澳門理工大學
Universidade Politécnica de Macau
Macao Polytechnic University

Faculty of Applied Sciences
Bachelor of Science in Computing

COMP413 ENTERPRISE SYSTEM AND APPLICATION DEVELOPMENT

Academic Year 2023/24

Movie Mental: A JavaEE Web Application

Student ID:	P2010459
Student Name:	Waley Lin

Table of Contents

1 Introduction

In today's digital era, the movie rental industry is undergoing a dramatic transformation, and the demand for online rental services has increased dramatically. This project aims to develop a JavaEE-based web application to adapt to this market shift and provide a comprehensive movie rental solution.

The app, called "Movie Mental", is designed to create an easy-to-manage and browse movie catalog, with each movie detailed with key information such as title, year of release, genre, starring role, production studio, director, film length, rental price, production cost and estimated box office revenue, etc. In addition, the project adopts the Model-View-Controller (MVC) architecture pattern, which ensures a high degree of modularity and maintainability of the application through the separation of logic, interface and input.

"Movie Mental" not only provides movie lovers with an intuitive platform to search and browse movies that interest them, but also allows users to register an account, log in, add movies to shopping cart, and view order history. The functionality of the website doesn't stop there, administrators can easily add new movies, update movie information, or delete movies.

This project report provides a detailed account of the overall technical approach and key technical design decisions during development, covering the technologies and frameworks employed by the application, their purpose and the rationale for their selection. The report also includes ER database diagrams, site maps, user interface screenshots, and class diagrams following UML specifications, which together outline the design concept, technical implementation, and functional features of the application in the modern movie rental field.

2 Design and Planning

This part focuses on the design philosophy of the entire system, the selected technology stack, and the decision-making basis for the framework. It describes why a certain technology or framework was chosen and how they meet the project needs.

2.1 General technical approach

This project uses the Java EE platform, especially its Web application development part, to build a movie rental system. The project's model layer utilizes the Java Persistence API (JPA) to operate on the database, ensuring standardization and simplification of data persistence. The controller layer uses Servlet to process user requests and dispatch them to the corresponding views. The view layer uses Java Server Pages (JSP) technology to dynamically generate HTML content and provide a good user interface and user experience. The design of the project follows the Model-View-Controller (MVC) design pattern, ensuring clear responsibilities and loose coupling at each level. In addition, the project is deployed on GlassFish server, a powerful application server that provides full support for the Java EE specification while also ensuring high performance and reliability of the application.

2.2 Key Technical Design Decision

In order to fully utilize the capabilities of Java EE, the project selected the following key technologies:

2.2.1 Framework selection:

Java EE was selected as the development framework, specifically the latest version 10.0.0 of Jakarta EE was used. As a subsequent version of Java EE, Jakarta EE provides a complete set of enterprise-level development specifications and components, which makes it more efficient to build scalable, secure, and easy-to-maintain enterprise-level applications. By using Jakarta EE, projects can take advantage of the modular components and modern APIs it provides, such as CDI, Servlet API, etc.

2.2.2 Front-end and back-end interaction:

This project uses **Servlet** as the controller component, which is responsible for receiving HTTP requests from the front-end, executing business logic, and passing the results to the **JSP** page for display. The use of Servlet ensures the flexibility and security of request processing, because the JSP page is safely stored in the WEB-INF directory and is not allowed to be accessed directly from the client, thereby avoiding potential security risks. In addition, the combination of Servlet and JSP also makes the project easy to maintain and expand because they follow the standards of the Java EE platform and the business logic is separated from the presentation layer.

2.2.3 Data persistence:

JPA is used as the ORM tool to map objects to the database through annotations, simplifying database operations. All DAO-related classes use JPA. For example, methods in the MovieDAO class use EntityManager instances to perform database operations such as add (persist), update (merge), delete (remove), and query (createQuery).

At the same time, type-safe queries can also be executed through **JPQL** (Java Persistence Query Language). It allows developers to write queries in an object-oriented manner instead of using traditional SQL statements. For example, in the searchByTitle method in the MovieDAO class, the JPQL statement is defined as "*SELECT m FROM Movie m WHERE LOWER(m.title) LIKE :query*", which is used to search based on the movie title, which improves the flexibility and maintainability of the query.

2.2.4 Service layer design:

The service layer is implemented using stateless **EJB**, which can improve the scalability of the application, and the declarative transaction management provided by the EJB container reduces programming complexity.

For example, in Servlet-related classes, such as the LoginServlet class, UserService is injected through the @EJB annotation, which is an EJB service. This injection method allows the components of the service layer to be easily used in the Servlet of the Web layer while maintaining loose coupling between the two. Utilizing this feature of EJB, the service layer can focus more on implementing business logic without having to care about the implementation details of other layers.

2.2.5 Dependency injection:

Use Jakarta EE's **@Inject annotation** to implement dependency injection, which simplifies the acquisition of resources and dependent objects, while improving the testability and maintainability of the code.

In Service-related classes, the @Inject annotation is used. Taking the MovieService class as an example, the @Inject annotation is used to inject MovieDAO instances. In this way, MovieService does not need to care about how to create or obtain an instance of MovieDAO. This approach reduces the coupling between classes and allows MovieService to focus more on its business logic. At the same time, when the implementation of MovieDAO needs to be replaced or modified, there is no need to modify the code of MovieService. You only need to change the injected implementation in the configuration.

2.2.6 Design for security

Ensure application robustness and security are enhanced through form **validation and appropriate error handling**. When handling user requests, such as during login and registration, the system handles potential errors by catching exceptions and displaying error messages. In the LoginServlet class, if the user's credentials are incorrect, the system displays an error message to the user instead of throwing an exception or displaying the system's internal information.

2.2.7 User interface and experience

This project uses **CSS style sheets** to uniformly manage the styles of web pages, mainly for the navigation bar that appears fixedly on each page. In addition, there are different style adjustments for different web pages, making the interface beautiful and user-friendly.

2.2.8 Server selection

GlassFish was chosen as the application server mainly because of its good support for Java EE technology. GlassFish provides a stable and efficient operating environment and supports a full set of Java EE specifications, including EJB, JPA, Servlets, etc., which is conducive to the smooth development and deployment of this project.

3 Visualization and Documentation

This part focuses on converting decisions made during the design and planning phases into concrete diagrams and documents. Specifically, these include: ER Diagram, Sitemap Diagram, User Interface screenshot, and Class diagrams.

3.1 ER Diagram

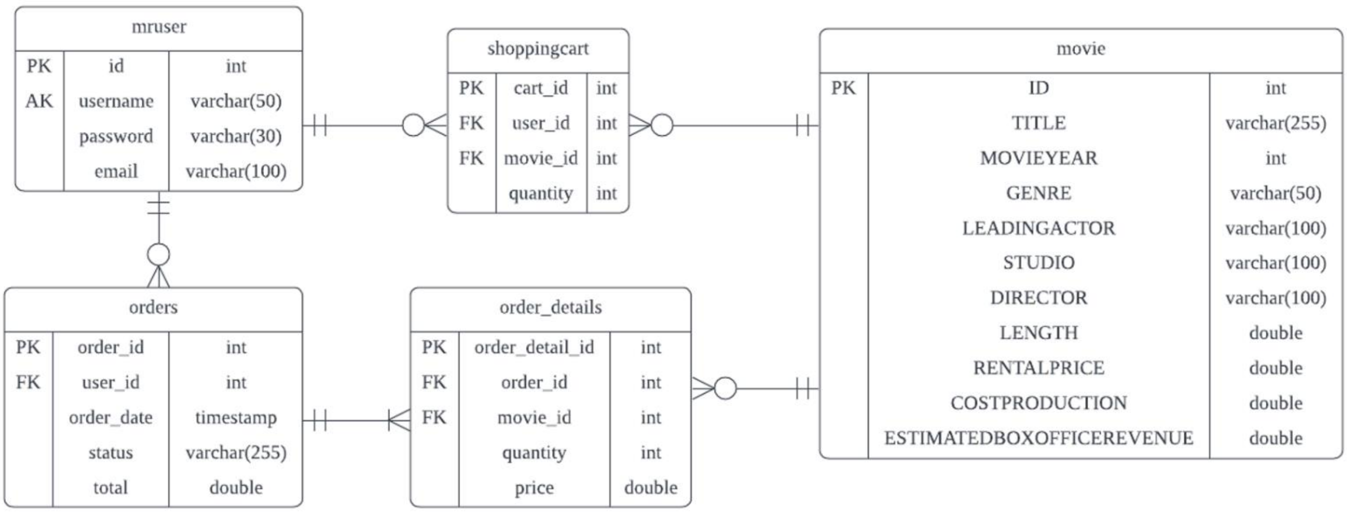


Figure 1 - ER Diagram

Figure 1 above shows the entity relationship diagram of the movie rental system, which contains five entities: mruser, movie, shoppingcart, orders and order_details, and the relationship between them is also represented by specific symbols.

3.2 Sitemap Diagram

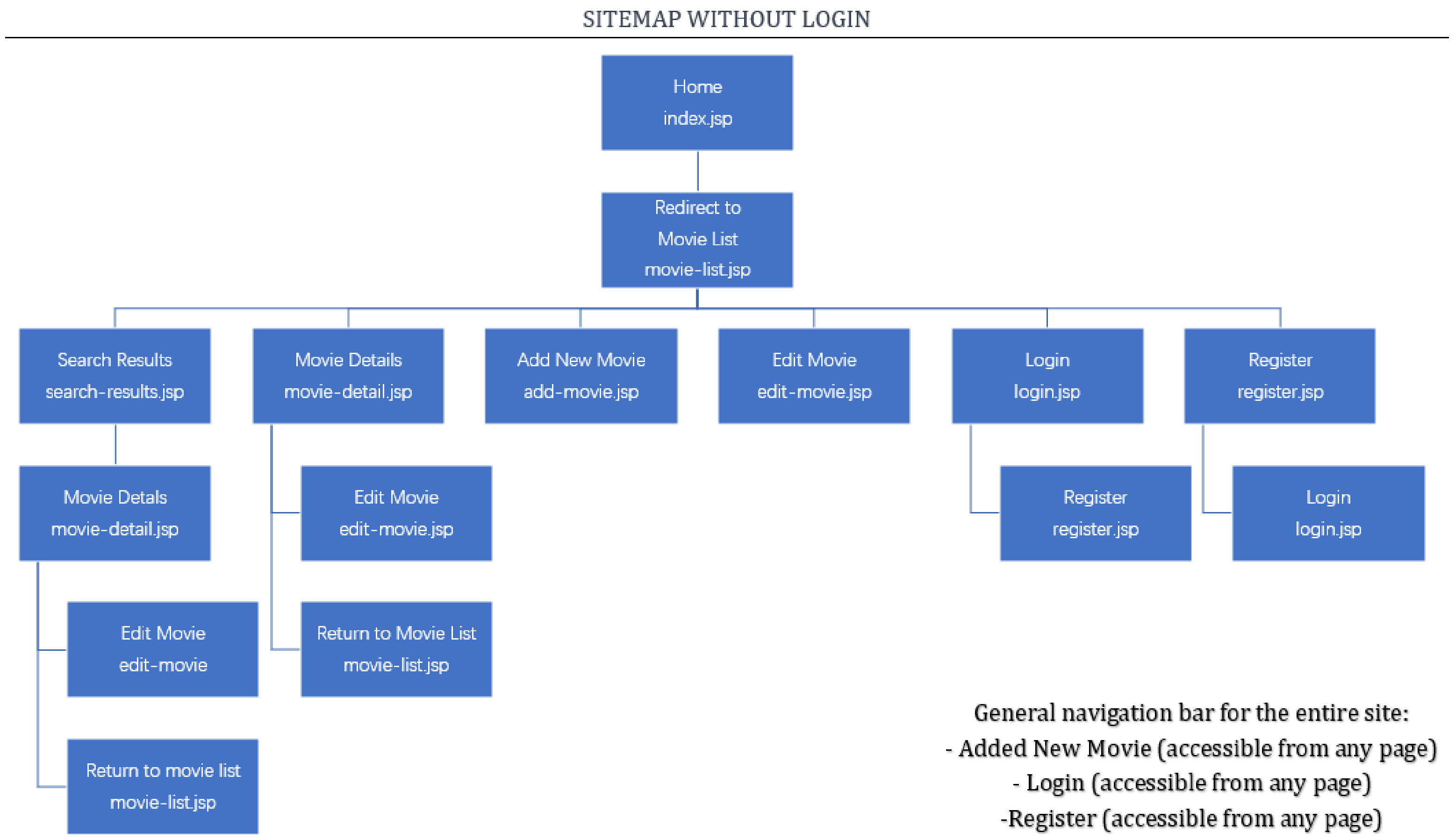
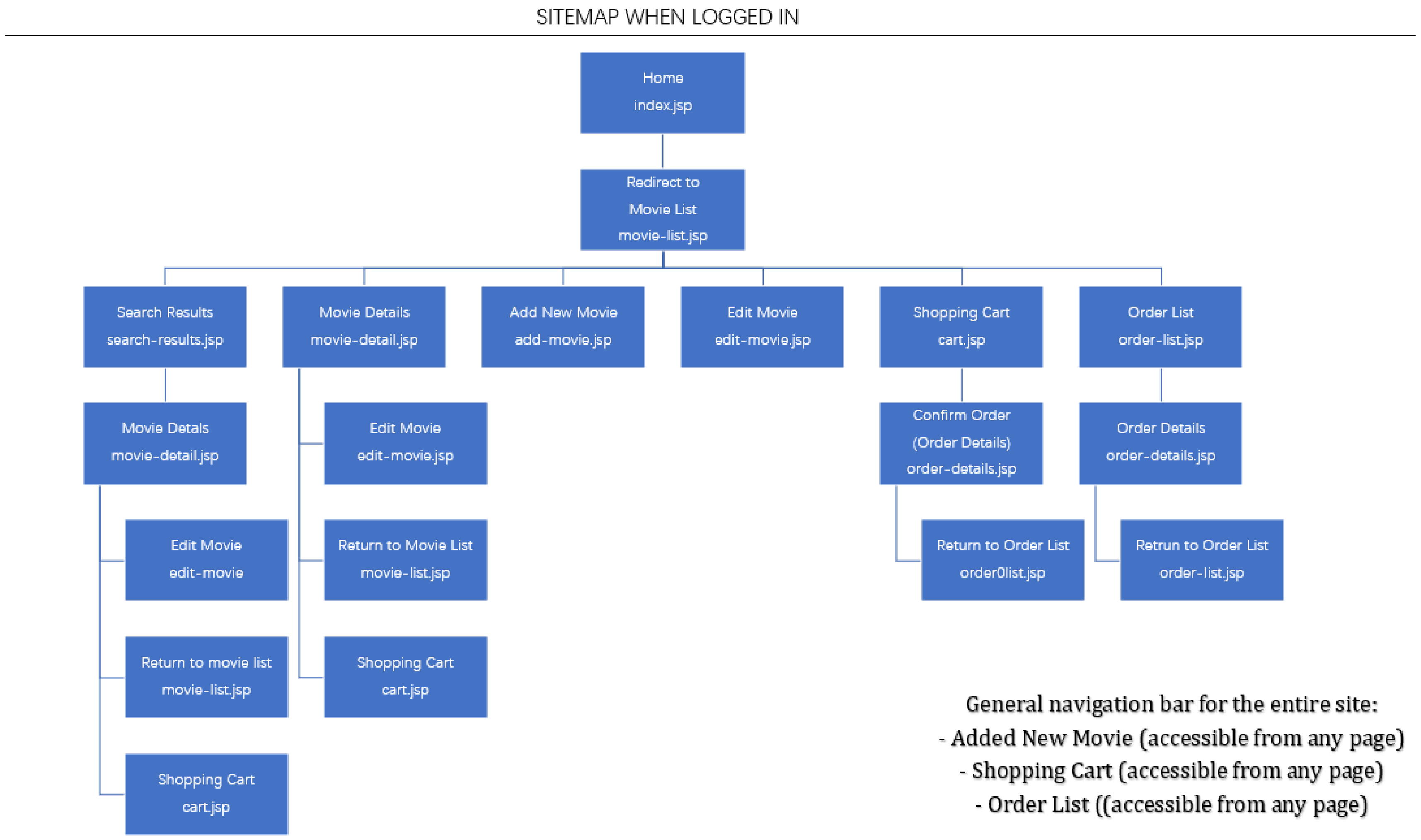


Figure 2 - Sitemap without Login



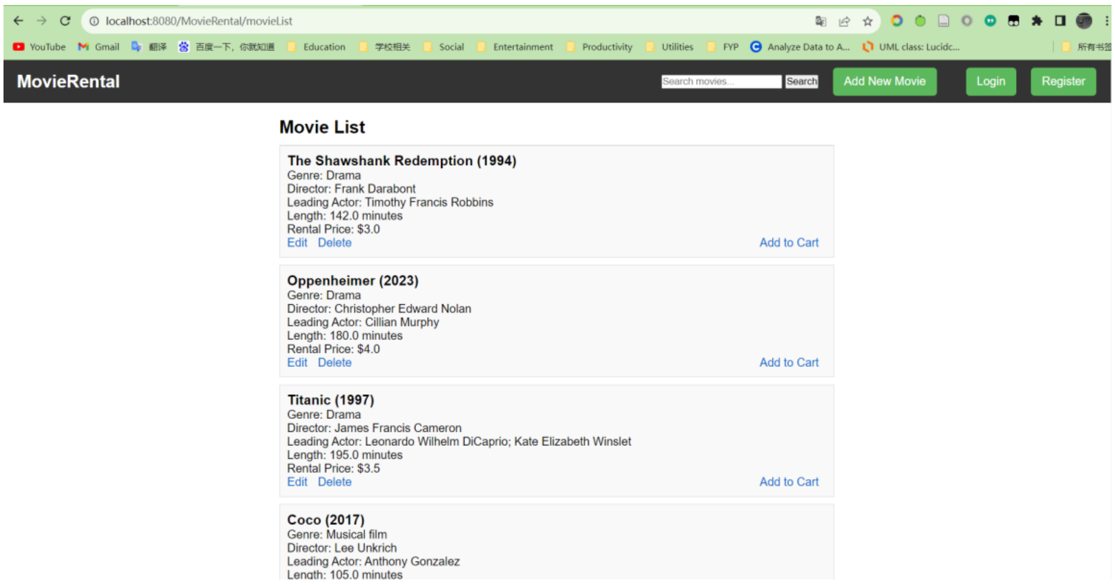
General navigation bar for the entire site:
- Added New Movie (accessible from any page)
- Shopping Cart (accessible from any page)
- Order List ((accessible from any page)

Figure 3 - Sitemap when Logged In

Figure 2 and Figure 3 above are the sitemaps in the unlogged state and logged in state respectively. The more detailed process can be seen in section 3.3 below.

3.3 User Interface Screenshot

This section will display the various interfaces of the Movie Rental Web application.



Movie List

The Shawshank Redemption (1994)
Genre: Drama
Director: Frank Darabont
Leading Actor: Timothy Francis Robbins
Length: 142.0 minutes
Rental Price: $3.0
Edit Delete Add to Cart

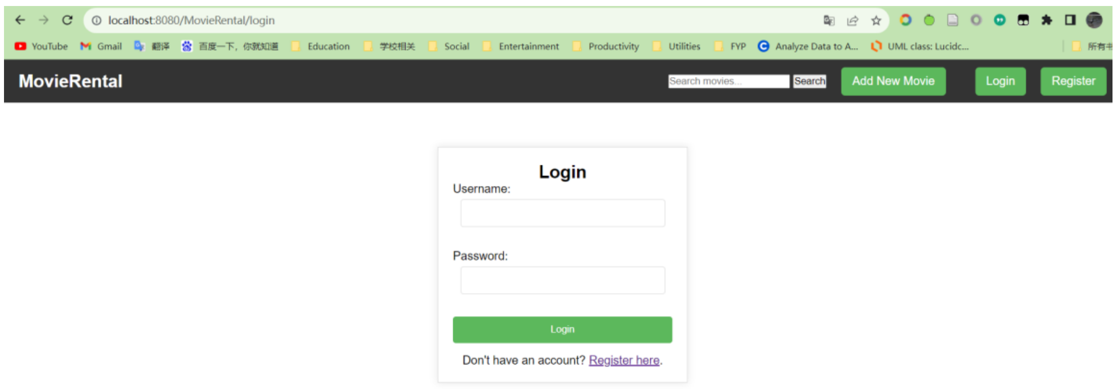
Oppenheimer (2023)
Genre: Drama
Director: Christopher Edward Nolan
Leading Actor: Cillian Murphy
Length: 180.0 minutes
Rental Price: $4.0
Edit Delete Add to Cart

Titanic (1997)
Genre: Drama
Director: James Francis Cameron
Leading Actor: Leonardo Wilhelm DiCaprio; Kate Elizabeth Winslet
Length: 195.0 minutes
Rental Price: $3.5
Edit Delete Add to Cart

Coco (2017)
Genre: Musical film
Director: Lee Unkrich
Leading Actor: Anthony Gonzalez
Length: 105.0 minutes

Figure 4 - Homepage/Movie List Interface without Login

Figure 4 above shows the home page of Movie Rental without logging in, which is also the list page of movies available for rent. This page lists all movies. The top of the page is the navigation bar, the far left is the website name, click to enter the home page, the far right is the search bar, add new movies, login and registration buttons. Movies can be edited and deleted directly in this interface, and users can also directly add them to the shopping cart.

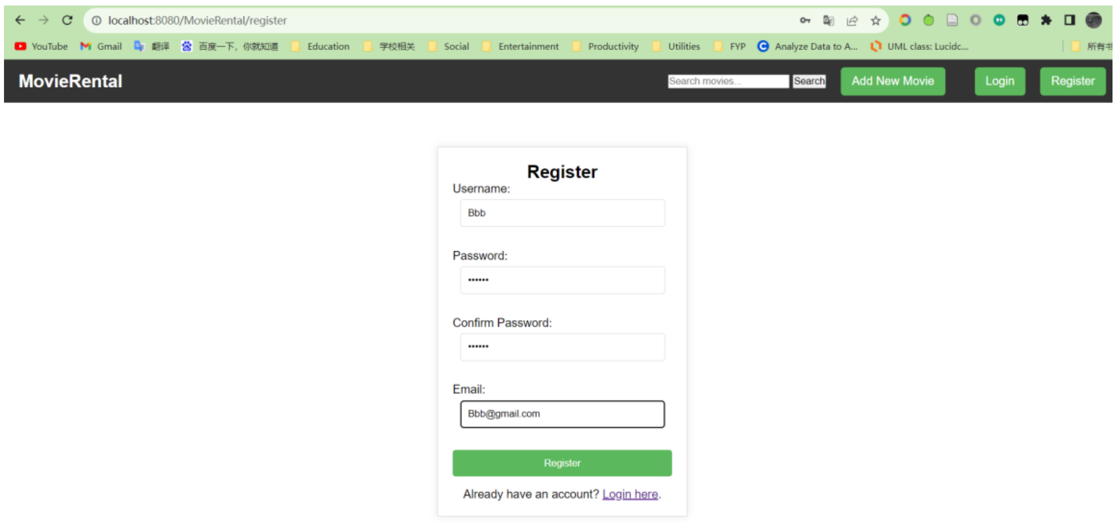


localhost:8080/MovieRental/login

Don't have an account? Register here.

Figure 5 - Login Interface

Figure 5 above shows the login page. If the user does not have an account, he or she can choose to register. Additionally, the navigation bar is fixed in every interface.



localhost:8080/MovieRental/register

Username: Bbb

Password: ••••••

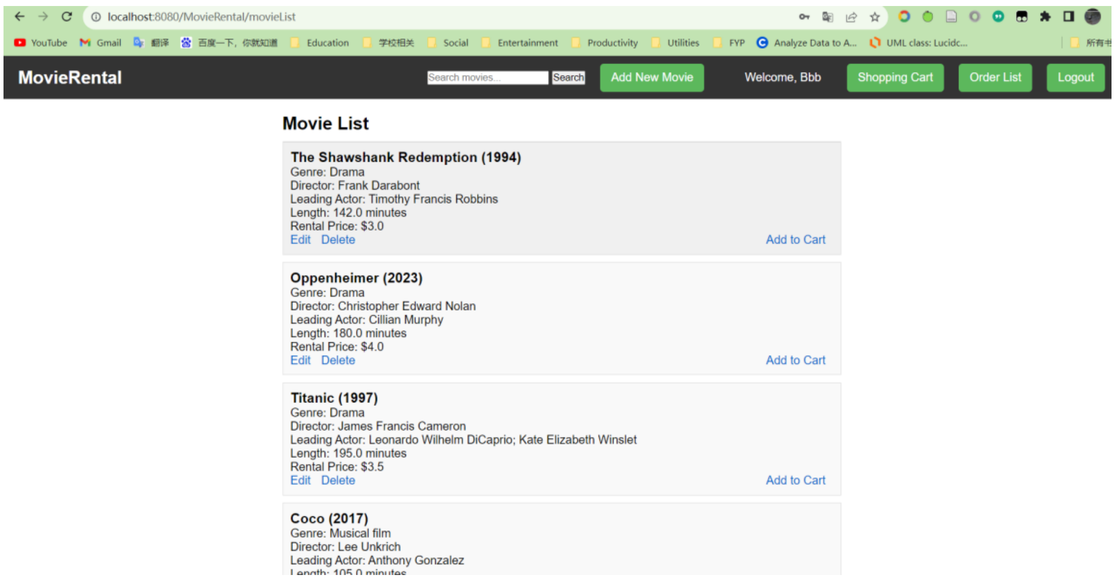
Confirm Password: ••••••

Email: Bbb@gmail.com

Already have an account? Login here.

Figure 6 - Register Interface

Figure 6 above shows the registration page. Users need to fill in a username, the same password twice and an email address. Then click the registration button and you will automatically jump to the login page.



Movie List

The Shawshank Redemption (1994)
Genre: Drama
Director: Frank Darabont
Leading Actor: Timothy Francis Robbins
Length: 142.0 minutes
Rental Price: $3.0
Edit Delete Add to Cart

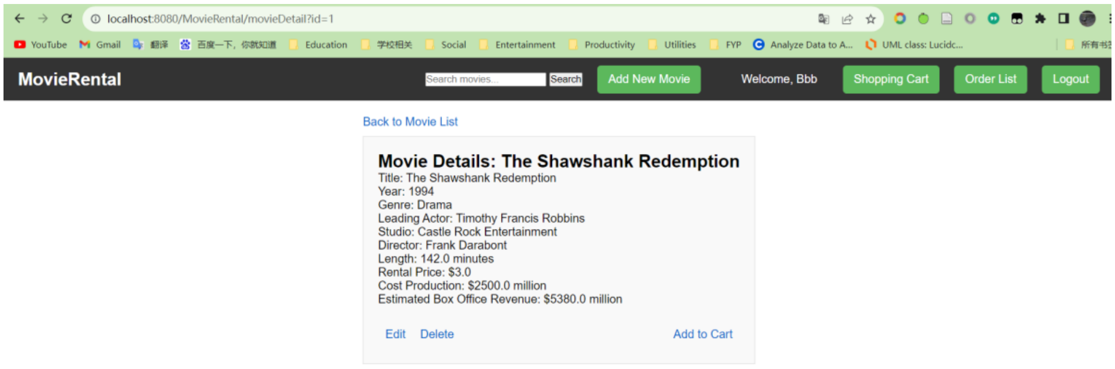
Oppenheimer (2023)
Genre: Drama
Director: Christopher Edward Nolan
Leading Actor: Cillian Murphy
Length: 180.0 minutes
Rental Price: $4.0
Edit Delete Add to Cart

Titanic (1997)
Genre: Drama
Director: James Francis Cameron
Leading Actor: Leonardo Wilhelm DiCaprio; Kate Elizabeth Winslet
Length: 195.0 minutes
Rental Price: $3.5
Edit Delete Add to Cart

Coco (2017)
Genre: Musical film
Director: Lee Unkrich
Leading Actor: Anthony Gonzalez

Figure 7 - Homepage/Movie List Interface with Login

Figure 7 above shows the home page after logging in. The right side of the navigation bar has changed. The login and registration buttons are gone, replaced by shopping cart, order list and logout buttons, and there is also text showing 'Welcome, Bbb'. Bbb here is the username, which will be displayed differently as different users log in.

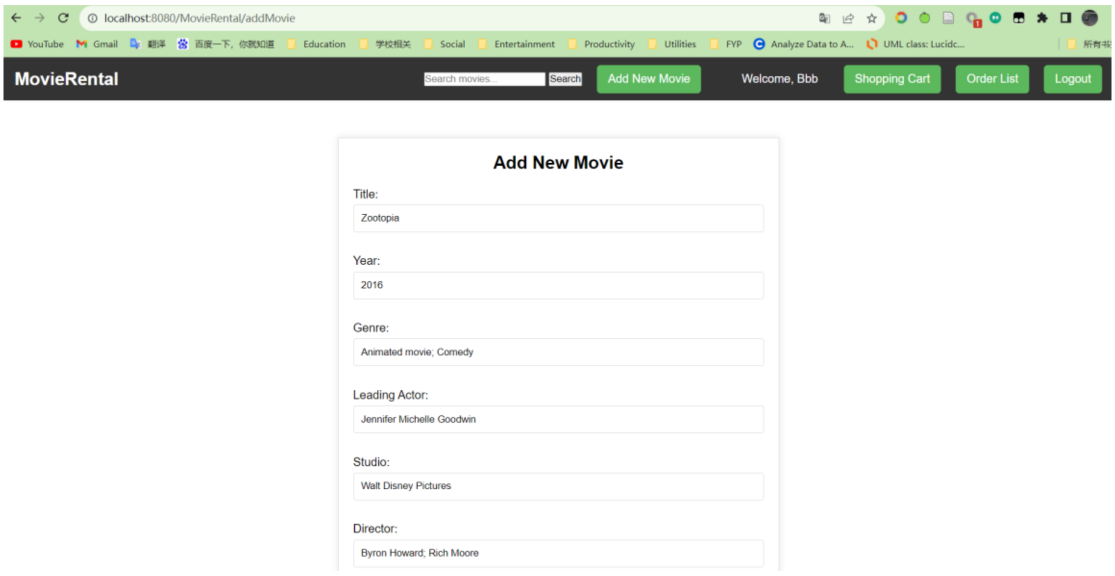


Movie Details: The Shawshank Redemption
Title: The Shawshank Redemption
Year: 1994
Genre: Drama
Leading Actor: Timothy Francis Robbins
Studio: Castle Rock Entertainment
Director: Frank Darabont
Length: 142.0 minutes
Rental Price: $3.0
Cost Production: $2500.0 million
Estimated Box Office Revenue: $5380.0 million

Edit Delete Add to Cart

Figure 8 - Movie Details Interface

Figure above 8 shows a detail page for a single movie, showing all properties about the movie. The interface also has buttons for editing and deleting movies, and the user also chooses to add to the shopping cart here. This interface can be entered by clicking on the box area corresponding to the movie in the movie list page, and it also supports returning to the movie list page. In addition, the movie details page can also be accessed without logging in.



Add New Movie

Title: Zootopia

Year: 2016

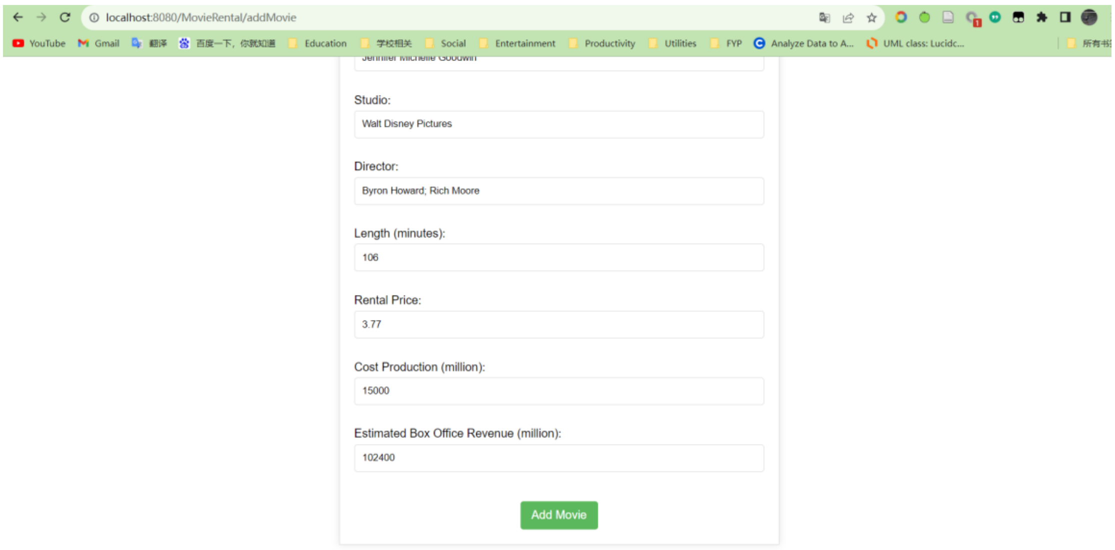
Genre: Animated movie; Comedy

Leading Actor: Jennifer Michelle Goodwin

Studio: Walt Disney Pictures

Director: Byron Howard; Rich Moore

Figure 9 - Add New Movie Interface (Part1)



Studio: Walt Disney Pictures

Director: Byron Howard; Rich Moore

Length (minutes): 106

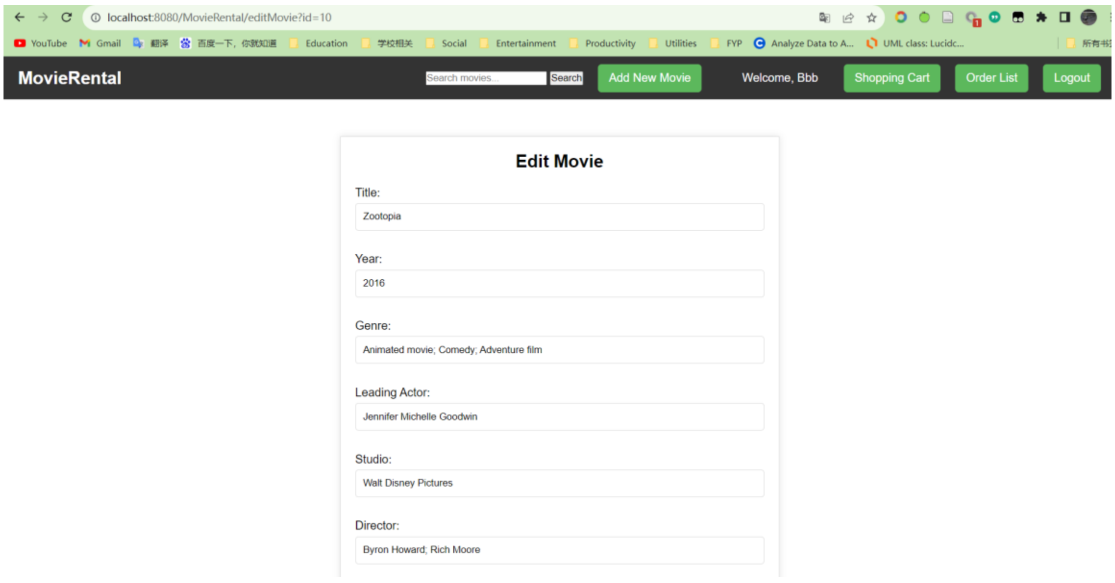
Rental Price: 3.77

Cost Production (million): 15000

Estimated Box Office Revenue (million): 102400

Figure 10 - Add New Movie Interface (Part2)

Figure 9 and Figure 10 above show the page for adding new movies. After filling in all the required values, click the 'Add Movie' button, the new movie will be created successfully, and the page will automatically jump to the movie list page.


localhost:8080/MovieRental/editMovie?id=10

MovieRental

Edit Movie

Title: Zootopia

Year: 2016

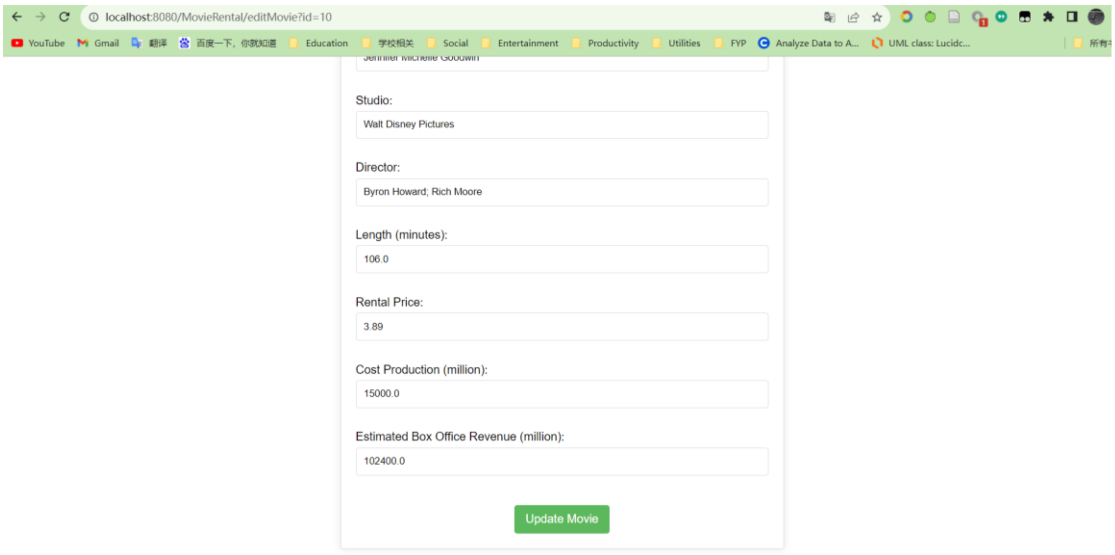
Genre: Animated movie; Comedy; Adventure film

Leading Actor: Jennifer Michelle Goodwin

Studio: Walt Disney Pictures

Director: Byron Howard; Rich Moore

Figure 11 – Edit Movie Interface (Part1)


localhost:8080/MovieRental/editMovie?id=10

Studio: Walt Disney Pictures

Director: Byron Howard; Rich Moore

Length (minutes): 106.0

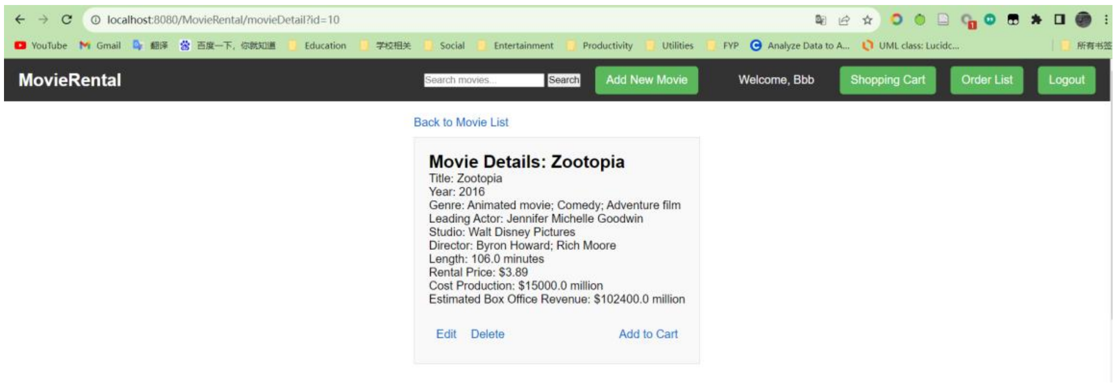
Rental Price: 3.89

Cost Production (million): 15000.0

Estimated Box Office Revenue (million): 102400.0

Figure 12 - Edit Movie Interface (Part2)

As can be seen from Figure 11 and Figure 12above, the movie editing page displays the originally set property values of the movie by default. Only the attributes you want to change need to be modified, avoiding repeated entry of other attributes. Here, 'Adventure film' is added to 'Genre', and the 'Rental Price' is changed to 3.89, and the others remain unchanged. Figure 13 below shows the modified movie information, and it can be seen that the editing is effective.



MovieRental

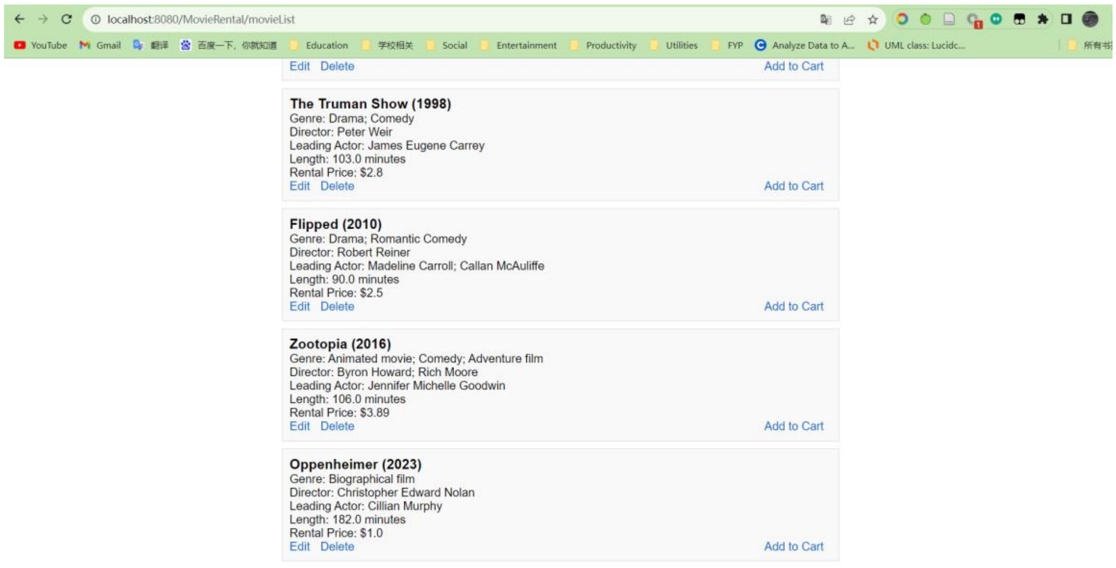
Welcome, Bbb

Back to Movie List

Movie Details: Zootopia
Title: Zootopia
Year: 2016
Genre: Animated movie; Comedy; Adventure film
Leading Actor: Jennifer Michelle Goodwin
Studio: Walt Disney Pictures
Director: Byron Howard; Rich Moore
Length: 106.0 minutes
Rental Price: $3.89
Cost Production: $15000.0 million
Estimated Box Office Revenue: $102400.0 million

Edit Delete Add to Cart

Figure 13 - The Modified Movie Details Interface



Edit Delete Add to Cart

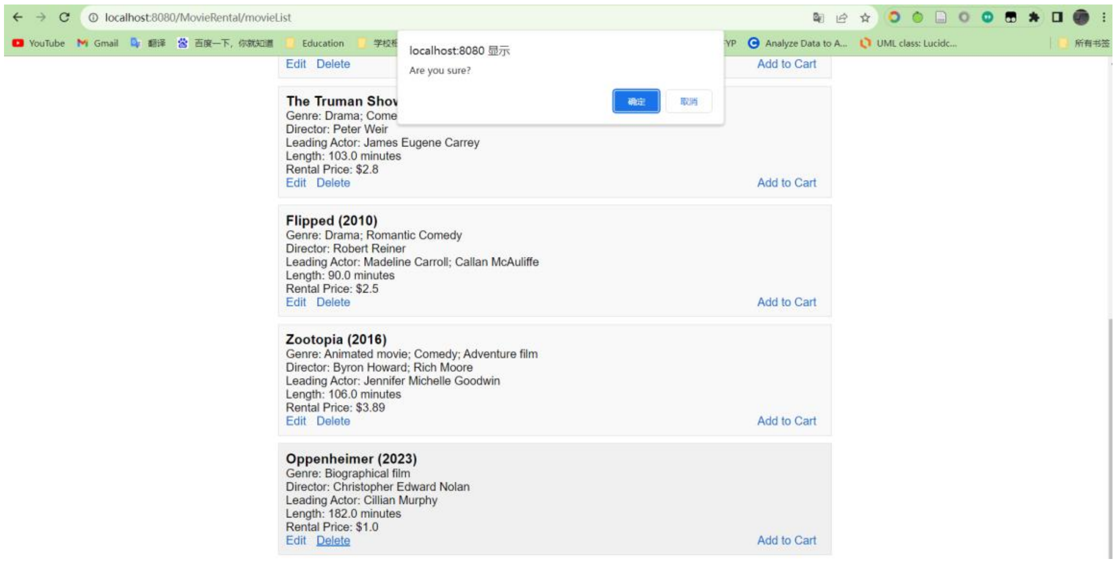
The Truman Show (1998)
Genre: Drama; Comedy
Director: Peter Weir
Leading Actor: James Eugene Carrey
Length: 103.0 minutes
Rental Price: $2.8
Edit Delete Add to Cart

Flipped (2010)
Genre: Drama; Romantic Comedy
Director: Robert Reiner
Leading Actor: Madeline Carroll; Callan McAuliffe
Length: 90.0 minutes
Rental Price: $2.5
Edit Delete Add to Cart

Zootopia (2016)
Genre: Animated movie; Comedy; Adventure film
Director: Byron Howard; Rich Moore
Leading Actor: Jennifer Michelle Goodwin
Length: 106.0 minutes
Rental Price: $3.89
Edit Delete Add to Cart

Oppenheimer (2023)
Genre: Biographical film
Director: Christopher Edward Nolan
Leading Actor: Cillian Murphy
Length: 182.0 minutes
Rental Price: $1.0
Edit Delete Add to Cart

Figure 14- Movie List Interface without Deletion



localhost:8080 显示
Are you sure?

Edit Delete Add to Cart

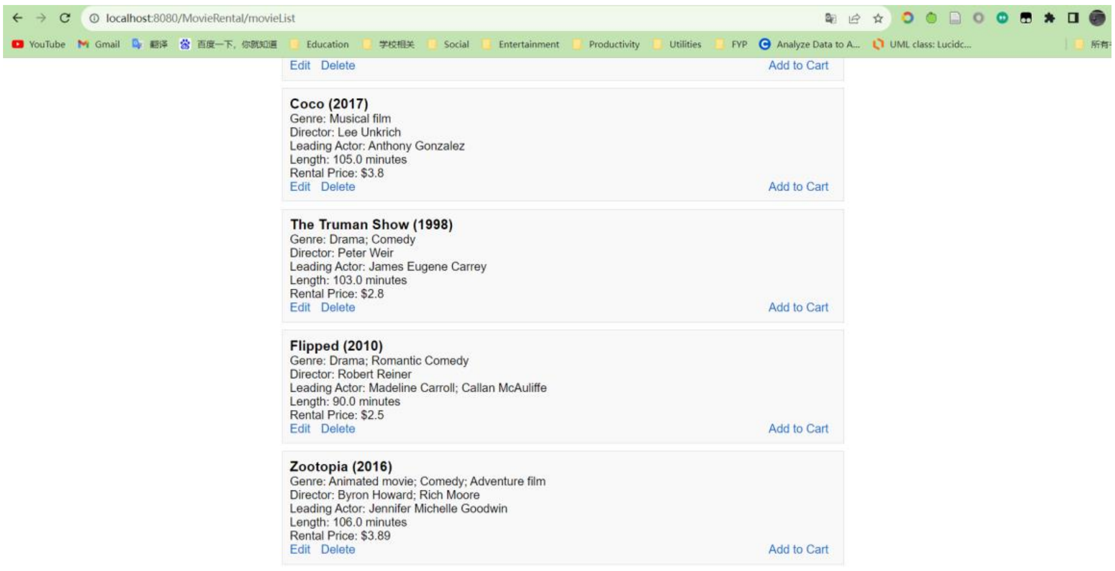
The Truman Shov
Genre: Drama; Come
Director: Peter Weir
Leading Actor: James Eugene Carrey
Length: 103.0 minutes
Rental Price: $2.8
Edit Delete Add to Cart

Flipped (2010)
Genre: Drama; Romantic Comedy
Director: Robert Reiner
Leading Actor: Madeline Carroll; Callan McAuliffe
Length: 90.0 minutes
Rental Price: $2.5
Edit Delete Add to Cart

Zootopia (2016)
Genre: Animated movie; Comedy; Adventure film
Director: Byron Howard; Rich Moore
Leading Actor: Jennifer Michelle Goodwin
Length: 106.0 minutes
Rental Price: $3.89
Edit Delete Add to Cart

Oppenheimer (2023)
Genre: Biographical film
Director: Christopher Edward Nolan
Leading Actor: Cillian Murphy
Length: 182.0 minutes
Rental Price: $1.0
Edit Delete Add to Cart

Figure 15 - Movie List Interface where Deletion in Progress


Edit Delete Add to Cart

Coco (2017)
Genre: Musical film
Director: Lee Unkrich
Leading Actor: Anthony Gonzalez
Length: 105.0 minutes
Rental Price: $3.8
Edit Delete Add to Cart

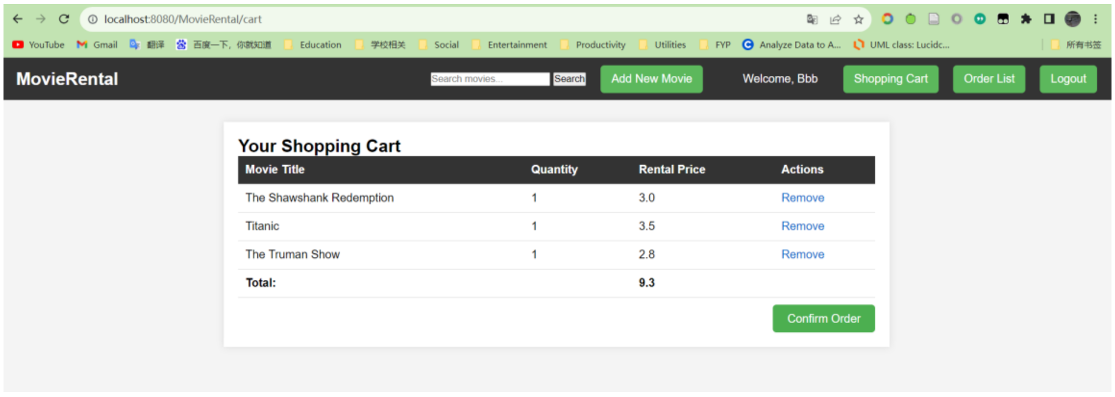
The Truman Show (1998)
Genre: Drama; Comedy
Director: Peter Weir
Leading Actor: James Eugene Carrey
Length: 103.0 minutes
Rental Price: $2.8
Edit Delete Add to Cart

Flipped (2010)
Genre: Drama; Romantic Comedy
Director: Robert Reiner
Leading Actor: Madeline Carroll; Callan McAuliffe
Length: 90.0 minutes
Rental Price: $2.5
Edit Delete Add to Cart

Zootopia (2016)
Genre: Animated movie; Comedy; Adventure film
Director: Byron Howard; Rich Moore
Leading Actor: Jennifer Michelle Goodwin
Length: 106.0 minutes
Rental Price: $3.89
Edit Delete Add to Cart

Figure 16 - Movie List Interface after Deletion Completed

Figure 14, Figure 15 and Figure 16 above show the process of deleting a movie. After clicking the delete button, a pop-up window will appear to confirm whether the movie is really deleted. Only after clicking the confirmation button, the movie will be deleted.



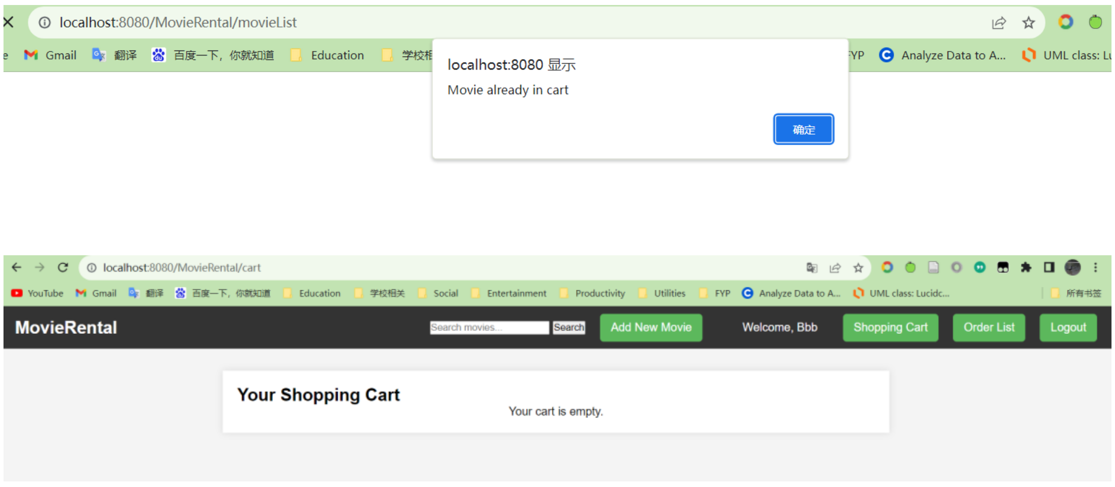
Your Shopping Cart

Movie Title	Quantity	Rental Price	Actions
The Shawshank Redemption	1	3.0	Remove
Titanic	1	3.5	Remove
The Truman Show	1	2.8	Remove
Total:		9.3	

Confirm Order

Figure 17 - Shopping Cart Interface

Figure 17 above shows the shopping cart interface. The number of added movies is limited to 1 by default, which means that when the movie already exists in the shopping cart, the system will pop up a pop-up window to prevent the user from adding it to the shopping cart again (Figure 18). The shopping cart interface also displays the total rental price for all movies combined. And it supports removal operations, allowing you to remove movies you don’t want to rent from your shopping cart.


localhost:8080/MovieRental/movieList

localhost:8080 显示

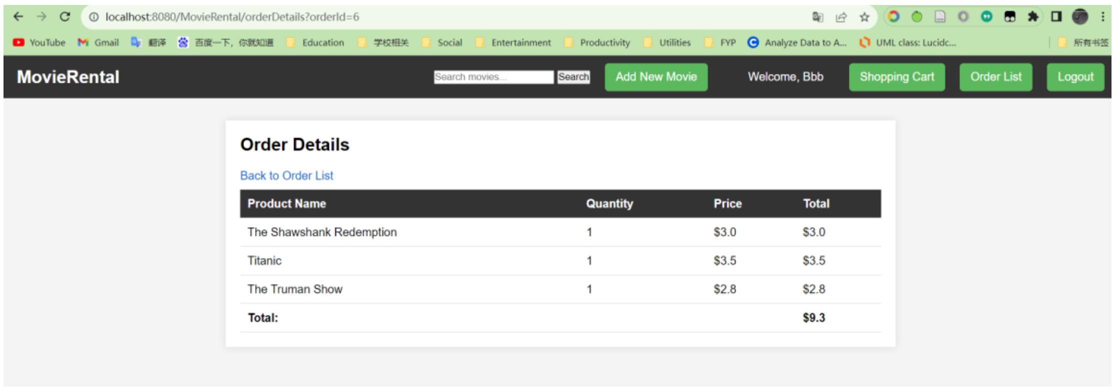
Movie already in cart

localhost:8080/MovieRental/cart

Your Shopping Cart

Your cart is empty.

Figure 18 - Shopping Cart Interface after Confirm Order


localhost:8080/MovieRental/orderDetails?orderId=6

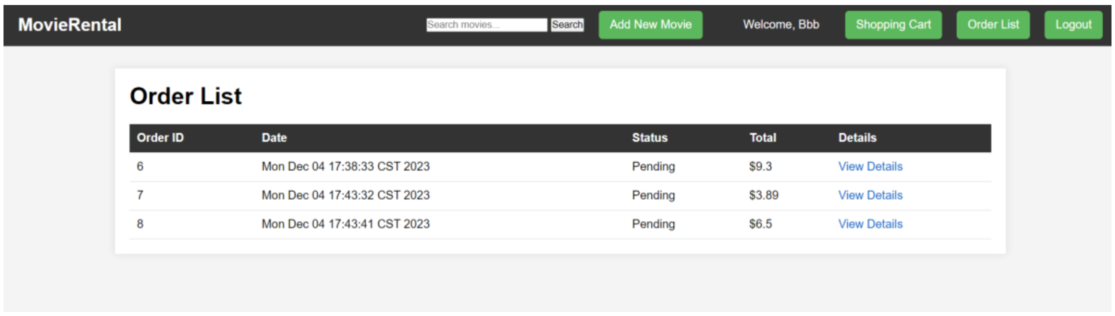
Order Details

Back to Order List

Product Name	Quantity	Price	Total
The Shawshank Redemption	1	$3.0	$3.0
Titanic	1	$3.5	$3.5
The Truman Show	1	$2.8	$2.8
Total:			**$9.3**

Figure 19 - Order Detail Interface

After clicking 'Confirm Order' in the shopping cart, the website will automatically clear the shopping cart, create its corresponding order, and the page will automatically jump to the newly created order details page. As shown in Figure 19 above, the page displays the product name, quantity, unit price and total price, and also provides a button to jump to the order list.

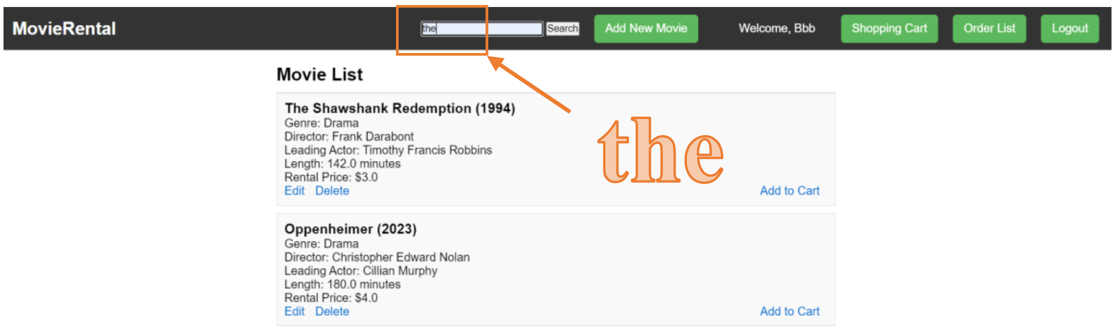


Order List

Order ID	Date	Status	Total	Details
6	Mon Dec 04 17:38:33 CST 2023	Pending	$9.3	View Details
7	Mon Dec 04 17:43:32 CST 2023	Pending	$3.89	View Details
8	Mon Dec 04 17:43:41 CST 2023	Pending	$6.5	View Details

Figure 20 - Order List Interface

Figure 20 above shows the order list interface, which includes the order ID, creation time, status and total price, and has a button that allows users to view the corresponding order details page.

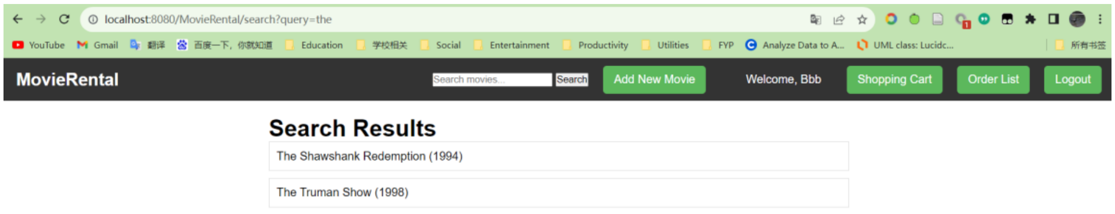


Movie List

The Shawshank Redemption (1994)
Genre: Drama
Director: Frank Darabont
Leading Actor: Timothy Francis Robbins
Length: 142.0 minutes
Rental Price: $3.0
Edit Delete Add to Cart

Oppenheimer (2023)
Genre: Drama
Director: Christopher Edward Nolan
Leading Actor: Cillian Murphy
Length: 180.0 minutes
Rental Price: $4.0
Edit Delete Add to Cart

Figure 21 - Search Bar with 'the'



Search Results

The Shawshank Redemption (1994)

The Truman Show (1998)

Figure 22 - Search Results Interface

As shown in Figure 21 above, enter 'the' in the search bar, and the results as shown in the picture will appear. You can click on the movie in the results to go to the corresponding movie details page (Figure 22). Note that searches in the search bar are case-insensitive.

In addition to the above description, it is also worth noting that when the user clicks the add to cart button when not logged in, the page will automatically jump to the login page, and the same is true for the order page.

3.4 Class Diagrams

This section contains UML class diagrams for all classes, ranging from Figure23 to Figure53.

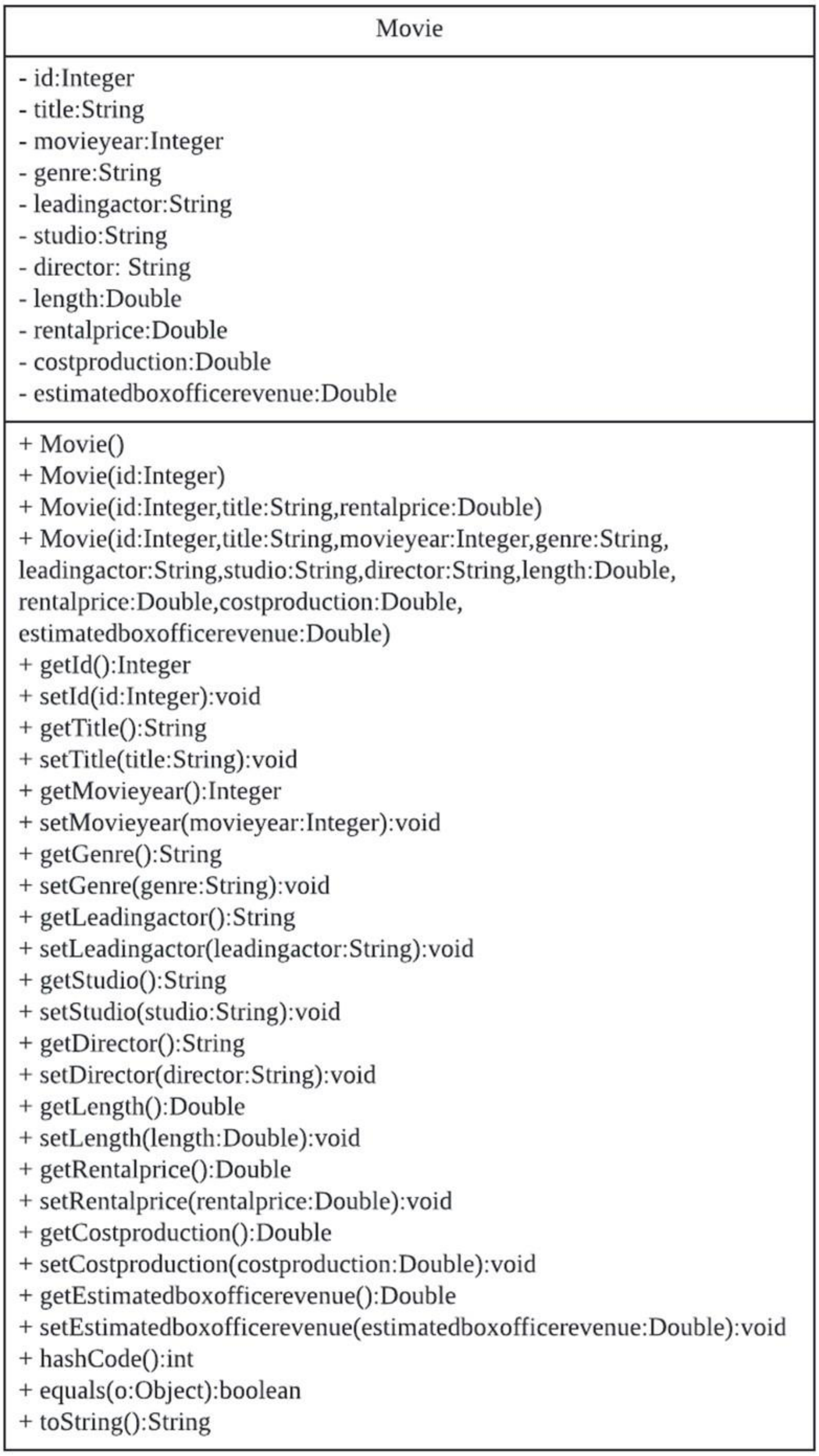

Movie
- id:Integer
- title:String
- movieyear:Integer
- genre:String
- leadingactor:String
- studio:String
- director: String
- length:Double
- rentalprice:Double
- costproduction:Double
- estimatedboxofficerevenue:Double
+ Movie()
+ Movie(id:Integer)
+ Movie(id:Integer,title:String,rentalprice:Double)
+ Movie(id:Integer,title:String,movieyear:Integer,genre:String, leadingactor:String,studio:String,director:String,length:Double, rentalprice:Double,costproduction:Double, estimatedboxofficerevenue:Double)
+ getId():Integer
+ setId(id:Integer):void
+ getTitle():String
+ setTitle(title:String):void
+ getMovieyear():Integer
+ setMovieyear(movieyear:Integer):void
+ getGenre():String
+ setGenre(genre:String):void
+ getLeadingactor():String
+ setLeadingactor(leadingactor:String):void
+ getStudio():String
+ setStudio(studio:String):void
+ getDirector():String
+ setDirector(director:String):void
+ getLength():Double
+ setLength(length:Double):void
+ getRentalprice():Double
+ setRentalprice(rentalprice:Double):void
+ getCostproduction():Double
+ setCostproduction(costproduction:Double):void
+ getEstimatedboxofficerevenue():Double
+ setEstimatedboxofficerevenue(estimatedboxofficerevenue:Double):void
+ hashCode():int
+ equals(o:Object):boolean
+ toString():String

Figure 23 - Movie Class

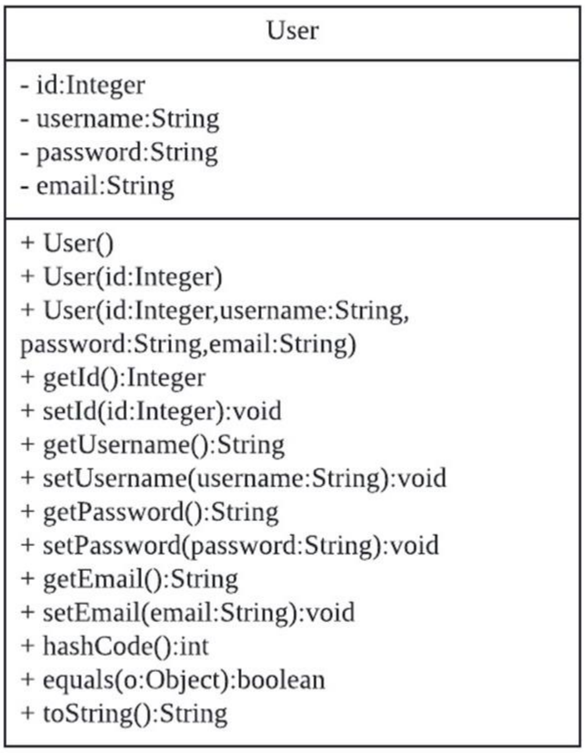


Figure 24 - User Class

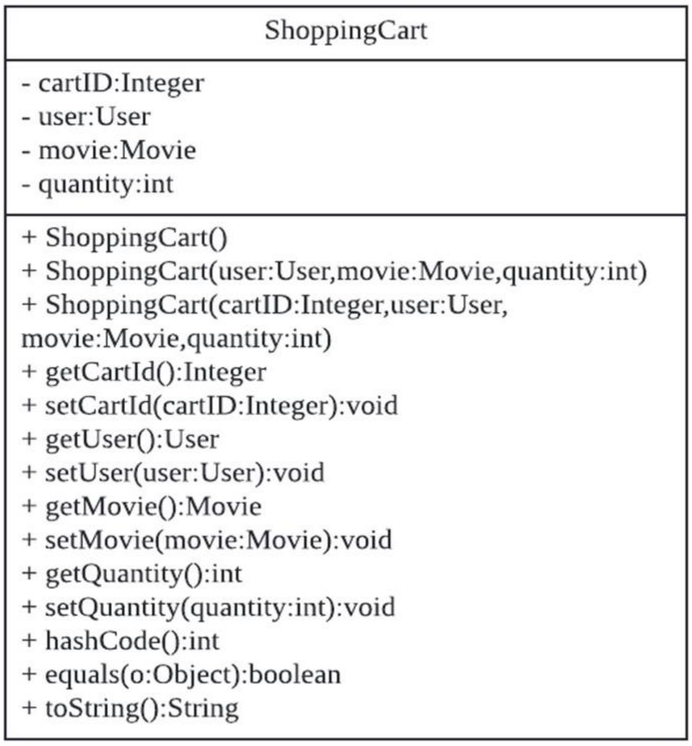

ShoppingCart
- cartID:Integer
- user:User
- movie:Movie
- quantity:int
+ ShoppingCart()
+ ShoppingCart(user:User,movie:Movie,quantity:int)
+ ShoppingCart(cartID:Integer,user:User,
movie:Movie,quantity:int)
+ getCartId():Integer
+ setCartId(cartID:Integer):void
+ getUser():User
+ setUser(user:User):void
+ getMovie():Movie
+ setMovie(movie:Movie):void
+ getQuantity():int
+ setQuantity(quantity:int):void
+ hashCode():int
+ equals(o:Object):boolean
+ toString():String

Figure 25 - ShoppingCart Class

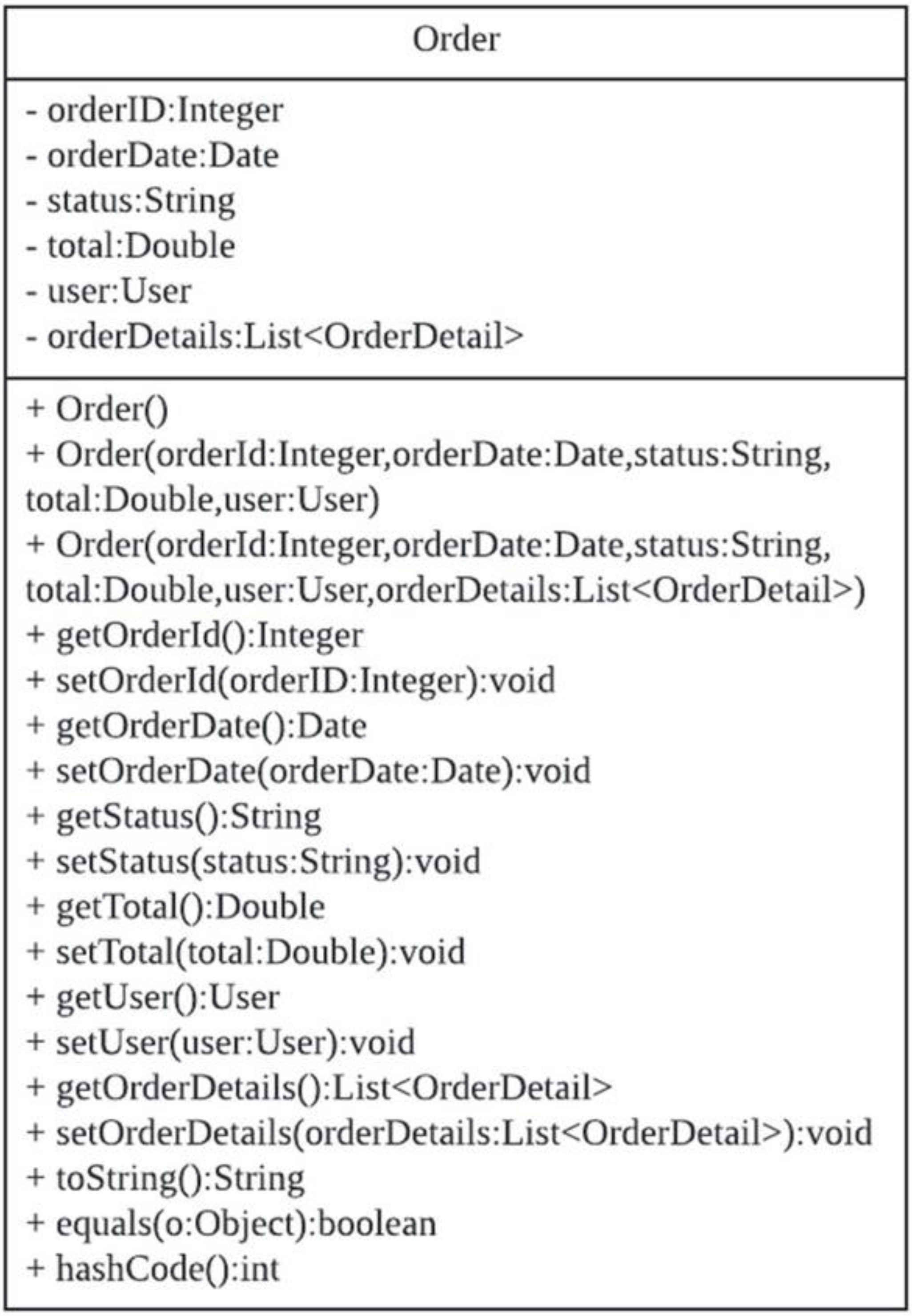

Order
- orderID:Integer
- orderDate:Date
- status:String
- total:Double
- user:User
- orderDetails:List<OrderDetail>
+ Order()
+ Order(orderId:Integer,orderDate:Date,status:String, total:Double,user:User)
+ Order(orderId:Integer,orderDate:Date,status:String, total:Double,user:User,orderDetails:List<OrderDetail>)
+ getOrderId():Integer
+ setOrderId(orderID:Integer):void
+ getOrderDate():Date
+ setOrderDate(orderDate:Date):void
+ getStatus():String
+ setStatus(status:String):void
+ getTotal():Double
+ setTotal(total:Double):void
+ getUser():User
+ setUser(user:User):void
+ getOrderDetails():List<OrderDetail>
+ setOrderDetails(orderDetails:List<OrderDetail>):void
+ toString():String
+ equals(o:Object):boolean
+ hashCode():int

Figure 26 - Order Class

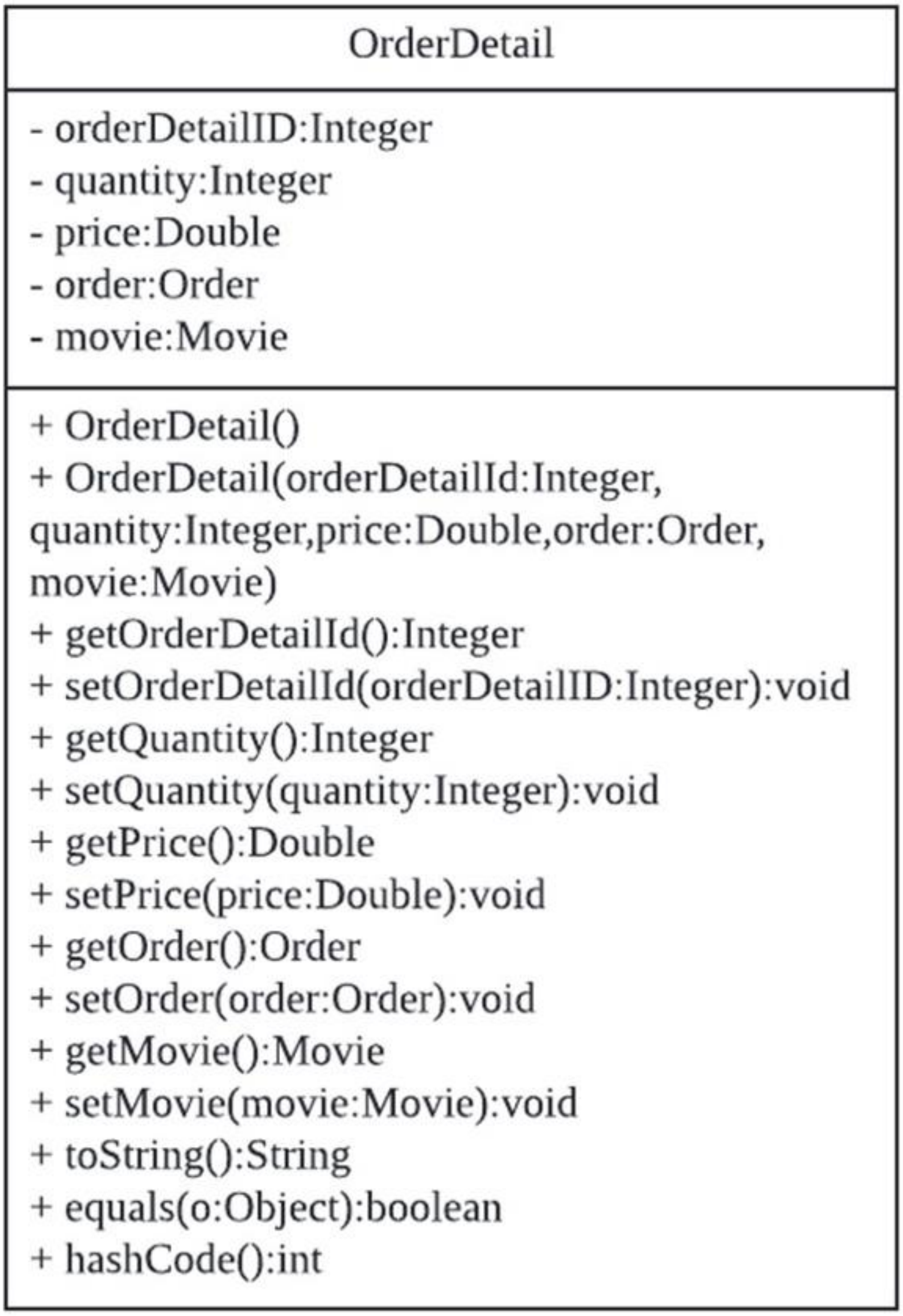

OrderDetail
- orderDetailID:Integer
- quantity:Integer
- price:Double
- order:Order
- movie:Movie
+ OrderDetail()
+ OrderDetail(orderDetailId:Integer, quantity:Integer,price:Double,order:Order, movie:Movie)
+ getOrderDetailId():Integer
+ setOrderDetailId(orderDetailID:Integer):void
+ getQuantity():Integer
+ setQuantity(quantity:Integer):void
+ getPrice():Double
+ setPrice(price:Double):void
+ getOrder():Order
+ setOrder(order:Order):void
+ getMovie():Movie
+ setMovie(movie:Movie):void
+ toString():String
+ equals(o:Object):boolean
+ hashCode():int

Figure 27 - OrderDetail Class

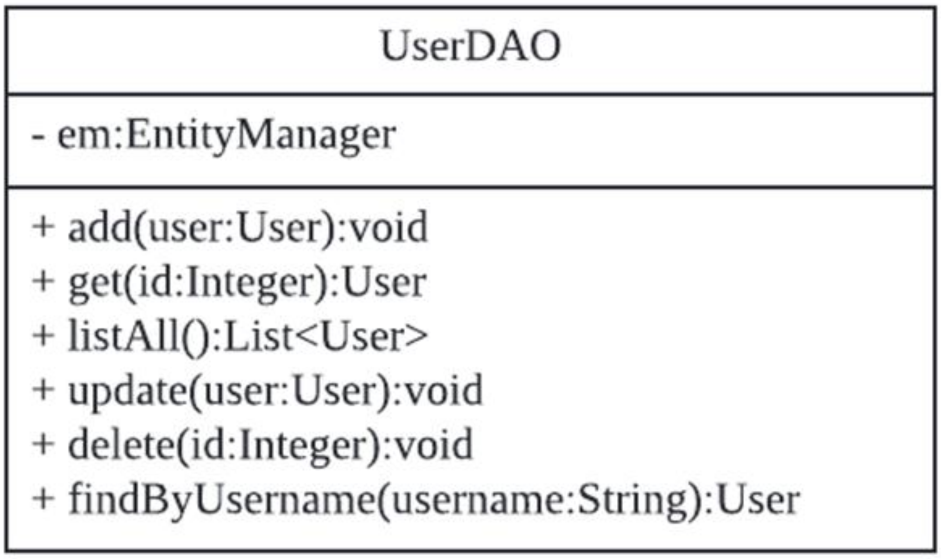


Figure 28 - UserDAO Class

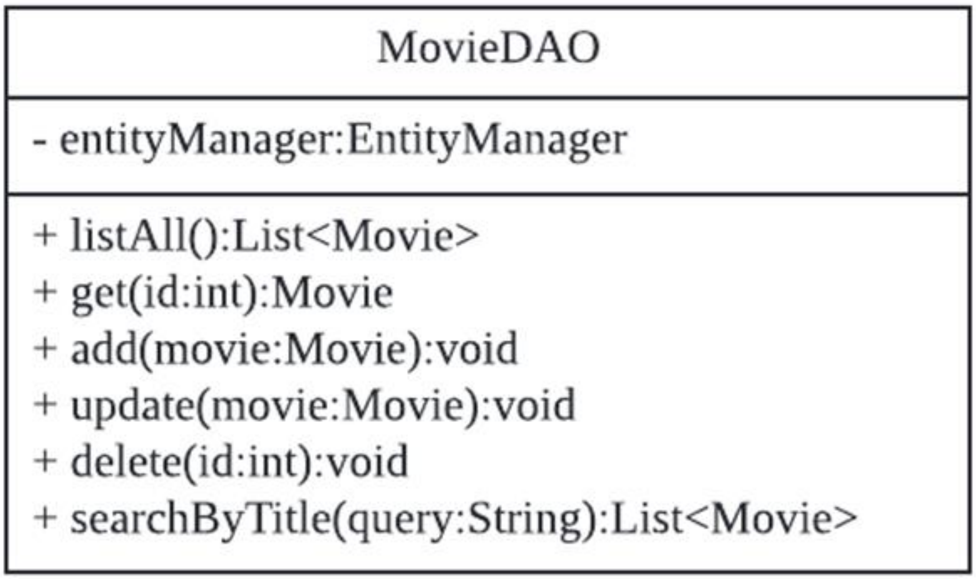

MovieDAO
- entityManager:EntityManager
+ listAll():List<Movie>
+ get(id:int):Movie
+ add(movie:Movie):void
+ update(movie:Movie):void
+ delete(id:int):void
+ searchByTitle(query:String):List<Movie>

Figure 29 - MovieDAO Class

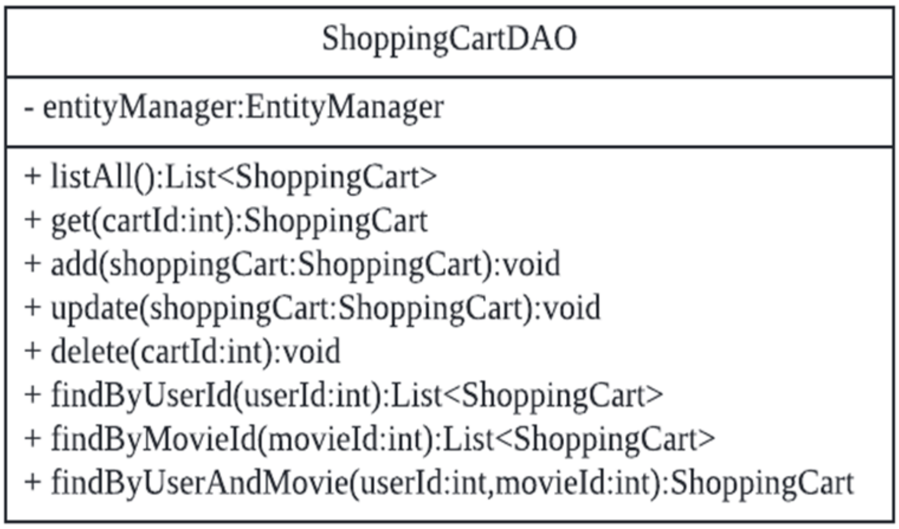

ShoppingCartDAO
- entityManager:EntityManager
+ listAll():List<ShoppingCart>
+ get(cartId:int):ShoppingCart
+ add(shoppingCart:ShoppingCart):void
+ update(shoppingCart:ShoppingCart):void
+ delete(cartId:int):void
+ findByUserId(userId:int):List<ShoppingCart>
+ findByMovieId(movieId:int):List<ShoppingCart>
+ findByUserAndMovie(userId:int,movieId:int):ShoppingCart

Figure 30 - ShoppingCartDAO Class

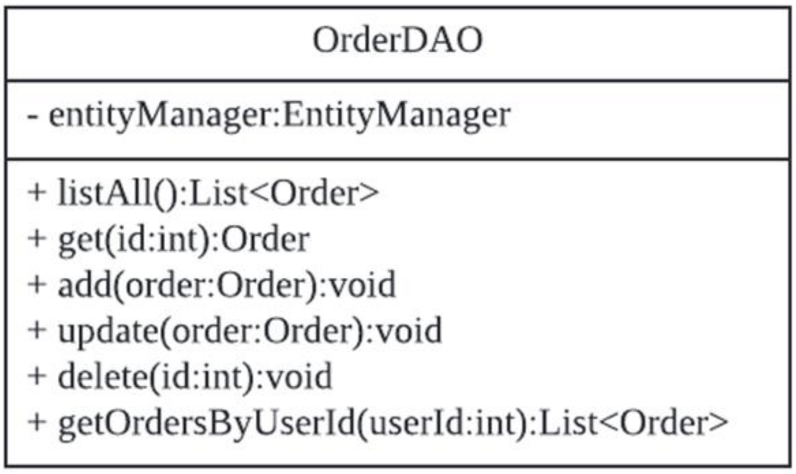

OrderDAO
- entityManager:EntityManager
+ listAll():List<Order>
+ get(id:int):Order
+ add(order:Order):void
+ update(order:Order):void
+ delete(id:int):void
+ getOrdersByUserId(userId:int):List<Order>

Figure 31 - OrderDAO Class

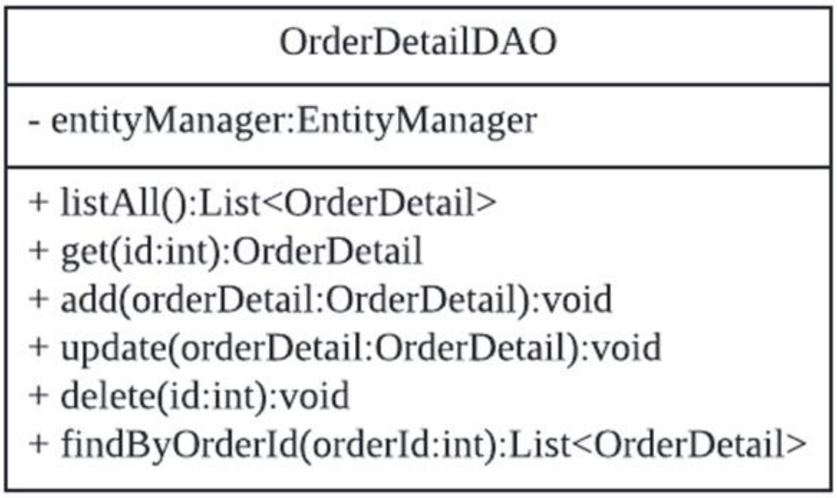

OrderDetailDAO
- entityManager:EntityManager
+ listAll():List<OrderDetail>
+ get(id:int):OrderDetail
+ add(orderDetail:OrderDetail):void
+ update(orderDetail:OrderDetail):void
+ delete(id:int):void
+ findByOrderId(orderId:int):List<OrderDetail>

Figure 32 - OrderDetailDAO Class

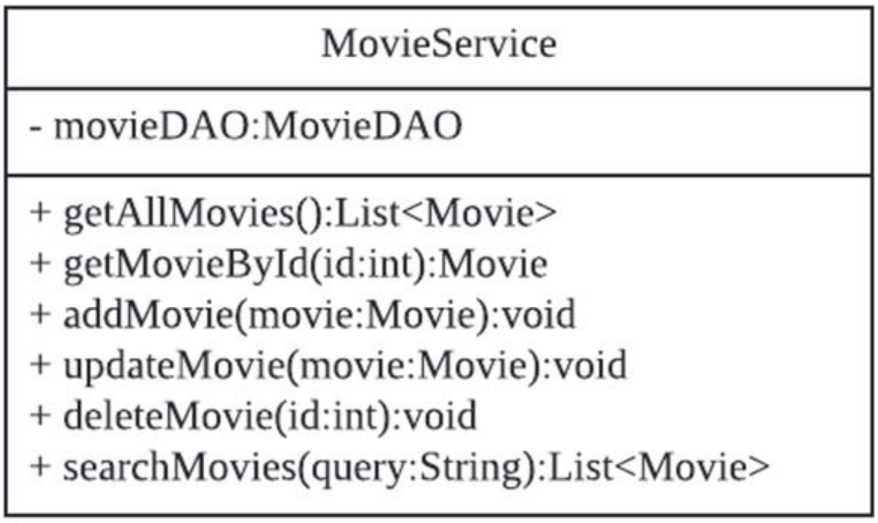

MovieService
- movieDAO:MovieDAO
+ getAllMovies():List<Movie>
+ getMovieById(id:int):Movie
+ addMovie(movie:Movie):void
+ updateMovie(movie:Movie):void
+ deleteMovie(id:int):void
+ searchMovies(query:String):List<Movie>

Figure 33 - MovieService Class

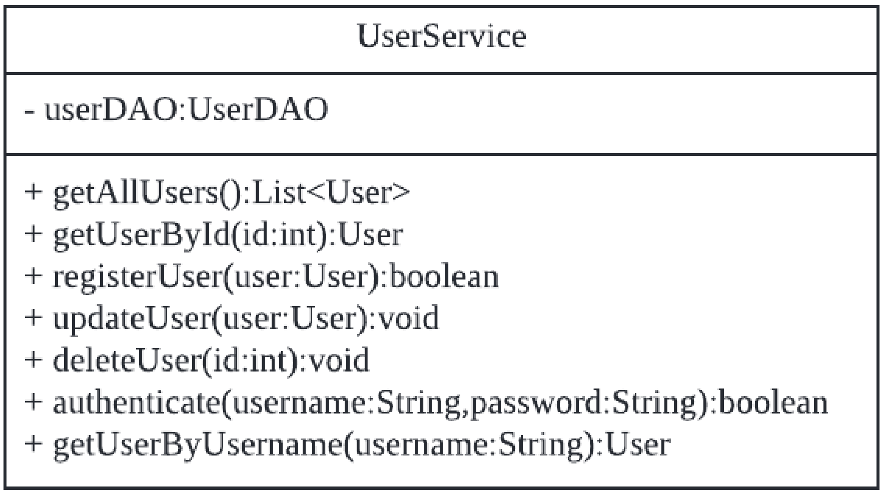
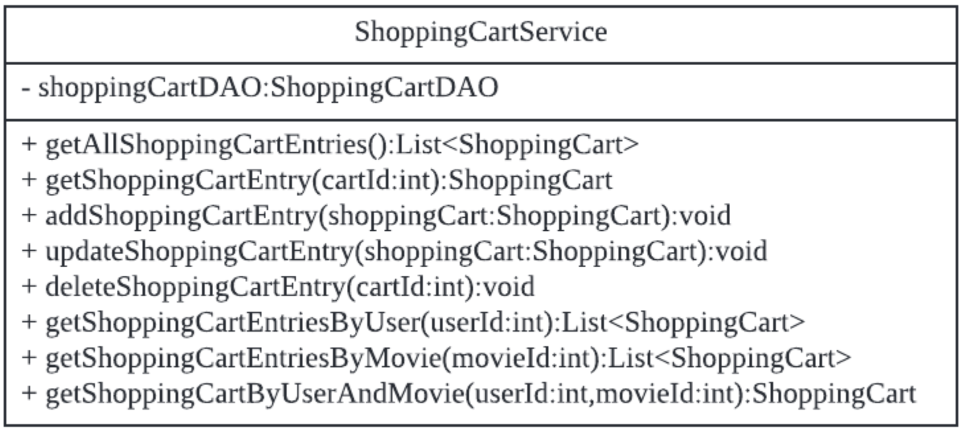
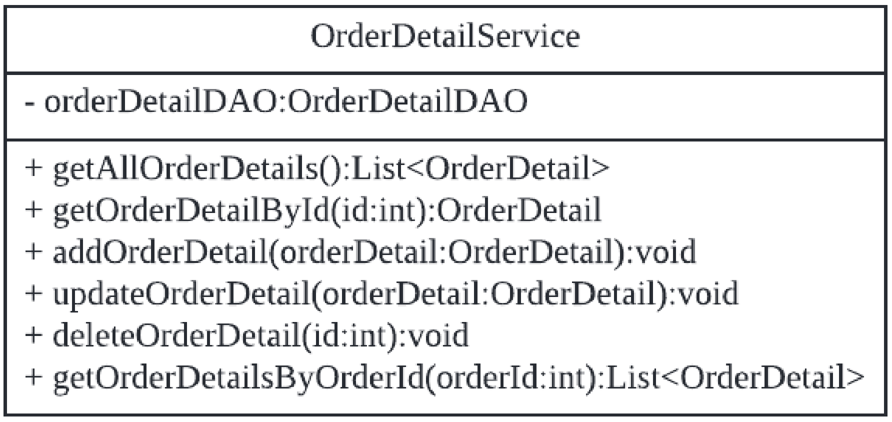

UserService
- userDAO:UserDAO
+ getAllUsers():List<User>
+ getUserById(id:int):User
+ registerUser(user:User):boolean
+ updateUser(user:User):void
+ deleteUser(id:int):void
+ authenticate(username:String,password:String):boolean
+ getUserByUsername(username:String):User

Figure 34 - UserService Class



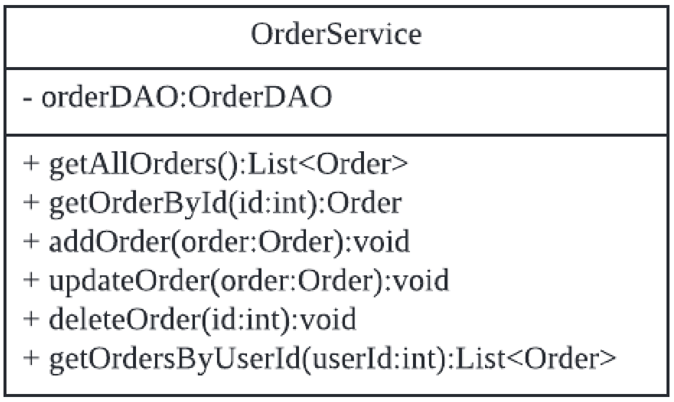
```
ShoppingCartService
- shoppingCartDAO:ShoppingCartDAO
+ getAllShoppingCartEntries():List<ShoppingCart>
+ getShoppingCartEntry(cartId:int):ShoppingCart
+ addShoppingCartEntry(shoppingCart:ShoppingCart):void
+ updateShoppingCartEntry(shoppingCart:ShoppingCart):void
+ deleteShoppingCartEntry(cartId:int):void
+ getShoppingCartEntriesByUser(userId:int):List<ShoppingCart>
+ getShoppingCartEntriesByMovie(movieId:int):List<ShoppingCart>
+ getShoppingCartByUserAndMovie(userId:int,movieId:int):ShoppingCart
```

Figure 35 - ShoppingCartService Class



```
OrderDetailService
- orderDetailDAO:OrderDetailDAO
+ getAllOrderDetails():List<OrderDetail>
+ getOrderDetailById(id:int):OrderDetail
+ addOrderDetail(orderDetail:OrderDetail):void
+ updateOrderDetail(orderDetail:OrderDetail):void
+ deleteOrderDetail(id:int):void
+ getOrderDetailsByOrderId(orderId:int):List<OrderDetail>
```

Figure 36 - OrderDetailService Class



```
OrderService
- orderDAO:OrderDAO
+ getAllOrders():List<Order>
+ getOrderById(id:int):Order
+ addOrder(order:Order):void
+ updateOrder(order:Order):void
+ deleteOrder(id:int):void
+ getOrdersByUserId(userId:int):List<Order>
```

Figure 37 - OrderService Class

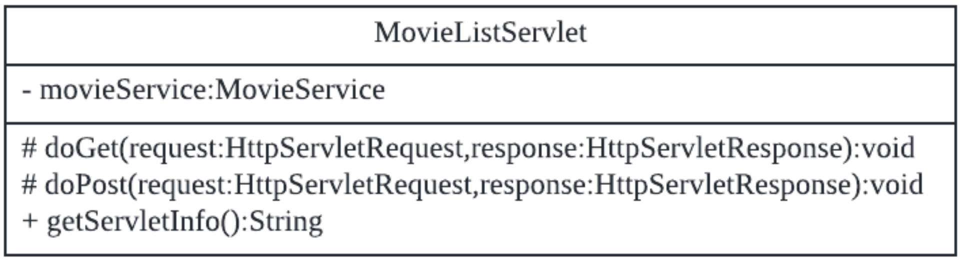

MovieListServlet
- movieService:MovieService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void
# doPost(request:HttpServletRequest,response:HttpServletResponse):void
+ getServletInfo():String

Figure 38 - MovieListServlet Class

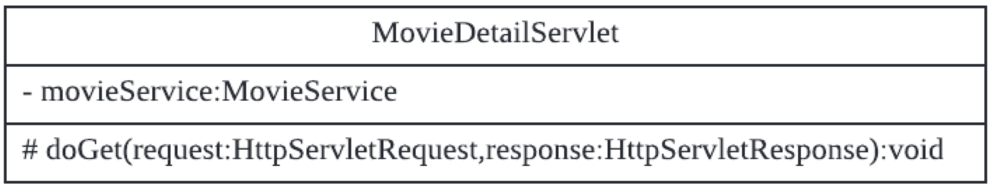

MovieDetailServlet
- movieService:MovieService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void

Figure 39 - MovieDetailServlet Class

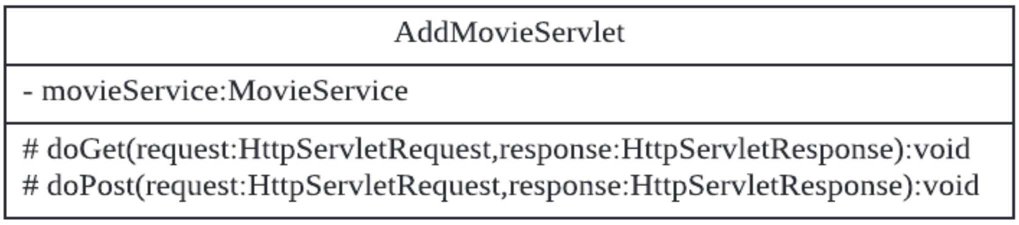

AddMovieServlet
- movieService:MovieService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void
# doPost(request:HttpServletRequest,response:HttpServletResponse):void

Figure 40 - AddMovieServlet Class

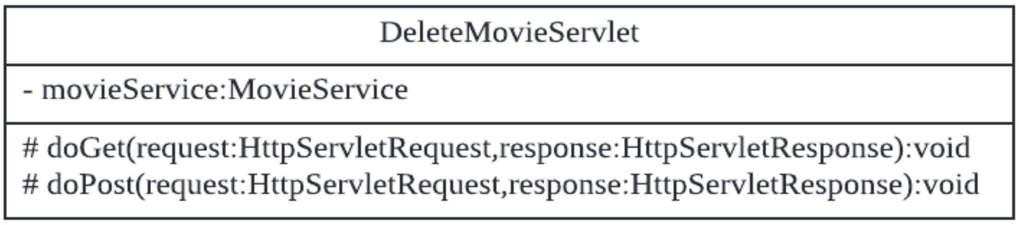

DeleteMovieServlet
- movieService:MovieService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void
# doPost(request:HttpServletRequest,response:HttpServletResponse):void

Figure 41 - DeleteMovieServlet Class

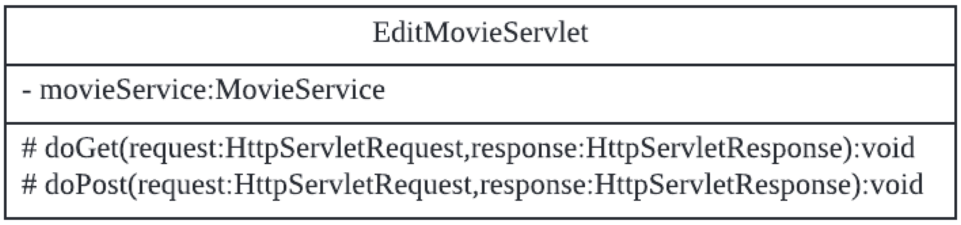

EditMovieServlet
- movieService:MovieService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void
# doPost(request:HttpServletRequest,response:HttpServletResponse):void

Figure 42 - EditMovieServlet Class

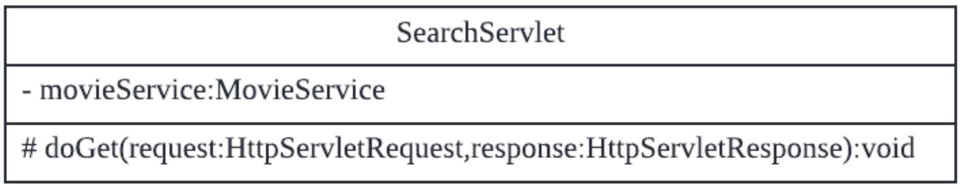

SearchServlet
- movieService:MovieService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void

Figure 43 - SearchServlet Class

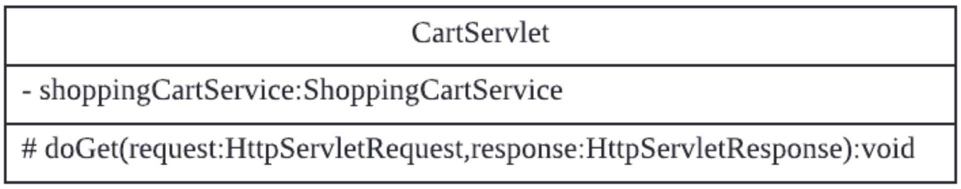

CartServlet
- shoppingCartService:ShoppingCartService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void

Figure 44 - CartServlet Class

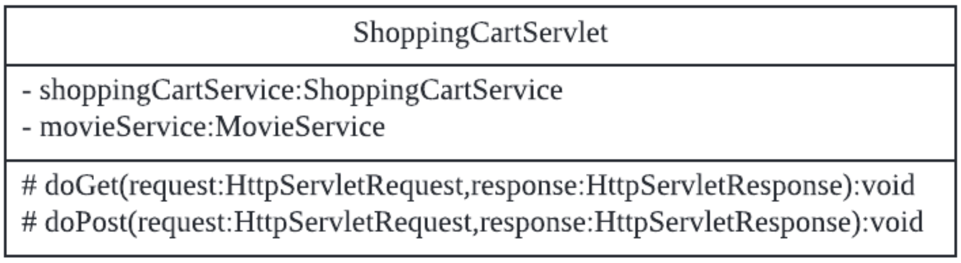

ShoppingCartServlet
- shoppingCartService:ShoppingCartService
- movieService:MovieService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void
# doPost(request:HttpServletRequest,response:HttpServletResponse):void

Figure 45 - ShoppingCartServlet Class

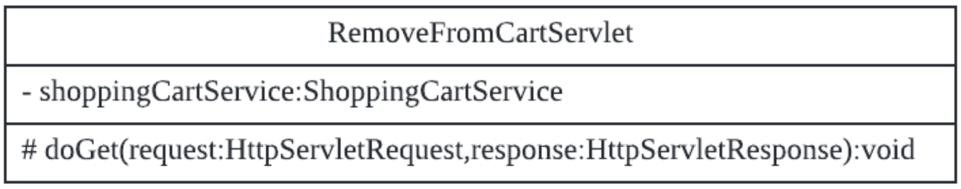

RemoveFromCartServlet
- shoppingCartService:ShoppingCartService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void

Figure 46 - RemoveFromCartServlet Class

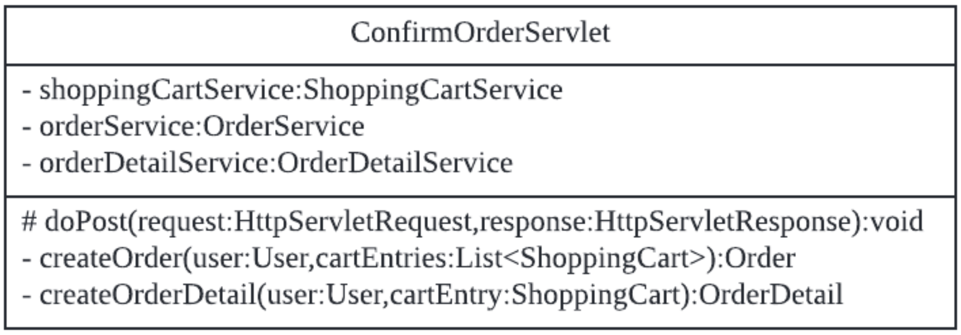

ConfirmOrderServlet
- shoppingCartService:ShoppingCartService
- orderService:OrderService
- orderDetailService:OrderDetailService
# doPost(request:HttpServletRequest,response:HttpServletResponse):void
- createOrder(user:User,cartEntries:List<ShoppingCart>):Order
- createOrderDetail(user:User,cartEntry:ShoppingCart):OrderDetail

Figure 47 - ComfirmOrderServlet Class

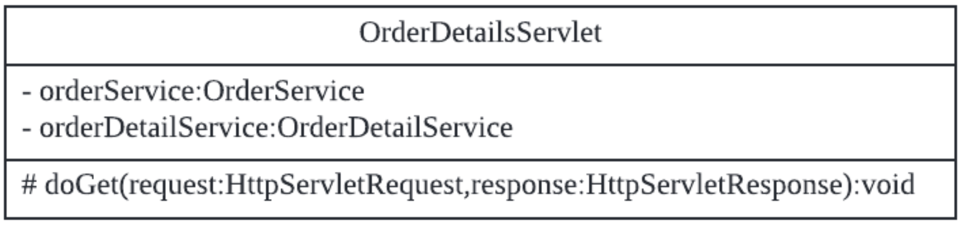

OrderDetailsServlet
- orderService:OrderService
- orderDetailService:OrderDetailService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void

Figure 48 - OrderDetailServlet Class

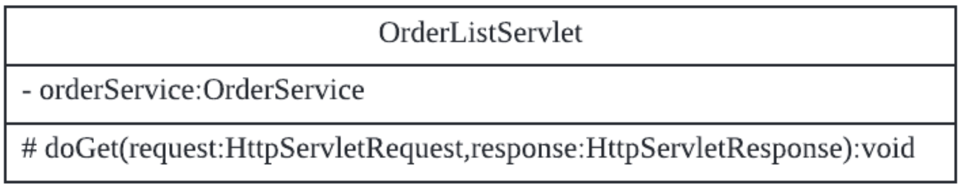

OrderListServlet
- orderService:OrderService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void

Figure 49 - OrderListServlet Class

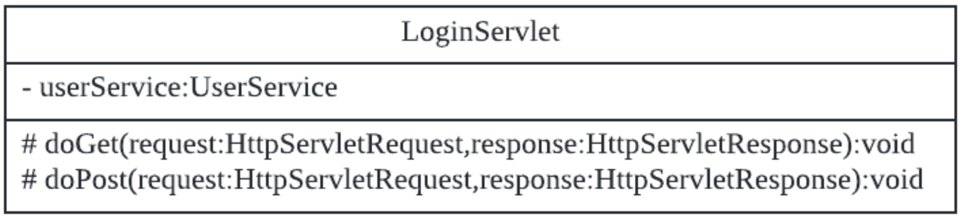

LoginServlet
- userService:UserService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void
# doPost(request:HttpServletRequest,response:HttpServletResponse):void

Figure 50 - LoginServlet Class

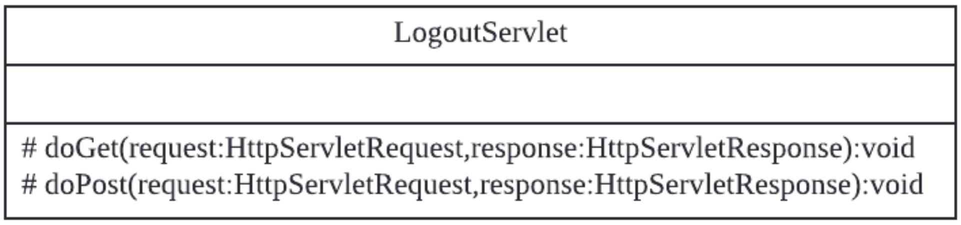

LogoutServlet
# doGet(request:HttpServletRequest,response:HttpServletResponse):void
# doPost(request:HttpServletRequest,response:HttpServletResponse):void

Figure 51 - LogoutServlet Class

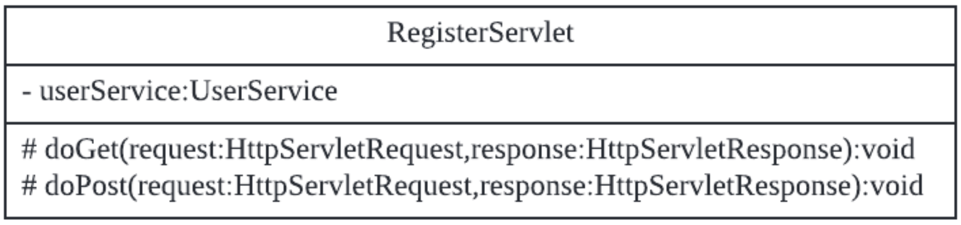

RegisterServlet
- userService:UserService
# doGet(request:HttpServletRequest,response:HttpServletResponse):void
# doPost(request:HttpServletRequest,response:HttpServletResponse):void

Figure 52 - RegisterServlet Class

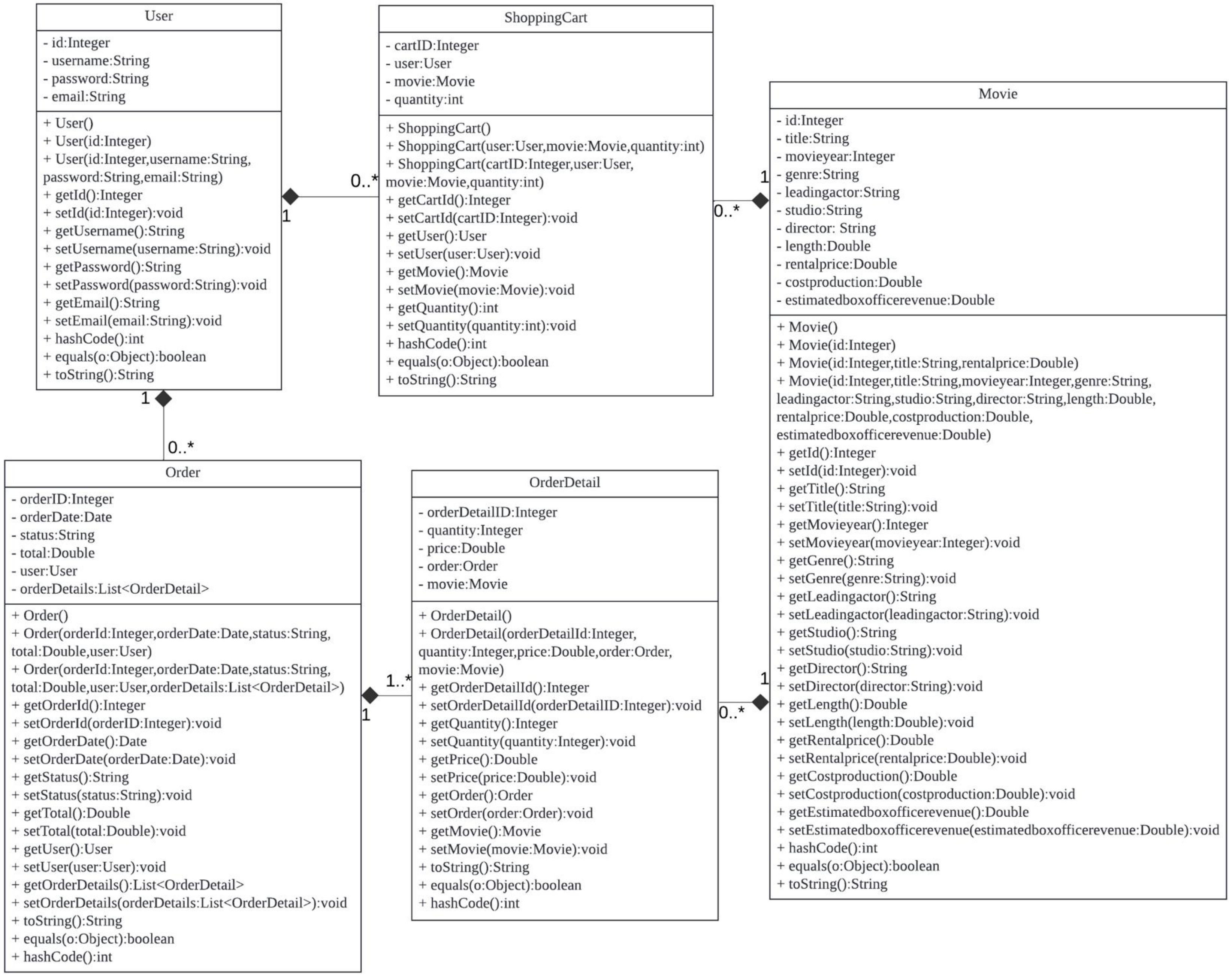

User
- id:Integer
- username:String
- password:String
- email:String
+ User()
+ User(id:Integer)
+ User(id:Integer,username:String, password:String,email:String)
+ getId():Integer
+ setId(id:Integer):void
+ getUsername():String
+ setUsername(username:String):void
+ getPassword():String
+ setPassword(password:String):void
+ getEmail():String
+ setEmail(email:String):void
+ hashCode():int
+ equals(o:Object):boolean
+ toString():String
ShoppingCart
- cartID:Integer
- user:User
- movie:Movie
- quantity:int
+ ShoppingCart()
+ ShoppingCart(user:User,movie:Movie,quantity:int)
+ ShoppingCart(cartID:Integer,user:User, movie:Movie,quantity:int)
+ getCartId():Integer
+ setCartId(cartID:Integer):void
+ getUser():User
+ setUser(user:User):void
+ getMovie():Movie
+ setMovie(movie:Movie):void
+ getQuantity():int
+ setQuantity(quantity:int):void
+ hashCode():int
+ equals(o:Object):boolean
+ toString():String
Movie
- id:Integer
- title:String
- movieyear:Integer
- genre:String
- leadingactor:String
- studio:String
- director: String
- length:Double
- rentalprice:Double
- costproduction:Double
- estimatedboxofficerevenue:Double
+ Movie()
+ Movie(id:Integer)
+ Movie(id:Integer,title:String,rentalprice:Double)
+ Movie(id:Integer,title:String,movieyear:Integer,genre:String, leadingactor:String,studio:String,director:String,length:Double, rentalprice:Double,costproduction:Double, estimatedboxofficerevenue:Double)
+ getId():Integer
+ setId(id:Integer):void
+ getTitle():String
+ setTitle(title:String):void
+ getMovieyear():Integer
+ setMovieyear(movieyear:Integer):void
+ getGenre():String
+ setGenre(genre:String):void
+ getLeadingactor():String
+ setLeadingactor(leadingactor:String):void
+ getStudio():String
+ setStudio(studio:String):void
+ getDirector():String
+ setDirector(director:String):void
+ getLength():Double
+ setLength(length:Double):void
+ getRentalprice():Double
+ setRentalprice(rentalprice:Double):void
+ getCostproduction():Double
+ setCostproduction(costproduction:Double):void
+ getEstimatedboxofficerevenue():Double
+ setEstimatedboxofficerevenue(estimatedboxofficerevenue:Double):void
+ hashCode():int
+ equals(o:Object):boolean
+ toString():String
Order
- orderID:Integer
- orderDate:Date
- status:String
- total:Double
- user:User
- orderDetails:List<OrderDetail>
+ Order()
+ Order(orderId:Integer,orderDate:Date,status:String, total:Double,user:User)
+ Order(orderId:Integer,orderDate:Date,status:String, total:Double,user:User,orderDetails:List<OrderDetail>)
+ getOrderId():Integer
+ setOrderId(orderID:Integer):void
+ getOrderDate():Date
+ setOrderDate(orderDate:Date):void
+ getStatus():String
+ setStatus(status:String):void
+ getTotal():Double
+ setTotal(total:Double):void
+ getUser():User
+ setUser(user:User):void
+ getOrderDetails():List<OrderDetail>
+ setOrderDetails(orderDetails:List<OrderDetail>):void
+ toString():String
+ equals(o:Object):boolean
+ hashCode():int
OrderDetail
- orderDetailID:Integer
- quantity:Integer
- price:Double
- order:Order
- movie:Movie
+ OrderDetail()
+ OrderDetail(orderDetailId:Integer, quantity:Integer,price:Double,order:Order, movie:Movie)
+ getOrderDetailId():Integer
+ setOrderDetailId(orderDetailID:Integer):void
+ getQuantity():Integer
+ setQuantity(quantity:Integer):void
+ getPrice():Double
+ setPrice(price:Double):void
+ getOrder():Order
+ setOrder(order:Order):void
+ getMovie():Movie
+ setMovie(movie:Movie):void
+ toString():String
+ equals(o:Object):boolean
+ hashCode():int
1
0..*
1
0..*
1
0..*
1
1..*
1
0..*

Figure 53 - UML Class Diagram