Online Movie Ticket Booking System

Feature A: Movie Management

Haoyang Cui 886794

Linyuan Zhao 947588

1. Introduction

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. In this submission, we built the first feature in our proposal, which is building a movie management system. There are two roles in this feature, which are the cinema manager and customer.

• Cinema Manager

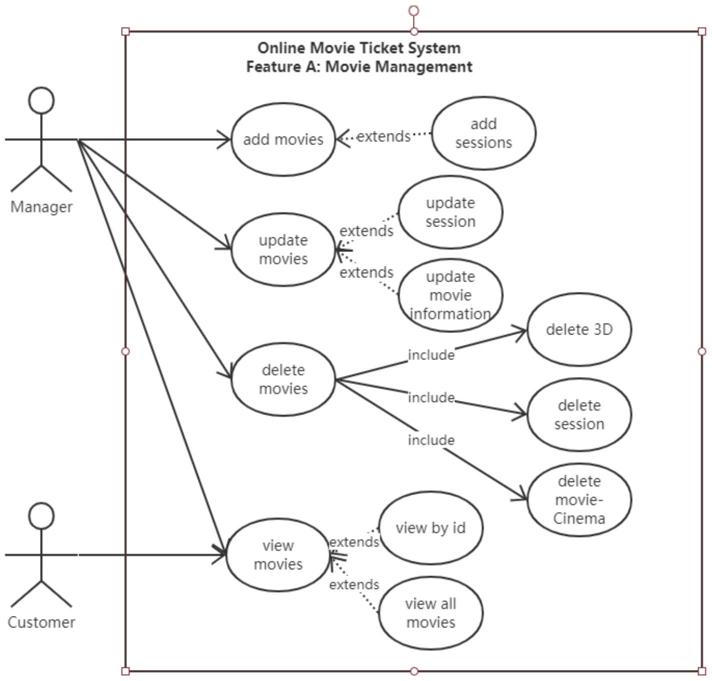
The cinema manager has the right to add, delete, edit and view movie information. Here, the movie information includes both the movie details and other movie related information, including cinemas, sessions and whether it's a 3D movie or not. The word 'session' represents the movie screening information,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

• Customer

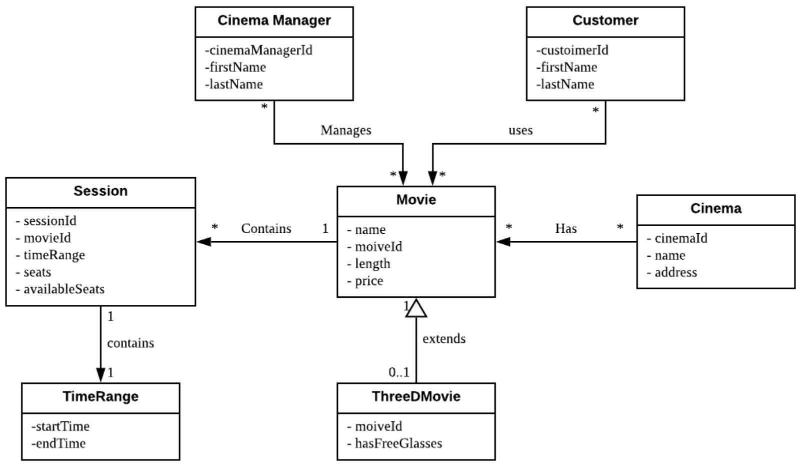
Customer could view all movie information in the system. Customers could firstly get a list of all movies and they could view detailed information of one specific movie by clicking on the view button of that movie's row.

2. Use Cases

Here are the use case diagrams of both roles.



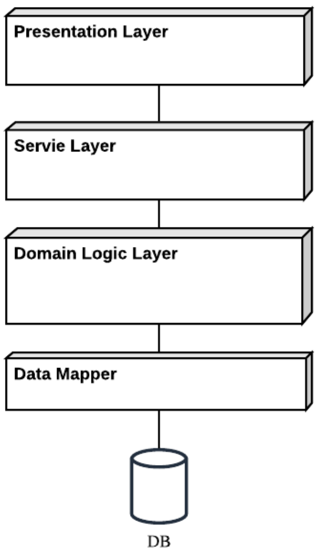
3. Domain Class Diagram & explanation



- **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 sam 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 could be shown 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 to the service layer
Service Layer	• Contains application-specific logic and handle with all application request • 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
Data Mapper	• Provides the mapping between DataBase and Domain/Service Layer • Make the rest of application layers independent with the structure of DataBase

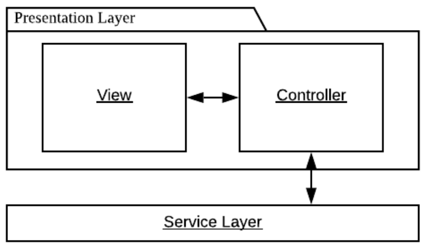
The component diagram of the system is shown as below:

Explanations

5. Presentation Layer

5.1. Pattern for Total Presentation Layer

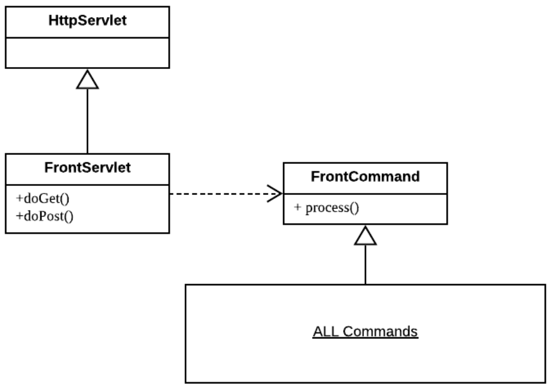
There are two components included in the presentation layer, which are the Views and Controllers. The structure of presentation layer is shown as below.



• MVC structure pattern is applied here.
• Service Layer would provide domain information and handle with front-end requests.
• Controller will handles the interactions between the view and the model (domain objects).
• View will display information and collect user input through a user interface.

5.2 Pattern for Controller

Front Controller pattern is used on the controllers structure. The abstract structure of the controller is shown as below. Every time the FrontServlet gets a request from View, it will parse the url, and then it will create and use an appropriate command which will handle the request.

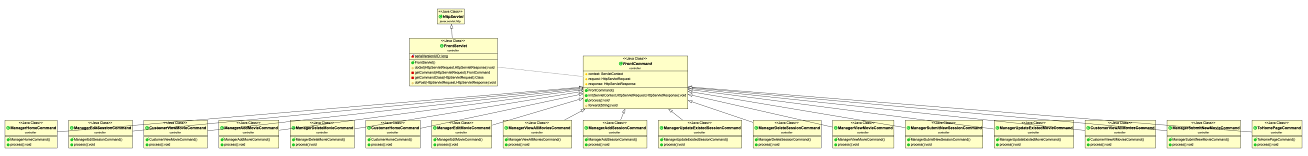


The reason why we applied Front Controller pattern is that:

- There is only one single entry controller, which will make both setting the configuration and implementation easier and more clear.
- As all commands inherent from FrontCommand class and different types of actions could be implemented separately, it's easier to extend this model by simply adding more command classes.

5.3 Actually 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/axxbxkzarc813v6/Presentation%20Layer%20Class%20Diagram.png?dl=0

5.4 Detail Responsibilities of all Commands

Command	Responsibility
CustomerHomeCommand	Return to the Customer Home Page

Command	Responsibility
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
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

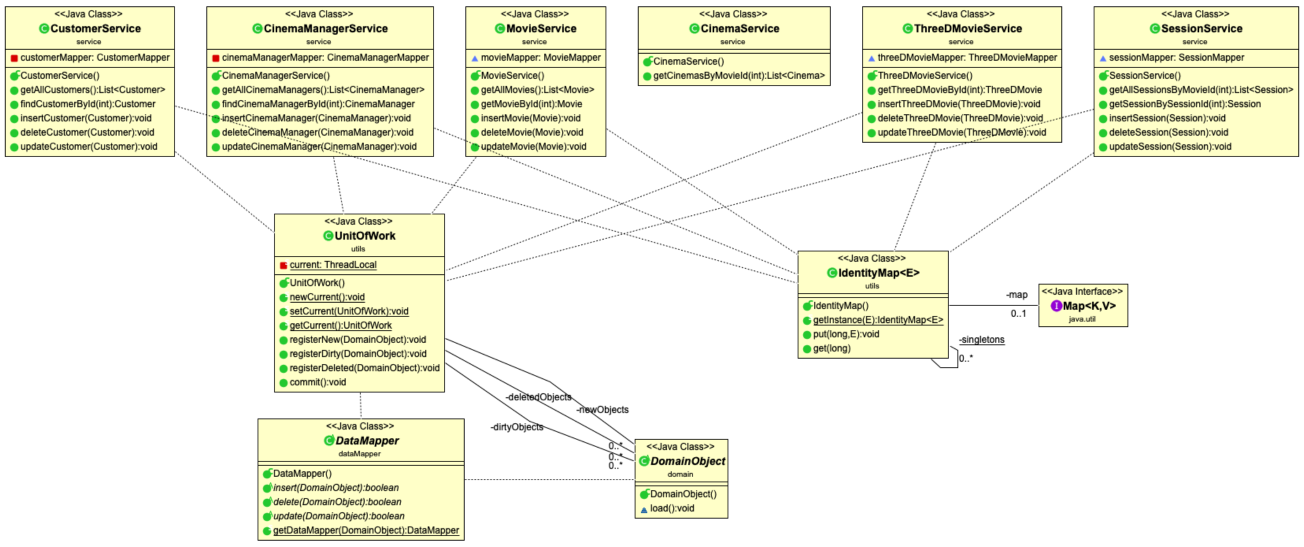
6. Service Layer

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”. Here **Operation Script** pattern 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



<<Java Class>> CustomerService service
customerMapper: CustomerMapper
CustomerService()
getAllCustomers():List<Customer>
findCustomerById(int):Customer
insertCustomer(Customer):void
deleteCustomer(Customer):void
updateCustomer(Customer):void
<<Java Class>> CinemaManagerService service
cinemaManagerMapper: CinemaManagerMapper
CinemaManagerService()
getAllCinemaManagers():List<CinemaManager>
findCinemaManagerById(int):CinemaManager
insertCinemaManager(CinemaManager):void
deleteCinemaManager(CinemaManager):void
updateCinemaManager(CinemaManager):void
<<Java Class>> MovieService service
movieMapper: MovieMapper
MovieService()
getAllMovies():List<Movie>
getMovieById(int):Movie
insertMovie(Movie):void
deleteMovie(Movie):void
updateMovie(Movie):void
<<Java Class>> CinemaService service
CinemaService()
getCinemasByMovieId(int):List<Cinema>
<<Java Class>> ThreeDMovieService service
threeDMovieMapper: ThreeDMovieMapper
ThreeDMovieService()
getThreeDMovieById(int):ThreeDMovie
insertThreeDMovie(ThreeDMovie):void
deleteThreeDMovie(ThreeDMovie):void
updateThreeDMovie(ThreeDMovie):void
<<Java Class>> SessionService service
sessionMapper: SessionMapper
SessionService()
getAllSessionsByMovieId(int):List<Session>
getSessionBySessionId(int):Session
insertSession(Session):void
deleteSession(Session):void
updateSession(Session):void
<<Java Class>> UnitOfWork utils
current: ThreadLocal
UnitOfWork()
newCurrent():void
setCurrent(UnitOfWork):void
getCurrent():UnitOfWork
registerNew(DomainObject):void
registerDirty(DomainObject):void
registerDeleted(DomainObject):void
commit():void
<<Java Class>> IdentityMap<E> utils
IdentityMap()
getInstance(E):IdentityMap<E>
put(long,E):void
get(long)
-map
0..1
<<Java Interface>> Map<K,V> java.util
-singletons
0..*
-deletedObjects
-newObjects
-dirtyObjects
0..*
0..*
0..*
<<Java Class>> DataMapper dataMapper
DataMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
getDataMapper(DomainObject):DataMapper
<<Java Class>> DomainObject domain
DomainObject()
load():void

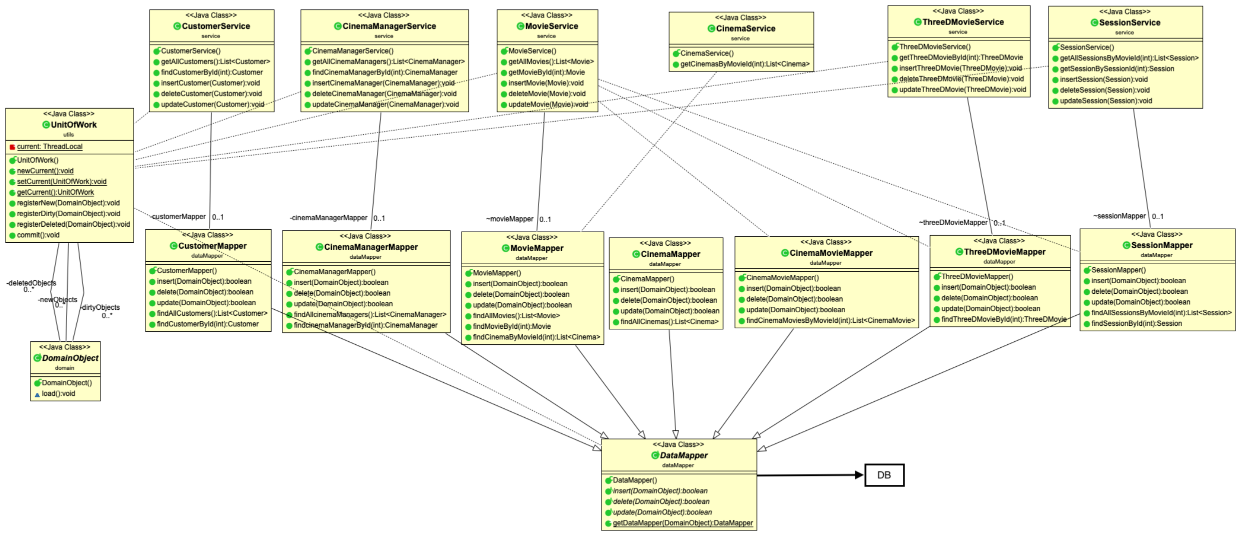
6.3 Responsibilities of all Services

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.

Service	Responsibility
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.

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

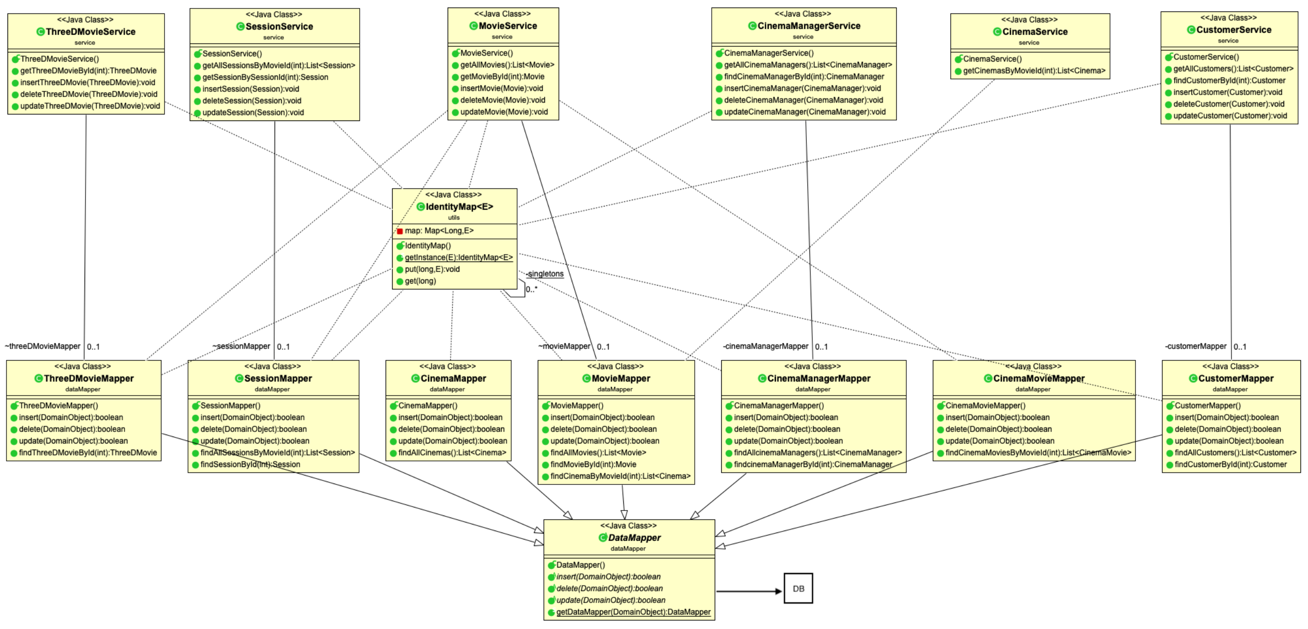


Explanations:
The unit of work pattern is used to maintain a list of objects that are affected by a business transaction. The reason why we applied it is that this pattern could make the DB connected operation more efficient, as it will tag all the objects changes in one business logic operation and 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:


<<Java Class>>
ThreeDMovieService
service
ThreeDMovieService()
getThreeDMovieById(int):ThreeDMovie
insertThreeDMovie(ThreeDMovie):void
deleteThreeDMovie(ThreeDMovie):void
updateThreeDMovie(ThreeDMovie):void
<<Java Class>>
SessionService
service
SessionService()
getAllSessionsByMovieId(int):List<Session>
getSessionBySessionId(int):Session
insertSession(Session):void
deleteSession(Session):void
updateSession(Session):void
<<Java Class>>
MovieService
service
MovieService()
getAllMovies():List<Movie>
getMovieById(int):Movie
insertMovie(Movie):void
deleteMovie(Movie):void
updateMovie(Movie):void
<<Java Class>>
CinemaManagerService
service
CinemaManagerService()
getAllCinemaManagers():List<CinemaManager>
findCinemaManagerById(int):CinemaManager
insertCinemaManager(CinemaManager):void
deleteCinemaManager(CinemaManager):void
updateCinemaManager(CinemaManager):void
<<Java Class>>
CinemaService
service
CinemaService()
getCinemasByMovieId(int):List<Cinema>
<<Java Class>>
CustomerService
service
CustomerService()
getAllCustomers():List<Customer>
findCustomerById(int):Customer
insertCustomer(Customer):void
deleteCustomer(Customer):void
updateCustomer(Customer):void
<<Java Class>>
IdentityMap<E>
utils
map: Map<Long,E>
IdentityMap()
getInstance(E):IdentityMap<E>
put(long,E):void
get(long)
-singletons
0..*
~threeDMovieMapper 0..1
~sessionMapper 0..1
~movieMapper 0..1
-cinemaManagerMapper 0..1
-customerMapper 0..1
<<Java Class>>
ThreeDMovieMapper
dataMapper
ThreeDMovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findThreeDMovieById(int):ThreeDMovie
<<Java Class>>
SessionMapper
dataMapper
SessionMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllSessionsByMovieId(int):List<Session>
findSessionById(int):Session
<<Java Class>>
CinemaMapper
dataMapper
CinemaMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllCinemas():List<Cinema>
<<Java Class>>
MovieMapper
dataMapper
MovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllMovies():List<Movie>
findMovieById(int):Movie
findCinemaByMovieId(int):List<Cinema>
<<Java Class>>
CinemaManagerMapper
dataMapper
CinemaManagerMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllcinemaManagers():List<CinemaManager>
findcinemaManagerById(int):CinemaManager
<<Java Class>>
CinemaMovieMapper
dataMapper
CinemaMovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findCinemaMoviesByMovieId(int):List<CinemaMovie>
<<Java Class>>
CustomerMapper
dataMapper
CustomerMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllCustomers():List<Customer>
findCustomerById(int):Customer
<<Java Class>>
DataMapper
dataMapper
DataMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
getDataMapper(DomainObject):DataMapper
DB

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:

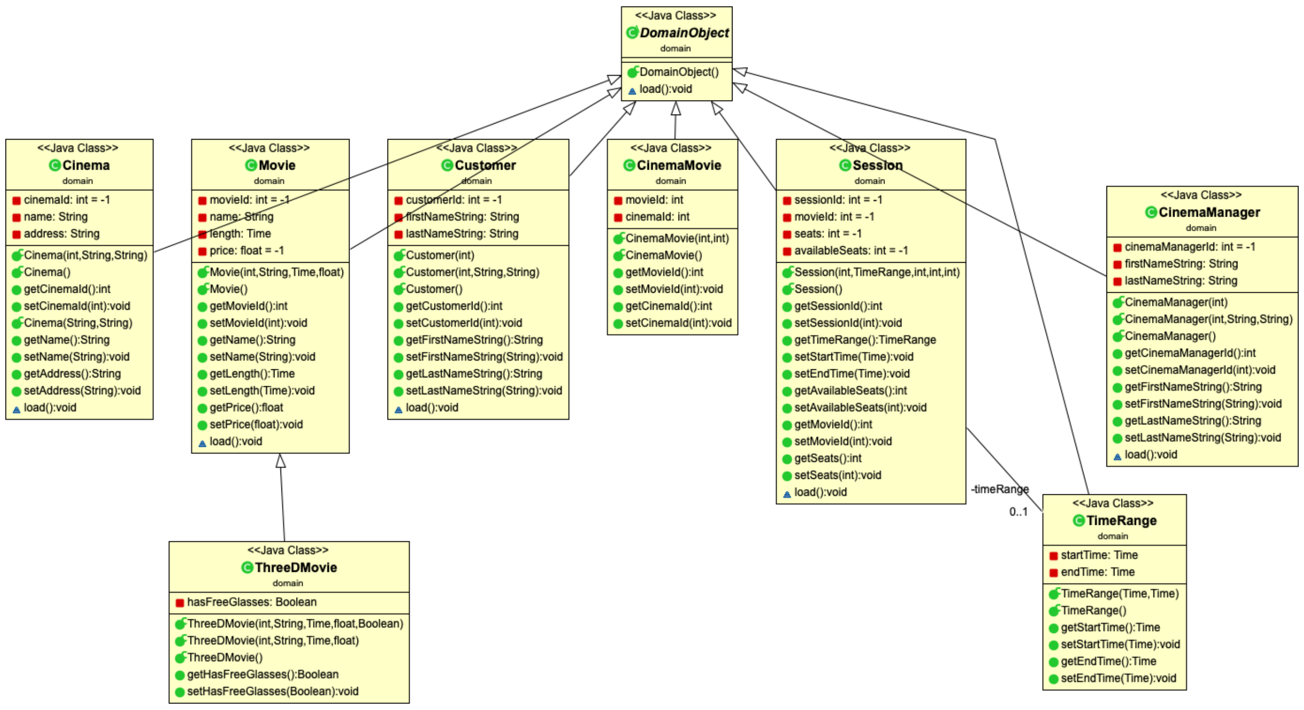
- Every time a view type of service is called, the service class will firstly check if the required object is already existed in identity.
- If so, it will load from the object from identity map and return it directly.
- If not, it will call related Data Mapper class and function, the function will load 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

Classes	Responsibility
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

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.

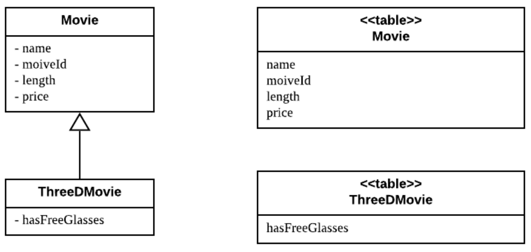
7.4.3 Association Table Mapping
This pattern is used to save the many-to-many relationship between database tables. It is implemented in out system as well. For example, the relationship between Cinema table and Movie table in database is many-to-many, as a result, an association table called Cinema_Movie 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:



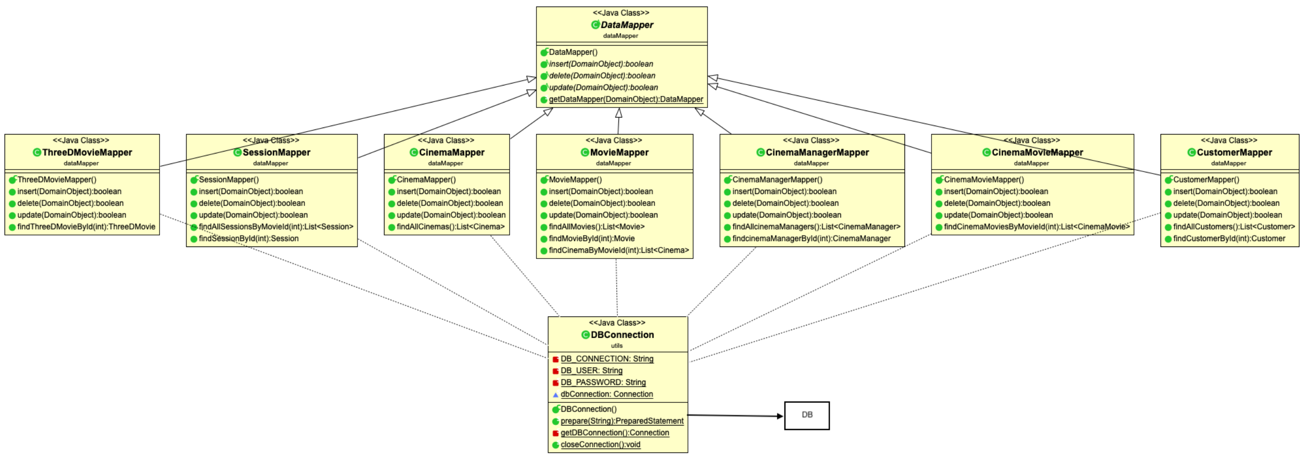
The reason why we use this inheritance is that it's easier to implement than concrete table inheritance and the table design is clear and without any wasted space (which is one advantaged compared with single table inheritance).

8. Data Mapper Layer (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:


<<Java Class>>
DataMapper
dataMapper
DataMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
getDataMapper(DomainObject):DataMapper
<<Java Class>>
ThreeDMovieMapper
dataMapper
ThreeDMovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findThreeDMovieById(int):ThreeDMovie
<<Java Class>>
SessionMapper
dataMapper
SessionMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllSessionsByMovieId(int):List<Session>
findSessionById(int):Session
<<Java Class>>
CinemaMapper
dataMapper
CinemaMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllCinemas():List<Cinema>
<<Java Class>>
MovieMapper
dataMapper
MovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllMovies():List<Movie>
findMovieById(int):Movie
findCinemaByMovieId(int):List<Cinema>
<<Java Class>>
CinemaManagerMapper
dataMapper
CinemaManagerMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllcinemaManagers():List<CinemaManager>
findcinemaManagerById(int):CinemaManager
<<Java Class>>
CinemaMovieMapper
dataMapper
CinemaMovieMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findCinemaMoviesByMovieId(int):List<CinemaMovie>
<<Java Class>>
CustomerMapper
dataMapper
CustomerMapper()
insert(DomainObject):boolean
delete(DomainObject):boolean
update(DomainObject):boolean
findAllCustomers():List<Customer>
findCustomerById(int):Customer
<<Java Class>>
DBConnection
utils
DB_CONNECTION: String
DB_USER: String
DB_PASSWORD: String
dbConnection: Connection
DBConnection()
prepare(String):PreparedStatement
getDBConnection():Connection
closeConnection():void
DB

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.

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 make the Unit of Work could tag different objects at the same time and commit to database together.

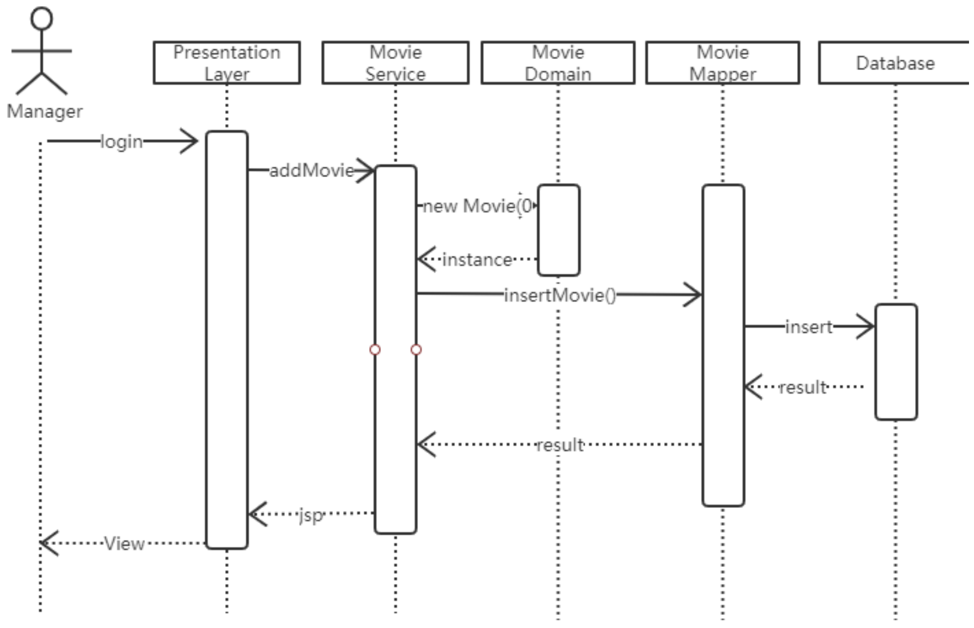
9.DataBase ER Diagram

10. System Overview

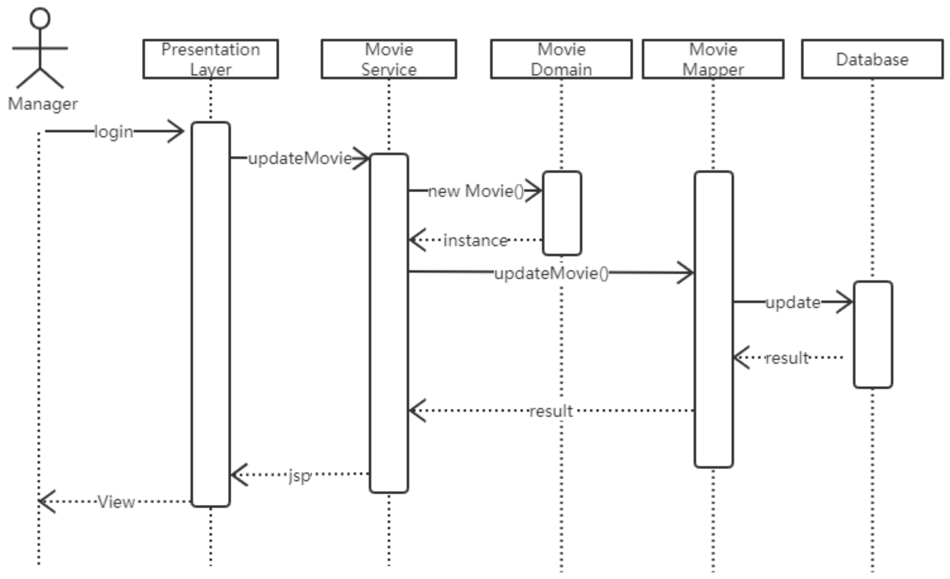
10.1 Interaction Diagrams

There are the interaction diagrams based on different user scenario

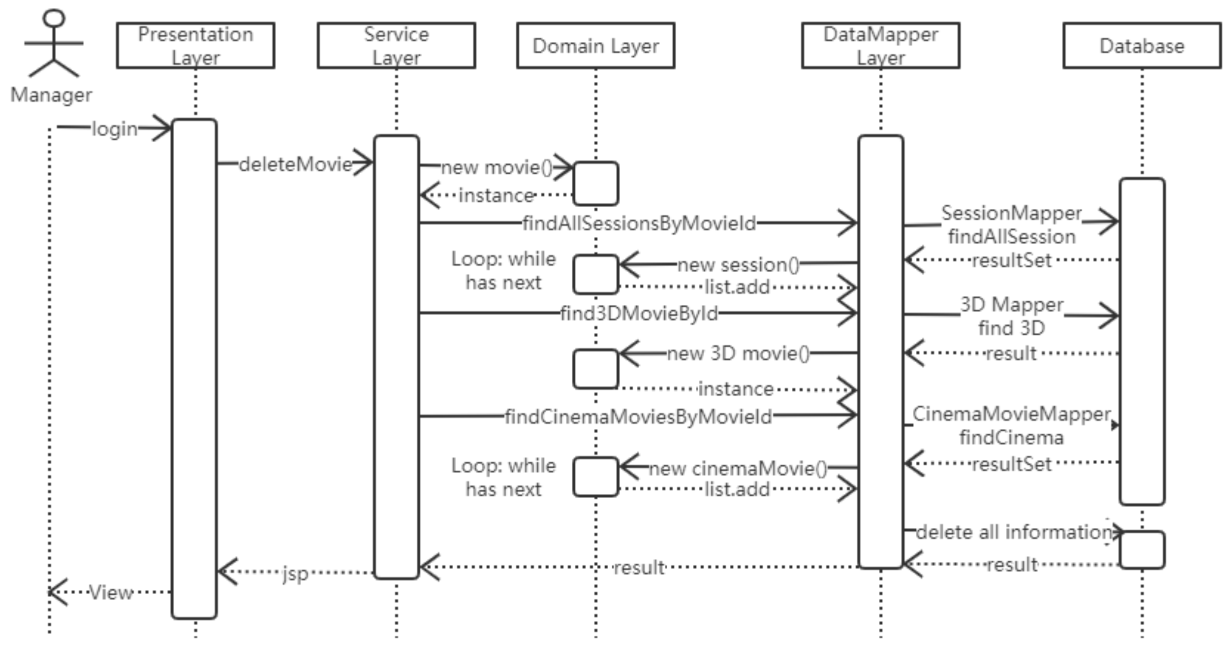
Scenario 1 Manager Add Movie



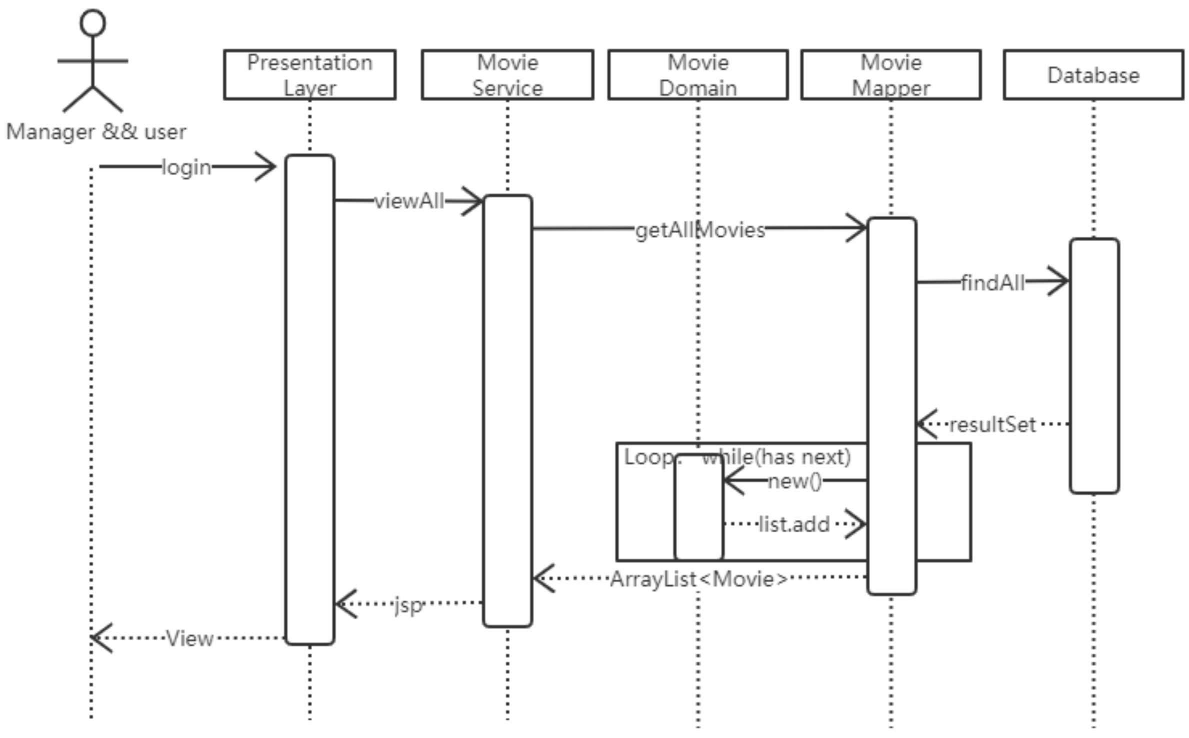
Scenario 2 Manager Edit Movie



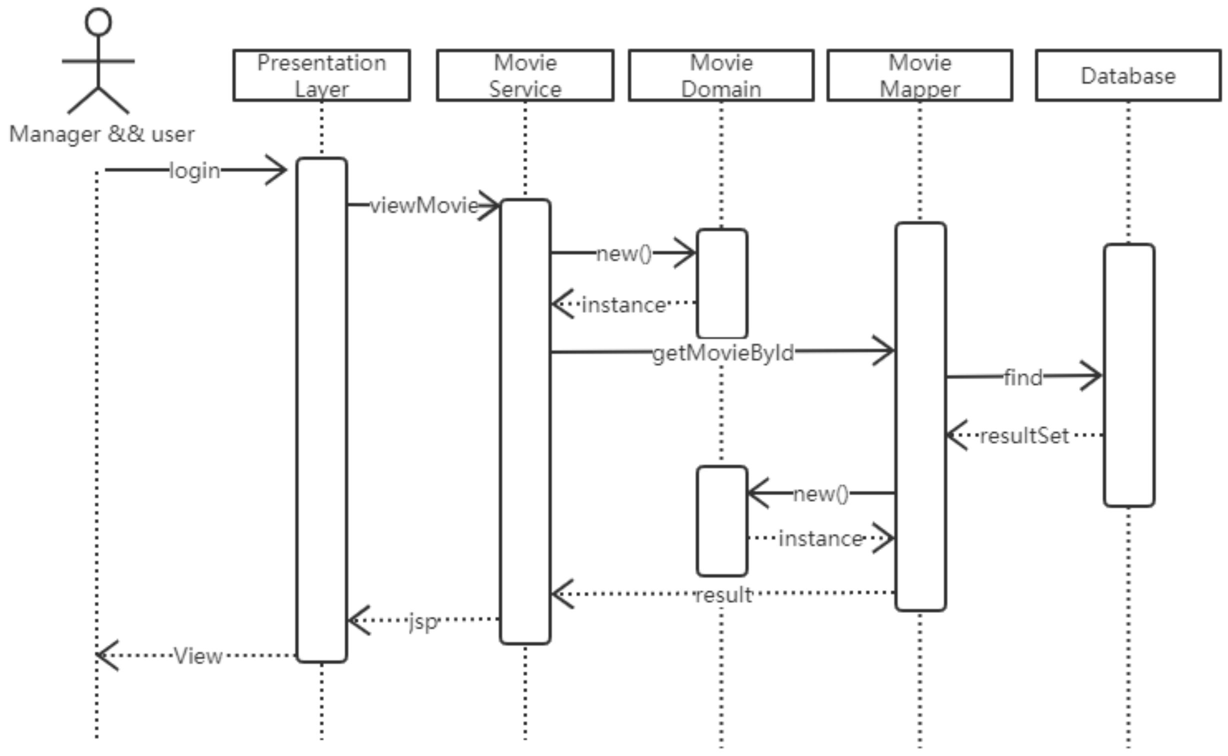
Scenario 3 Manager Delete Movie



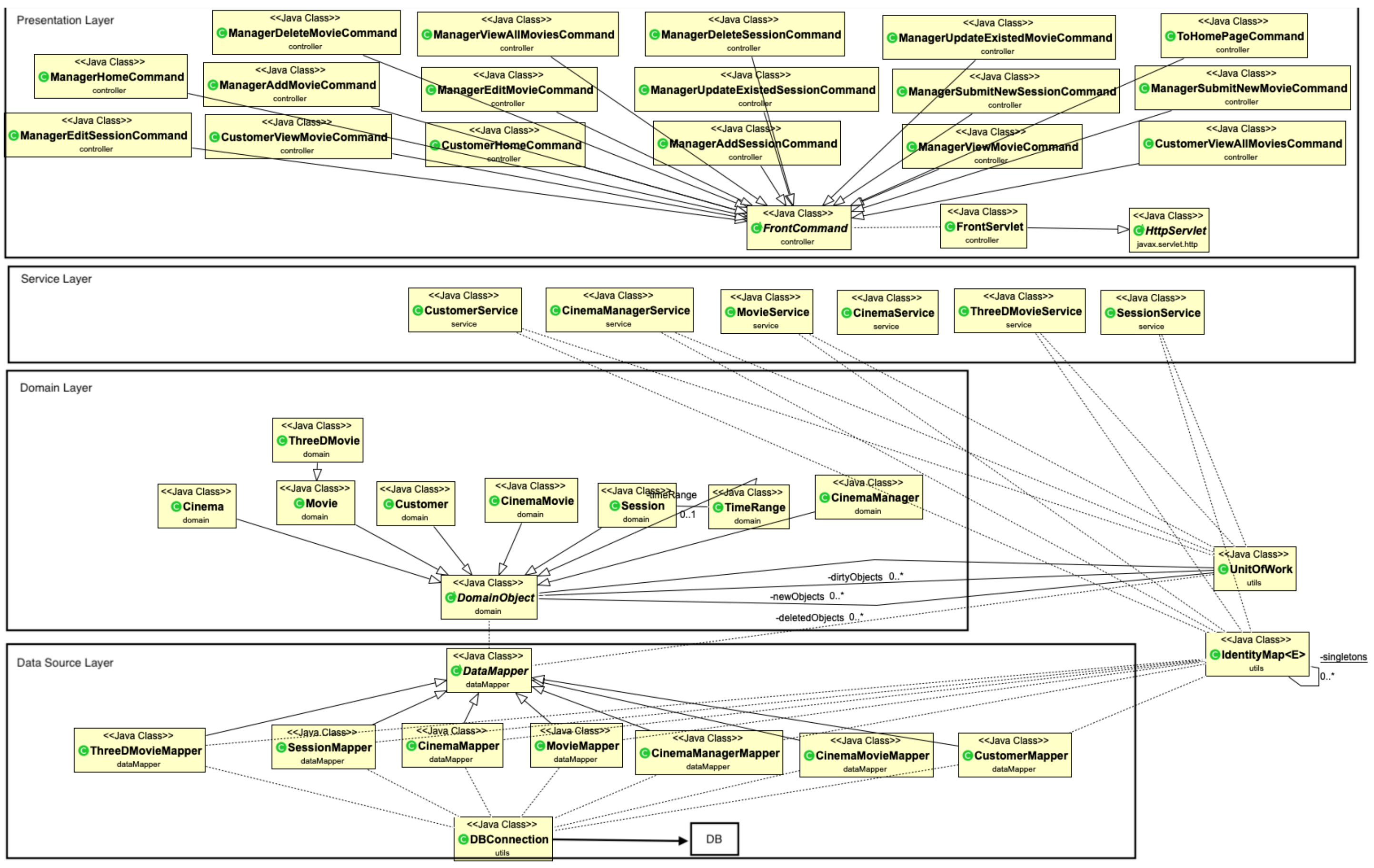
Scenario 4 Manager/Customer View All Movie



Scenario 5 Manager/Customer View One Specific Movie



10.2 Overall System Design

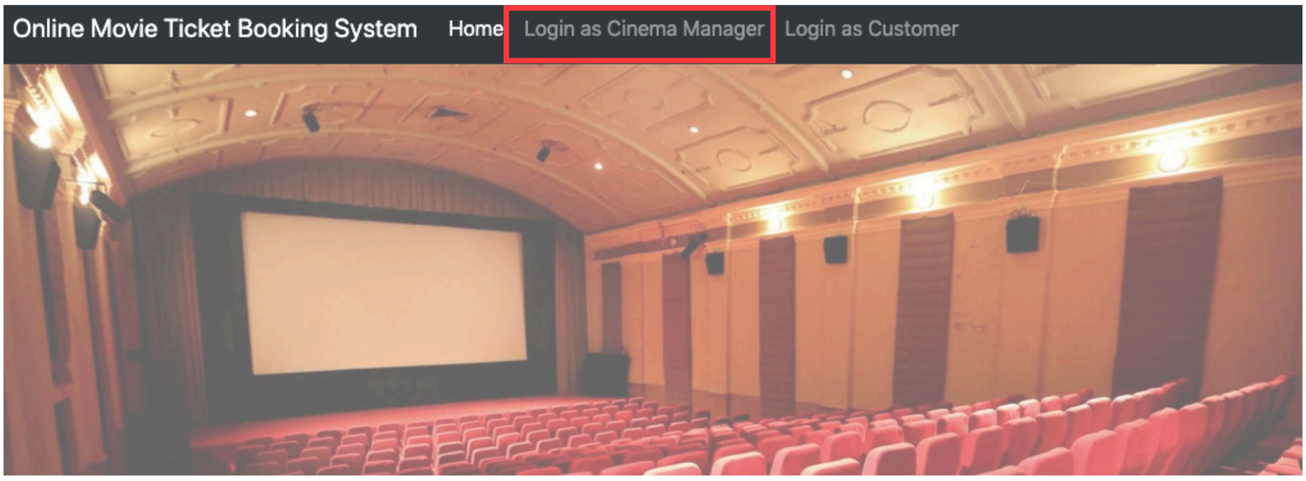

Presentation Layer
<<Java Class>> ManagerDeleteMovieCommand controller
<<Java Class>> ManagerViewAllMoviesCommand controller
<<Java Class>> ManagerDeleteSessionCommand controller
<<Java Class>> ManagerUpdateExistedMovieCommand controller
<<Java Class>> ToHomePageCommand controller
<<Java Class>> ManagerHomeCommand controller
<<Java Class>> ManagerAddMovieCommand controller
<<Java Class>> ManagerEditMovieCommand controller
<<Java Class>> ManagerUpdateExistedSessionCommand controller
<<Java Class>> ManagerSubmitNewSessionCommand controller
<<Java Class>> ManagerSubmitNewMovieCommand controller
<<Java Class>> ManagerEditSessionCommand controller
<<Java Class>> CustomerViewMovieCommand controller
<<Java Class>> CustomerHomeCommand controller
<<Java Class>> ManagerAddSessionCommand controller
<<Java Class>> ManagerViewMovieCommand controller
<<Java Class>> CustomerViewAllMoviesCommand controller
<<Java Class>> FrontCommand controller
<<Java Class>> FrontServlet controller
<<Java Class>> HttpServlet javax.servlet.http
Service Layer
<<Java Class>> CustomerService service
<<Java Class>> CinemaManagerService service
<<Java Class>> MovieService service
<<Java Class>> CinemaService service
<<Java Class>> ThreeDMovieService service
<<Java Class>> SessionService service
Domain Layer
<<Java Class>> ThreeDMovie domain
<<Java Class>> Cinema domain
<<Java Class>> Movie domain
<<Java Class>> Customer domain
<<Java Class>> CinemaMovie domain
<<Java Class>> Session domain
-timeRange
0..1
<<Java Class>> TimeRange domain
<<Java Class>> CinemaManager domain
<<Java Class>> DomainObject domain
-dirtyObjects 0..*
-newObjects 0..*
-deletedObjects 0..*
<<Java Class>> UnitOfWork utils
<<Java Class>> IdentityMap<E> utils
-singletons
0..*
Data Source Layer
<<Java Class>> DataMapper dataMapper
<<Java Class>> ThreeDMovieMapper dataMapper
<<Java Class>> SessionMapper dataMapper
<<Java Class>> CinemaMapper dataMapper
<<Java Class>> MovieMapper dataMapper
<<Java Class>> CinemaManagerMapper dataMapper
<<Java Class>> CinemaMovieMapper dataMapper
<<Java Class>> CustomerMapper dataMapper
<<Java Class>> DBConnection utils
DB

Appendix

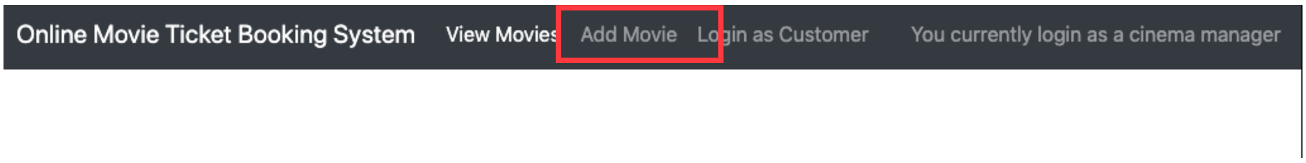
A. Deployment URL: https://online-movie-ticket-booking.herokuapp.com/
B. Scenario Testing Guide:
There are the steps you can follow to test all functions in feature A. Because of the time limitation, we just roughly implemented a front-end system for backend function testing and there may be some bugs existed in this frond-end, especially in some parameter input areas. We will replace this one in our next submission and make some design more reasonable.

Scenario 1: Manager Add Movie

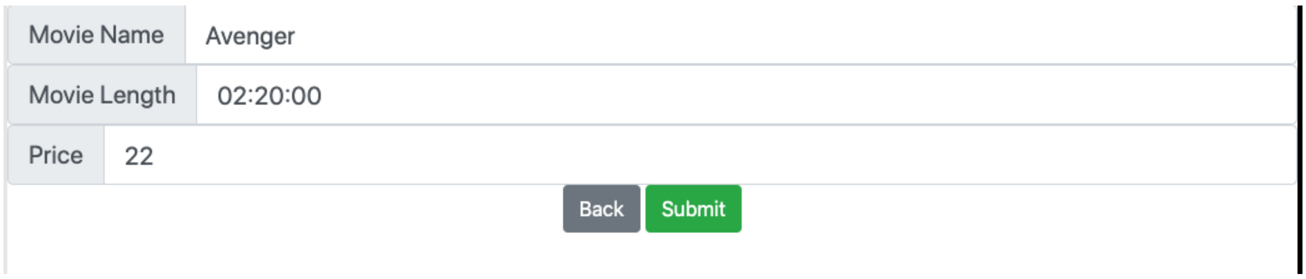
1. Visit the website by clicking this url: https://online-movie-ticket-booking.herokuapp.com/
2. Click on "Login as Cinema Manager" to operate as cinema manager



3. Click on "Add Movie" to add movie information


Online Movie Ticket Booking System
View Movies
Add Movie
Login as Customer
You currently login as a cinema manager

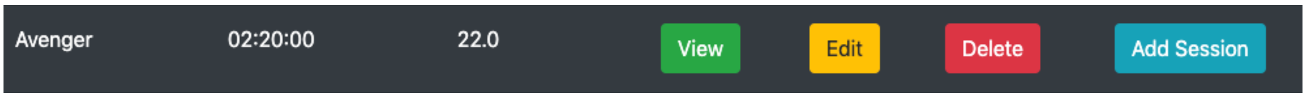
4. Input movie information, the data format is as below



Movie Name	Avenger
Movie Length	02:20:00
Price	22

5. Click on "Submit" button

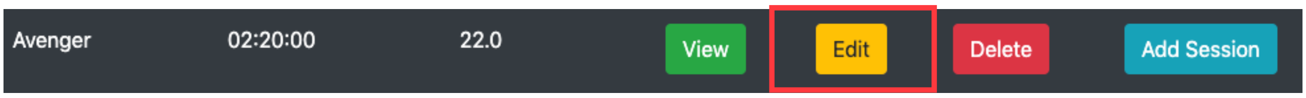
6. Click on “View Movies” button to view the movie just added. You could see the movie has already been added



Avenger	02:20:00	22.0	View	Edit	Delete	Add Session

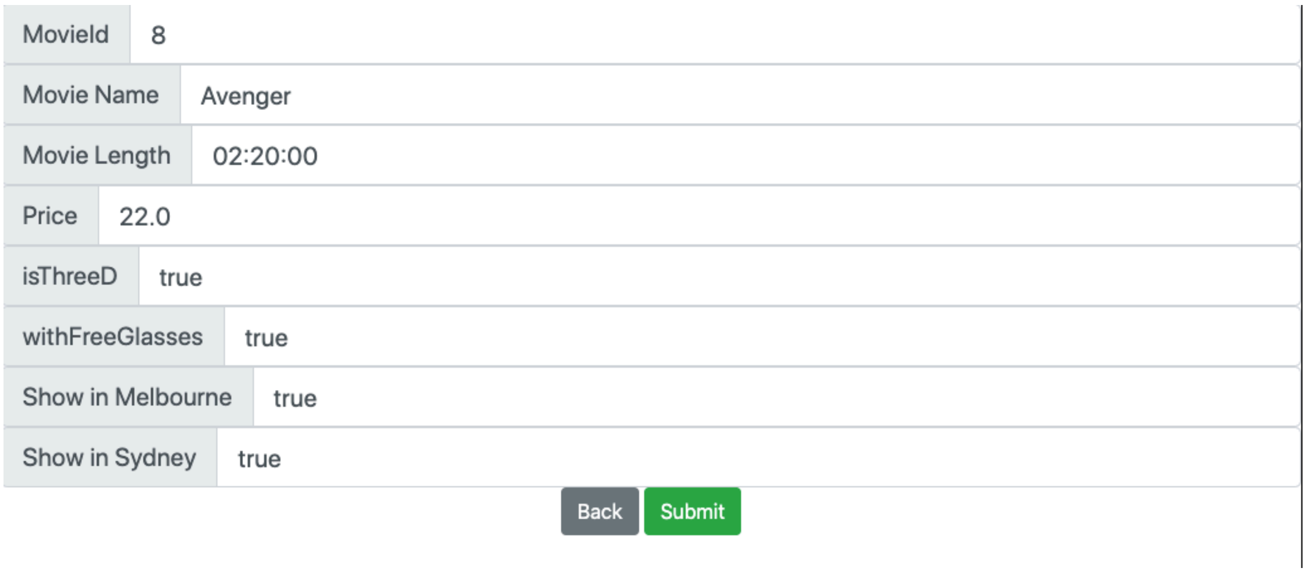
Scenario 2: Manager Edit Movie

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



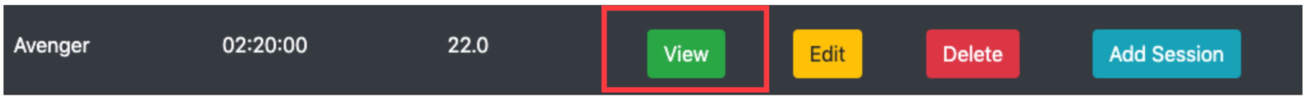
Avenger	02:20:00	22.0	View	Edit	Delete	Add Session

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)



MovieId	8
Movie Name	Avenger
Movie Length	02:20:00
Price	22.0
isThreeD	true
withFreeGlasses	true
Show in Melbourne	true
Show in Sydney	true

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.



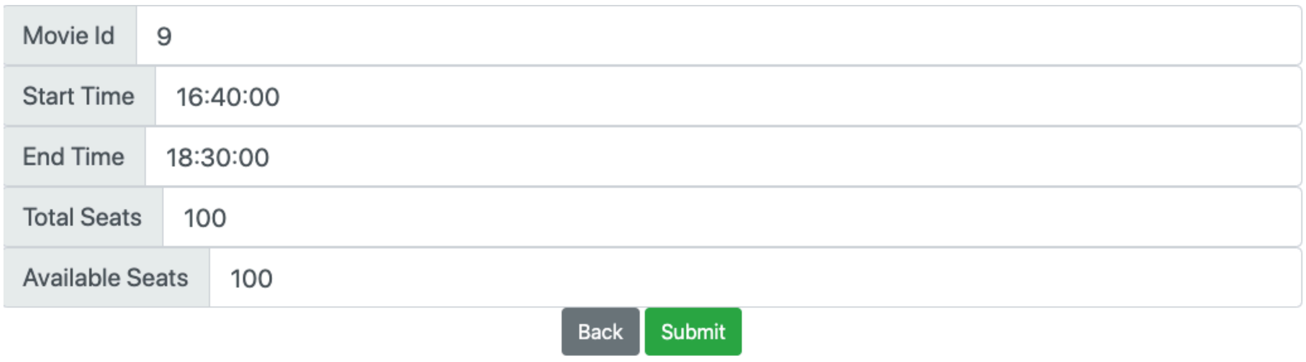
Avenger	02:20:00	22.0	View	Edit	Delete	Add Session

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

Sub-scenario : add session

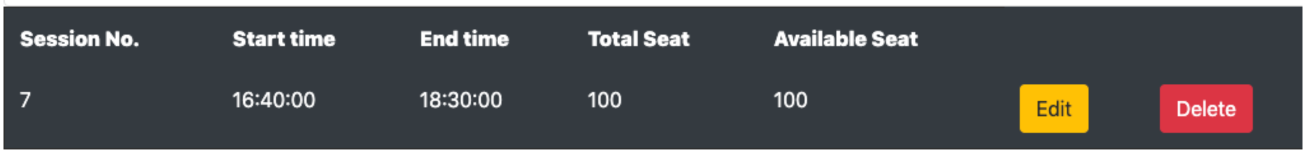
1. Back to the movie list page
2. Click on the “add session button” behind any movie (Please remember which one you select)

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)



Movie Id	9
Start Time	16:40:00
End Time	18:30:00
Total Seats	100
Available Seats	100

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



Session No.	Start time	End time	Total Seat	Available Seat		
7	16:40:00	18:30:00	100	100	Edit	Delete

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

Scenario 3: Manager Delete Movie

1. Back to the Movie List page
2. Click on the "delete" behind any movie (Please remember which one you select)
3. Click on the "View Movie" button in cinema manager home page
4. Now you can see the movie has been deleted

Scenario 4: 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 5: Customer View Movie

1. Visit the website by clicking this url: https://online-movie-ticket-booking.herokuapp.com/
2. This time, click on the "Login as Customer" button
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.
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.