Cinema Online Ticketing System

Part II

Sirui Yan (siruiy) 792320
Jingyi Li (jingyil14) 812509

DOMAIN CLASS DIAGRAM

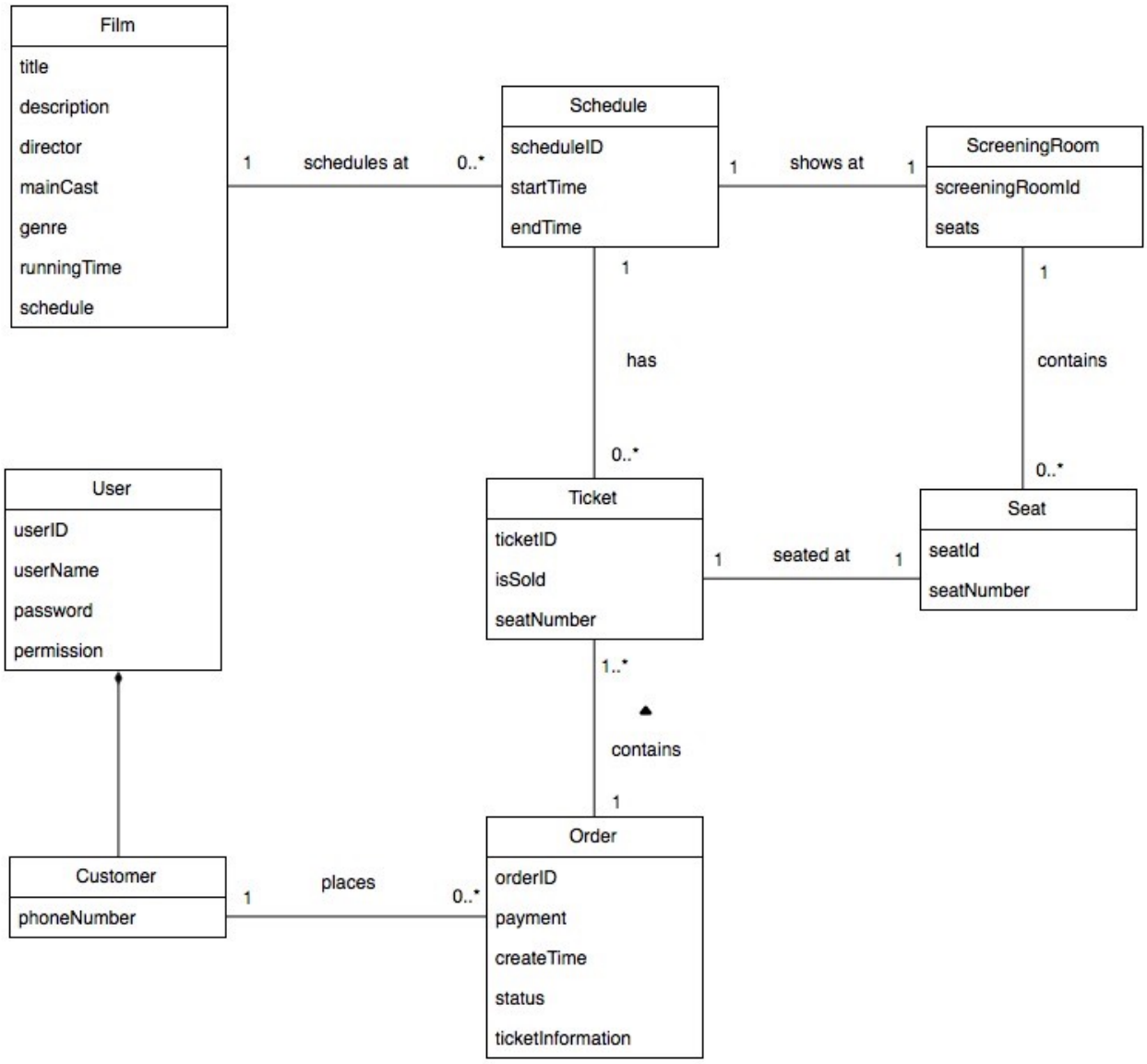

Figure 1. domain class diagram

The domain class diagram is as shown in Figure 1. It consists of the following 8 class:

- **Film:** contains the information of a film (title, description, director, main cast, genre, running time, and schedules).
- **Schedule:** contains the schedule information, such as start time and end time, of a film.
- **ScreeningRoom:** contains the information of a screening room : its ID and the seats in this room.
- **Seat:** contains the information of a seat, such as it seats ID and its seat number.

- **Ticket:** contains the information of a ticket (ticket ID, whether it is sold, and the seat number).
- **Order:** contains the information of a order such as order ID, the information of the tickets in the order, payment etc.
- **User:** contains the information of a user, including user ID, username, password, and the permission (administrator or customer). User class is the generalisable version its child class Customer.
- **Customer:** contains the information of a customer, including phone number and all orders of the customer.

HIGH LEVEL ARCHITECTURE

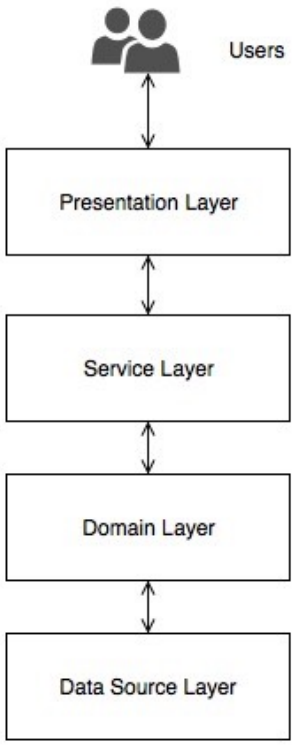

Figure 2. high level architecture

The system uses a four-tier layered architecture as shown in Figure 2. There are four layers in the high level architecture: presentation layer, service layer, domain layer and data source layer from top to bottom. Each layer in the architecture only depends on the layers below it and has no information about the layers above it. The layering architecture is more reusable

and easier to maintain than unlayered architecture because each layer is encapsulated and can work as the interface provided to other layers.
The data source layer connects the system and source of data such as databases. The domain layer defines the logic of the system. In the service layer, we implements user interfaces for the representation layer. The representation layer is the layer that interacts with users and communicates to the system. We will discuss each layer in details in detailed architecture section.

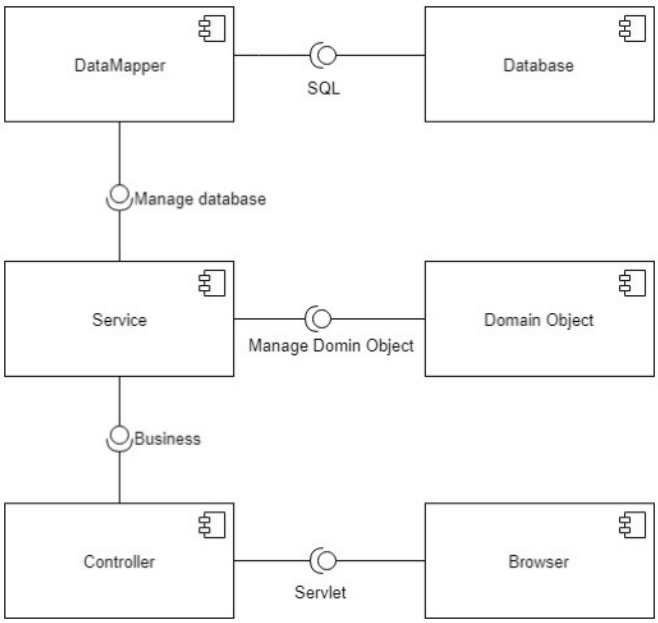

Figure 3. component diagram

Figure 3 shows the component diagram of the system. Datamapper component manipulates database through SQL statement and provides an interface for Service component to let it manage the CRUD operations in the database. Service component converts results from database to domain object by the interface of Domain Object component. Controller component calls functions in Service component through its interface to deal with the business logic. Browser, which is the user interface, gets information from Controller component through Servlet.

DETAILED ARCHITECTURE

Data Source Layer

Implemented Patterns: Data mapper, Identity map, Unit of Work

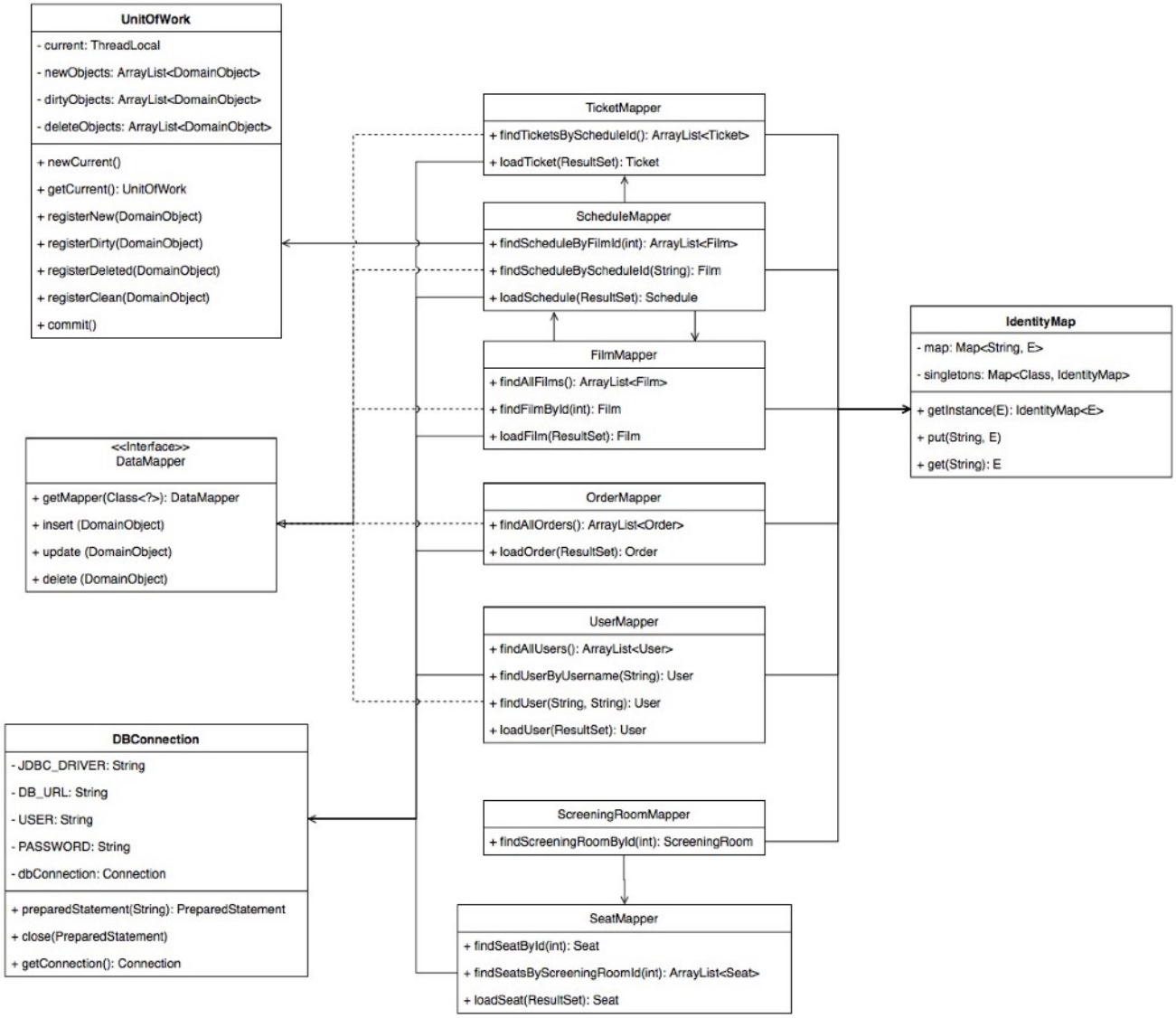

Figure 4. data source layer class diagram

- Data Mapper

The class of data source layer is shown in Figure 3. Data mapper pattern is used as the architectural design in the data source layer. For maintainability and reusability, we used domain model pattern in our system. Thus, we chose Data mapper pattern for data source layer because it has high compatibility with domain model pattern.

The seven data mappers in the data source layer connects the database with the system: they can read data from the database and make CRUD operations to the database according to system instructions. TicketMapper, ScheduleMapper, FilmMapper,

OrderMapper and UserMapper are implemented from DataMapper interface since they all have insert, update and delete action to the database. ScreeningRoomMapper and SeatMapper do not implement the interface because the screening room and seat information are immutable, these two mappers are only responsible for reading the data from the database.

- Identity Map

Identity map pattern is applied to make sure every domain object is read from database only once. For domain objects whose values can be modified by user, such as Film and Schedule, identity map pattern guarantees their consistency in the system; for immutable domain objects such as ScreeningRoom, identity map pattern can reduce some unnecessary costs like reading the data from the database multiple times.

- Unit Of Work

We used unit of work pattern to process the edit of film schedules because the admin can make multiple create, update and delete operations to the schedule table in the database in one edit, using unit of work pattern would save some cost of multiple connection to the database.

Domain Layer

Implemented Patterns: Lazy load (Lazy initialisation), Identity field, foreign key mapping, association table mapping, embedded value, single table inheritance

- Lazy Load

Figure 5 shows the class diagram of domain layer. The lazy load pattern is applied when the information of movie is loaded. We use lazy initialisation way to implement this pattern: when a Film object is initialised, only the basic information (film ID, title, description, director, main cast, genre, running time) of a film is loaded, the schedules of a film is only loaded when the getSchedule() method is called. This pattern is chosen because admin home page only need the basic information of films, the schedule information is only required when the user view the schedule information or edit the schedule of a film. Loading all the information of a film when it is initialised would be cost inefficient, it would take take extra cost to check the information of schedule in another table and store the schedule data in the memory. Thus,

we choose lazy initialisation pattern for to load film information. The value of schedule can be null, so we use scheduleLoaded to identify if the schedule information is loaded.

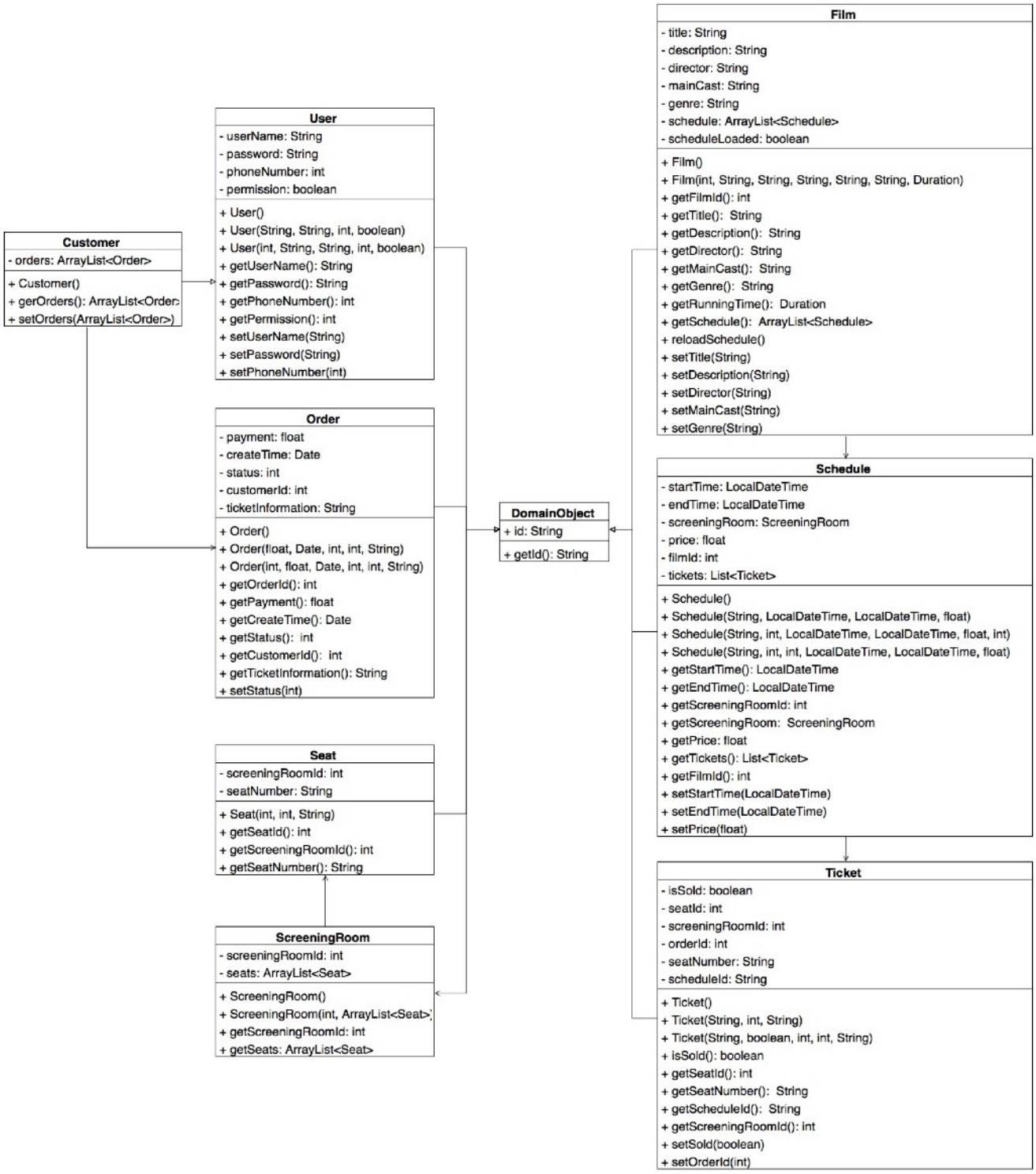

Figure 5. domain layer class diagram

- Identity Field

Each object in the domain correspond to an unique row in the database, so we need to identify which is the corresponding row of an domain object when we read or write the information from or to the database. Therefore, we adopted identify field pattern for domain layer and store the primary key in each domain object.

- Foreign Key Mapping

Some objects have one-to-many association with other objects, such as one customer can have multiple orders. Therefore, many tables in our database has foreign key, take the order for example, order table has customer id, which is the primary key in users table, as foreign key to store the user id that this order belongs to. As a result, in the design of order object, we stored customer id in order object, which applying foreign key mapping in this object. Other objects, which also have foreign keys, are implemented the foreign key mapping pattern as well.

- Association table mapping

In our system, there is no many-to-many association between different objects. Therefore, we only used foreign key mapping to process the one-to-many association.

- Embedded Value

In the seat table in the database, the screening room id is stored for every seat. However, the system always requires the information of all the seat in a screening room. For example, when the administrator creates or deletes a schedule, the system needs to create or delete tickets for all the seats in the screening room of the schedule. It would take unnecessary cost if we read all the seats in a screening room from database every time, so we use embedded value pattern to implement the screening room as a domain object and stores all the seats in this room in the screening room object.

- Single Table Inheritance

For users in the system, there are two kinds of users, which are customers and admin. They all have username, password and phone number, while only customer has orders. Therefore, in the design of these classes, customer class inherits user class. When mapping the inheritance into tables in database, we chose single table inheritance, that is putting admin and customers in one table and add a permission column to specify this user is an

admin or not. This inheritance pattern is using because there is only one admin in our system, so creating a new table for admin wastes space and makes the database design meaninglessly complicated. Keeping all users in one table also avoids join operation which lets our system more efficient.

Service Layer

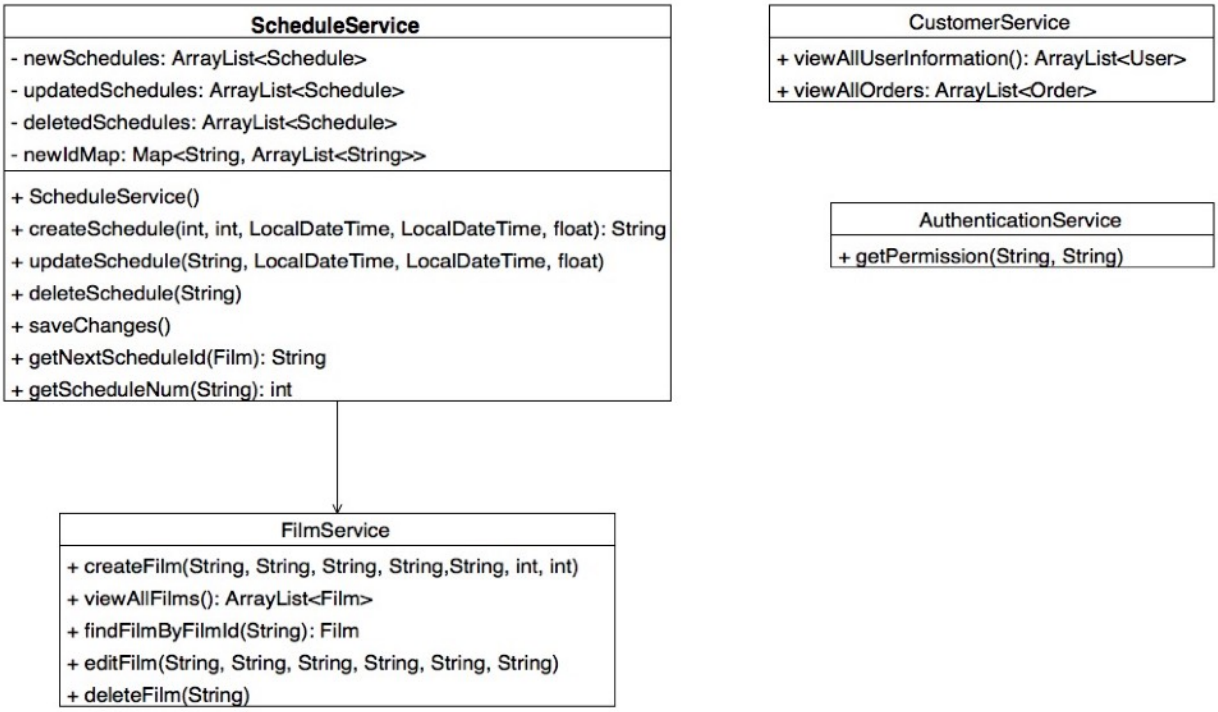

Figure 6. service layer class diagram

AuthenticationService is responsible for checking if the input username and password is valid, and what is the permission of the user if it is valid. CustomerService is responsible for handling the request to view all the users and orders in the database. FilmService is responsible for the operations about films, such as creating a film, view all the films, edit the basic information of a film, delete a film and get the information of a film. ScheduleService is responsible for handling creating, updating, and deleting a schedule.

Presentation Layer

Implemented Patterns: Page Controller, Transform View

Page Controller is chosen as the pattern for input controller in our web app, that is each page has its own controller to process the requests from users. For example, LogInControllerServlet for log in page, AddFilmControllerServlet for add film page and so on. The user request in our web app is quite straightforward. Each page usually just has one or two requests, such as only one post request in log in page and only one get request in view order page. Therefore, a servlet is suitable for this situation, with the benefit of ease of understanding and convenience of development.

The pattern used for handling the view in our project is Transform View. In our website, controller of each page converts between the object from service layer and JSON. Then, the page uses javascript to render the JSON data from the backend. The reason of choosing this pattern is that the HTML of our pages is complicated, so generating HTML in servlet will make the code messy. What's more, this pattern encapsulates the backend to deal with business logic and the view can focus on displaying data.

INTERACTION DIAGRAM

Figure 7, Figure 8, Figure 9, Figure 10, Figure 11, Figure 12, Figure 13, Figure 14, Figure 15, Figure 16 and Figure 17 shows the sequence digram for log in, view all films, view all the information of a film, add a film, edit film basic information (title, description, director, main cast, genre and running time), add a schedule for a film, edit a schedule, delete a schedule,

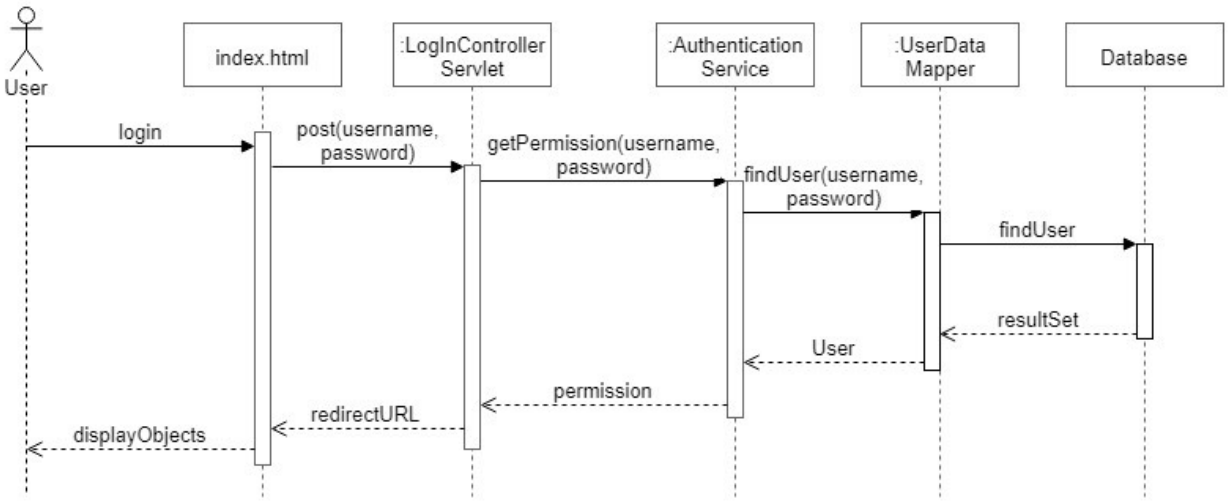

Figure 7. log in sequence diagram

delete a film, view all orders and view all user information respectively.

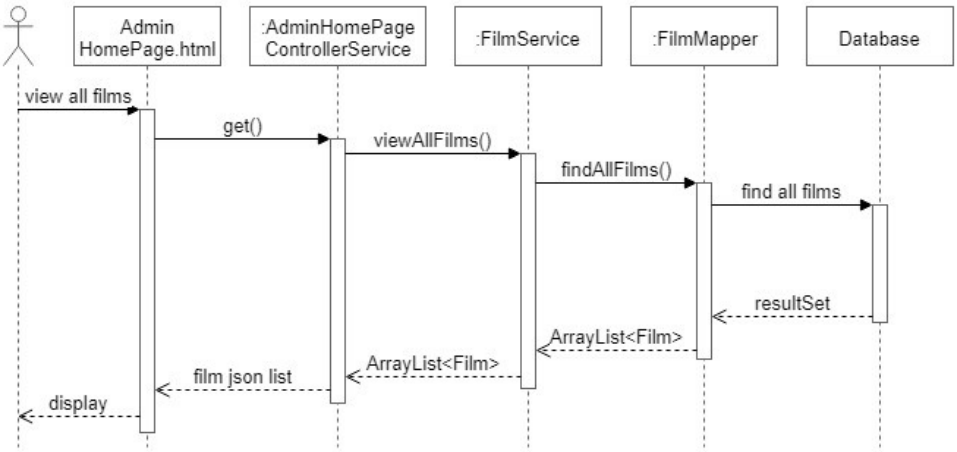

Figure 8. view all films sequence diagram

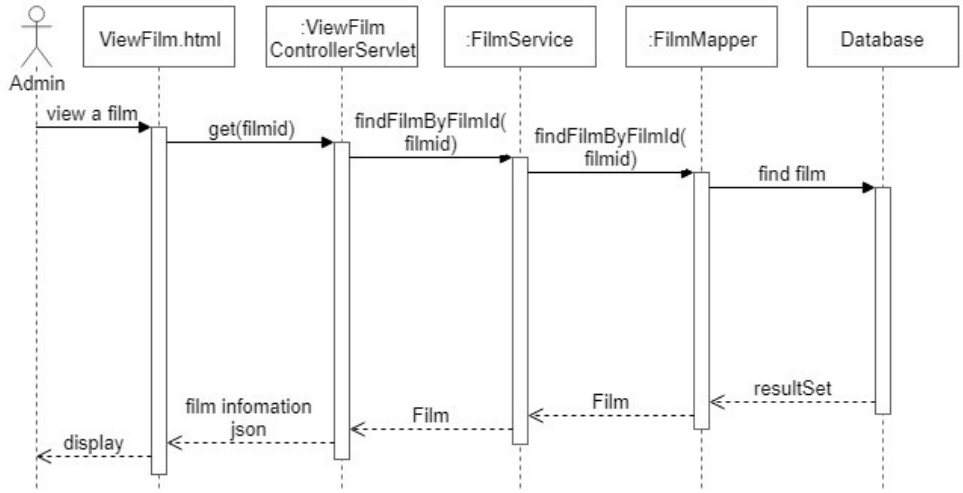

Figure 9. view film information sequence diagram

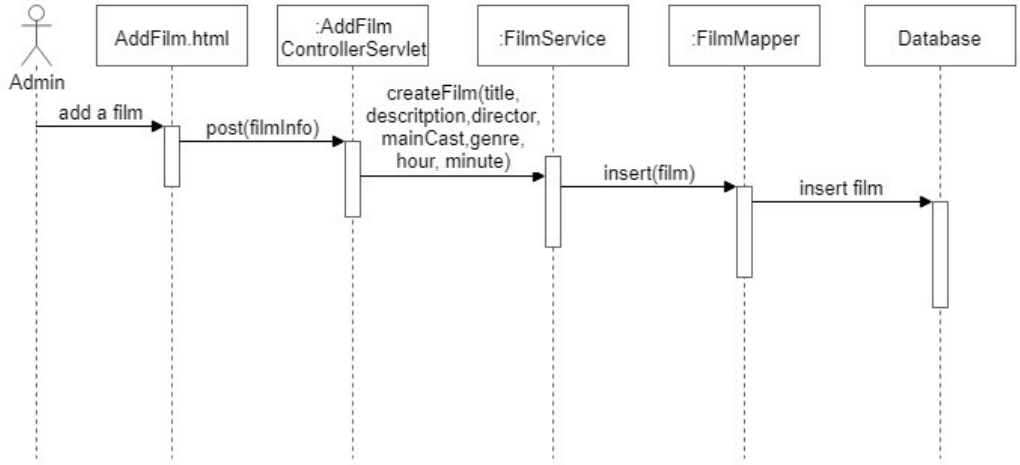

Figure 10. add a film sequence diagram

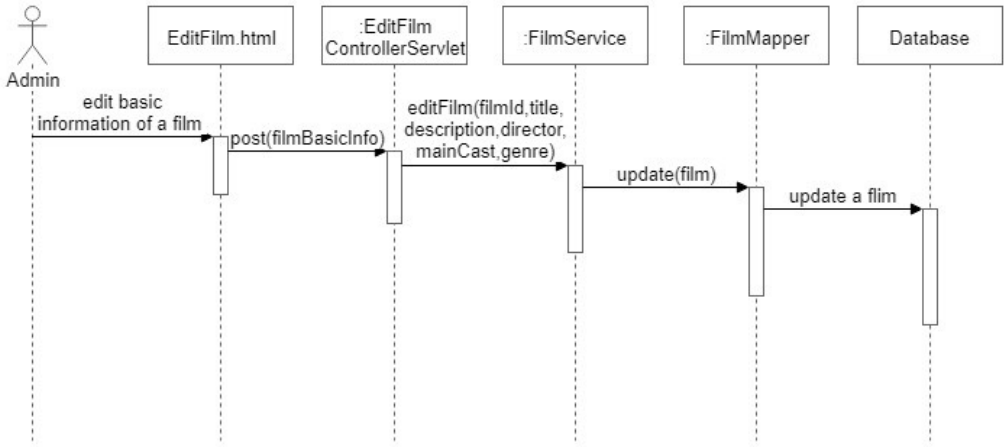

Figure 11. edit film basic information sequence diagram

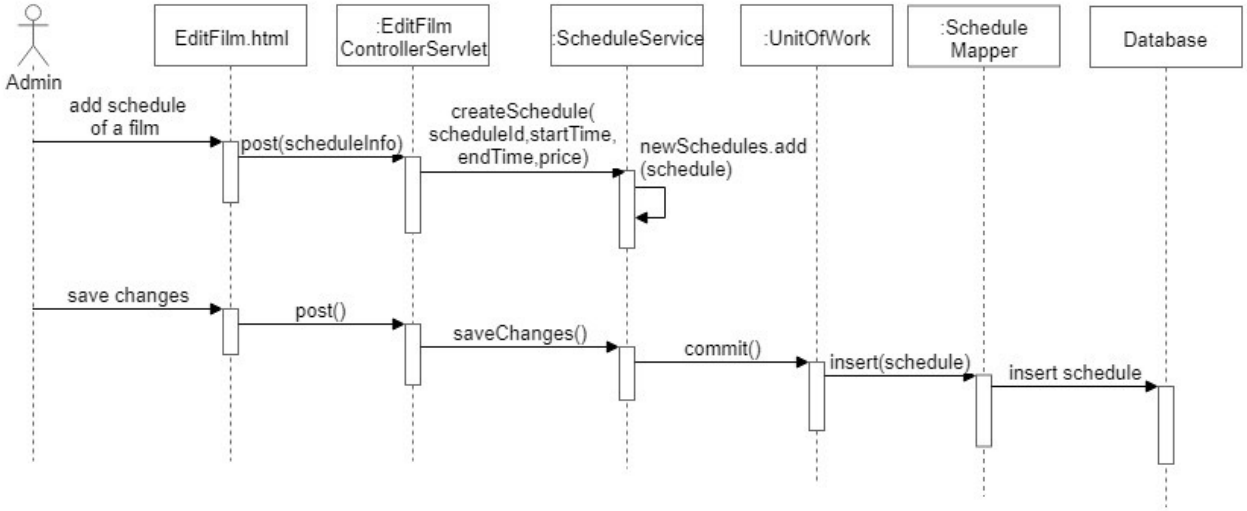

Figure 12. add a schedule sequence diagram

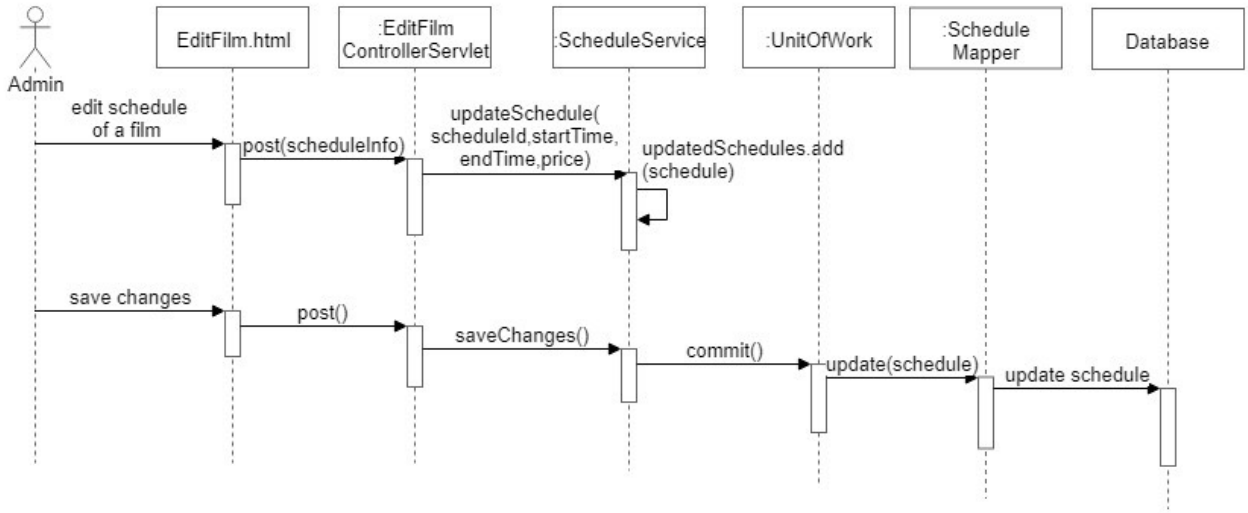

Figure 13. edit a schedule sequence diagram

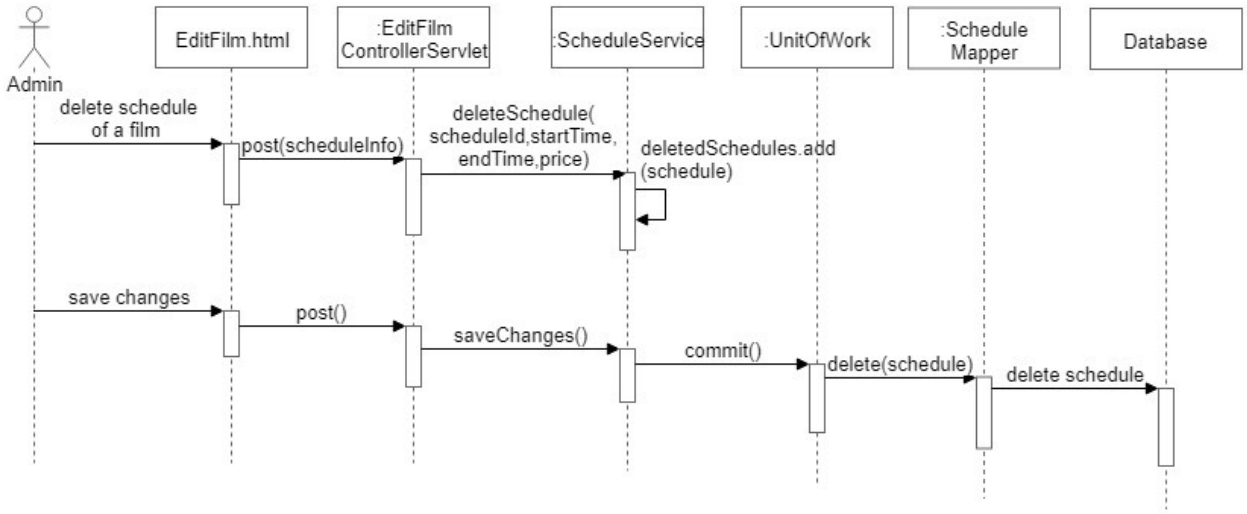

Figure 14. delete a schedule sequence diagram

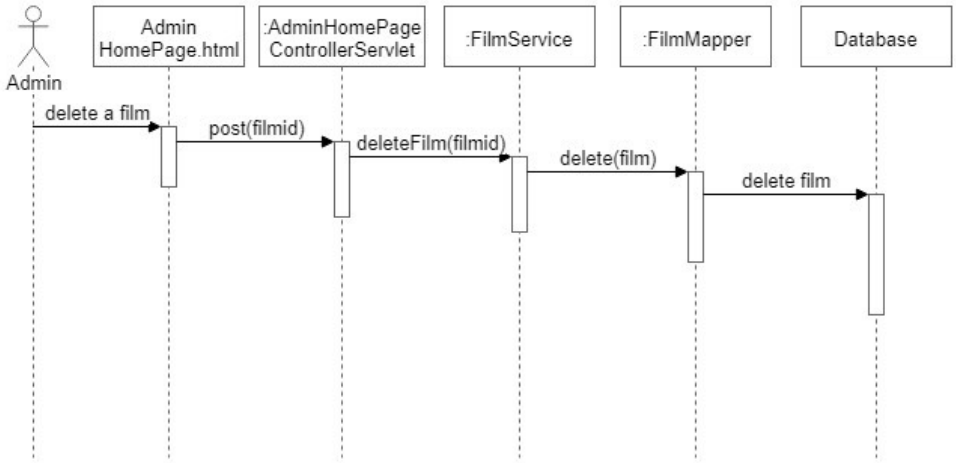

Figure 15. delete a film sequence diagram

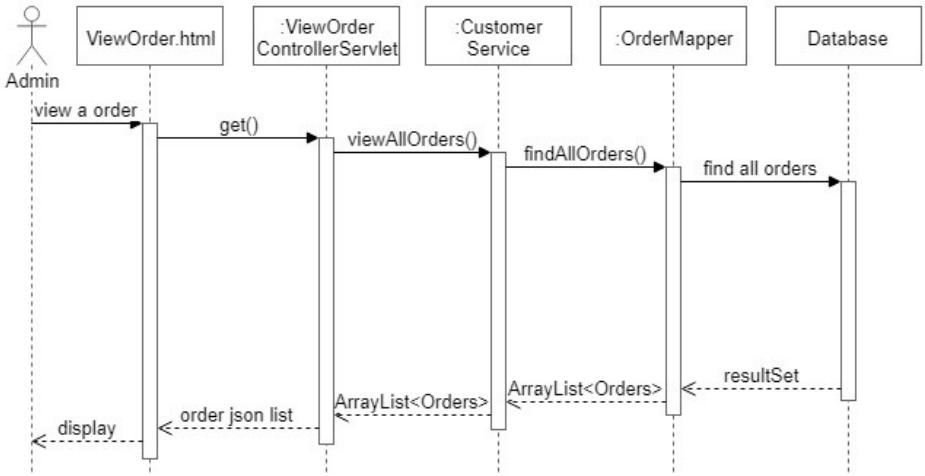

Figure 16. view all orders sequence diagram

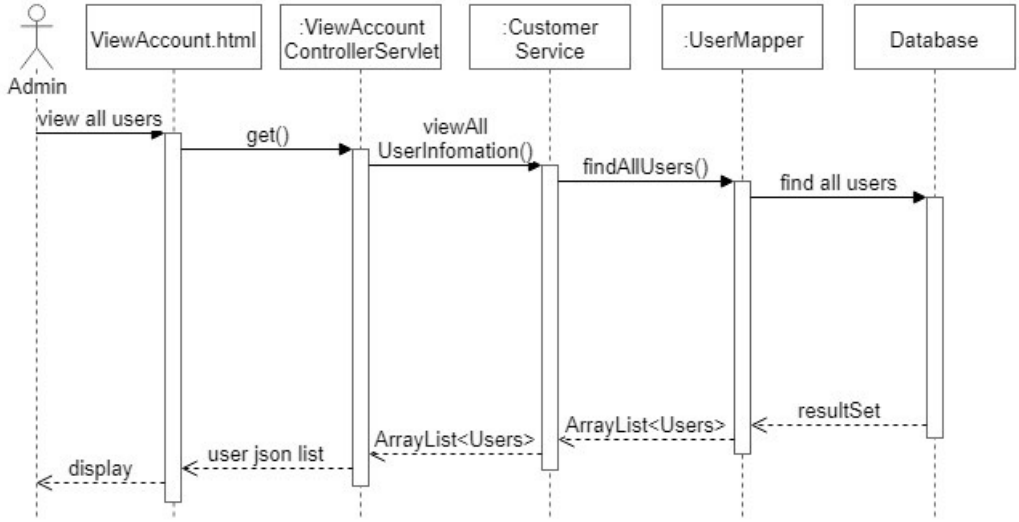

Figure 17. view all users sequence diagram

USER GUIDE

bitbucket repository link: https://bitbucket.org/jingyil14/cinema-online-ticketing-system.git
(See Feature A Branch)

heroku link: http://online-ticketing.herokuapp.com

Scenario 1: Log In

The user for feature A is administrator, the username and password is as follows:

username: admin

password: admin

Creating a new account feature is not available

Scenario 2: View All Films

See the home page or click movies tab on the navigation bar

Scenario 3: Add a Film

Click add button on the right top corner

Please input information in every text input

The uploaded poster will not be the real poster stored in the database

Scenario 4: View the information of a film

Click view button on the film you are interested in to view all the information of a film

Click schedule in the view film information page to see the schedules of the film

Scenario 5: Edit a film

Click edit button on the film you are interested in to edit the information of a film

Click schedule in the edit film information page to edit the schedules of the film

Don't forget to click submit button on the information page and save button on schedule page if you want to save all the changes you have made.

Click submit will only save the changes to the basic information, click save on schedule page only save the changes made to schedules

Running time is not editable

Input an integer when you edit or create a schedule

Scenario 6: Delete a Film

Click edit button on the film you want to delete

Scenario 7: View All Orders

Click orders tab on the navigation bar

Scenario 8: View All Users

Click accounts tab on the navigation bar