Event Management System

Milestone 1

Submitted to:

Dr. Mahmoud Khalil

Eng. Mazen Tarek

Submitted by:

Ali Mandour	24P0115
Hosny Lashin	24P0113
Marwan Ashraf	24P0101
Mazen Kandil	24P0423

4/23/2025

1. Project Description:

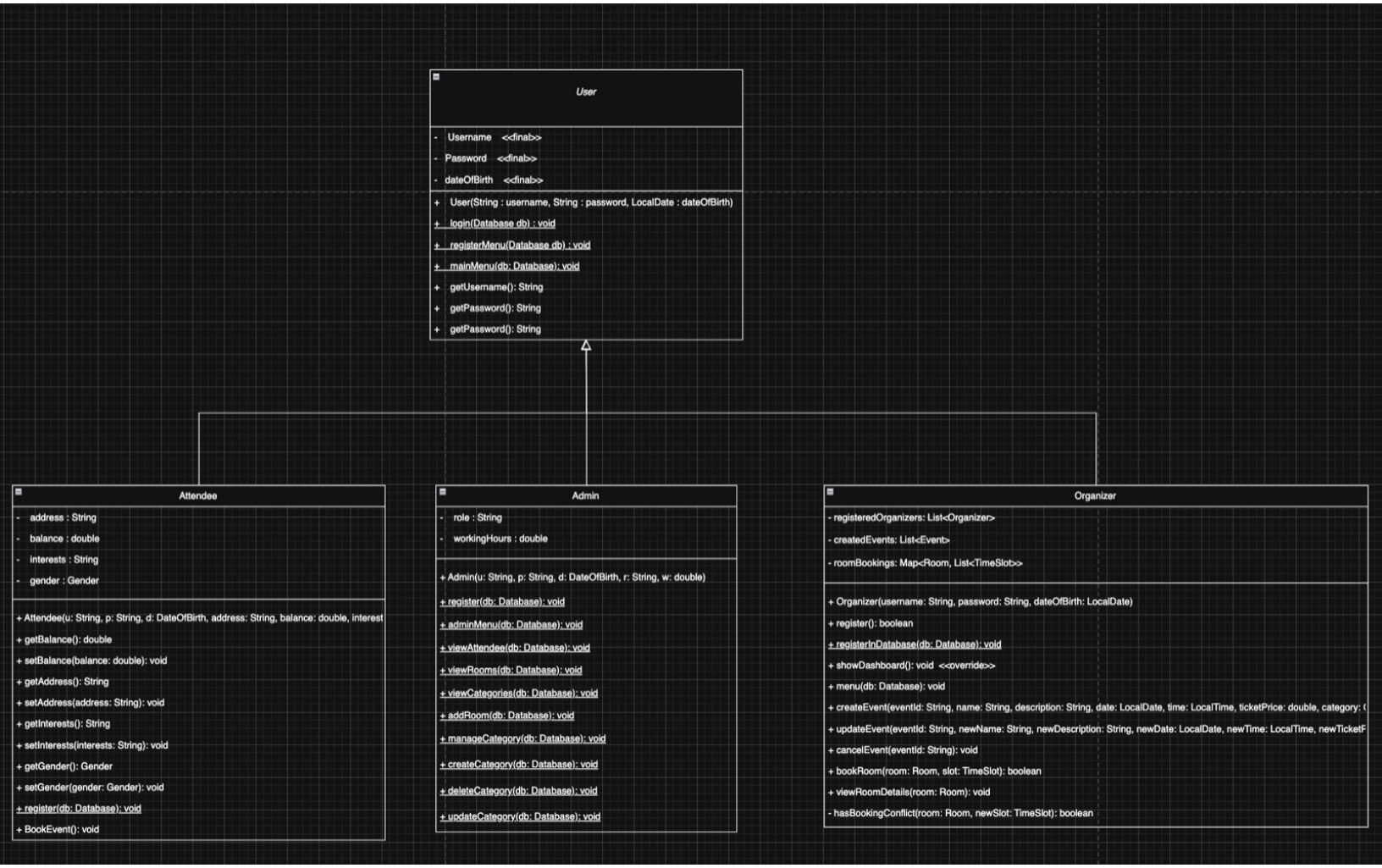
This project is an Object-Oriented Event Management System designed to manage various functionalities related to events and user roles. The system focuses on implementing backend features while following object-oriented programming principles.

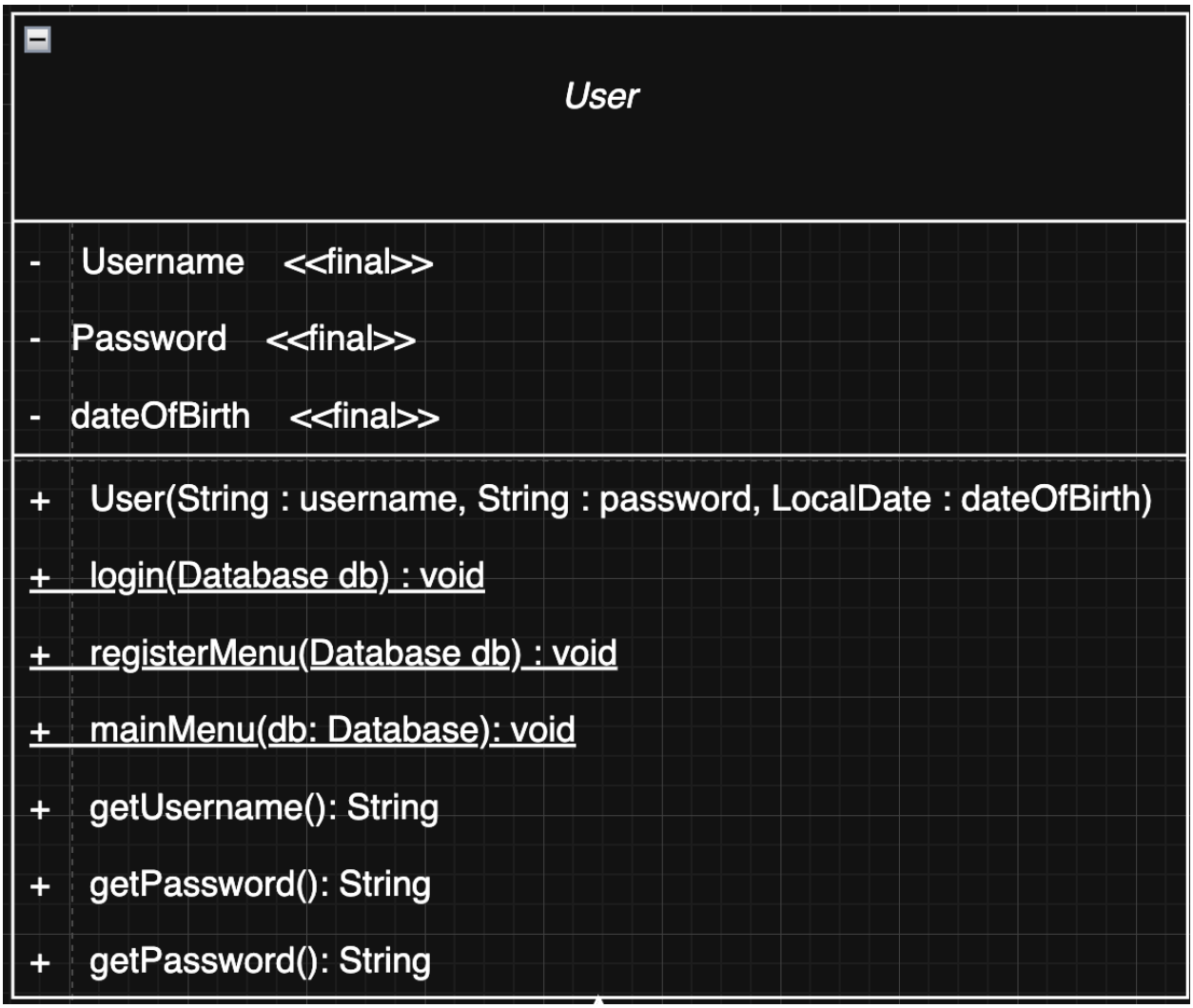
The application supports three main user roles:

1. **Admin:**
   The Admin has full control over the system. They are responsible for managing rooms that can be rented for events. Additionally, the Admin has access to a dashboard that provides an overview of all events, organizers, and attendees.
2. **Organizer:**
   Organizers are users who can create and manage events. They are also responsible for renting available rooms for their events through the system.
3. **Attendee:**
   Attendees are the end-users who can browse through the list of available events. They can choose an event to attend and purchase tickets through the system.

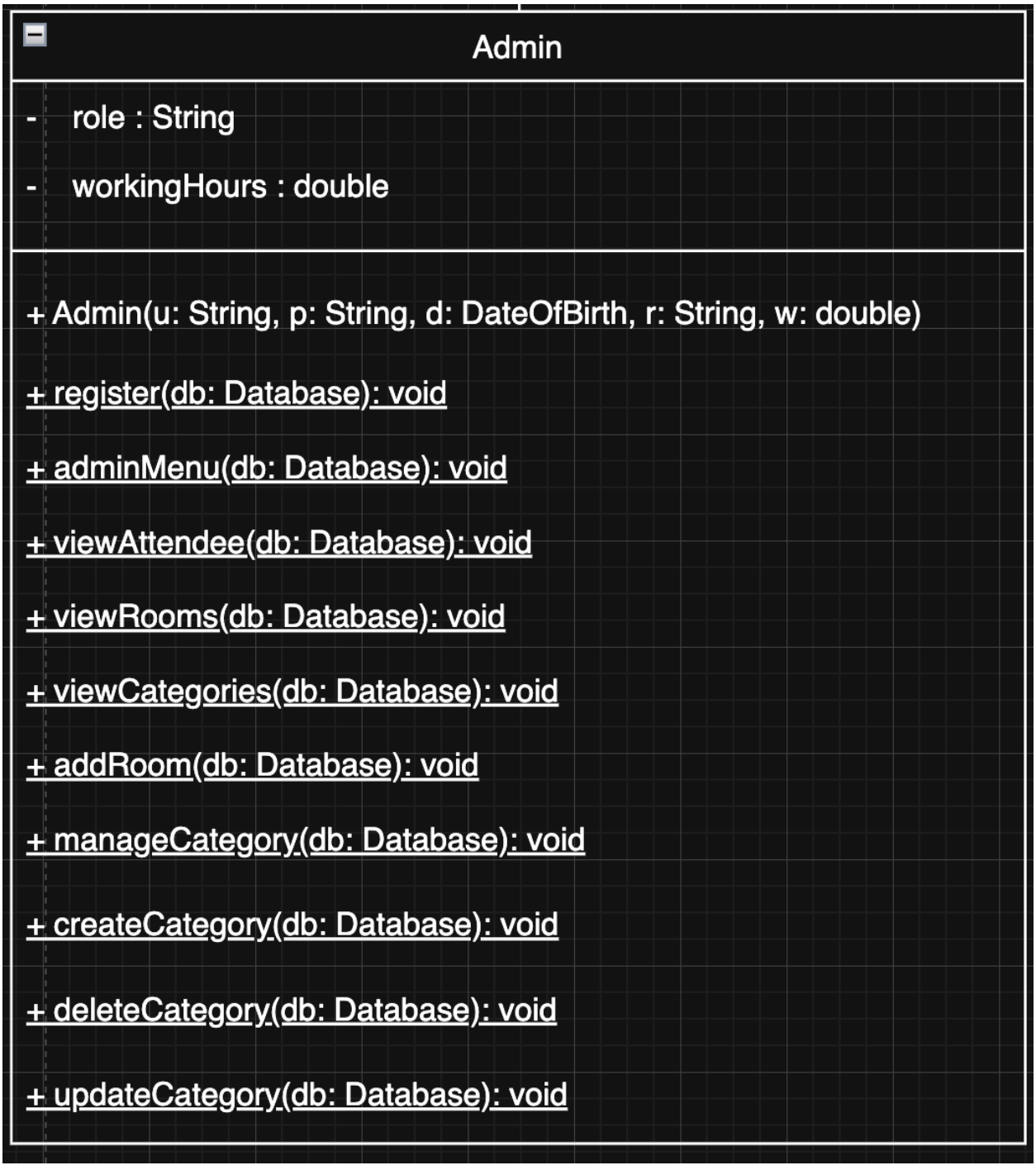
The system provides a structure for role-based access, event creation, room management, and ticket purchasing, with a focus on modularity, scalability, and ease of maintenance through proper use of Object Oriented Programming concepts.

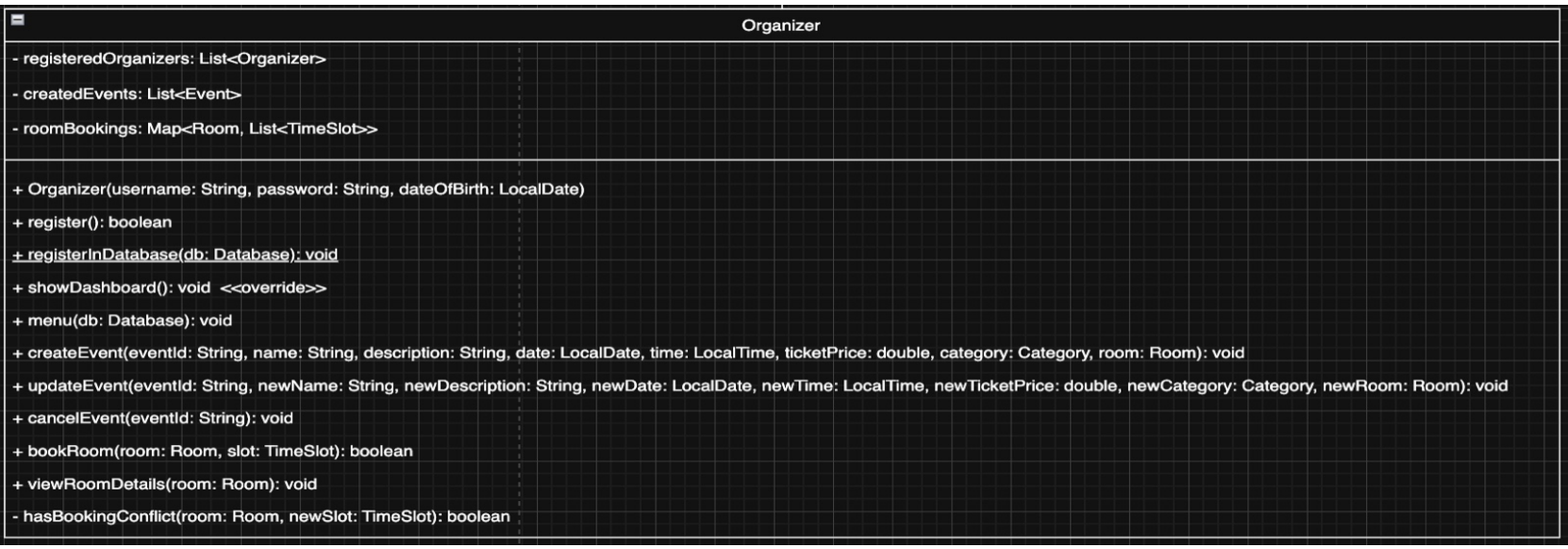
2. UML Diagram:

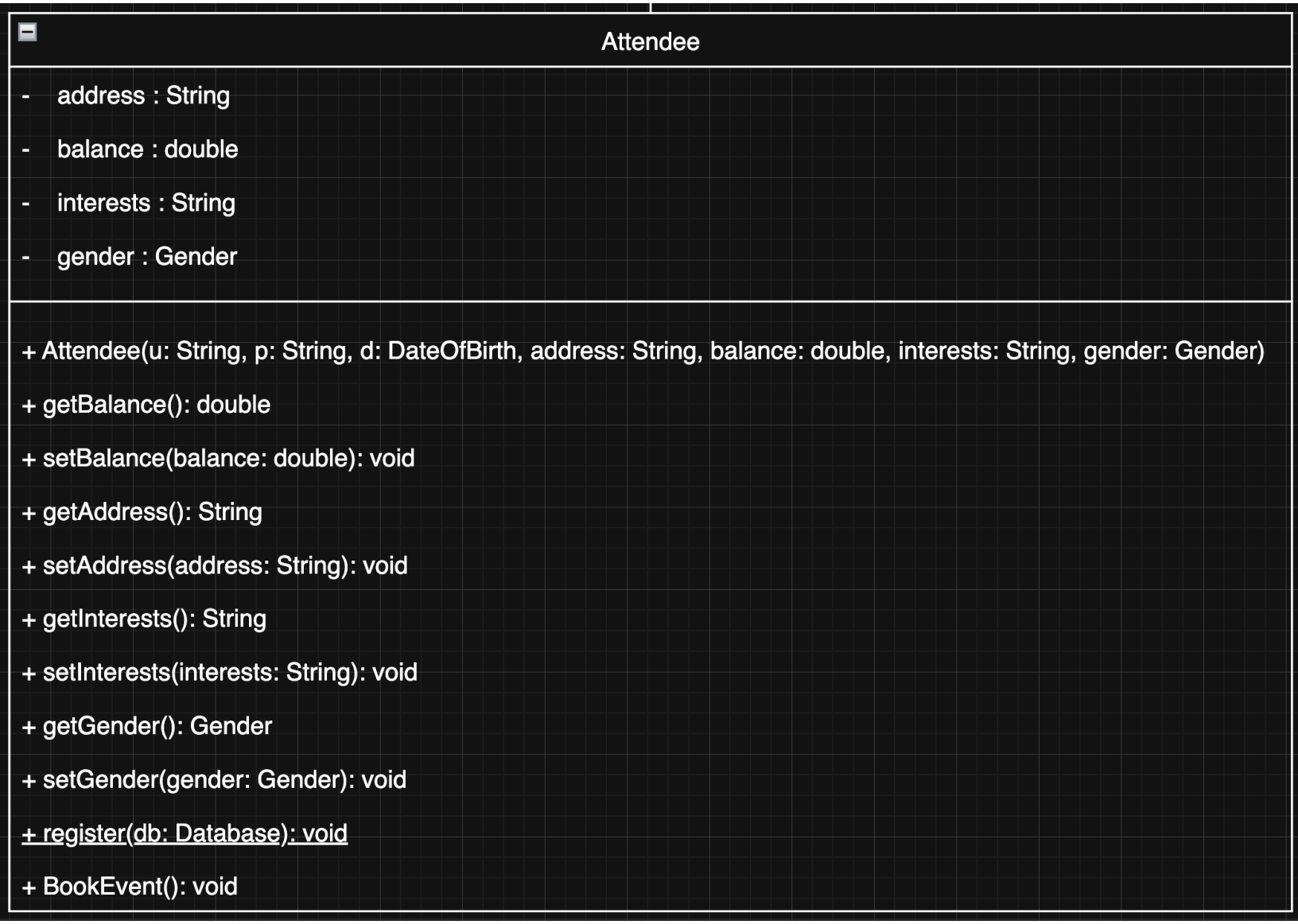


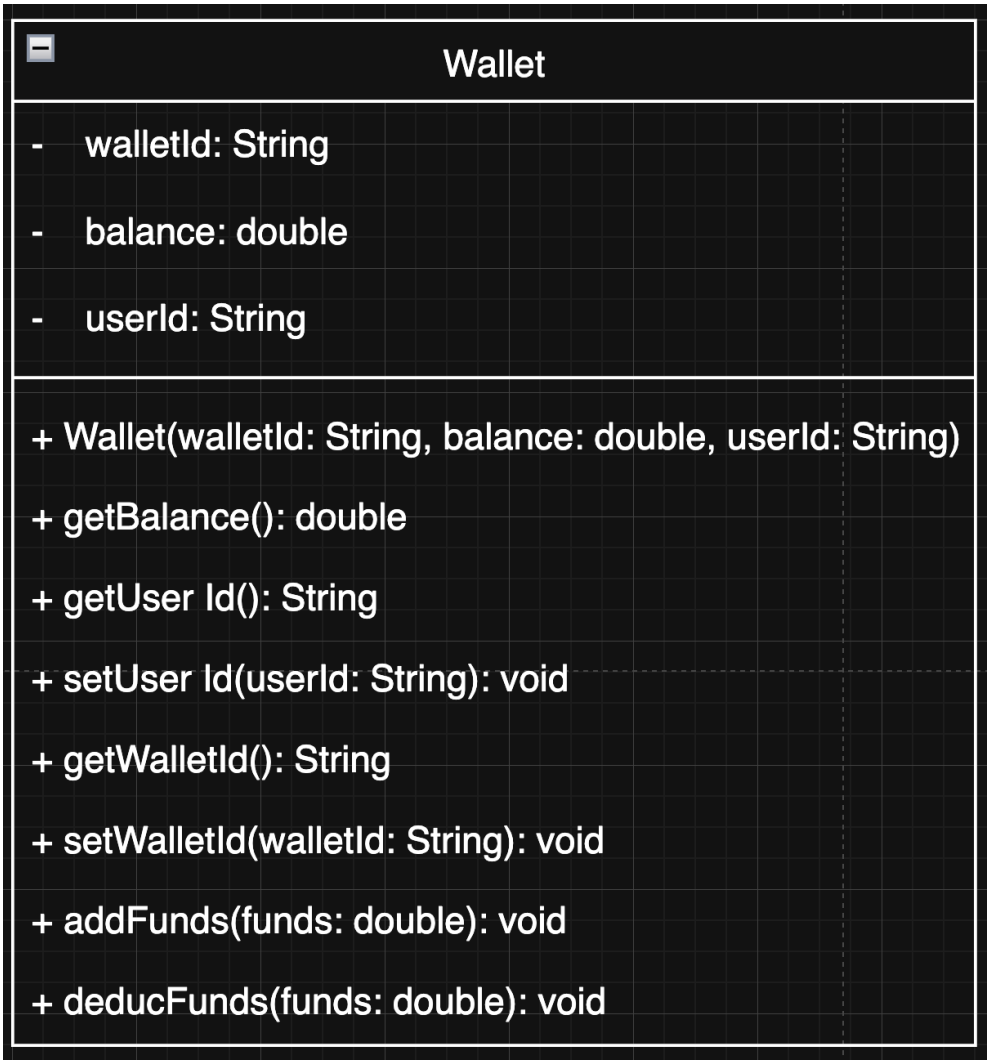


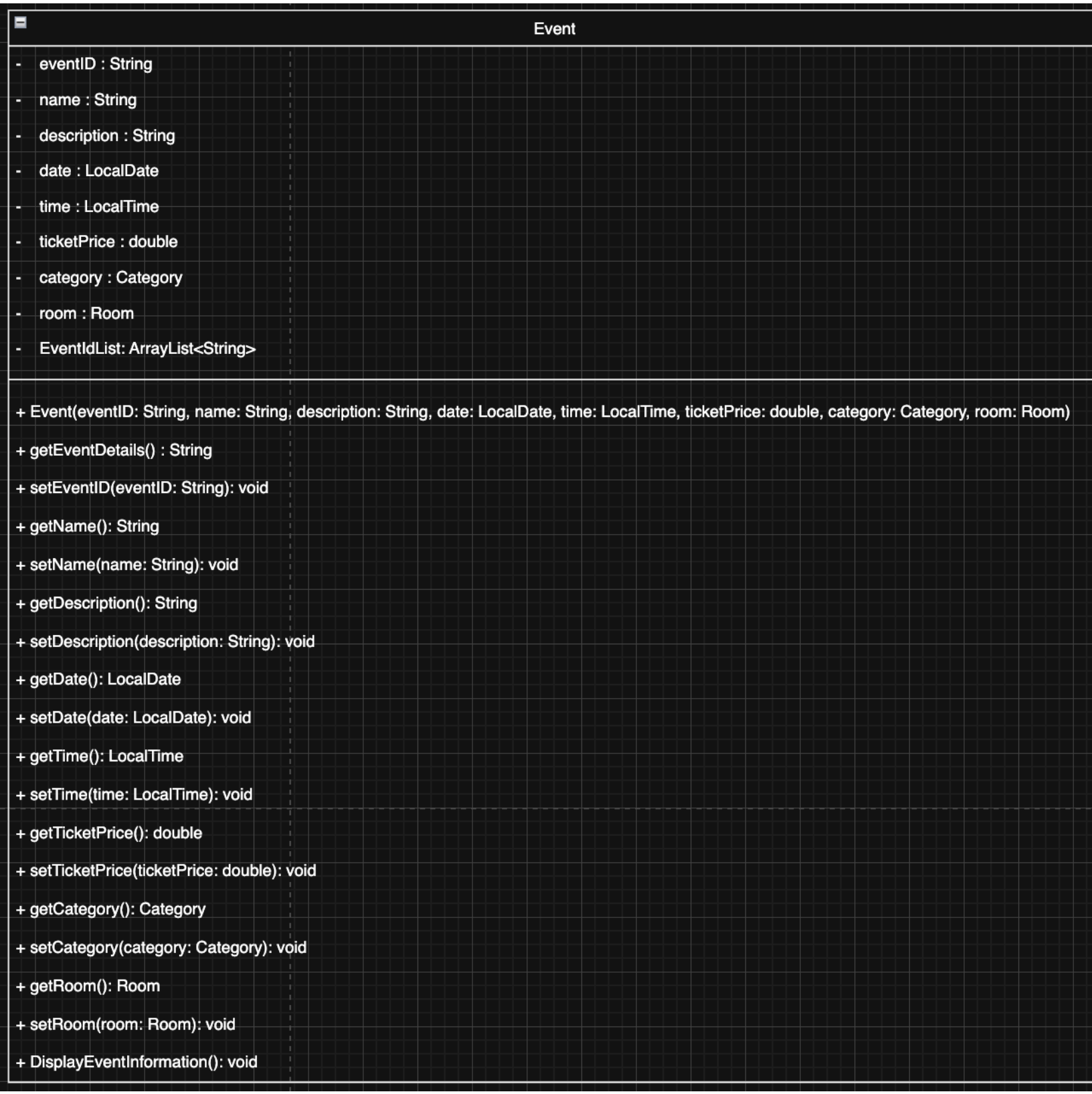
User abstract parent class

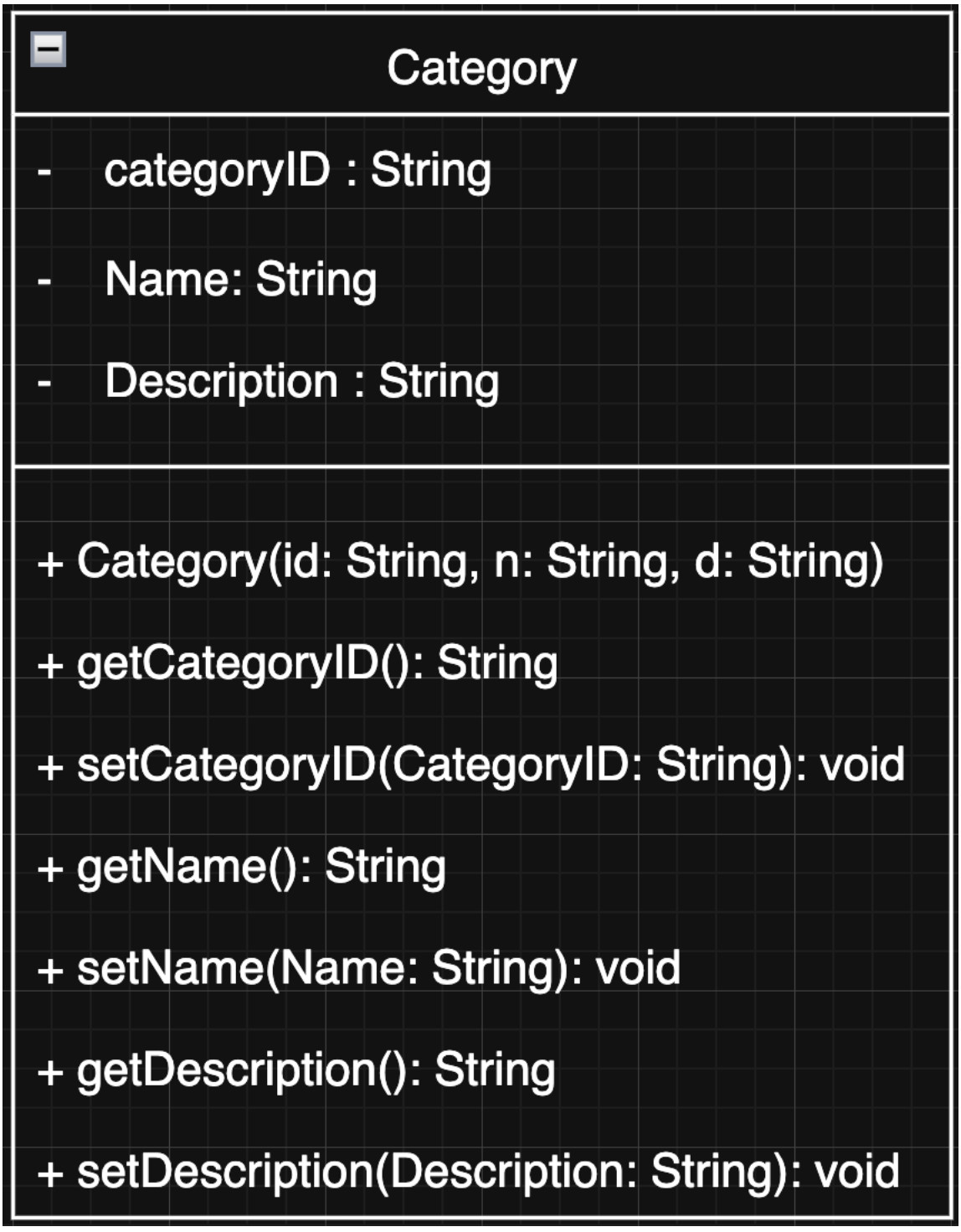

Admin
- role : String
- workingHours : double
+ Admin(u: String, p: String, d: DateOfBirth, r: String, w: double)
+ register(db: Database): void
+ adminMenu(db: Database): void
+ viewAttendee(db: Database): void
+ viewRooms(db: Database): void
+ viewCategories(db: Database): void
+ addRoom(db: Database): void
+ manageCategory(db: Database): void
+ createCategory(db: Database): void
+ deleteCategory(db: Database): void
+ updateCategory(db: Database): void

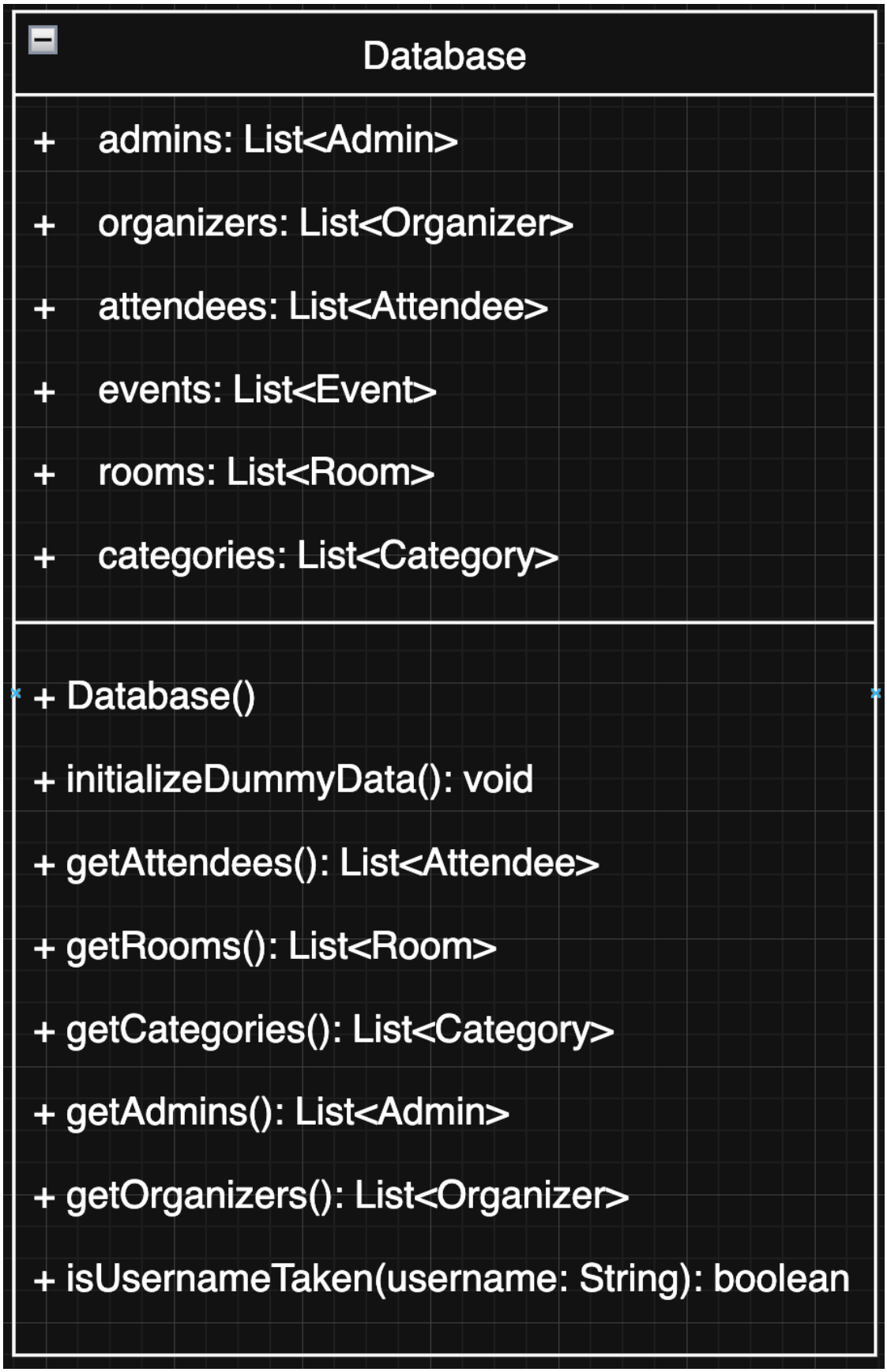

Organizer
- registeredOrganizers: List<Organizer>
- createdEvents: List<Event>
- roomBookings: Map<Room, List<TimeSlot>>
+ Organizer(username: String, password: String, dateOfBirth: LocalDate)
+ register(): boolean
+ registerInDatabase(db: Database): void
+ showDashboard(): void <<override>>
+ menu(db: Database): void
+ createEvent(eventId: String, name: String, description: String, date: LocalDate, time: LocalTime, ticketPrice: double, category: Category, room: Room): void
+ updateEvent(eventId: String, newName: String, newDescription: String, newDate: LocalDate, newTime: LocalTime, newTicketPrice: double, newCategory: Category, newRoom: Room): void
+ cancelEvent(eventId: String): void
+ bookRoom(room: Room, slot: TimeSlot): boolean
+ viewRoomDetails(room: Room): void
- hasBookingConflict(room: Room, newSlot: TimeSlot): boolean

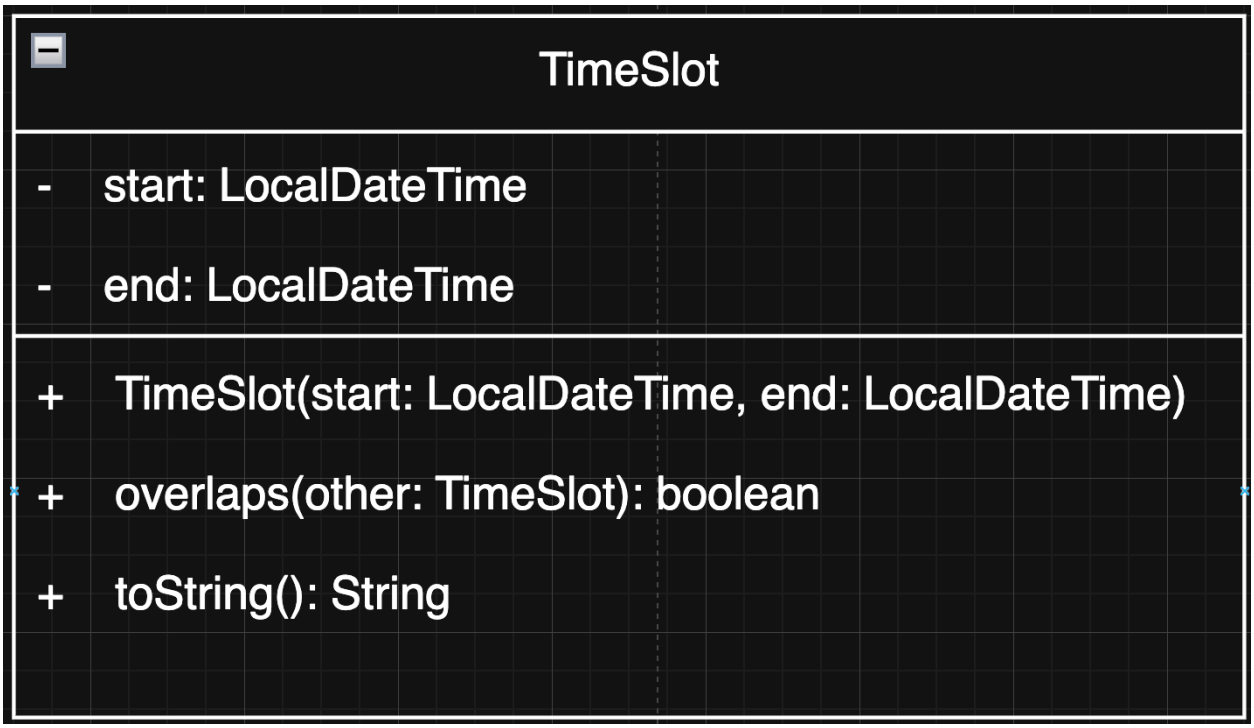

Attendee
- address : String
- balance : double
- interests : String
- gender : Gender
+ Attendee(u: String, p: String, d: DateOfBirth, address: String, balance: double, interests: String, gender: Gender)
+ getBalance(): double
+ setBalance(balance: double): void
+ getAddress(): String
+ setAddress(address: String): void
+ getInterests(): String
+ setInterests(interests: String): void
+ getGender(): Gender
+ setGender(gender: Gender): void
+ register(db: Database): void
+ BookEvent(): void

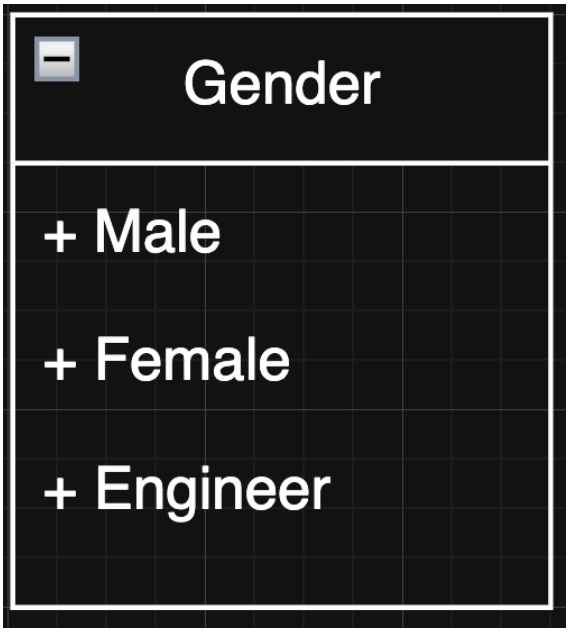

Wallet
- walletId: String
- balance: double
- userId: String
+ Wallet(walletId: String, balance: double, userId: String)
+ getBalance(): double
+ getUser Id(): String
+ setUser Id(userId: String): void
+ getWalletId(): String
+ setWalletId(walletId: String): void
+ addFunds(funds: double): void
+ deducFunds(funds: double): void

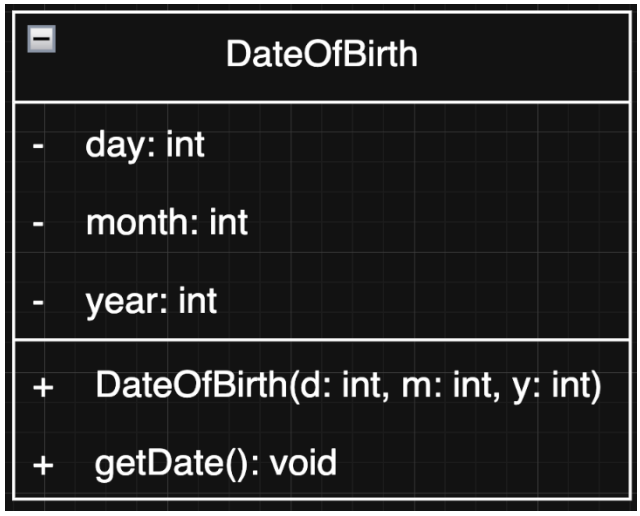

Event
- eventID : String
- name : String
- description : String
- date : LocalDate
- time : LocalTime
- ticketPrice : double
- category : Category
- room : Room
- EventIdList: ArrayList<String>
+ Event(eventID: String, name: String, description: String, date: LocalDate, time: LocalTime, ticketPrice: double, category: Category, room: Room)
+ getEventDetails() : String
+ setEventID(eventID: String): void
+ getName(): String
+ setName(name: String): void
+ getDescription(): String
+ setDescription(description: String): void
+ getDate(): LocalDate
+ setDate(date: LocalDate): void
+ getTime(): LocalTime
+ setTime(time: LocalTime): void
+ getTicketPrice(): double
+ setTicketPrice(ticketPrice: double): void
+ getCategory(): Category
+ setCategory(category: Category): void
+ getRoom(): Room
+ setRoom(room: Room): void
+ DisplayEventInformation(): void


Category
- categoryID : String
- Name: String
- Description : String
+ Category(id: String, n: String, d: String)
+ getCategoryID(): String
+ setCategoryID(CategoryID: String): void
+ getName(): String
+ setName(Name: String): void
+ getDescription(): String
+ setDescription(Description: String): void


Database
+ admins: List<Admin>
+ organizers: List<Organizer>
+ attendees: List<Attendee>
+ events: List<Event>
+ rooms: List<Room>
+ categories: List<Category>
+ Database()
+ initializeDummyData(): void
+ getAttendees(): List<Attendee>
+ getRooms(): List<Room>
+ getCategories(): List<Category>
+ getAdmins(): List<Admin>
+ getOrganizers(): List<Organizer>
+ isUsernameTaken(username: String): boolean


TimeSlot
- start: LocalDateTime
- end: LocalDateTime
+ TimeSlot(start: LocalDateTime, end: LocalDateTime)
+ overlaps(other: TimeSlot): boolean
+ toString(): String


Gender
+ Male
+ Female
+ Engineer


DateOfBirth
- day: int
- month: int
- year: int
+ DateOfBirth(d: int, m: int, y: int)
+ getDate(): void

3. Class Explanation:

3.1. User Class:

The User class is an abstract class that serves as the base for different types of users in an event management system, such as Admin, Attendee, and Organizer. It encapsulates common user attributes, including a username, password, and dateOfBirth, which are initialized through a constructor.

The class provides methods to retrieve the username and password but does not allow modifications, ensuring data integrity. The class also includes static methods for user authentication and registration. The login() method prompts users for credentials, checks them against stored user data in a Database object, and grants access to role-specific menus upon successful validation. If login fails, users can retry or return to the main menu. The registerMenu() method allows new users to register based on their role, while the mainMenu() method serves as the entry point, offering options to log in, register, or exit the system.

The class handles input validation to prevent errors from invalid user inputs. Overall, the User class provides a structured and secure way to manage user interactions within the application.

3.2. Attendee Class:

The Attendee class represents a regular user who participates in events within the system. It inherits from the User class and includes additional personal attributes such as address, balance, interests, and gender, each with corresponding getter and setter methods for encapsulated access and modification. The constructor initializes all these fields when a new Attendee object is created.

A key feature of this class is the register method, which enables new attendees to sign up through console input. It performs checks for valid input values such as unique usernames, secure passwords (minimum 8 characters), a positive balance, and a valid date of birth. It also collects other attendee details like address, interests, and gender—though the implementation contains syntax and logical errors (such as incorrect scanner usage, improper handling of gender selection, and uninitialized variables) that would need to be fixed for proper functionality.

Additionally, the class includes a placeholder method BookEvent that allows attendees to book events by selecting from a list of event IDs. However, this method is currently incomplete and lacks full logic for interacting with actual event data. Despite its flaws, the Attendee class is structured to handle user registration and interaction, forming an essential part of the system for managing event participants.

3.3. Admin Class:

The Admin class extends the User class and represents administrative users within the event management system. It introduces two specific fields: role, which typically holds the value "Admin", and workingHours, which tracks the administrator's assigned working time. The class includes a static method register that facilitates admin account creation by collecting user input for username, password, birthdate, and working hours. It incorporates input validation, such as ensuring unique usernames and enforcing password and date constraints.

Once registered, admins can access the adminMenu, which provides various administrative functionalities. These include adding new rooms (addRoom), viewing attendees, rooms, and categories (viewAttendee, viewRooms, and viewCategories), and managing event categories through the manageCategory method. Within category management, admins can create (createCategory), delete (deleteCategory), and (potentially) update categories, though the updateCategory method is currently unimplemented. The class relies heavily on the Database object to store and retrieve data, ensuring that administrative actions are reflected across the system. This structure makes the Admin class a crucial component for maintaining and organizing the system’s resources.

3.4. Organizer Class:

The Organizer class is a key part of the Event Management System and represents users who are responsible for managing events. It inherits from the base User class, thereby gaining common attributes like username, password, and date of birth.

This class maintains two static lists: one for all registered organizers and another for all events created by organizers. These lists serve as a storage system for managing organizer accounts and events within the system.

The organizer has the ability to register themselves, which involves adding their instance to the list of registered organizers. Upon successful registration, a confirmation message is displayed. In case of an error during the process, an appropriate failure message is shown.

Organizers are provided with a dashboard that displays a menu of available actions. These actions include “Create Event”, “Update Event”, “View Available Rooms”, and “Rent Room”.

To create an event, the organizer provides details such as event ID, name, description, date, time, ticket price, category, and the room to be used. If all inputs are valid, a new event is created and added to the list of created events.

The update event feature allows the organizer to modify an existing event’s details using the event ID. If the event is found, its attributes are updated by information given by the organizer.

The cancel event functionality lets the organizer remove an event from the system by specifying its event ID. If the event exists, it is removed from the list of created events, and a confirmation message is displayed.

The room booking and management functionalities are a fundamental part of the Event Management System, responsible for ensuring that rooms are

reserved in a conflict-free and organized manner. These operations are implemented through a series of methods that handle booking validation, actual reservation, and viewing room details.

The conflict-checking mechanism is implemented through a private method that determines whether a new time slot overlaps with any previously booked slots for a given room. This method retrieves the list of existing bookings for the specified room and checks each one for any time conflict with the new slot. If an overlap is detected, it returns true, signaling that the new booking request would cause a scheduling conflict.

To handle the actual booking process, a public method is provided which allows rooms to be reserved for specific time slots. Before making any reservations, the method first ensures that the room has an entry in the internal booking registry. It then calls the conflict-checking method to validate that the requested time slot does not interfere with any existing bookings. If no conflicts are found, the booking is added to the room's list of reservations, and the method returns true to indicate that the booking was successful. In the event of a conflict, the method simply returns false, preventing the reservation and alerting the caller that the slot is already occupied.

The system also includes a method to view detailed information about a room. This method outputs the room's ID, its capacity, and a list of its available amenities. It also displays the number and details of any time slots that have already been booked for that room. This comprehensive overview helps organizers quickly assess the availability and suitability of a room before making a booking decision.

Each method in the class includes basic error handling to catch any runtime exceptions and provide user-friendly feedback.

3.5. Event Class:

The Event class represents the core entity within the Event Management System, encapsulating all the relevant details of an event. It is used by organizers to create, manage, and display information about events.

This class includes several private attributes that define the essential properties of an event. These properties include the event ID, name, description, date, time, ticket price, event category, and the room where the event will be held. The use of encapsulation ensures that the internal state of the event is protected and accessed only through public getter and setter methods.

Each attribute has a corresponding getter and setter method, which allows for controlled access and modification of the event's data.

The class also includes a method to display event information, which prints out all the key details of the event in a user-friendly format. This includes the event's name, description, date, time, ticket price, category name, and room ID.

The Event class works closely with other classes such as Category and Room, demonstrating composition — a fundamental principle of object-oriented programming. This design ensures modularity and reusability within the system.

3.6. Room Class:

The Room class serves as a fundamental component within the event management system, functioning as a data structure that encapsulates all relevant information about physical event venues. Designed with simplicity and maintainability in mind, this class focuses exclusively on storing and managing the static properties of event spaces.

This class maintains three primary attributes that define a venue's characteristics. The room field provides a unique identifier for each venue, enabling easy reference and differentiation between spaces. The capacity integer specifies the maximum number of attendees the room can accommodate, a critical factor for event planning and safety compliance. Perhaps most importantly, the amenities list catalogs all available facilities and equipment within the space, such as projectors, sound systems, or specialized lightning, which helps organizers match venues to event requirements.

The class implements careful access control through its method design. While getter methods allow external components to read room specifications, the absence of corresponding setters for the roomId and capacity fields ensures these fundamental properties remain immutable after instantiation. This design choice promotes data integrity and prevents accidental modifications to venue's core characteristics. The sing exception to this immutability is the addAmentiy() method, which provides a controlled mechanism for updating a rooms' facilities preventing duplicate entries.

Thes streamlined design of the Room class offers several advantages for system evolution and maintenance. Its limited scope makes the class exceptionally stable, reducing the likelihood of bugs or unintended side effects when other system components change.

3.7. Category Class:

The Category class is a supportive component in the Event Management System that helps classify and organize events based on their type or nature.

This class contains three primary attributes: category ID, the name of the category, and a description. These attributes help define what the category represents and distinguish it from other categories in the system.

The class follows the principles of encapsulation, where all attributes are marked private and are accessed or modified through public getter and setter methods. This approach ensures that category information is updated in a controlled manner.

The category object is associated with an event during event creation, allowing events to be grouped or sorted by their type. This association is useful for both system administrators and attendees.

3.8. Database Class:

The Database class functions as the core in-memory storage system for an event management application, maintaining collections of different user types (Admin, Organizer, Attendee), as well as Event, Room, and Category objects. Upon creation, it initializes these collections and populates them with sample data using the initializeDummyData() method to simulate a functional environment. This includes adding sample users, rooms with specific features, and a category for classification. The class also provides getter methods for accessing each list, allowing other components of the system to retrieve and use this data. Additionally, it includes a utility method isUsernameTaken, which checks if a given username is already in use across all user types, helping to enforce unique usernames and avoid conflicts during registration. This class plays a vital role in managing and organizing application data efficiently without relying on a persistent database during development or testing.

3.9. Wallet Class:

The Wallet class represents a digital wallet associated with a specific user in the system. It contains three main fields: walletId, a unique identifier for the wallet; balance, a double value representing the current amount of funds available; and userId, which links the wallet to a particular user. The constructor initializes these attributes when a new wallet is created. The class provides getter and setter methods for all three fields, allowing controlled access and updates. Additionally, it includes two key methods: addfunds(double funds) to increase the wallet’s balance and deducfunds(double funds) to subtract a specified amount from it. It's important to note that deducfunds() does not currently check if the balance is sufficient before deducting, which could lead to negative values if not handled elsewhere. Overall, this class offers basic functionality to manage a user's virtual balance in scenarios like bookings, refunds, or account crediting

3.10. Date Of Birth Class:

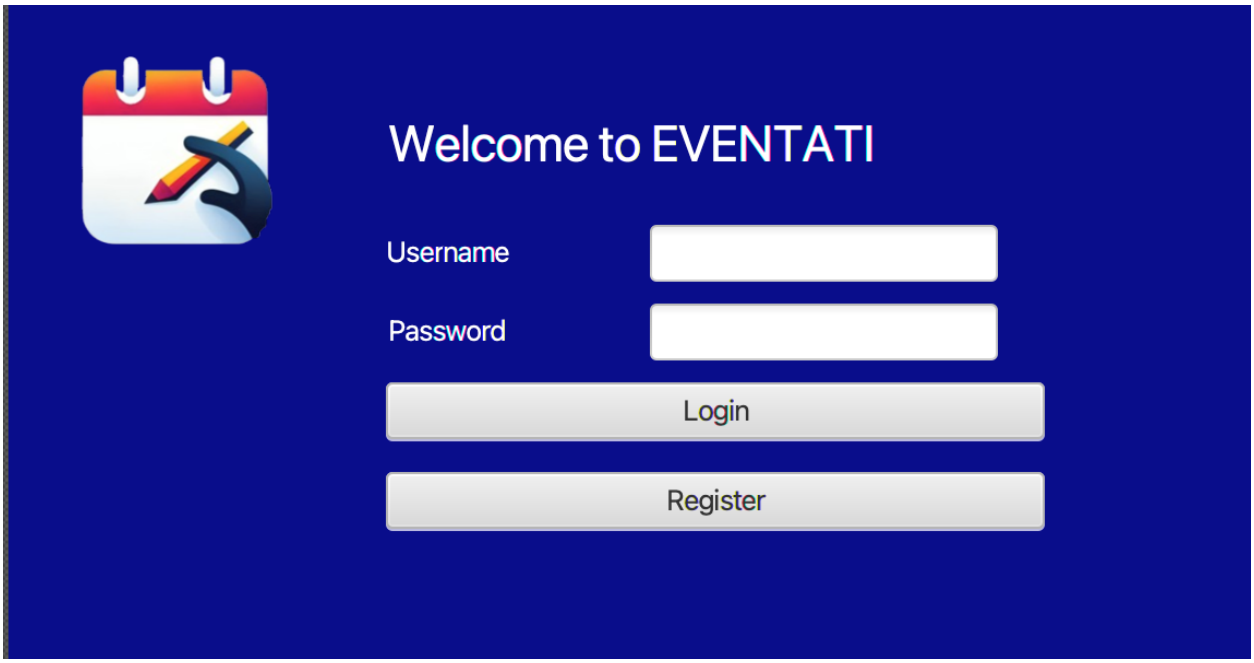
The DateOfBirth class is a simple data model that stores a person's date of birth using three integer fields: day, month, and year. It includes a constructor that initializes these fields based on the values provided during object creation. The class also defines a method named getDate, which prints the date of birth in the standard day/month/year format. This class can be used in applications where storing and displaying a user's date of birth is necessary, such as in user profile management systems or registration forms.

3.11. Time Slot Class:

The TimeSlot class represents a specific time interval, defined by a start and end time using LocalDateTime. It is designed to model time periods and provides functionality to determine if two time slots overlap. The constructor initializes the start and end attributes, ensuring that each TimeSlot instance has a clearly defined beginning and end. The overlaps method checks if there is any intersection between the current time slot and another by verifying that one does not end before the other starts, or start after the other ends. Additionally, the class includes a toString method that formats the time slot as a readable string, displaying the date and time range in a simplified format. This class is particularly useful in scheduling systems where detecting conflicts between time periods is crucial.

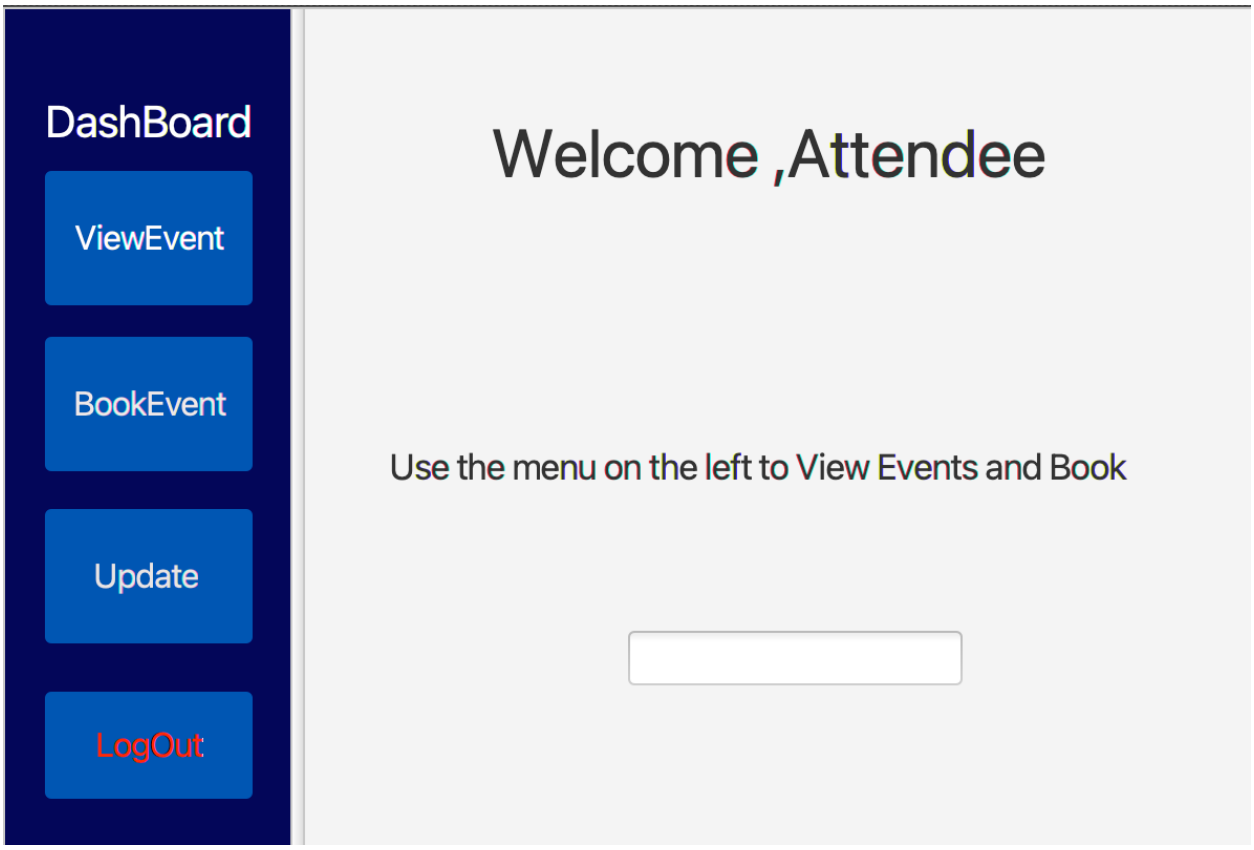
The GUI

first the login window that determines which scene will come up next



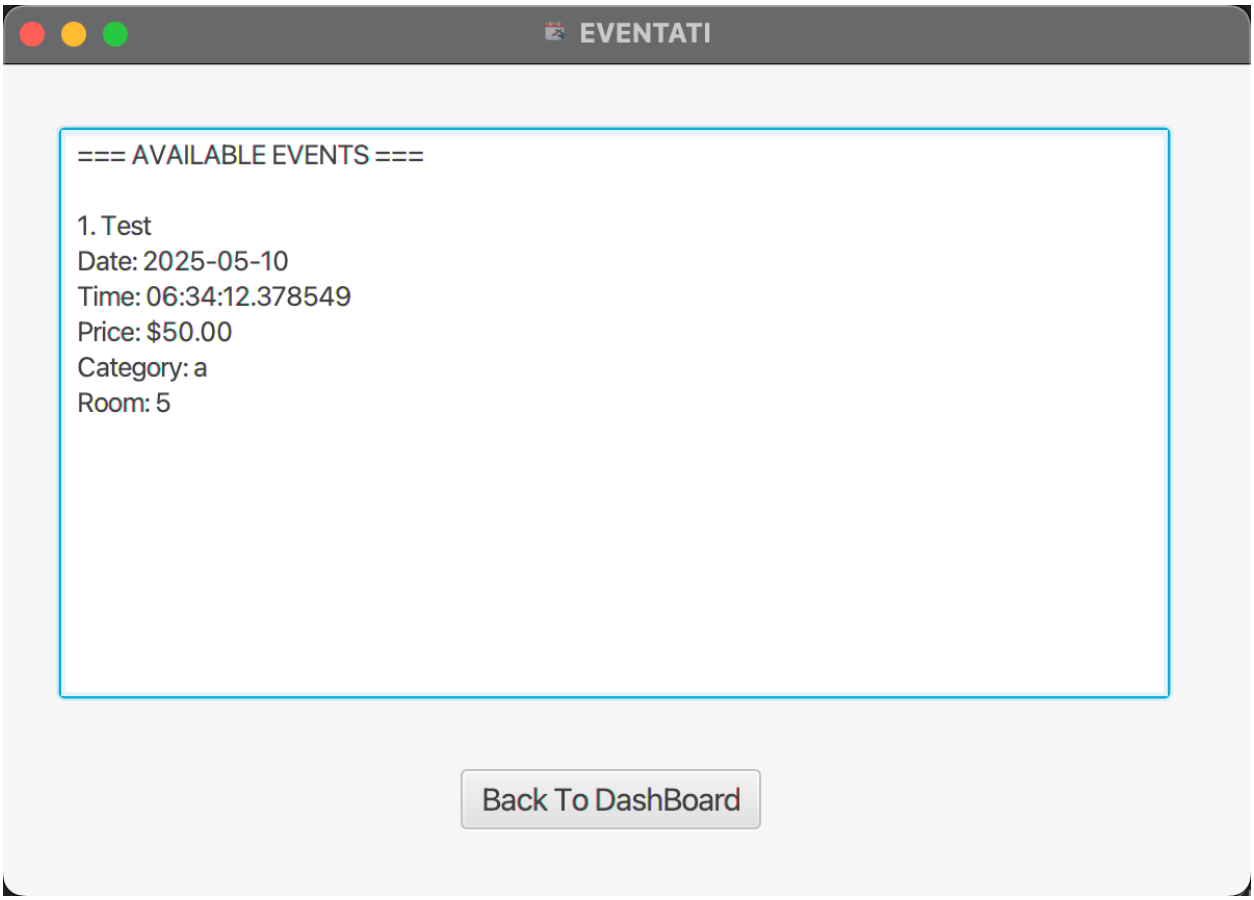
Attendee Menu

If the User logs in with the Attendee credentials the Attendee menu comes up that shows all the function they can use and shows there balance

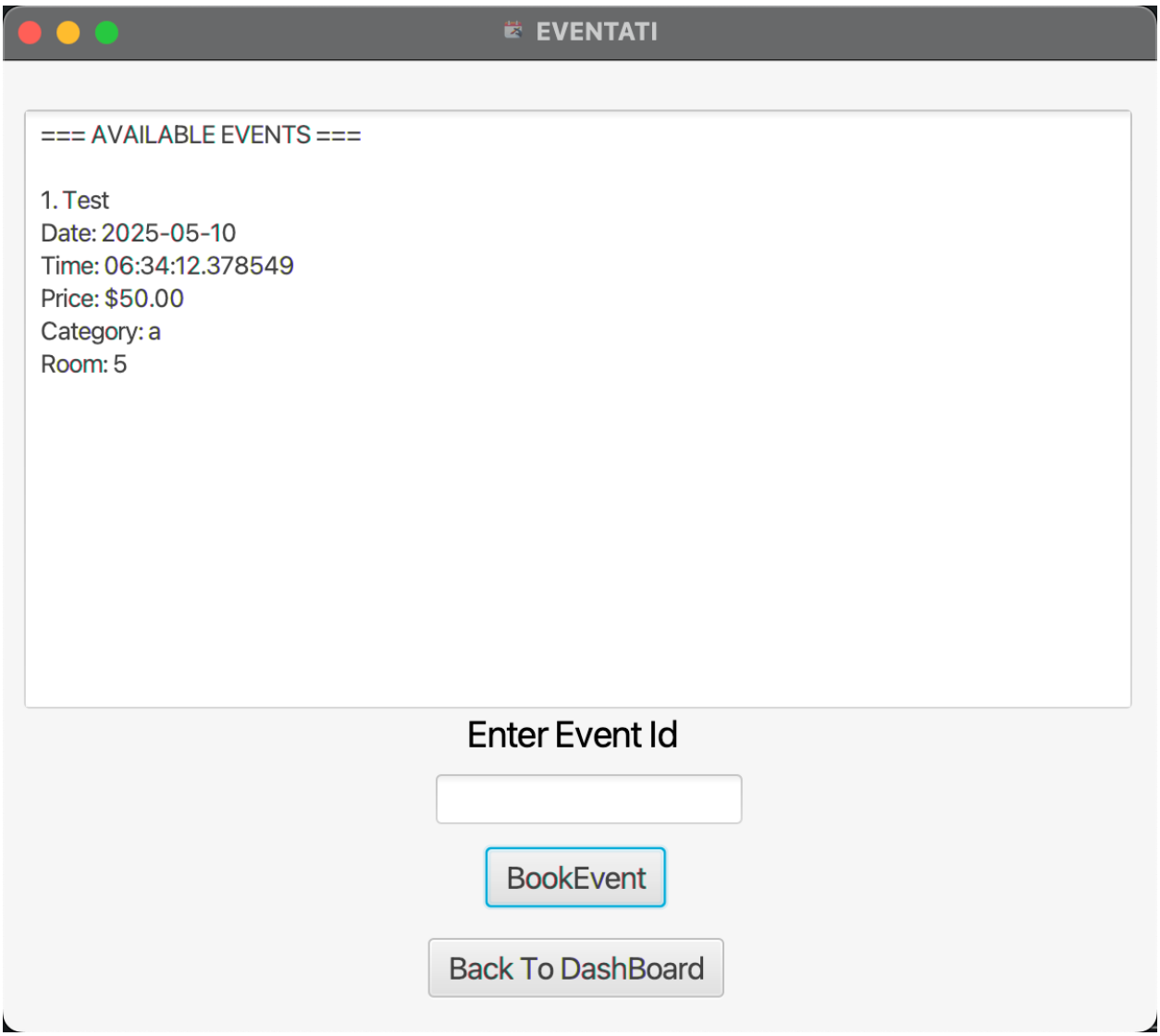


Inside the Attendee Menu

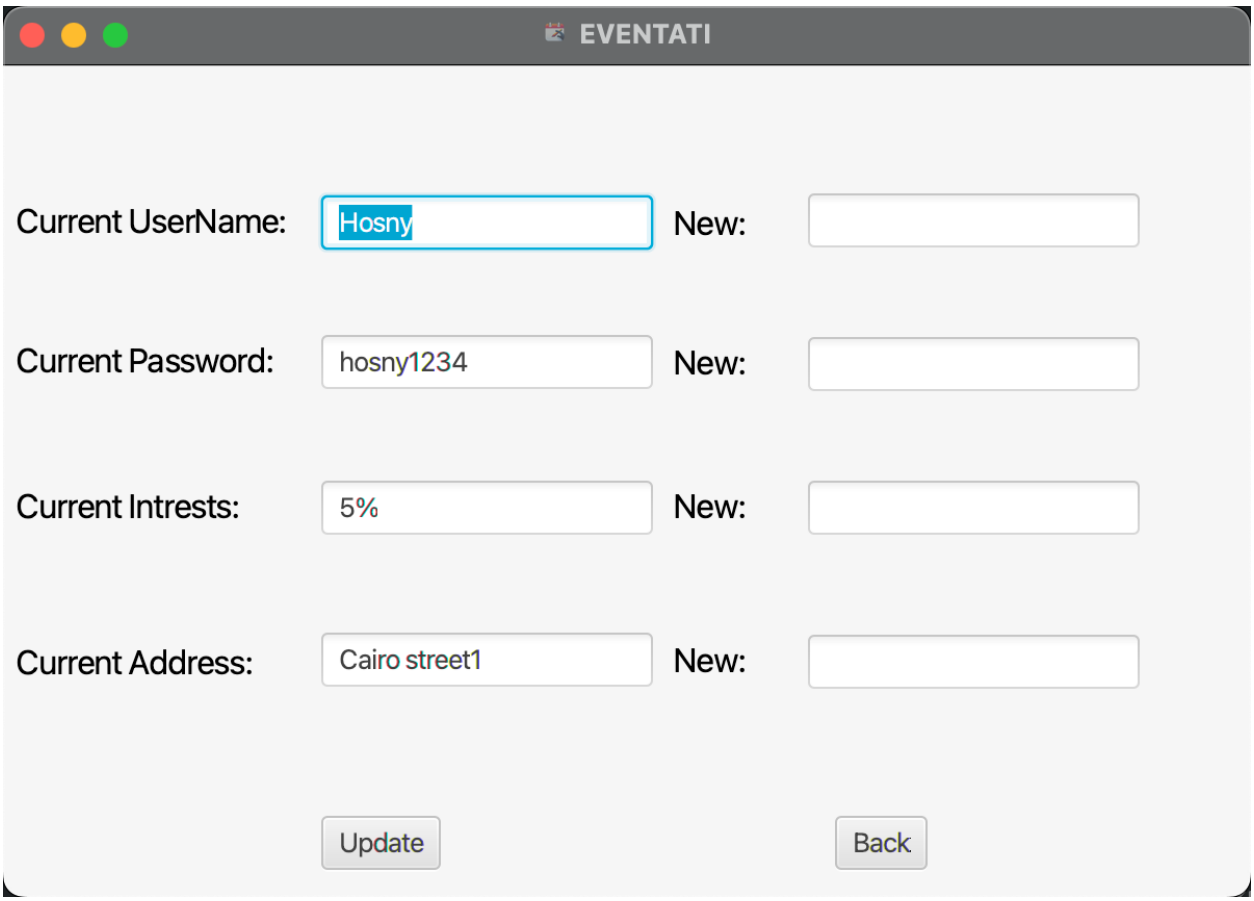
ViewEvent



BookEvent

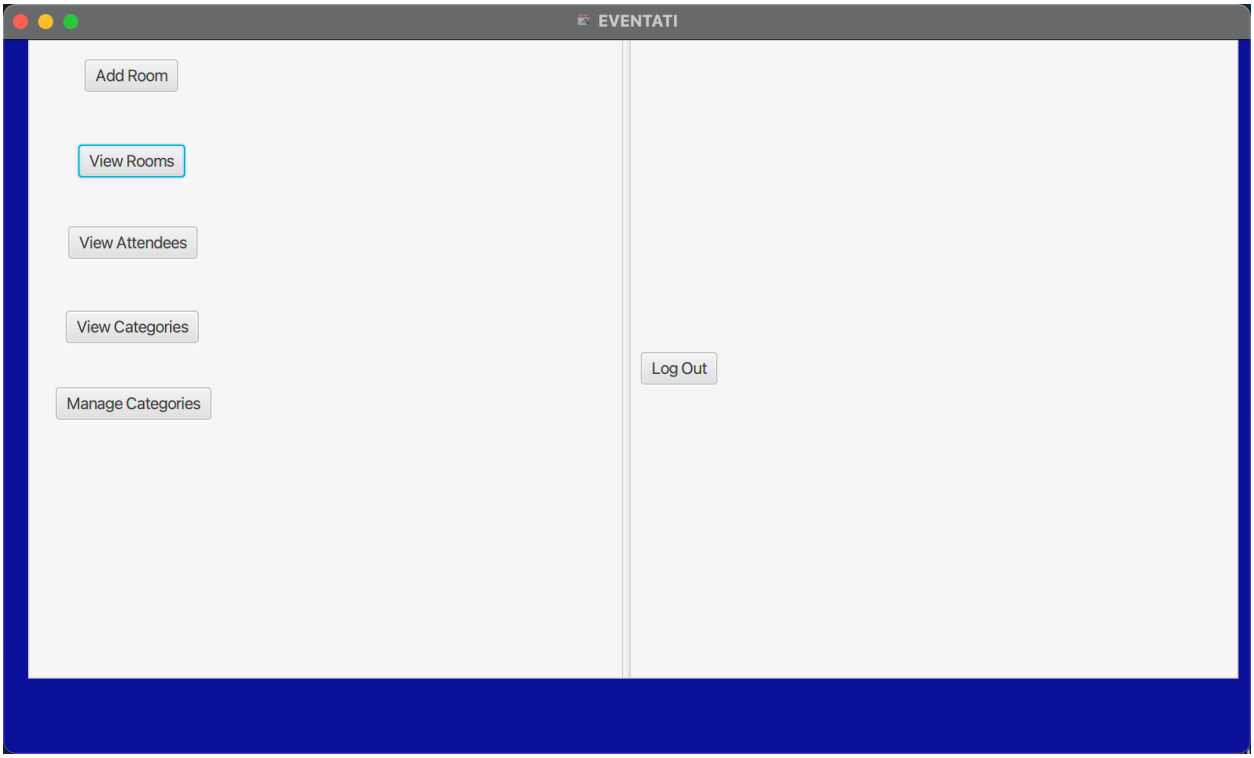

EVENTATI
=== AVAILABLE EVENTS ===
1. Test
Date: 2025-05-10
Time: 06:34:12.378549
Price: $50.00
Category: a
Room: 5
Enter Event Id
BookEvent
Back To DashBoard

Update

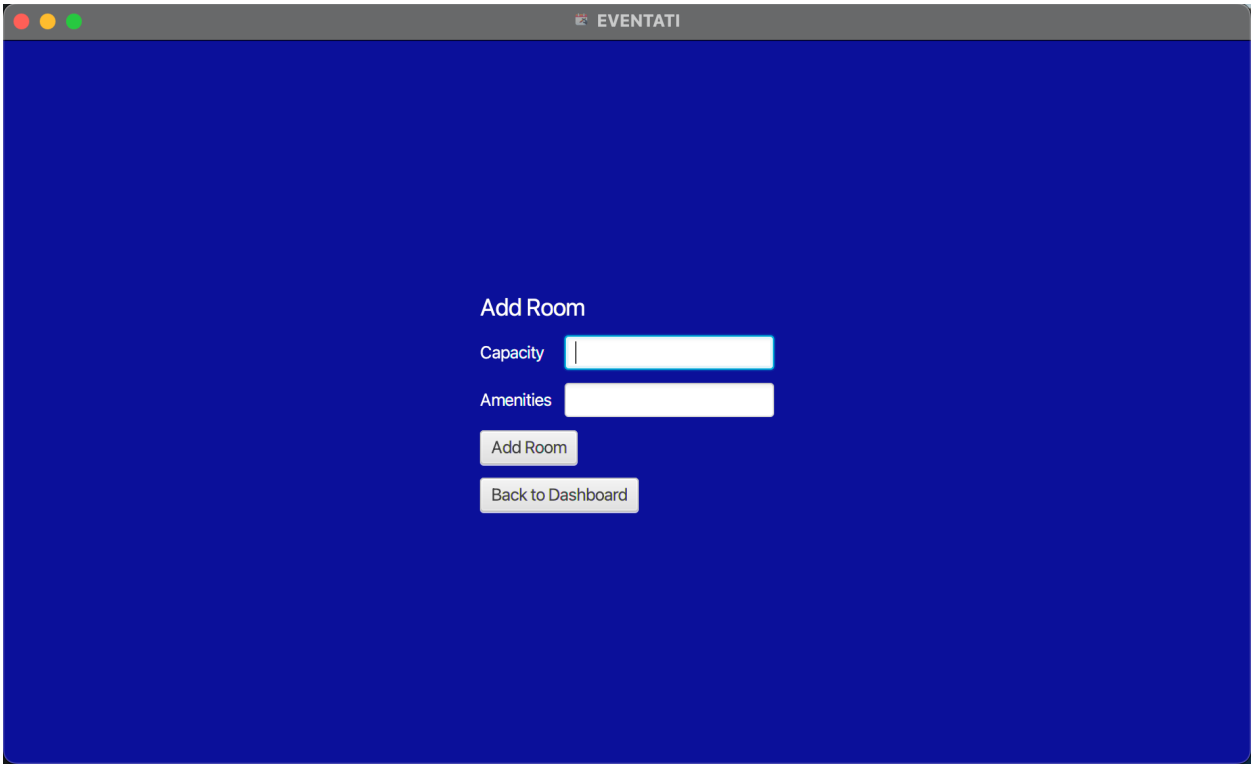

EVENTATI
Current UserName:
Hosny
New:
Current Password:
hosny1234
New:
Current Intrests:
5%
New:
Current Address:
Cairo street1
New:
Update
Back

Admin Menu

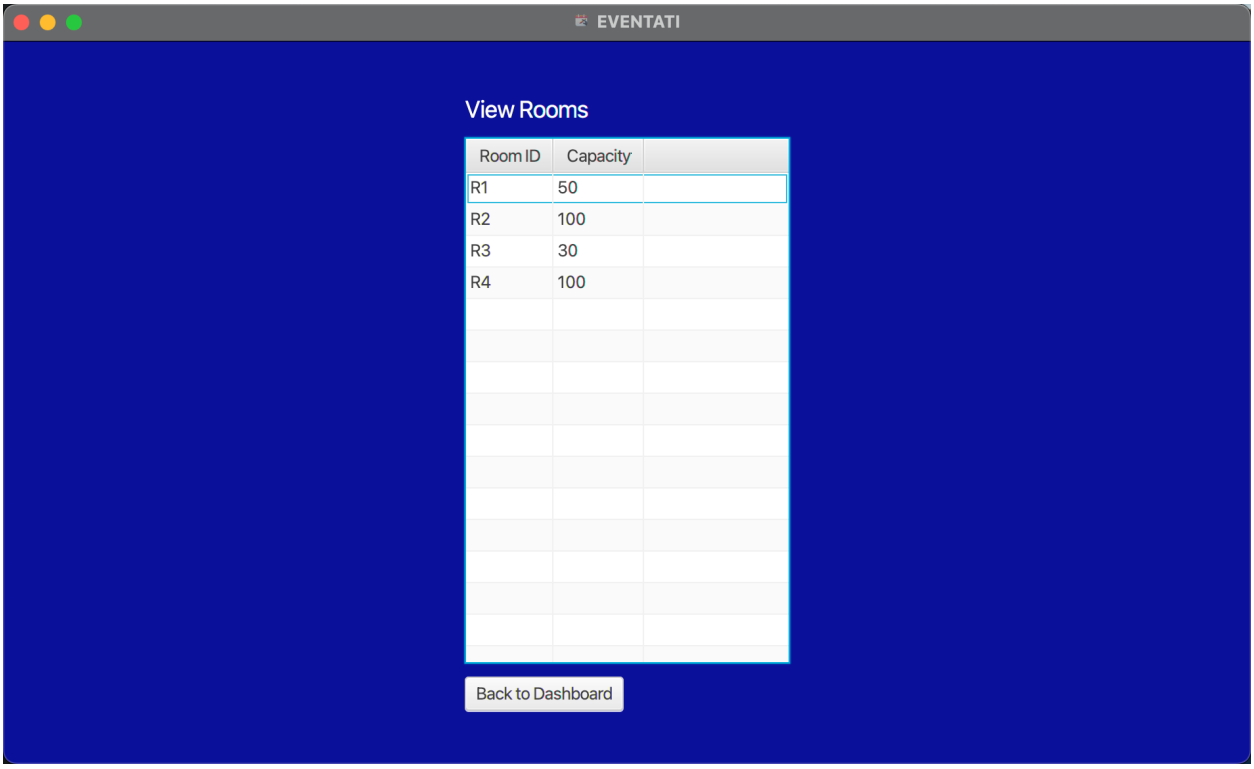
If the User logs in with the Admin credentials the Attendee menu comes up that shows all the function they can use



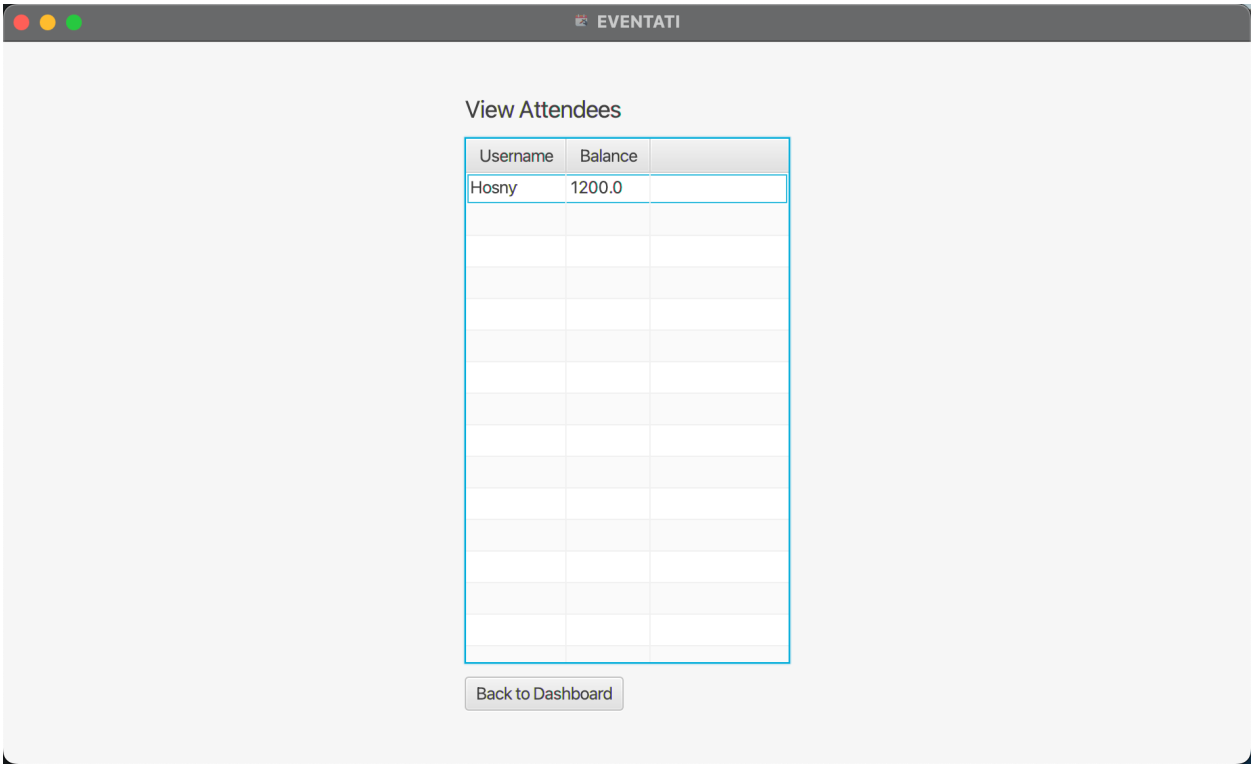
Add Room


EVENTATI
Add Room
Capacity
Amenities
Add Room
Back to Dashboard

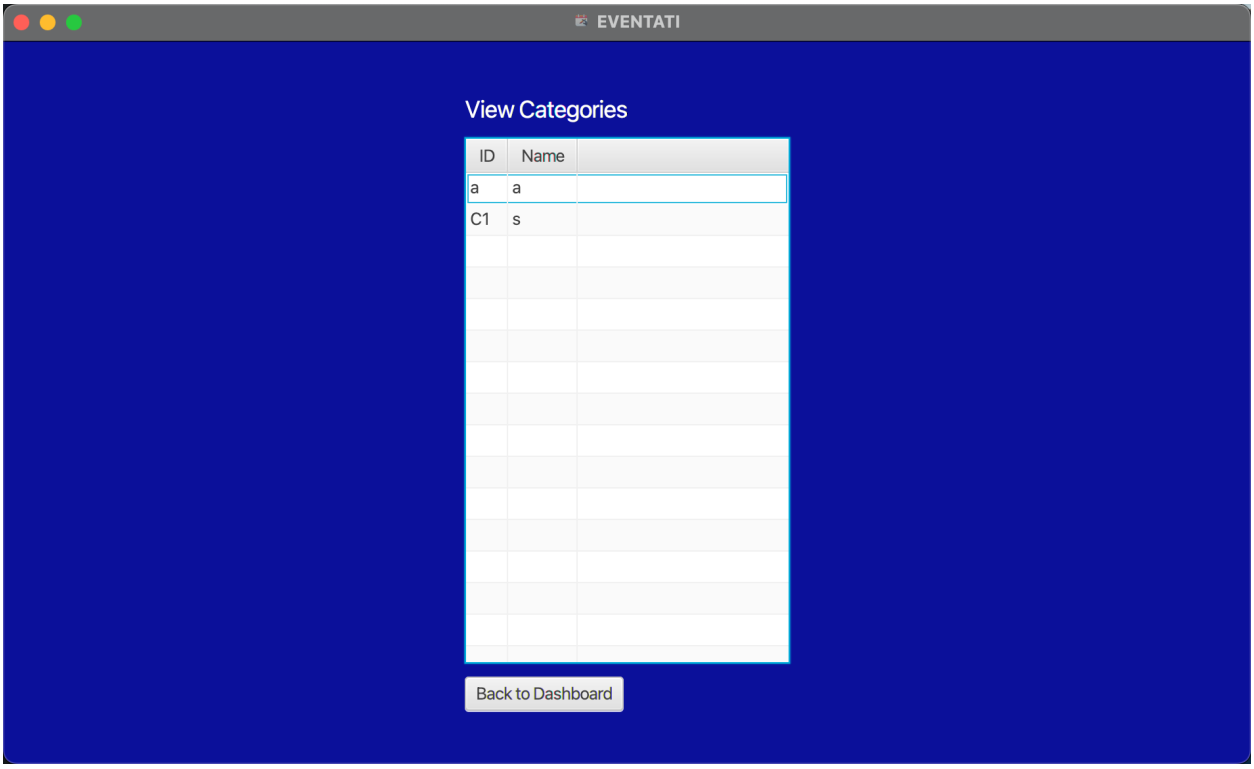
View Rooms


EVENTATI
View Rooms
Room ID
Capacity
R1
50
R2
100
R3
30
R4
100
Back to Dashboard

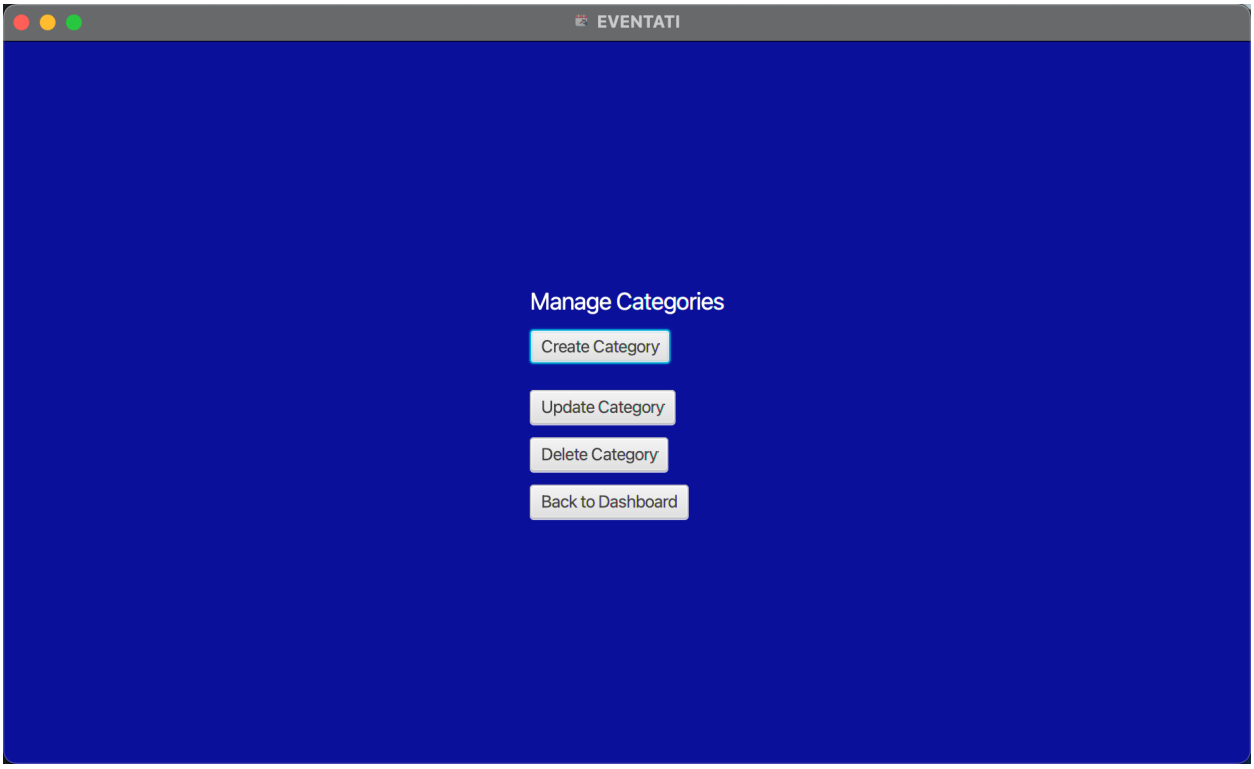
View Attendees



View Categories

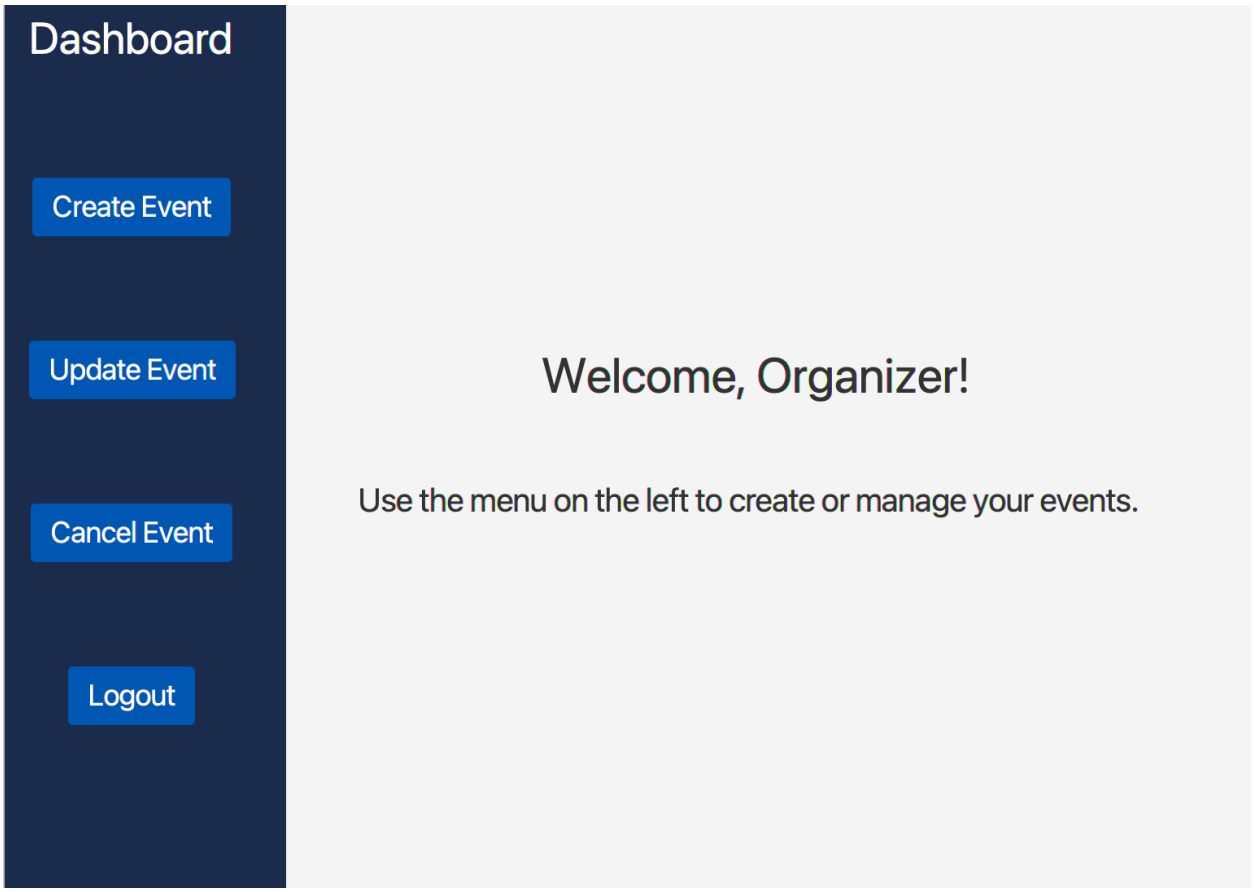


Manage Categories



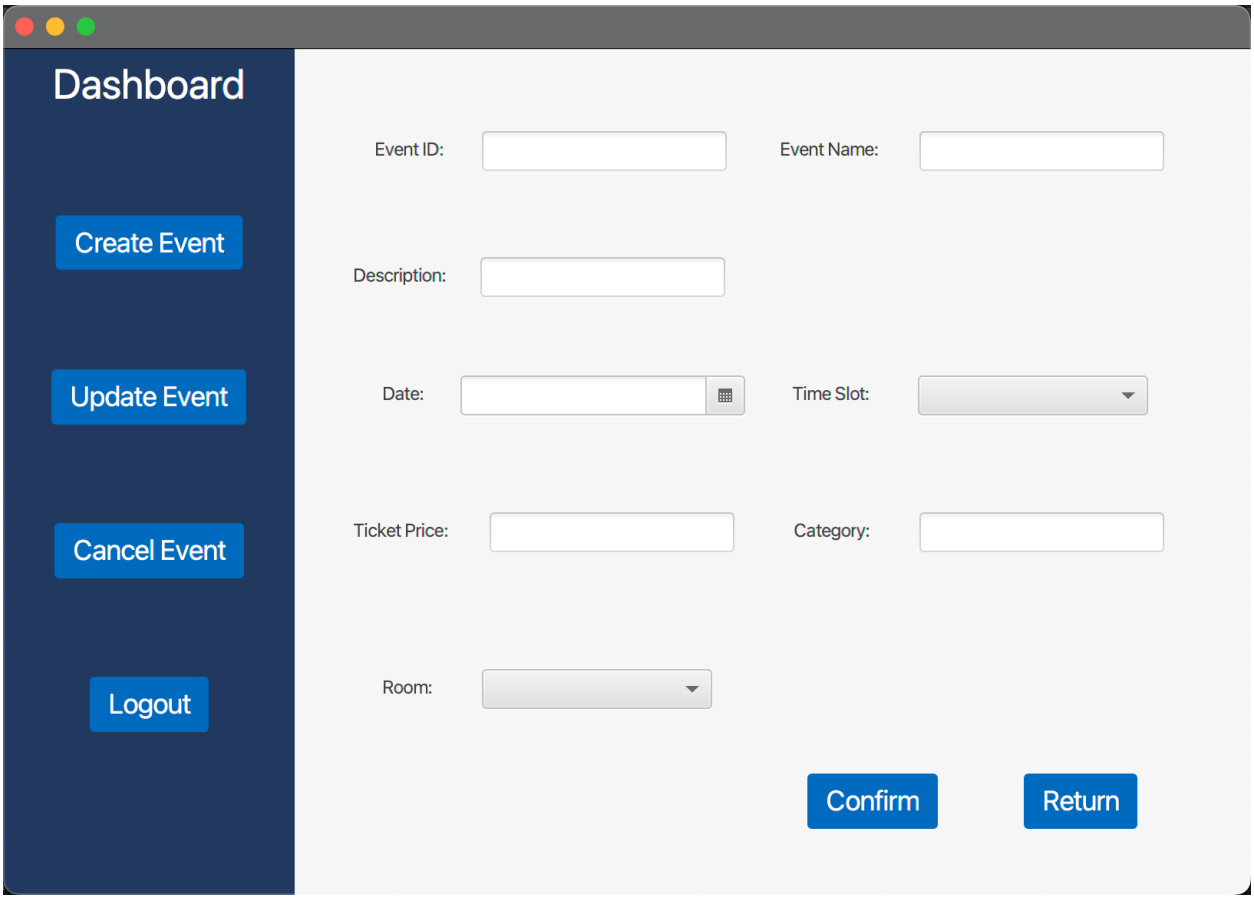
Organizer Menu

If the User logs in with the Admin credentials the Attendee menu comes up that shows all the function they can use

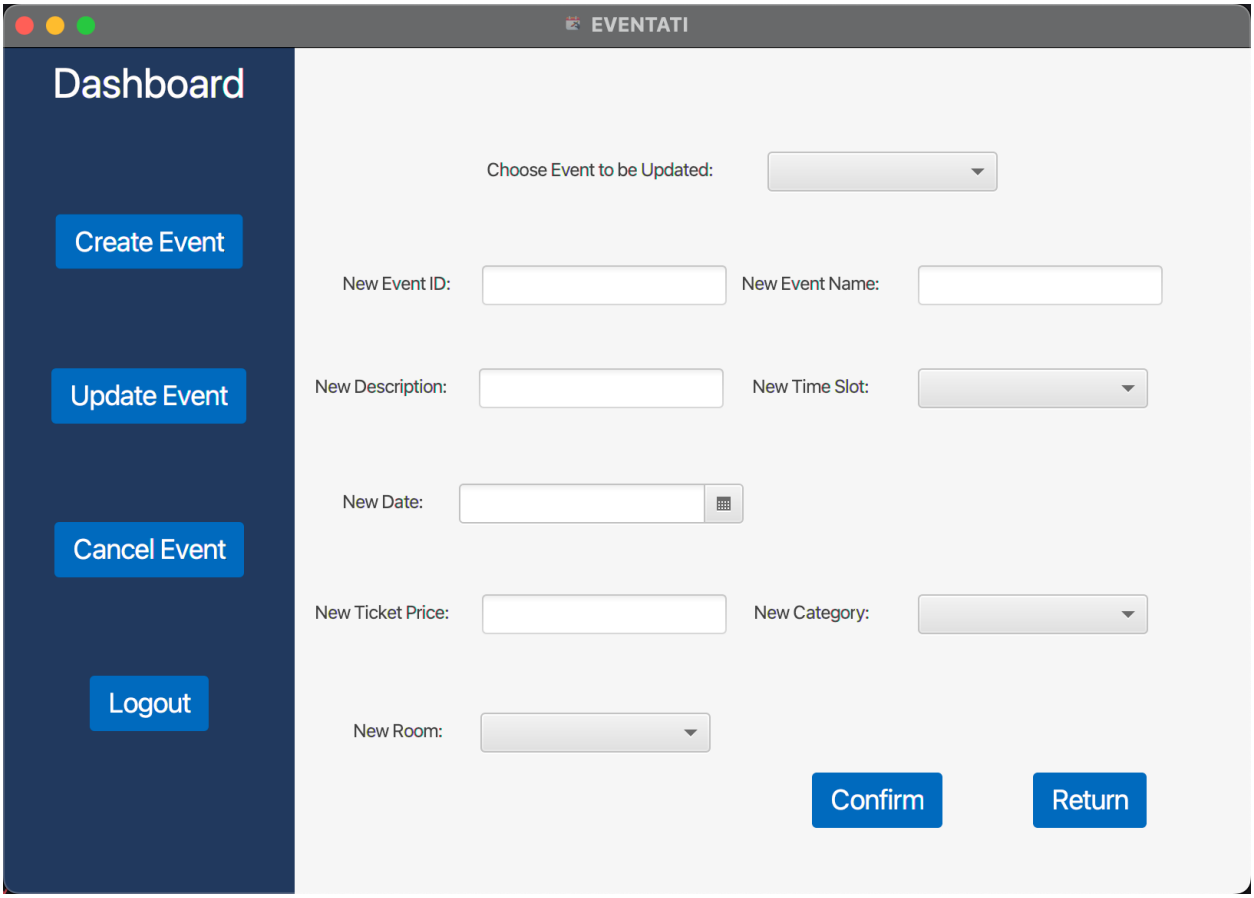


Inside the Organizer Menu

Create Event



Update Event



Cancel Event