Architecture and Design Document

"Alfie" Music Events System

Software Design and Architecture
SWEN90007 SM2 2023 Project
Getters Setters

Joel Fressard Kenna - jfkenna@student.unimelb.edu.au - 995401
Sebastian Bobadilla Chara - sbobadillach@student.unimelb.edu.au - 1305851
Georgia Rose Lewis - grlewis@student.unimelb.edu.au - 982172
Anjaney Chirag Mahajan - anjaneychira@student.unimelb.edu.au - 1119668

Professor Eduardo Oliveira & Professor Maria Rodriguez Read
University of Melbourne
Semester 2 2023

Due: September 19th 2023 17:00 AEST

1 Revision History

Date	Version	Description	Author
2023-09-13	1.00-D1	Initial document set-up	Anjaney C Mahajan
2023-09-15	1.01-D1	Completed domain mode, data mapper, identity field, and mapping sections.	Anjaney C Mahajan
2023-09-18	1.02-D1	Copied over event planner use cases from part 1, began updating use cases	Joel Fressard Kenna
2023-09-19	1.03-D1	Completed Event Planner Use Cases 1 to 13	Joel Fressard Kenna
2023-09-19	1.04-D1	Completed pattern descriptions of unit of work, lazy loading, and authentication and authorisation. Also added the design rationale for those sections.	Sebastian Bobadilla Chara
2023-09-19	1.05-D1	Copied over customer use cases from part 1 that have been modified, and made changes	Georgia Rose Lewis
2023-09-20	1.06-D1	Updated Book ticket use case to reflect latest fixes	Joel Fressard Kenna
023-09-20	1.07-D1	Added Admin use cases 1 to 7	Joel Fressard Kenna
2023-09-20	1.08-D1	Added to and updated customer use cases, reviewed and edited the entire report, including pattern descriptions and design rationale	Georgia Rose Lewis

Contents

2 Class Diagrams

The project's architecture is divided into three layers, each of which interacts with the others. The class diagrams presented below are categorized according to their respective layers within the architecture:

- **UI/Presentation Layer**: The React client application serves as both a user interface for rendering pages and input validation, acting as an interface for communication with the backend's controllers. These controllers, in turn, call upon services within the domain layer to fetch data or create new objects, which are subsequently returned to and displayed by the client.
- **Domain Layer**: This layer is responsible for handling requests from the presentation layer and managing all application-specific domain logic, including tasks like creating venues and managing user accounts. Within this layer, services interact with the data source layer to perform operations such as creating, updating, or deleting objects.
- **Data Source Layer**: This layer contains the mappers. The focus of this layer is on the transformation of objects to and from the database representation.

2.1 Presentation Layer

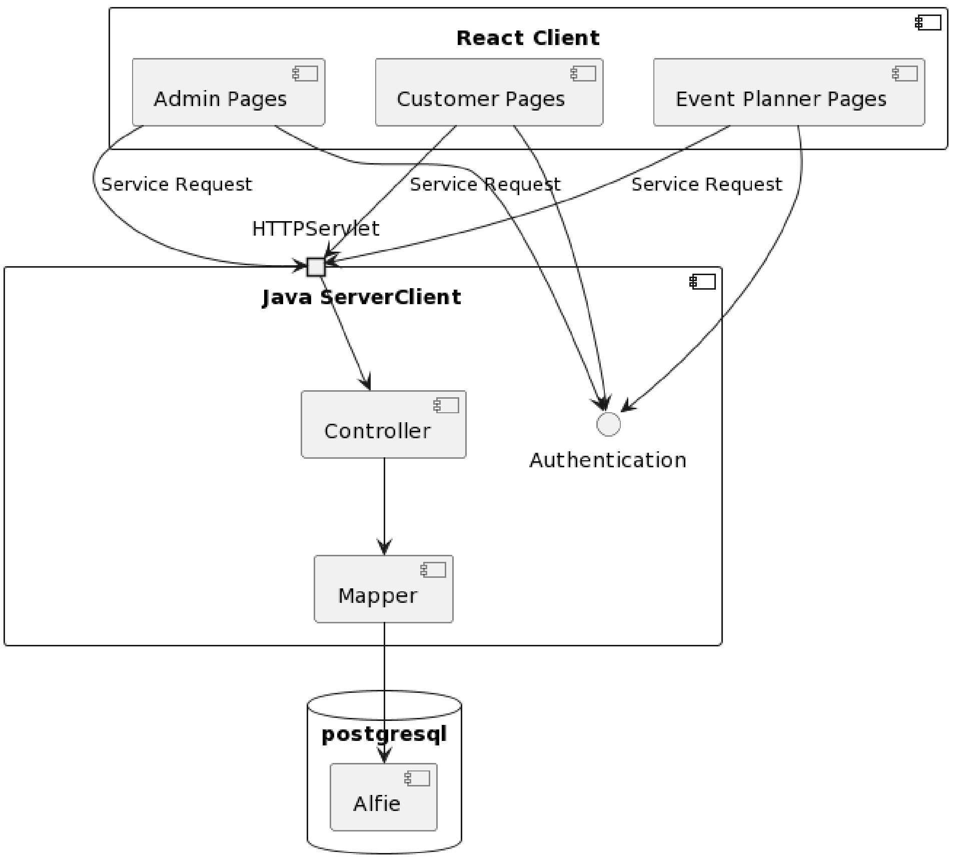
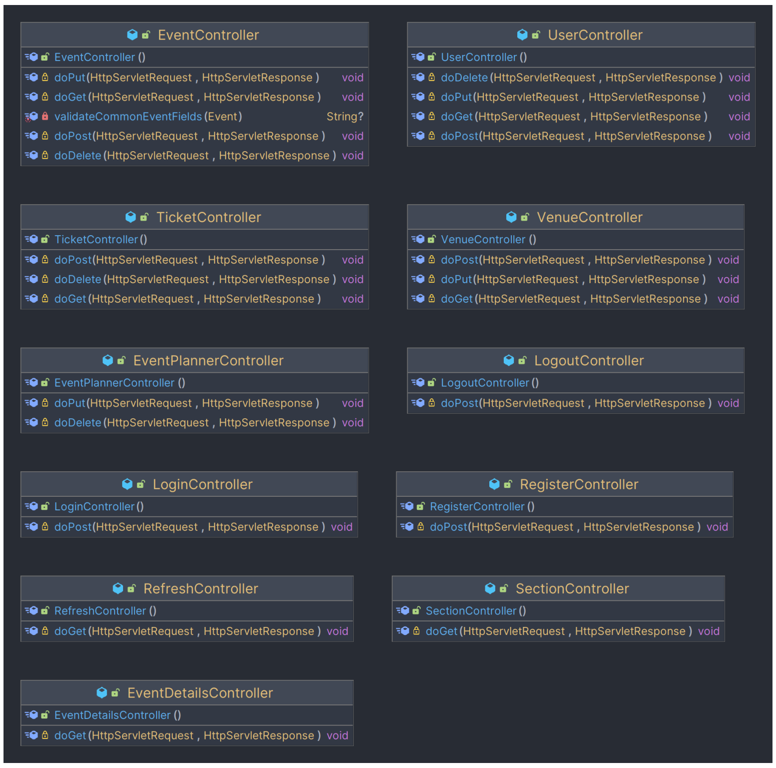


Figure 1: Component Diagram of Presentation Layer

2.2 Domain Layer



```
EventController
EventController ()
doPut(HttpServletRequest , HttpServletResponse )        void
doGet(HttpServletRequest , HttpServletResponse )        void
validateCommonEventFields(Event)                        String?
doPost(HttpServletRequest , HttpServletResponse )       void
doDelete(HttpServletRequest , HttpServletResponse )     void

UserController
UserController ()
doDelete(HttpServletRequest , HttpServletResponse )     void
doPut(HttpServletRequest , HttpServletResponse )        void
doGet(HttpServletRequest , HttpServletResponse )        void
doPost(HttpServletRequest , HttpServletResponse )       void

TicketController
TicketController ()
doPost(HttpServletRequest , HttpServletResponse )       void
doDelete(HttpServletRequest , HttpServletResponse )     void
doGet(HttpServletRequest , HttpServletResponse )        void

VenueController
VenueController ()
doPost(HttpServletRequest , HttpServletResponse )       void
doPut(HttpServletRequest , HttpServletResponse )        void
doGet(HttpServletRequest , HttpServletResponse )        void

EventPlannerController
EventPlannerController ()
doPut(HttpServletRequest , HttpServletResponse )        void
doDelete(HttpServletRequest , HttpServletResponse )     void

LogoutController
LogoutController ()
doPost(HttpServletRequest , HttpServletResponse )       void

LoginController
LoginController ()
doPost(HttpServletRequest , HttpServletResponse )       void

RegisterController
RegisterController ()
doPost(HttpServletRequest , HttpServletResponse )       void

RefreshController
RefreshController ()
doGet(HttpServletRequest , HttpServletResponse )        void

SectionController
SectionController ()
doGet(HttpServletRequest , HttpServletResponse )        void

EventDetailsController
EventDetailsController ()
doGet(HttpServletRequest , HttpServletResponse )        void
```

Figure 2: The Controllers of the Music Events System

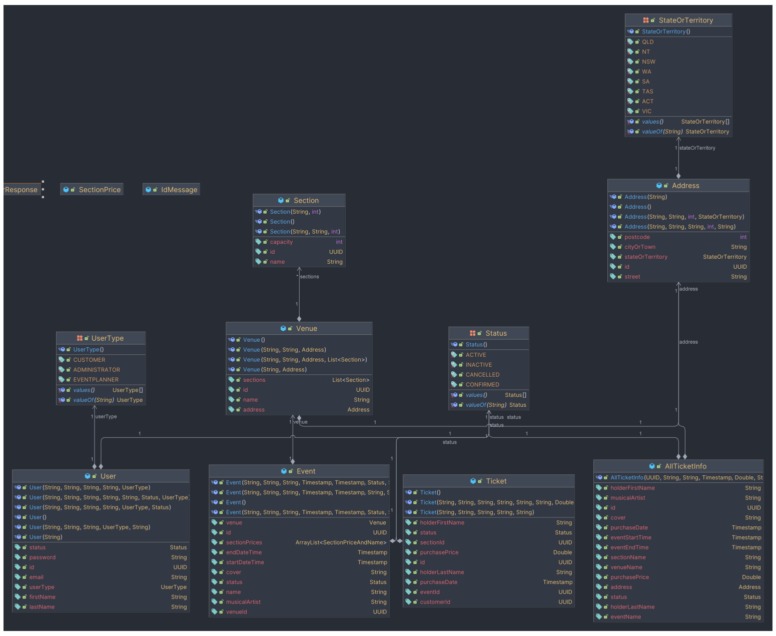


Figure 3: The Domain Models of the Music Events System

2.3 Database Mappers Layer

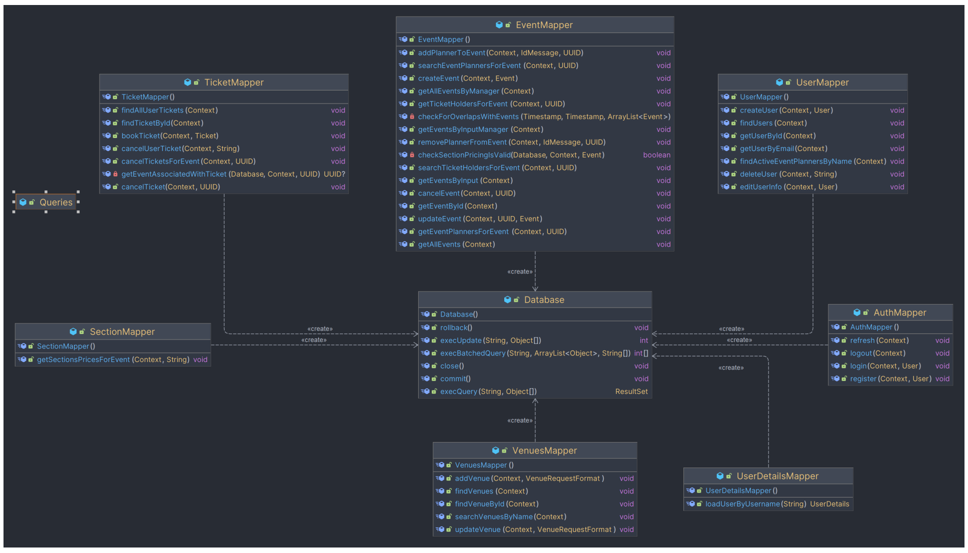

EventMapper
EventMapper()
addPlannerToEvent(Context, IdMessage, UUID) void
searchEventPlannersForEvent (Context, UUID) void
createEvent(Context, Event) void
getAllEventsByManager(Context) void
getTicketHoldersForEvent (Context, UUID) void
checkForOverlapsWithEvents (Timestamp, Timestamp, ArrayList<Event>)
getEventsByInputManager (Context) void
removePlannerFromEvent (Context, IdMessage, UUID) void
checkSectionPricingIsValid(Database, Context, Event) boolean
searchTicketHoldersForEvent (Context, UUID) void
getEventsByInput (Context) void
cancelEvent(Context, UUID) void
getEventById(Context) void
updateEvent (Context, UUID, Event) void
getEventPlannersForEvent (Context, UUID) void
getAllEvents (Context) void
TicketMapper
TicketMapper()
findAllUserTickets (Context) void
findTicketById(Context) void
bookTicket(Context, Ticket) void
cancelUserTicket(Context, String) void
cancelTicketsForEvent (Context, UUID) void
getEventAssociatedWithTicket (Database, Context, UUID) UUID?
cancelTicket(Context, UUID) void
UserMapper
UserMapper()
createUser(Context, User) void
findUsers(Context) void
getUserById(Context) void
getUserByEmail(Context) void
findActiveEventPlannersByName (Context) void
deleteUser (Context, String) void
editUserInfo (Context, User) void
Queries
«create»
Database
Database()
rollback() void
execUpdate (String, Object[]) int
execBatchedQuery (String, ArrayList<Object>, String[]) int[]
close() void
commit() void
execQuery (String, Object[]) ResultSet
SectionMapper
SectionMapper()
getSectionsPricesForEvent (Context, String) void
AuthMapper
AuthMapper()
refresh (Context) void
logout(Context) void
login(Context, User) void
register(Context, User) void
VenuesMapper
VenuesMapper()
addVenue (Context, VenueRequestFormat) void
findVenues (Context) void
findVenueById (Context) void
searchVenuesByName (Context) void
updateVenue (Context, VenueRequestFormat) void
UserDetailsMapper
UserDetailsMapper()
loadUserByUsername (String) UserDetails

Figure 4: The data persistence layer of the Music Events System

3 Pattern Descriptions

In this section is an overview of the various pattern implementations employed in the project, providing a comprehensive overview of their execution and including supporting diagrams for enhanced clarity.

3.1 Domain model

As illustrated in Figure 3, the domain models establish a seamless one-to-one mapping between the database tables and the domain classes within the Music Events System, which helps enhance the clarity and ease of managing domain logic. This approach yields has the advantage of domain objects closely mimic database tables, with just a handful of database-specific differentiation, resulting in streamlined development and maintenance. Figure 4 offers a vivid portrayal of the database schema, underlining the benefits of this architectural alignment.

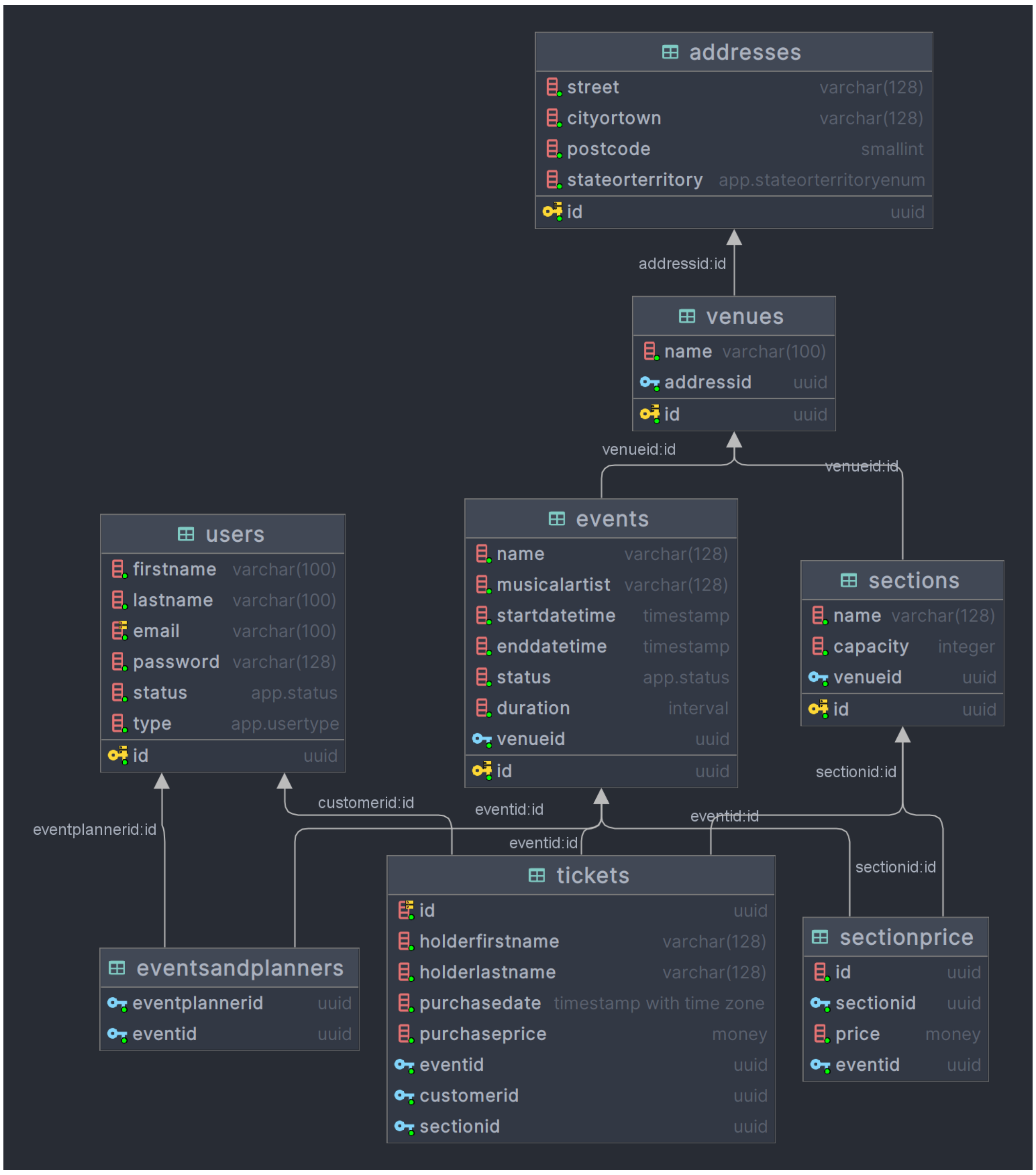


Figure 5: Music Events Database Schema

3.2 Data mapper

We opted for a domain model to structure our domain logic, and in the data persistence layer, we implemented the data mapper pattern. To maintain simplicity and ease of maintenance, we used a one-to-one mapping approach, creating a dedicated data mapper for each corresponding domain class. This approach reduced the systems complexity, a potential issue when dealing with numerous domain objects and a single, overarching system data mapper. Moreover, it ensured that each individual mapper possessed high cohesion. This readability and developer experience, resulting in a more streamlined development process. See Figure 6 for a customer to register their account in the Music Events System, highlighting the cohesive nature of the design.

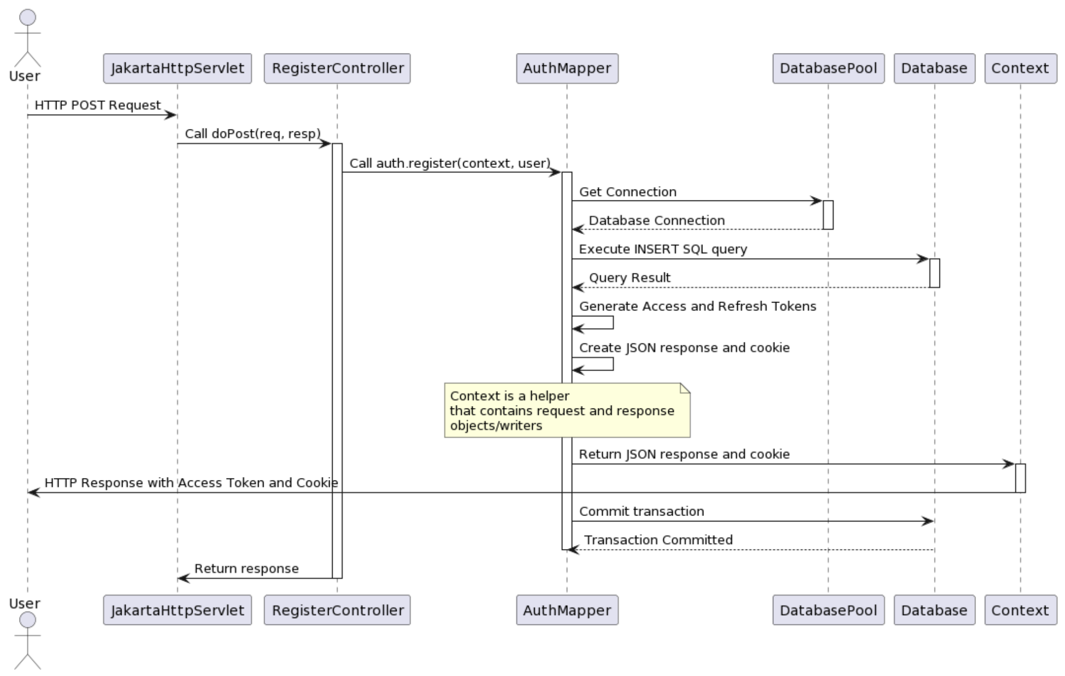


Figure 6: Sequence Diagram for Registration Use Case

3.3 Unit of work

The concept of the unit of work as we saw in the subject was conflicting with the client-server architecture being implemented in the project, following the approach of having the interface running entirely in the client and communicating through HTTP request to a REST API server. An architectural decision that arises naturally is to make all business transactions to be on a single request. This means that the objects that are created on the server that interact with the database will no longer have long lives in the application life cycle, because they will only exist in the time that the request is being processed, and then will be deallocated.

With this in mind, keeping a literal unit of work for each business transaction (each request) would be just adding another layer that is only going to generate more overheads in the system, and instead of being helpful it could complicate the interaction with the database.

To accommodate for this we decided, as mentioned before, to have business transactions to be processed on a single request being made to the server. Then, for still having a true unit of work concept, each request that modifies anything in the database will be made with a database transaction. This means that we delegate to the database the concurrency check that arises from not having an explicit unit of work and move on to have an optimistic concurrency control.

3.4 Lazy load

Following on what was mentioned in the previous section, the concept of lazy load may be a little bit tricky to translate to the client-server architecture. We no longer have the long-lived objects, and whenever we raise an object we might as well raise all of its fields to be sent to the client because to make another request would be a bigger overhead.

However we still have a very clear way of lazy loading, meaning we only get the data that we need: pagination. In most applications we are going to have lists of data, which are going to be very expensive to get and send over to the client. Through pagination we achieve lazy loading by sending over a limited number of elements to the client on each request as the client moves through pages. This ensures that we are only loading the data that is needed, keeping the response time to a minimum.

An additional optimisation that could be implemented is to cache previously queried pages. This would however require a more complicated implementation because we not only need to store the data, but also to define the way for that data to be refreshed without a big disruption to the user workflow and at the time ensure that the data is up to date with the latest versions in the database.

3.5 Identity field

We employed the 'identify field' pattern to establish relationships by incorporating a database ID field within our object structure. This field preserved object identity alignment between in-memory objects and corresponding database records within database. By assigning a UUID to each domain object, which was mapped to the database reference ID for the same object, utilising the unique ID associated with each table's domain object.

Below, you will find the domain class and corresponding database schema for 'Venue,' illustrating the mapping of the database ID. In the schema, the 'id' column in the 'Venues' table directly corresponds to the 'id' attribute in the 'Venues' class.

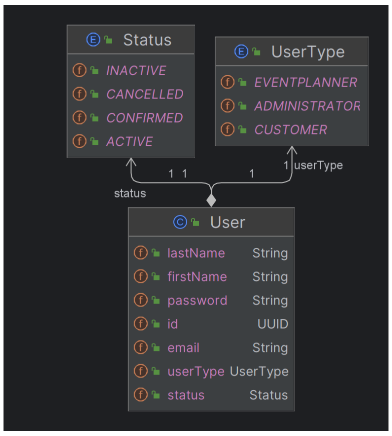


Figure 7: 'User' Domain Class

3.6 Foreign key mapping

We use the identity field pattern for linking domain objects to their corresponding database rows, encounters the need to preserve references to other domain objects. It's achieved by foreign key mapping, where objects like events linked to venues are stored in the database with a reference to their parent object's ID. For example, the 'Sections' table maintains a reference to its parent 'Venues' object, facilitating efficient queries for retrieving and storing of venues and their sections while ensuring the persistence of these references for future operations.

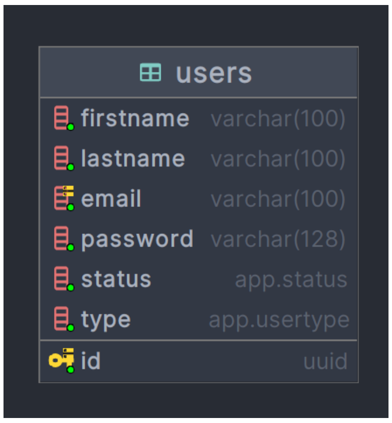


Figure 8: 'User' Database Schema

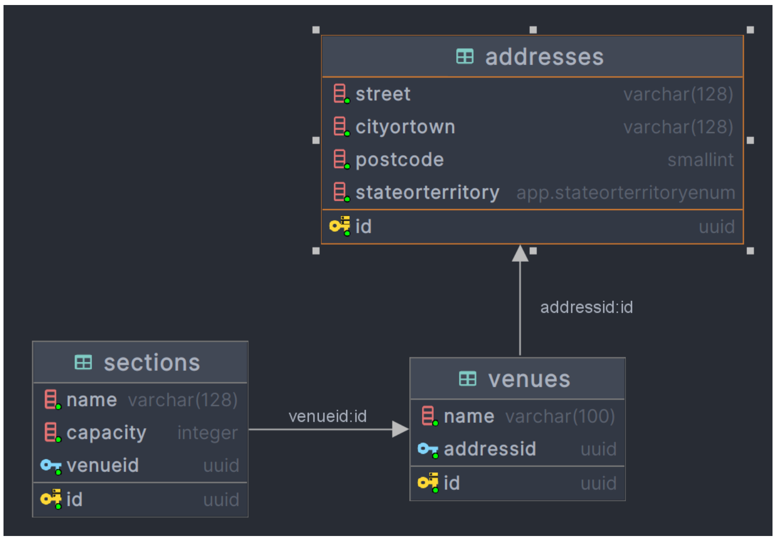


Figure 9: 'Venues' Domain Class

3.7 Association table mapping

We employ the association table mapping pattern to address many-to-many relationships that lack a single-valued endpoint. This approach is helpful for instances where one entity can be related to multiple instances of another entity. For instance, in the context of 'Event Planners' and 'Events,' where an 'Event Planner' can be associated with one or more 'Events,' and an 'Event' can be managed by one or more 'Event Planners,' we utilise association table mapping to solve this.

As illustrated in the figure below, we establish a table named 'eventsandplanners.' This table includes foreign keys from both the 'Events' and 'Event Planner' tables, enabling us to manage and navigate this complex many-to-many relationship efficiently.

3.8 Embedded value

An embedded value pattern involves the mapping of an object's values into the fields of its containing entity. This pattern is applied to related values within domain objects by representing them as their own encapsulated objects. For Music Events system, we applied the embedded value pattern to create a separate class for 'Address,' which encompasses attributes such as street, 'cityortown', 'stateorterritory', and postcode. This 'Address' class is then utilized within the 'Venue' class.

In the database schema, the address-related values were stored independently rather than being incorporated directly into the 'venue' table. The database schema table for 'Venue,' the address fields ID is kept within the 'venue' table itself. The 'Venue' class, where the 'Address' class is represented as an

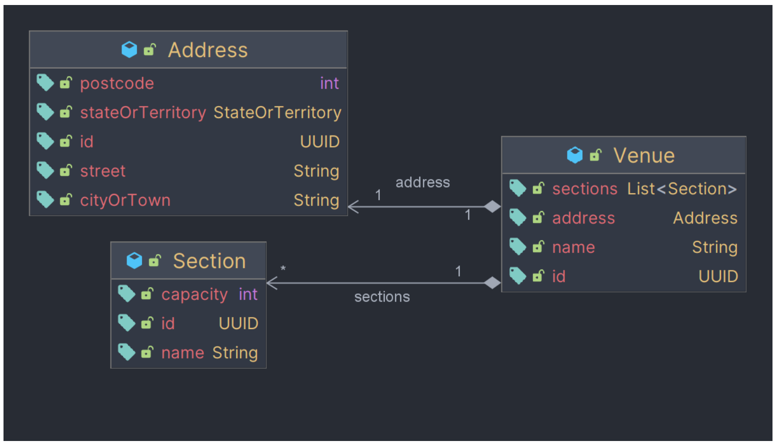


Figure 10: 'Venues' Database Schema

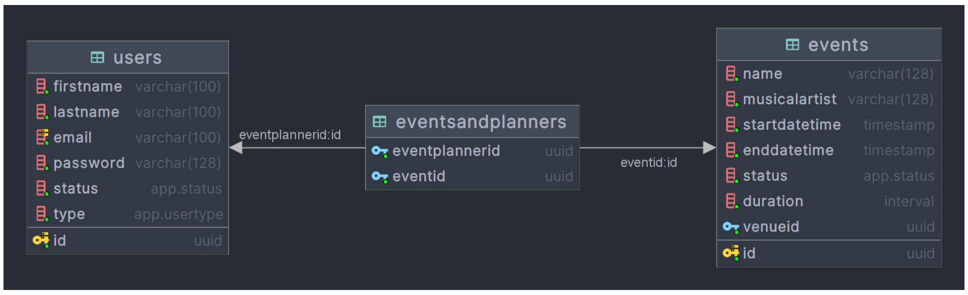


Figure 11: 'eventsandplanners' database schema

instance variable, highlights the separation of concerns between the object-oriented representation and the database storage structure.

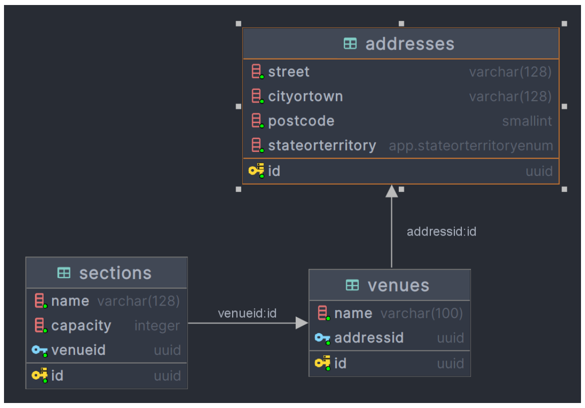


Figure 12: 'Venues' Domain Class

3.9 User Type Single Table Inheritance

To efficiently represent the inheritance structure of three types of accounts (Customer, Event Planners, and Administrator), we chose to use Single Table Inheritance (STI) model was deemed the most suitable approach for the database design. It provides an elegant and efficient way to manage different account types within the system, ensuring clarity and maintainability in the database schema.

In this approach, a unified 'User' table was created to store common account-related data for all types of accounts. Additionally, the table incorporated fields for the 'account type' to distinguish the account's

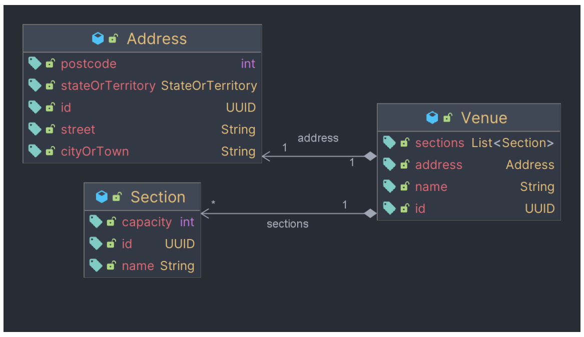


Figure 13: 'Venues' Database Schema

specific subclass, as well as the status of the user in the system. This approach worked particularly well for the users of our system, as they all require the same data to be stored in them. This design decision simplifies data management and allows the Music Event System to identify the account type and appropriately filter content. The database schema's representation of the 'User' table, demonstrates our use of the account type information. The domain model shows a visual representation of the 'User' table and its associations within the Music Event System, demonstrating how this inheritance structure is implemented.

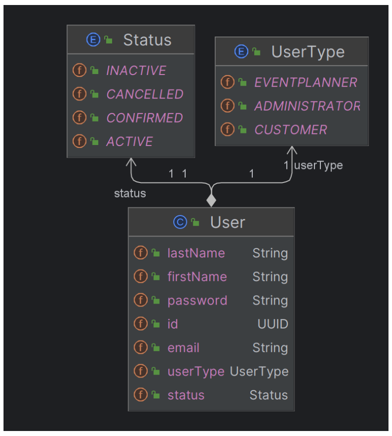


Figure 14: 'User' Domain Class

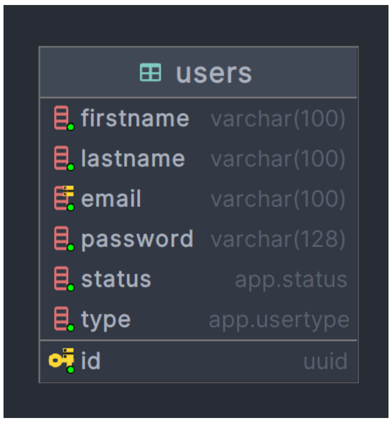


Figure 15: 'User' Database Schema

3.10 Authentication and Authorization

For the implementation of security in our application we opted for a stateless method - JSON Web Tokens (JWT). While not being strictly a security mechanism, they are more a proposed standard for safe communication of claims between two parties. However, they made their way into the authentication and authorisation world because of the mechanisms that this approach provides.

The way JWT-based authentication works is as follows:

1. The client sends their credentials to the server.
2. The server validates that the user credentials are valid, and retrieves additional metadata of the user to be used as *claims*.
3. The server encodes a three section token using a chosen algorithm. The three sections are the header (containing information about the token), the payload (the claims), and the signature (a hash obtained from the claims and processed with the algorithm chosen).
4. The server returns the encoded token to the user.
5. The user utilises the received token on every subsequent request to validate its identity.

While this provides a layer of safety by ensuring that tokens can't be tampered with (because any change to the claims would invalidate the signature and make the token invalid), their use can cause different issues to arise. How do we know if a user still has a valid session, and what happens if a third party gets hold of the token?

To account for both those problems we use a pair of tokens: an access token and a refresh token. The access token has a short lifespan, and the refresh token a large lifespan. The user has both, and whenever the access token expires, a new one can be requested without needing to authenticate again by using the refresh token. It is then possible to refresh the refresh token during the refresh action as well to avoid having the user needing to authenticate again after it expires.

Then, based on the values of the claims we can implement authorisation on a per-method basis, because we can include the user type as a role and then retrieve than from the token in the server and check if the user that made the request is not only authenticated, but has the required permissions to access that endpoint or UI page.

4 Design Rationale

4.1 Unit of Work

The unit of work as described in the previous section was implemented in all and any request that has an action that modifies the database. This means, any CREATE, UPDATE, or DELETE operation. This way, we ensure that the unit of work is implemented through having a database transaction that matches the business transaction, which is encapsulated completely in a request. This delegates the responsiblity of keeping track of new, modified, and deleted objects to the database.

It is important to note that we are implementing soft deletes in our project. This means that whenever we want to “delete” a record, all we do is update its status field to show INACTIVE or CANCELLED. This is in order to preserve relations, and prevent breaking dependencies. We also help ensure unique records, for example for the users. If a user closes its account, and reopens it later, the user will still have access to all its previous history in the system and keep its old ID, rather than that all being erased.

4.2 Lazy Loading

Lazy loading was implemented as in the description of lazy loading in the previous section on all the tables that make a list of all items in the database, for example, listing all users, venues, events, and so on. This ensures a cleaner UI and also prevents unnecessary transmission of data.

This is also helpful to the user, because it’s easier for them to navigate in the application, and the whole interaction with the UI feels more responsive and fast. However, in our application we have implemented plain pagination. This means that whenever the users goes back to a previous page, a new request is made for getting that data. This decision comes from the notion that we are having an optimistic concurrency control, so it would be more meaningful to the user when changing a page to see an event again on a page change, which means that a new event was added.

4.3 Database Pooling

Considering the database, getting a database connection is a very expensive operation. If we were to create a new connection for each and every request, we would incur in an unnecessary overhead and additional resource consumption. The way we solved this and optimised the interactions with the database is thorough a database connection pool.

The database connection pool then is in charge of initialising a number of connections that can be called upon to process a set of queries, and then return it to the pool for it to be reused in a later request.

Some additional configuration required for this to match the Unit of Work implementation we have in this application was that we need to turn off the auto-commit option in the connections. It is a Java design decision that all queries automatically commit as as soon as they are done. For being able to properly handle the business transaction with database transactions, we need to explicitly turn this off. This way, the connections will only call BEGIN by default, but the COMMIT and ROLLBACK are handled manually by the developer.

4.4 JWT implementation

The way that JWT authentication and authorisation was implemented in our application is as follows:

Client

The access token is stored in memory through React’s context, giving us have access to all the user claims from anywhere in the component tree. Then, for giving the user a smoother interaction, we implemented a request and response interceptor with the Axios[1] library.

[1]HTTP client for browsers and Node.js, https://axios-http.com

This interceptor works by adding the JWT received on login (obtained from React context at this point) to the Authorization header in the request. However, this becomes interesting when it comes to checking the response. If the response indicates a 401 error (Unauthorised) this could mean that the access token that was used had expired.

If this is the case, the interceptor is going to put a hold on that request, and perform a refresh. If it succeeds and gets a new access token, then it will retry the previously failed request and move on with execution. However, if it fails, that means the refresh token has expired as well and the user is redirected to the login page to authenticate again. It is worth noting that the client will not have access to the refresh token, it just gets added on every request the client makes to the server. This will be explained in the following section.

Server

On the server side, the implementation we have is just in charge of 4 simple tasks:

- Create the access token
- Create the refresh token
- Validate a token
- Return the tokens to the client

Creating and validating the tokens is a trivial task that is completed with the libraries available. We can just return the access token to client with no problem, however, we run into a more interesting question when we come to the refresh token.

We have different alternatives with it, and the most accepted one is to set is as a cookie on the response to the client. For this, we need to make sure we set a couple of parameters in the cookie. First of all, we need to ensure that the cookie is set as HttpOnly. This way, the client cannot get the value of the cookie and use it to get any number of access tokens it wants. Another thing we want to turn on, is the Secure flag on the cookie which just adds one more layer of security to our whole flow.

Once all this is set, the proper security filter chains can be implemented through Spring Security to validate the tokens and authorise request based on roles from the claims.

5 New/Updated Use Cases

The following represents a new/updated set of use cases based on feedback received from the teaching team and the inclusion of new, previously overlooked use cases that have become necessary. Please note that any unmodified or removed use cases from part 1A are not included in this section.

5.1 Administrator Use Cases

UC-A-01 View All Users

Brief description	Allows the administrator to view all users in the system
Preconditions	• The User is logged in • The User is viewing the admin dashboard page
Post Conditions	• The User selects "All Users" • A table containing the details for system users, including name, email, and role is visible • Only 10 users are displayed at a time • Forward and Back buttons for navigating through the table are displayed
Event flow	1. A table listing all users, along with their details (name, email, role etc.) is displayed 2. The User selects the Forward and Back buttons to view more users
Alternate flows	AF01 Access from Nav Bar 1. Instead of step 1, the User selects "View Users" from the navbar 2. All other steps remain the same

UC-A-02 Create User

Brief description	Allows the administrator to create a user with a specified role: Admin, Event Planner, or Customer.
Preconditions	• The User is logged in • The User is on the "View All Users" page
Post Conditions	• The entered user details are added to the application database, creating a new user record • The User is navigated to a "View User" page, where the details of the newly created user are visible
Event flow	1. From the "View All Users" page, the User clicks "Create Users" 2. The User is navigated to a form where they can enter User details 3. The User enters the required user details. 4. The User selects a role for the new user (Admin, Customer etc.) 5. The administrator selects "Submit"
Alternate flows	AF01 Cancel Registration Process 1. Between Step 2 and Step 5, the User selects "Cancel" 2. The User is redirected to the "View all Users" page. 3. Note that all data entered into the form will be lost AF02 Duplicate Registration 1. Upon completing, if a user submits a registration using an email that already exists in the system database, 2. An error notification will appear
Exception flows	EF01 Role Selection Error (a) If, during step 4, the administrator fails to select a role option. (b) The system displays an error message prompting the administrator to select a role before proceeding. EF02 Empty Text Fields 1. The User submits a registration form with missing fields 2. Validation messages will be displayed on the missing fields 3. Note that no new user will be created in the system EF03 Cancel 1. The user cancels the update process by clicking "Cancel" 2. Any non-submitted changes are lost.

UC-A-03 View User

Brief description	Allows the administrator to view the details of a user
Preconditions	• The User is logged in • The User is viewing a list of users on the "View All Users" page
Post Conditions	• The selected user's details, including name, role, and status, are are visible. • "Edit" and "Delete" buttons are visible
Event flow	1. The User selects a user from the list of users 2. The User is redirected to a page where the selected user's details are displayed.

UC-A-04 Update User Details

Brief description	Allows the administrator to update a user's account in the system
Preconditions	• The User is currently logged in • The User is on the "View User Details" page
Post conditions	• The entered user details are updated to the database. • The user is redirected to the "View User Details" page, where the updated user details are visible
Event flow	1. From the "View User Details" page, the User selects "Edit" 2. A form where the details of the user being viewed can be edited is displayed 3. The User keys in the updated user details 4. The User selects "Update"
Alternate flows	All alternate flows from Create User apply
Exception flows	All exception flows from Create User apply

UC-A-05 Delete User

Brief description	Allows the administrator to remove a user from the system
Preconditions	• The User is currently logged in • The User is on the "View User Details" page
Post conditions	• The deleted user's database record is marked as inactive. • The User is navigated to the "View User Details" page for the deleted user • The deleted user's status is now displayed as "Inactive"
Event flow	1. From the "View User Details" page, the User selects "Delete"

UC-A-06 View All Venues

Brief description	Allows the administrator to view all venues in the system
Preconditions	• The User is logged in • The User is viewing the admin dashboard page
Post Conditions	• The User selects "All Venues" • A table containing the details for venues in the system, including name and address details, is visible • Forward and Back buttons for navigating through the table are displayed
Event flow	1. A table listing all venues in the system is displayed 2. The User selects the Forward and Back buttons to view more venues
Alternate flows	AF01 Access from Nav Bar 1. Instead of step 1, the User selects "View Venues" from the navbar 2. All other steps remain the same

UC-A-07 Create Venue

Brief description	Allows the administrator to create a venue in the system
Preconditions	• The User is logged in • The User is viewing the "View All Users" page
Post Conditions	• The entered Venue details are added to the application database, creating a new Venue record. • The User is redirected to a page displaying details of the newly created venue
Event flow	1. From the "View All Venues" page the administrator selects "Create New Venue" 2. A form where the User can enter venue details is displayed 3. The User fills in the venue details 4. The User selects "Create"
Alternate flows	EF03 Cancel 1. The user cancels the update process by clicking "Cancel" 2. Any non-submitted changes are lost
Exception flows	EF01 Section Type Error (a) If, during step 3, the administrator fails to select a section type option. (b) The system displays an error message prompting the administrator to select a valid type before proceeding. EF02 Duplicate Venue 1. The user submits a venue with the same name. 2. An error message will be displayed to the user 3. No new venue will be created in the system EF03 Venue Selection Error (a) If, during step 4, the administrator fails to select a role option. (b) The system displays an error message prompting the administrator to select a venue before proceeding. (c) No new venue will be created in the system EF04 Empty Text Fields 1. The User submits a registration form with missing fields 2. Validation messages will be displayed on the missing fields 3. Note that no new user will be created in the system

UC-A-08 Edit Venue

Brief description	Allows the administrator to edit details of an existing venue in the system.
Preconditions	• The administrator is currently logged into the system with an active session. • A venue to be edited exists in the system.
Post Conditions	• The venue details are updated.
Event flow	1. Within the "All Venues" section, the administrator searches for and selects the venue to be edited. 2. The administrator is navigated to a page with that venue's information. 3. The administrator selects the "Edit Venue" option for the selected venue. 4. The system displays the venue's current details in an editable form. 5. The administrator modifies the necessary venue information. 6. Once the changes are made, the administrator submits the form to update the venue's details.
Alternate flows	AF01 Cancel Edit Process 1. Between Step 2 and Step 3, the administrator selects to cancel the editing process. 2. The administrator is navigated back to the page with information about that venue, and no changes are saved.
Exception flows	All Exception flows from Create Venue apply

5.2 Event Planner Use Cases

UC-E-01 View All Managed Events

Brief description	Allow the user to view a list of all events that they manage
Preconditions	• The User has an account in the system • The User is logged into their account
Post conditions	• The User is able to see a list of all events that they manage
Event flow	1. The User navigates to the "Your Managed Events" page 2. A list of events that the User manages is displayed.
Exception flows	EF01 The User manages no events (a) No list appears, and a message indicating that no items were found is displayed

UC-E-02 Search All Managed Events

Brief description	Allow the user to search the events that they manage
Preconditions	• The User has an account in the system • The User is logged into their account • The User is viewing a list of events on the "Your Managed Events" page
Post conditions	• The User is able to see a list of all events that they manage
Event flow	1. The User enters the name of the event they wish to search for in the search bar next to the list of events 2. The User clicks "Search" 3. A list of events matching the search query is displayed
Exception flows	EF01 The User manages no events (a) No list appears, and a message indicating that no items were found is displayed

UC-E-03 View Event Details

Brief description	Allow the user to view the details of a single event that they manage
Preconditions	• The User has an account in the system • The User is logged into their account • The User is currently on the "Your Managed Events" page viewing a list of managed events
Post conditions	• The User is redirected to the "Event Details" page corresponding to the selected event • The details of the selected event are visible on the page • A list of ticket holders for the selected event is displayed on the page • "Edit Event", "Cancel Event", and "Cancel All Tickets" buttons are displayed on the page
Event flow	1. The User selects an event from the list of managed events

UC-E-04 Create Event

Brief description	Allow the user to create a new event
Preconditions	• The User has an account in the system • The User is logged into their account • The User has navigated to the "Create Event" page
Post conditions	• A new event is created in the database with details as specified by the User • The newly created event records the User as its manager • The User is redirected to the "Event Details" page, which displays details about the newly created event
Event flow	1. The User enters basic event details such as the event name and start date 2. The User enters the name of the venue they want to host the event at in the venue search bar 3. The User clicks the "search" button 4. A list of venues matching the User's search query are displayed to the user 5. The User selects their desired venue from the list of venues 6. The User enters section prices for each section of the venue 7. The User clicks the "Submit" button.
Alternate flows	AF01 Cancel 1. The User cancels the event creation by clicking the "Cancel" button 2. The user is redirected to the "Your Managed Events" page. Note that no new event is created in the system, and all data entered into the page is lost
Exception flows	EF01 Overlapping Events at Venue (a) After steps 1-7, if there is already an event booked at the same venue that overlaps with the new event, an error message is displayed indicating an event is already scheduled for the selected period. (b) No new event is created, the User is not redirected away from the page, and all entered data remains present in the page fields. EF02 Empty or Invalid Fields (one or more required fields are either left blank or populated with invalid data) (a) A validation message is displayed for each empty or invalid field. (b) No new event is created, the User is not redirected away from the page, and all entered data remains present in the page fields.

UC-E-05 Update Event

Brief description	Allow the user to update an existing event that they manage
Preconditions	• The User has an account in the system • The User is logged into their account • The User manages at least one event • The User has navigated to the "Edit Event" page via the "Event Details" page • The User manages the event they are viewing
Post conditions	• The event is updated to reflect the new details entered by the User • The User is redirected to the "Event Details" page, which displays details about the updated event
Event flow	1. Pre-populated fields of event data are displayed on the page 2. The User enters updated event details such as a new name or start date 3. The User enters updated section prices for each section of the venue 4. The User clicks the "Submit" button.
Alternate flows	See alternate flows described in Create Event
Exception flows	All exception flows described in Create Event also apply to this use case.

UC-E-06 View Event Managers

Brief description	Allow the user to view the event planners who manage an event
Preconditions	• The User has an account in the system • The User is logged into their account • The User manages at least one event • The User has navigated to the "Edit Event" page via the "Event Details" page • The User manages the event they are viewing
Post conditions	• A subpage containing details of the event's managers is displayed to the user • A searchable list of event managers (event planners who manage the event) is displayed inside the subpage • A searchable list of all event planners (including non managers) is also displayed inside the subpage • The lists clearly display the name and email for each event planner • The "View Managers" button is replaced with a "Hide Managers" button
Event flow	1. While on the "Edit Event" page, the User clicks the "View Managers" button
Alternate flows	AF01 View and then hide event managers 1. After completing step 1, the User clicks the "Hide Managers" button 2. The User is redirected to the original version of the "Event Details" page, where the list of planners is no longer visible

UC-E-07 Search Managers for Event

Brief description	Allow the user to search through the managers of an event
Preconditions	• The User has an account in the system • The User is logged into their account • The User manages at least one event • The User has navigated to the "Edit Event" page via the "Event Details" page • The User manages the event they are viewing • The User has the "Event Managers" subpage open • The User is currently viewing the table of event managers
Post conditions	• The table of event managers is updated to contain managers matching the search query
Event flow	1. While viewing the table of event managers, the User enters the name or email of the planner they wish to search for into the search bar 2. The User clicks "Search" 3. The table displays a list of event managers matching the search query, with a "Remove Planner" button alongside each planner
Exception flows	EF01 No managers of the event match the search query (a) The User completes step 1. (b) No list appears, and instead a message indicating that there are no matching managers for the event is displayed

UC-E-08 Add Manager to Event

Brief description	Allow the user to add other event planners as managers of an event
Preconditions	• The User has an account in the system • The User is logged into their account • The User manages at least one event • The User has navigated to the "Edit Event" page via the "Event Details" page • The User manages the event they are viewing • The User has the "Event Managers" subpage open • At least one event planner other than the User exists in the system
Post conditions	• The selected event planner is added as a manager of the event • The "Event Managers" subpage is updated to display the newly added planner in the list of event planners
Event flow	1. The User enters the name of the event planner they wish to add in the search bar 2. The User clicks "Search" 3. The a list of event planners matching the search query is displayed 4. The User selects the "Add Planner" button next to one of the options returned by the search
Exception flows	EF01 The selected planner already manages the event (a) After completing steps 1 to 4, a validation message indicating that the selected planner already manages the event is displayed (b) The selected planner remains a manager of the event, but no additional record is added to the database

UC-E-09 Remove Manager from Event

Brief description	Allow the user to remove other event planners as managers of an event
Preconditions	• The User has an account in the system • The User is logged into their account • The User manages at least one event • The User has navigated to the "Edit Event" page via the "Event Details" page • The User manages the event they are viewing • The User is currently viewing the event's list of managers • The event has at least one manager who is not the User
Post conditions	• The selected event planner is removed as a manager of the event • The list of event managers is updated to no longer display the removed planner
Event flow	1. While viewing the list of event managers, the User selects the "Remove Planner" button alongside one of the event's managers
Exception flows	EF01 The user attempts to remove themselves as a manager of the event (a) The User completes step 1. (b) An error message is displayed indicating that the event manager cannot remove themselves as a manager of the event (c) No change occurs, and the User remains a manager of the event

UC-E-10 View Ticket Holders for Event

Brief description	Allow the manager of an event to see all tickets to that event
Preconditions	• The User has an account in the system • The User is logged into their account • The User manages at least one event • The User has navigated to the "Event Details" page • The User manages the event they are viewing
Post conditions	• The User is able to see a list of all tickets for the event.
Event flow	1. While on the "View Managed Events" page, the user selects an event from the list of events 2. The User is navigated to the "Event Details" page, where a table containing a list of tickets is displayed
Exception flows	EF01 There are currently no ticket holders for the event (a) The User completes step 1 and is redirected (b) No list appears, and instead a message indicating that there are currently no ticket holders for the event is displayed

UC-E-011 Search Ticket Holders for Event

Brief description	Allow the manager of an event to search through all tickets to that event
Preconditions	• The User has an account in the system • The User is logged into their account • The User manages at least one event • The User has navigated to the "Event Details" page via the "View Managed Events" page • The User manages the event they are viewing • The User is viewing the "Ticket Holders" table
Post conditions	• The User is able to see a list of tickets for the event that match the search query.
Event flow	1. The User enters the name of the ticket holder they wish to search for into the table's search bar 2. The User clicks "Search" 3. The table displays a list of tickets matching the search query
Exception flows	EF01 There are no ticket holders who match the search query (a) The User completes step 1. (b) No list appears, and instead a message indicating that there were no matching ticket holders is displayed

UC-E-12 Cancel Event Ticket

Brief description	Allow the manager of an event to cancel a single ticket issued for the event
Preconditions	• The User has an account in the system • The User is logged into their account • The User manages at least one event • The User has navigated to the "Event Details" page via the "View Managed Events" page • The User manages the event they are viewing • There is at least one ticket holder for the event the User is viewing • The User is currently viewing the "Ticket Holders" table
Post conditions	• The selected ticket is cancelled • The "Ticket Holders" table is updated to display the status of the selected ticket as "Cancelled" • If the User later returns to the "Event Details" page, they will no longer see the cancelled ticket in the "Ticket Holders" table
Event flow	1. While reviewing the "Ticket Holders" table, the User sees a ticket they want to cancel 2. The User selects the "Cancel" button displayed alongside the ticket information

UC-E-13 Cancel All Tickets for Event

Brief description	Allow the manager of an event to cancel all tickets that have been issued for the event
Preconditions	• The User has an account in the system • The User is logged into their account • The User manages at least one event • The User has navigated to the "Event Details" page via the "View Managed Events" page • The User manages the event they are viewing
Post conditions	• All tickets currently issued for the event the User was viewing are cancelled • The user is redirected to the view managed events page • If the User later returns to the "Event Details" page, they will no longer see the cancelled tickets
Event flow	1. While viewing the "Event Details" page, the User selects "Cancel all tickets"

5.3 Customer Use Cases

UC-C-01 View Tickets

Name	View All Tickets
Brief description	Allows a User to view the tickets they have purchased for events.
Preconditions	• The User has an account in the system. • The User is logged into their account on the system. • The User has purchased a ticket to an event.
Post conditions	• The User is able to view the ticket and details about the event.
Event flow	1. The User navigates to their tickets page. 2. The User selects to "View Tickets". 3. The list of purchased tickets for upcoming events appears.
Alternate flows	1. N/A
Exception flows	EF01 There are no tickets purchased for any events. 1. The User completes step 2. 2. An empty list is shown, with a message that the User has no events upcoming. EF02 The User has purchased a ticket for an event but the event has been cancelled. 3. The User completes step 2, and a list of all events is shown. 4. The event that has been cancelled is labelled "Cancelled".

UC-C-02 Purchase Tickets

Name	Purchase Tickets to Event
Brief description	Allows the User to purchase tickets for an event.
Preconditions	• The User has an account in the system. • The User is logged into their account on the system. • The event that the user wishes to purchase tickets for is upcoming, has tickets available, and has sales open.
Post conditions	• The User now has purchased all tickets to the event • The User can view their ticket in their upcoming tickets page • The User is then redirected to the
Event flow	1. The User navigates to the event search page. 2. The User searches for the event that they wish to purchase a ticket for by the name of the event. 3. The User selects the event that they want to buy tickets for. 4. The User inputs the name to be on the tickets. 5. The User selects which section they want to sit in. 6. The User selects "Add Ticket" repeat steps 4 and 5 until they have input information about all tickets they wish to purchase 7. The User selects "Purchase". 8. The User selects "Book Tickets"
Exception flows	EF01 The User cancels the operation. (a) In any of the steps 1-6, the User may navigate to any other page, and the operation will abort. No tickets will be reserved.

UC-C-03 View Ticket Details

Name	View Ticket for Event
Brief description	Allows the User to view their tickets for a particular event.
Preconditions	• The User has an account in the system. • The User is logged into their account on the system. • The User has purchased a ticket to an event.
Post conditions	• The User is viewing the details of a ticket that they have purchased for an event.
Event flow	1. The User navigates to their tickets page. 2. The User selects the event for whose tickets they wish to view. 3. The User selects the ticket they want to view from the list of tickets that they have purchased. 4. The ticket is loaded up on the interface, and the user is able to view the details of the ticket, including event name, performer, venue, and event time.
Alternate flows	1. N/A
Exception flows	EF01 The User cancels the operation 1. At any point, the User selects to "Cancel", and they are returned to the homepage of the system.

UC-C-04 Cancel Ticket

Name	Cancel Ticket
Brief description	Allows the User to cancel a ticket that they have purchased for an event.
Preconditions	• The User has an account in the system. • The User is logged into their account on the system. • The User has purchased a ticket to an event.
Post conditions	• The User no longer has the ticket that they have purchased. • The ticket that they purchased is returned to the pool of tickets for this event, and is available for other Users to purchase.
Event flow	1. The User views the details of the event that they have purchased the tickets for that they wish to cancel. *[See UC-C-08]* 2. The User selects the ticket that they wish to cancel. 3. The User selects to "Cancel" the tickets. 4. The User selects to "Confirm" the cancellation. 5. The tickets are removed from their account, and the User can view them, but they are labelled as cancelled.
Exception flows	EF01 The User cancels the cancellation at any point. 1. In step 2, the User selects to "Unselect", or in step 3 or 4, the User selects to go "Back". 2. The operation is cancelled and the User continues to keep their tickets.

UC-C-05 View All Events

Name	View All Events
Brief description	Allows the User to view a list of all events in the system.
Preconditions	• The User has an account in the system. • The User is logged into their account on the system.
Post conditions	• The User is viewing a list of all events in the system.
Event flow	1. The User navigates to the page of all events. 2. The User can then scroll through the list of events, and move between pages of the event.
Alternate flows	1. N/A
Exception flows	EF01 No events are in the system 1. The User is notified that there are no events in the system, and encouraged to try again.

UC-C-05 View All Events

Name	View A Single Event's Details
Brief description	Allows the User to view the details surrounding an event in the system.
Preconditions	• The User has an account in the system. • The User is logged into their account on the system.
Post conditions	• The user is viewing the details of a single event.
Event flow	1. The User navigates to the page of all events. 2. The User selects an event that they are interested in knowing the details of. 3. The User is taken to a new page and is able to view all of the event's information.
Alternate flows	AF01 Search for event 1. The User searches for an event using the search bar (as in UC-C-07), selects the event they are interested. 2. The User arrives at step 3, and is able to view all of the event's information.
Exception flows	EF01 No events are in the system 1. The User is notified that there are no events in the system, and encouraged to try again.

UC-C-07 Search for Events

Name	Search for Events
Brief description	Allows the User to search for events that they are interested in.
Preconditions	• The User has an account in the system. • The User is logged into their account on the system. (UC-S-01)
Post conditions	• The User is viewing information about an event that they are interested in.
Event flow	1. The User searches for the event that they wish to know more about by typing the event's name into the search area. 2. The User is given a list of events that match what they typed into the search area. 3. The User selects the event that they are interested in learning about, and the information about that event is given.
Alternate flows	AF01 The User selects the wrong event. 1. After step 3, the User selects to go "Back", and returns to the list of all events. 2. The User continues from step 1.
Exception flows	EF01 No events match their search 1. The User is notified that there are no events that match their search, and encouraged to try again.