Event Organizer

Project documentation

Student(s):

* Calinescu Mirela Petra

**Group: 30444**

**Contents**

[I Project specification 4](#_Toc163031808)

[1.1 Domain Model Diagram 5](#_Toc163031809)

[II Use-Case model 5](#_Toc163031810)

[2.1 Users and stakeholders 5](#_Toc163031811)

[2.2 Use-Case identification 6](#_Toc163031812)

[2.3 UML Use-Case diagram 10](#_Toc163031813)

[III Architectural design 11](#_Toc163031814)

[3.1 Conceptual architecture 11](#_Toc163031815)

[3.2 Package diagram 12](#_Toc163031816)

[3.3 Class diagram 12](#_Toc163031817)

[3.4 Database (E-R/Data model) diagram 13](#_Toc163031818)

[13](#_Toc163031819)

[3.5 Sequence diagram 13](#_Toc163031820)

[3.6 Activity diagram 13](#_Toc163031821)

[IV Supplementary specifications 14](#_Toc163031822)

[4.1 Non-functional requirements 14](#_Toc163031823)

[4.2 Design constraints 15](#_Toc163031824)

[V Testing 16](#_Toc163031825)

[5.1 Testing methods/frameworks 16](#_Toc163031826)

[5.2 Future improvements 17](#_Toc163031827)

[VI Bibliography 17](#_Toc163031828)

# I Project specification

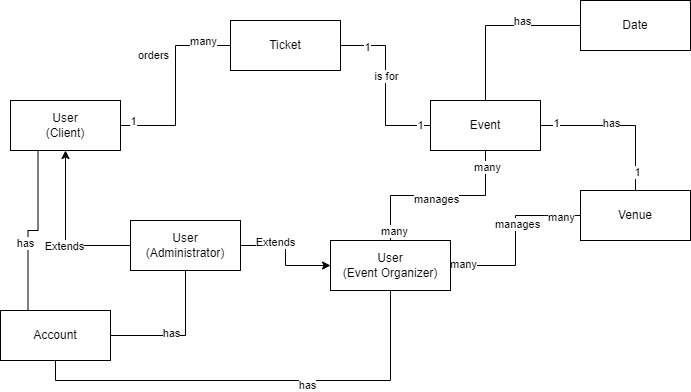
The project’s aim is to develop a client-server application for an event organizing firm that allows ticket booking for clients. The application will be web-based for convenience.

There will be three types of users: clients, event coordinators and administrators. Clients have only read access to the information while the other two can perform more CRUD operations.

The final product should allow the following actions:

* Sign in and sign up for all users.
* Viewing events and booking tickets for clients.
* CRUD operations on the events and various charts for organizers and admins.
* CRUD operations on users for administrators.

## 1.1 Domain Model Diagram



# II Use-Case model

The use-case model for the application offers an overview on the functionalities of the project and each role’s responsibilities.

## 2.1 Users and stakeholders

* Users

1. Clients

* Role: Attendees or potential attendees of events who are interested in viewing and booking tickets.
* Access Level: Primarily read-only access to event information. Their main interactions involve browsing events and ticket purchases.

1. Organizers

* Role: Individuals or teams responsible for creating, organizing, and managing events and venues listed within the application.
* Access Level: CRUD (Create, Read, Update, Delete) capabilities concerning events and venues.

1. Admins

* Role: High-level users tasked with overseeing the application's overall functionality, user management, and system integrity.
* Access Level: Full CRUD operations on all aspects of the application, including users, events and venues.
* Stakeholders

1. Project Manager

* Orchestrates the development and deployment of the application, ensuring milestones are met, and resources are allocated efficiently.

1. Development Team

* The individuals directly involved in building the application.

1. Investors

* Those who have invested in the firm or own it and are looking for a return on their investment through the success of the application.

1. Clients

* While also users of the application, clients are considered stakeholders due to their direct impact on the application's success through their engagement and satisfaction.

## 2.2 Use-Case identification

**Use Case 1: Login**

* **Use Case Name:** User Login
* **Level:** User-Goal
* **Main Actor:** Client/Event Coordinator/Admin
* **Main Success Scenario:** The user navigates to the login page, enters their credentials (username and password), and submits the information. The system verifies the credentials and grants access to the user's respective dashboard or home page.
* **Extension:**
  + User enters incorrect credentials: The system displays an error message and prompts the user to try again.

**Use Case 2: Create Account**

* **Use Case Name:** Create Account
* **Level:** User-Goal
* **Main Actor:** Client/Event Coordinator
* **Main Success Scenario:** The user selects the "Sign Up" option, fills out the registration form with necessary details (such as email, username, password), and submits it. The system creates the account and sends a confirmation email to the user.
* **Extension:**
  + User enters an already taken username/email: The system alerts the user and asks for a different username/email.
  + User enters invalid information: The system displays validation errors and requests correct information.

**Use Case 3: Edit Own Credentials**

* **Use Case Name:** Edit Credentials
* **Level:** User-Goal
* **Main Actor:** Client
* **Main Success Scenario:** The client accesses their profile settings, updates their credentials (such as password, email, phone number), and saves the changes. The system updates the account details accordingly.
* **Extension:**
  + Client enters invalid information: The system displays an error and requests valid information.

**Use Case 4: Add New Event**

* **Use Case Name:** Add Event
* **Level:** Subfunction
* **Main Actor:** Event Coordinator/Admin
* **Main Success Scenario:** The event coordinator or admin navigates to the event management section, selects "Add New Event", fills in the event details (name, date, venue, etc.), and submits the form. The system creates the event and makes it visible to clients.
* **Extension:**
  + Invalid event details: The system shows an error message and asks for correct information.
  + Venue capacity exceeded or not available: The system warns the coordinator and suggests adjusting the capacity or changing the venue.

**Use Case 5: Add New Venue**

* **Use Case Name:** Add Venue
* **Level:** Subfunction
* **Main Actor:** Event Coordinator/Admin
* **Main Success Scenario:** The event coordinator or admin accesses the venue management section, selects "Add New Venue", inputs the venue details (name, location, capacity), and submits the form. The system adds the venue to the list of available venues for events.
* **Extension:**
  + Venue already exists: The system alerts the administrator and prevents duplicate entries.
  + Invalid venue details: The system requests correct information.

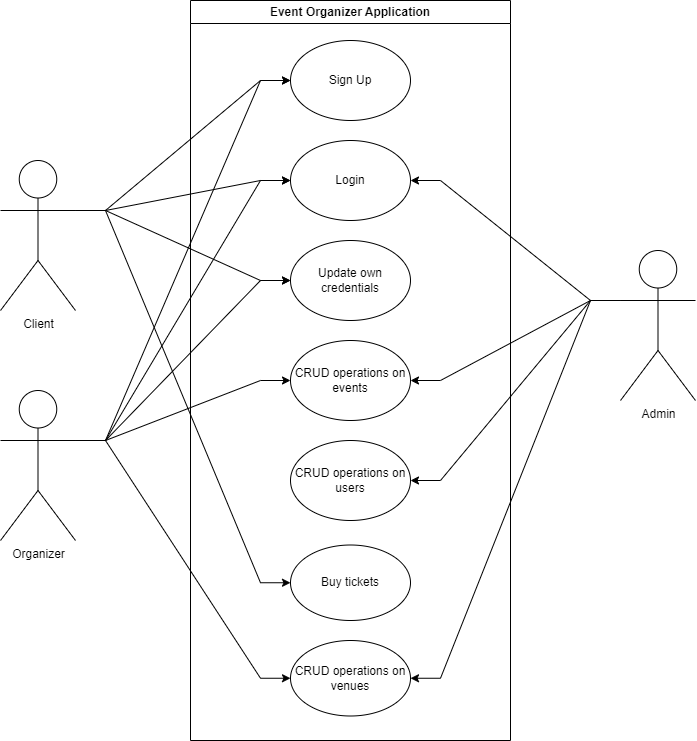
**Use Case 6: Delete Events**

* **Use Case Name:** Delete Event
* **Level:** Subfunction
* **Main Actor:** Event Coordinator/Admin
* **Main Success Scenario:** The coordinator or admin selects an event from the event management section and chooses the "Delete" option. The system asks for confirmation, and upon agreement, it removes the event from the system.
* **Extension:**
  + Event has active bookings: The system either prevents deletion or requires the coordinator to manage refunds or notifications to clients.

**Use Case 7: Buy Tickets**

* **Use Case Name:** Purchase Tickets
* **Level:** User-Goal
* **Main Actor:** Client
* **Main Success Scenario:** The client selects an event, chooses the number of tickets, enters payment information, and confirms the purchase. The system processes the payment, reserves the tickets, and sends a confirmation email to the client.
* **Extension:**
  + Event sold out: The system informs the client that no tickets are available and may offer to add the event to a wishlist.

## 2.3 UML Use-Case diagram



# III Architectural design

Here we outline the three-tier architecture, encompassing the presentation, service, and persistence layers, and discuss the integration of technologies such as React, Spring, and PostgreSQL. This design ensures scalability, security, and maintainability, catering to the diverse needs of clients, event coordinators, and administrators.

## 3.1 Conceptual architecture

The envisioned application is a web-based client-server system designed to streamline the process of event management and ticket booking. The conceptual architecture of this application is structured around a layered approach, consisting of persistence, service, and presentation layers. This multi-layered architecture is instrumental in separating the application's responsibilities, thereby promoting a clean separation of concerns which is essential for maintaining a scalable, maintainable, and secure system.

At the core of the application lies the persistence layer, which is responsible for data storage and management. This layer will leverage the features of PostgreSQL to handle complex queries, transactions, and data integrity. The Java Persistence API (JPA) repository will be used for data access, providing an abstraction layer over the database and facilitating a seamless interaction with the database without the need for boilerplate code.

The service layer sits above the persistence layer and contains the business logic of the application. It acts as a bridge between the presentation layer and the persistence layer, ensuring that user requests are processed and responded to appropriately. This layer will be implemented using the Spring Framework, capitalizing on its extensive suite of features for enterprise application development, including security, transaction management, and dependency injection. This layer will handle authentication, authorization, and the execution of CRUD operations on events and users as per the roles and permissions defined for clients, event coordinators, and administrators.

The presentation layer is the user-facing part of the application, designed to provide an intuitive and responsive user interface for interacting with the application's features. Backend-wise it consists of various controllers. The choice of React for the frontend development facilitates the creation of a dynamic and engaging user experience, with real-time updates and interactive UI components. This layer will present information in a clear and accessible format, allowing clients to view events and book tickets, and enabling coordinators and administrators to perform their respective operations with ease.

The chosen layered architecture aligns with the application's requirements by promoting separation of concerns, which enhances maintainability and scalability. Each layer can be developed and tested independently, facilitating parallel development and simplifying updates and upgrades. The use of Spring with JPA for the backend ensures a solid and well-supported foundation for business logic and data management, while React's component-based architecture makes the frontend flexible and efficient. Together, these technologies provide a powerful and cohesive framework for developing a feature-rich, user-friendly event organizing and ticket booking application.

## 3.2 Package diagram

*< (Package Diagram)/>*

## 3.3 Class diagram

*< (Class Diagram)/>*

## 3.4 Database (E-R/Data model) diagram

## *A screenshot of a computer Description automatically generated*

## 3.5 Sequence diagram

*< (Sequence Diagram)/>*

## 3.6 Activity diagram

*< (Activity Diagram)/>*

# IV Supplementary specifications

*< Se va scrie o mica introducere./>*

## 4.1 Non-functional requirements

1. **Availability**: This NFR ensures that the application is accessible to users whenever needed, minimizing downtime and ensuring reliability. For an event organizing application, high availability is critical, especially during peak times like major event announcements or ticket releases, to handle high user traffic without service disruptions. Implementing redundant systems, failover mechanisms, and regular maintenance can achieve this, ensuring users can always access the service when needed.
2. **Security**: Given the application involves handling user data, including personal information and payment details, security is paramount. This includes ensuring data protection, secure authentication and authorization mechanisms, and protecting against common web vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). Implementing robust encryption for data transmission, secure storage practices, and regular security audits will help maintain the integrity and confidentiality of user data, fostering trust and compliance with data protection regulations.
3. **Scalability**: The ability to handle growth in users and data without compromising performance is essential for the application, especially as the number of events and users increases. Scalability can be addressed both vertically (by adding more resources to existing servers) and horizontally (by adding more servers). Using cloud services that offer auto-scaling capabilities and designing the system with microservices architecture can enhance scalability, allowing the application to efficiently manage varying loads.
4. **Performance**: The application should be responsive and capable of processing user requests quickly, even under high load. This includes fast loading times for web pages, quick response to user actions, and efficient processing of transactions and data queries. Performance optimizations might involve efficient database indexing, caching frequently accessed data, and minimizing the use of resource-intensive operations. Ensuring a smooth and fast user experience is crucial for user satisfaction and retention.
5. **Accessibility across devices**: With users accessing the application from various devices, including desktops, tablets, and smartphones, the application must be designed to be responsive and functional across all these platforms. This involves using responsive design principles in the frontend, ensuring compatibility with different browsers, and optimizing the user interface for touch and small screens on mobile devices. Ensuring accessibility across devices broadens the user base and ensures a consistent experience regardless of how users access the application.

## 4.2 Design constraints

**Languages and Frameworks**:

The backend of the application will be developed in Java, utilizing the Spring Framework, including Spring Boot for microservices architecture and Spring Security for authentication and authorization mechanisms. The choice of Java and Spring is due to their robustness, extensive community support, and the comprehensive ecosystem for building enterprise-level applications. The frontend will be developed using React, a JavaScript library for building user interfaces, chosen for its component-based architecture which facilitates the development of dynamic and responsive web applications.

**Security Measures**:

Security is a paramount concern, especially given the application's handling of user data and transactions. JWT (JSON Web Tokens) will be used for securing RESTful APIs, providing a stateless authentication mechanism that is well-suited for scalable web applications. JWT will be implemented in conjunction with Spring Security to ensure secure user authentication and authorization across the application. Additional security measures will include HTTPS for secure communication, input validation to prevent injection attacks, and regular security audits to identify and mitigate potential vulnerabilities.

**Technologies**:

The application will leverage various technologies to enhance performance, scalability, and user experience. This includes the use of Docker containers for deploying the services besides the aforementioned languages and frameworks. These technologies are chosen for their scalability, reliability, and widespread adoption in the industry.

**Database Specifications**:

PostgreSQL will be the database of choice due to its advanced features, reliability, and support for complex data types and transactions. The application will use JPA (Java Persistence API) repositories for database interactions, abstracting the complexity of direct database operations and ensuring a clean separation between the application logic and database layer.

**Device and Browser Compatibility**:

The frontend design will need to ensure compatibility across a wide range of devices and browsers, adhering to responsive design principles. This includes testing and optimization for major browsers (such as Chrome, Firefox, Safari, and Edge) and ensuring that the application's interface is usable and aesthetically pleasing on both desktop and mobile devices.

**Performance and Scalability Constraints**:

The application architecture must be designed to handle varying loads, particularly during high-demand periods such as major event launches or ticket sales. This includes implementing caching and load balancing.

# V Testing

*< Se va discuta la laborator./>*

## 5.1 Testing methods/frameworks

## 5.2 Future improvements

# VI Bibliography