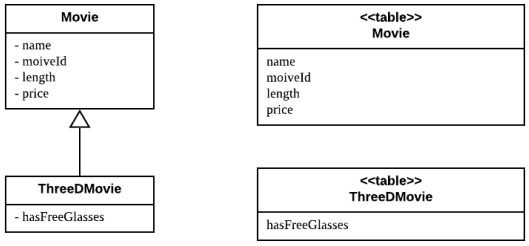
This pattern is used to save the many-to-many relationship between database tables. It is implemented in out system as well. For example, the relationship between Cinema table and Movie table in database is many-to-many, as a result, an association table called Cinema_Movie is created in the database and the table is used in the system for searching all cinemas that could display one specific movie. The advantage of this pattern is that it's necessary and simple to represent many-to-many relationship in database and the mapping between the domain relationships and database tables could also be done simply and mechanically.

7.4.4 Embedded Value

This pattern is used to map one table's information into several objects in the Domain Layer when some object is more suitable to be represented as several classes in Domain Layer but not necessary to be stored separately in the database. Our system implementing this pattern by the Session class and TimeRange class in Domain Layer, which is stored as one table (Session) in the DB. The reason why we do this is that the TimeRange is quite different with other attributes of session object and it's reasonable to define it separately as a domain object. But in the database table, we usually collect time range information and other session information together, especially after implementing lazy load pattern on Session class. The advantage of this pattern is that it could make the relational database and related domain objects neater.

7.4.5 Class table inheritance

This pattern is used to represent an inheritance hierarchy of classes with one table for each class. In our system, it is used between Movie and ThreeDMovie classes and Movie and ThreeDMovie table. The detailed information is as follows:



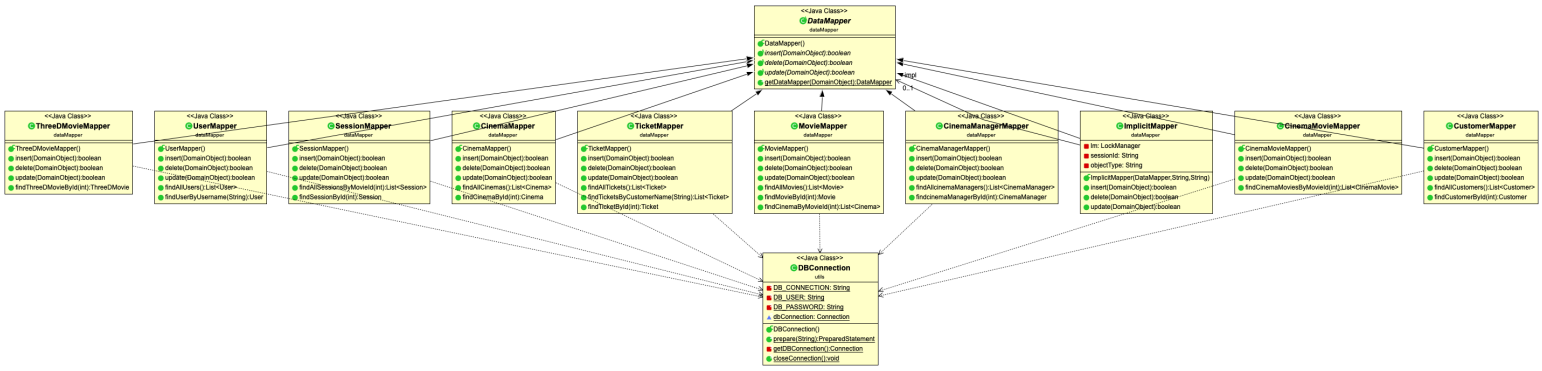
The reason why we use this inheritance is that it's easier to implement than concrete table inheritance and the table design is clear and without any wasted space (which is one advantaged compared with single table inheritance).

8. Data Source Layer

8.1 General design

Data mapper pattern is used in the data source layer. The significant advantage of applying data mapper pattern is that it could keep other layers and objects independent with the database. All these layer could behavior as they don't know the existence of database. Apart from that, Unit of Work (will be explained later) and Identity Map (Already explained in Service Layer Section) are also used in the Data Mapper Layer to optimize the system's performance.

The structure of Data Mapper Layer is as follows:



All mapper classes extends DataMapper Class, which could make the design more clear and also essential for implementing UnitofWork Pattern.

8.2 Responsibilities of mapper classes:

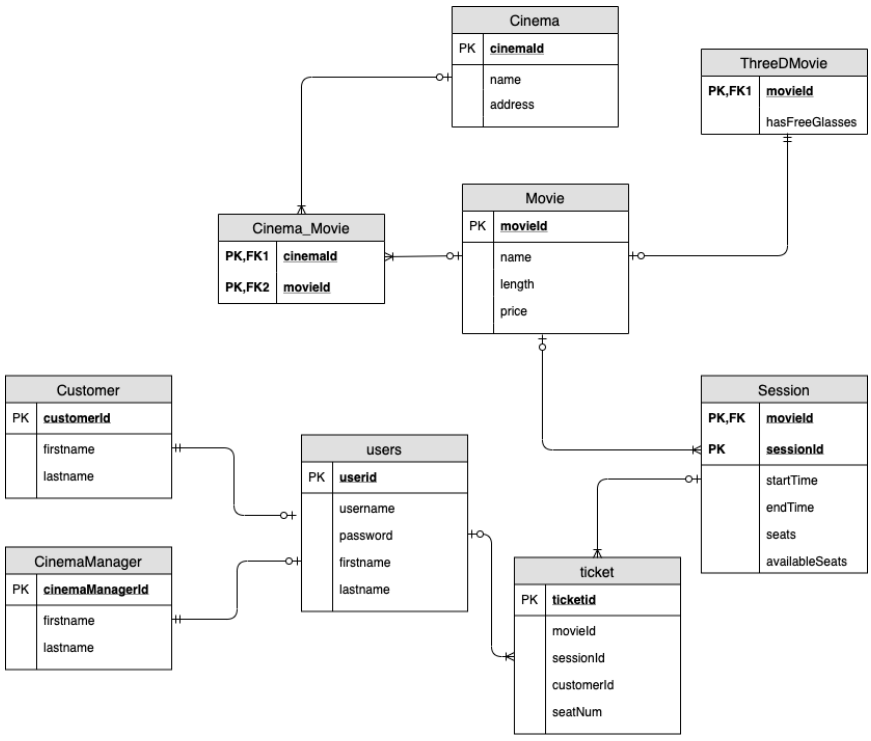
Mapper	Responsibility
DataMapper	The class that all the other classes in DataMapper Layer will extend
MovieMapper	Movie CRUD mapping between domain/service layer and DB.
SessionMapper	Session CRUD mapping between domain/service layer and DB.
CustomerMapper	Customer CRUD mapping between domain/service layer and DB.
CinemaManagerMapper	CinemaManager CRUD mapping between domain/service layer and DB.
ThreeDMapperMapper	ThreeDMovie CRUD mapping between domain/service layer and DB.
CinemaMovieMapper	CinemaMovie CRUD mapping between domain/service layer and DB.
CinemaMapper	Provides Viewing function of Cinema table in DB.
UserMapper	User CRUD mapping between domain/service layer and DB
TicketMapper	Ticket CRUD mapping between domain/service layer and DB

Mapper	Responsibility
ImplicitMapper	Lock CRUD mapping, sits between domain/service layer and other DataMapper classes. In charge of acquiring and releasing offline locks so that other domain logic classes don't need to implement acquiring and releasing lock function in their own part.

8.3 Unit of Work

The UnitofWork class use the Data Mapper Layer in its commit function. By using DomainObject Class(all classes in Domain Layer extends DomainObject Class) and DataMapper(all classes in Data Mapper Later extends DataMapper Class), the commit function is able to parse one object to its corresponding mapper based on its class type, which makes the Unit of Work could tag different objects at the same time and commit to database together.

9. Database Design & ER Diagram



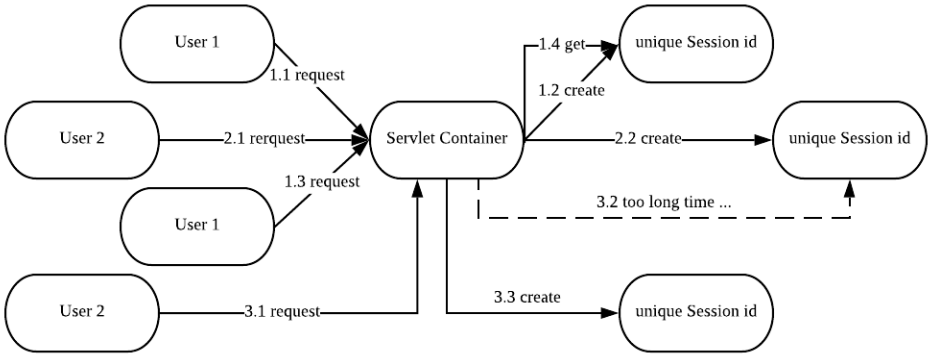
10. Additional Patterns

10.1 Session

10.1.1 General Design:

Session is necessary for our application as there are multiple across requests and we need to keep it stateful. Here we chose the *HttpSession* function inside Servlet to store data across requests, which uses **Server Session State** to store the session information.

How the session works in our system is as below:



- When a new user sends one request, the *Servlet Container* will automatically create one new *session* with a unique *session Id*.
- When the user sends some other requests within time, the container will check and return the existed *session* which belongs to that user.
- If the request has over time, and the session belongs to the user has already been destroyed, the system will treat the request as new one and create new *session* with unique *session Id* again.

The reasons of selecting this pattern are as follows:

- Using the Http Session Class, we could simply add the session Id as one new parameter on all related functions, which means we could change much less on the original system. This could decrease the possibility that we change the original system a lot and then cause some big issues.
- **Server Session State** is very suitable for one server system, as the states could be resisted in a serialized form. Also, this pattern works well with other system patterns, including **security** and **concurrency control**.

10.1.2 Comparison with Alternative Patterns

- **Client Session State:** this pattern has many similar features with **Server Session State**. However, using *Cookies*, which is the most common way to achieve *Client Session State*, may have potential security

problems. Also, *Server Session State* works better with other patterns like *Concurrency Control* because the unique session Id could be used as identifier of different clients.

- **Database Session State**: the **performance** of this pattern might be a big problem under our current situation. As the Database and Server are distributed, the transmission of session information between remote Database and remote Server will be expensive, especially for the free Heroku Server. Apart from that, it is much more **complex** to implement this pattern compared with current session pattern.

10.1.3 Pattern Implementation Justification

The Session pattern is widely used in the system, and two of the most important parts of all are the **Concurrency Control** and **Security.**

- **Concurrency Control:** the session ID information in sessions is used in the *ExclusiveWriteManager* function as the representation of **owner** for concurrency control. When a client asks for some exclusive writing permission of some resources, the Concurrency Control component will use the session ID of the client as the primary key in Lock Database table, and query the table to make sure whether the client could obtain the requested permission. Also, the releasing lock function is achieved by deleting one specified session ID & resource ID record from the Lock Database table.
- **Security**: the user information in sessions is used by security pattern for authentication, authorization and some other functions. In our system, when one user send some request to our system, **Shiro**'s Authentication enforcer will firstly check whether the information is valid. After checking it as valid, the Authorization enforcer will check the "role" information based on the account information in session, and then only gives the user the permissions relevant with his role.

10.2 Concurrency Control

10.2.1 General Design

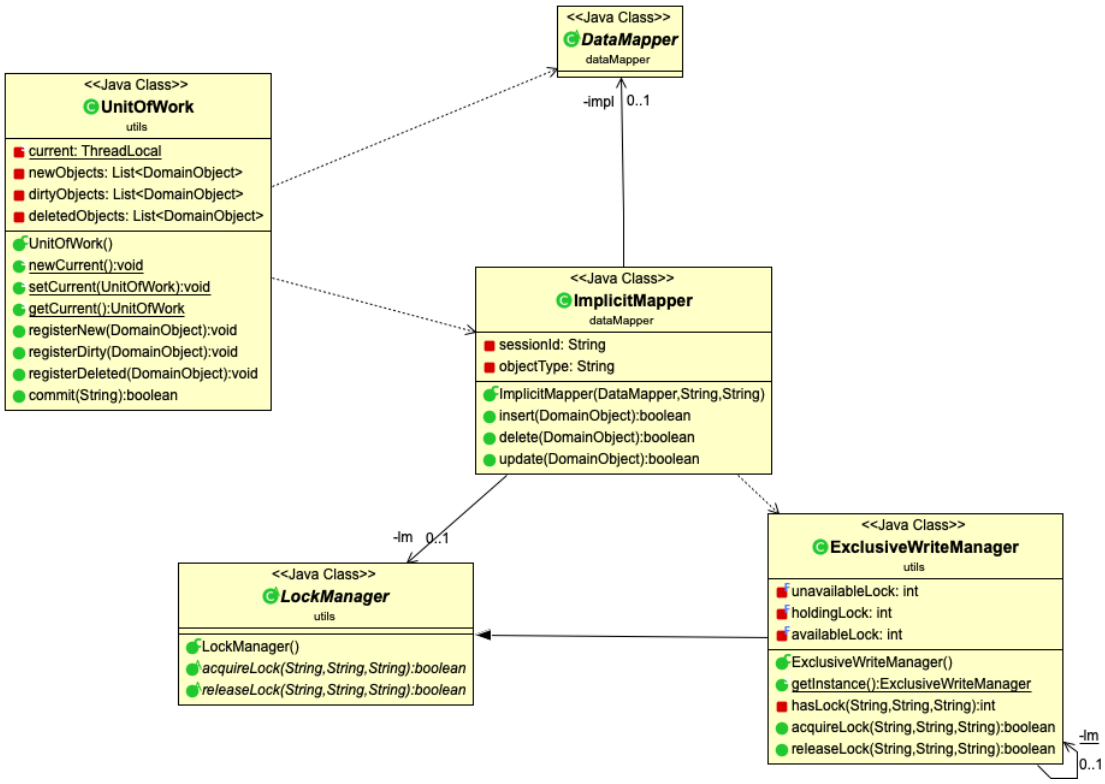
Concurrency Control is an essential part of enterprise application. In our system, we implement **Pessimistic offline lock** and **Implicit lock** to control the concurrent requests. Some detailed design information is shown as below:

- ***Pessimistic Offline Lock****:*
  - *Lock Type & Lock Protocol:* **Exclusive Write Lock** is used in our application. However, we make a little change on applying this pattern: As buying tickets is one frequent scenario of our system, making sure multiple users could buy ticket (will update the movie's session information) is very important. Different from normal Pessimistic Offline Lock pattern which will block the total edit processes, we make the client only acquire for exclusive lock after clicking on the Submit button. In this way, we successfully increase the pattern's liveness feature, which makes multiple users could edit booking-ticket information at the same time, and conflict will only happen when different data is **submitted** at the same time. This implementation makes the liveness of Pessimistic Lock become almost the same as Optimistic Lock. However, this will also bring the losing data disadvantage. The final selection is based on some trade-offs on liveness and losing small amount of data in our system.

- *Lock Manager:* Only one lock manager is implemented in our system, which is the *ExclusiveWriteLockManager* class. All classes needs concurrency control will acquire singleton from this class.

- ***Implicit Lock***
  - A Class named *ImplicitLockMapper* is built in our system to prevent the situation that all the services implement acquiring lock and releasing lock functions inside. To achieve this, this class will be called by the *UnitofWork* class in the *commit* function. Thus, as all insert, update and delete operations will have to call the *commit* function originally, we successfully add concurrency control on all related functions.

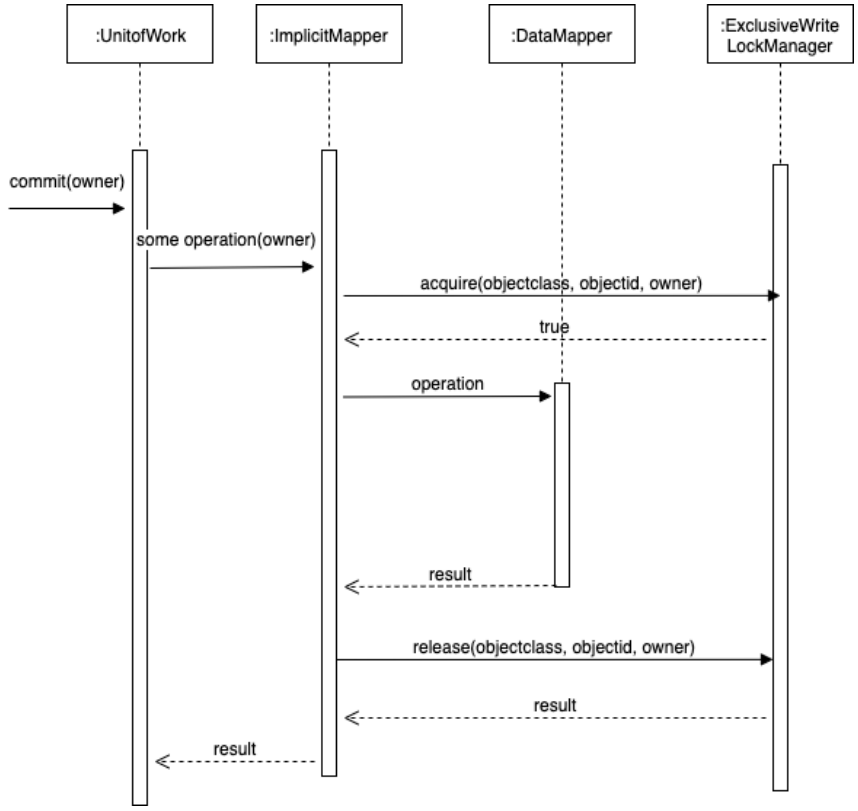
The details could be shown as follows:



Concurrency Control works through following steps

1. Some Service calls *commit* method in UnitofWork
2. *UnitofWork* calls *ImplicitMapper* for insert/delete/update operations
3. These operations will acquire related lock. If successful, *ImplicitMapper* will call relevant DataMapper class to execute the update, otherwise, False will be returned

The following sequence diagram shows this process:



10.2.2 Comparison with Alternative Patterns

- ***Optimistic Offline Lock:*** Basically our pattern has similar features compared with optimistic offline lock. Both have good liveness but may lose some data. However, implementing optimistic lock pattern will be much more complex in our existed system as we need to add the attributes like *version* and *modifiedBy* on all the tables and classes from Database Layer to Presentation Layer. There will be too many changes and may also cause some faults. Therefore, we think our current pattern is a better solution compared with Optimistic lock.
- ***Exclusive Read Lock****:* exclusive read lock will decrease the accessibility and make the system less liveness. Considering the importance of liveness in our system, we don't think this is a suitable pattern for it.
- ***Read/Write Lock****:* this pattern will also decrease the system liveness and it's more complex to implement. Thus, we didn't select it as out concurrency control pattern.

10.2.3 Some Pattern Implementation Justification

Pessimistic offline lock and **Implicit lock** is implemented in many parts of our system. The Pessimistic Lock pattern implementation includes the Lock DB table, and the *ExclusiveWriteLockManager* class. And Implicit Lock calls this class in the *ImplicitMapper* class, which will acquire for lock before changing information in Database and will release the lock after operations completed. Finally, the *UnitofWork* class will call the *ImplicitMapper* class in its commit method and will check the concurrency situation.

10.3 Distribution

10.3.1 General Design

Data Transfer Object and **Remote Facade** pattern is used in our system for distribution feature. Some detailed design information is shown as below:

Data Transfer Object:

- DTO pattern is used in our system to make object be coarse-grained. We implement relevant DTO classes on all domain classes, even through some of them is not used currently.
- Also, for each pair of DTO and domain classes, we have one related *Assembler* class to transfer domain object to DTO object and convert DTO object back into domain object. The *Assembler* classes are also in charge of calling relevant *DataMapper* classes to update or insert data from input DTO objects, which increase our system's performance facing remote clients' insert/update requirement.
- Finally, as in our system, only **one** parameter is needed in deleting operations, which is the object's id, there is not need to convert this single parameter into some DTO object. Thus, we didn't encapsulate *delete* methods in our *DTO* and *Assembler* classes.

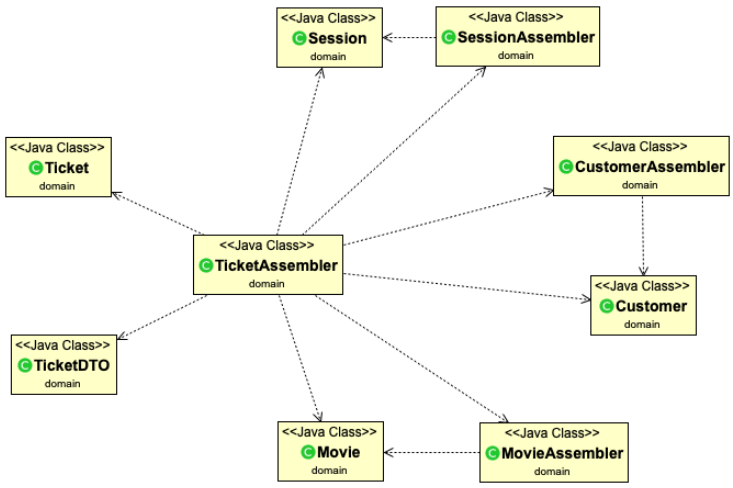
Remote Facade

- *Remote Facade* provides a coarse-grained interface. Classes inside are in charge of return DTO object or JSON strings to remote callers, and it could also collect input (either DTO object or JSON string) and convert into domain object for further operations.
- As our service classes are designed quite well and most of the methods correspond to exactly one relevant Use Case, we implement the remote facade based on service classes, which could make the distribution system obtain low coupling and high cohesion features.

10.3.2 Pattern Implementation Justification

Data Transfer Object:

The detailed implementation class diagram of DTO pattern is below (Here we use Ticket DTO/Assembler/ Object as an example):

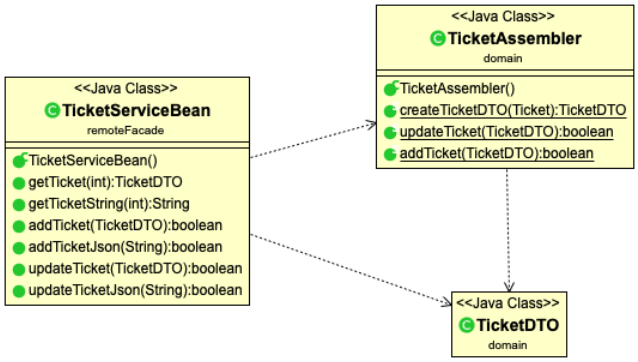


Explanations:

- All domain classes have their relevant DTO classes and Assembler classes. In all the DTO classes, only **primitive types** of attributes are included, as objects are hard to transfer through network by remote callers. For example, as what's showed in the diagram above, instead of having *Movie, Session, Customer* attributes, TicketDTO will hold related *Movie, Session, Customer* as JSON so that it could reduce the number of calls by send all needed information at one time.
- Each *DTO-Assembler* class-pair will only handler **remote** calls, which is separated from the original system where *Service-Domain* class-pairs only handler **local** calls. Therefore, we successfully avoid the situations where some classes have to handler both remote and local calls, which makes the system be less coupling and helps to decrease the possibility of related logic faults.

Remote Facade:

The detailed implementation class diagram of Remote Facade is as below (use Ticket Remote Facade as an example)



Explanations:

- We implemented one relevant ObjectServiceBean for every domain object to make the distributed system have a clear and understandable structure.
- ObjectServiceBean class will be called by remote processes, both JSON and DTO object could be used as transmission data type.
- These remote facade classes will only be used by remote callers, which successfully avoid influence on existed local caller system.
- High remote transmission efficiency is achieved by always trying to send all information needed by one remote request in one time.

10.4 Security

10.4.1 General Design & Implementation Justification

For this part, our system has many security features including *Authentication, Authorization, Validation and Secure Pipe.*

Here are some details:

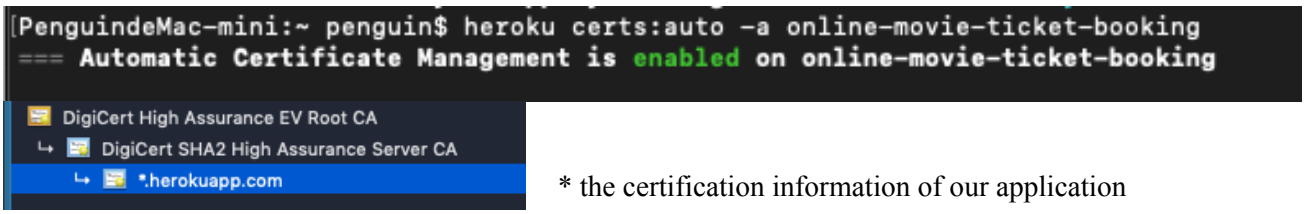
Authentication: This pattern will make sure the user is who they say they are. We implements a centralized authentication enforcement using Shiro, which will verify login information with *users* DB. By the use of filter, we could make sure all identification needed function will ask for login firstly.

Authorization: One centralized access controller is built in the *AppRealm* class in our system. We achieve authorization check by check the "*role*" information in users DB using Shiro. The authorization information and result will be stored in session.

Validation: This pattern will clean the validate data before the data is used in the system. One important usage of this pattern is avoiding SQL injection. To avoid that, we use *preparedStatement* in all SQL operation. In this way, SQL injection could be avoid

Secure Pipe: this pattern use a standardized way to ensure the integrity and privacy of data sent over a network. We achieve this pattern using the functions provide by Heroku.

- By enable security certificate, we ensure our system using SSL in data transmission.

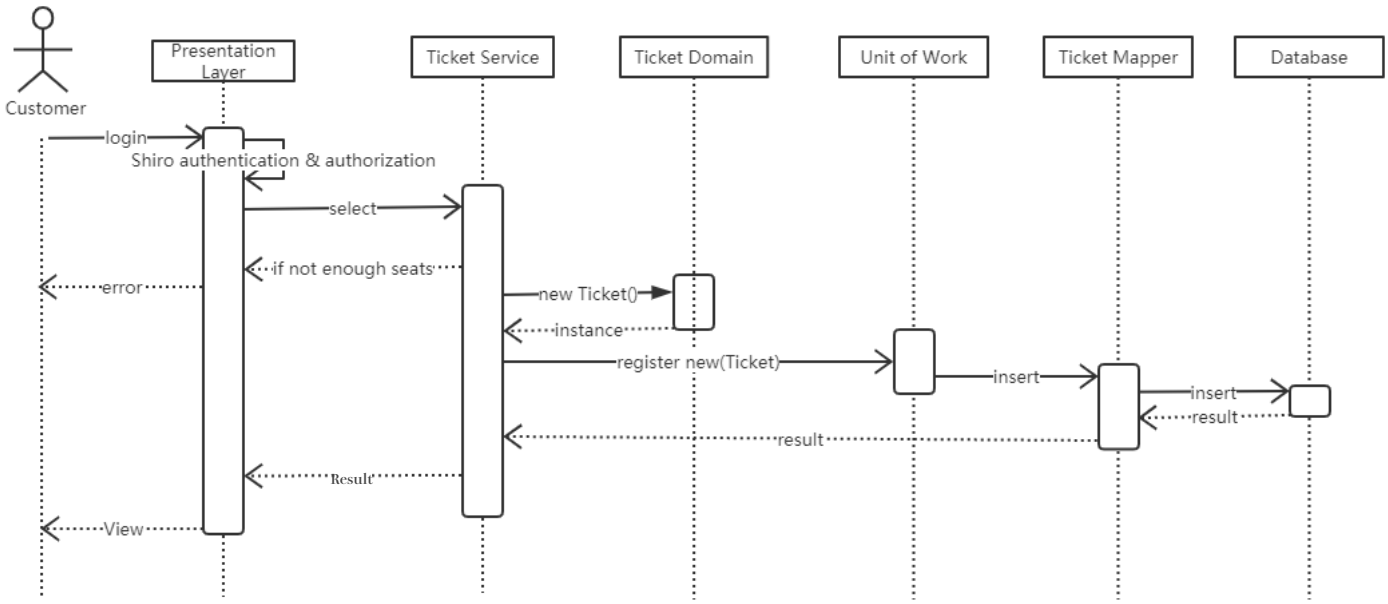


* the certification information of our application

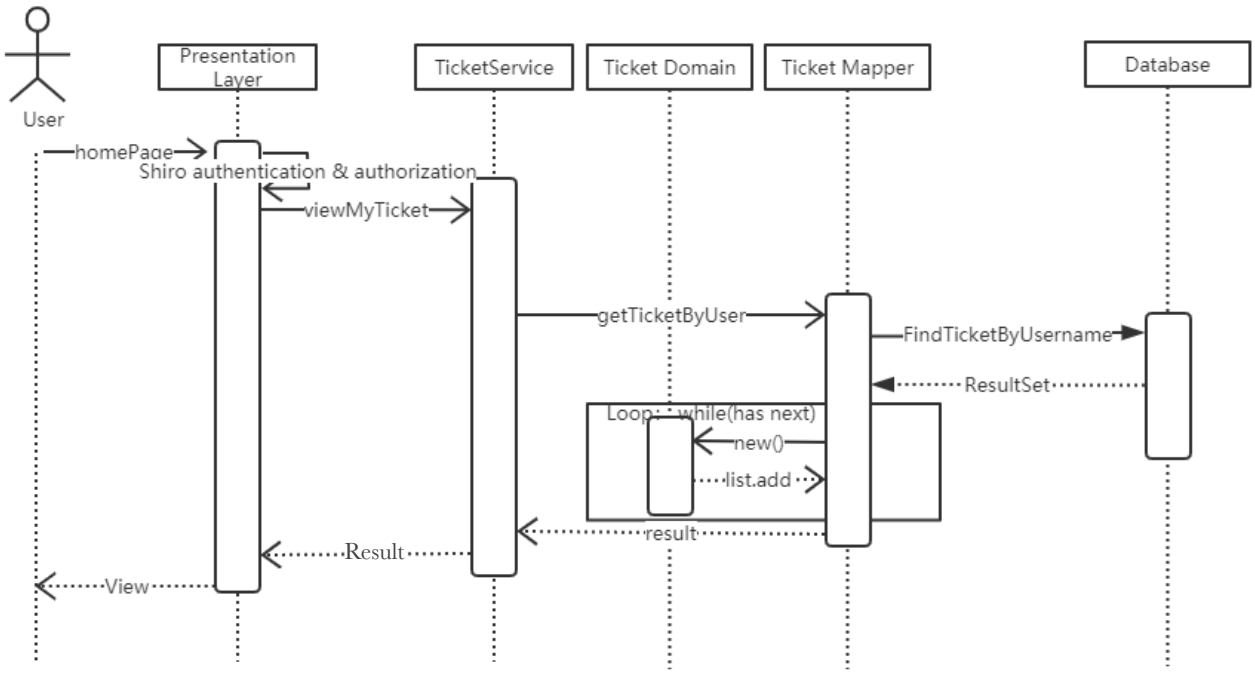
11. Interaction diagrams of Main Use Cases

New ones in Feature B

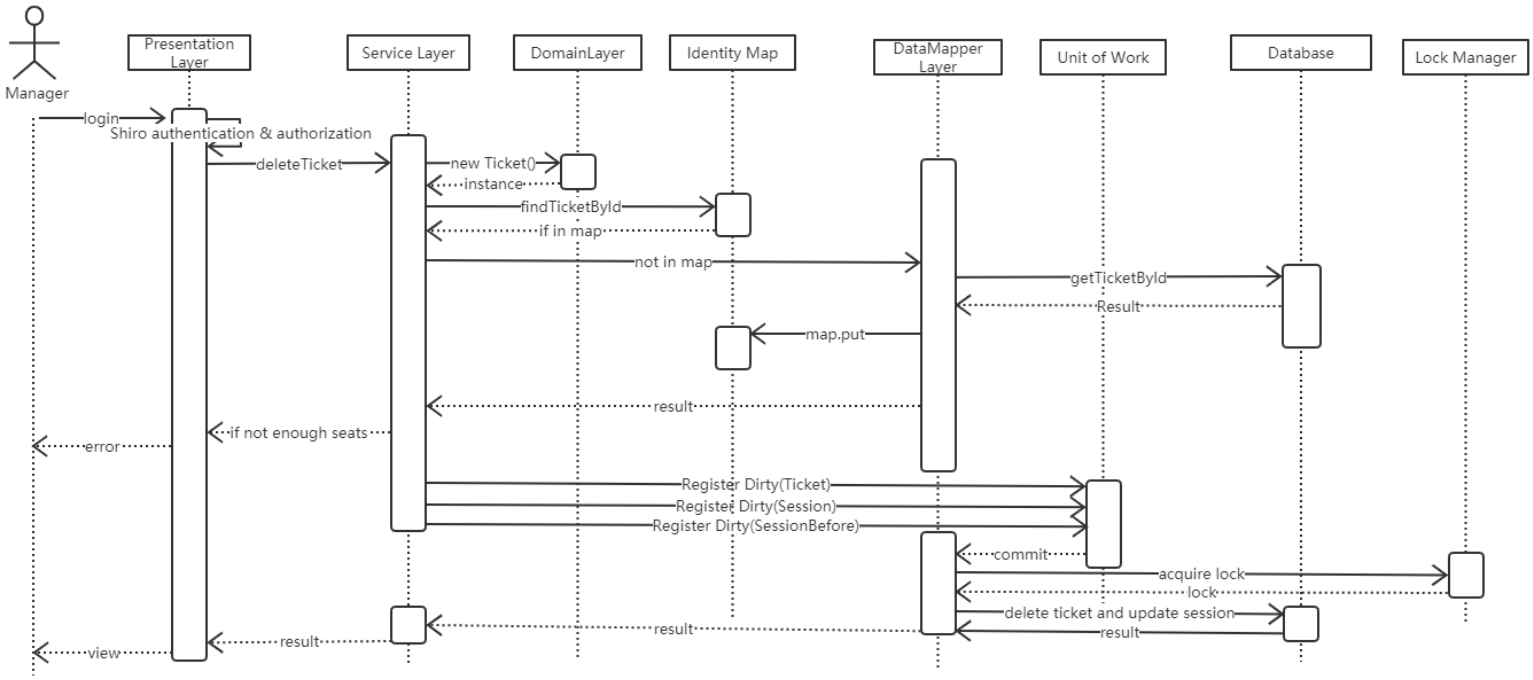
Buy Ticket



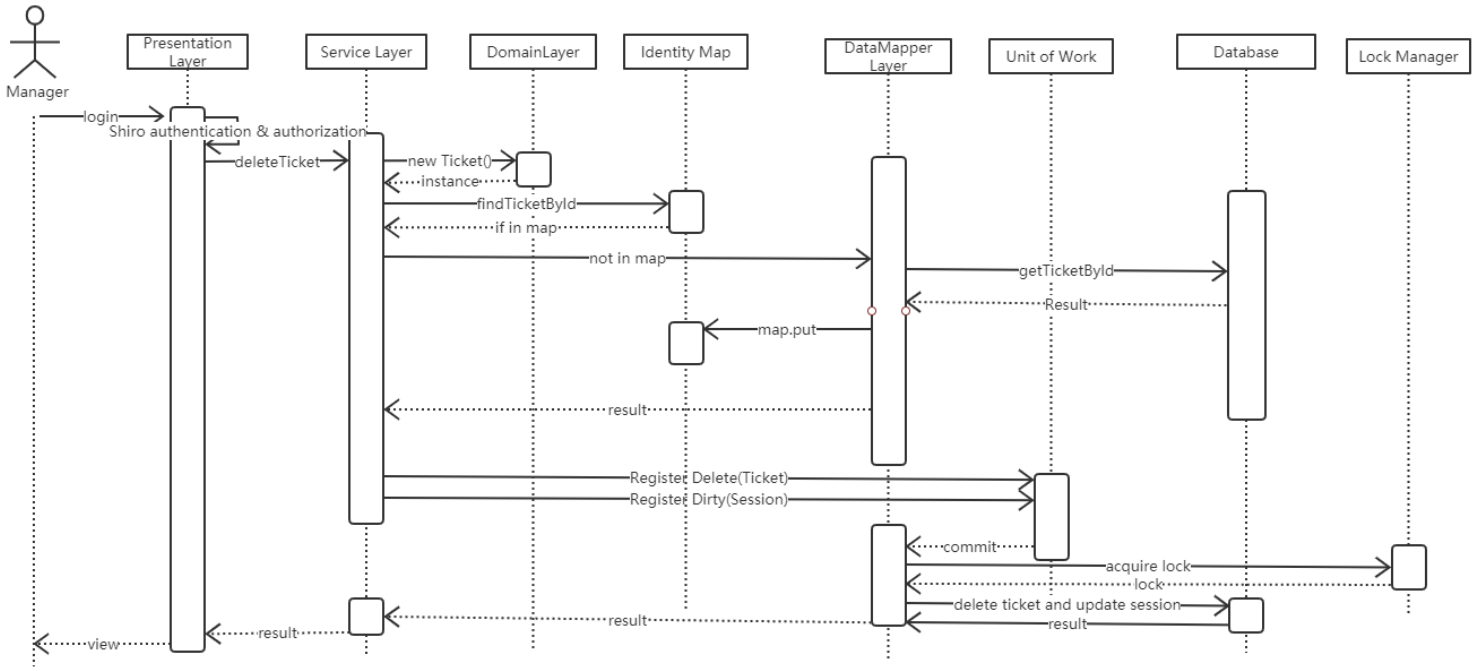
View Ticket



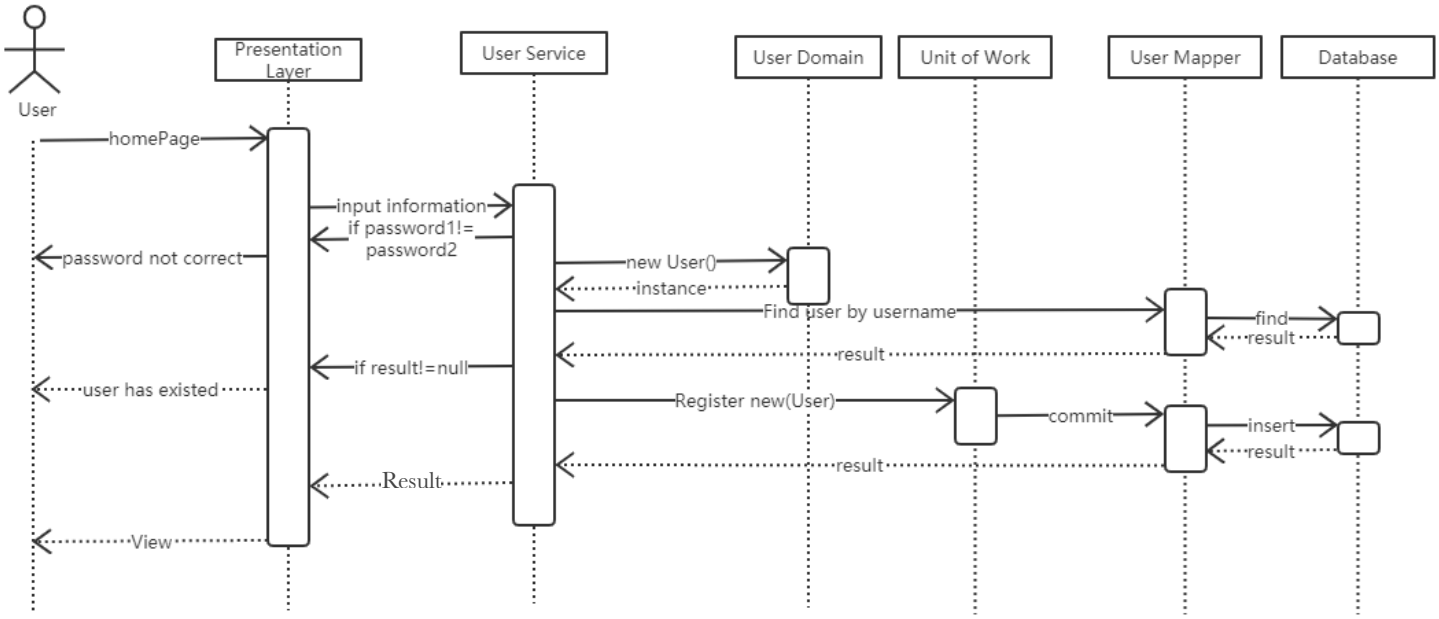
Update Ticket



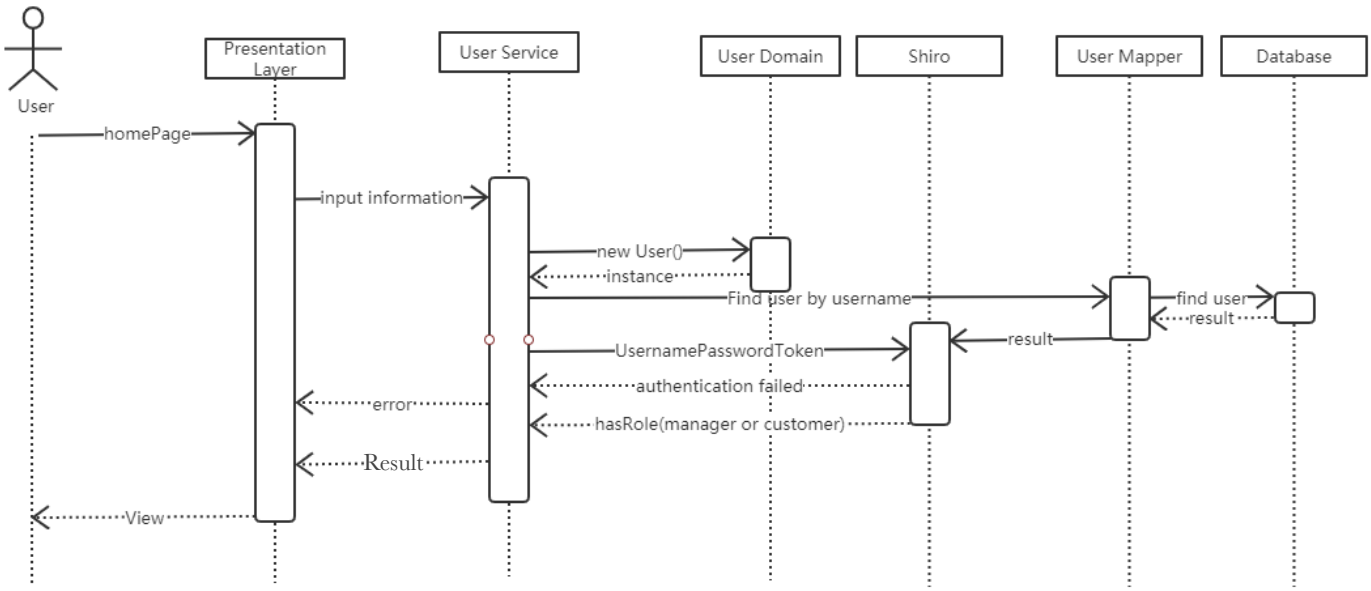
Delete Ticket



User Register

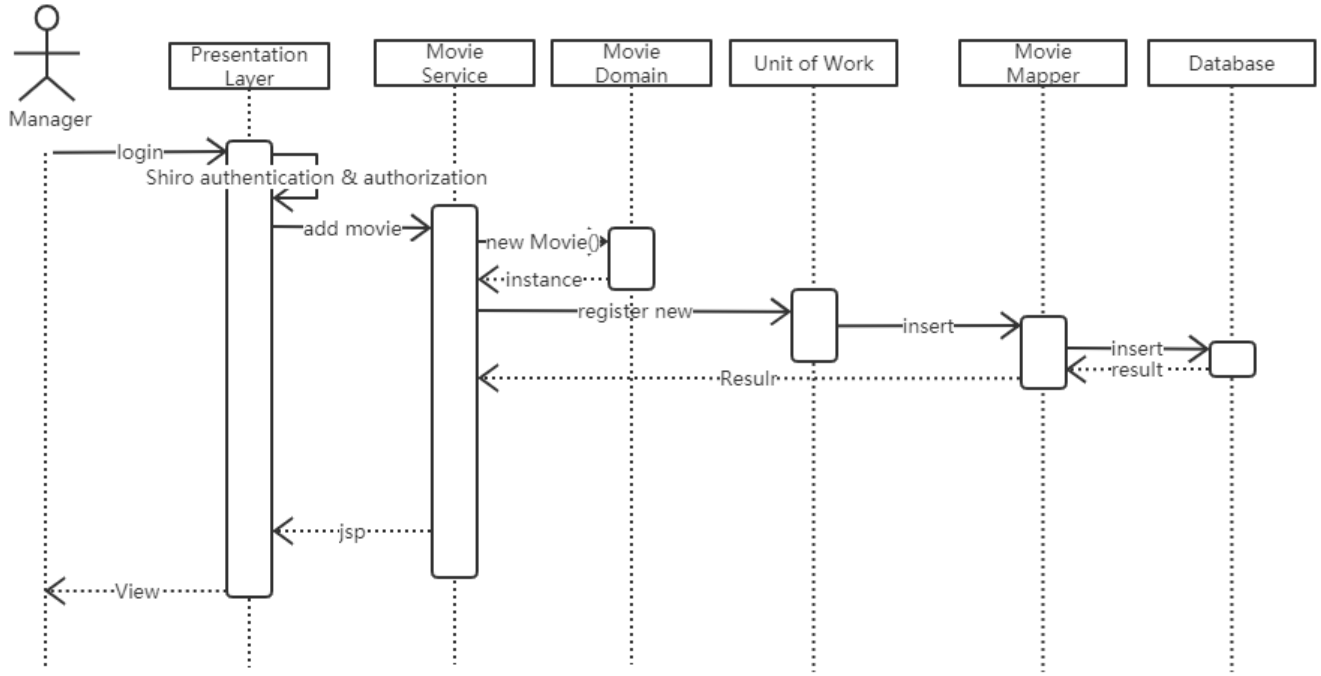


Login

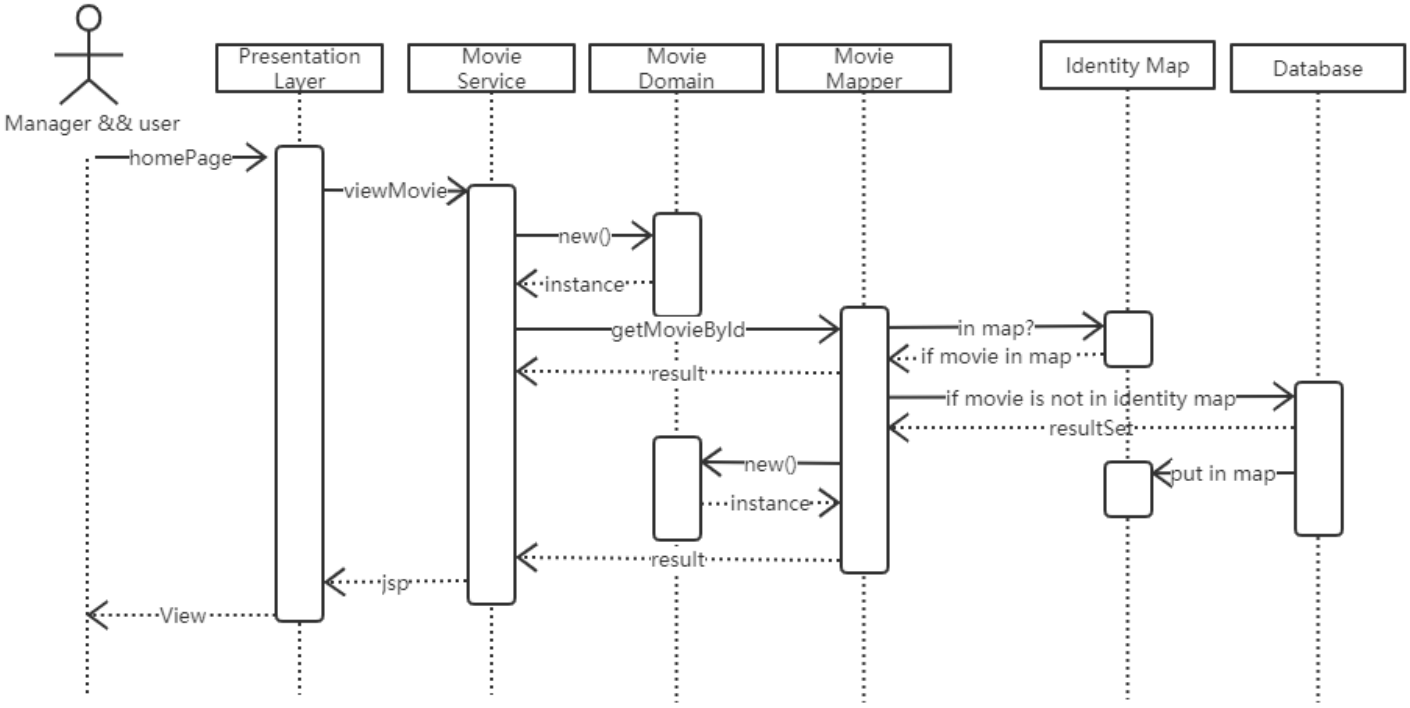


Existed Ones in Feature A (Submission 2)

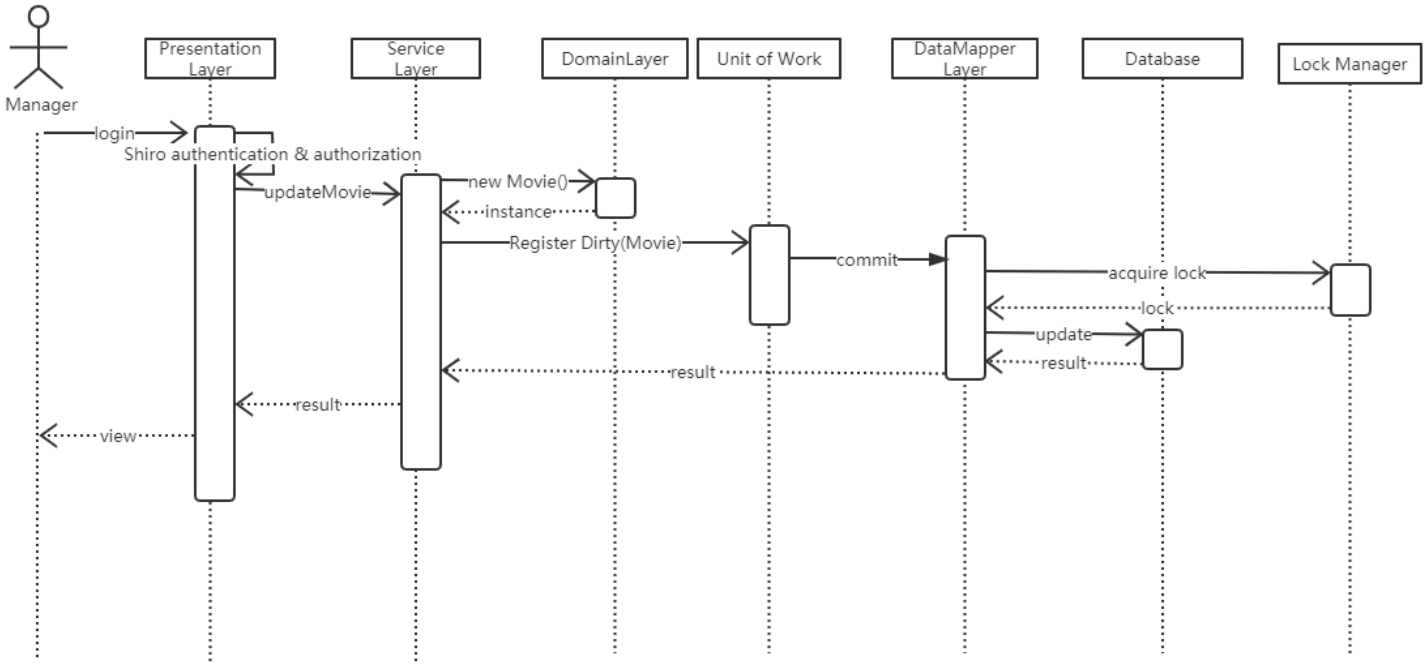
Add Movie



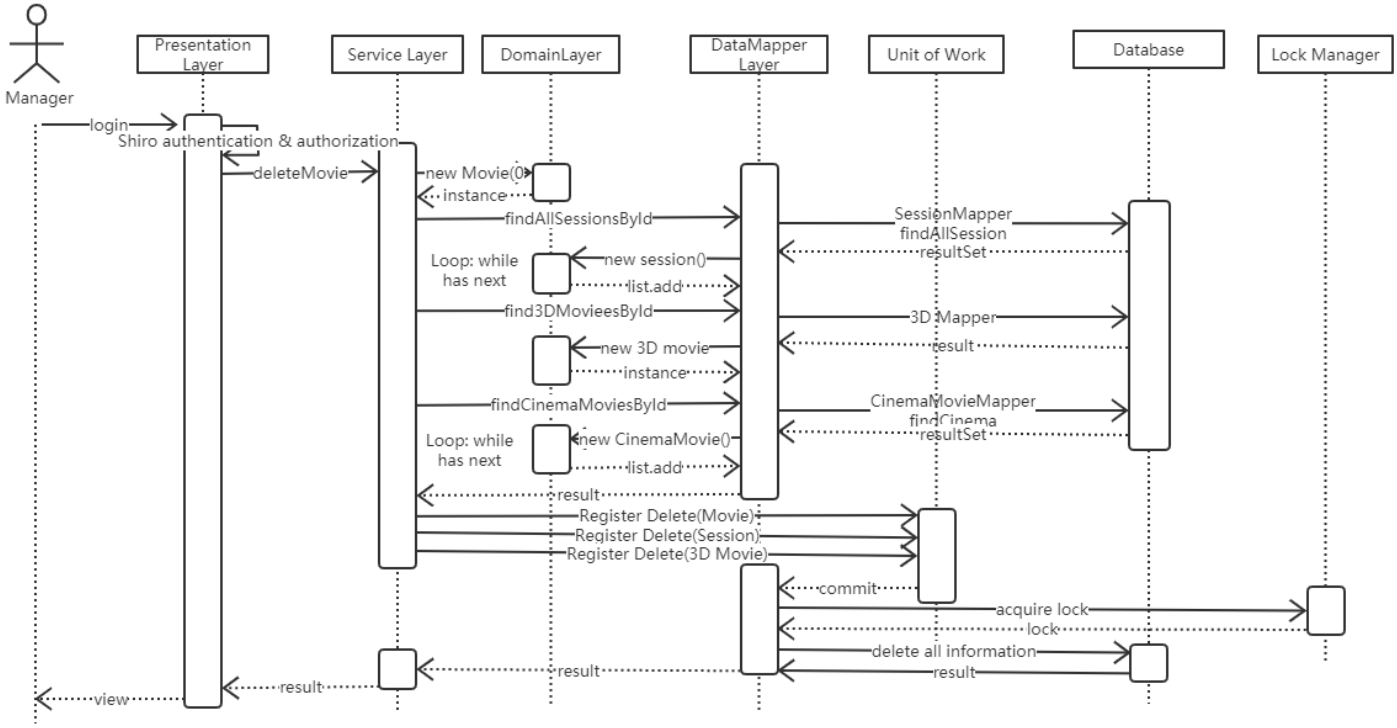
View Movie



Update Movie



Delete Movie



Appendix

A. Deployment URL: https://online-movie-ticket-booking.herokuapp.com/

B. Github:

https://github.com/HaoyangCui0830/SWEN90007-Project-OnlineMovieTicketBookingSystem

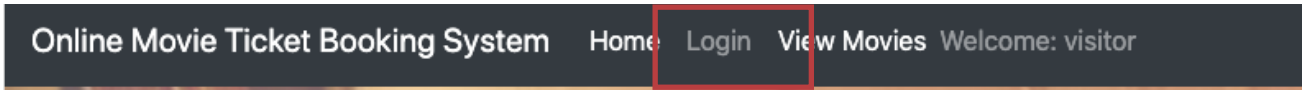
C. Scenario Testing Guide:

There are the steps you can follow to test all functions in feature A & B.

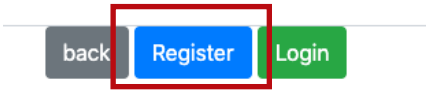
Feature B (Newly added for this Submission)

Scenario 1: Customer Register & Login

1. Visit the website by clicking this url: https://online-movie-ticket-booking.herokuapp.com/
2. Click on the Login Button



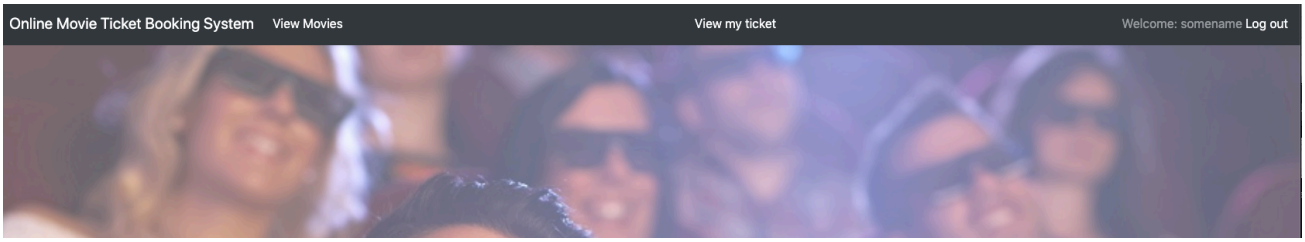
3. Click on the Register Button



4. Input Information, the format should like below:

5. Click on the Submit Button

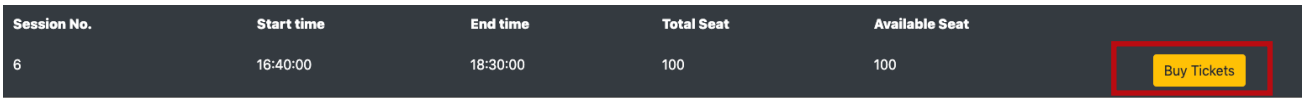
6. Now You have already login as a Customer



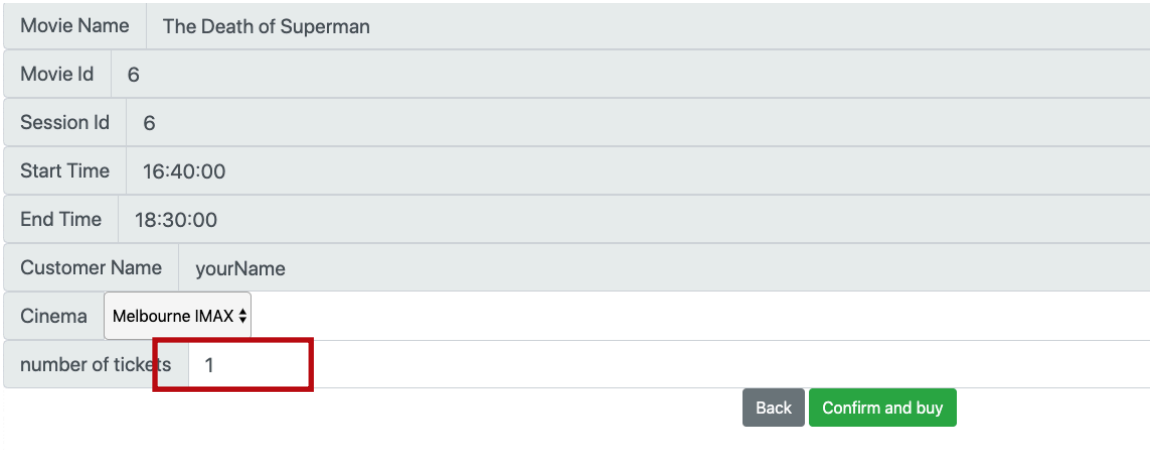
Scenario 2: Customer Buy Ticket & View Ticket

1. Click on the “View Movies” button

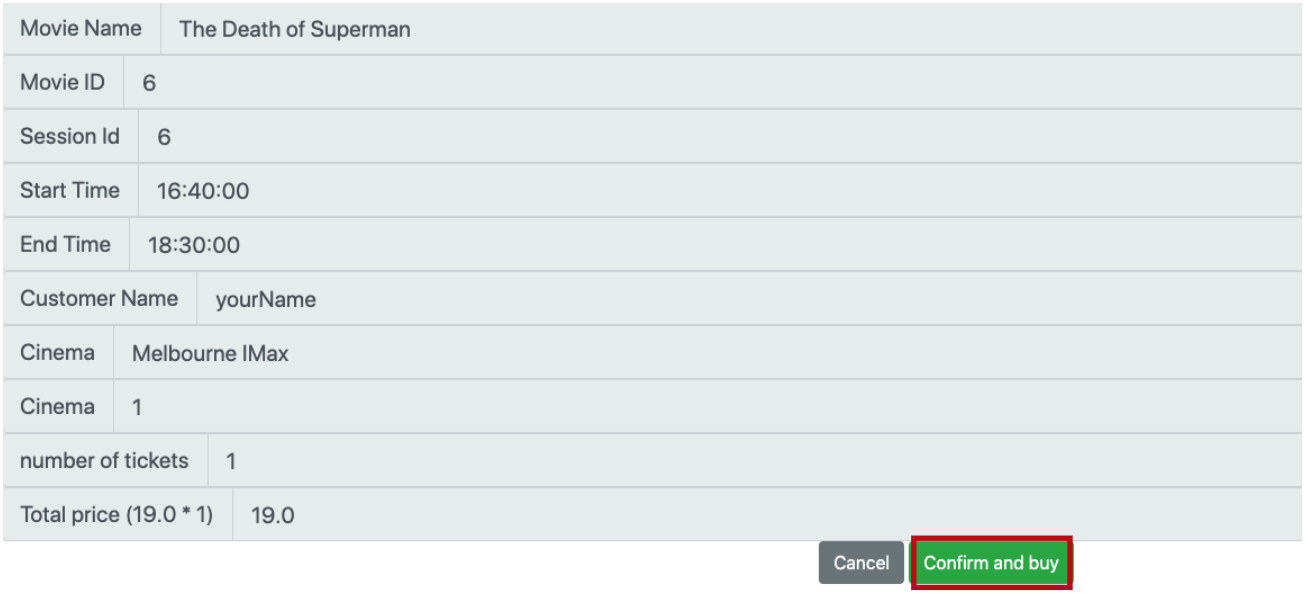
2. You could see all the Movie information, Select one you like and click on the “View” button behind it.
3. Now, Click on the “Buy Tickets” button to buy tickets



4. You could input the number of tickets you want to buy, and then click on the “confirm and buy” button

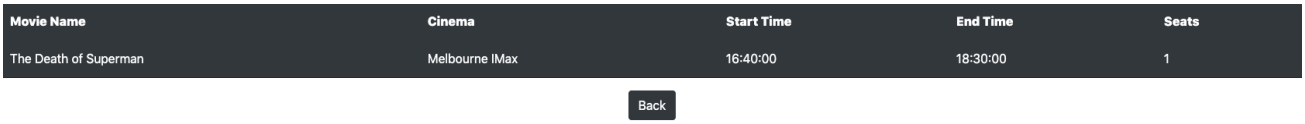


5. Now you could view all your ticket information, if that’s correct, click on the “Confirm and buy” button again to pay



6. Now You have already paid the tickets, Click on the “View my ticket” button to view your tickets

7. You ticket information will be listed in the page



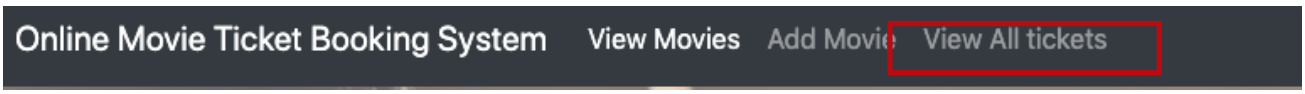
Scenario 3: Manager Edit Ticket

1. Now, If you has login as a customer, click on the “Logout” button to logout

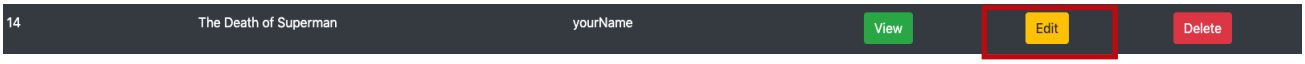
2. Click on the “Login” button again

3. Input the manager account:
Username: admin
Password: 12345678
And then click on the “Login” button

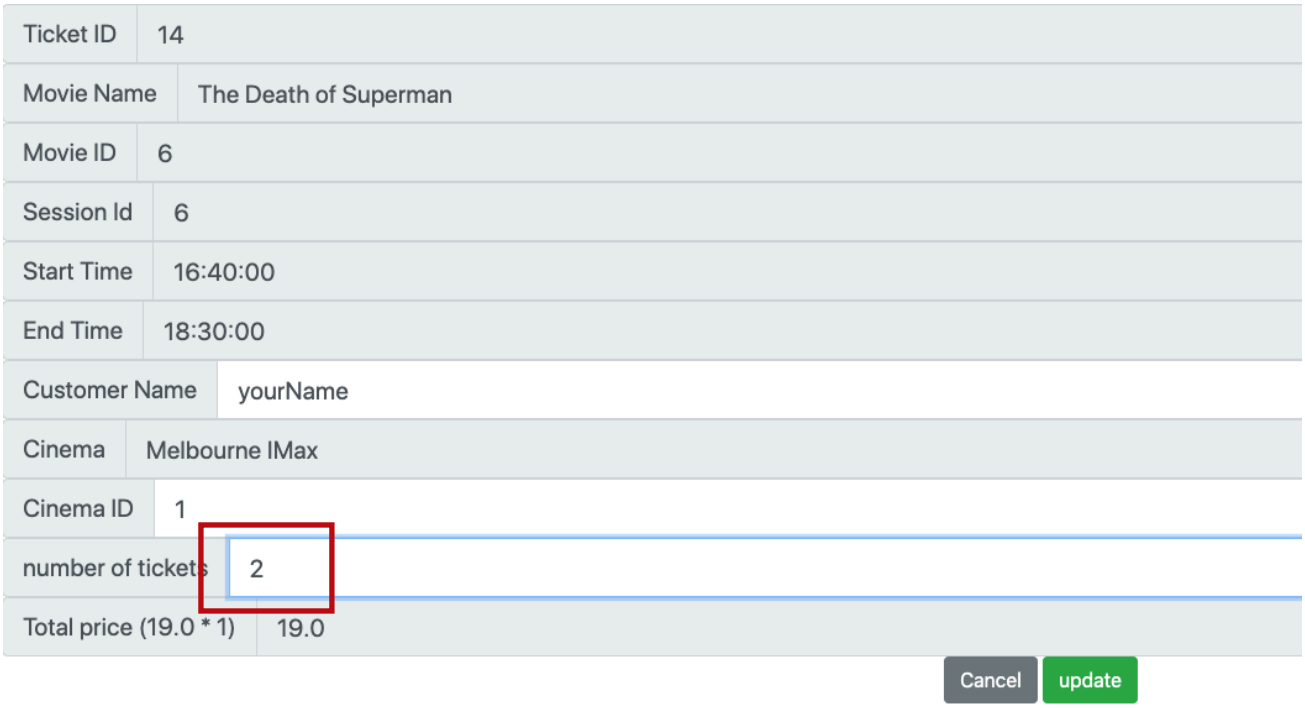
4. Now you have login as a cinema Manager, click on the “View All Ticket” button to do ticket management



5. You could Click on the “Edit” button behind the ticket you just created to edit ticket information



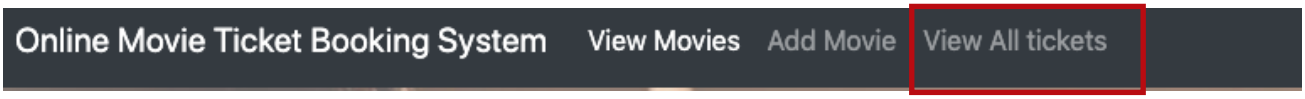
6. Change the number of tickets to a different numbe



7. Click on the “Update” button. Now you have successfully edit ticket information

Scenario 4: Manager Delete Ticket

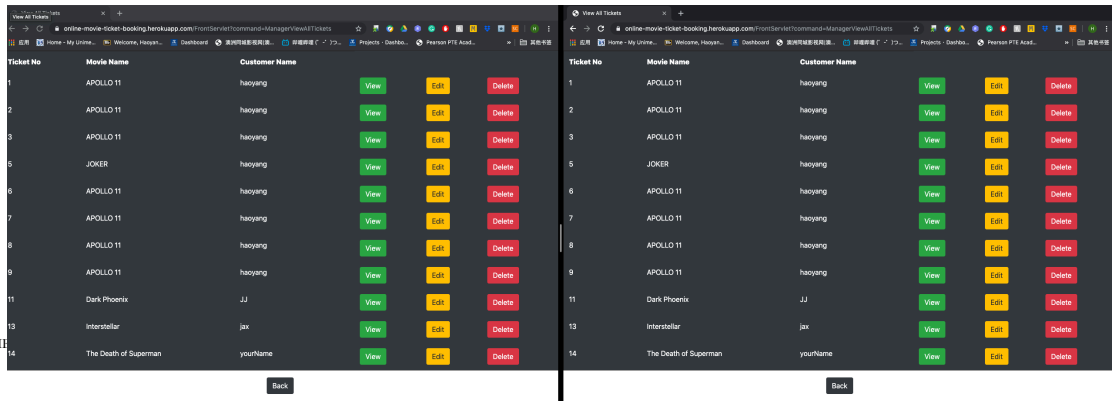
1. Click on the “View All Tickets”



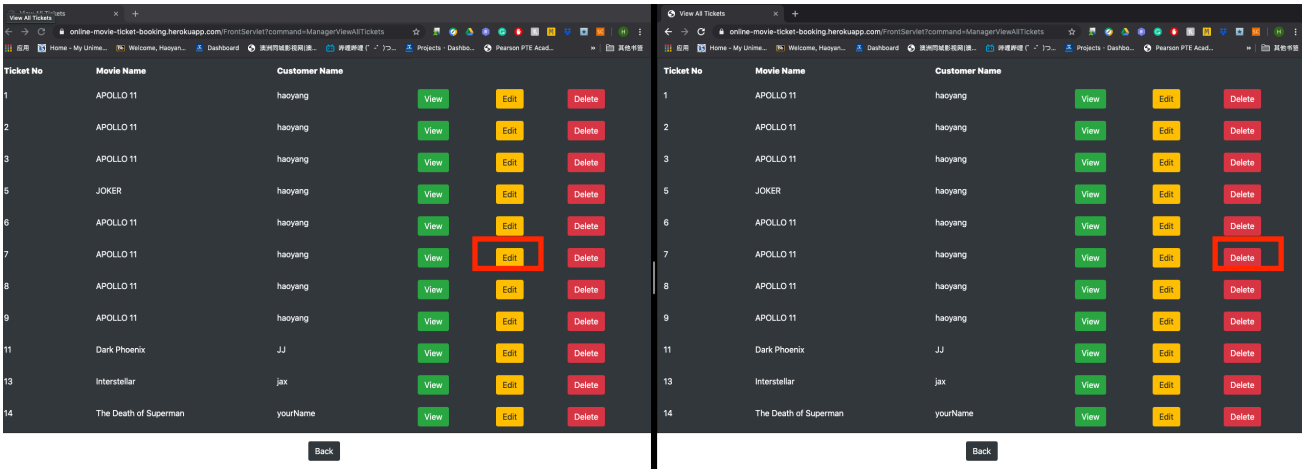
2. Select any ticket and click on “Delete” button
3. And then you successfully delete one ticket

Additional Scenario 1: Concurrency Control Function Test (Using Edit Ticket as an example)

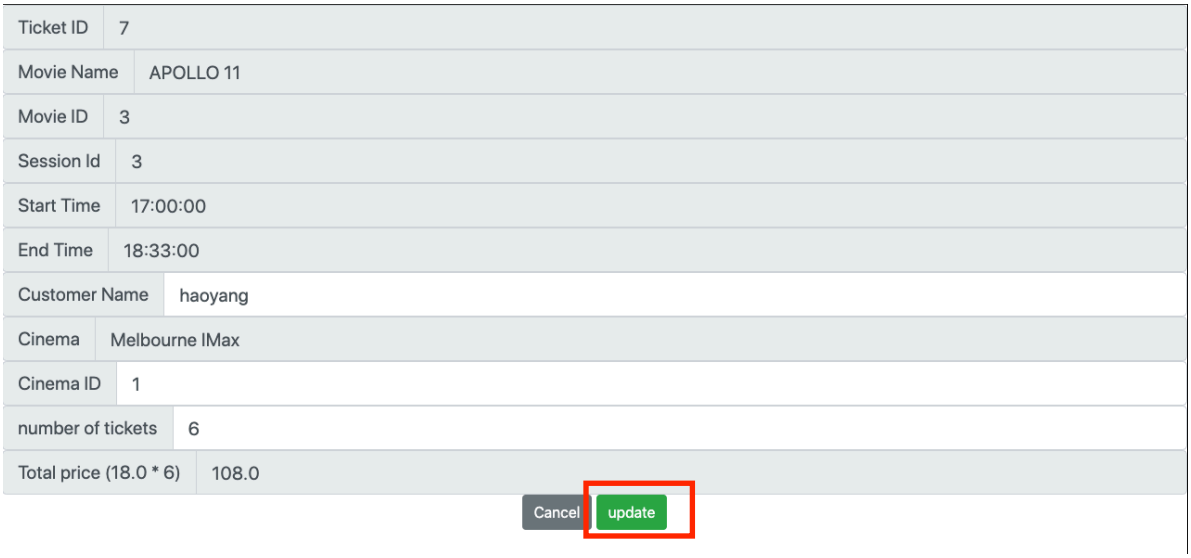
1. Login as manager using the “admin” account
2. Click “View All Ticket” button, copy the URL and open a new window. Paste the URL and open another “View All Tickets” page. It looks like this:



3. In one page, select one ticket and click on the “Edit” button behind it. Please Remember which ticket you select. Then Click on the “delete” button for the same ticket on the other page



4. Now, Click on the “Submit” button on the Edit Page



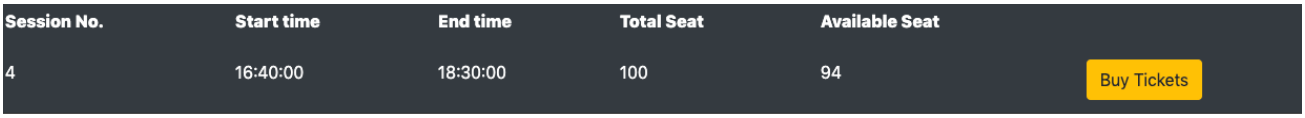
5. It will jump to error page because different processes are changing the same resource at the same time, which means the Concurrency control works.

Additional Scenario 2: Security (Using Buy Ticket as an example)

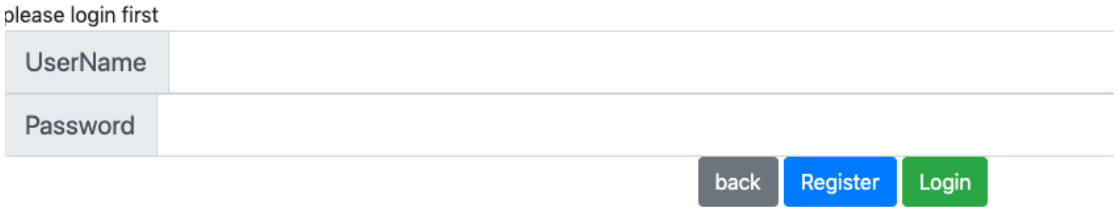
1. Open the Website again: https://online-movie-ticket-booking.herokuapp.com/ . And Click on “Logout” button if you currently login as any role.
2. Click on “View Movie” button.

3. Select one movie and click on the “view” button behind it.

4. Click on the “Buy Ticket” button behind any movie session

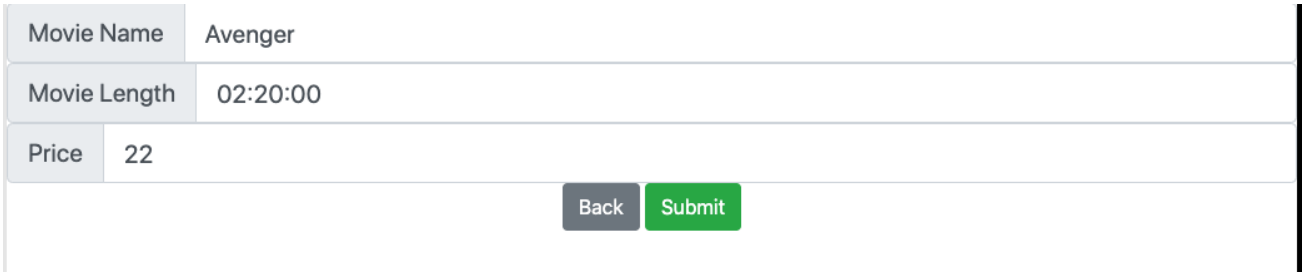


5. Now Security Module works, and it will ask for login(Authentication and Authorization) first.

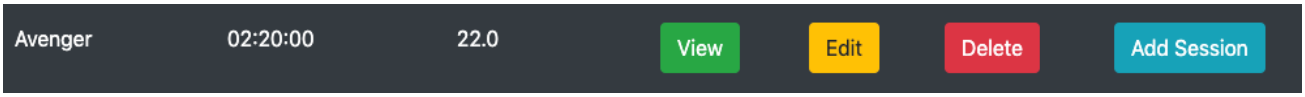


Feature A: (Last Submission)
Scenario 1: Manager Add Movie

1. Visit the website by clicking this url: https://online-movie-ticket-booking.herokuapp.com/
2. Login as a manager using the "admin" account
3. Click on "Add Movie" to add movie information
4. Input movie information, the data format is as below

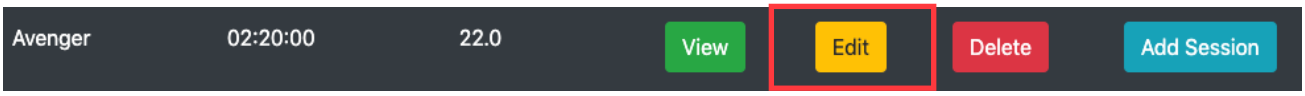


5. Click on "Submit" button
6. Click on "View Movies" button to view the movie just added. You could see the movie has already been added

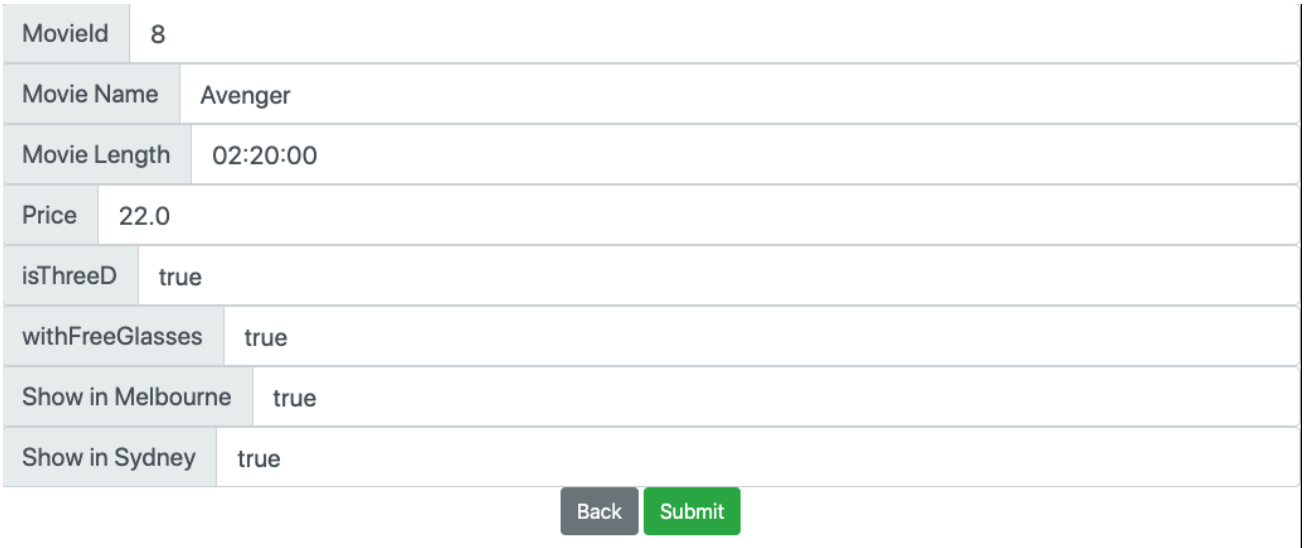


Scenario 2: Manager Edit Movie

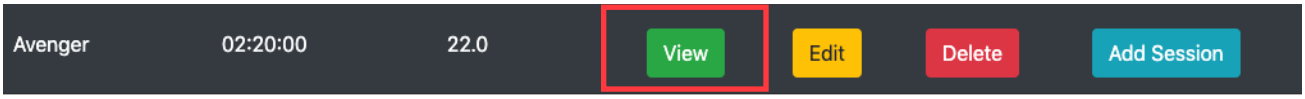
1. In the view movie page, click on the "Edit" button



2. Input the information in the edit page, the data format should be as below. (Please don't change the movieId, it is only shown for view here)



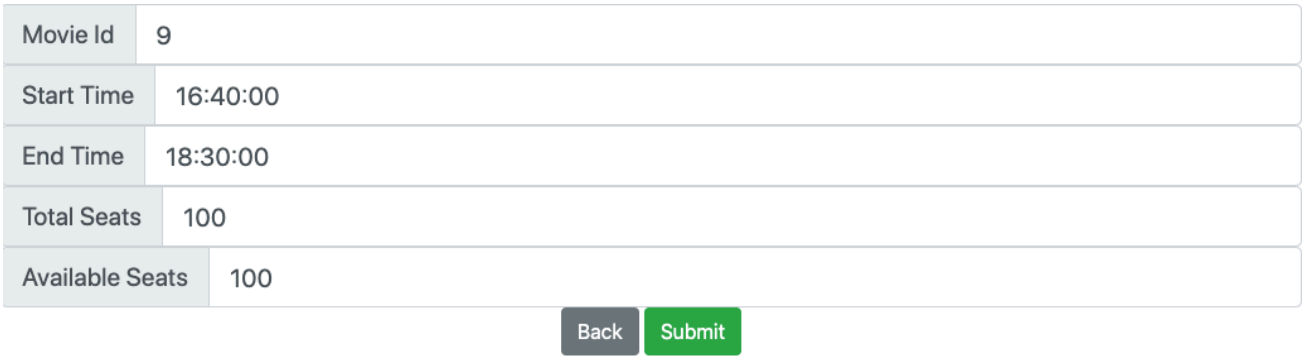
3. Click on the “submit” button
4. Click on the “View Movie” button
5. Click on the “View” button behind the movie we just added.



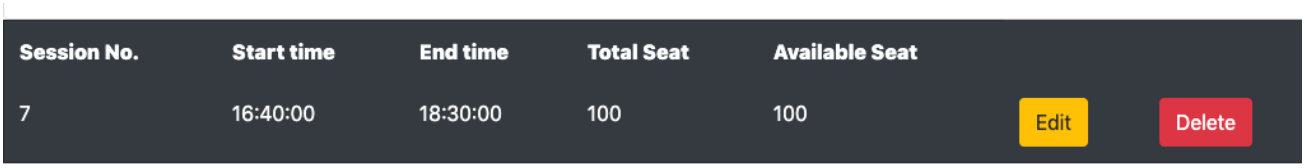
6. All information just be added could be seen here.

Sub-scenario : add session

1. Back to the movie list page
2. Click on the “add session button” behind any movie (Please remember which one you select)
3. Input the information in the edit page, the data format should be as below. (Please don’t change the movieId, it is only shown for view here)



4. Click on the “submit” button
5. Click on the “view movie” button to view the movie list
6. Click on the “view” button behind the movie you just added session on
7. All information is successfully added



8. The session could also be edited or delete if you like :)

Scenario 3: Manager View Movie

1. Click on the “View Movie” button in cinema manager home page
2. Here you can view the movie list which you may have seen for several time
3. By clicking on the “View” button behind any movie you can the movie’s detailed information

Scenario 4: Customer View Movie

1. Visit the website by clicking this url: https://online-movie-ticket-booking.herokuapp.com/
2. This time, Login as a customer using either existed customer account or create new one
3. Click on the “View Movies” button

4. Now, you can see all the movies in the movie list
5. You can also click the “view” button behind any movie to see its detailed information

SOME ISSUES MAY HAPPEN:

1. As we are using the free heroku deployment, some operation might be quite slow. So sometimes if you operate too quick the information may hasn’t changed as it’s still handling. In some extrema situation, it may even crash, like the following logs.

   If that issue happens, please execute the scenario again, and it usually works this time.

2. The Database key of heroku will change automatically after some time, which means the application needs to change the connection information and deploy again. If the system keeps failing connecting (and the error information is the DBConnection function), it might be that problem. If that happens, please email me (haoyangc@student.unimelb.edu.au) and I will change the connection information and re-deploy it.