Product Requirements Document

Submission 2

TeamWZ

SWEN90007 SM2 2023 Project

Student Name	Student ID	UniMelb Username	Github Username	Email
HanXiang Wang	1343151	HANXIANGW	Hanxiang-WANG	hanxiangw@student.unimelb.edu.au
Kehan Zhang	1331979	KEHZHANG	KehanZhang0712	kehzhang@student.unimelb.edu.au
Xiaoran Zhang	1253830	XIAORANZ3	xiaoranz1	xiaoranz3@student.unimelb.edu.au
Zhiyao Wang	1290370	ZHIYAOW1	ZhiyaoW	zhiyaow1@student.unimelb.edu.au

THE UNIVERSITY OF MELBOURNE

SCHOOL OF COMPUTING & INFORMATION SYSTEMS

Revision History

Date	Version	Description	Author
22/8/2023	03.00	Identify field, foreign key mapping	Kehan Zhang
24/8/2023	03.01	Adjust the identify field, foreign key mapping, and add association table mapping, embedded value	Kehan Zhang
26/8/2023	03.02	Description of identify field, foreign key mapping, and add association table mapping, embedded value	Xiaoran Zhang
30/8/2023	03.10	Lazy load and unit of work	Zhiyao Wang Hanxiang Wang
31/8/2023	03.11	Data mapper	Hanxiang Wang
1/9/2023	03.12	Add sequence diagram and description of data mapper	Xiaoran Zhang
3/9/2023	03.20	Adjust lazy load, unit of work and inheritance	Zhiyao Wang Hanxiang Wang
4/9/2023	03.21	Adjust unit of work	Kehan Zhang
6/9/2023	03.22	Description of lazy load and unit of work, adjust the Json data.	Xiaoran Zhang
10/9/2023	03.30	Update front-end page	Zhiyao Wang
10/9/2023	03.31	Update back-end controller layer	Hanxiang Wang
12/9/2023	03.40	Finished the Authentication and Authorization	Hanxiang Wang
14/9/2023	04.00	Integration of frond-end and backend	Zhiyao Wang Hanxiang Wang
15/9/2023	04.00	Integration of frond-end and backend	Zhiyao Wang Hanxiang Wang Kehan Zhang XiaoranZhang
17/9/2023	05.00	Add class diagram and update report	Xiaoran Zhang
18/9/2023	06.00	Debug demo	Zhiyao Wang Hanxiang Wang Kehan Zhang

Contents

1. Introduction 4
1.1 Proposal 4
1.2 Target Users 4
1.3 Conventions, terms and abbreviations 4
2. Actors 5
3. Use Cases 6
3.1 Use Case descriptions 6
3.2 Use Case Diagram 10
4. Domain Model 11
4.1 Domain Model Description 11
4.2 Domain Model Diagram 11
5. Architectural Patterns 12
5.1 Overview 12
5.2 Data Mapper 12
5.3 Unit of work 18
5.4 Lazy load 18
5.5 Identity field 19
5.6 Foreign key mapping 20
5.7 Association table mapping 21
5.8 Embedded value 21
5.9 Inheritance 21
5.10 Authentication and Authorization 22
6. Class Diagram 24
7. Source Code Directories Structure 26
8. Libraries and Frameworks 27

1. Introduction

1.1 Proposal

This document has updated the use cases based on feedback from Part 1. Also, this document also specifies a series of patterns, including Data mapper, Unit of work, Lazy load, Identity field, Foreign key mapping, Association table mapping, Embedded value, inheritance patterns, Authentication and Authorization.

1.2 Target Users

This document is mainly intended for SWEN90007 students and the teaching team.

1.3 Conventions, terms and abbreviations

This section explains the concept of some important terms that will be used throughout this document. These terms are detailed alphabetically in the following table.

Term	Description
Administrator	Special user who responsible for managing the music event system
Customer	Users who use the website to book tickets
Event Planner	Users who create music events (price of ticket and venue)
Tickets	The corresponding performance tickets for each music event.
Venue	The corresponding performance location for each music event.
Bookings	Customers book concert tickets on the music event platform.
Events	Music events that can be found on this platform.

2. Actors

Actor	**Description**
Customer	Customers who use the app to book tickets, etc
Event Planner	Event planners who create events
Administrator	Administrators who manage the system

3. Use Cases

3.1 Use Case descriptions

Use Case 1: Search Music Acts

Actors

1. Customer

Basic Flow

The customer logs into the online event booking application. After logging in, they access the search feature. They enter the name of the music act they are looking for. The system searches and displays relevant results matching the customer's query.

Use Case 2: View Calendar of Events

Actors

1. Customer

Basic Flow

Upon accessing the online event booking application, the customer chooses the calendar option. The system displays a calendar populated with upcoming music events for the next six months.\

Use Case 3: Book Tickets

Actors

1. Customer

Basic Flow

The customer logs into the online event booking application. After browsing, they select a desired music event. They then specify the number of tickets they wish to book, choosing the seat type for each ticket. Once selected, the system reserves the tickets for the customer.

Use Case 4: View Bookings

Actors

1. Administrators
2. Customer
3. Event Planner

Basic Flow

After logging into the online event booking application, the actor (either a customer or an event planner) chooses the option to view their bookings. The system then displays a list of bookings associated with the logged-in actor.The administrator can also view all the event bookings, displaying details of customers who have purchased tickets, and the associated ticket details.

Use Case 5: Cancel Booking

Actors

1. Customer
2. Event Planner

Basic Flow

Upon logging into the application, the actor selects a booking they wish to cancel. They then select the option to cancel this booking. The system processes the request and cancels the selected booking.

Use Case 6: Manage Application

Actors

1. Administrator

Basic Flow

The administrator logs in to the online event booking application. Once entered, they can access application settings and configurations, including event details, booking rules, and more. Administrators make the necessary changes and updates. Then the system saves and applies these changes.

Use Case 7: Create Venue

Actors

1. Administrator

Basic Flow

The administrator logs into the online event booking system. They navigate to the venue creation section and input the required details for the new venue, including specifying sections and capacities. The system then saves the new venue information.

Use Case 8: Create Event

Actors

1. Event Planner

Basic Flow

The event planner logs into the online event booking system. They select the option to create a new musical event, providing necessary details like venue, date, and time. Once all details are provided, the system adds the new event.

Use Case 9: Modify Event

Actors

1. Event Planner

Basic Flow

The event planner logs into the system. They select an existing event they wish to modify. The planner updates the event details as needed. The system then saves these updated details.

Use Case 10: Set Ticket Price

Actors

1. Event Planner

Basic Flow

Upon logging into the system, the event planner selects an event and its specific seat type. They then set a price for tickets based on the chosen seat type and musical artist. The system then updates this ticket price.

Use Case 11: Administrator Views Users

Actors

1. Administrator

Basic Flow

The administrator logs into the online event booking application. They navigate to a section that allows them to view all registered users of the system.

Use Case 12: Event Planners Association with Event

Actors

1. Event Planner

Basic Flow

Event planners log into the online event booking system. Within the system, an event can be associated with one or more event planners. Planners can add themselves or be added by other planners to events, granting them the ability to manage and modify the associated event.

Use Case 13: View Venue Sections and Capacity

Actors

1. Administrator

Basic Flow

The administrator logs into the system. They navigate to the venue section, where they can view details of all venues, including their sections (mosh, standing, seated, VIP, etc.). The system also displays the capacity for each section.

Use Case 14: Sign up

Actors

1.Event Planner

2.Customer

Basic Flow

When the users visit the system for the first time, they need to sign in, if they don't have the account, they can click the sign-up button to sign up with some personal information. And different roles will get different permissions.

Use Case 15: Sign in

Actors

1.Administrator

2.Event Planner

3.Customer

Basic Flow

When the users visit the system for the first time, they can sign in with their account and password. If they forget the password, they can click the button to change it with their email or phone verification. When they sign in, they will get different permissions depending on their roles.

Use Case 16: Manage profile

Actors

1.Administrator

2.Event Planner

3.Customer

Basic Flow

When the users successfully sign in, they can manage their profile, including their history, personal information, account details, payment methods, etc.

3.2 Use Case Diagram

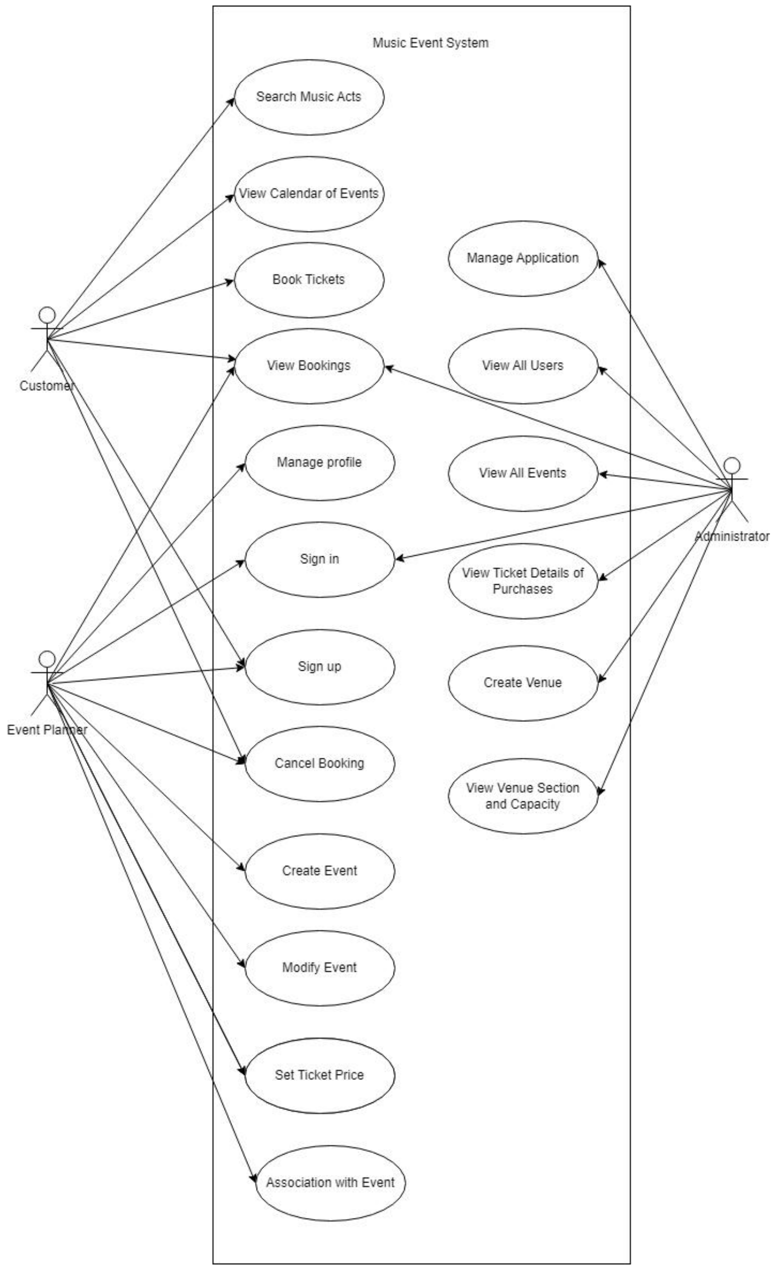

Figure3.1 use case diagram

4. Domain Model

4.1 Domain Model Description

Based on the specifications provided for the Music Event System, the system entities, attributes, and business rules can be summarized as:

- **Admin** can check the information of **Event**, **Reservation**, **Ticket**, **EventPlanner** and **Customer**;
- **Admin** can operate the **Venue** but cannot create **Event**;
- There is *only one* **Admin**;
- **Venue** has different parts which are: **Mosh**, **Standing**, **Seated** and **VIP**;
- *One or more* **EventPlanner** can create *one* **Event**;
- **Event** must be held at *proper* time, date, and **Venue**;
- **Customer** can search **Event** through eventName or eventDate;
- *One* **Customer** can reserve *one or more* **Ticket** in *one* transaction;
- **EventPlanner** can modify or cancel **Event**;
- **EventPlanner** can modify **Ticket** information such as price;
- Both **Customer** and **EventPlanner** can check or cancel **Reservation**

Entities have been bolded; attributes have been underlined; and important *associations* have been italicized.

4.2 Domain Model Diagram

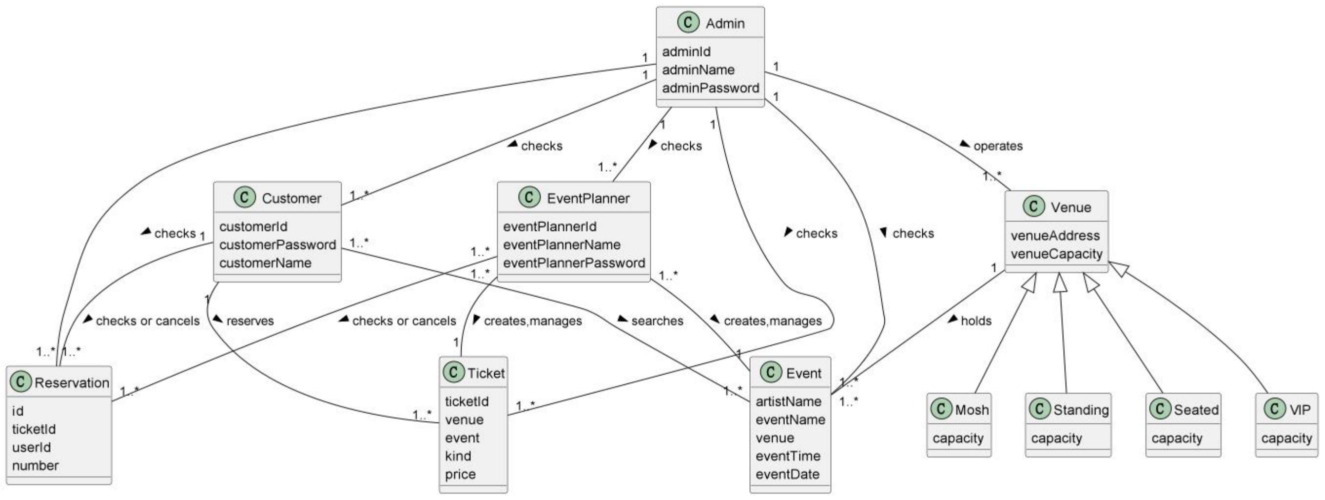

Figure4.1 Domain model diagram

5. Architectural Patterns

5.1 Overview

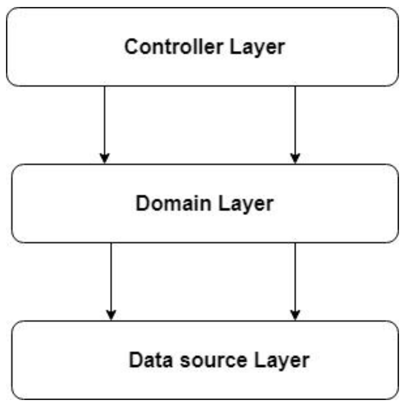

Figure 5.1 Overview layer of Music Event System

The Music Event System makes use of three layers: controller layer, domain layer and data source layer.The control layer is mainly responsible for controlling the overall behavior of the user interface and application, receiving user input or external requests, and routing them to appropriate domain layer components, as well as managing the business processes and control logic of the application. For the domain layer, it includes core business logic and domain objects, representing the business concepts and entities in the system, handling business rules, logic, and data processing, as well as being responsible for data validation, transformation, and persistence, and interacting with the data source layer to obtain or update data. Finally, the data source layer is responsible for communicating with data storage and external data sources.

5.2 Data Mapper

Data mapper is a software design pattern typically used to separate the data storage layer of an application from its business logic layer for better maintainability and scalability. Its main goal is to abstract data persistence details from application code by establishing a mapping between application objects and databases.

The data source of this system consists of five mappers, namely EventMapper, ReservationMapper, TicketMapper, UseMapper and VenueMapper. These five types of mappers all include methods for inserting, querying, updating, and deleting data, typically using preprocessed statements to perform SQL operations. For example, the insertEvent method in EventMapper uses preprocessing statements to insert user related information into the database, such as username, email, password, and permissions. Then, it returns the number of inserted rows (shown in Figure 5.3).

Moreover, these five types of mappers are all used to map data from databases to Java objects for application operations. All Mapper classes in this system include mechanisms for handling database operation errors, typically capturing and handling SQLExceptions or other related exceptions. It also includes methods for obtaining data sets, such as methods for obtaining all users, venues, events, etc.

These Mapper classes may have some special operations or characteristics in their specific implementation. Firstly, the events in the EventMapper are associated with the EventPlanner, such as inserting events and updating and deleting events. Secondly, the ReservationMapper class is used to manage ticket reservations, including inserting, querying, and deleting reservation information. This is different from other Mapper classes because it handles specific business areas (ticketing booking). The third feature is user permission and type management in UserMapper, such as verifying user login based on username and password, and searching for users based on user type (such as regular user, administrator, activity planner). Finally, the VenueMapper class is used to manage venue information, including inserting and retrieving venue information. The uniqueness lies in its involvement in the capacity of venues and different types of capacity, such as rock and roll areas, standing areas, seating areas, and VIP areas.

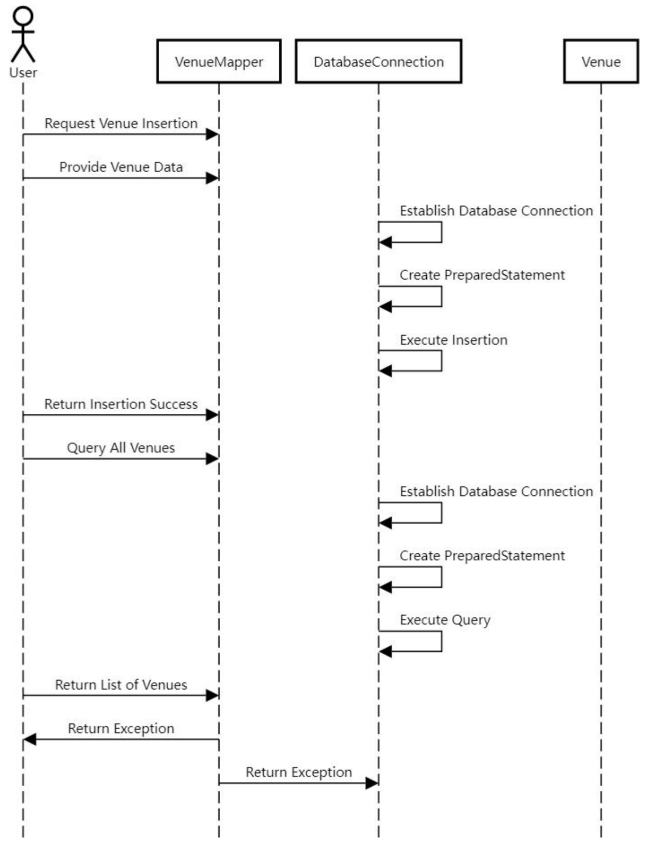

Figure 5.2 VenueMapper Sequence Diagram

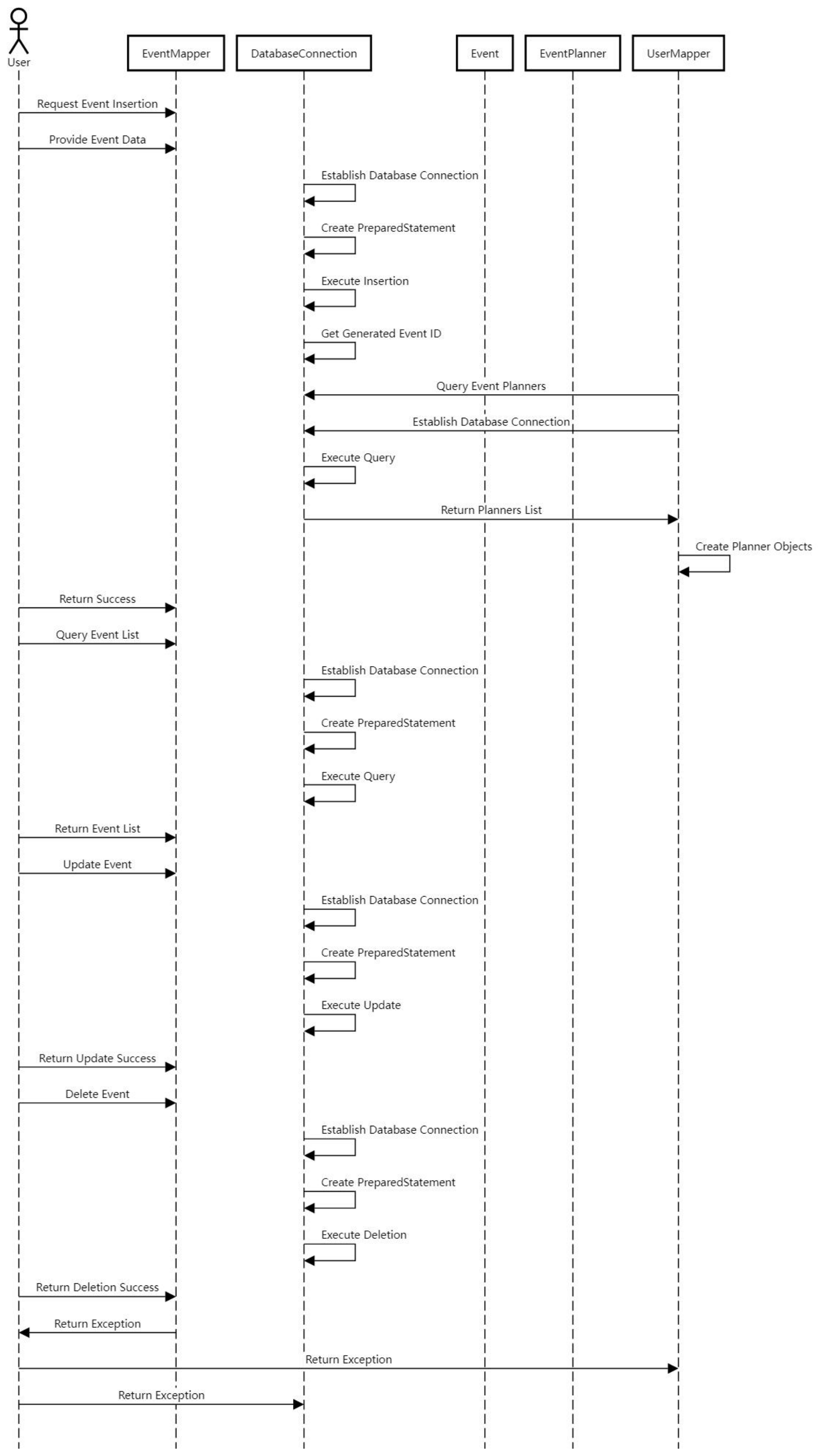

Figure 5.3 EventMapper Sequence Diagram

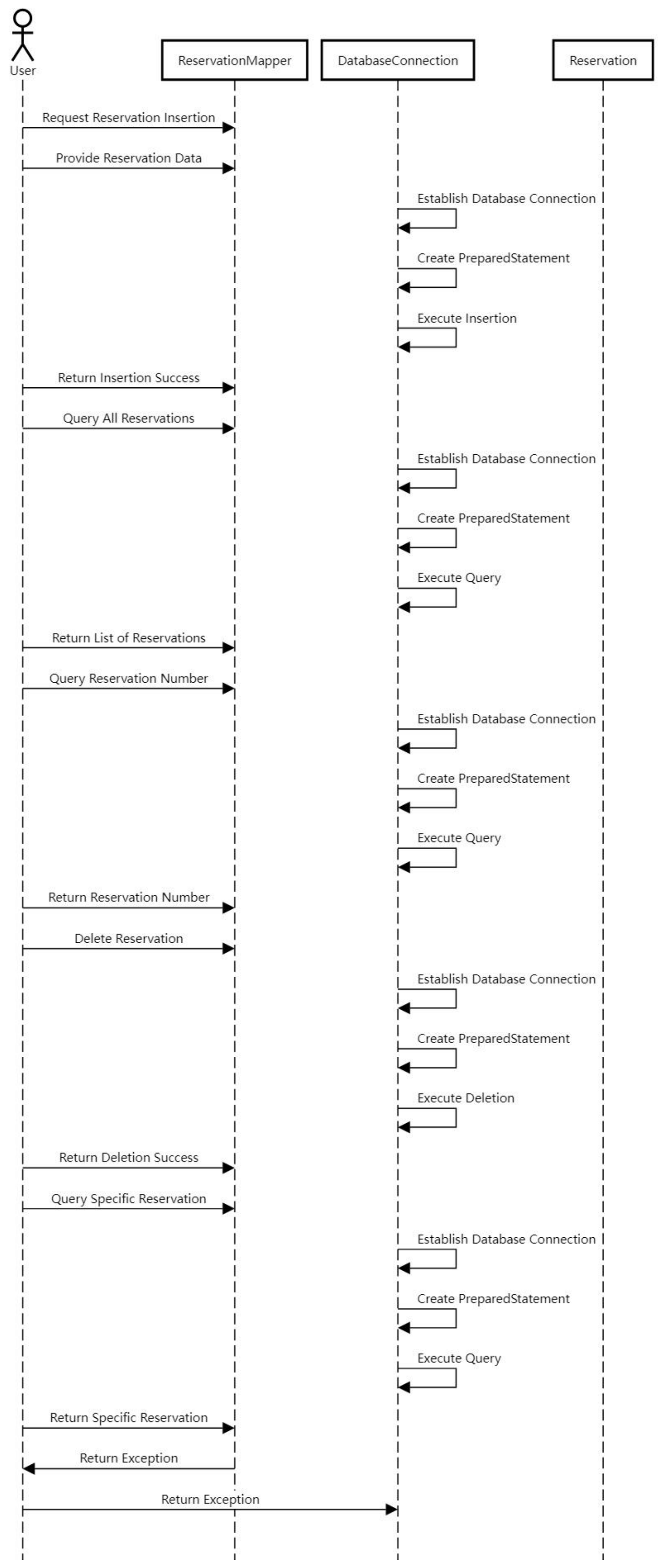

Figure 5.4 ReservationMapper Sequence Diagram

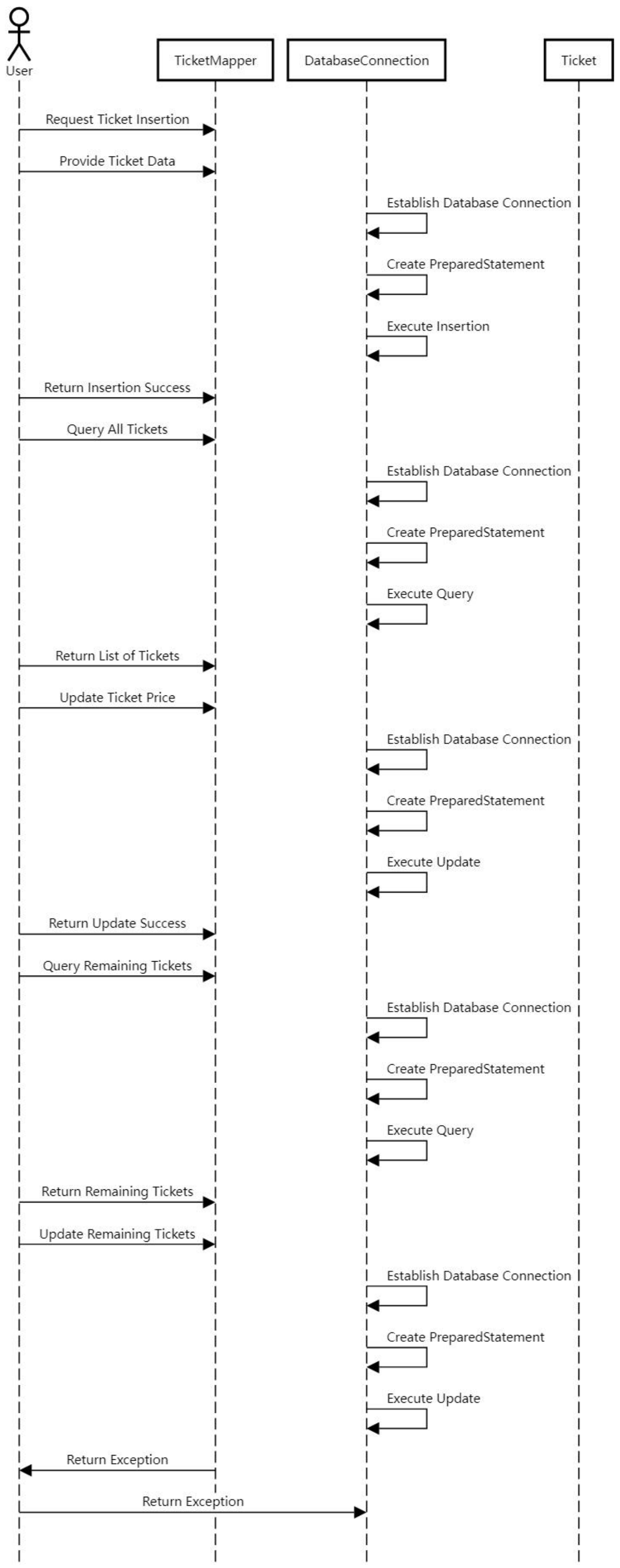

Figure 5.5 TicketMapper Sequence Diagram

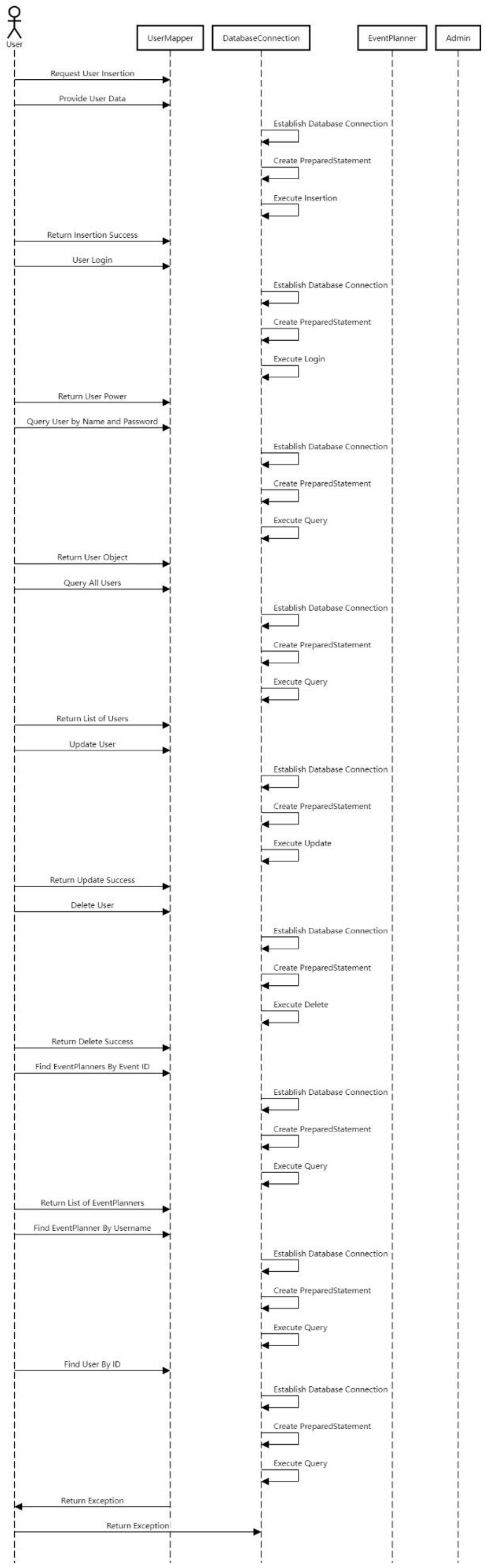

Figure 5.6 UserMapper Sequence Diagram

5.3 Unit of work

Unit of work is used to manage and execute a series of related operations or tasks, usually within the context of a transaction. In backend development, it helps to ensure that a set of related database operations are either successfully executed or all fail to maintain data consistency.

In this system, a unit of work is used to maintain database consistency. This is very useful for handling complex data operations and ensuring data integrity, including a static current variable in the UnitOfWork class, which is a ThreadLocal object. That means that each thread has its own independent UnitOfWork instance, which ensures that each thread can have its own unit of work in a multi-threaded environment. Moreover, there are three list variables: newObjects, dirtyObjects, and deletedObjects, which are lists used to track objects in different states. NewObjects is used to track newly created objects, dirtyObjects is used to track changes but not yet saved objects, and deletedObjects is used to track deleted objects.Also, the system created registerNew, registerDirty, and registerDeleted methods in the UnitOfWork class. These methods are used to register objects in the corresponding list and process them in the commit method. Among them, registerNew is used for new objects, registerDirty is used for dirty objects, and registerDeleted is used for deleted objects.

The most core method in this class is the commit method, which can start database transactions, such as inserting new objects into the database, updating changed but not yet saved objects, deleting deleted objects from the database, and committing database transactions. Specifically, if an exception occurs in the commit method, it will roll back the transaction to ensure data consistency. Finally, regardless of whether an exception occurs, the list of all objects will be cleared, the database transaction will end, and the automatic commit mode of the database connection will be restored to the default value.

5.4 Lazy load

Lazy load is a performance optimization technique used in web pages and applications, which is used to delay the loading of resources, such as images, until needed by the user or visible in the browser viewport. Its purpose is to reduce the number of resources that need to be downloaded and processed during initial page loading, thereby accelerating page loading speed and reducing the demand for user bandwidth and device performance.

There are many ways to lazy load, such as lazy loading of content, lazy loading of text, lazy loading of content, and so on. The final decision of this system is to choose lazy loading of content. Because if image lazy loading and text lazy loading are used, if the delay loading time is set too long, users may see blank or incomplete content, which will reduce user satisfaction. The lazy loading of content used in this system involves dividing the page content into two stages of loading. Firstly, only basic content (such as event names) was loaded during initial loading, and then when requested by the user, the "View Details" button was clicked to load and display detailed information. This method helps to reduce initial loading time, improve page performance, and provide a smoother user experience.

5.5 Identity field

Identity field is a pattern in database design that refers to the use of specific column attributes in relational database tables to ensure that each row of data has a unique and incremental identifier to help manage and maintain data and ensure data integrity. This mode allows the database to automatically assign unique values without the need for manual user intervention. The purpose of this pattern is to facilitate the unique identification of each row in the data table to support operations such as data retrieval, update, and deletion while ensuring the uniqueness of the data.

For this system, based on the uniqueness, incrementality, and automatic allocation characteristics of Identity Field Patterns, our team has decided to use self-increasing fields to implement Identity Field Patterns, so that the IDs of each table are self-increasing fields, such as event_ id, association_ id, ticket_ id, user_ id, revenue_ id. Because auto increment fields ensure that each row has a unique identifier, there is no need for manual intervention. Using self-increasing fields to implement Identity Field Pattern can also make inserting new records easier. So, they don't need to explicitly specify the identifier for each new record, the database will automatically handle this. In summary, using self-increasing fields as an implementation of Identity Field Patterns is because they provide a convenient, efficient, and standardized way to assign unique identifiers to each row in a database table. This is very helpful in Identity Field Pattern for database design and data management.

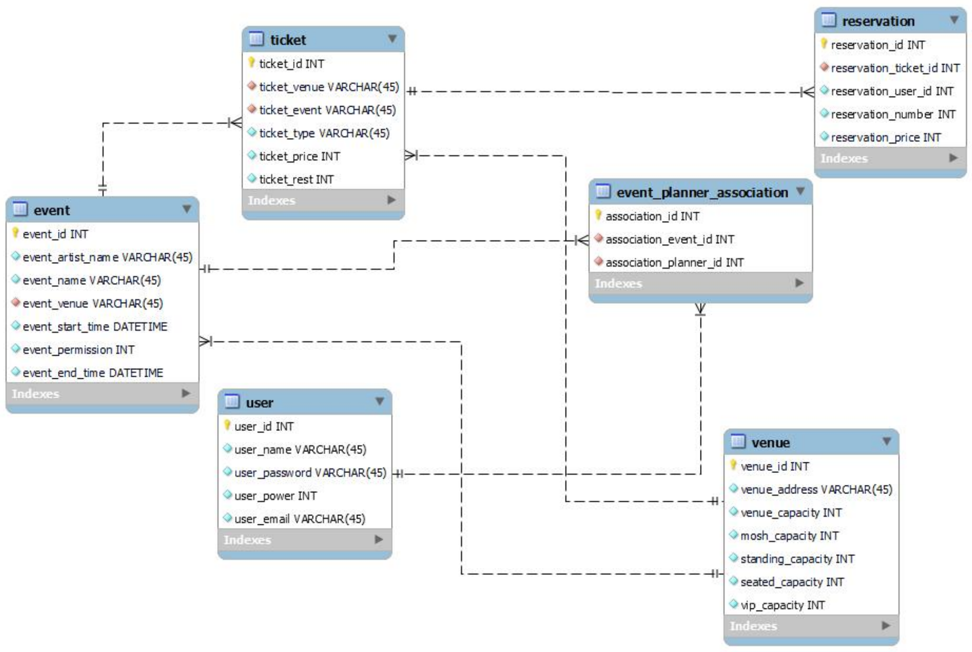

Figure5.7 Entity relationship (ERD)

5.6 Foreign key mapping

Foreign key mapping is the process of establishing relationship mappings between multiple tables in the database to organize and retrieve data more effectively. Foreign key mapping is usually used to define dependencies between tables, typically involving the relationship between columns in one table and columns in another table.

For this system, we have six tables. Five of the tables have foreign keys set, the ticket table, event table, revenue table, reservation table and event_ planner_ association table. In the reservation table, reservation_ticket_id is foreign key, which is associated with the ticket table with ticket_id, because of that, the records in the reservation table and the ticket table show two relationships, one-to-one and many-to-one. In the ticket table, there are two foreign keys: ticket_event and ticket_venue,which are associated with the event table and venue table respectively. That means that the ticket table has a one-to-one relationship with each of these two tables. The event_venue is a foreign key in the event table, which is associated with the venue table. That means the event table and the ticket table have a one-to-one relationship. There are two foreign keys in the even_planner_association table, namely association_ event_ id and association_ user_id, so that the even_planner_association table has a many-to-many relationship with event table and user table.

5.7 Association table mapping

Association table mapping is a database design and ORM schema used to handle many to many relationships. In databases, association tables are usually used to represent the relationship between two entities, which typically contain two foreign keys pointing to two associated entities.

The system implemented the association table mapping pattern for event_planner_association table to represent the relationship between event and user. The event_planner_association table has two foreign keys, which are association_ event_ id and association_ user_id that points to two entities, event, and user, respectively. Therefore, the event_planner_association table handles many-to-many relationships and simplifies the data model to make data storage and querying easier and more consistent. It also improves code readability and maintainability by clearly mapping database relationships and business logic into the object model.

5.8 Embedded value

Embedded values in databases refer to a pattern that combines multiple related data fields into a single field or data structure to better represent and manage these data. It helps to simplify database schemas, improve data readability, and reduce data redundancy.

In the system, the team implemented an embedded values pattern for the event table and venue table in two different ways.As for event table, the embedded values are used to store the start date and end date of the time (event_start_time and event_end_time), including the year, month, day, hour, minute, and second in these two columns, which ensures consistency and ease of maintenance of this information. Combining these fields into an embedded value object and storing it in an entity table containing all other related data can reduce the number of columns and complexity of the data table.

5.9 Inheritance

Inheritance is an important concept in object-oriented programming in software development. It is a mechanism that allows one class to inherit the properties and methods of another class, which means that subclasses can reuse the code of their parent class without having to write the same code from scratch. This helps improve the reusability and maintainability of the code.

This system has a base class called User, which contains some public properties such as userId, userName, password, power, and email, as well as corresponding getter and setter methods. The team created two derived classes, Admin and EventPlanner, which both inherit from the User base class, which is a manifestation of inheritance, which means that the Admin and EventPlanner classes inherit the properties and methods of the User class without having to redefine them.

In the Admin class, a constructor is defined that accepts userId, userName, password, and email parameters, and uses the super keyword to call the constructor of the parent class User to initialize these properties. This approach allows the Admin class to inherit the properties of the User class and add some administrator specific behaviors. In the EventPlanner class, constructors are also defined, similar to the Admin class. In addition, a private property called createdEvents has been added to represent the list of events created by the event planner, and corresponding methods have been provided to manage this list.

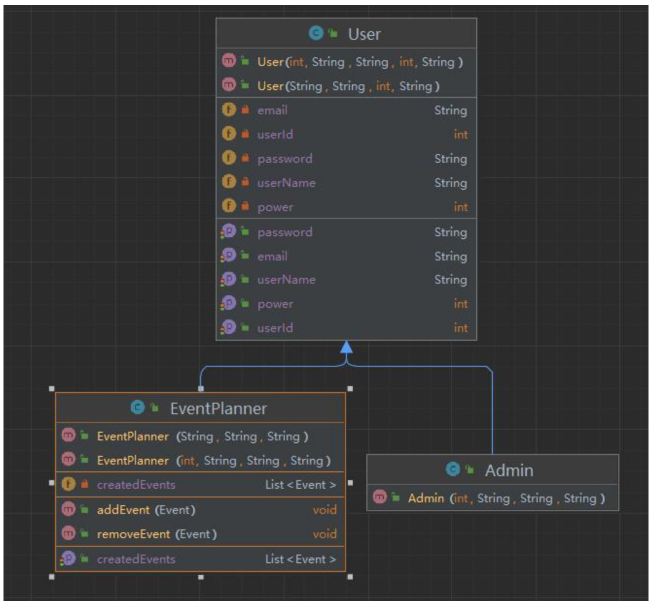
User
User(int, String, String, int, String)
User(String, String, int, String)
email String
userId int
password String
userName String
power int
password String
email String
userName String
power int
userId int
EventPlanner
EventPlanner (String, String, String)
EventPlanner (int, String, String, String)
createdEvents List<Event>
addEvent (Event) void
removeEvent (Event) void
createdEvents List<Event>
Admin
Admin (int, String, String, String)

Figure 5.8 Inheritance

5.10 Authentication and Authorization

Authentication and Authorization are two key concepts in the field of computer security, used to ensure that only suitable users can access system resources and perform specific operations. Authentication is the process of confirming a user's identity, usually completed by verifying the identity credentials or other identity information provided by the user. Authorization is the process of determining which resources and actions an authenticated user or entity can access and perform, ensuring that they can only access the resources they have permission to.

In this system, when a user registers, the 'hashPassword' method is first used to hash encrypt the plaintext password provided by the user. It uses Spring Security's Pbkdf2PasswordEncoder to perform password hash operations. The hashed password can be stored in the database to increase password security. When a user attempts to log in, they first use the 'checkPasswordMatchesHash' method to verify whether the plaintext password provided by the user matches the previously stored hash password in the database. If the match is successful, the user will be considered authenticated. Once the user successfully logs in, the 'getUserPowerFromSession' method will be used to retrieve the user's permission level. For example, 1 represents customer, 2 represents admin, and 3 represents planner. Depending on the user's permission level, they can be allowed or denied access to specific resources or perform specific operations. These methods work together to ensure secure processing of user authentication and authorization

6. Class Diagram

This section will describe the class diagram of the system. This class diagram represents a system consisting of multiple key classes, each of which bears different roles and functions. Including the following seven key categories:

1. Admin: represents the administrator role of the system and has the authority to manage system users.

2. Event Planner: Represents the role of an event planner, who can create and manage various events.

3. Event: Represents an event in the system, including information such as artist, name, venue, start and end times. An event can have multiple event planners.

4. Reservation: represents the information of the reservation, including reservation number, ticket number, user ID, quantity, and price. Associated with users and tickets.

5. Ticket: describes various types of tickets, including venue, event, type, price, and remaining ticket quantity information.

6. User: represents the users of the system, including administrators and event planners. Users have attributes such as username, password, permissions, and email.

7.Venue: Refers to the venue where various activities may occur, including address, capacity, and different types of seats.

These classes have their relationships. Both Admin and EventPlanner inherit from User to obtain the user's common properties and methods. EventPlanner can create and manage multiple events, representing one-to-many relationships. There is a many-to-many relationship between Event and EventPlanner, where an event can be created by multiple event planners, and a single event planner can be responsible for multiple events.Venue relies on Admin, indicating that venue management relies on the authority of the administrator. Reservation relies on User, indicating that the reservation is associated with the user, and user information is a part of the reservation.

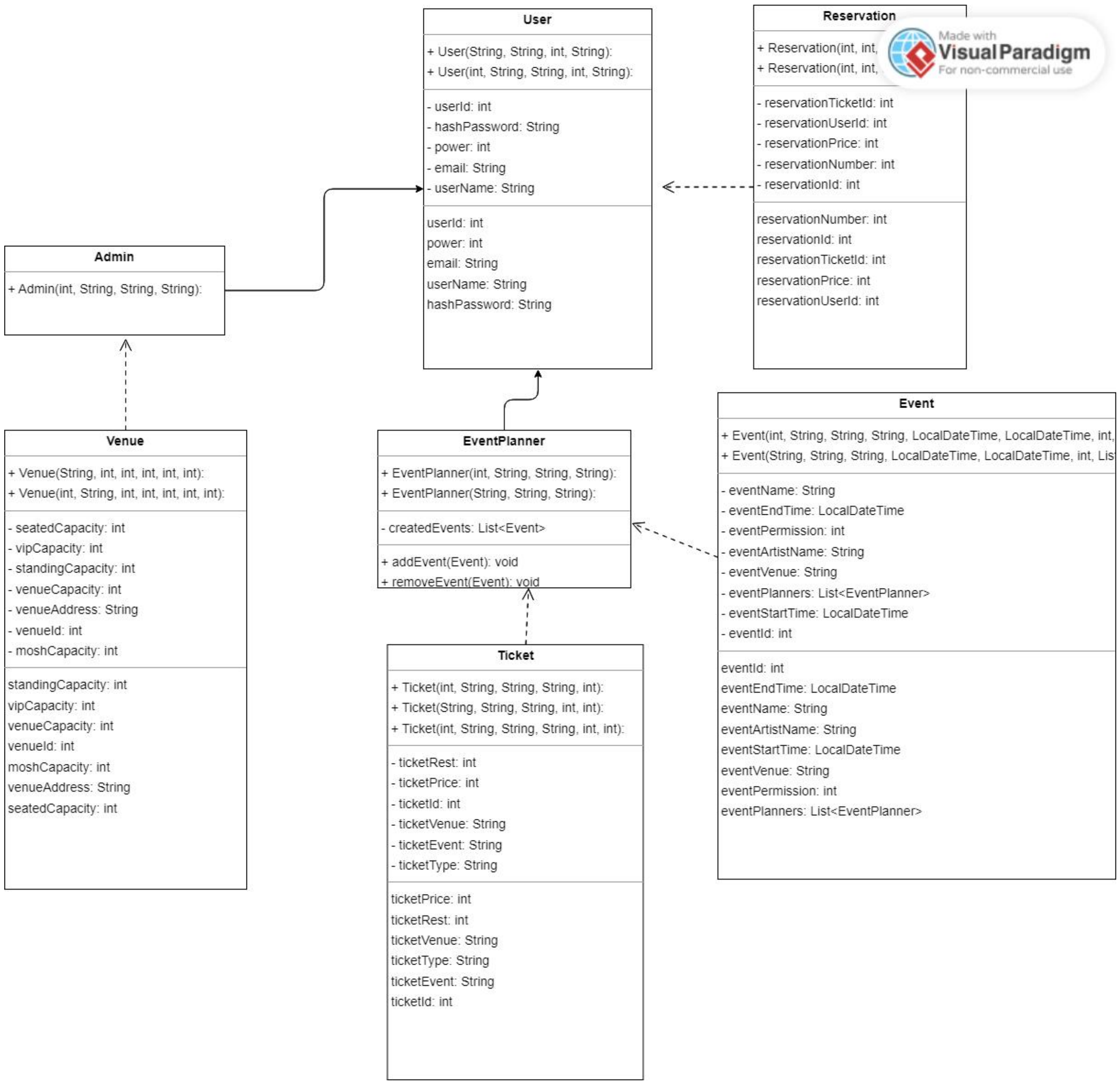

Figure5.9 Class Diagram

7. Source Code Directories Structure

```
service[WZ]
          |--------src
                    |-------- main
                              |--------java
                                        |--------controller
                                                  |--------Controllers
                                        |--------datasource
                                                  |--------Data Mappers
                                        |--------domain
                                                  |-------- Domain Model
                                        |--------util
                                                  |-------- Unit of Work
                                                  |-------- Authentication and Authorization

web
|--------Views
```

8. Libraries and Frameworks

This section describes libraries and *frameworks* used by the Music event System

Library/Framework	**Usage**
Spring Security	Used for password hashing