TicketBoss

Software Systems Architecture

Group 4 - Class 3

M.EIC 2023/2024

Gustavo Costa	up202004187@edu.fe.up.pt
João Pinheiro	up202008133@edu.fe.up.pt
João Oliveira	up202004407@edu.fe.up.pt
Pedro Fonseca	up202008307@edu.fe.up.pt
Ricardo Cavalheiro	up202005103@edu.fe.up.pt

Index

Logical View

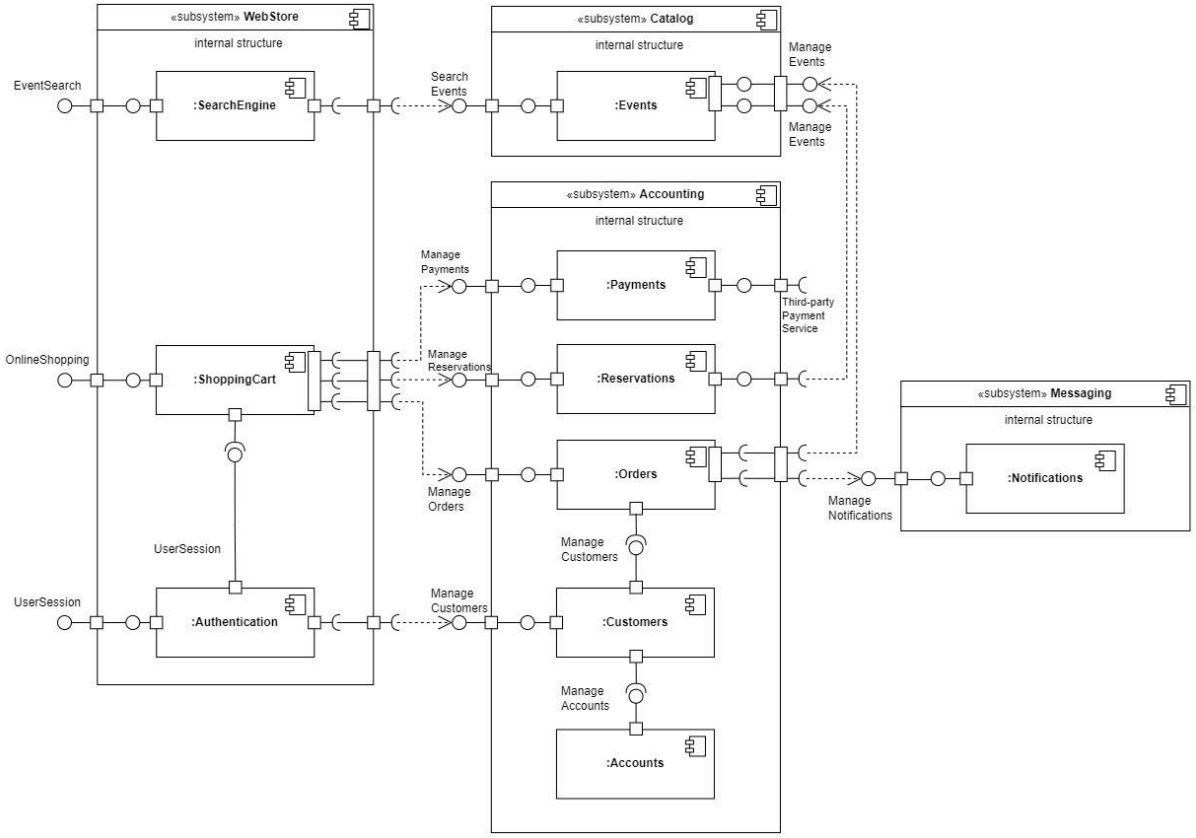

Figure 1 - Logical View

The TicketBoss system architecture is designed to facilitate efficient ticket sales for various entertainment events while ensuring a seamless user experience. The logical view (**Figure1**) displays subsystem components (“WebStore”, “Catalog”, “Accounting”, and “Messaging”) that handle different aspects of the system's functionality.

The “WebStore” subsystem encompasses the search engine, the shopping cart, and the authentication. Each is responsible for a different job:

- Search engine: Searches through the catalog of events.
- Shopping cart: Handles the user’s shopping cart and calls the respective part components responsible for handling orders, payments, and reservations.
- Authentication: Handles the authentication of users.

The “Catalog” subsystem only has one part component which is called “Events” that is responsible for managing the events a user can buy tickets to.

The “Accounting” subsystem has five part components:

- Payments: Handles payments and calls a third-party payment processor.
- Reservations: Handles reservation of seats in the shopping cart of a user, coordinating with the "Events" part component to update seat availability in real-time.

- Orders: Handles the user’s orders, coordinating with the "Events" part component to update seat availability in real-time, and calls the part component responsible for notifying the user of a fulfilled order.
- Customers: Handles the relationship between the orders and the users who bought them, a.k.a., the customers.
- Accounts: Handles users’ accounts.

The final subsystem is the “Messaging” subsystem, which is responsible for notifying the users of fulfilled orders and successful transactions

This component-driven architecture promotes scalability and maintainability, allowing for easy integration of new features or updates. Furthermore, by separating concerns into distinct components and subsystems, the system enhances fault tolerance and reliability, minimizing the impact of potential failures.

Overall, the TicketBoss architecture is designed to meet the system's functional requirements while addressing important quality attributes such as performance, security, and reliability.

Process View

The TicketBoss system's process view illustrates the sequence of interactions among its various components to fulfill specific user scenarios.

This process view highlights the orchestration of system components to deliver seamless user experiences while ensuring data integrity, security, and reliability. By breaking down complex interactions into manageable steps, the system effectively handles various user scenarios and maintains consistency across transactions.

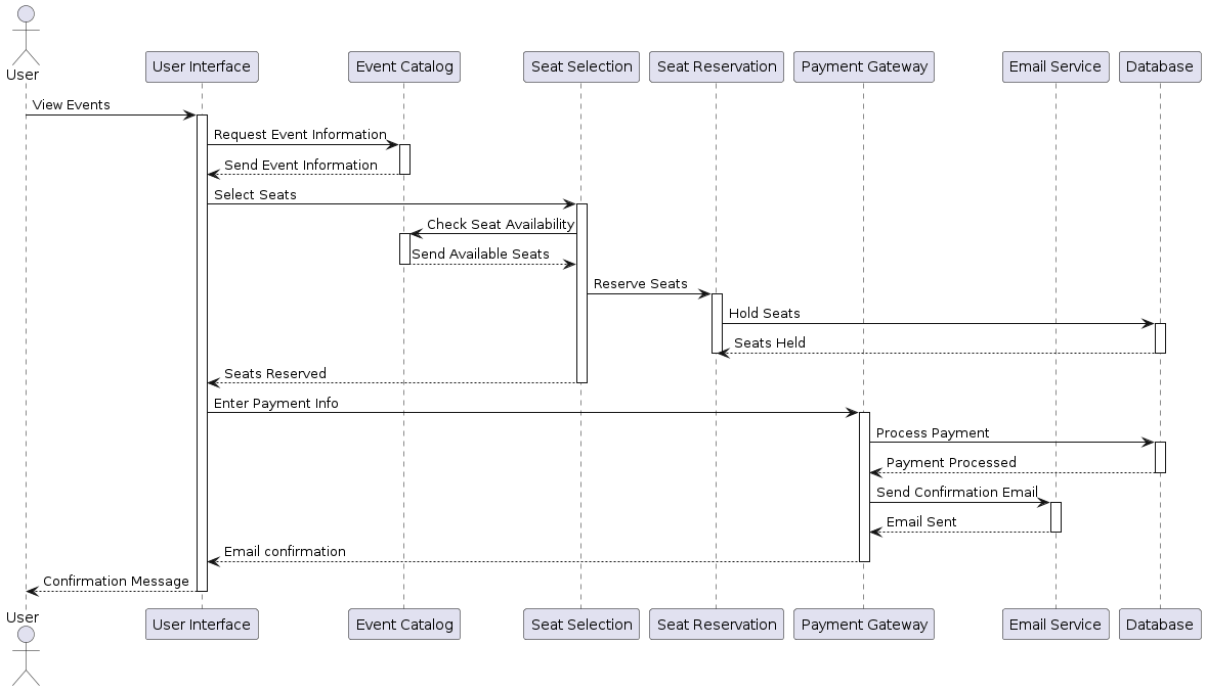

Figure 2.1 - Process View

In the first scenario, a user interacts with the system through the User Interface module to view available events. Upon selecting seats, the Seat Selection module checks seat availability with the Event Catalog. Subsequently, the Ticket Reservation module holds the selected seats in the database for a specified period. The user then proceeds to enter payment information, which is processed securely through the Payment Gateway. Finally, upon successful payment, the Email Service sends a confirmation email to the user, completing the transaction.

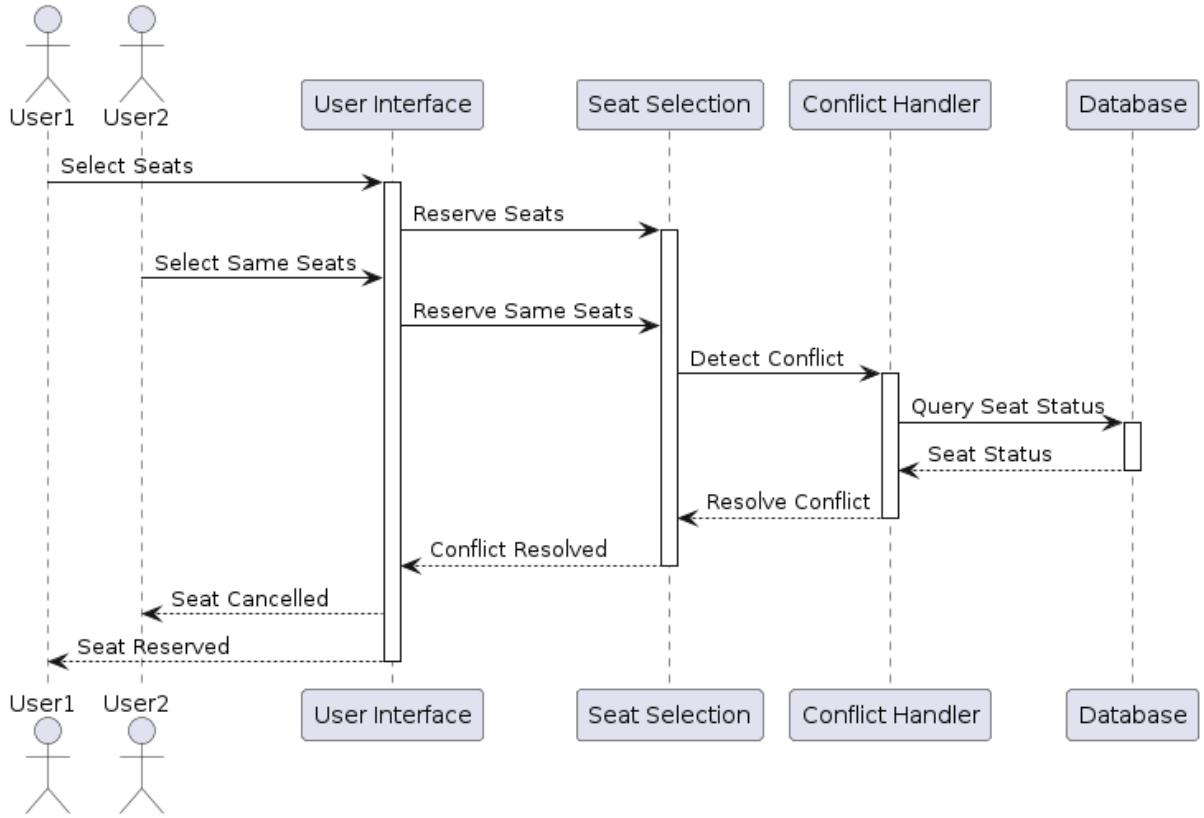

Figure 2.2 - Process View of a Conflict

In the second scenario, two users simultaneously attempt to reserve the same seats for an event. Both users interact with the User Interface module to select seats, which triggers a conflict detection mechanism within the Seat Selection module. The Conflict Handler module is then invoked to resolve the conflict by querying the database for seat status. Once resolved, the appropriate notifications are sent to both users, informing them of the outcome.

The modular design of the TicketBoss system enables flexibility, scalability, and ease of maintenance. Each component performs specific tasks independently, facilitating easier debugging, testing, and future enhancements. Additionally, the system's fault-tolerant architecture minimizes the impact of failures, ensuring uninterrupted service for users.

Use-Case View

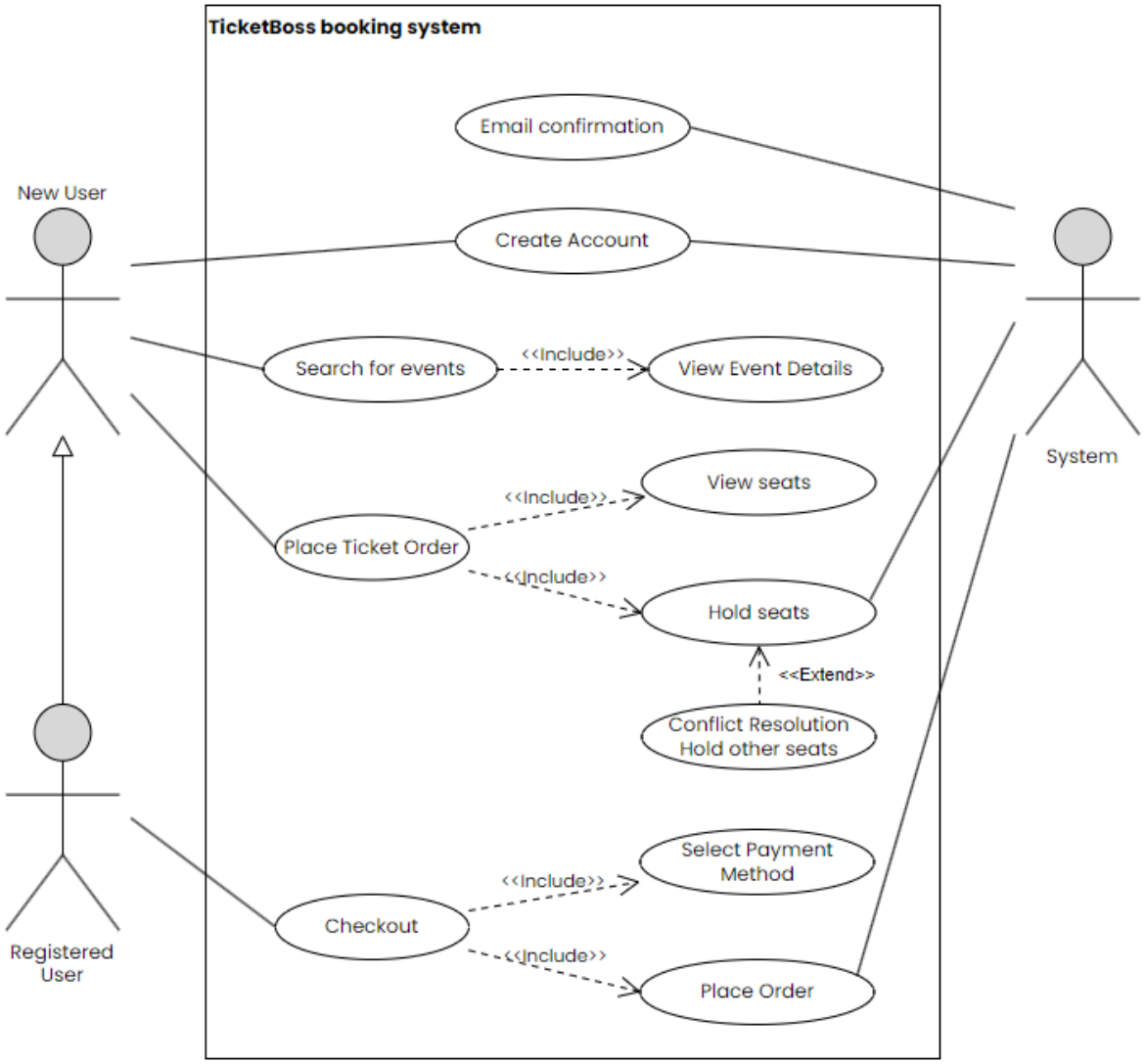

Figure 3 - Use Case View

The above diagram depicts a use-case view for TicketBoss. Here's a breakdown of the functionality:

Actors:

- Registered User: An individual who has created an account within the TicketBoss system.
- New User: An individual who has not yet created an account within TicketBoss.
- System: TicketBoss servers.

Use Cases:

1. User Selects Seats and Buys Them (Registered User):
    - The registered user searches for events using the search functionality.
    - The system displays a list of available events.
    - The user selects an event to view details and available seats.
    - The system displays the event details and a seating chart.
    - The user selects the desired seats and views their corresponding prices.
    - The system holds the selected seats for 10 minutes.
    - The user proceeds to checkout by entering their billing information and selecting a payment method.
    - The system processes the payment and sends a confirmation email to the user.
    - The seats are reserved for the user, and no longer available for purchase by others.

2. Two or More People Select Seats for the Same Event, With a Conflict (They Try to Select the Same Seats) (Registered Users, User A and User B):

    Precondition: Multiple users are browsing the same event with available seats.
    - User A selects one or more seats for the event.
    - The system holds the selected seats for User A.
    - User B attempts to select some of the same seats already chosen by User A.
    - The system detects a conflict and prevents User B from selecting the conflicting seats.
    - The system displays an error message to User B informing them that the chosen seats are unavailable.
    - The system offers alternative seat selections to User B (display map with highlighted available seats).

Additional Use Cases:

- User Allows Seats to Expire (Registered User):
    - The user selects seats and the system holds them for 10 minutes.
    - If the user does not complete the checkout process within the 10-minute window, the system automatically releases the held seats.
- User Selects Seats, Then Selects Other Seats for the Same Event Before the First Ones Expire:
    - The system should allow the user to hold additional seats for the same event as long as the total doesn't exceed 10 seats.
    - The system should update the user's selections and hold all chosen seats until checkout or expiration.
- Two or More People Select Seats for the Same Event at the Same Time (Within Ten Minutes of Each Other) There is No Conflict (Registered Users):
    - The system should grant the seats to the users who selected them first.
    - The system should notify the unsuccessful users that the seats they chose are no longer available.

- User Views Seats Without Selecting Any (Registered User or New User):
    - The user can browse events and view available seating charts without registering or creating an account.
    - The system will not allow seat selection or checkout functionalities unless the user creates an account or logs in.
- New User:
    - A new user can create an account by providing their personal information.
    - Once an account is created, the new user becomes a Registered User with full access to system functionalities.
- User Holds Seats in Two or More Events at a Time (Registered User):
    - The system allows a registered user to hold seats for up to 10 events simultaneously.
    - During checkout, the user can select which seats to complete the purchase for.
    - Similar to single-event checkout, the system will process payment and send confirmation emails.
- User Tries to Exceed the Maximum by Holding Multiple Seats in Multiple Events (Registered User):
    - The system should prevent users from exceeding the 10-seat maximum holding capacity.
    - The system should notify the user when they attempt to hold more than 10 seats and advise them to remove seats from their selection.
- User Holds Multiple Seats to the Max. Some Expire and the User Gets More Seats, Reaching the Max Again (Registered User):
    - The system automatically releases seats that are not purchased within the 10-minute holding period.
    - The user can select new seats as long as the total holding capacity remains under 10 seats.

Physical View

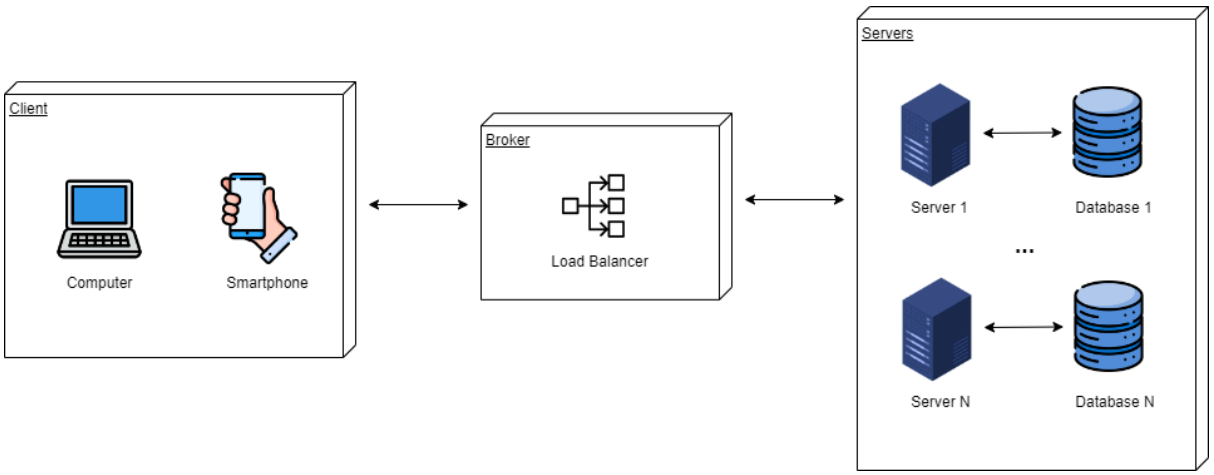

Figure 4 - Physical View

The Physical View of TicketBoss employs a load balancer to distribute client requests across multiple servers for optimal performance and fault tolerance. Each server is connected to its respective database, ensuring efficient data management. This architecture enables scalability by accommodating increasing user demand with the addition of servers.

Architectural Patterns

Three Tiers

The TicketBoss system separates concerns into three distinct tiers: the User Interface (presentation tier), the backend supporting the UI (application tier), and the Database group (data tier). This separation allows for independent development and modification of each tier, improving scalability and maintainability.

Microservices

The “Reservations” and the “Notifications” part components in the TicketBoss system can be seen as microservices. They are small, autonomous services that fulfill specific business capabilities. They can be developed, deployed, and scaled independently, which enhances the system’s flexibility and resilience.

Event-Driven

The “Notifications” part component in the TicketBoss system is a perfect example of an event-driven component. When a transaction is successful (an event), it triggers the service to send a confirmation email to the user. This pattern enables the system to respond promptly to state changes and enhances user experience.

Layered

The TicketBoss system is already using a form of layered architecture, with distinct layers for user interface, business logic, and data storage. Each layer has specific roles and responsibilities, and each layer depends only on the layer beneath it. This separation of concerns promotes flexibility and maintainability.

Service-Oriented

The “Payments” part component in the TicketBoss system can be seen as a service in a Service-Oriented Architecture. It provides a service (payment processing) to other components (in the “WebStore” subsystem) through a communication protocol (API call to the external payment service). This pattern promotes interoperability and reusability.

Broker

This architectural pattern is noticeable in the physical diagram, in which the broker functions as a load balancer, distributing requests from clients among servers to optimize performance. It routes requests based on factors like server load and availability, ensuring balanced resource usage and enhancing system reliability and scalability. We didn’t mention client-server as an architectural pattern as brokers are extensions of Client-Server systems.

Quality Attributes

Scalability

What does it mean?

Large-scale events attract a significant number of attendees, translating to a substantial volume of data, including event-related transactions. To effectively manage this, the system must seamlessly be able to scale both horizontally and vertically to accommodate the surge in users.

Acceptance Level? 8/10

The system's performance requirements are contingent upon the events hosted on the platform. The system must maintain responsiveness, particularly during peak periods such as popular events. While occasional failures may occur, they should be manageable within an acceptable range.

How important?

When taking into account the user experience, this is very important for the user to buy tickets at the same time as everyone else on the platform.
If not done, will only cause dissatisfaction of the user. Users aren't able to buy tickets and will most certainly not like to use the platform. Luckily for these platforms, tickets usually aren’t sold anywhere else so the user doesn’t have a choice but to use it.
A system failure may lead to big losses in revenue and sales.

Security

What does it mean?

It's important to prioritize the privacy of users' data, ensuring that their information remains confidential and protected from any unauthorized access. It also highlights the necessity of implementing robust security measures specifically tailored to safeguard payment transactions, thereby guaranteeing the integrity and safety of financial interactions within the system.

Acceptance Level? 8/10

The system requires robust security measures to safeguard user data and ensure the successful completion of payment transactions. Moreover, it's crucial to prevent ticket theft, as users expect to fully utilize the experiences they've paid for.

How important?

Protecting user data and ensuring secure payment transactions are fundamental aspects of maintaining trust and reliability in the platform. Additionally, preventing ticket theft directly impacts user satisfaction and the perceived value of the services provided. Therefore, addressing these concerns is crucial otherwise it will be putting at risk both the users and the organization.

Usability

What does it mean?

A user-friendly platform prioritizes easy navigation and accessibility, especially for finding events. It ensures events are easily searchable and presents information clearly, including dates, locations, descriptions, and ticket prices. Making it easy to buy tickets is important for keeping customers happy. When we focus on making this process simple and efficient, it makes customers happier and more likely to keep coming back.

Acceptance Level? 7/10

Users should be able to effortlessly discover events and purchase tickets within just a few clicks, with a process that is intuitively designed and user-friendly. This means that navigating the platform should be straightforward, allowing users to quickly find events of interest and seamlessly proceed through the ticket-purchasing process

How important?

Usability is vital for a user-friendly platform, especially for event booking applications. Though it can take a backseat to security or scalability concerns, it should still be a priority. The interface should be intuitive, responsive, and easy to navigate, ensuring users can find events and purchase tickets without hassle. Ignoring usability can lead to user dissatisfaction, so it's crucial to strike a balance with other considerations.

Performance

What does it mean?

Efficient performance is key for the platform to succeed. It's about being quick, responsive, and running smoothly. Fast navigation and timely feedback keep users happy, while making sure tasks are handled efficiently saves resources and reduces downtime. This creates a seamless experience that users appreciate and keeps them coming back for more.

Acceptance Level? 6/10

The system's responsiveness to user interactions is crucial, requiring quick reactions to clicks and requests for information to deliver a smooth experience. Furthermore, it should maintain optimal performance even during peak periods of activity, ensuring uninterrupted access for all users.

How important?

Performance is crucial for the success of the entire business. If the system is not fast and responsive, event organizers may opt out of selling their tickets on the platform, potentially leading to a loss of revenue and credibility. Beyond just business transactions, system performance profoundly affects the user experience and overall satisfaction. A sluggish or unreliable platform can frustrate users, leading to dissatisfaction and possibly driving them away from TicketBoss. Therefore, prioritizing system performance is not just important; it's indispensable for maintaining trust and attracting users.