UNIVERSITY OF PIRAEUS – DEPARTMENT OF INFORMATICS
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Postgraduate Program "Informatics"
Πρόγραμμα Μεταπτυχιακών Σπουδών «Πληροφορική»

MSc Thesis
Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**"Event Management and Ticketing Platform with Java Spring Boot"** «Σύστημα Διαχείρισης Εκδηλώσεων και Έκδοσης Εισιτηρίων με Java Spring Boot»
Student's name-surname: Ονοματεπώνυμο Φοιτητή:	**Georgios Simos** Γεώργιος Σίμος
Father's name: Πατρώνυμο:	**Apostolos** Απόστολος
Student's ID No: Αριθμός Μητρώου:	**ΜΠΠΛ 2238**
Supervisor: Επιβλέπων:	**Efthymios Alepis, Professor** Ευθύμιος Αλέπης, Καθηγητής

March 2026 / Μάρτιος 2026

3-Member Examination Committee
Τριμελής Εξεταστική Επιτροπή

Alepis Efthymios **Professor**	**Virvou Maria** **Professor**	**Sotiropoulos Dionisios** **Associate Professor**
Αλέπης Ευθύμιος Καθηγητής	Βίρβου Μαρία Καθηγήτρια	Σωτηρόπουλος Διονύσιος Αναπληρωτής Καθηγητής

Abstract

This thesis presents the design and development of an integrated seat reservation and ticketing system for events and performances, titled ***Vistaseat.com***.

The system is a full-stack web application that streamlines the discovery, reservation, and purchase of tickets for various events, including theater plays, cinema screenings, concerts, festivals, sports matches and visits to museums or archaeological sites.

The platform is implemented using **Java Spring Boot** for the backend and **HTML, CSS, JavaScript, and Thymeleaf** for the frontend, supported by a **PostgreSQL** database managed through **Spring Data JPA** and **Hibernate**. It operates under two distinct modes: **User Mode** and **Administrator Mode**.

User Mode provides secure authentication, allowing guests and registered users to browse events, reserve seats, complete **payments via PayPal**, and download unique **PDF tickets** containing ticket numbers and barcodes for validation. Registered users can also submit testimonials to provide feedback and promote user interaction.

Administrator Mode enables the management of users, venues, events, bookings, testimonials and contact messages through a dedicated dashboard.

Overall, *Vistaseat.com* delivers a scalable and efficient solution for event ticketing, integrating robust backend design with effective user and administrative functionalities.

Περίληψη

Η μεταπτυχιακή διατριβή παρουσιάζει τον σχεδιασμό και την ανάπτυξη ενός ολοκληρωμένου συστήματος κράτησης θέσεων και έκδοσης εισιτηρίων για εκδηλώσεις και θεάματα, με τίτλο ***Vistaseat.com***.

Το σύστημα αποτελεί μία διαδικτυακή εφαρμογή που έχει ως σκοπό την αναζήτηση, την κράτηση και την αγορά εισιτηρίων για διάφορες εκδηλώσεις, όπως θεατρικές παραστάσεις, κινηματογραφικές προβολές, συναυλίες, φεστιβάλ, αθλητικούς αγώνες, καθώς και επισκέψεις σε μουσεία και αρχαιολογικούς χώρους.

Η πλατφόρμα έχει υλοποιηθεί με **Java Spring Boot** στο backend και **HTML, CSS, JavaScript** και **Thymeleaf** στο frontend, ενώ χρησιμοποιεί βάση δεδομένων **PostgreSQL** μέσω **Spring Data JPA** και **Hibernate**. Το σύστημα λειτουργεί σε δύο διακριτές λειτουργίες: **Λειτουργία Χρήστη** και **Λειτουργία Διαχειριστή**.

Η Λειτουργία Χρήστη παρέχει ασφαλή πιστοποίηση χρηστών, επιτρέποντας σε επισκέπτες και εγγεγραμμένους χρήστες να περιηγούνται σε εκδηλώσεις, να κάνουν κρατήσεις θέσεων, να ολοκληρώνουν **πληρωμές μέσω PayPal** και να κατεβάζουν μοναδικά **εισιτήρια PDF** που περιλαμβάνουν αριθμό εισιτηρίου και barcode για επαλήθευση. Οι εγγεγραμμένοι χρήστες μπορούν επίσης να υποβάλλουν σχόλια και αξιολογήσεις, ενισχύοντας την αλληλεπίδραση μεταξύ άλλων χρηστών.

Η Λειτουργία Διαχειριστή παρέχει τη δυνατότητα διαχείρισης χρηστών, χώρων, εκδηλώσεων, κρατήσεων, testimonials και μηνυμάτων επικοινωνίας μέσω ενός ειδικά διαμορφωμένου πίνακα ελέγχου (dashboard).

Συνολικά, το **Vistaseat.com** προσφέρει μια επεκτάσιμη και αποτελεσματική λύση για τη διαχείριση εισιτηρίων εκδηλώσεων, συνδυάζοντας ισχυρή backend αρχιτεκτονική με αποτελεσματική και φιλική προς τον χρήστη λειτουργικότητα.

Acknowledgements

First, I would like to express my sincere gratitude to my advisor, Professor Efthymios Alepis, Head of the Department of Informatics, University of Piraeus, for his guidance, insight, and encouragement throughout the preparation of this thesis. His mentorship was instrumental in deepening my understanding in the field of Software Engineering and in the successful completion of this work. I am especially thankful for the freedom he granted me to pursue my own ideas, which allowed me to design a challenging thesis project addressing real-world problems faced by professionals in the development of ticket management systems.

I would like to thank Professor Maria Virvou for imparting the principles of designing robust and scalable software systems during my studies. Her teaching in the Software Engineering course was invaluable in guiding the development of this thesis. The understanding of fundamental Software Engineering principles, such as maintainability, modularity, and separation of concerns played a pivotal role in shaping the architecture and implementation of this system.

I would also like to thank my friend and fellow classmate, Kostas Ganitis, for his valuable comments, insightful discussions, and continuous support throughout my studies and the implementation of this thesis.

Finally, I want to thank my family for their continuous support, patience and love.

Contents

Table of Figures

Foreword

The motivation behind this thesis originated from my interest in full-stack web development and my desire to explore how modern software architectures can be applied to real-world problems. The idea of developing an integrated seat reservation and ticketing platform, titled *Vistaseat.com*, emerged from observing the growing need for efficient, scalable, and user-friendly solutions in the field of event management and online ticketing.

In an era where digital transformation reshapes how users discover and access entertainment, the need for reliable and secure ticketing systems is evident. This thesis addresses that need through the development of a web-based platform offering dual functionality: user operations such as event exploration, seat selection, reservation, and payment, and administrative management of users, venues, events, and bookings. Throughout its development, I gained valuable practical experience in software design, RESTful API development, database management, and user interface design, which deepened my understanding of modern software engineering practices.

1. Introduction

Software development has undergone rapid transformations over the past three decades, yet some technologies have demonstrated remarkable resilience and adaptability. Among them, Java stands out as one of the few programming languages that have continuously evolved to meet the changing needs of enterprise applications [1]. Since its introduction in the mid-1990s, Java has remained a cornerstone of enterprise application development, continuously evolving through regular updates that introduce modern language features while maintaining its trademark stability [2].

1.1 The evolution of Java programming language

Java was introduced in May 1995 by James Gosling at Sun Microsystems and quickly established itself as a fundamental and popular programming language [3]. Its core strengths such as platform independence through the Java Virtual Machine (JVM), a mature ecosystem of libraries and frameworks, and strong backward compatibility have made it the backbone of countless business-critical systems worldwide.

Three decades later, Java remains far from obsolete. As of 2026, it continues to rank among the most widely used programming languages for enterprise-level applications [4]. One of the main reasons for this sustained relevance lies in its continuous evolution. With every new release, Java incorporates modern features that enhance developer productivity and system efficiency while maintaining the stability demanded by long-term enterprise solutions. Notable advancements include the introduction of lambda expressions and the Stream API in Java 8, local variable syntax in Java 11, Java Records in Java 16, sealed classes and pattern matching for switch in Java 17, virtual threads in Java 21, compact source files and instance main methods in Java 25. All these enhancements reflect Java's responsiveness to changing programming paradigms and its commitment to cleaner, more expressive code.

1.2 What is the Spring Framework and Spring Boot

1.2.1 The Spring Framework

Figure 1. Spring framework logo

The Spring Framework was conceived as a lightweight, comprehensive application-development framework for the Java platform. Its core purpose is to simplify Java enterprise-level development by providing support for dependency injection (DI), inversion of control (IoC), modular

architecture, transaction management, and data-access abstraction. Its most fundamental components are:

- **Dependency Injection**: Dependency Injection (DI) is a design pattern that enables objects to receive the dependencies they require from an external component rather than creating them internally. Dependencies can be supplied through constructor parameters, factory method arguments, or properties set after object creation. In frameworks such as the Spring container, these dependencies are automatically injected when the bean is instantiated.
- **Inversion of Controls**: Inversion of Control (IoC) is the principle by which the responsibility for creating, configuring, and managing the lifecycle of application objects (beans) is shifted from the application code to the Spring-managed container. In other words, instead of the classes instantiating and managing their dependencies directly, we declare how they should be configured and wired, and the Spring IoC container takes care of instantiating them, injecting required dependencies, and managing their lifecycle.

Its modular structure (Spring Core Container, Spring JDBC, Spring MVC, Spring ORM, etc.) allows developers to pick the components they need and build loosely-coupled, maintainable systems. In this way, Spring acts as a foundation: you assemble modules, configure beans, define wiring, and take full control of how your application is structured [5].

1.2.2 Spring Framework vs Spring Boot

Figure 2. Spring vs Spring Boot

Spring Boot is not a separate unrelated framework. It builds directly on the Spring ecosystem but shifts the emphasis from full-manual configuration to convention, opinionated defaults and auto-configuration. While Spring requires more explicit setup (defining bean wiring, handling web server deployment, configuring servlet containers or data-sources manually), Spring Boot reduces boilerplate by offering starter dependencies, embedded servers, and auto-configuration logic while maintaining the ability to override configurations when required [6]. In essence, Spring represents the foundational framework, whereas Spring Boot acts as an accelerator layer built atop it - leveraging Spring's modules while streamlining deployment through features such as embedded servers like Tomcat and self-contained executable packages.

1.3 Thesis Objectives

The primary goal of this thesis is the design and development of *Vistaseat.com*, a comprehensive web-based system for event reservation and online ticket management. The system aims to

streamline the process of booking and issuing tickets for a wide variety of cultural and entertainment events, including theatre performances, museum visits, archaeological site tours, cinema screenings, music concerts, festivals, and sporting events. By integrating these activities into a unified digital platform, *Vistaseat.com* seeks to enhance efficiency, accessibility, and user experience for both event organizers and attendees [7] [8].

The platform is structured around two main operational modes: **User Mode** and **Administrator Mode**, each designed to address distinct functional requirements.

In **User Mode**, the system provides secure authentication mechanisms that distinguish between guest users and registered users. Both user types can explore available events, view event details, and select their number of seats. The system supports the full booking workflow, from seat selection and payment processing via credit/debit card or PayPal to the generation of digital tickets. Each completed booking produces a unique PDF ticket that includes a ticket number and barcode, facilitating verification and entry control at venues. Additionally, registered users are granted extended privileges, such as the ability to view their account details and order history, submit testimonials and provide feedback on the platform's functionality, thereby promoting community interaction through sharing their experiences [9].

In **Administrator Mode**, the system provides an integrated dashboard for managing all core entities of the platform, including users, venues, events, occurrences, bookings, testimonials and contact messages. Administrators can create and modify event and occurrence details, configure venue layouts, monitor booking activity, and oversee user accounts. This administrative interface is designed to offer clarity, efficiency, and control over platform operations, ensuring that the system remains scalable and maintainable as the volume of data and users increases.

The underlying problem addressed by this thesis extends beyond ticket management. It concerns the broader challenge of designing efficient, maintainable, and user-centric web systems capable of supporting complex business processes. By implementing *Vistaseat.com*, this work demonstrates how a modern Java-based architecture can deliver both functional depth and architectural robustness suitable for enterprise-level applications. The project's practical contribution lies in providing a working prototype that embodies software engineering principles such as modularity, separation of concerns, reusability, and security within the real-world context of online ticketing.

1.4 Motivation behind this Thesis

The motivation for undertaking this thesis originated from my desire to engage deeply with the challenges that define real-world software development. Beyond the academic requirement, the development of *Vistaseat.com* represents a deliberate attempt to bridge theoretical understanding with the practical demands of professional software engineering [10]. The project was conceived as an opportunity to simulate the conditions faced by developers in industry who work with complex systems, integrate multiple technologies, and address both functional and non-functional requirements under realistic constraints.

2. Requirements Analysis

Functional requirements describe what the system must do - specific actions, behaviors, and outputs needed to meet user or business needs. On the other hand, non-functional requirements specify the standards a system must meet to ensure it works effectively and efficiently under all expected conditions.

2.1 Functional Requirements

- Role-based access control. Available roles in each mode are:
  - Guests
  - Registered Users

  User Mode
  - Domain Admins
  - Event Admins
  - Venue Admins

  Admin Mode
- All users in User Mode can make reservations.
- Accounts are created using an email and password. User authentication and authorization are handled through Spring Security with form-based login, and passwords are securely hashed using BCrypt.
- Registered users can browse their order history and submit testimonials through their account. All submitted testimonials are initially hidden, pending admin approval.
- All users in User Mode can explore events by event type such as:
  - Theater plays & performances
  - Cinema screenings
  - Music concerts & festivals
  - Sport events
  - Museum visits
  - Archaeological site visits.
- When browsing a selected event, users can apply search filters such as event name and date range.
- Supports payment through multiple payment methods, including credit/debit cards and PayPal.
- Bookings can have one of the following statuses:
  - PENDING: The booking has been created but awaits payment confirmation. It remains active for 15 minutes.
  - CONFIRMED: The booking is successfully completed following payment.
  - CANCELLED: The booking was canceled due to an error, payment failure, or expiration of the pending period.
  - REFUNDED: The related Occurrence was canceled, and the payment amount has been refunded to the customer.

- When a booking is pending, users have 15 minutes to complete the payment. If the payment is not completed within this period, the booking is automatically canceled and the reserved seats are released back into availability.
- A PDF ticket is generated for each seat, containing a unique ticket number and a barcode for scanning.
- All users in User Mode can submit contact forms about any issue they face while using the app.
- Domain Admins can:
  - Deactivate the accounts of registered users.
  - Create an account for a new admin.
  - Add, edit, search, and delete Venues, Events, and Event Occurrences. Deletion is only permitted when the entity is not linked to any active or dependent entities (e.g., Venues associated with existing or past Events cannot be deleted).
  - A new Occurrence can be added only if the Venue is available during the selected time range. Each new Occurrence must have at least a 30-minute buffer before and after any existing scheduled Occurrences.
  - Each Occurrence is marked with a color availability badge, depending on the number of remaining seats:
    - Available: > 50%, color: 
    - Limited: 50–25%, color: 
    - Few Left: 25-0.1%, color: 
    - Sold Out: 0%, color: 
  - Search, cancel, and reschedule bookings. Rescheduling is only allowed if the new Occurrence selected has available seats.
  - View bookings details such as tickets and payment information.
  - Toggle the visibility (visible or hidden) of testimonials submitted by registered users or delete them if necessary.
  - Mark contact messages as resolved by explicitly recording the actions taken to address the reported issues. Unresolved messages cannot be deleted.

2.2 Functional Requirements Modeling

Functional requirements can be effectively modeled using UML and Use Case Diagrams [11]. These diagrams describe the functionality of an information system from the perspective of its end users by capturing the interactions between actors and the system. Use Case Diagrams provide a clear and structured representation of the system's intended behavior, facilitating the understanding and validation of functional requirements [12].

2.2.1 Use Case Diagrams: User Mode

The following Use Case Diagrams display the main actions that various actors can perform during User Mode:

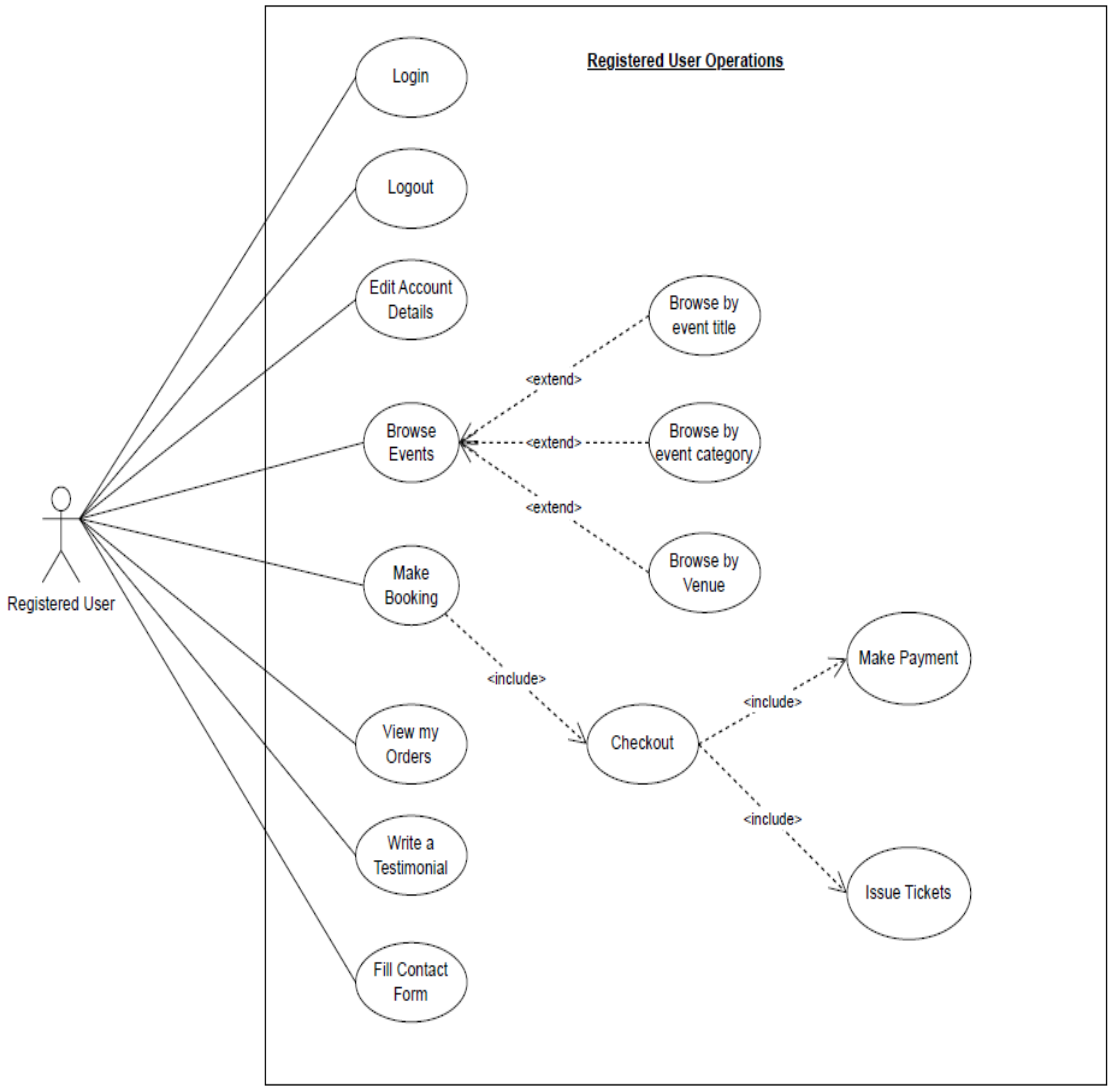


Figure 3. User Mode: Registered user operations

Actor

Registered User: An authenticated user who accesses the system using a unique username-password and can perform all available operations while the system is in User Mode. Some of the use cases are:

Use Case	**Login**
Description	Allows a user to log in using their unique credentials and experience the full experience of the system
Actor	Registered User
Requirements	The client has already created an account using their email and password

Primary Flow	1. The user clicks the “Connect” button located on the header. 2. The user provides an email address and password. 3. The system authenticates the user. 4. The system redirects the user to the previously requested page.
Alternative Flows	1. A user that has not yet logged in, is about to make a reservation, for a specific event. 2. The system prompts the user to log in or continue as a guest. 3. The user opts to Log in. 4. The system authenticates the user using email and password. 5. The system resumes the reservation process.

Use Case	**Make Booking**
Description	Allows a user to make a reservation about their preferred event.
Actor	Registered User
Requirements	The client has already created an account using their email and password and is logged in.
Primary Flow	1. The user is at the event page 2. The event still has available tickets 3. The user selects the number of tickets to be reserved. 4. The user clicks the “Proceed to Checkout” button. 5. The user reviews the Event and reservation info. 6. The user accepts the terms of use. 7. The user clicks “Continue to Payment” button. 8. The user selects one of the available payment methods to complete the transaction. 9. The tickets are issued on PDF format upon successful payment.
Alternative Flows	None

Use Case	**Write a Testimonial**
Description	Allows a user to submit a testimonial and provide feedback about their experience while using the system.
Actor	Registered User
Requirements	The client has already created an account using their email and password and is logged in.
Primary Flow	1. The user is at the “My Account” page. 2. The user provides a rating between 0 and 5 stars.

	3. The user writes a review between 5 and 500 characters. 4. The user clicks the “Submit Testimonial” button.
Alternative Flows	None

Figure 4. User Mode: Guest operations

Actor

Guest: Anonymous users who have limited access. They can browse events, make bookings and submit contact forms without registering. Some of the use cases are:

Use Case	**Sign Up**

Description	Allows an anonymous user to create an account by providing an email and a password.
Actor	Guest
Requirements	None
Primary Flow	1. The user clicks the “Sign Up” button on the Sign In page. 2. The user provides an email address and password. 3. The system creates an account for the user. 4. The system redirects the user to the Sign In page.
Alternative Flows	None

Use Case	**Fill Contact Form**
Description	Allows an anonymous user to submit messages to the support team, through a contact form, about any issue they encountered while using the system.
Actor	Guest
Requirements	None
Primary Flow	1. The user clicks the “Contact” button on the footer. 2. The user provides their full name. 3. The user provides an email. 4. The user fills in the subject of the message. 5. The user can select the category of their message from the drop-down list (optional). 6. The user writes their message that needs to be between 20 and 1000 characters. 7. The user must accept the terms of use. 8. The user clicks the “Send Message” button on the footer.
Alternative Flows	None

2.2.2 Use Case Diagram: Admin Mode

The following Use Case Diagram displays the main actions that an administrator can perform during Admin Mode:

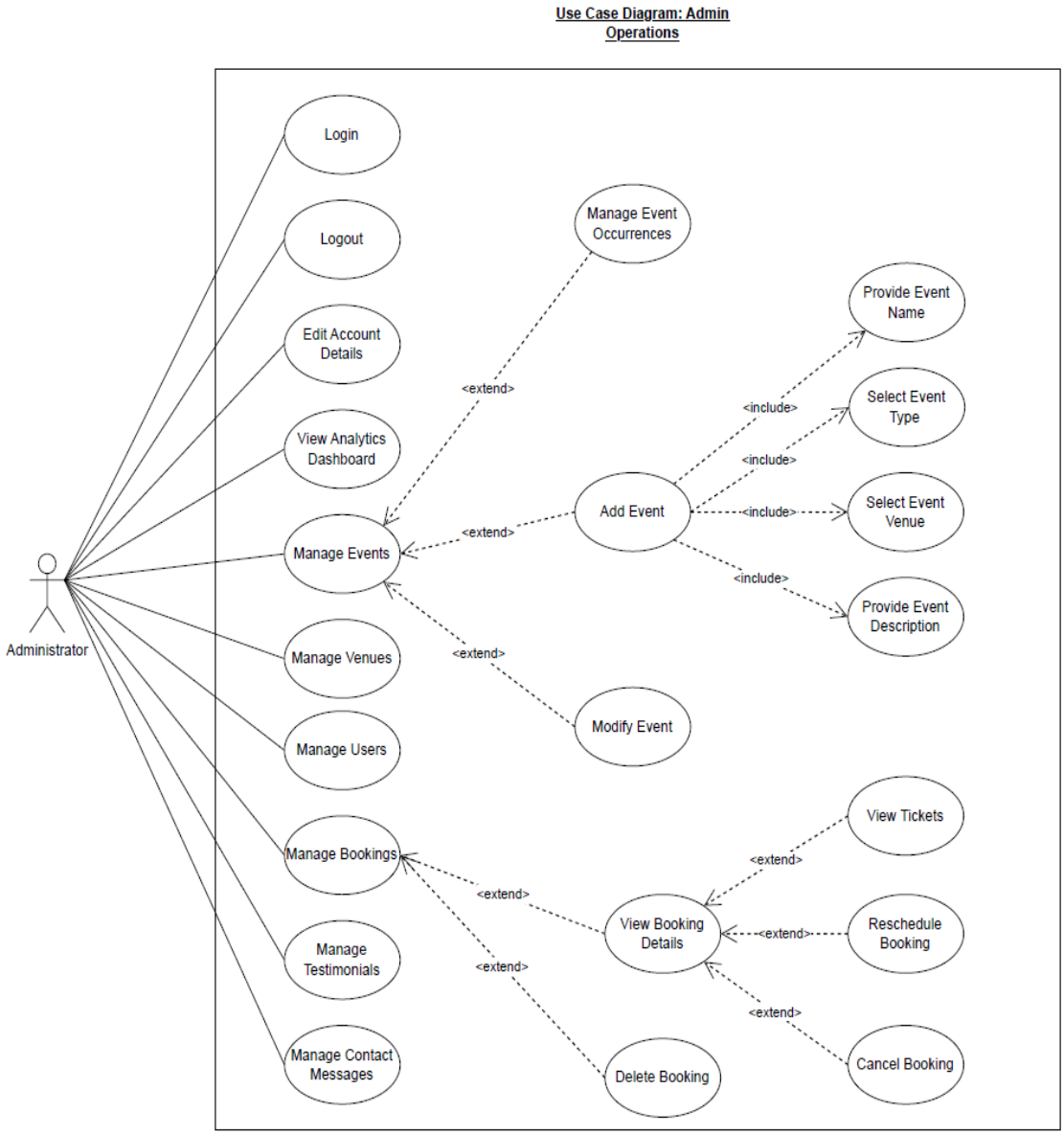


Figure 5. Admin Mode: Admin operations

Actor

Administrator: A user with elevated privileges who manages system data and operations, including venues, events, occurrences, user status (active/inactive), bookings, testimonials, and contact messages. Some of the use cases are:

Use Case	**Add New Occurrence**

Description	Allows an administrator to create a new occurrence for an existing event.
Actor	Admin
Requirements	An existing event
Primary Flow	1. The user clicks the “Add Occurrence for this Event” button on the manage occurrences for event view. 2. The user selects the date and time of the occurrence. 3. The user set a ticket price for this specific occurrence. 4. The user sets the occurrence duration. 5. The user sets the number of available seats.
Alternative Flows	None

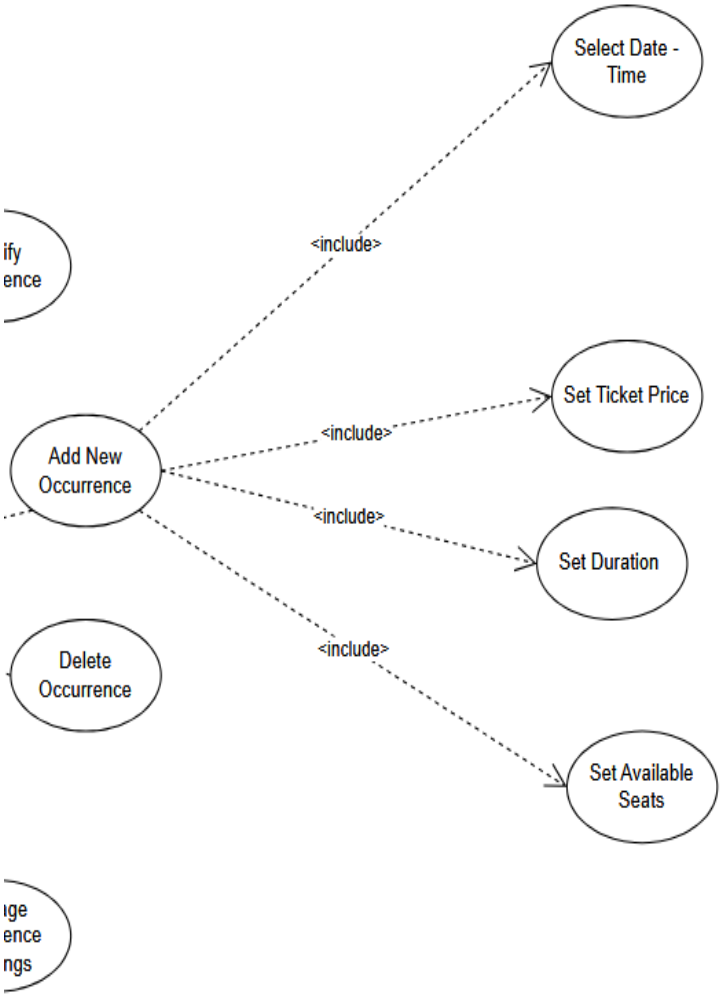


Figure 6. Use Case: Add new occurrence

Use Case	Change User Status
Description	Allows an administrator to change user account status between active or inactive
Actor	Admin
Requirements	An existing user account
Primary Flow	1. The admin clicks the “Change User Status” button on the manage users view. 2. The user account status is changed.
Alternative Flows	None

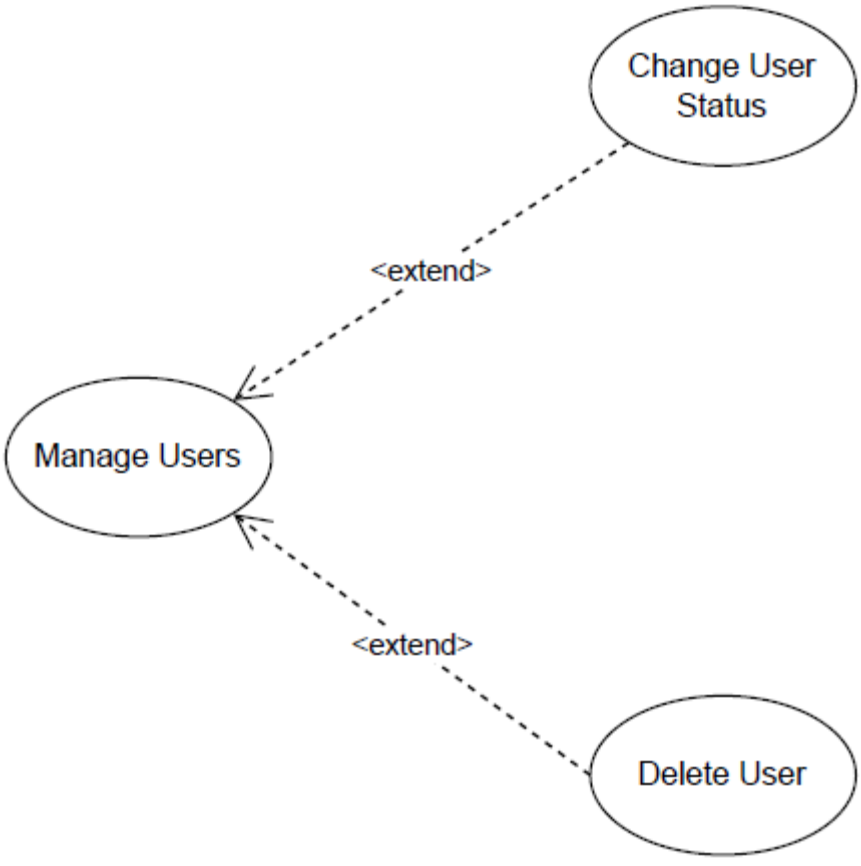


Figure 7. Use Case: Manage users

2.3 Non-functional requirements

- Smooth performance:
  - The system should support a large number of concurrent users (over 10,000 requests/sec) without noticeable degradation in performance.
  - Page loading, event browsing and searching, seat booking, and payment completion should each be completed within a user-acceptable timeframe.
- Security:
  - The system implements two separate security filter chains — one for guests and registered users, and another for administrators — restricting access to endpoints based on user roles.
  - Each user must log in with a unique email address.
- User Experience:
  - User interfaces should be visually appealing, intuitive, and easy to use for all users, regardless of their familiarity with modern information systems.
- Scalability:
  - The system architecture should be scalable, allowing the seamless addition of new event types, user authorization levels, and payment methods.
- Reliability:
  - Minimize response time during errors by displaying user-friendly notifications and redirecting appropriately.
  - Ensure data integrity and consistency across all application endpoints.

3. System Architecture and Technologies

The system follows a layered monolithic architecture structured according to the Spring Model-View-Controller (Spring MVC) pattern [13] [14], while adopting certain principles inspired by the Service-Oriented Architecture (SOA) [15] to maintain clear separation of concerns and modularity.

The presentation layer includes all user-facing views and interfaces. Within it, the controller layer handles HTTP requests and responses, using Data Transfer Objects (DTOs) to pass data between the client and server.

The service layer contains the core business logic of the application. It implements key operations such as creating users, venues, and events, managing bookings, rescheduling reservations, checking ticket availability for specific occurrences and more.

Communication with the persistence layer is managed through the repository layer, implemented using Spring Data JPA. Data is mapped between Java objects and database tables via Hibernate, an underlying Object-Relational Mapping (ORM) tool. The system's database is built with PostgreSQL.

The architecture and its layers are illustrated below:

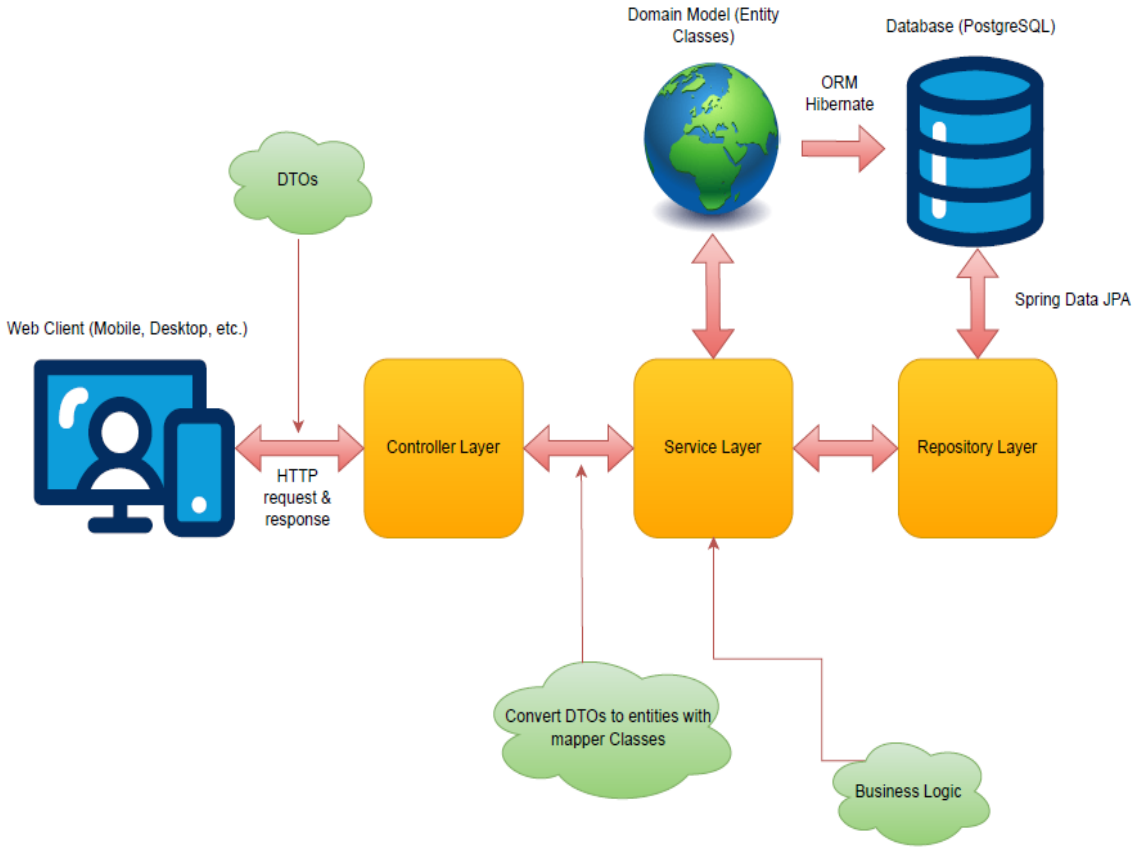


Figure 8. System architecture

3.1 Presentation Layer

The presentation layer is implemented using Thymeleaf templates combined with HTML, CSS, JavaScript, and Bootstrap. Controllers handle HTTP requests and responses, acting as the bridge between the client interface and the underlying application logic. Data exchanged between the presentation and service layers is encapsulated in Data Transfer Objects (DTOs) to ensure that internal model entities remain hidden from the client.

3.1.1 Thymeleaf Template Engine

Figure 9. Thymeleaf logo

Thymeleaf is a modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text. It is responsible for rendering dynamic web pages and managing user interaction. Thymeleaf's main goal is to provide an elegant and maintainable approach to building templates. It achieves this through the concept of Natural Templates, which allows developers to embed logic directly into template files without breaking their usability as static design prototypes. This approach enhances collaboration and bridges the gap between design and development teams [16].

3.1.2 Bootstrap

Bootstrap is one of the most widely used front-end frameworks for developing responsive and visually consistent web interfaces. It offers a rich collection of prebuilt components, grid layouts, and utility classes built with modern HTML, CSS, and JavaScript. Its popularity comes from its simplicity, cross-browser compatibility, and strong community support. By leveraging Bootstrap's ready-made design system, developers can prototype and style interfaces quickly without writing extensive custom CSS. This greatly reduces development time while ensuring a clean and professional appearance across different screen sizes and devices. For this project, Bootstrap v5.3 was used [17].

3.1.3 Controllers

Controllers serve as the core communication bridge between the client and the server, responsible for handling HTTP requests and returning appropriate HTTP responses. In a Spring Boot application, controllers are annotated with @Controller or @RestController, depending on whether they return rendered views or data objects. They define methods that map to specific HTTP operations using annotations such as @GetMapping, @PostMapping, @DeleteMapping, @RequestMapping and more.

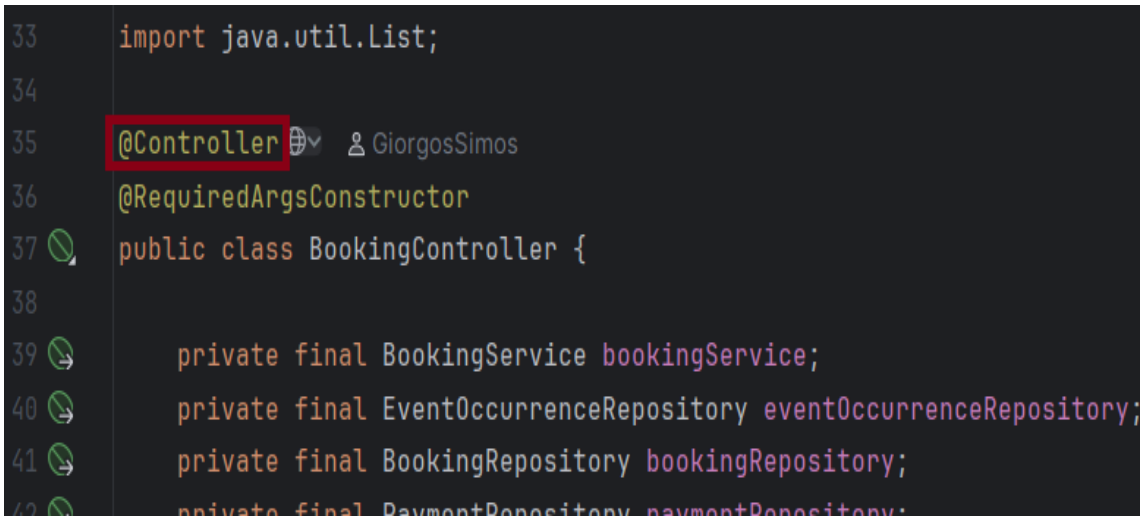


Figure 10. Controller class for booking endpoints

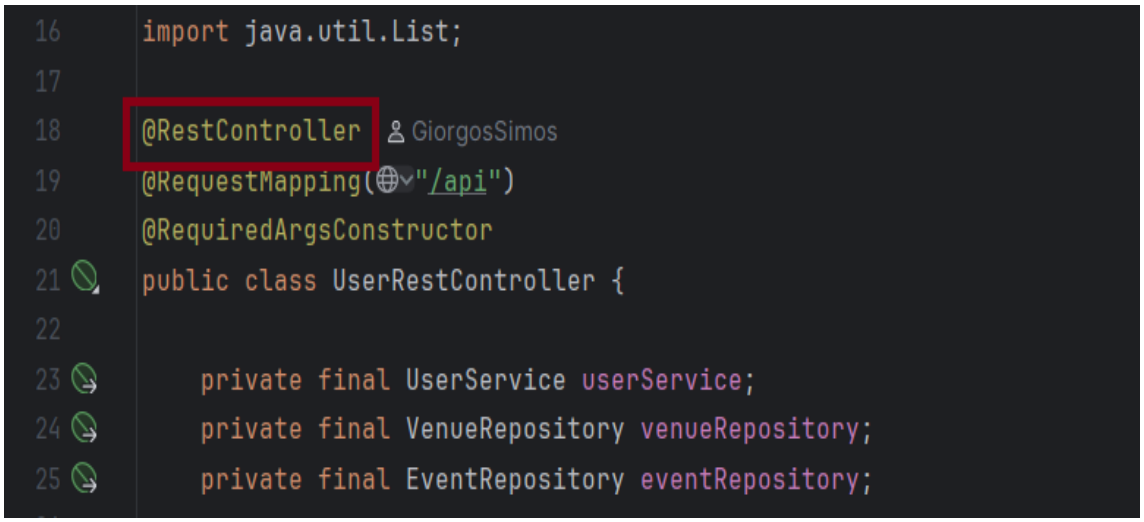


Figure 11. Rest Controller class for user endpoints

Controllers also coordinate with the service layer to execute business logic and use Data Transfer Objects (DTOs) to safely exchange data between the client and the backend.

3.1.4 Data Transfer Objects (DTOs)

Data Transfer Objects (DTOs) are simplified data containers used to transfer information between different layers of an application, particularly between the service (or business logic) layer and the presentation layer. They help maintain a clear separation between the internal data model and what is exposed to the client, ensuring that sensitive or irrelevant information remains hidden. DTOs are especially useful when sending or receiving data through controllers, as they prevent direct interaction with database entities. By defining only the necessary fields for a given operation, DTOs improve security, reduce data payload size, and make the codebase easier to maintain and extend.

3.1.5 Mapper Classes

Mapper classes are responsible for converting data between different object types, such as transforming entities into Data Transfer Objects (DTOs) and vice versa. This separation keeps the business logic clean and prevents data conversion code from cluttering service or controller layers. They are typically annotated with @Component, which marks them as Spring-managed beans, allowing them to be automatically detected and injected where needed through dependency injection. This ensures a single shared instance of each mapper and promotes cleaner, more maintainable code.

```java
@Component 17 usages  GiorgosSimos
@RequiredArgsConstructor
public class BookingMapper {

    private final EventOccurrenceRepository eventOccurrenceRepository;
    private final UserRepository userRepository;

    public Booking toEntity(BookingDto  bookingDto) {  GiorgosSimos

        Booking booking = new Booking();

        EventOccurrence occurrence = eventOccurrenceRepository.findById(bookingDto.getOccurrenceId())
                .orElseThrow(() -> new EntityNotFoundException("Event Occurrence not found"));
        booking.setEventOccurrence(occurrence);
```

Figure 12. Class responsible for mapping booking DTOs to booking entities

3.2 Service (Business Logic) Layer

The service layer, also known as the business logic layer, implements the core functionality of the application. It defines the methods that describe what the system does such as processing requests, applying business rules, and coordinating operations between controllers and repositories.

Classes in this layer are typically annotated with @Service, which is a specialized form of @Component. The @Component annotation marks a class as a Spring-managed bean, allowing it to be automatically detected and instantiated by the Spring container. Dependencies between classes are managed through dependency injection, most commonly using the @Autowired annotation, which automatically injects the required beans into a class.

By combining these annotations, the service layer remains modular, testable, and decoupled from specific implementations, promoting cleaner code and easier maintenance throughout the app.

```java
@Service  GiorgosSimos
@RequiredArgsConstructor
public class EventServiceImpl implements EventService {

    private final EventRepository eventRepository;
    private final EventMapper eventMapper;
    private final EventOccurrenceRepository eventOccurrenceRepository;

    @Override  1 usage  GiorgosSimos
    @Transactional
    public EventDto createEvent(@NotNull EventDto eventDto) {
```

Figure 13. Class responsible for implementing event business Logic

3.3 Repository Layer

The repository layer serves as the data access foundation of the application. It is responsible for communicating with the database and performing all persistence operations, such as storing, retrieving, updating, and deleting records. This layer is built upon Spring Data JPA, Hibernate, and PostgreSQL, which together provide a powerful, efficient, and reliable data management solution. Spring Data JPA abstracts most of the low-level database interactions, Hibernate handles the object-relational mapping between Java entities and database tables, and PostgreSQL serves as the robust relational database system that ensures data consistency and reliability.

3.3.1 Spring Data JPA

Spring Data JPA is a high-level abstraction layer built on top of the Java Persistence API (JPA) [18]. It simplifies data access by removing the need for boilerplate code and providing a consistent interface for database operations. At the core of this system are repository interfaces that extend JpaRepository, which comes with a wide range of ready-to-use methods such as findAll(), findById(), save(), and deleteById(). In addition, Spring Data JPA supports derived query methods, where method names automatically translate into SQL queries — for example, findByEmail() or findByEventDateBetween().

```
16  public interface BookingRepository extends JpaRepository<Booking, Long> {  GiorgosSimos
17
18      Long countBookingsByEventOccurrenceId(Long eventOccurrenceId);  1 usage  GiorgosSimos
19
20      // Total number of bookings across all occurrences of a given event
21      Long countBookingsByEventOccurrence_Event_Id(Long eventId);  1 usage  GiorgosSimos
22
23      // Total number of bookings across all occurrences scheduled in a given venue
24      Long countByEventOccurrence_Event_Venue_Id(Long eventVenueId);  1 usage  GiorgosSimos
25
26      Page<Booking> findByEventOccurrenceId(Long eventOccurrenceId, Pageable pageable);  1 usage  GiorgosSimos
27
28      Page<Booking> findByIdAndEventOccurrenceId(Long bookingId, Long eventOccurrenceId, Pageable pageable);  1
29
30      // List of bookings across all occurrences of a given event name
31      Page<Booking> findByEventOccurrence_Event_NameContainingIgnoreCase(String eventName, Pageable pageable);
```

Figure 14. Booking repository class

For more complex requirements, developers can define custom JPQL(Java Persistence Query Language) or native SQL queries using the @Query annotation. This combination of flexibility and simplicity drastically reduces the amount of code required for data access while improving readability and maintainability.

```
@Query("""  1 usage  GiorgosSimos
    select distinct e
    from Event e
    where e.eventType = :type
        and lower(e.name) like lower(concat('%', :name, '%'))
        and exists (select 1
                    from EventOccurrence o
                    where o.event = e AND o.eventDate > CURRENT TIMESTAMP
                    )
""")
List<Event> findUpcomingWithAtLeastOneOccurrenceByTypeAndNameLike(@Param("type") EventType type,
                                                                  @Param("name") String name);

@Query("""  1 usage  GiorgosSimos
    select distinct e
    from Event e
    where e.eventType = :type
```

Figure 15. Custom JPQL definitions

3.3.2 ORM Hibernate

Figure 16. Hibernate logo

Hibernate is an Object-Relational Mapping (ORM) framework that automatically maps Java objects to relational database tables [19]. It eliminates the need for manual SQL statements for most CRUD operations, simplifying database interaction and reducing development effort. Through annotations such as @Entity, @Table, @Id, @Column, @OneToMany and more, Hibernate defines how entities are created and relate to each other, how primary and foreign keys are managed, and how column constraints are applied.

```
@Getter  GiorgosSimos
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "bookings")
public class Booking {
    @Id
    @Column(name = "booking_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // Multiple bookings can refer to a single event occurrence
    @ManyToOne
    @JoinColumn(name = "event_occurrence_id", nullable = false)
    private EventOccurrence eventOccurrence;

    // Multiple bookings can be made by a single user, user_id is a FK
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = true) // Nullable for guests
    private User user;

    @Pattern(regexp = "^[A-Za-zÀ-ÖØ-öø-ÿ]{2,30}(?:[ '-][A-Za-zÀ-ÖØ-öø-ÿ]{2,30})?$", message = "Invalid first name")
    @Column(name = "first_name", nullable = false) // For registered users autocompleted with user.getFirstName()
    private String firstName;
```

Figure 17. Booking entity class

By handling relationships like **one-to-many**, **many-to-one**, and **many-to-many**, Hibernate ensures that data integrity is maintained across all entities. Its automatic schema generation and migration support further streamline the development process, allowing developers to modify the data model without heavy refactoring or manual database adjustments. Overall, Hibernate significantly reduces development time and migration costs by providing a unified, object-oriented approach to data persistence.

3.3.3 PostgreSQL

PostgreSQL is an advanced Relational Database Management System (RDBMS) known for its stability, performance, and strong adherence to SQL standards [20]. It offers powerful features such as transactional integrity, concurrency control, and extensive indexing options, making it a preferred choice for enterprise-level applications. In the context of *Vistaseat.com*, PostgreSQL provides the reliability and scalability needed to manage large volumes of user, booking, and event data efficiently. Its robust architecture, combined with Spring Data JPA and Hibernate, ensures that the system can handle complex transactions while maintaining high performance and data consistency.

3.4 Spring Security

Spring Security enables fine-grained control over which users can access specific parts of the system [21] [22]. In *Vistaseat.com*, where both regular users and administrators operate through distinct interfaces, defining **two separate filter chains** is crucial to isolate authentication and authorization flows [23].

```
@Configuration  GiorgosSimos *
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    private final UserRepository _userRepository;  2 usages

    public SecurityConfig(UserRepository userRepository) { _userRepository = userRepository; }

    @Bean  GiorgosSimos
    public UserDetailsService userDetailsService() { return new CustomUserDetailsService(_userRepository); }

    // Used for storing encrypted passwords in the database and login
    @Bean  GiorgosSimos
    public BCryptPasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

    // Authentication provider used by admins
    @Bean  GiorgosSimos
    public AuthenticationProvider adminAuthenticationProvider() {
        AdminOnlyAuthenticationProvider adminAuthenticationProvider = new AdminOnlyAuthenticationProvider();
        adminAuthenticationProvider.setUserDetailsService(userDetailsService());
        adminAuthenticationProvider.setPasswordEncoder(passwordEncoder());
        return adminAuthenticationProvider;
    }

    // Authentication provider used by users
    @Bean  GiorgosSimos
    public AuthenticationProvider userAuthenticationProvider() {
        UserOnlyAuthenticationProvider userAuthenticationProvider = new UserOnlyAuthenticationProvider();
        userAuthenticationProvider.setUserDetailsService(userDetailsService());
        userAuthenticationProvider.setPasswordEncoder(passwordEncoder());
        return userAuthenticationProvider;
```

Figure 18. Spring Security configuration

Each chain applies only to its respective endpoints, ensuring that admin routes remain inaccessible to normal users.

```java
// Filter chain for Admin actions
@Bean  GiorgosSimos
@Order(1)
public SecurityFilterChain adminSecurityFilterChain(HttpSecurity http, CustomAuthFailureHandler customAuthFailureHandler)
        throws Exception {

    http
            // Rules in this filterChain only apply to these admin-related endpoints.
            .securityMatcher("/adminLogin", "/adminLogout", "/adminDashboard", "/adminDashboard/**")

            // When someone logs in, check their credentials using adminAuthenticationProvider
            .authenticationManager(new ProviderManager(List.of(adminAuthenticationProvider())))

            .exceptionHandling( ExceptionHandlingConfigurer<HttpSecurity> ex -> ex
                    // If the user isn't logged in but tries to access an admin page → redirect them to /adminLogin
                    .authenticationEntryPoint(( HttpServletRequest req, HttpServletResponse res, AuthenticationException e) -> res.sendRe
                    // If they are logged in but don't have permission → send them to /error?status=403
                    .accessDeniedHandler(( HttpServletRequest req, HttpServletResponse res, AccessDeniedException e) -> res.sendRedirec
            )

            .csrf(AbstractHttpConfigurer::disable)
            .formLogin( FormLoginConfigurer<HttpSecurity> httpForm -> {
                httpForm.loginPage("/adminLogin")
                        .loginProcessingUrl("/adminLogin")
                        .usernameParameter("email") // Instead of username, we're using email
                        .passwordParameter("password")
                        .defaultSuccessUrl( defaultSuccessUrl: "/adminDashboard",  alwaysUse: true)
                        .failureHandler(customAuthFailureHandler)
                        .permitAll();// endpoints that are publicly accessible
            })
```

Figure 19. Security filter chain used for Admin Mode operations

```java
// Filter chain for user actions
@Bean  GiorgosSimos
@Order(2)
public SecurityFilterChain userSecurityFilterChain(HttpSecurity http, CustomAuthFailureHandler customAuthFailureHandler)
        throws Exception {

    http
            .authenticationManager(new ProviderManager(List.of(userAuthenticationProvider())))
            .exceptionHandling( ExceptionHandlingConfigurer<HttpSecurity> ex -> ex
                    .authenticationEntryPoint(( HttpServletRequest req, HttpServletResponse resp, AuthenticationException e) -> resp.se
            )
            .csrf(AbstractHttpConfigurer::disable)
            .formLogin( FormLoginConfigurer<HttpSecurity> form -> form
                    .loginPage("/userLogin")
                    .loginProcessingUrl("/userLogin")
                    .usernameParameter("email")
                    .passwordParameter("password")
                    // Use saved request so users return to the page where they clicked "Connect"
                    .successHandler(userLoginSuccessHandler())
                    .failureHandler(customAuthFailureHandler)
                    .permitAll()
            )
            .logout( LogoutConfigurer<HttpSecurity> logout -> logout
                    .logoutUrl("/userLogout")
                    .logoutSuccessUrl("/home?logout=true")
                    .invalidateHttpSession(true)
                    .deleteCookies( ...cookieNamesToClear: "JSESSIONID")
                    .permitAll()
```

Figure 20. Security filter chain used for User Mode operations

Such configuration minimizes the risk of privilege escalation, strengthens system integrity, and maintains a clear separation of concerns between user roles [24].

3.4.1 Authentication

Authentication is the process of verifying the identity of a user before granting access to the system. In *Vistaseat.com*, both registered users and administrators authenticate through email and password, stored securely in the database. When a user submits their details, Spring Security compares the provided credentials with the hashed values stored in the database. Admins are authenticated through a different endpoint and are assigned an elevated role that allows them to access restricted sections such as the admin dashboard and its functionalities. This role-aware approach ensures distinct authentication and maintains strict privilege boundaries.

3.4.2 Form-Based Session login

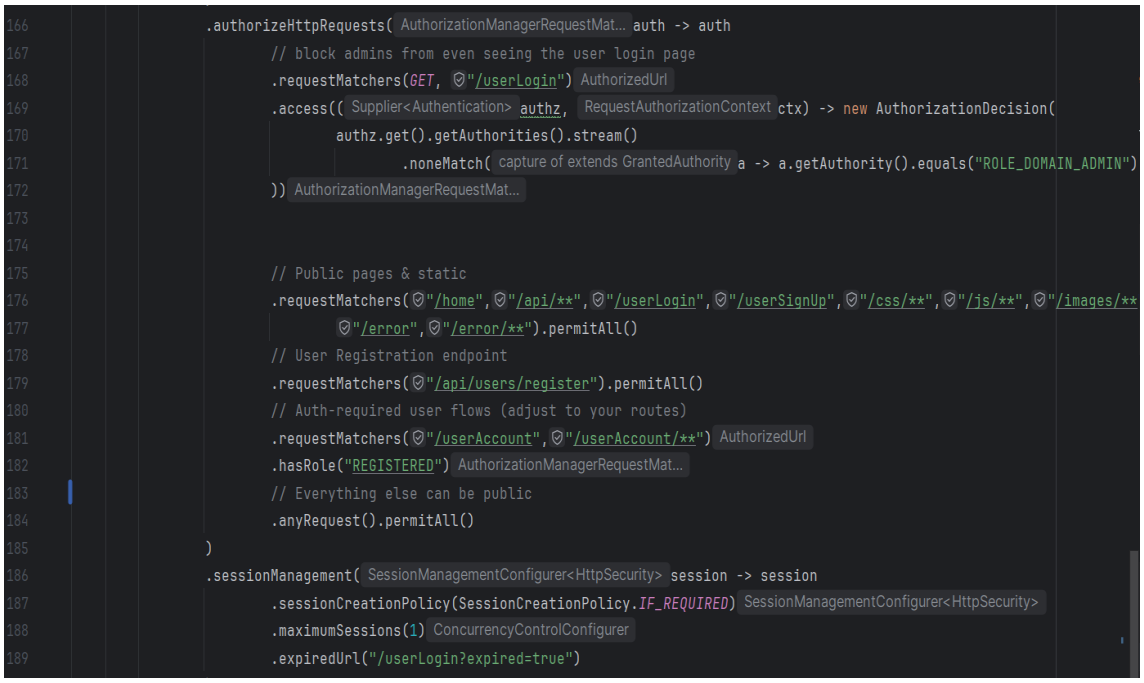
Vistaseat.com employs a stateful, form-based session login method instead of token-based or stateless approaches like JWT. This was chosen because Vistaseat.com operates as a traditional web platform rather than a distributed microservice environment. A session-based approach allows Spring Security to manage authentication automatically through the server's HTTP session, simplifying the process and maintaining user state between requests. It also integrates naturally with Thymeleaf templates and server-rendered views, reducing implementation complexity and the need for manual token handling [25].

3.4.3 Authorization – Role Based Access Control (RBAC)

Once authenticated, each user's access to endpoints is determined by their assigned role. The security configuration defines two distinct filter chains: one for user-level endpoints and another for administrator routes.

```
.authorizeHttpRequests( AuthorizationManagerRequestMat... authorizeRequests -> authorizeRequests//creating exceptions fo

        .requestMatchers("/adminLogin", "/adminLogout").permitAll() AuthorizationManagerRequestMat...
        .requestMatchers("/css/**", "/js/**", "/images/**").permitAll()

        // Only users with the DOMAIN_ADMIN role can see the admin dashboard and its sub-pages
        .requestMatchers("/adminDashboard", "/adminDashboard/**") AuthorizedUrl
        .hasRole("DOMAIN_ADMIN") AuthorizationManagerRequestMat...
        // For everything else in this security chain, you must be logged in (authenticated)
        .anyRequest().authenticated());

http.requestCache(RequestCacheConfigurer::disable);

http.sessionManagement( SessionManagementConfigurer<HttpSecurity> session -> session
        // Create a session only if needed
        .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED) SessionManagementConfigurer<HttpSecurity>
        // Allow one active session per user
        .maximumSessions(1) ConcurrencyControlConfigurer
        .expiredUrl("/adminLogin?expired=true")
```

Figure 21. Authorization request handling



```
.authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
        // block admins from even seeing the user login page
        .requestMatchers(GET, "/userLogin") AuthorizedUrl
        .access(( Supplier<Authentication> authz, RequestAuthorizationContext ctx) -> new AuthorizationDecision(
                authz.get().getAuthorities().stream()
                        .noneMatch( capture of extends GrantedAuthority a -> a.getAuthority().equals("ROLE_DOMAIN_ADMIN")
        )) AuthorizationManagerRequestMat...


        // Public pages & static
        .requestMatchers("/home","/api/**","/userLogin","/userSignUp","/css/**","/js/**","/images/**
                "/error","/error/**").permitAll()
        // User Registration endpoint
        .requestMatchers("/api/users/register").permitAll()
        // Auth-required user flows (adjust to your routes)
        .requestMatchers("/userAccount","/userAccount/**") AuthorizedUrl
        .hasRole("REGISTERED") AuthorizationManagerRequestMat...
        // Everything else can be public
        .anyRequest().permitAll()
)
.sessionManagement( SessionManagementConfigurer<HttpSecurity> session -> session
        .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED) SessionManagementConfigurer<HttpSecurity>
        .maximumSessions(1) ConcurrencyControlConfigurer
        .expiredUrl("/userLogin?expired=true")
```

Figure 22. Role Based Access Control (RBAC)

This separation ensures that the authentication and authorization logic for each group remains independent and clearly scoped. Additionally, role-based access control is reinforced through method level annotations [e.g. @PreAuthorize("hasRole('DOMAIN_ADMIN')")] within controller methods.

```
@PreAuthorize("hasRole('DOMAIN_ADMIN')") GiorgosSimos
@GetMapping("/adminAccount")
public String adminAccount(Model model) {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    User user = (User) auth.getPrincipal();

    model.addAttribute( attributeName: "firstName", user.getFirstName());
    model.addAttribute( attributeName: "lastName", user.getLastName());
    model.addAttribute( attributeName: "email", user.getEmail());
    model.addAttribute( attributeName: "phone", user.getPhone());

    return "adminAccount";
}
```

Figure 23. Method level authorization control (Admin Mode)

```java
@PreAuthorize("hasRole('REGISTERED')") GiorgosSimos
@PostMapping("/userAccount/update")
public String updateUserDetails(@RequestParam String firstName,
                                @RequestParam String lastName,
                                @RequestParam String phone,
                                Principal principal,
                                RedirectAttributes redirectAttributes) {
    String email = principal.getName();

    User user = userRepository.findByEmail(email)
            .orElseThrow(() -> new RuntimeException("No user found with email: " + email));

    user.setFirstName(firstName);
    user.setLastName(lastName);
    user.setPhone(phone);

    try{
        userRepository.save(user);

        // Refresh the user in SecurityContextHolder after update
        UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                user,                    // the updated User object
                null,                    // credentials -> not providing a password, the user is already authenticated
                user.getAuthorities());  // the roles/authorities of the user
        SecurityContextHolder.getContext().setAuthentication(authToken);

        redirectAttributes.addFlashAttribute( attributeName: "message",
                attributeValue: "Your account details were updated successfully!");
```

Figure 24. Method level authorization control (User Mode)

This double layer of enforcement - at both URL and method level - prevents unauthorized access to sensitive functionality and guarantees consistent policy application throughout the application.

3.4.4 Password Encoding

Spring Security's BCryptPasswordEncoder is used to hash passwords before storing them in the database.

```java
// Used for storing encrypted passwords in the database and login
@Bean GiorgosSimos
public BCryptPasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Figure 25. Password hashing using BCrypt

BCrypt applies a computationally intensive algorithm with a built-in salt, making it highly resistant to brute-force and rainbow table attacks [26].

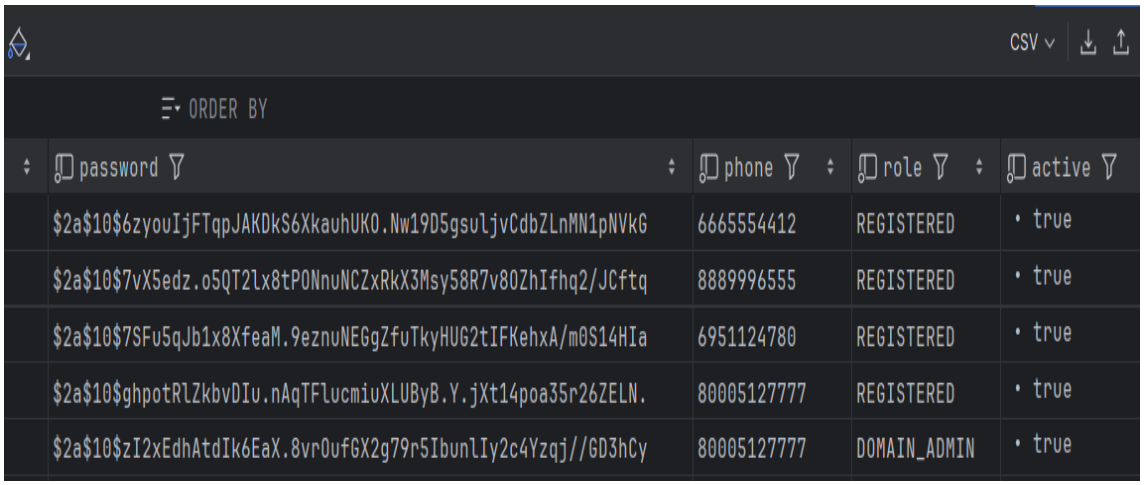


Figure 26. Hashed password stored in DB

Storing only hashed values instead of plain-text passwords is essential for protecting user data and complying with modern data protection regulations such as the GDPR [27]. Under these laws, developers are required to adopt appropriate technical safeguards to prevent unauthorized access or exposure of personal data, and strong password hashing is a key component of that obligation [28].

4. Data Model and Database Design

4.1 Domain Entities - Entity Relationship Diagram (ERD)

An entity is a thing or object in the real world with an independent existence. Each entity has attributes - the particular properties that describe it [29, p. 63].

The application entities are:

- **User:** Models the various users of the application. Holds information such as user first and last name, email, password, phone number, registration date-time and user status (active or not). When in User Mode, a user role can be registered or they can browse the app as guests. In Admin Mode, a user can have the following roles:
  - Domain Administrator: Top level administrator, manages all venues and events
  - Venue Administrator: Manages one or more venues and its events
  - Event Administrator: Manages one or more events
- **Venue:** Models the various venues that are managed by the app. Holds information such as venue id, name, address and maximum capacity.
- **Event:** Models events hosted at the venues. Holds information about the event id, name, event description and type. There are six different event types available:
  - Theater plays & performances
  - Cinema Screenings

  - Music Festivals & Concerts
  - Sport Events
  - Museum Visits
  - Archaeological Site Visits
- **Event Occurrence:** Models the occurrences of an event. Holds information about the occurrence date-time, the duration and the price.
- **Booking:** Models an order-reservation made by a customer. Holds information such as booking number, customer first and last name, email, phone number, the date-time a booking was made, the amount paid for the reservation, the number of booked tickets, expiration date and booking status with the following values:
  - Pending: The booking has been created but awaits payment confirmation. It remains active for 15 minutes.
  - Confirmed: The booking is successfully completed following payment.
  - Cancelled: The booking was canceled due to an error, payment failure, or expiration of the pending period.
  - Refunded: The related Occurrence was canceled, and the payment amount has been refunded to the customer.
- **Ticket:** Models a unique ticket for each seat that was booked by a customer. Holds information such as ticket number, barcode number and status with the following values:
  - Active
  - Expired
  - Cancelled
- **Payment:** Models a payment transaction for a specific booking. Holds information such as total payment amount, payment date and time, payment method (credit/debit card or PayPal) and payment status with the following values:
  - Completed
  - Failed
  - Refunded
- **Testimonial:** Models testimonials about the app experience that registered users can submit. Holds information such as rating, review, submission timestamp and a visibility flag (visible/hidden).
- **Contact Message:** Models contact forms submitted by all users in User Mode to report issues while using the app. Admins receive these messages and are responsible to resolve them. Holds information such as creation date-time, message, subject, category, status (pending or resolved), date-time it was resolved and notes about the actions an admin took to resolve the issue.

The relationships between the entities mentioned above are:

- A domain administrator manages one or more venues.
- A domain administrator manages one or more events.
- A venue can host one or more events.
- An event can have one or more occurrences.
- A user (registered or guest) can make one or more bookings.

- A booking can reference only one occurrence.
- A booking can reference one or more payments.
- A payment refers to one booking.
- A booking refers to one or more tickets.
- A ticket can reference only one booking.
- A registered user can submit one or more testimonials.
- A user can submit one or more contact messages

These entities and their relationships as they were described above are presented in the following **Entity – Relationship Diagram (ER):**

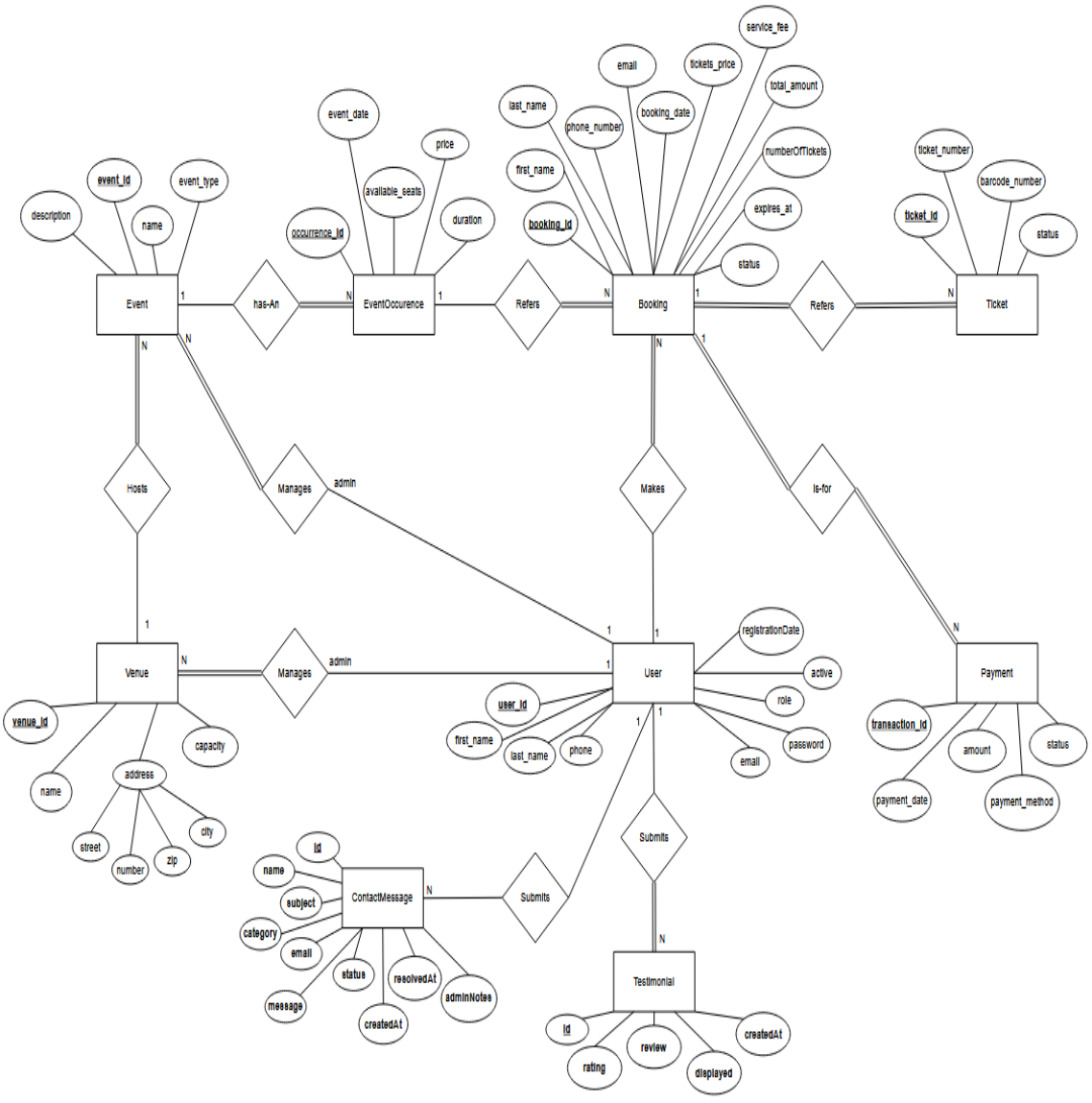


Figure 27. Visatseat.com Entity Relationship Diagram

4.2 Structural Design

Class diagrams are UML structure diagrams which show the structure of the designed system at the level of classes and interfaces. It shows their features, constraints, relationships - associations, dependencies, etc [30]. The domain model diagram for our system is displayed below:

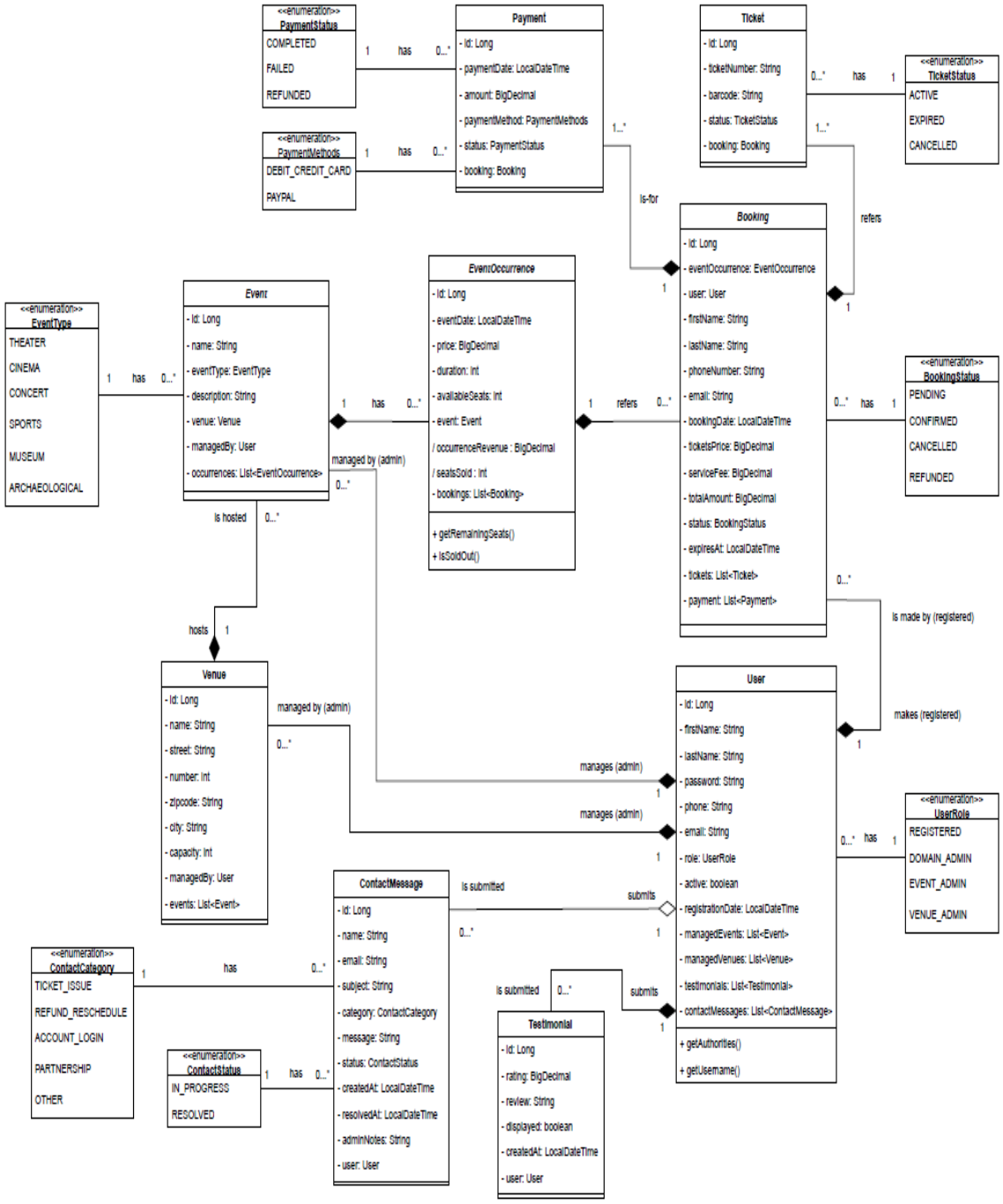


Figure 28. Class Diagram

4.3 Relational Model

The relational model represents the database as a collection of relations. Each relation resembles a table of values. When a relation is thought of as a table of values, each row in the table represents a collection of related data values. In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation [29, p. 150-151]. *Vistaseat.com* implements the following relational model:

Table (Relation)	*Columns (Attributes)*
User	**user_id,** first_name, last_name, email, phone, password, role, active, registrationDate
Venue	**venue_id**, admin_id(fk), name, street, number, zip, city, capacity
Event	**event_id**, venue_id(fk), admin_id(fk), name, event_type, description
Event Occurrence	**event_occurrence_id**, event_id(fk), event_date, price, duration, available_seats
Booking	**booking_id**, user_id(fk), event_occurrence_id(fk), first_name, last_name, phone_number, email, booking_date, tickets_price, service_fee, total_amount, number_of_tickets, expires_at, status
Payment	**transaction_id**, booking_id(fk), payment_date, amount, payment_method, status
Ticket	**ticket_id,** booking_id(fk), ticket_number, barcode_number, status
Testimonial	**id**, user_id(fk), rating, review, created_at, displayed
Contact Message	**Id**, user_id(fk), name, email, subject, category, message, status, created_at, resolved_at, admin_notes

Figure 29. Database tables & columns

The Referential integrity constraints that are implemented on the *Vistaseat.com* system are displayed in the following schema:

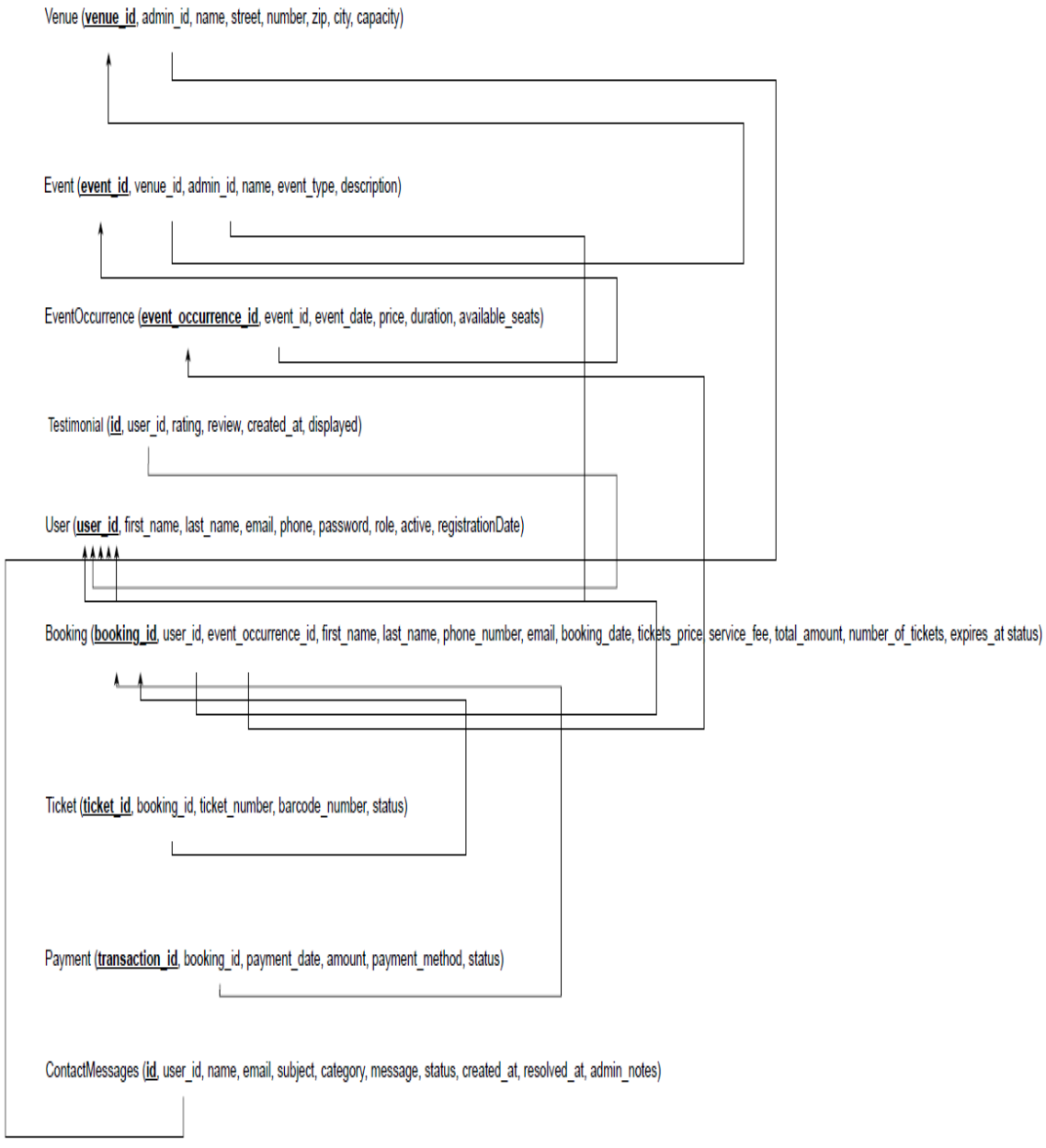


Figure 30. Relational model

4.4 Physical Database Schema

The above relational schema,, implemented with the Relational Database Management System (RDBMS) pgAdmin 4 of PostgreSQL, is shown below:

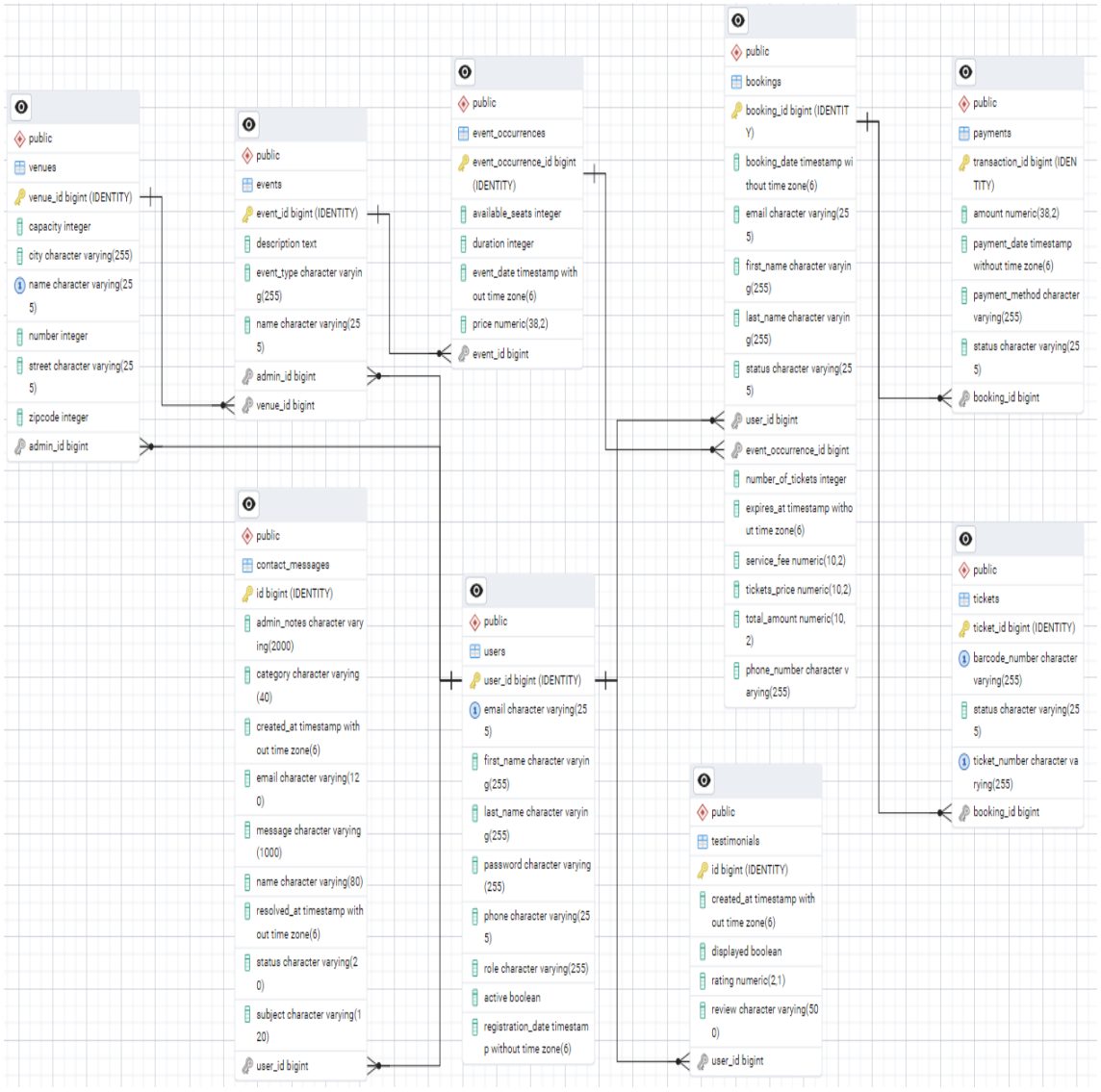


Figure 31. Relational model implemented in PostgreSQL(pgAdmin 4)

5. System Presentation

As we mentioned earlier, *Vistaseat.com* is an integrated seat reservation and ticketing system for events and performances. It operates under two distinct modes: **User Mode** and **Administrator Mode**.

Figure 32. Vistaseat.com logo

5.1 User Mode

User Mode is available for both registered users and guests. Guests are anonymous users who have limited access. They can experience the basic functionality of the system such as, browse

events, make bookings, and submit contact forms without having to submit credentials and register. Alternatively, they can choose to sign up and create an account. Registered users are authenticated users who access the system using a unique username and password and can perform all extra operations such as browse their order history and submit testimonials through their account. Both the sign in and sign up actions are performed by pressing the *"Connect"* button positioned at the top right corner of the header.

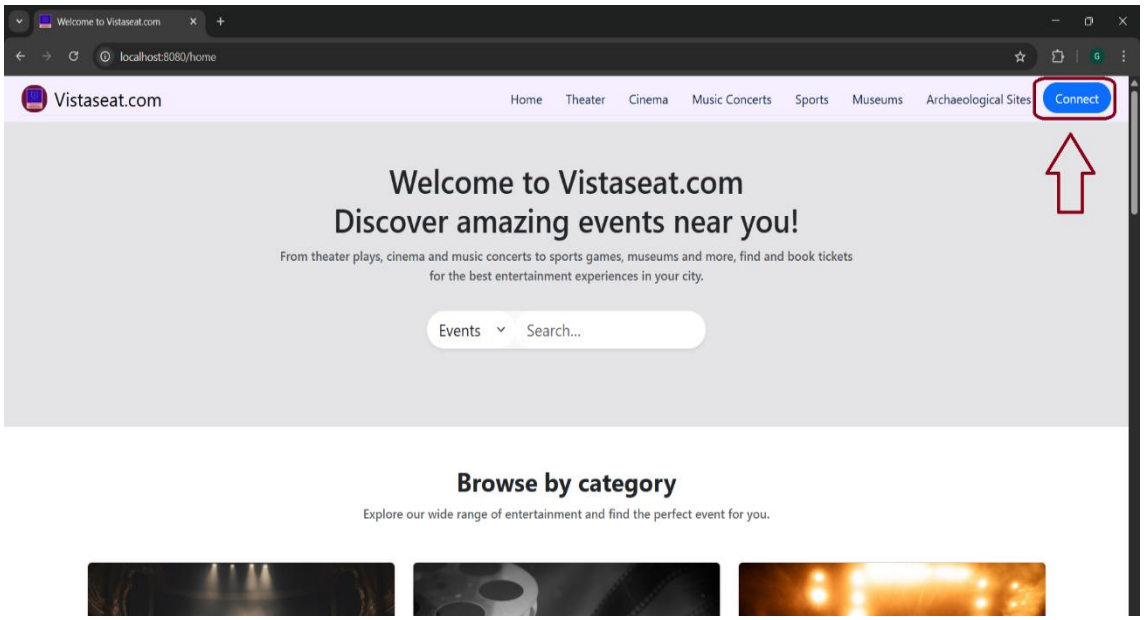


Figure 33. Landing page

5.1.1 Sign In

Users who have already created an account can sign in using their email and password.

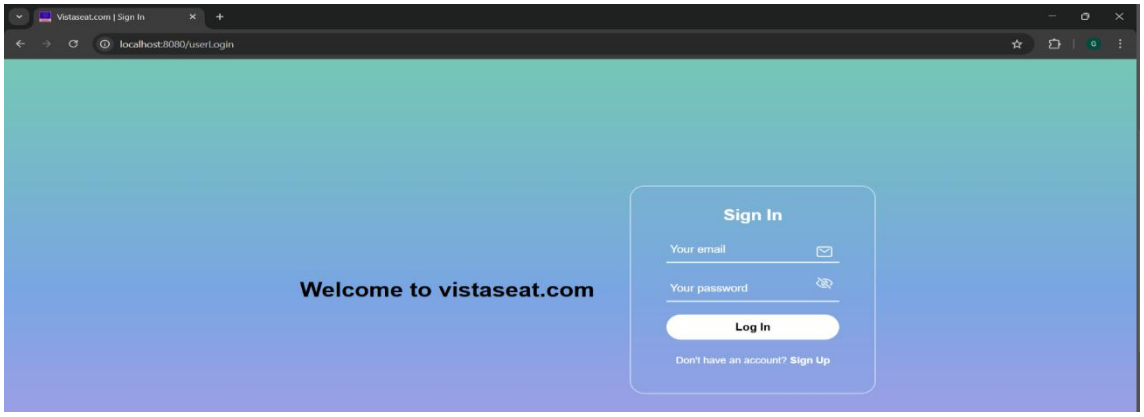


Figure 34. Sign In form

5.1.2 Sign Up

Guests can opt to sign up and create an account to experience the full functionality of the system. To sign up, they must provide the following information:

- First Name / Last Name: 2–30 characters. Letters (A–Z, a–z), apostrophes ('), and hyphens (-) allowed.
- Phone Number: 7–20 characters. Digits (0–9), spaces, dashes (-), parentheses (()), and the plus sign (+) allowed.
- Email: Must follow a valid email format (e.g., example@domain.com).
- Password: Minimum 8 characters. Must include at least one lowercase letter, one uppercase letter, one digit, and one of the following special characters: ! @ # $ _.

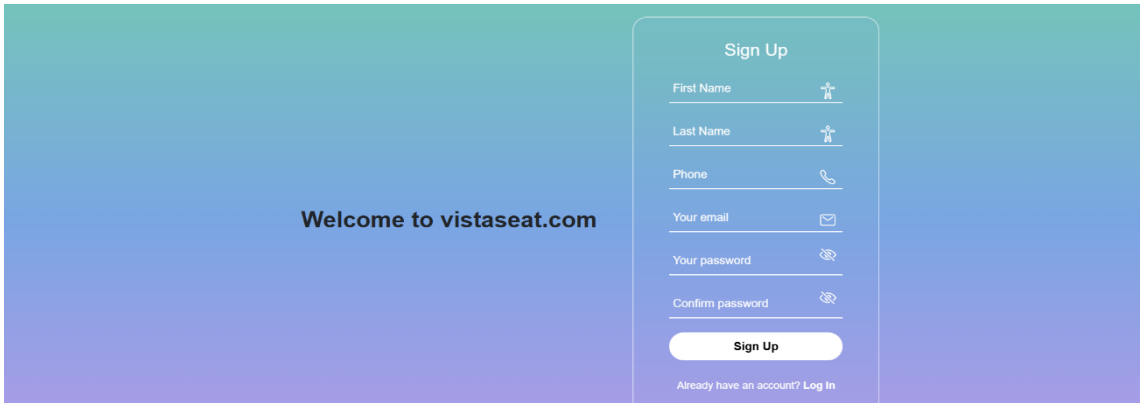


Figure 35. Sign Up form

5.1.3 Browsing Events

Multiple ways are provided for users to browse events. They can search for events by event name or venue name

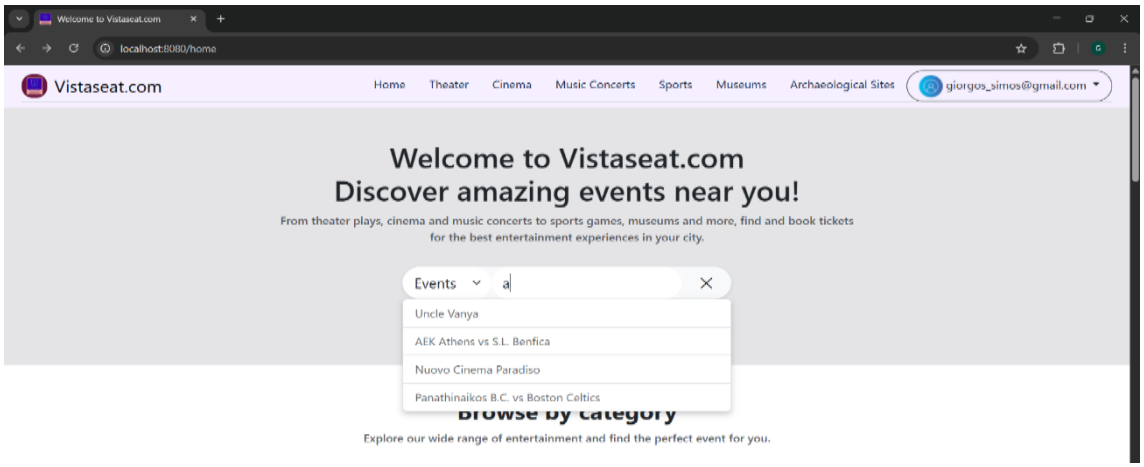


Figure 36. Browse events by event name

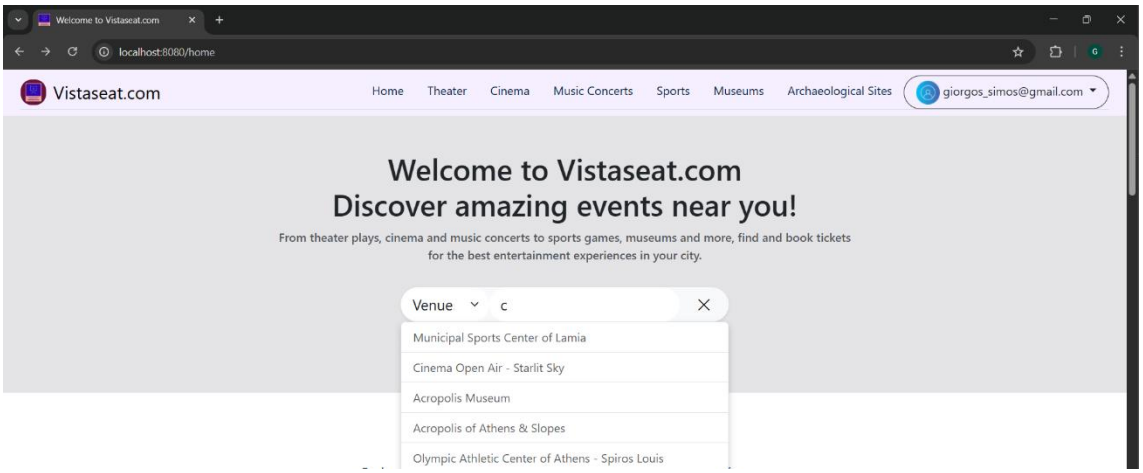


Figure 37. Browse events by venue name

Alternatively, users can browse events by choosing one of the event categories. Also, browsing events through categories is available through the header and the footer of the system. The supported categories are:

- Theater
- Cinema
- Music Concerts
- Sports
- Museums
- Archaeological Sites

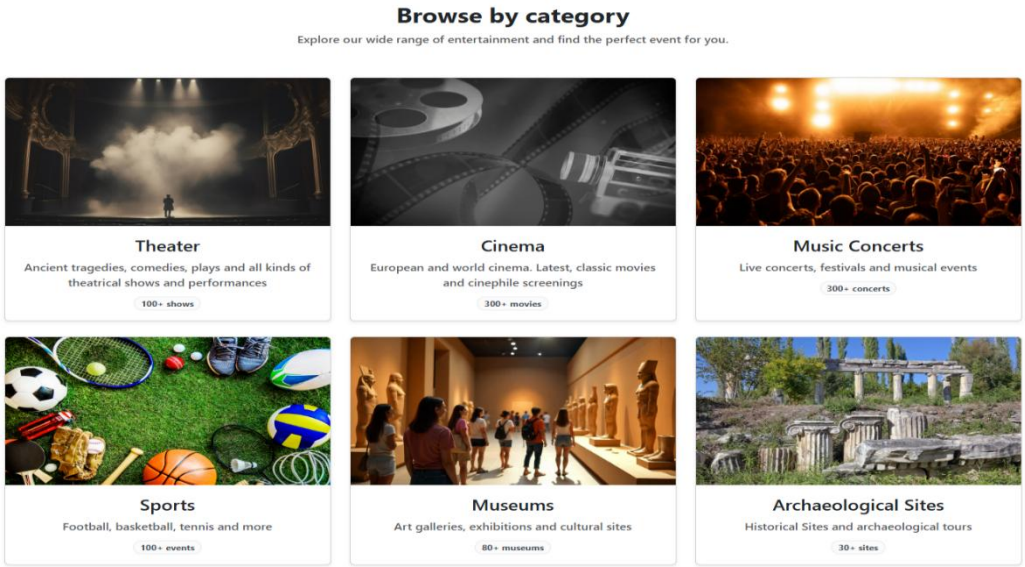


Figure 38. Browse events by category

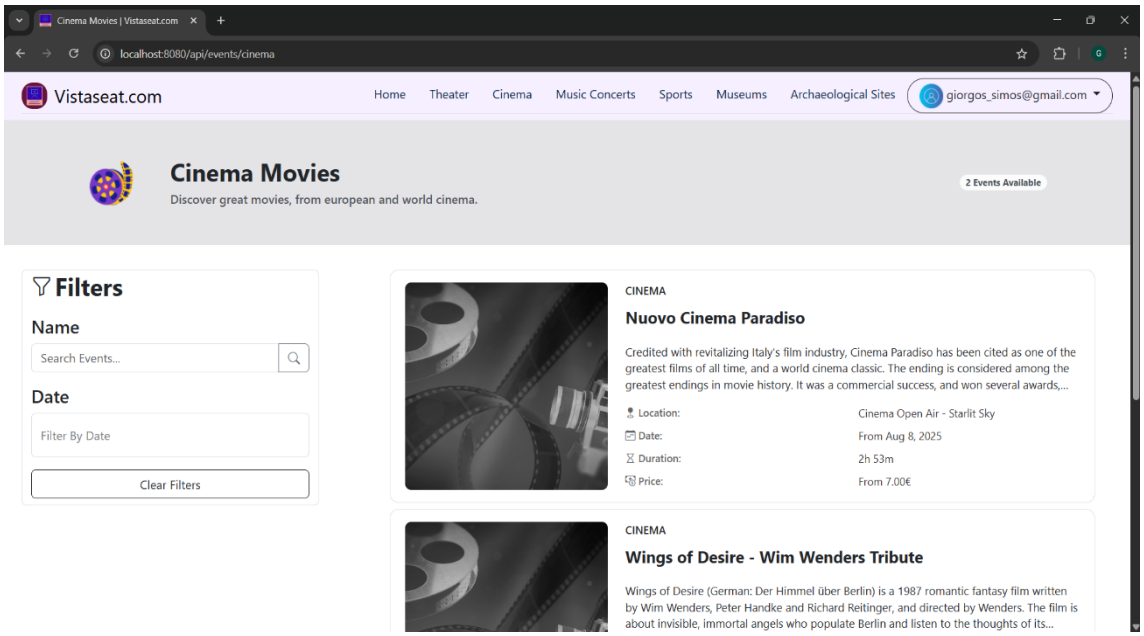


Figure 39. Available events (Cinema)

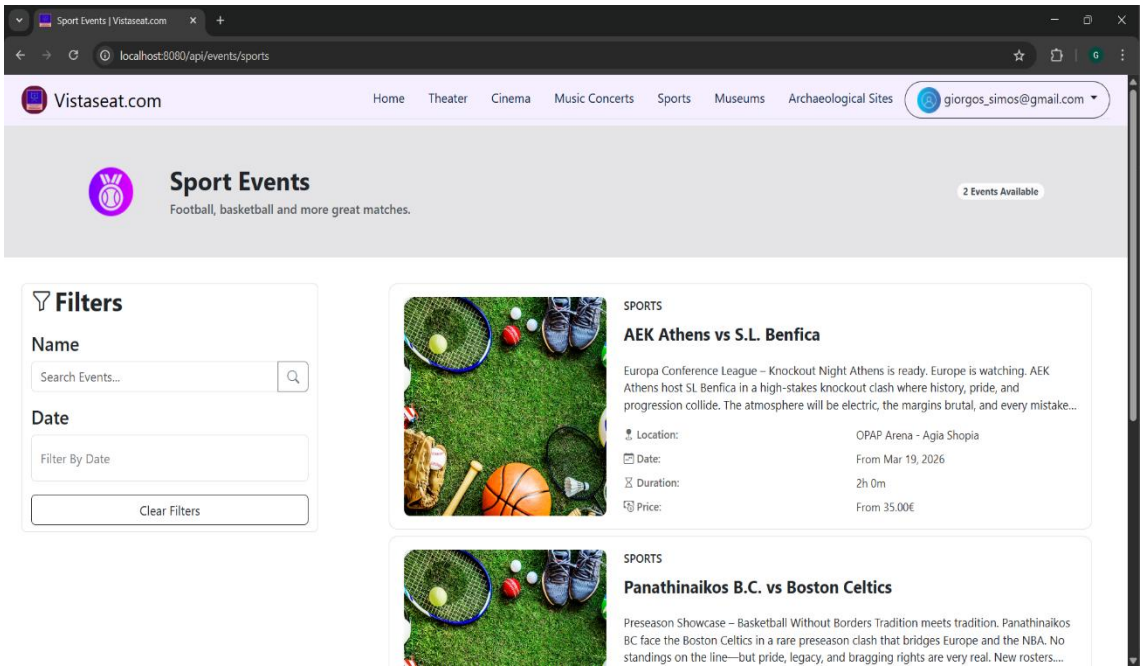


Figure 40. Available events (sports)

Also, they can browse through all featured events and venues, by using a scroll widget

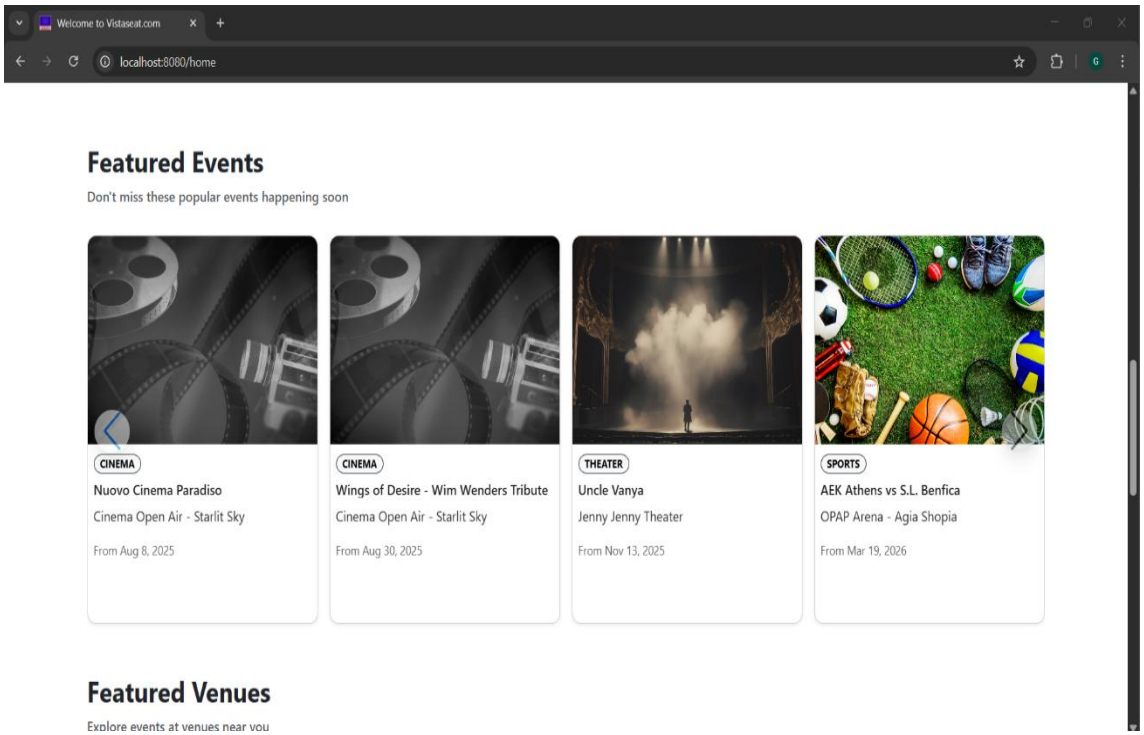


Figure 41. Featured events section

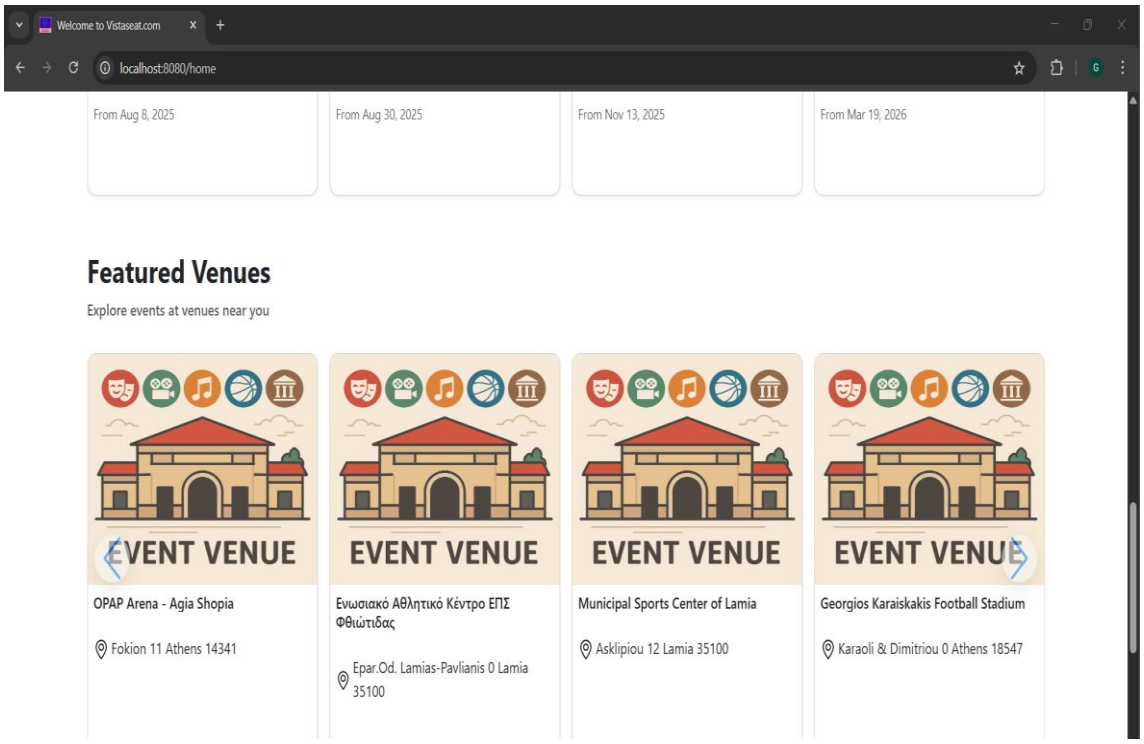


Figure 42. Featured venues section

5.1.4 Select Tickets for an Event

After a user has selected an event, they are redirected to the event details page. In that page they can view information such as the venue that the event is held, duration, price and a brief description about the event.

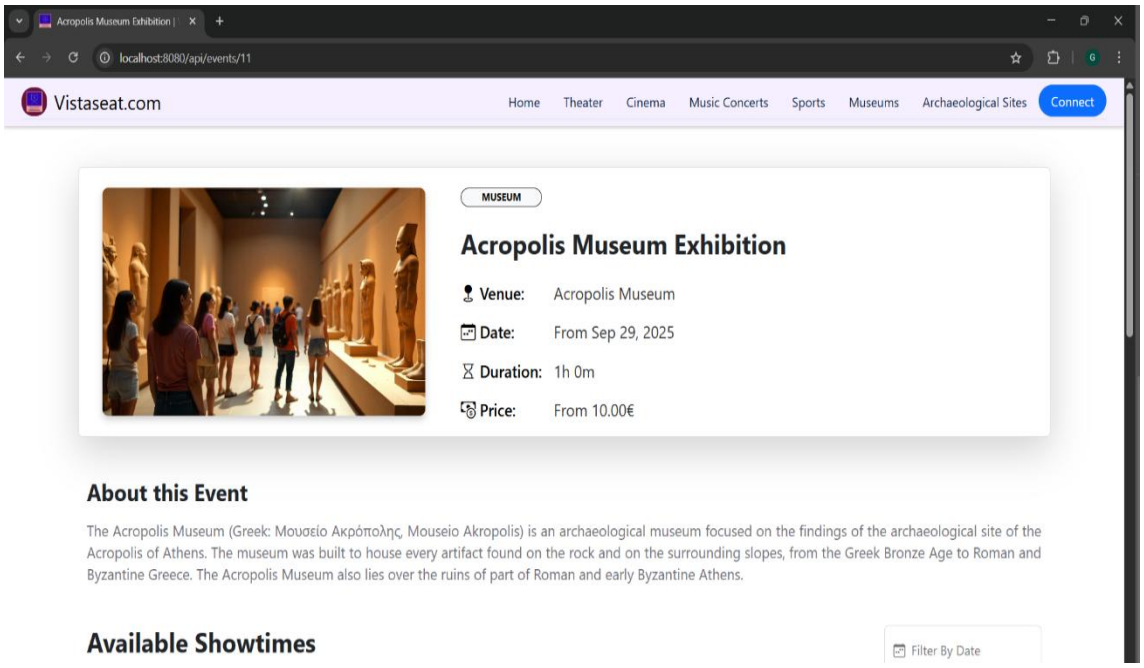


Figure 43. Select tickets section

In the Available Showtimes section, all event occurrences are displayed in chronological order, starting with the most recent. Users can further filter the list by selecting a specific date or a date range.

Ticket availability for each occurrence card is indicated by a colored ribbon on the left side of the card and an availability badge positioned at the center-right. Availability is classified into four color-coded states:

- Available: > 50%, green
- Limited: 50–25%, grey
- Few Left: 25-0.1%, orange
- Sold Out: 0%, red

When an occurrence has no remaining tickets, that means it is sold out and the “Tickets” button on the right of the card is disabled.

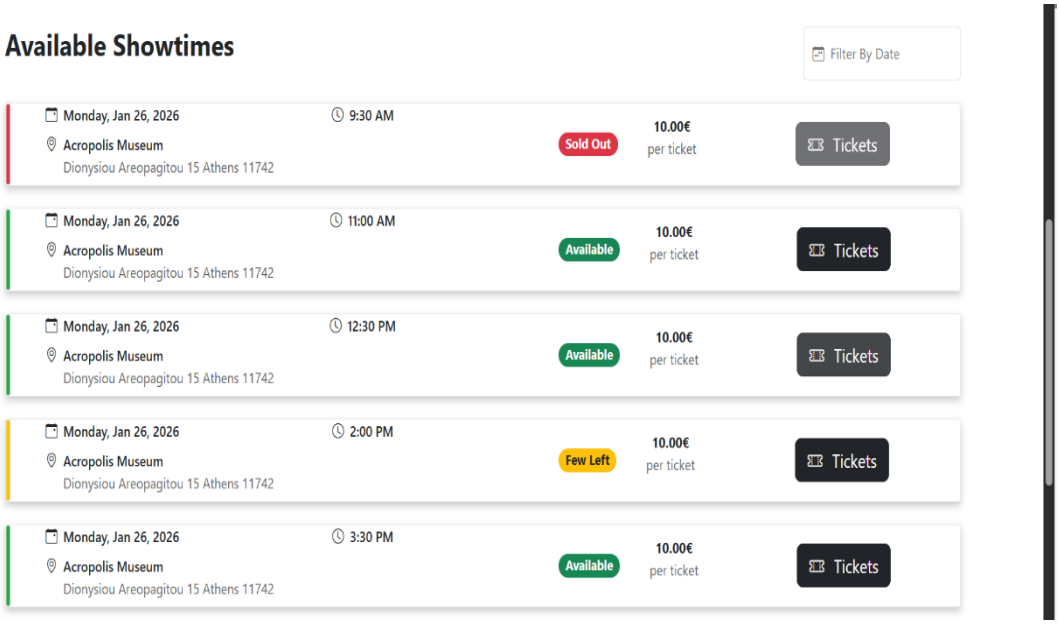


Figure 44. Available showtimes for a selected event

The maximum number of tickets, a user is allowed to book per order, is 10.

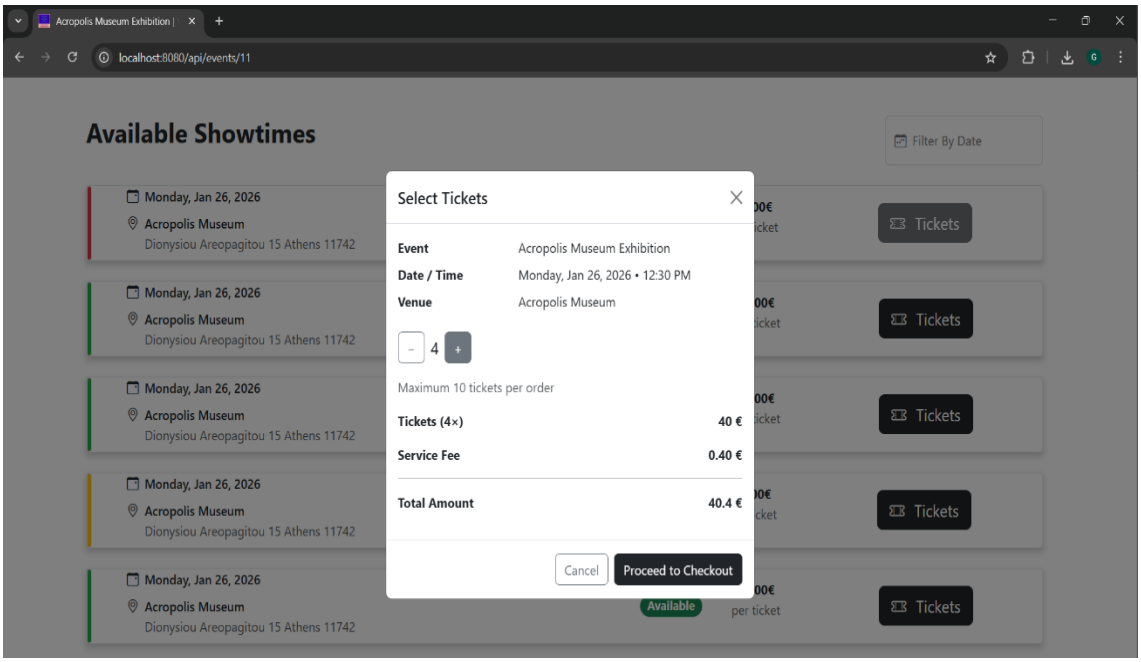


Figure 45. Select number of tickets

When a guest user initiates a reservation, the system optionally prompts them to log in.

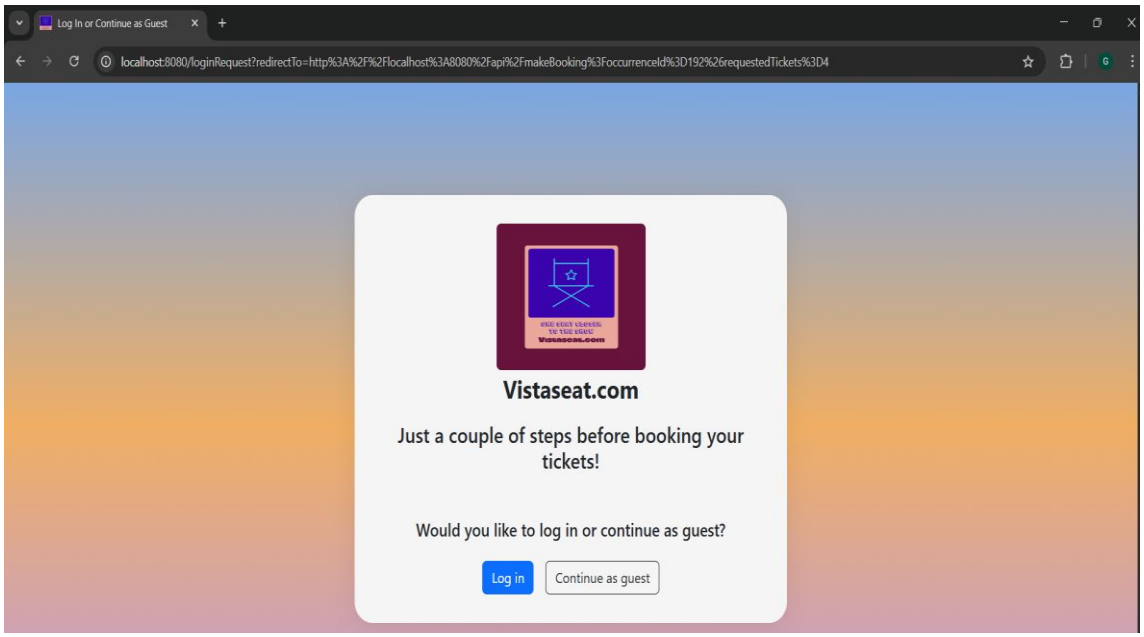


Figure 46. Prompt user to log in before booking (optional)

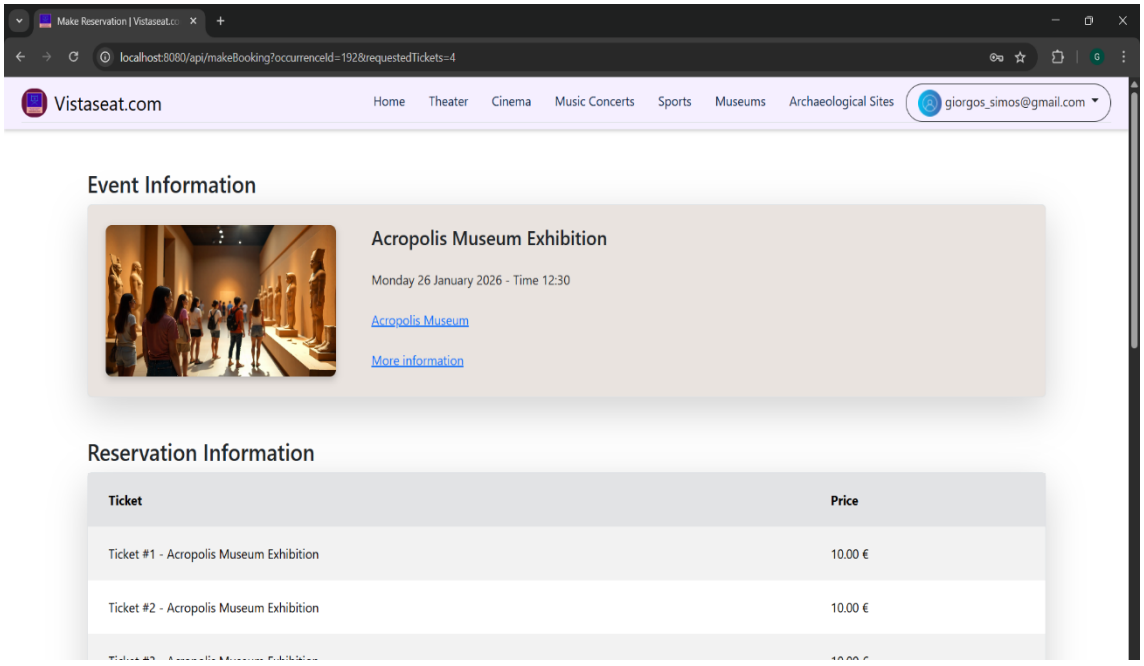


Figure 47. Order overview

Logged-in users have their contact details automatically pre-filled in the reservation form, eliminating the need to re-enter this information for each new reservation and reducing user input friction. They simply must accept the Terms of Use and press the “Continue to Payment” button.

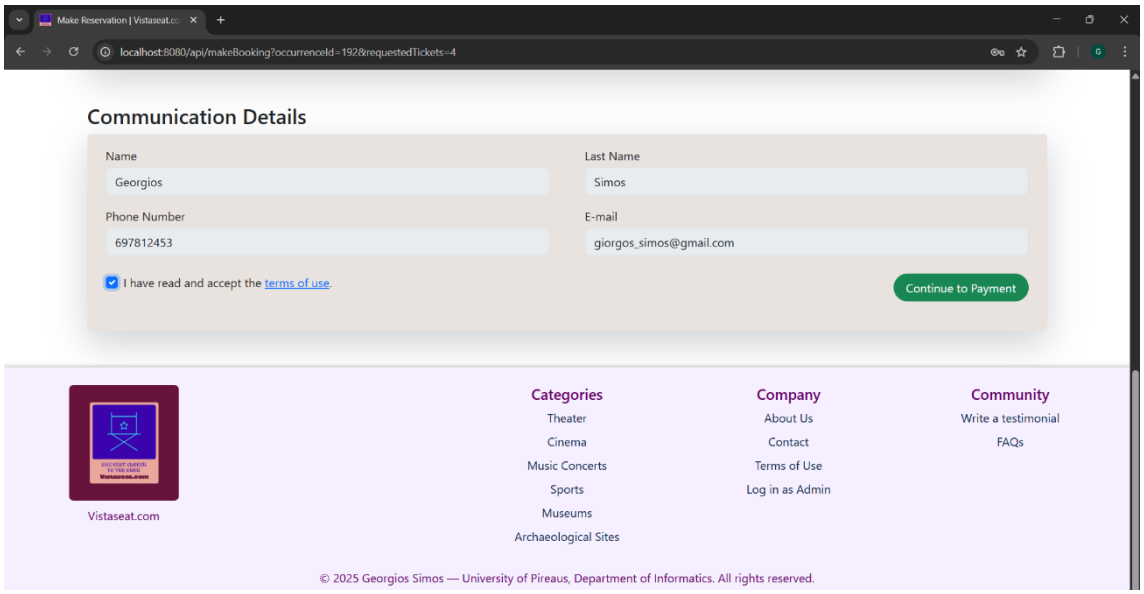


Figure 48. Client communication details

5.1.5 Payment Mehtods

After successfully completing the reservation stage, the user is redirected to the payment page. At this stage, a booking has been created with pending status while it awaits payment confirmation. It remains active for 15 minutes. If the payment is not confirmed within 15 minutes, the reservation is cancelled. The payment page displays the total cost (ticket price plus service fee), ticket quantity, event date, and venue.

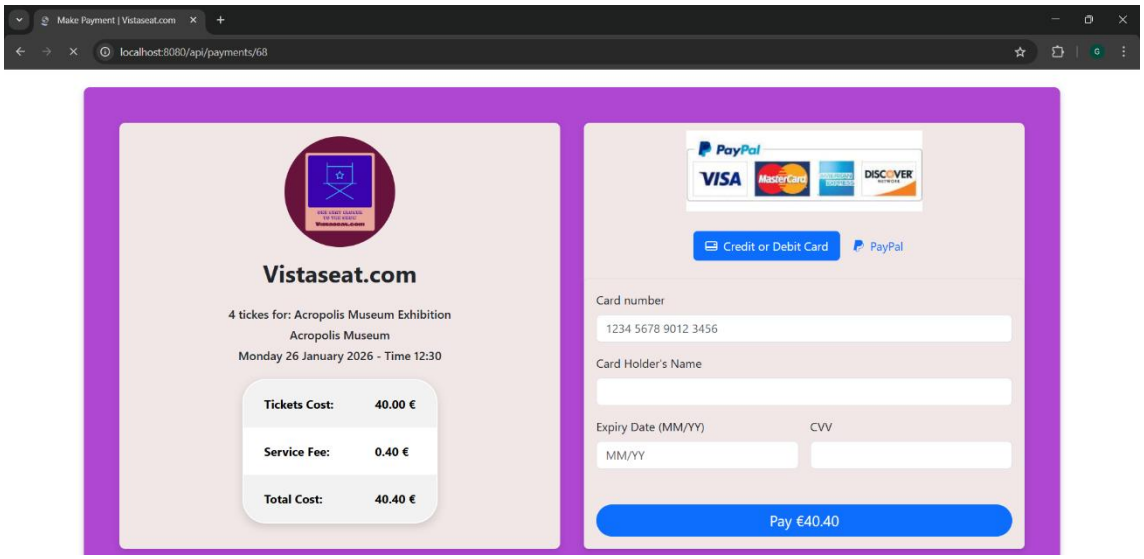


Figure 49. Payment overview

The available payment methods are credit or debit card, and PayPal.

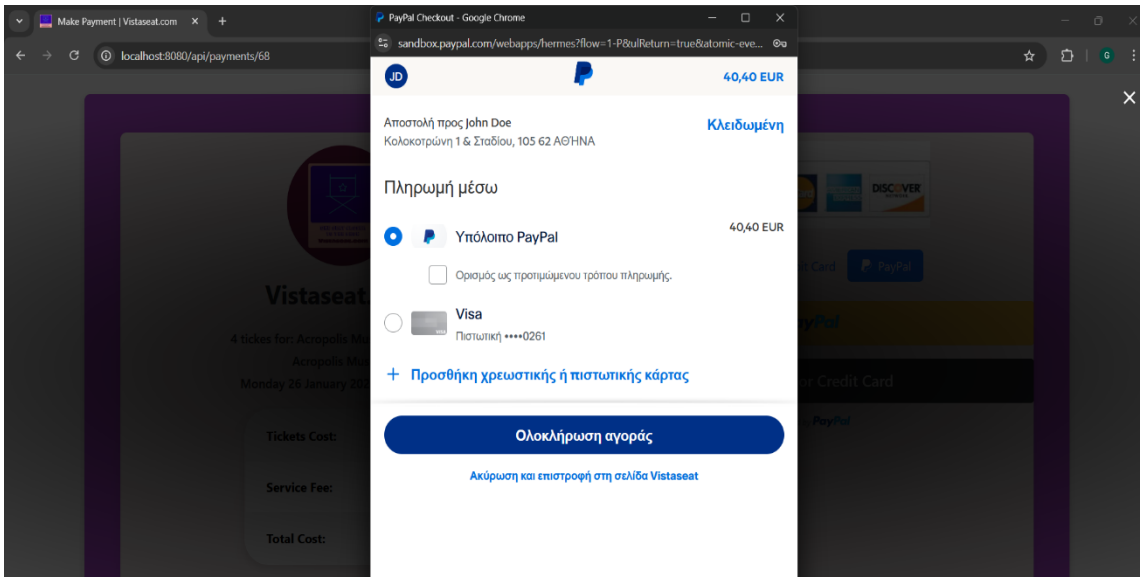


Figure 50. PayPal integration

Once the payment is completed, the booking is now confirmed.

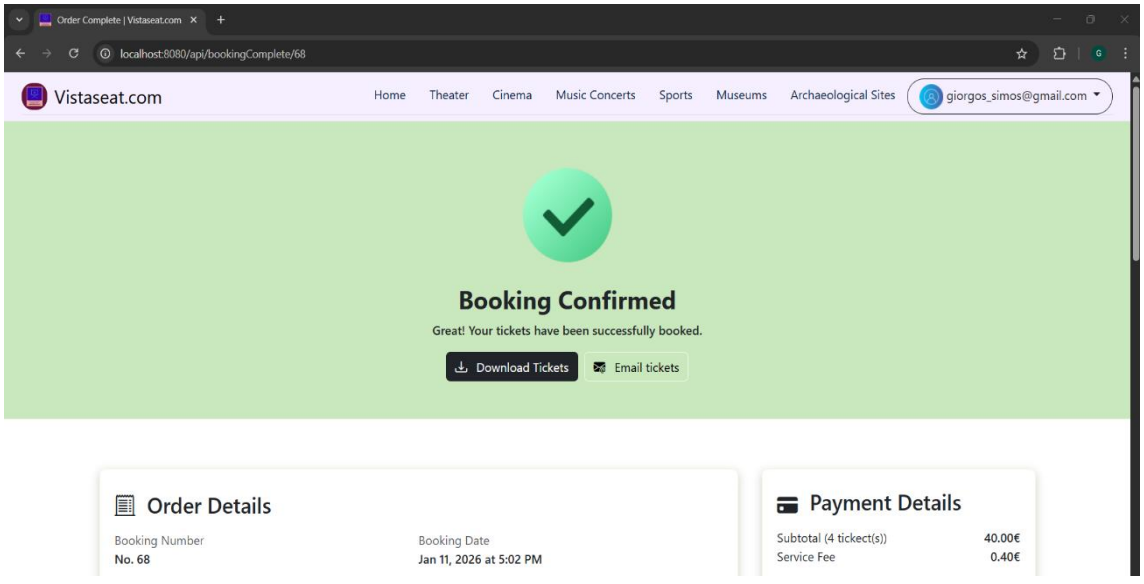


Figure 51. Booking confirmed page

Users can view additional reservation details, including order information, payment method, ticket details and event specifics.

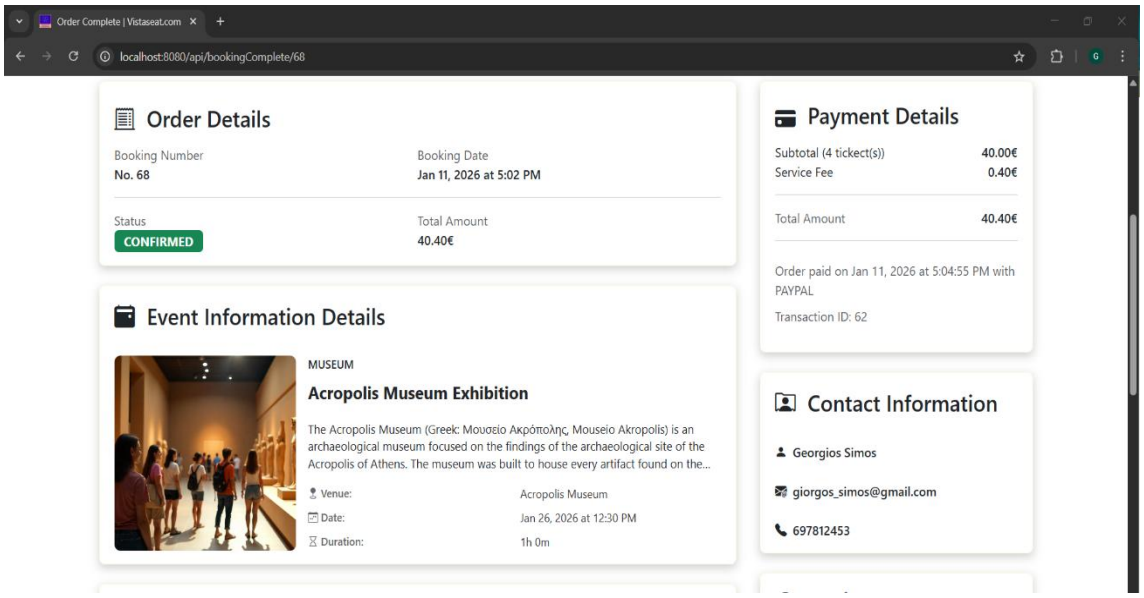


Figure 52. Event information details

Also, they can choose to download their tickets in pdf format.

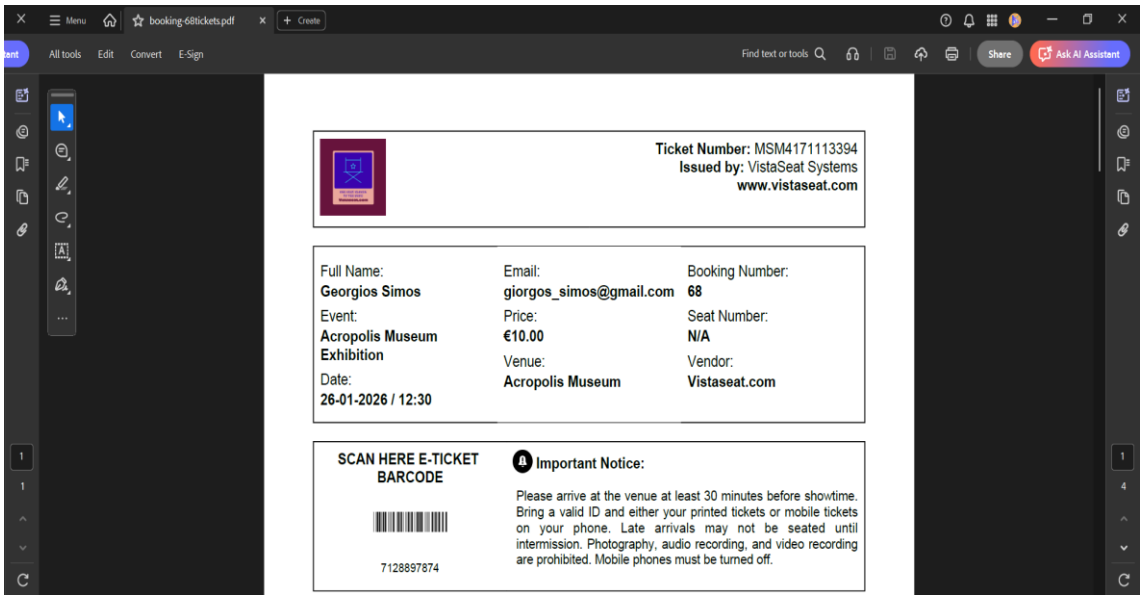


Figure 53. Tickets in PDF format

5.1.6 User Account

Logged in users can change their account details and view active or past orders.

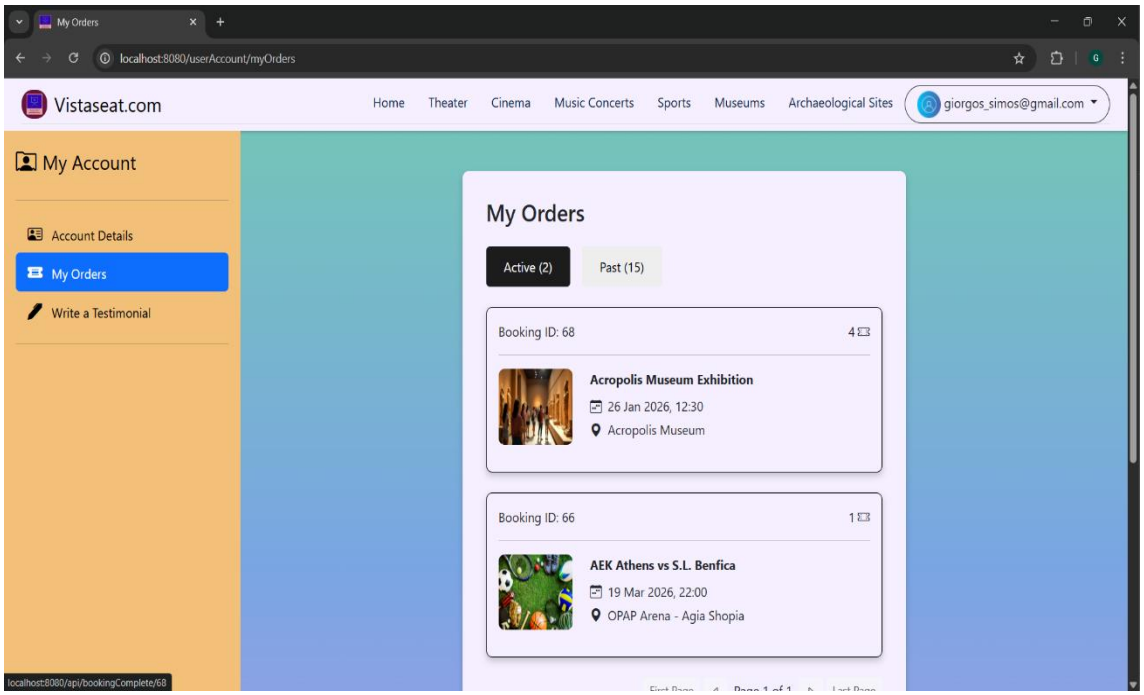


Figure 54. User account (My Orders)

Users can also submit testimonials to share their experience with the application. Testimonials are displayed in the testimonials section only after being approved by an administrator.

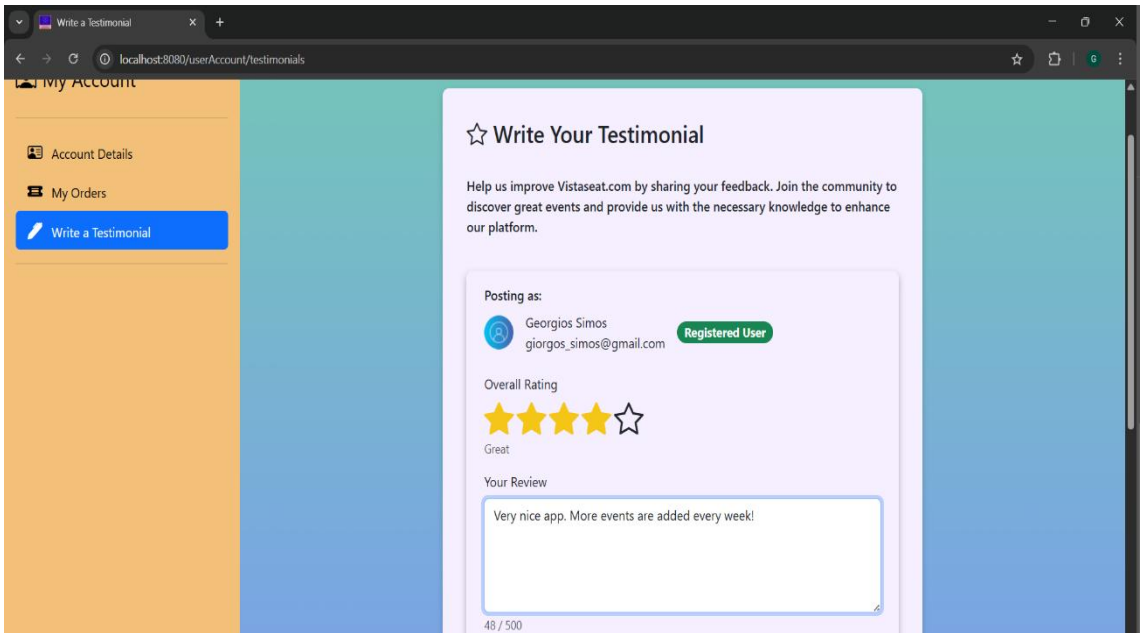


Figure 55. Write a testimonial

5.1.7 Testimonials Section

At the bottom of the home page, above the footer, a testimonials section is displayed. This section allows all users to view testimonials previously submitted by registered users.

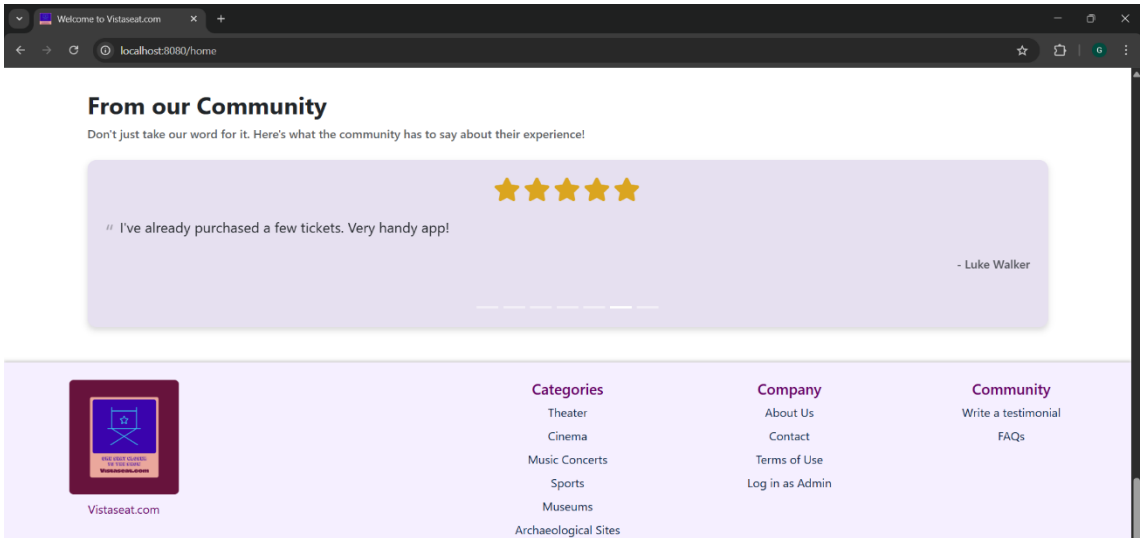


Figure 56. Testimonials section

5.1.8 Contact Form

All users, both guests and registered, can submit contact forms, notifying the administrators about any potential issues they encountered while using the system.

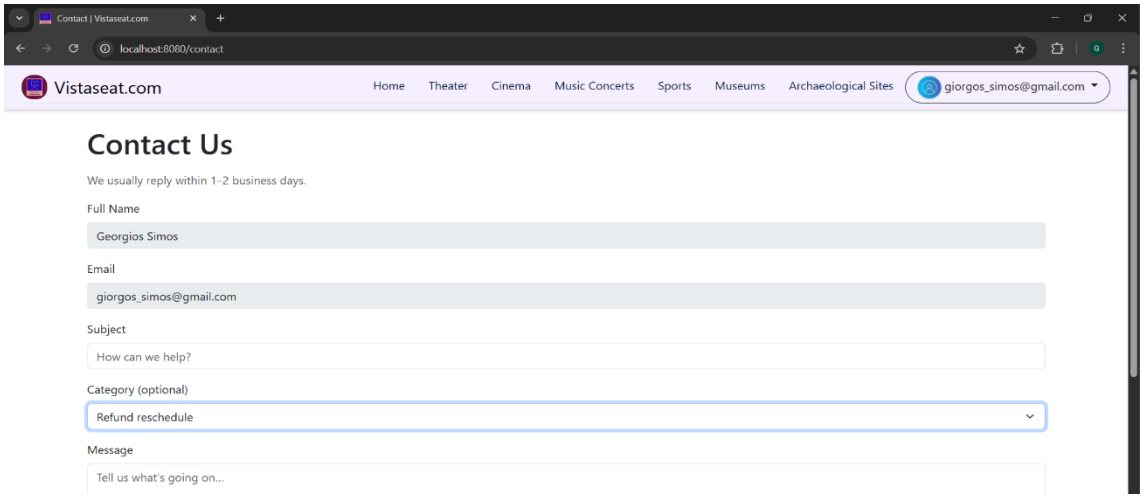


Figure 57. Contact form

The category of the issue can be one of the following:

- Ticket Issue
- Ask for a refund or reschedule a booking
- Account login issue
- Enquire for a possible partnership with *Vistaseat.com*
- Any other issue

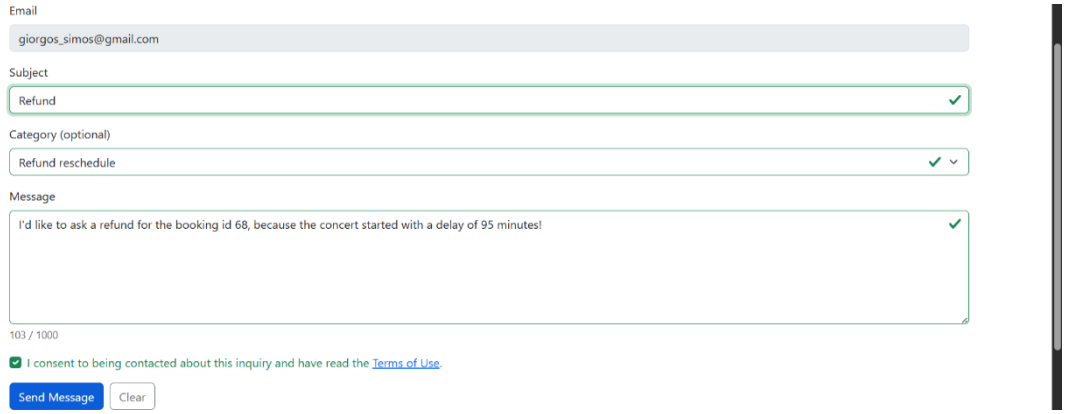


Figure 58. Contact forms can be submitted by all users (registered or guests)

Each time a contact form is submitted, an administrator is required to review the reported issue and provide a resolution.

5.1.9 Other Services

From the page footer, users can access additional sections that provide further information about the system, including the About Us page, Terms of Use, and Frequently Asked Questions (FAQs).

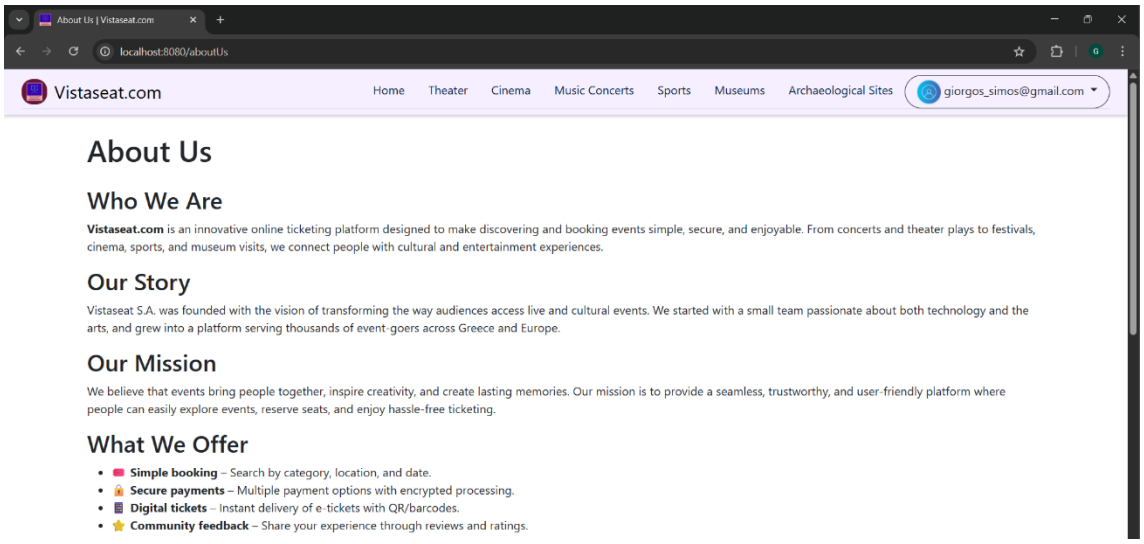


Figure 59. About Us section

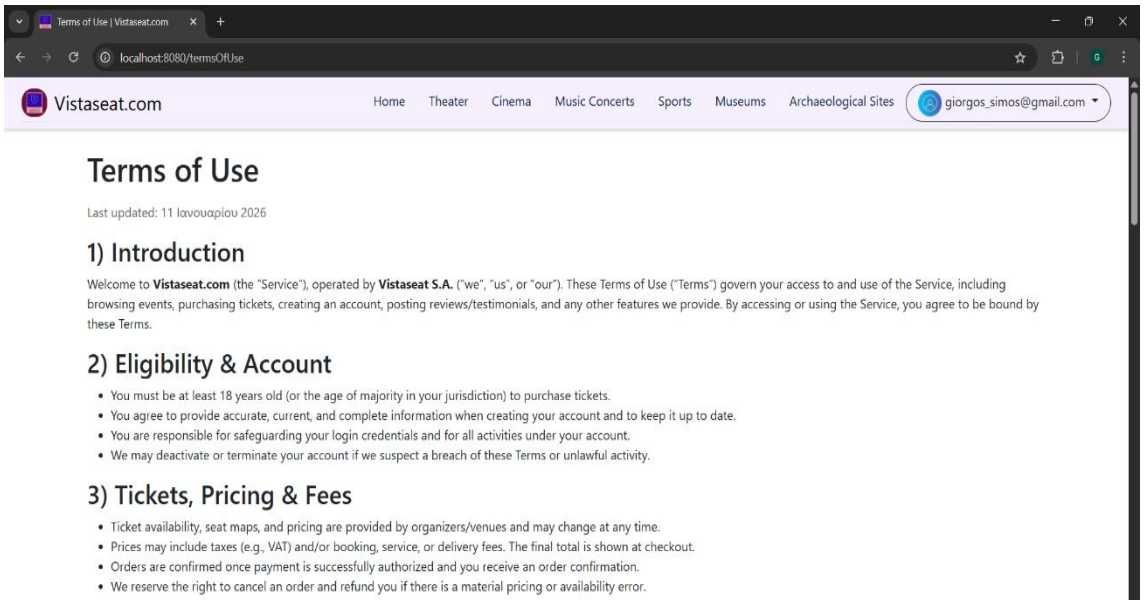


Figure 60. Terms of Use section

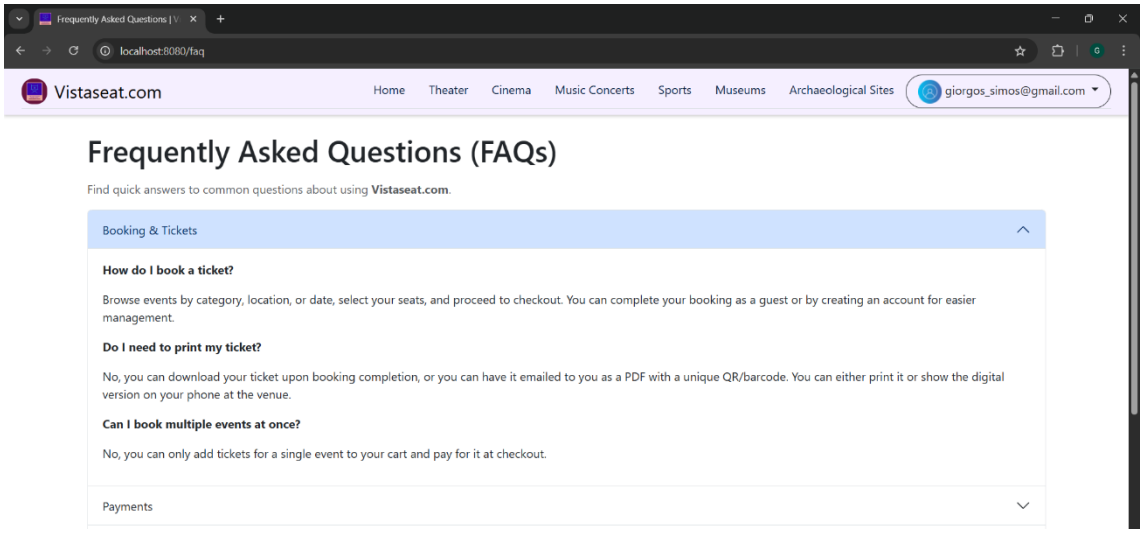


Figure 61. Frequently Asked Questions (FAQ) section

5.2 Administrator Mode

The objective of *Vistaseat.com* is not only to enable ticket reservations and purchases, but also to operate as a complete platform for managing hosted content, users, and user activity. Unlike User Mode, Admin Mode is only available to authenticated users with administrator roles.

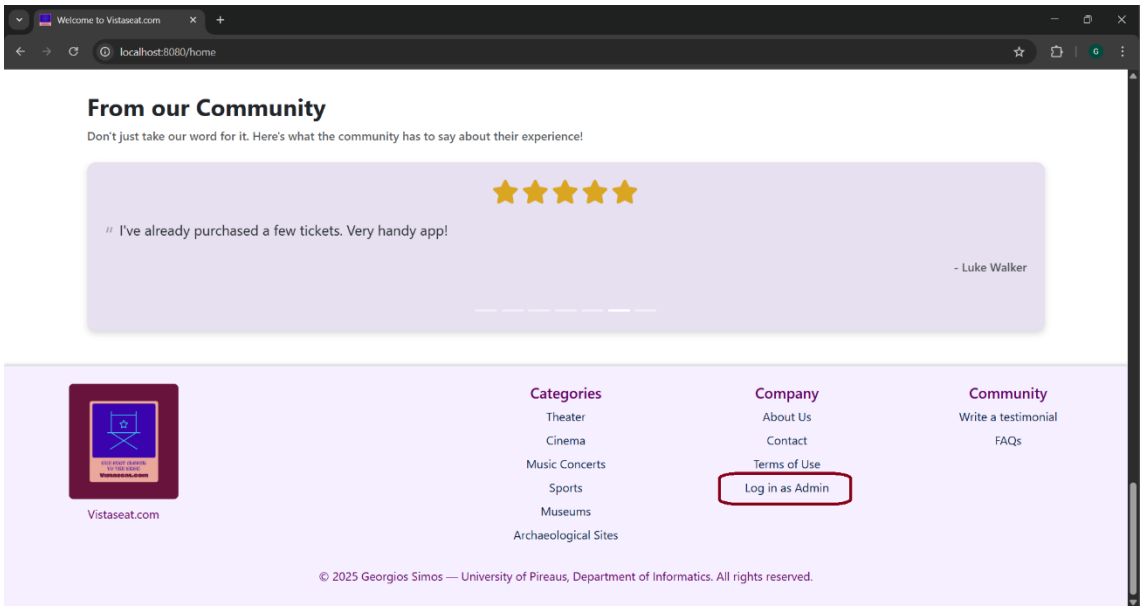


Figure 62. Log In as Admin

Admins can log in by providing their email address and password.

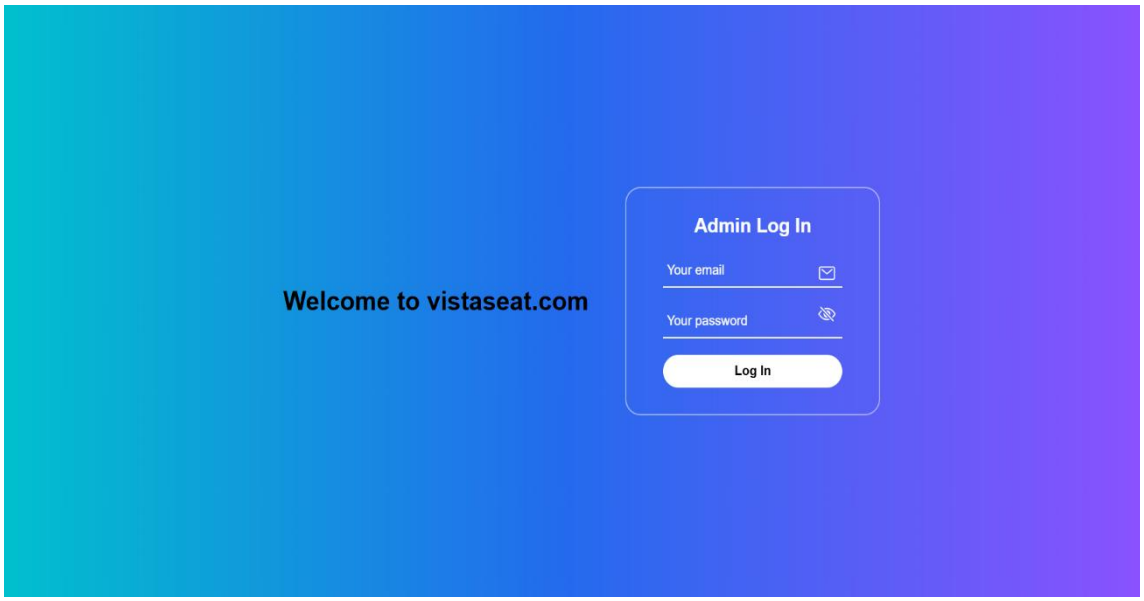


Figure 63. Admin Log In form

5.2.1 Admin Dashboard

Upon successful authentication, the administrators are redirected to the Admin Dashboard, where they can manage the application's content. Administrative functions include managing Users,

Venues, Events, Event Occurrences, Bookings, Testimonials, and Contact Messages that require resolution. On the top corner, the first and last name of the admin in visible. A sidebar on the left provides navigation, while the main panel displays summary cards with key system metrics, Including total users, events, venues, bookings and the total revenue the system has produced so far.

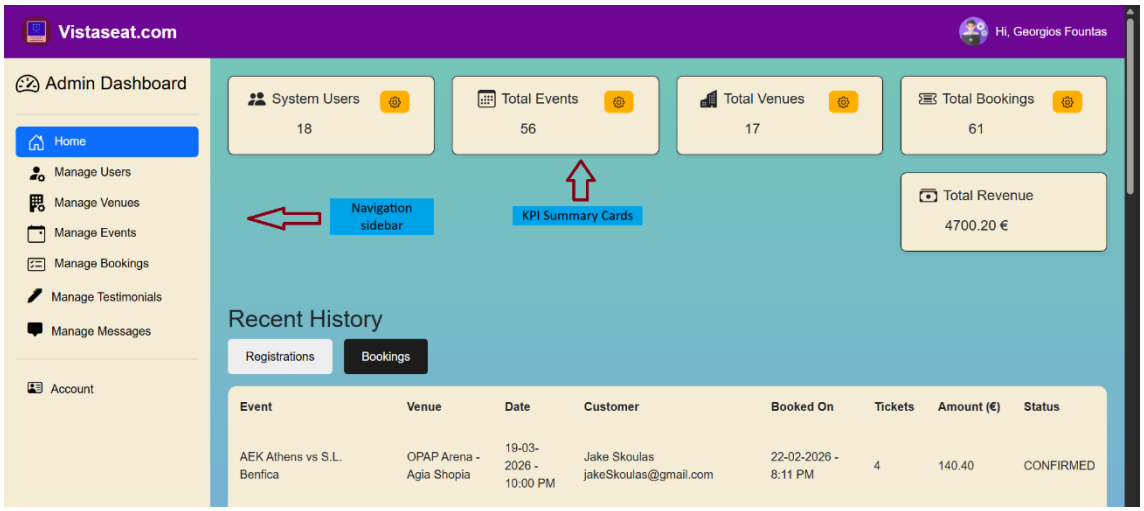


Figure 64. Admin dashboard

At the bottom of the dashboard, a Recent History section presents two tabs showing snapshots of recent system activity.

The first tab displays the 10 most recent user registrations.

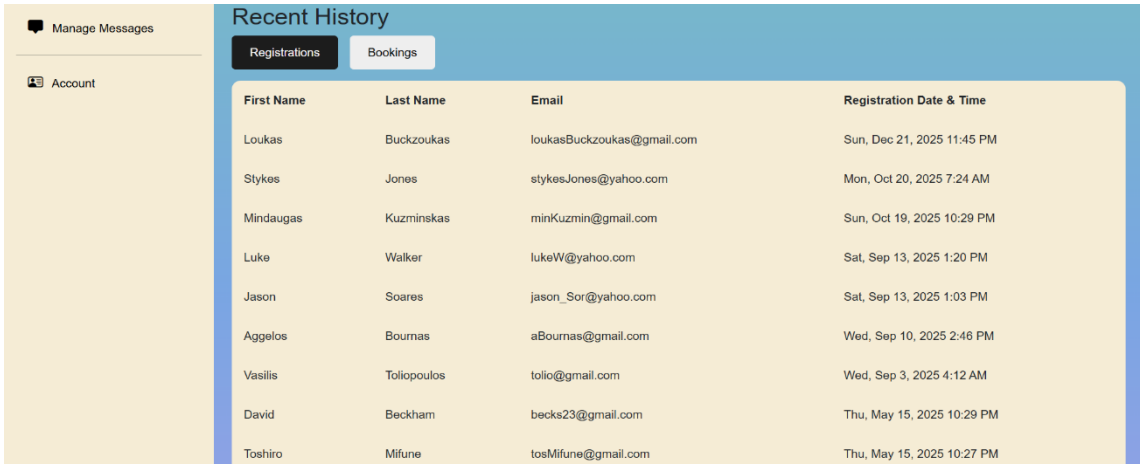


Figure 65. Recent history tab (Registrations)

The second tab displays the 10 most recent bookings.

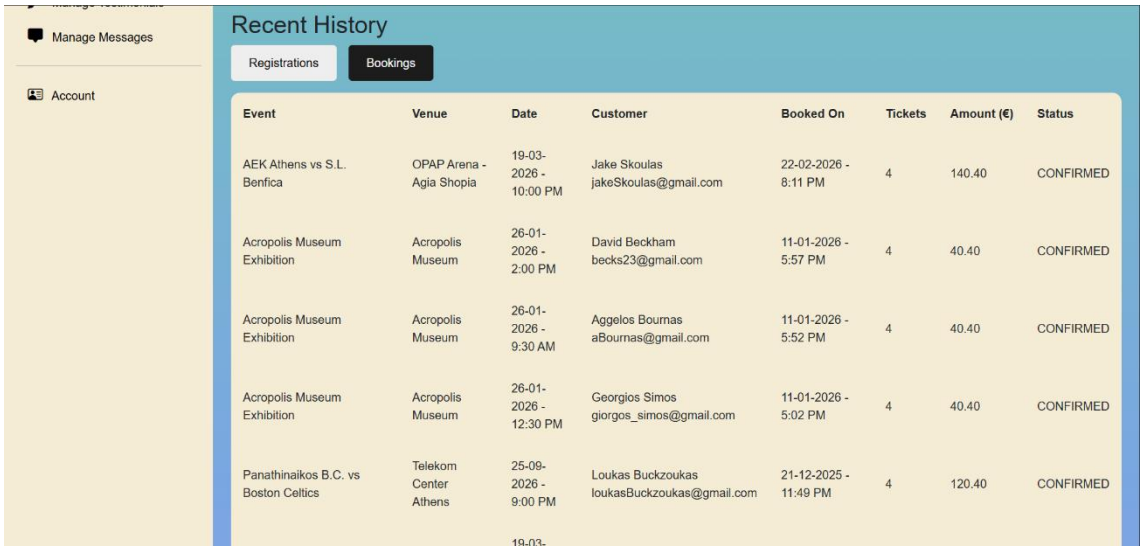


Figure 66. Recent history tab (Bookings)

5.2.2 Admin Account Section

In the account section, admins can modify their account information except for their email address, which acts as a unique account identifier.

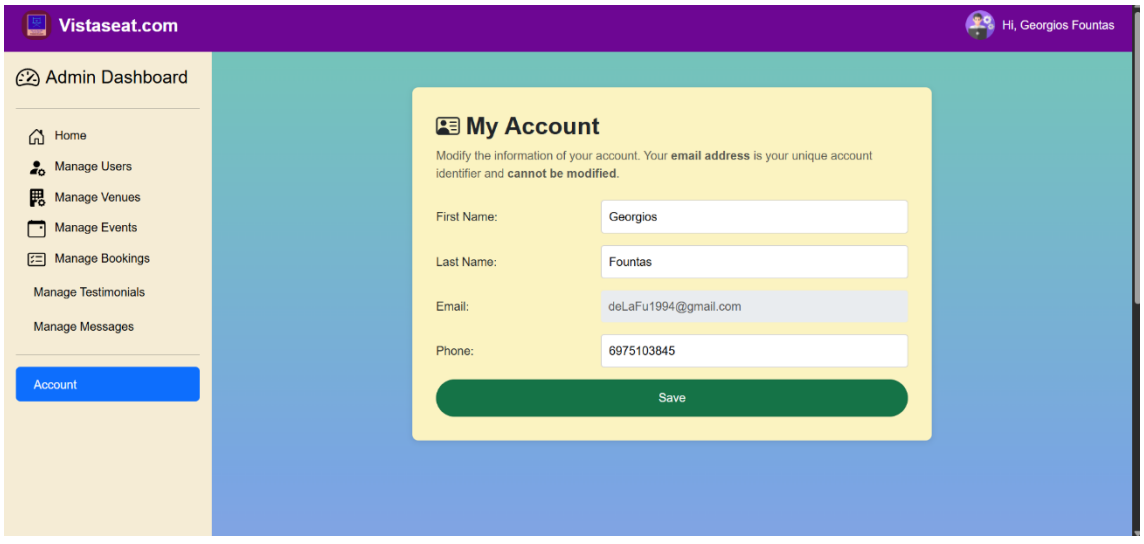


Figure 67. Admin account section

5.2.3 Manage Users Section

In the Manage Users section, all registered users are displayed. The information displayed is first name, last name, email, user role, phone, status and registration date. The list can be filtered by

user role (registered user or administrator). Administrators can delete existing users; however, users with at least one booking cannot be deleted. Administrators may also change a user's status (active or inactive) in cases of Terms of Use violations.

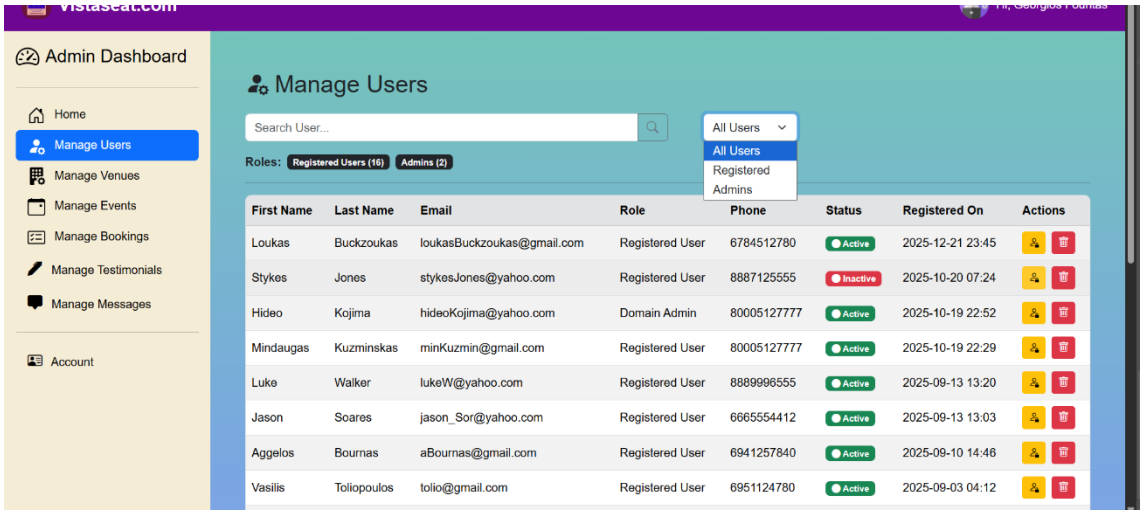


Figure 68. Manage users section

5.2.4 Manage Venues Section

In the Manage Venues Section, all the stored venues are displayed. The information displayed is venue name, address (street, number, zipcode and city), venue capacity and count of scheduled events.

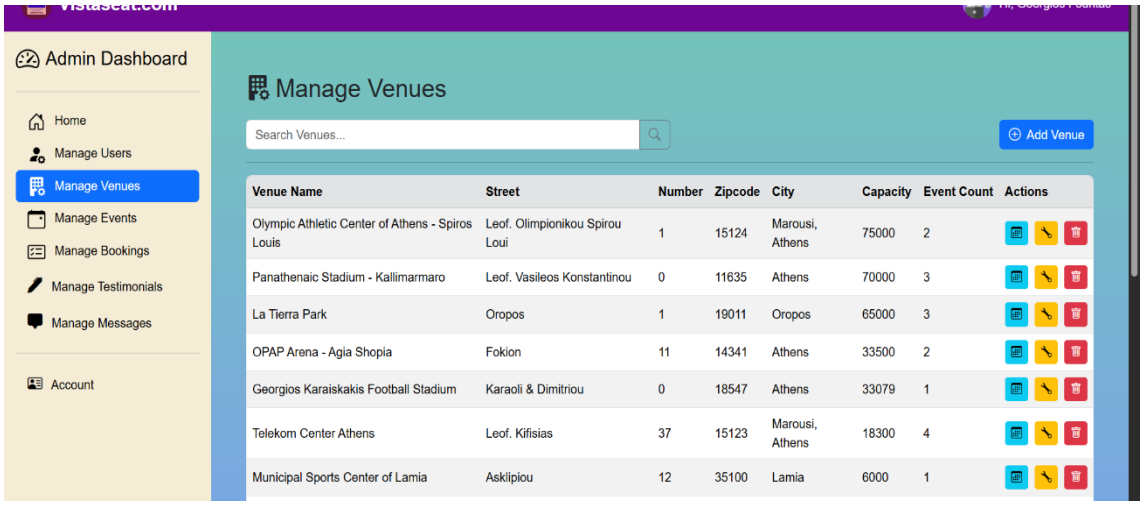


Figure 69. Manage venues section

The available operations are:

- Add a new venue by filling in the required form.

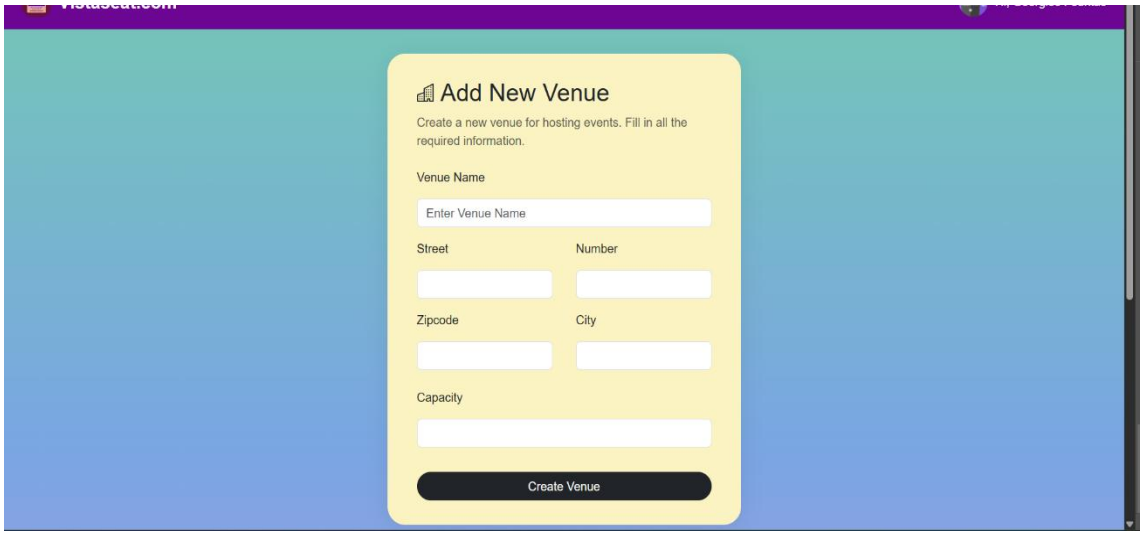


Figure 70. Add a new venue form

- Edit an existing venue.

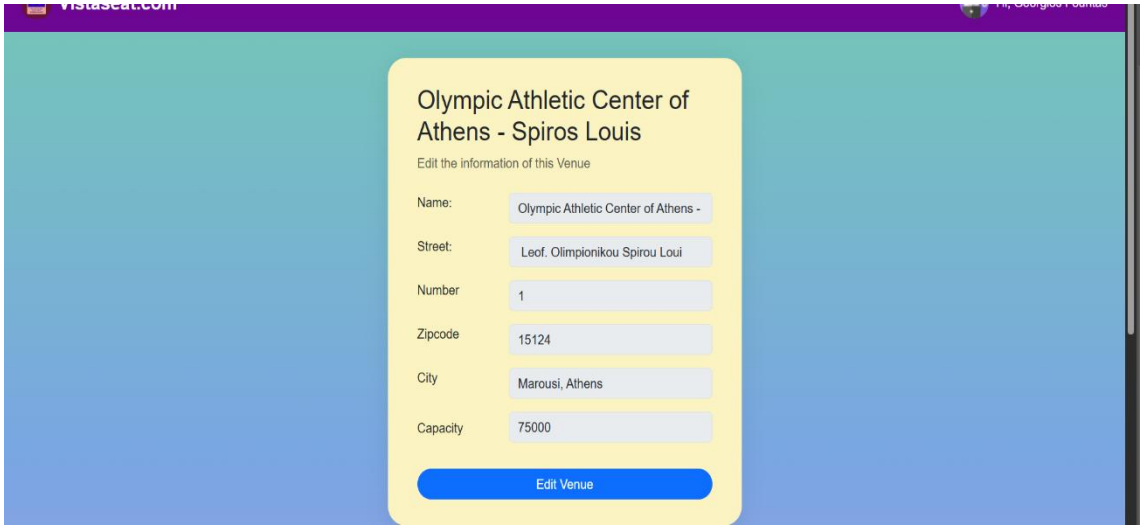


Figure 71. Edit an existing venue

- Delete an existing venue. This operation is only allowed for venues that don't have any events scheduled.
- View scheduled events for a selected venue. The calendar next to the name of the venue, is a clickable button that allows us to display all the occurrences for the selected venue.

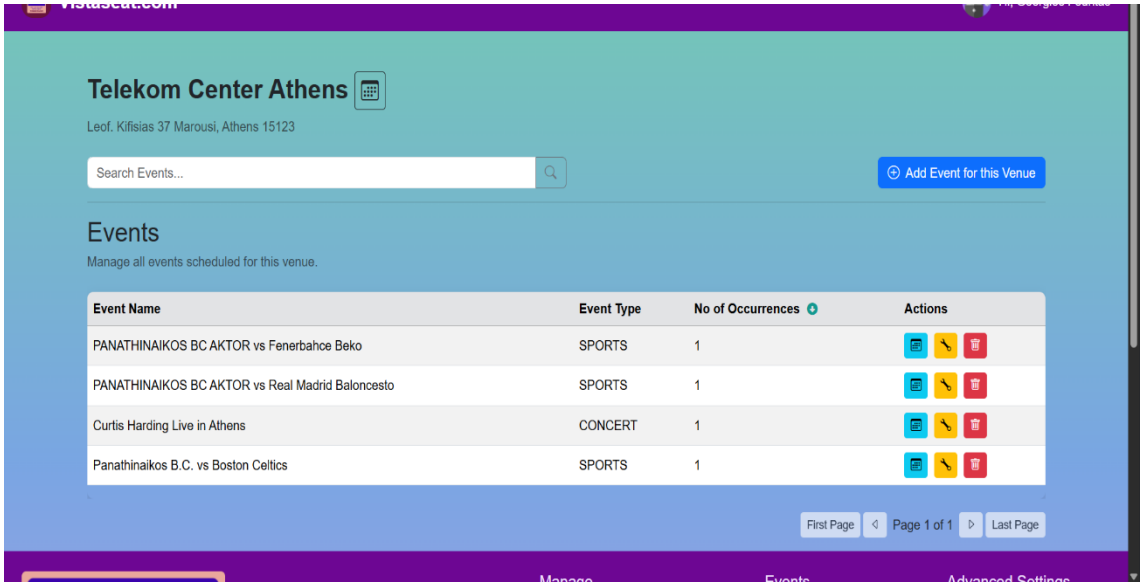


Figure 72. View scheduled events for a selected venue

There, the admin can find information such as the count of total occurrences, count of total bookings and total revenue generated from this venue.

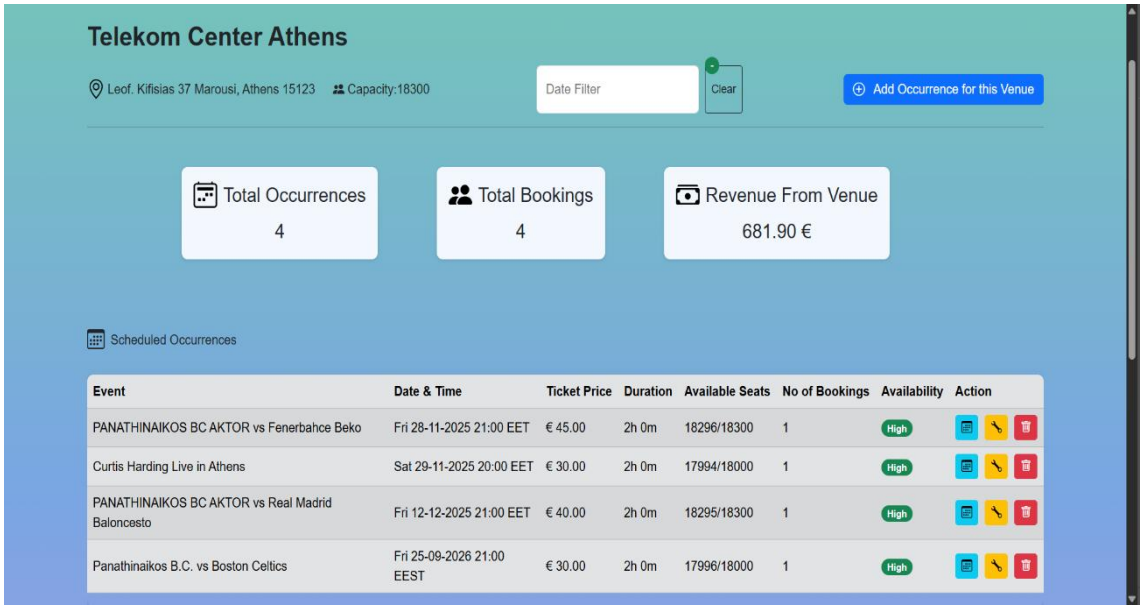


Figure 73. View scheduled occurrences for a selected venue

They can also choose to add a new occurrence for any of the scheduled events of the selected venue.

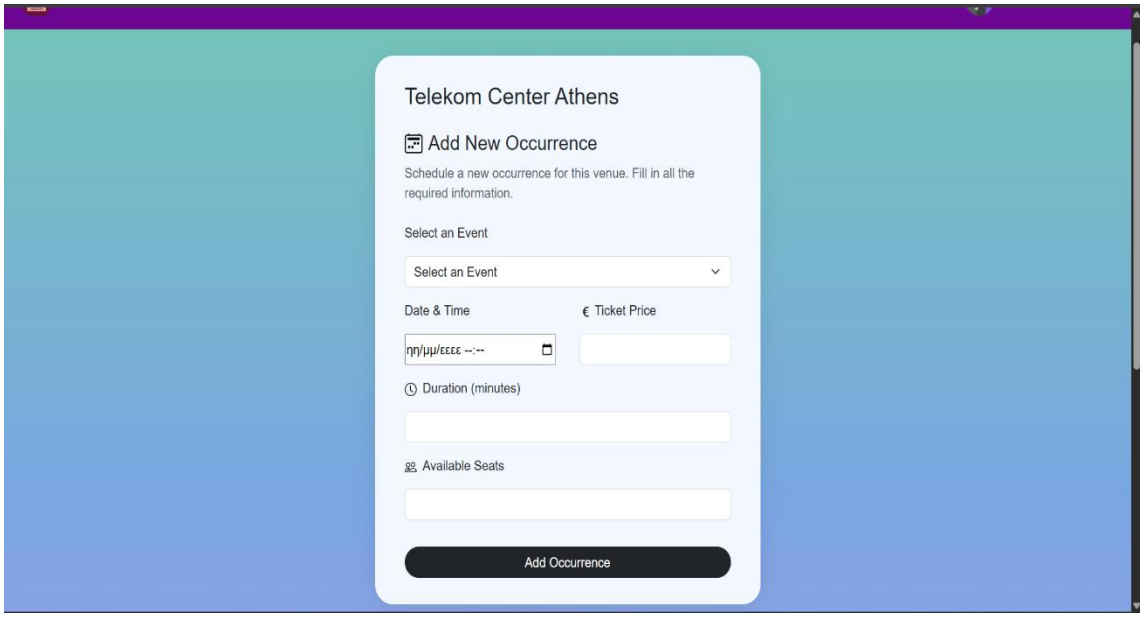


Figure 74. Add a new occurrence for a selected venue

Or they can choose to display all the occurrences for a selected event of the specified venue.

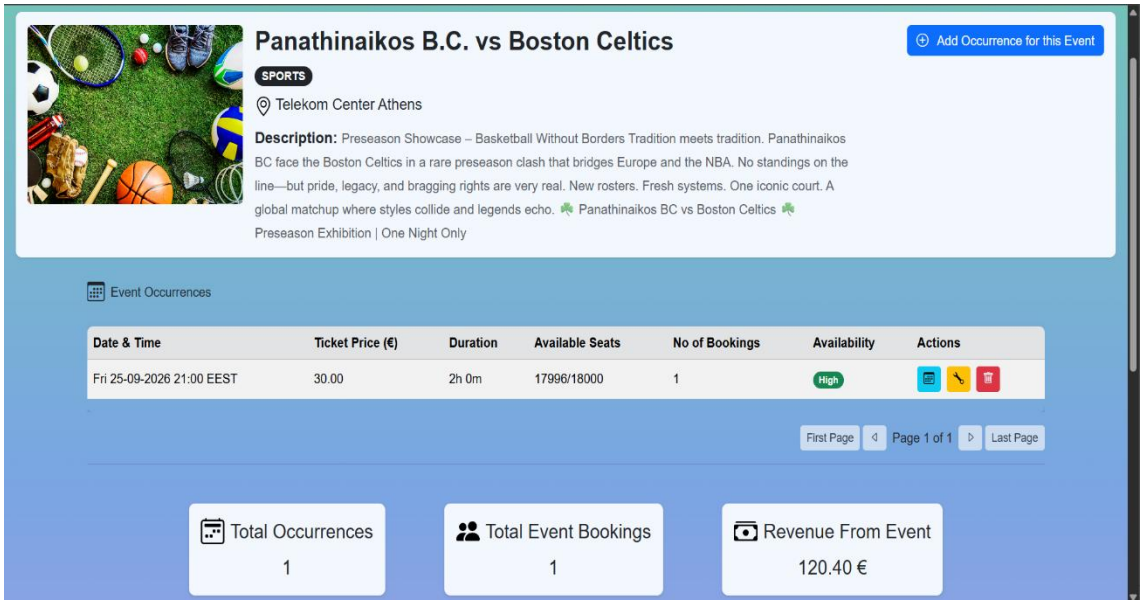


Figure 75. View all occurrences for a selected event

Additionally, admins can create a new event for the current venue.

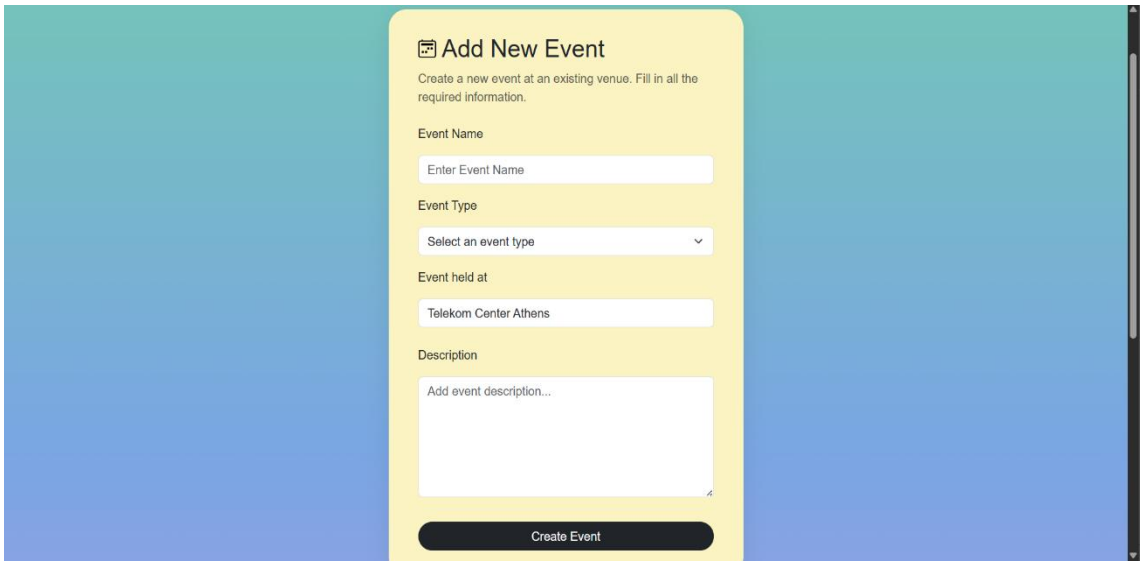


Figure 76. Add a new event for a selected venue form

5.2.5 Manage Events Section

In the Manage Events Section, all the stored events are displayed. The information displayed is the event name, event type, the location of the event (venue name) and count of scheduled occurrences. Admins can filter events by event name or event type.

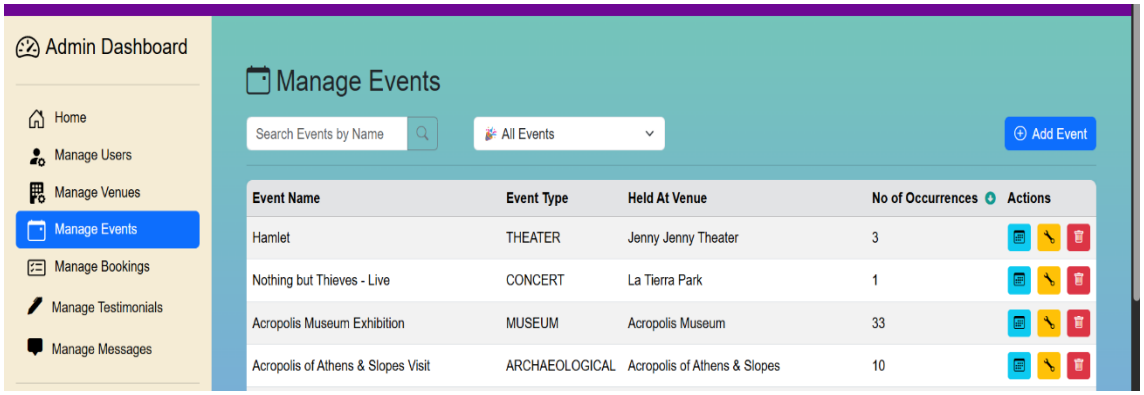


Figure 77. Manage events section

The available operations are:

- Add a new event for any of the listed venues
- Delete an existing event. This operation is only allowed for events that don't have any occurrences scheduled.
- Edit an existing event.

- View all the occurrences for a selected event.

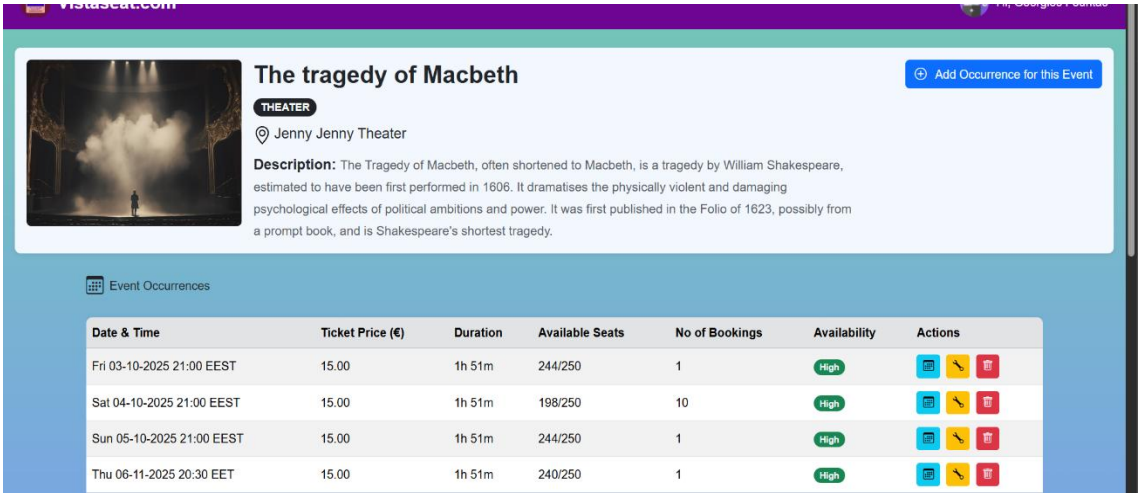


Figure 78. View occurrences of a selected event

- Add a new occurrence for the specified event.

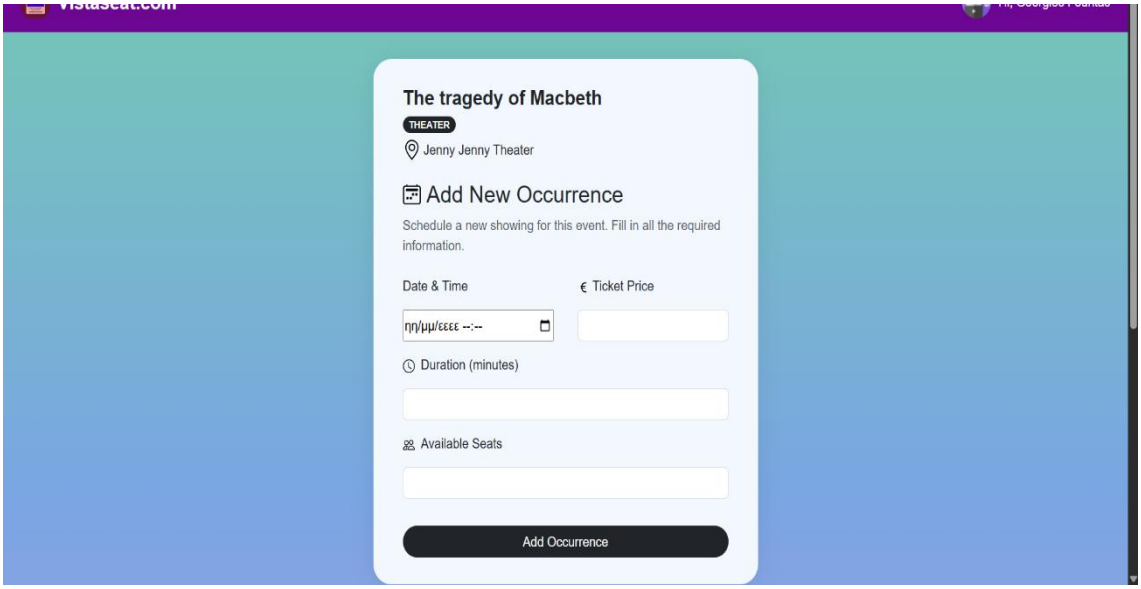


Figure 79. Add a new occurrence for a selected event

- Display all bookings for a selected occurrence of a specific event. The information available to administrators includes the booking ID, event name, venue name, occurrence datetime, customer details (name and email), confirmation timestamp, number of tickets, total cost, and booking status. Administrators can also filter bookings by booking ID or by the customer's last name.

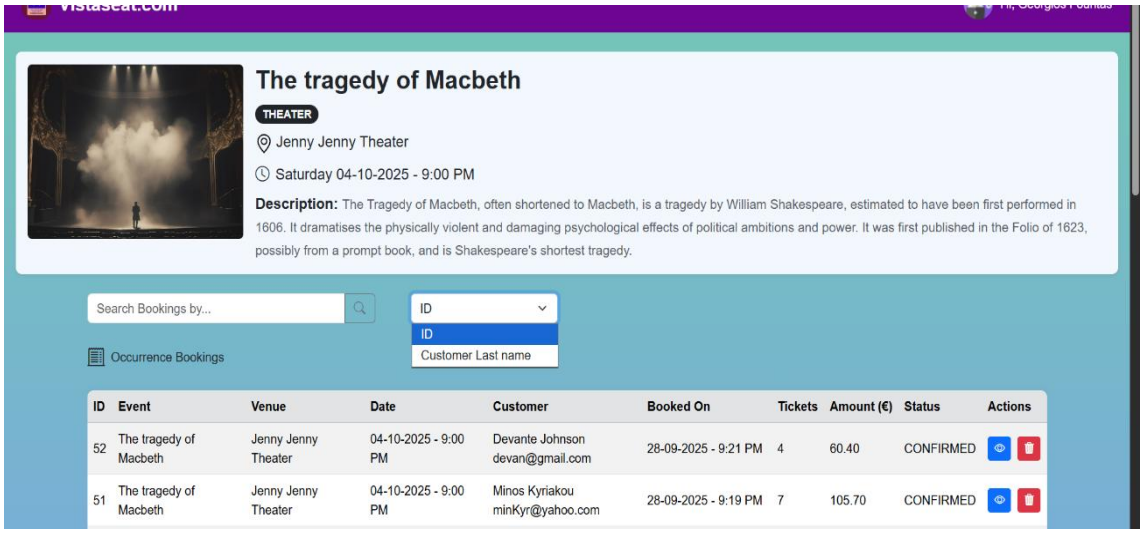


Figure 80. Display all bookings for a selected occurrence

At the bottom of the page, just above the footer, additional metrics are displayed through summary cards, including the total number of bookings, the number of confirmed tickets, and the total revenue for the selected occurrence.

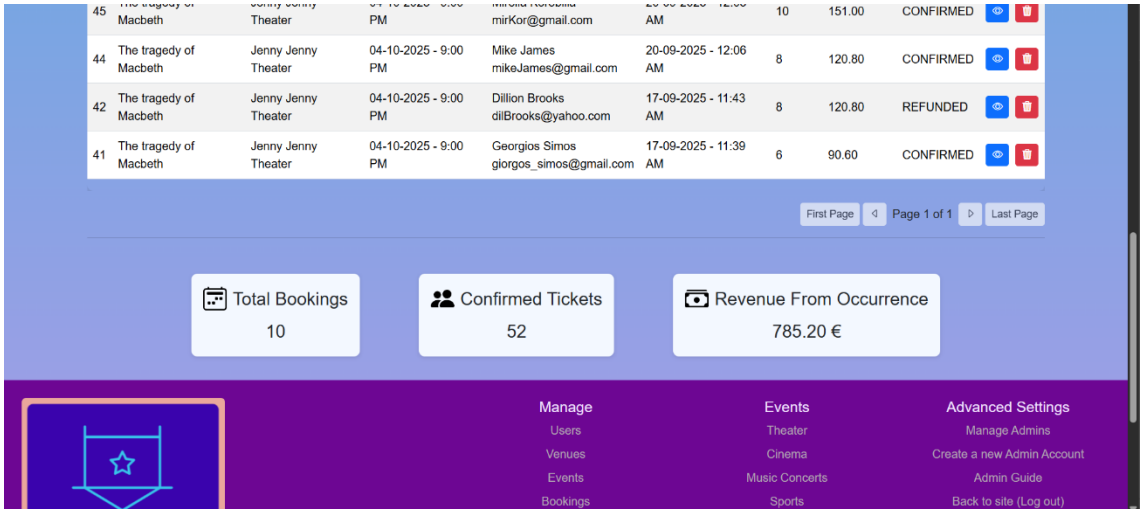


Figure 81. Summary cards for occurrence bookings

5.2.6 Manage Bookings Section

In the Manage Bookings Section, all the bookings are displayed. The information available to administrators includes the booking ID, event name, venue name, occurrence datetime, customer details (name and email), confirmation timestamp, number of tickets, total cost, and booking

status. Admins can also filter bookings by booking ID, event name, customer name, venue name or an occurrence date range (e.g. 15 to 22 Oct).

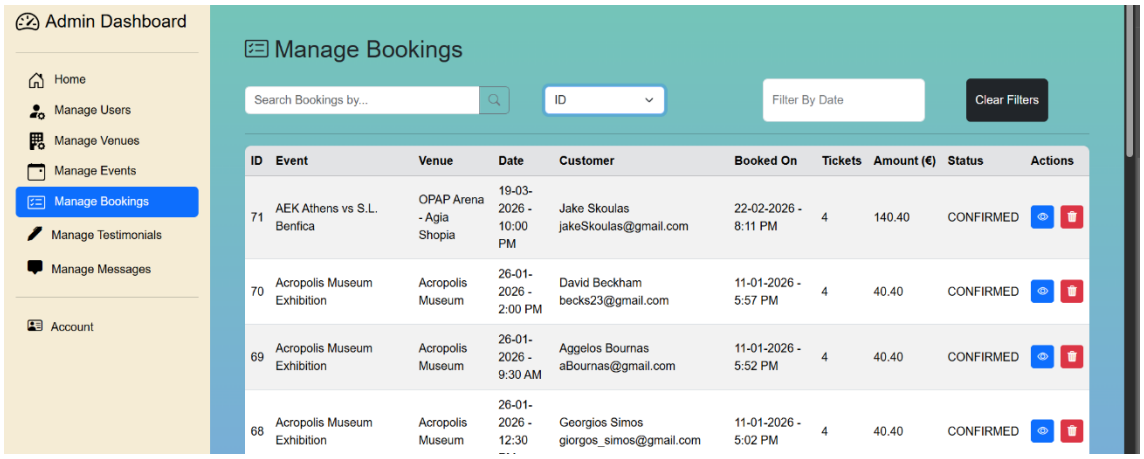


Figure 82. Manage bookings section

The available operations here are:

- Delete an existing booking, provided that it is not associated with existing tickets or payments.
- Display booking details.

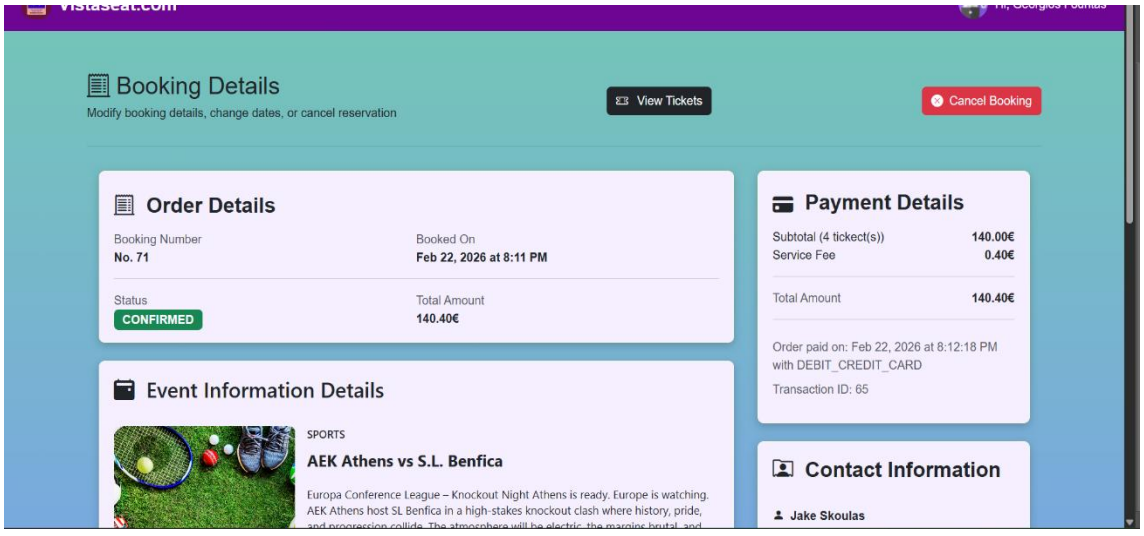


Figure 83. View booking details

This section provides additional functionality, including cancellation of the current booking.

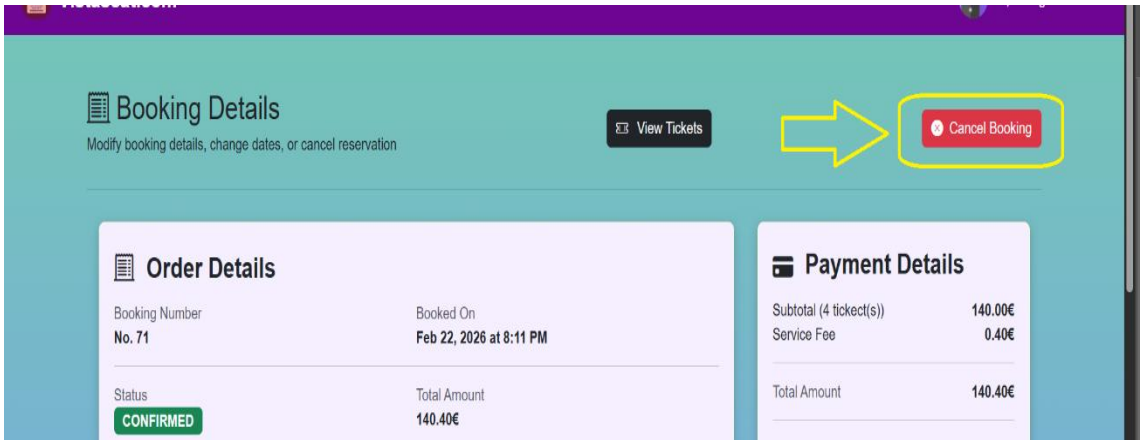


Figure 84. Cancel booking option

Upon successful cancellation, the full amount charged at the time of booking confirmation is refunded to the client.

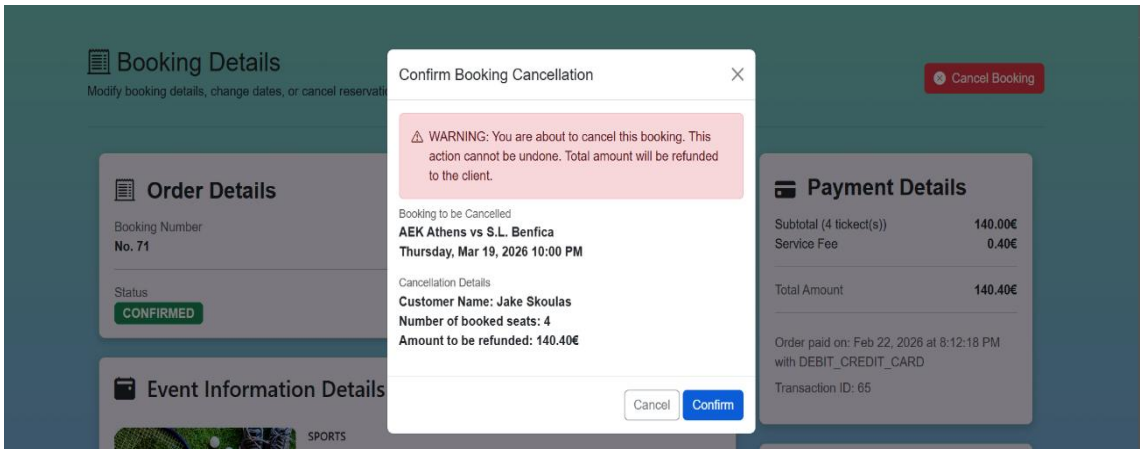


Figure 85. Confirm booking cancellation dialog

Additionally, there is an option to display the issued tickets.

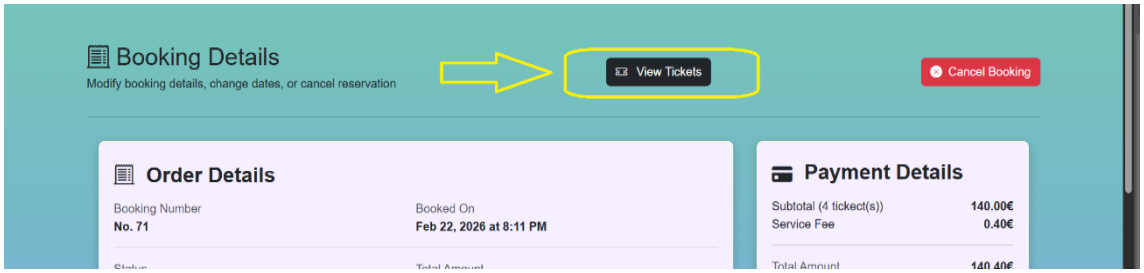


Figure 86. View tickets option

Admins can opt to download them in pdf format or forward them via email.

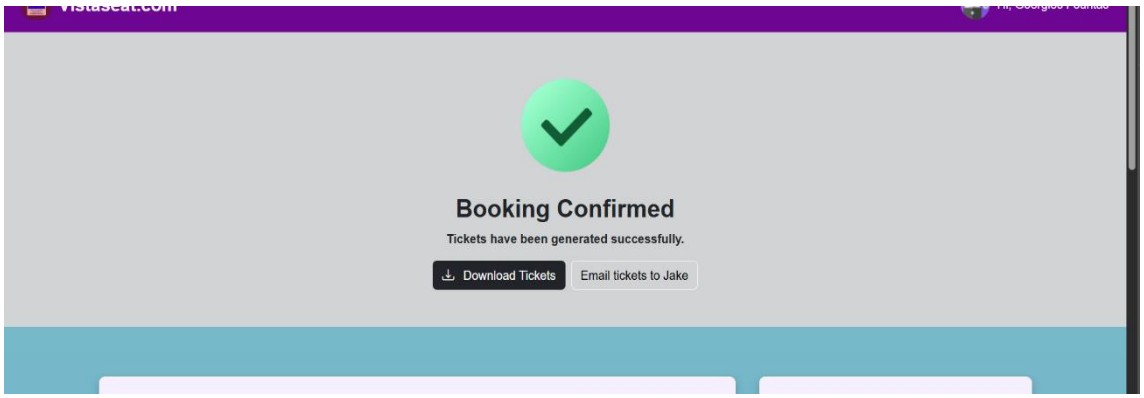


Figure 87. Forward tickets options

Or download each ticket in PDF format separately.

Figure 88. Download each ticket in PDF format

Furthermore, admins can reschedule an existing booking to a different date, provided that an alternative occurrence of the event is available and there are remaining seats.

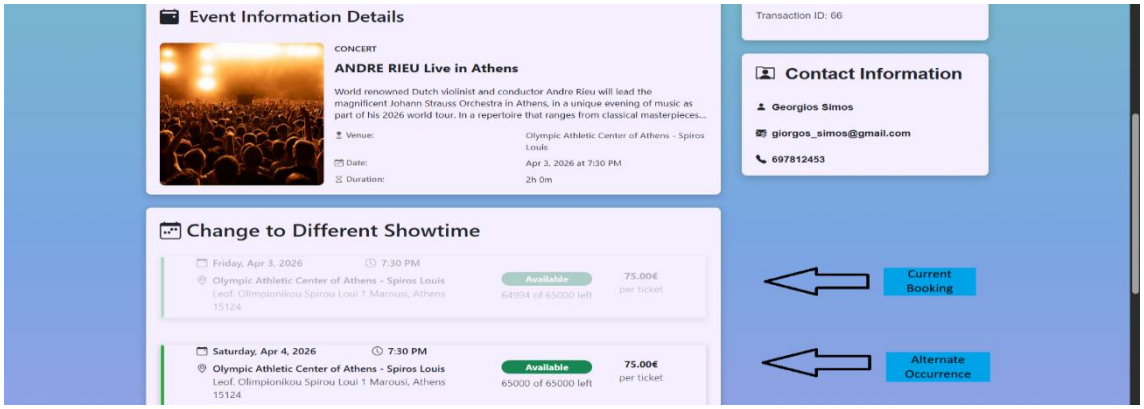


Figure 89. Booking reschedule

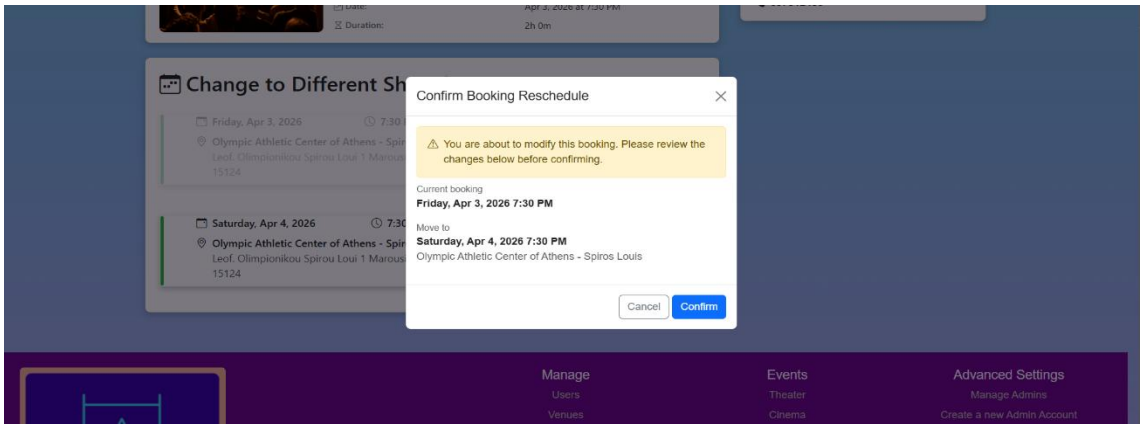


Figure 90. Confirm booking reschedule dialog

5.2.7 Manage Testimonials Section

In the Manage Testimonials section, all submitted testimonials are displayed. The information available to administrators includes the testimonial ID, rating (0–5 stars), review content, author, submission timestamp, and visibility status (hidden or visible). The available operations are:

- Delete a testimonial
- Toggle visibility status.

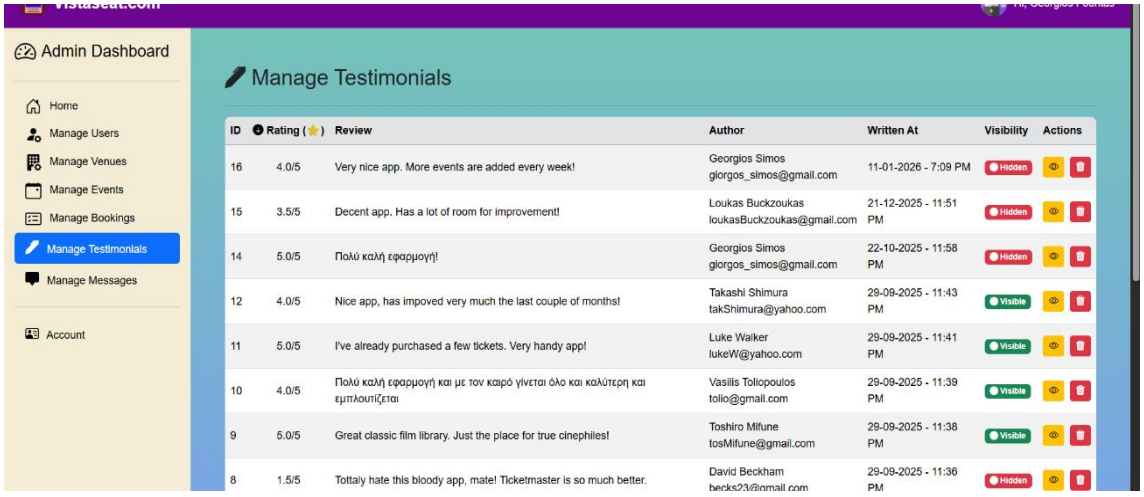


Figure 91. Manage testimonials section

By default, all testimonials are marked as hidden. Administrators are responsible for reviewing each testimonial to ensure compliance with the Terms of Use and determining whether it will be displayed on the home page.

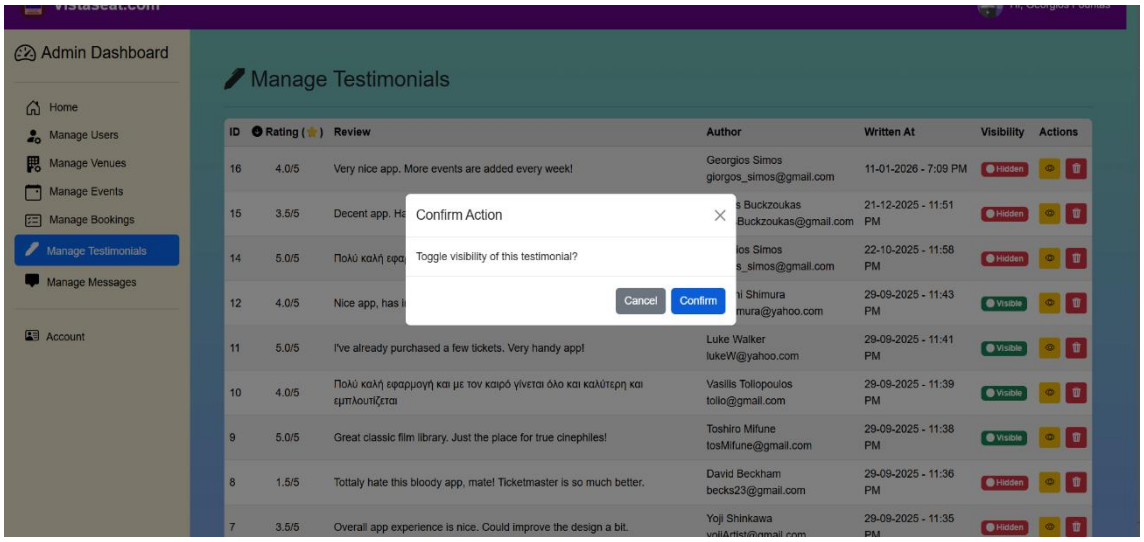


Figure 92. Toggle testimonial visibility option

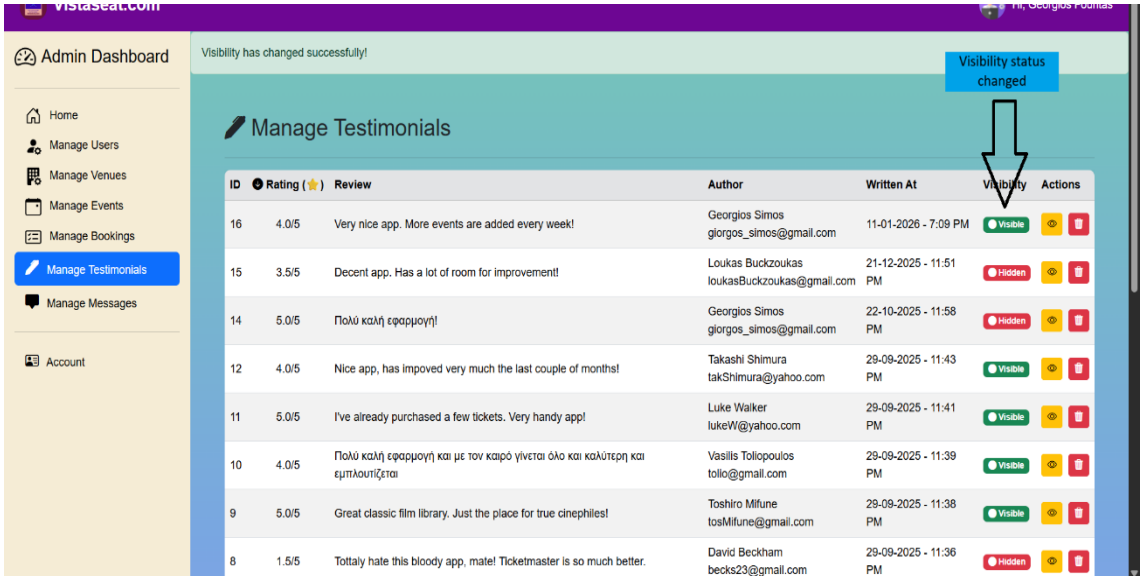


Figure 93. Testimonial visibility changed successfully

5.2.8 Manage Contact Messages Section

In the Manage Contact Messages section, all issues submitted through User Mode are displayed, regardless of whether the sender is a registered user. The information available to administrators includes the message ID, subject, message content, issue category, author, submission timestamp, status (resolved or pending), and any admin notes. Administrators can also filter messages by author name.

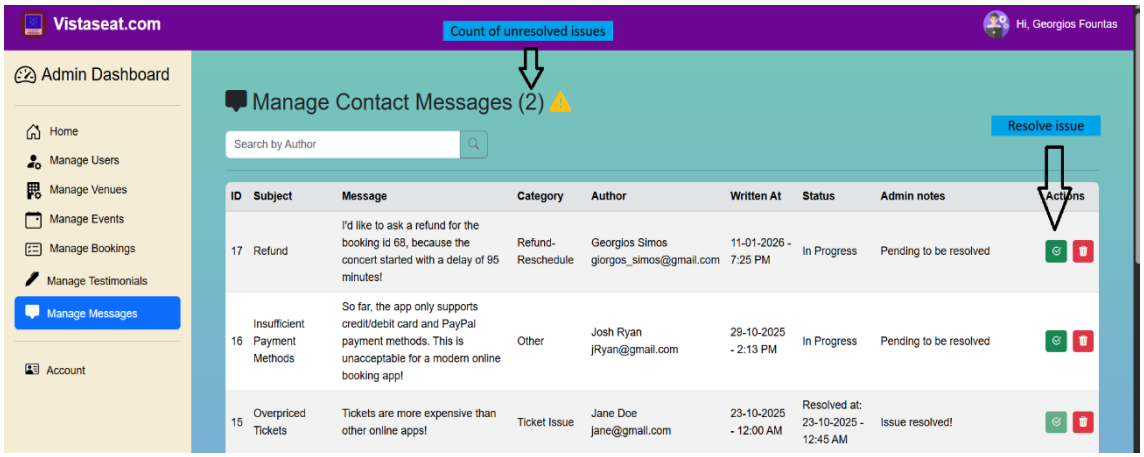


Figure 94. Manage contact messages section

Administrators can provide explanatory notes to address how they resolved an occurring issue.

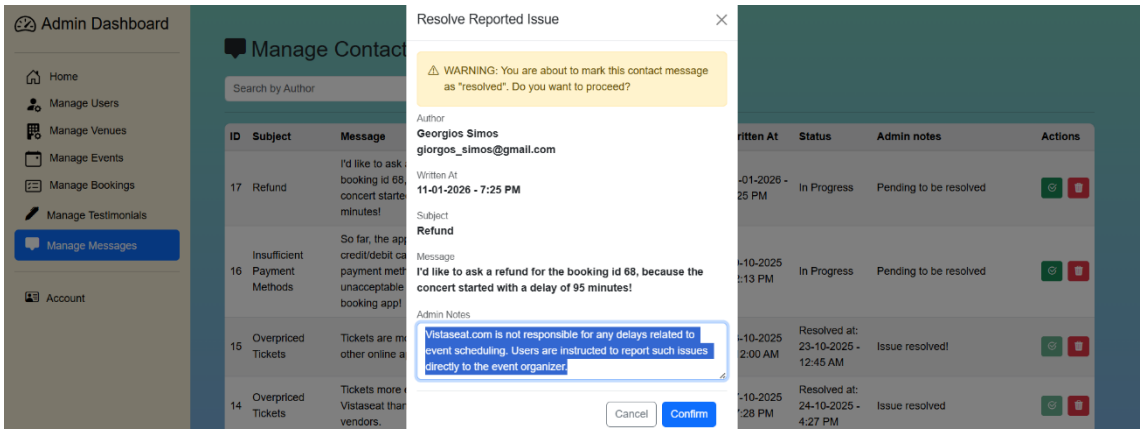


Figure 95. Resolve reported issue box

Upon successful resolution, the count of unresolved issues is updated, and the administrative notes section is populated with details describing how the issue was resolved.

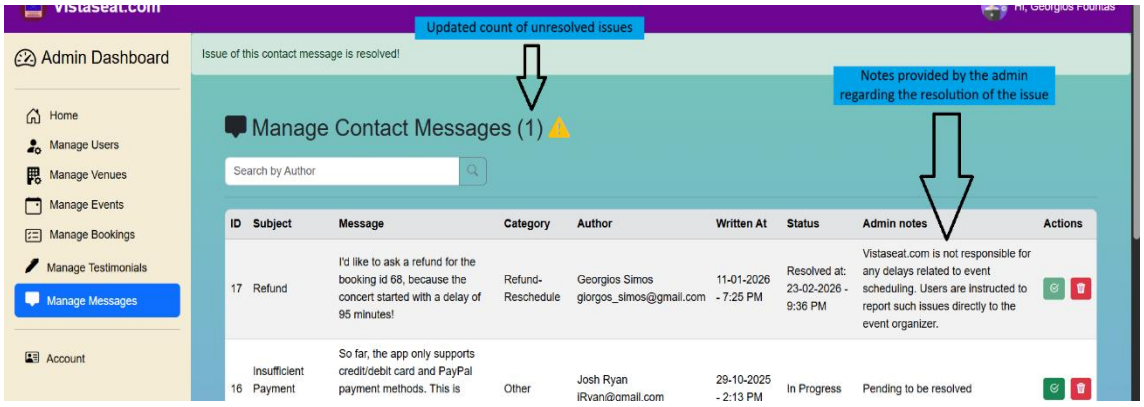


Figure 96. Reported issue resolved successfully

Messages that have not yet been resolved cannot be deleted from the system.

5.2.9 Admin Guide Section

The page footer includes an Admin Guide option.

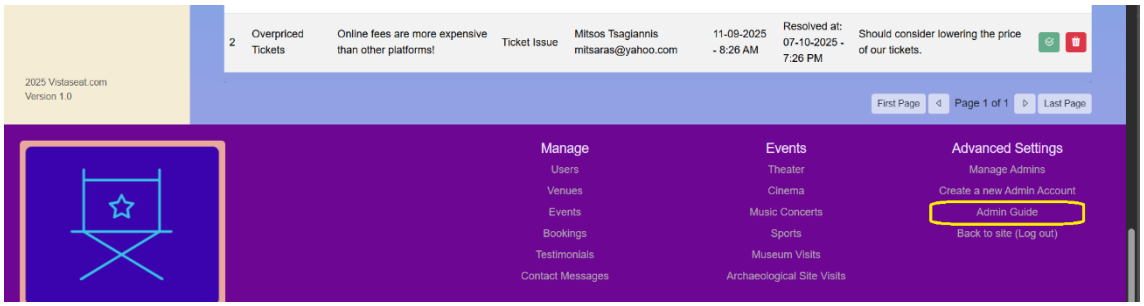


Figure 97. Admin Guide option

This section provides an overview of the system's available administrative features.

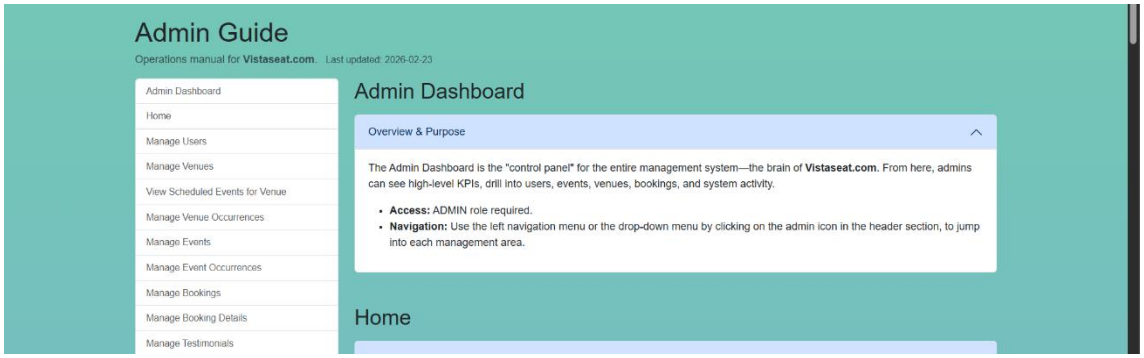


Figure 98. Admin Guide section

It is intended to assist administrators who are new to the platform and not yet familiar with its operational logic.

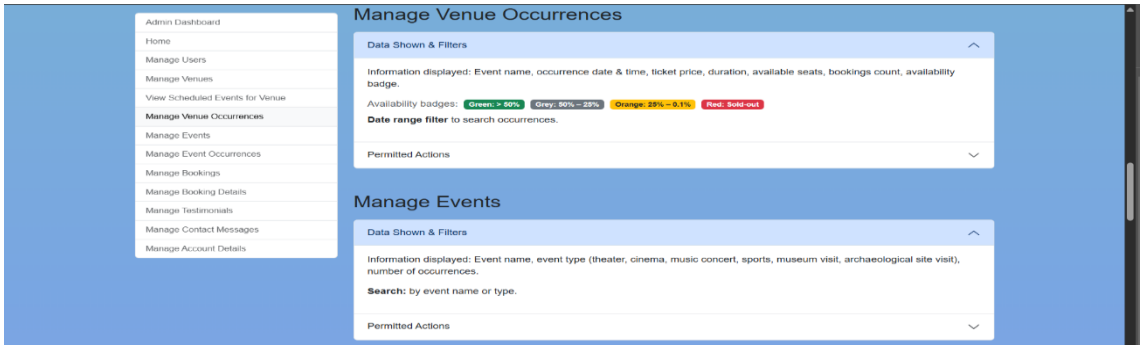


Figure 99. Overview about the various admin operations

5.2.10 Create New Administrator Account

The system provides functionality that allows a currently authenticated administrator to create new administrator accounts. This feature aims to enable controlled expansion of system management capabilities.

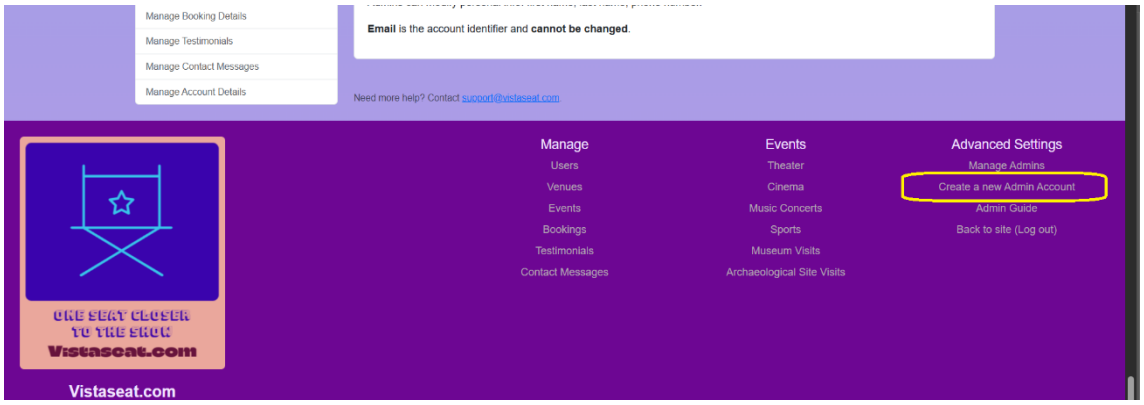


Figure 100. Create a new admin account option

When creating a new administrator, the required account details must be provided, and the new account is assigned the corresponding administrative privileges upon successful registration. This ensures that administrative access remains monitored and centrally managed.

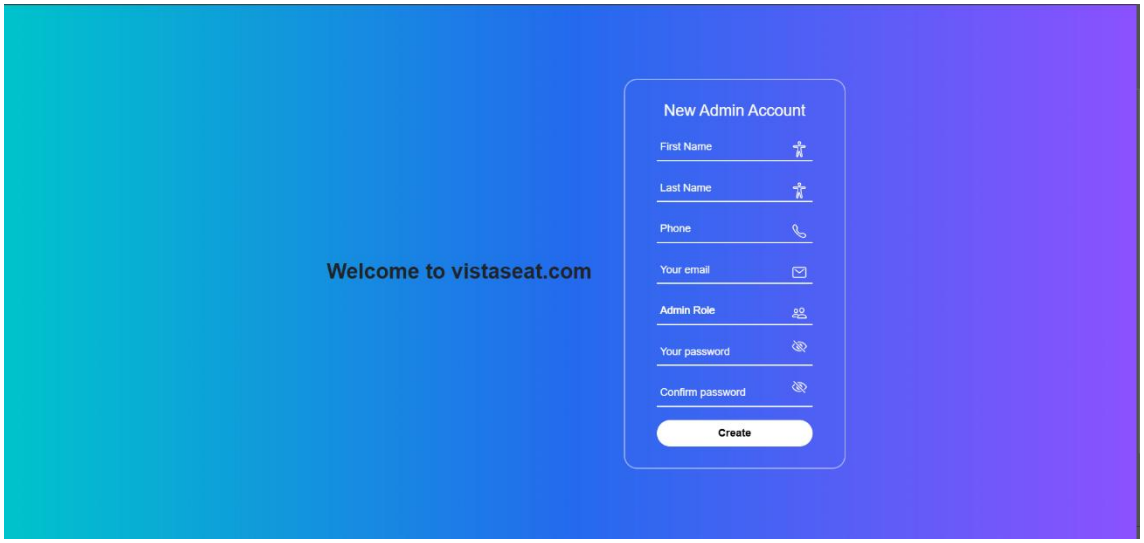


Figure 101. Create a new admin account form

6. Conclusions – Future Steps

The development of *Vistaseat.com* served as a comprehensive exploration into the complexities of modern software engineering within the domain of event management and reservation systems. By utilizing the Java Spring Boot ecosystem with a layered monolithic architecture, this effort addressed the intricate challenge of synchronizing diverse entities, ranging from ticket inventory to secure payment processing, into a smooth user experience.

6.1 Achievements

The primary objective of this thesis was the design and implementation of a robust, full-stack solution capable of addressing real-world scenarios in digital ticketing. The following milestones represent the core achievements of the project:

- **Architectural Design**: Successfully implemented a layered architecture following the Spring Model-View-Controller (MVC) pattern. By integrating principles inspired by Service-Oriented Architecture (SOA), the system ensures a clean separation of concerns, providing easier maintenance and scalability.
- **Security Implementation**: Utilized Spring Security to implement a structured defense-in-depth strategy. This included:
    - **Role-Based Access Control (RBAC):** Precise pattern matching to restrict sensitive URL patterns and API endpoints based on authenticated user roles.
    - **Dual Filter Chains:** Separate security configurations for administrative versus standard user/guest traffic.
    - **Session Orchestration:** Implementation of stateful, form-based login sessions with a strictly enforced 30-minute inactivity timeout to mitigate unauthorized access risks.
    - **Cryptographic Integrity**: Utilization of the BCrypt hashing algorithm for non-reversible, salted password storage.
- **Database Entity Orchestration:** Managed the multiple dependencies between users, venues, events, event occurrences, bookings, payments, tickets testimonials and contact messages. Navigating these relationships proved to be a significant technical challenge, particularly in ensuring data integrity across the entire transaction lifecycle.
- **Secure Payment Implementation:** Integrated the PayPal REST API (Sandbox Environment) to facilitate secure, simulated financial transactions. This implementation demonstrates the system's ability to handle external redirect flows and transaction status verification, providing a high degree of user convenience while maintaining PCI-DSS-compliant logic by offloading sensitive data handling to a trusted provider.
- **Unified Digital Ecosystem:** *Vistaseat.com* provides a centralized platform for a wide variety of events. This integration addresses the modern need for accessibility and diversity for both organizers and clients.

- **Hands-On Implementation of Academic Knowledge:** The project successfully bridged the gap between academic theory and professional practice by tackling real-world requirements such as secure authentication, persistent storage, and dynamic content delivery.

6.2 Enhanced Functionality and Improvements

While the current version of the application fulfills its primary functional requirements, it serves as a foundation for further expansion. To transition *Vistaseat.com* into a production-ready, high-availability system, the following enhancements could be introduced:

- **Hierarchical Venue Management:** Refactoring the data model to support multiple auditoriums per venue, allowing for more complex scheduling and resource allocation.
- **Dynamic UI/UX Improvements:** Implementing an interactive seat selection option. This would utilize a frontend visualization of auditorium floor plans, allowing users to select specific numbers for their seat rather than just seat categories.
- **Multi-Level Administrative Roles:** Expanding the authorization framework to include dedicated Event and Venue Administrator dashboards, enabling third-party organizers to manage their own listings independently.
- **Automated Communication Workflows:** Integrating an SMTP-based notification service to provide automated email confirmations and digital ticket delivery upon successful transaction completion.
- **Data Intelligence:** Developing an Analytics Dashboard for executives. By leveraging the existing database, the system could provide insights into booking trends, peak booking dates, and event performance metrics.

7. Bibliography

[1] Αλέπης, Ε., & Παναγιωτόπουλος,Ι-Χ.- Αντικειμενοστραφείς γλώσσες προγραμματισμού Java, Εκδόσεις Βαρβαρήγου, 2019

[2] Efthymios Alepis, Maria Virvou – Object oriented architecture for affective multimodal e-learning interfaces, 2010

[3] Tiobe Index – Programming Languages popularity indicator (https://www.tiobe.com/tiobe-index/)

[4] Why Java is still a good choice for enterprise development (https://scand.com/company/blog/java-for-enterprise-software-development/), Victor Krapivin, May 2025

[5] The IoC Container - Introduction to the Spring IoC Container and Beans (https://docs.spring.io/spring-framework/reference/core/beans/introduction.html)

[6] A Comparison between Spring and Spring Boot (https://www.baeldung.com/spring-vs-spring-boot), May 2024

[7] Efthymios Alepis, Maria Virvou – Multimodal object oriented user interfaces in mobile affective interaction, Feb 2011

[8] Efthymios Alepis, Maria Virvou – Emotional Intelligence in Multimodal Object Oriented User Interfaces, 2009

[9] Efthymios Alepis, Vasiliki Matzavela – A survey for the evolution of adaptive learning in mobile and electronic devices, 2017

[10] Jose A. Moinhos Cordeiro, Maria Virvou, Boris Shishkov – Software and Data Technologies, 5th International Conference ICSOFT, July 2010

[11] Kalliopi Tourtoglou, Maria Virvou – User Modelling in a Collaborative Learning Environment for UML, 2008

[12] Use Case Diagrams (https://www.ibm.com/docs/en/rational-soft-arch/10.0?topic=diagrams-use-case), May 2025

[13] Introduction to Spring Web MVC Framework (https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html)

[14] Khalil Elbaz - Measuring Maintainability of Web Applications Using an Extensible MVC Architecture, September 2022

[15] What is Service Oriented Architecture (SOA) (https://www.ibm.com/think/topics/soa)

[16] Thymeleaf Template Engine (https://www.thymeleaf.org/), December 2024

[17] Get started with Bootstrap (https://getbootstrap.com/docs/5.3/getting-started/introduction/)

[18] Eugen Paraschiv, Introduction to Spring Data JPA (https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa), November 2025

[19] Hibernate ORM Guide (https://hibernate.org/orm/documentation/7.2/)

[20] PostgreSQL Documentation (https://www.postgresql.org/docs/current/)

[21] How Spring Security Works: Architecture, Authentication & Best Practices (https://www.penligent.ai/hackinglabs/how-spring-security-works-architecture-authentication-best-practices/), December 2025

[22] Constantinos Patsakis, Efthymios Alepis - I know what you streamed last night: On the security and privacy of streaming, 2018

[23] Authorization Architecture in Spring Security (https://docs.spring.io/spring-security/reference/servlet/authorization/architecture.html)

[24] Eugenia Politou, Alexandra K. Michota, Efthymios Alepis, Matthias Pocs, Constantinos Patsakis – Backups and the right to be forgotten in the GDPR: An uneasy relationship, 2018

[25] Stateful vs Stateless Security: Understanding the Two Models that Power Modern Applications (https://medium.com/@ayoubtaouam/stateful-vs-stateless-security-understanding-the-two-models-that-power-modern-applications-9b547566974d), Ayoub Taouam, January 2026

[26] Tharushka Heshan, Password hashing using BCrypt in Spring Security (https://tharushkaheshan.medium.com/password-hashing-using-bcrypt-in-spring-security-part-8-d867a83b8695), November 2024

[27] Eugenia Politou, Efthymios Alepis, Constantinos Patsakis – Forgetting personal data and revoking consent under the GDPR: Challenges and proposed solutions, March 2018

[28] Eugenia Politou, Efthymios Alepis, Maria Virvou, Constantinos Patsakis - Privacy and Data Protection Challenges in the Distributed Era, 2022

[29] Ramez Elmarsi, Shamkant B. Navathe – Fundamentals of Database Systems, 7th Edition, Pearson 2016

[30] UML Class and Object Diagrams Overview (https://www.uml-diagrams.org/class-diagrams-overview.html)