Faculty of Engineering of the University of Porto


# Homework 07

TicketBoss

**M.EIC010 - Software Systems Architecture**

**1MEIC03 - T32**

## Professors

Ademar Aguiar

Neil Harrison

## Students

Ana Rita Baptista de Oliveira - up202004155@edu.fe.up.pt

Diogo Alexandre da Costa Melo Moreira da Fonte - up202004175@edu.fe.up.pt

João Paulo Moreira Araújo - up2020004293@edu.fe.up.pt

Tiago Nunes Moreira Branquinho - up202005567@edu.fe.up.pt

# Introduction

In this report, we delve into the landscape of ticket sales for entertainment events, where efficiency, reliability, and user experience are crucial. As such, TicketBoss emerges as a system designed to manage ticket sales for numerous entertainment events seamlessly. TicketBoss empowers users to navigate through event listings, select desired seats, and securely complete their purchases.

This report presents the architectural design of TicketBoss, encapsulating its logical, process, use-case, and physical views. Through detailed documentation and insightful analysis, we explain the underlying system's components, interactions, and functionalities. Our architectural decisions are guided by a comprehensive approach aimed at meeting key quality attribute requirements, ensuring scalability, performance, and robustness.

The architecture of TicketBoss embraces architectural patterns and principles to foster modularity, extensibility, and maintainability. By leveraging best practices learned in class, we strive to deliver a system that not only meets current demands but also accommodates future enhancements and advancements technology-wise.

Through this report, we aim to provide a comprehensive understanding of TicketBoss, offering insights into its design rationale, key functionalities, and how it effectively addresses the diverse needs of both users and event organizers.

# Architecture Design

The TicketBoss system is designed to provide a seamless experience for users looking to purchase tickets for various events. This architecture aims to ensure efficient ticket sales, seat selection, and transaction processing while maintaining system reliability and scalability.

The following are the assigned Views of TicketBoss's system.

## Logical View

For the Logical View of the TicketBoss system, we'll start by identifying the major components and their relationships to each other:

### Components

- **User Interface (UI)**:
    - Responsible for presenting entertainment events to users and facilitating interactions such as seat selection and ticket purchase.

    - Subcomponents:
        - **Event Listing Interface**: Displays available entertainment events.
        - **Authentication Interface**: Allows users to authenticate with an account to purchase tickets.
        - **Seat Selection Interface**: Allows users to select seats for a chosen event.
        - **Checkout Interface**: Allows users to terminate their purchase, including entering payment and billing information.
        - **Confirmation Interface**: Displays confirmation details after a successful purchase.  Also allows users to view past purchases.
- **Event Manager**:
    - Manages information related to entertainment events.
    - Subcomponents:
        - **Event Catalog**: Stores details about available events, such as event name, date, time, venue, and available seats.
        - **Seat Reservation Manager**: Handles seat availability, reservation, and expiration logic.
        - **Pricing Management**: Calculates prices based on seat selection and any applicable discounts or fees.
        - **Checkout Management**: Handles the user order and links to the payment gateway.
- **User Manager**:
    - Manages user-related operations such as authentication, user profile management, and communication.
    - Subcomponents:
        - **Authentication Service**: Handles user login and authentication.
        - **Profile Management**: Manages user profiles, including payment methods and preferences.
        - **Communication Service**: Sends confirmation emails to users after successful purchases.
- **Payment Gateway**:
    - Interfaces with external payment processing services to handle credit card

transactions securely.

- **Notification Manager**:
    - Handles the generation and delivery of notifications and e-mails to users for various events such as seat reservation expiration, purchase confirmation, event updates, and new events (if allowed).
    - Subcomponents:
        - **Notification Generator**: Creates notifications, e.g., based on the purchased event seats.
        - **Notification Delivery Gateway**: Sends notifications to users via email.
- **Event Database Component**:
    - Stores information regarding events, including event details, seat availability, and pricing. It serves as the primary data source for the Event Manager component.
- **User Database Component**:
    - Stores user information, including profiles, purchase history, and payment details. It facilitates the secure storage of sensitive information.

## Connectors

To ease cluttering the diagram, you can link the connections to the respective number. Taking this information in mind, these are the components:

1. **UI Authentication - Authentication Service:**
    - This connector enables the User Interface to communicate with the Authentication Service component, allowing users to authenticate their accounts before the ticket purchase process.
2. **Authentication Service - User Database:**
    - This connector facilitates communication between the Authentication Service and the User Database component, allowing the service to verify user credentials against stored user information securely.
3. **Profile Management - User Database:**
    - This connector allows the Profile Management component to interact with the User Database, enabling the management and retrieval of user profile information such as preferences and payment methods.

4. **Communication Service - Notification Generator:**
    - This connector links the Communication Service to the Notification Generator component, enabling the service to trigger notifications based on various events such as successful purchases or seat reservation expirations.
5. **Notification Generator - Notification Delivery Gateway:**
    - This connector connects the Notification Generator to the Notification Delivery Gateway, allowing generated notifications to be delivered to users via email or other preferred communication channels.
6. **Notification Generator - User Database:**
    - This connector enables the Notification Generator to access user information stored in the User Database, allowing personalized notifications to be created based on user preferences and purchase history.
7. **UI Event Listing - Event Catalog:**
    - This connector facilitates communication between the User Interface and the Event Catalog component, allowing retrieval of information about available events for display to users.
8. **Event Catalog - Event Database:**
    - This connector links the Event Catalog to the Event Database component, providing access to event-related data such as event details, seat availability, and pricing.
9. **Event Catalog - UI Seat Selection:**
    - This connector enables the Event Catalog to communicate with the User Interface to display available seats for selected events and facilitate seat selection by users.
10. **UI Seat Selection - Seat Reservation Management**:
    - This connector allows the User Interface to interact with the Seat Reservation Management component, initiating the reservation process for selected seats during the ticket purchase workflow.
11. **Seat Reservation Management - Event Database**:
    - This connector links the Seat Reservation Management component to the Event Database, enabling the management of seat availability and reservations based on real-time event data.

12. **Seat Reservation Management - Pricing Management**:
    - This connector facilitates communication between the Seat Reservation Management and Pricing Management components, ensuring that prices for selected seats are calculated accurately during the checkout process.
13. **Pricing Management - Event Database**:
    - This connector connects the Pricing Management component to the Event Database, allowing access to event-related pricing information such as base ticket prices and any applicable discounts or fees.
14. **Pricing Management - UI Checkout**:
    - This connector enables the Pricing Management component to communicate with the User Interface, providing pricing details to be displayed during the checkout process for user review.
15. **UI Checkout - Checkout Management**:
    - This connector links the User Interface to the Checkout Management component, initiating the checkout process after users have selected seats and reviewed pricing details.
16. **Checkout Management - Payment Gateway**:
    - This connector connects the Checkout Management component to the Payment Gateway, facilitating the processing of credit card transactions securely during the checkout process.
17. **Checkout Management - UI Confirmation**:
    - This connector enables the Checkout Management component to communicate with the User Interface, providing confirmation details to be displayed to users after a successful purchase.
18. **UI Confirmation - Communication Service**:
    - This connector links the User Interface to the Communication Service, triggering the sending of confirmation emails to users after successful purchases for their records.
19. **Checkout Management - Event Database**:
    - This connector connects the Checkout Management component to the Event Database, allowing it to store details regarding the checkout, including a successful and an unsuccessful transaction.

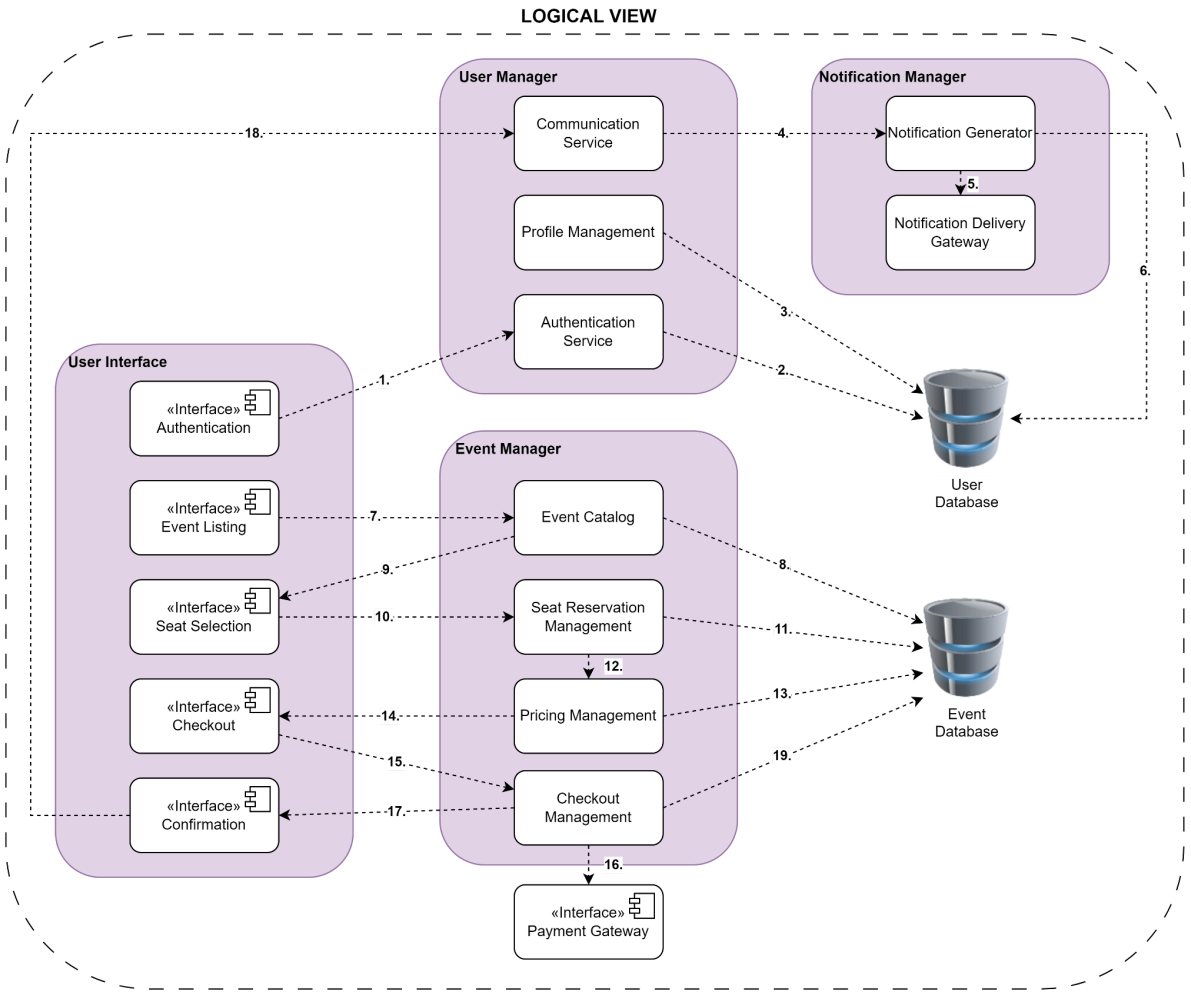


**Figure 1.** Logical View Diagram

# Process View

In this section, we have a sequence diagram that demonstrates the key interactions between the several components of the system:

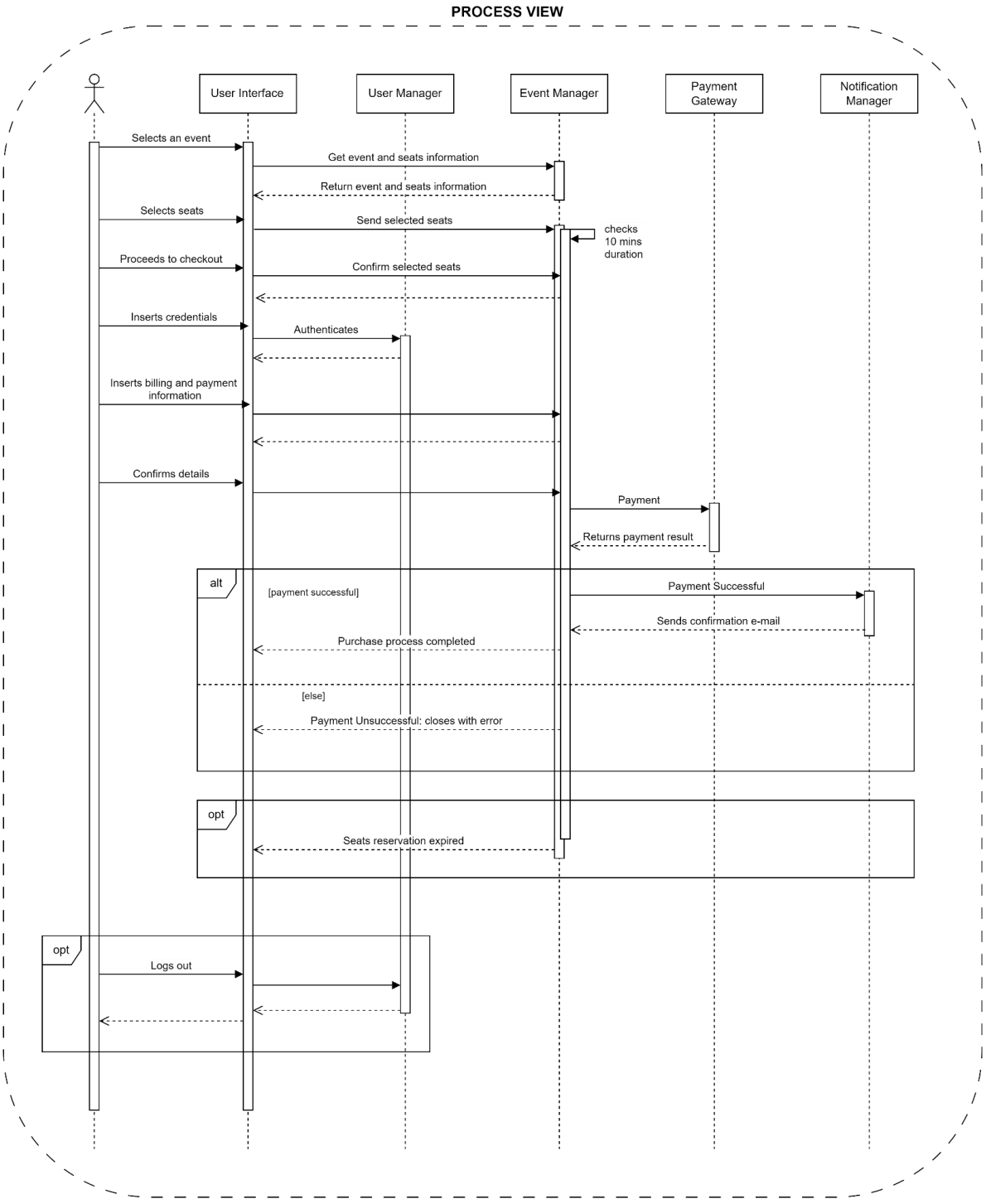


**Figure 2.** Process View Diagram

All the user interactions are made using the User Interface and this interface communicates with the other components.

The component Event Manager is responsible for all the interactions that involve the event catalog and the seats reservations/purchases. It also is responsible for creating the 10 minutes timer, when event seats are selected.

After the selection of seats, there are two options:

1) Proceed to checkout and respective payment;
2) After 10 minutes, the seats selection expires.

When the payment is executed, it may be successful or not. If successful, the purchase is completed and an email notification is sent to the user. If not, the purchase and seat reservation is canceled with an error.

# Physical View

The physical view of the TicketBoss system outlines the tangible infrastructure components and their interactions. This view illustrates the distribution of system elements across hardware and network resources.

## Customer

- **Mobile Phones**: TicketBoss is accessible via mobile phones, catering to users who prefer the convenience and mobility of handheld devices and to showcase their tickets at event entrances.
- **Desktops and Laptops**: Users can also access TicketBoss through desktop and laptop computers, via a web application.

## Internet

- The Internet serves as the primary medium through which client devices communicate with the TicketBoss backend infrastructure.
- Utilizing standard protocols and security measures, the Internet ensures reliable and secure data transmission between clients and servers.

## Backend Infrastructure

- **Load Balancer**: Positioned at the forefront of the backend infrastructure, the load balancer efficiently distributes incoming traffic among multiple servers to optimize

performance and maintain system reliability. It ensures that no single server becomes overwhelmed with requests, thereby preventing bottlenecks and ensuring scalability.

- **Server Cluster (N servers)**: The backend comprises a cluster of servers, with the number varying dynamically based on demand and load. Each server within the cluster hosts the application logic and processes user requests. The servers operate in parallel, collectively handling user interactions and system operations.
- **Event Database**: Each server in the cluster hosts a dedicated instance of the Event Database, which stores information related to entertainment events available for ticket sales and those that have ended. This includes event details such as event name, venue, date, seat map, available seats, and pricing information. The database is designed for efficient retrieval and manipulation of event data, ensuring fast response times during user interactions.
- **User Database**: Similarly, each server hosts a User Database, responsible for managing user-related information such as user profiles, authentication credentials, purchase history, and seat reservations. The User Database facilitates secure user authentication and authorization processes, ensuring that only authorized users can access sensitive functionalities such as ticket purchases and account management.

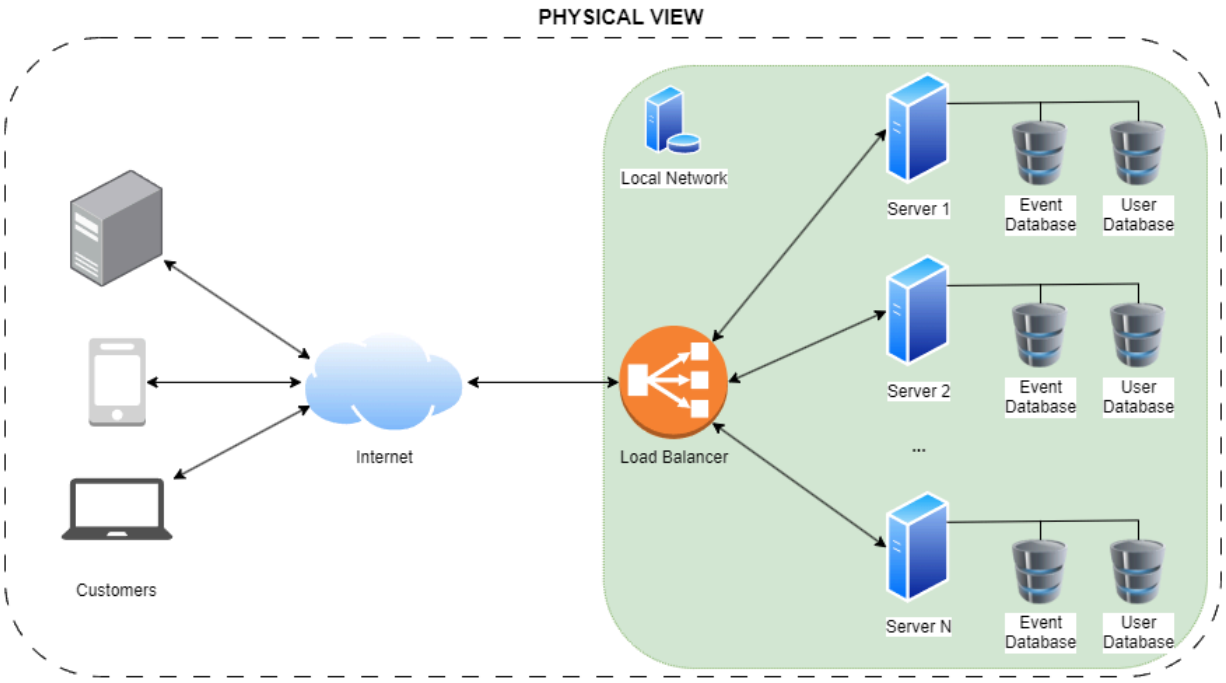


**Figure 3.** Physical View Diagram

## Use-case View

The Use-Case view outlines the interactions between the users and the system. It serves as a way to connect the previous views, as well as the Implementation view and the Deployment view not explored in this document.

In this section, we will explore some use cases of the system and identify and describe the core elements of each scenario. For each use case, we will start by identifying the **title** and **description**, to find the main goal of the use case and what it does. Next, we will identify the **actors**, i.e. the entities that interact with the use case, the **preconditions,** and the **postconditions**, meaning the conditions needed to be met before and after the use case, respectively. Lastly, we will identify possible paths for each use case, namely, a **primary path** with the most common route to success, an optional **alternate path** when there is a less common route taken to success, and an optional **exception path**, for when there is the possibility of an unsuccessful conclusion.

The Figure 4 Use Case Diagram illustrates the interaction between actors, use cases, and the system.

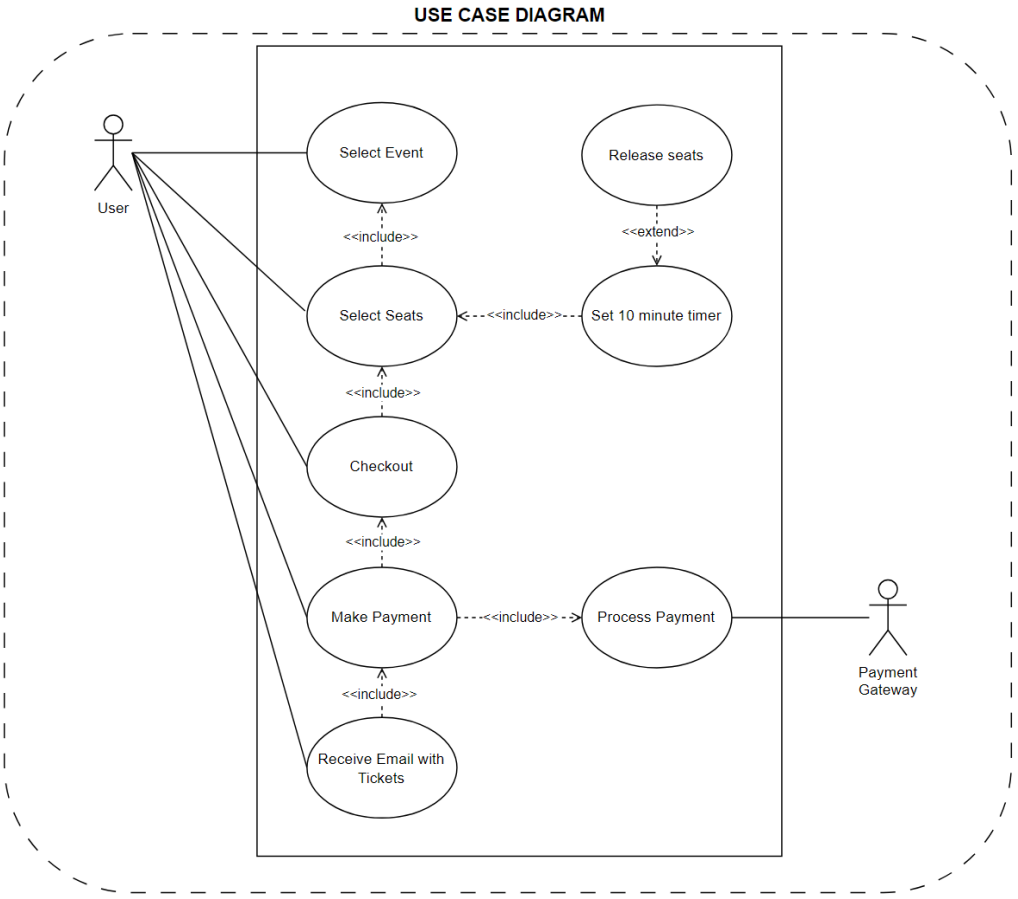


**Figure 4.** Use Case Diagram

- **Use Case 1**:

**Title:** User selects seats and buys them

**Description:** The use case consists of the selection and acquisition of seats for a certain event.

**Actors:** User and System.

**Preconditions:** Account created.

**Postconditions:** Purchase completed (payment and confirmation done) and e-mail notification with purchased tickets received. The seats purchased are no longer available in the selection and acquisition process. In an unsuccessful end, it should be displayed an error page explaining what happened, in the UI.

**Paths:**

- **Primary Path:**
    1) Select the event.
    2) Select the seats.
    3) Check the seat prices.
    4) Proceed to checkout.
    5) Fulfill payment and billing information.
    6) Confirm order details.
    7) After payment is received, an e-mail notification is sent to the provided e-mail address.
    8) Check e-mail to have access to the tickets.
- **Alternate Path:**
    1) Select the event.
    2) Select the seats.
    3) Check the seat prices.
    4) Wait 5 minutes to decide whether to buy the tickets.
    5) Proceed to checkout.
    6) Fulfill payment and billing information.
    7) Confirm order details.
    8) After payment is received, an e-mail notification is sent to the provided e-mail address.
    9) Check e-mail to have access to the tickets.
- **Exception Path: (use case 2)**
    1) Select the event.

2) Select the seats.
3) Check the seat prices.
4) Wait 11 minutes to decide whether to buy the tickets.
5) Seats have been already released and bought for someone else.
6) Purchase not completed with error “Seats no longer available”.

**Priority and Frequency of Use:** Top priority (main functionality) and very high frequency of use.

- **Use Case 2**:

**Title:** User selects seats and allows them to expire

**Description:** The use case consists of the selection of seats for a certain event without concluding the purchase, letting it expire.

**Actors:** User and System.

**Preconditions:** Account created.

**Postconditions:** The seats are released for someone else to buy.

**Paths:**

- **Primary Path:**
  1) Select the event.
  2) Select the seats.
  3) Check the seat prices.
  4) Wait for the 10 minute timer to expire.
- **Alternate Path:**
  1) Select the event.
  2) Select the seats.
  3) Check the seat prices.
  4) Proceed to the checkout.
  5) Wait for the 10 minute timer to expire.

**Priority and Frequency of Use:** Top priority, happens very often.

- **Use Case 3**:

**Title:** User selects seats, and then before the first ones expire, selects other seats for the same event. (Hmm. What should the system do? )

**Description:** The use case consists of the selection of a seat for a certain event, and later (without letting the selection expire) selection of other seats for the same event.

**Actors:** User and System.

**Preconditions:** Account created.

**Postconditions:** The seats are released for someone else to buy, if the user lets the 10 minute timer run out, or the purchase completed (payment and confirmation done) and e-mail notification with purchased tickets received, the seats purchased are no longer available in the selection and acquisition process, if the user finishes the purchase before the timer runs out.

**Paths:**

- **Primary Path:**
    1) Select the event.
    2) Select the seats.
    3) Check the seat prices.
    4) While the timer is still active (10 minute window), select additional seats. The timer does not reset.
    5) Wait for the timer to expire.
- **Alternate Path:**
    1) Select the event.
    2) Select the a seat.
    3) Check the seat prices.
    4) While the timer is still active (10 minute window), select additional seats. The timer does not reset.
    5) Proceed to checkout.
    6) Fulfill payment and billing information.
    7) Confirm order details.
    8) After payment is received, an e-mail notification is sent to the provided e-mail address.
    9) Check e-mail to have access to the tickets.

**Priority and Frequency of Use:** Top priority, happens very often.

- **Use Case 4**:

**Title:** Two or more people select seats for the same event at the same time (within ten minutes of each other.) There is no conflict.

**Description:** Users independently select seats for the same event within a ten-minute window of each other without causing conflicts or seat reservation issues.

**Actors:** User1, User2 (and potentially more users), and the System.

**Preconditions:** Both users have accounts created.

**Postconditions:** The seats are successfully reserved for each user based on their selections without overlapping or double bookings (no conflicts caused).

**Paths:**

- **Primary Path:**
    1) User1 selects seats for a specific event.
    2) The system holds the seats for User1 and begins a 10 minute timer.
    3) User2, within the timeframe of User1's selection, selects seats for the same event (the system only allows User2 to select seats that don't conflict with User1's selection)
    4) Both users proceed to checkout and complete the purchase.
    5) The system manages the seat reservations separately for User 1 and User 2 based on their selections, ensuring that each user's chosen seats are reserved and accounted for.

# Architecture Patterns

## Layers

This pattern is prevalent in the presented architecture. TicketBoss organizes the system into different layers, such as the presentation, application, and data access layers. That promotes the solution's modularity and maintainability, as the concerns are separated among different layers.

## Client-Server

Clients interact with the system via different kinds of user interfaces, sending requests to the server group for browsing events, and selecting and buying seats. This pattern involves consistency in the balanced distribution of data, which requires a lot of communication between clients and servers. This ensures scalability, fault tolerance, availability, and efficiency.

## Broker

This pattern eases the communication between components in a distributed system. Those can communicate without much detail, as the broker handles message routing and

ensures reliability upon the delivery of requests.

### Model-View-Controller

The MVC pattern is present in this solution, for instance in the presentation layer, where there must be a separation between data, the components displayed in the UI, and the user interaction handler. That separation further enhances reusability and modularity.

### Microservices Architecture

The presented application is structured as a group of loosely coupled services, like event management, seat visualization and selection, and payment processing. Each of those components can be independently deployed and possess its databases. Even though the communication between those services increases in complexity, and data consistency can become challenging, the implementation of this technique eases the release of updates to the production stage, and promotes fault isolation, maintainability, and scalability, thus, being worth it.

### Publish-Subscribe

This pattern could be present considering the existence of a notification module, used for updates about seat availability and event occupation. It represents the delivery of messages to interested subscribers, without parts being aware of each other.

## Quality Attributes - Requirements

TicketBoss's quality attributes were already explored in the previous sections, but this section aims to distill them further for easier comprehension and how they will be addressed within the architecture:

- **Scalability**: TicketBoss is capable of handling varying levels of traffic and event sizes. To address scalability, the architecture is designed to scale both vertically and horizontally to accommodate a large number of users accessing multiple events concurrently and to allow the distribution of workload across multiple servers. Scalability ensures that the system can handle peak loads during popular events, especially during the first release hours, without degradation in performance.
- **Performance**: Performance is crucial for TicketBoss to provide a seamless user experience. The system is optimized for fast event listing retrieval, seat selection, and purchase transactions. Minimal latency is essential to ensure that users can

swiftly navigate through the platform and complete their transactions without delays. Techniques such as database indexing, query optimization, caching mechanism, and asynchronous processing are employed to minimize latency and ensure smooth performance both on the user and server side.

- **Reliability**: Reliability is vital to ensure uninterrupted ticket sales and user satisfaction. TicketBoss exhibits high availability, ensuring that users can access the system and purchase tickets at any time without service disruptions. Additionally, transactional integrity is maintained throughout the whole purchasing process to prevent data loss or inconsistencies, which could lead to user dissatisfaction.
- **Security**: Security measures are imperative to protect users' sensitive information during transactions. TicketBoss adheres to industry standards for data encryption, secure communication protocols, and secure storage of payment details. Robust authentication and authorization mechanisms are implemented to prevent unauthorized access to user accounts and transactions and to mitigate the risk of data breaches.
- **Usability**: Usability plays a significant role in enhancing the user experience of TicketBoss. The system features an intuitive user interface that allows users to easily browse events, select seats, and complete purchases within a few steps. Clear and concise instructions are provided throughout the booking process to guide users effectively. A clear feedback mechanism and error handling are implemented to guide users through the ticket-buying process and minimize frustration.
- **Maintainability**: Maintainability is essential for the long-term sustainability of TicketBoss. The system is modular and well-documented, allowing for easy maintenance and updates. Codebase readability, clear separation of concerns, and adherence to coding standards facilitate efficient troubleshooting, debugging, and future enhancements.
- **Modularity**: Modularity is very important, as it allows for easy integration of new functionalities. The solution is designed with extensibility in mind for changing business needs. It also enables better fault isolation and is related to the Microservices architectural pattern mentioned above.
- **Interoperability**: Interoperability is crucial to achieving a smooth developing experience. The system supports industry standards regarding data formats and protocols and is seamless to integrate external systems, such as payment gateways and email providers.
- **Testability**: Testability is essential to enable comprehensive testing of TicketBoss

functionalities. The validation of system behaviors is obtained with mocking and stubbing, which allows to isolate dependencies. That ensures the good functioning of the solution.

These quality attribute requirements are foundational to ensuring that TicketBoss delivers a reliable, high-performance ticketing solution that meets the evolving needs of event organizers and attendees.

# Key Architectural Decisions - Rationale

## Microservices Architecture

The adoption of a microservices architecture offers several advantages for this project. By breaking down the system into smaller and independently deployable services, we achieved scalability and maintainability. Each microservice can be developed, deployed, and scaled independently. This architecture enables the system to remain operational even if one service fails. Additionally, microservices facilitate technology diversity, enabling us to choose the most appropriate tools and frameworks for each service's requirements.

## Publish-Subscribe Architecture

The necessity for effective user notification delivery of event modifications and purchases led to the decision to implement the Publish-Subscribe architectural pattern in TicketBoss. By using this approach, the system improves scalability and decoupling by enabling smooth communication between components that aren't directly dependent on one another. The Publish-Subscribe approach ensures timely information distribution by enabling dynamic updates to be sent in real-time to interested subscribers without requiring mutual knowledge from each component. This pattern improves the system's extensibility and flexibility, facilitating efficient administration of seat bookings and event occupancy while fostering a responsive user experience.

## Decentralized Data Management

To enhance scalability and fault tolerance, TicketBoss employs a decentralized approach to data management. Each server in the cluster hosts dedicated instances of the **Event** and **User Databases**, allowing for distributed data storage and processing. This decision ensures that the system can handle high traffic volumes and maintain performance during peak times.

## Security Protocol

Security is paramount in a system handling sensitive transactions. TicketBoss implements industry-standard security protocols for data encryption and secure communication. The rationale is to protect user data and prevent unauthorized access, ensuring trust and compliance with data protection regulations.

# Conclusion

The TicketBoss architecture offers a complete solution for managing ticket sales for entertainment events. Through a modular architecture, the system efficiently handles user interactions, seat reservations, and transactions. The logical view shows several components for user interface, user management, event management, notification management, and data administration. Process view presents dynamic interactions for key use cases, with the interactions between the several components. Additionally, detailed use-case scenarios address various user interactions and potential conflicts. The architecture emphasizes performance, reliability, and security, catering to specific requirements of the assignment. Overall, TicketBoss architecture has a robust design for efficient ticketing operations.