Faculdade de Engenharia da Universidade do Porto

Ticket Boss

Software Systems Architecture

Team T24:

Henrique Oliveira Silva up202007242@up.pt

João Pedro Matos Araújo up202007855@up.pt

Rui Filipe Cunha Pires up202008252@up.pt

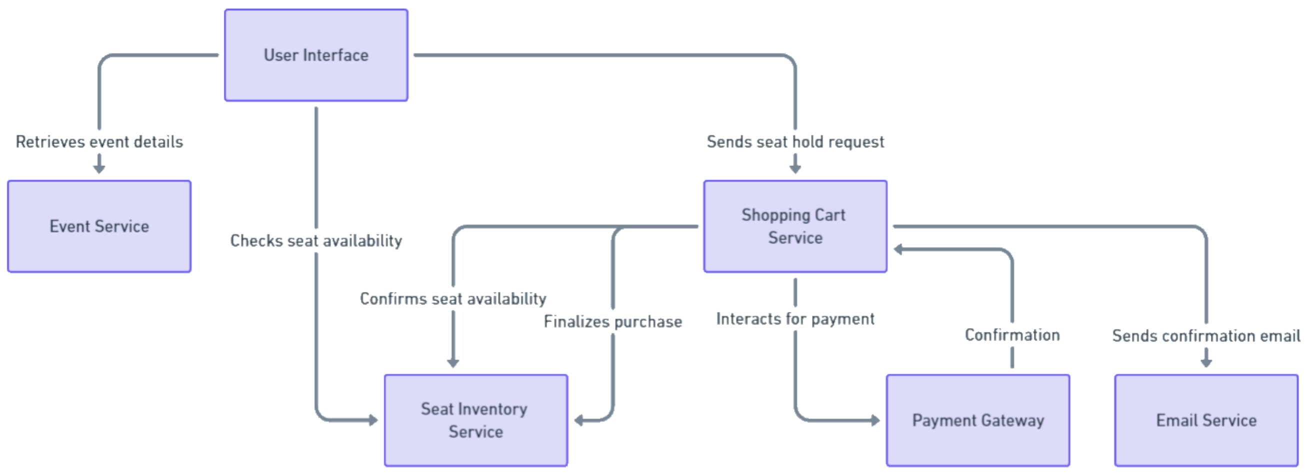
Pedro Miguel Nunes up202004714@up.pt

Diogo Miguel Ferreira Costa up202007770@up.pt

1. Architecture Overview

TicketBoss should be designed using a microservices architecture to ensure scalability, maintainability, and flexibility. It will consist of several components working together to provide the required functionalities.

2. Logical View: Components and Connectors Diagram



The system will consist of the following main components:

- **User Interface (UI)**: This is the web application that users will interact with to browse events, select seats, and purchase tickets.
- **Event Service**: This service manages event information, including details like event name, date, time, and seat availability.
- **Seat Inventory Service**: This service tracks the real-time availability of seats for each event. It ensures only one user can hold a particular seat at a time.
- **Shopping Cart Service**: This service manages the user's selected seats and calculates the total price. It also holds the seats for a limited time (10 minutes in this case).
- **Payment Gateway**: This is an external service (not implemented within TicketBoss) that securely processes credit card payments.
- **Email Service**: This service sends confirmation emails to users after successful purchases.

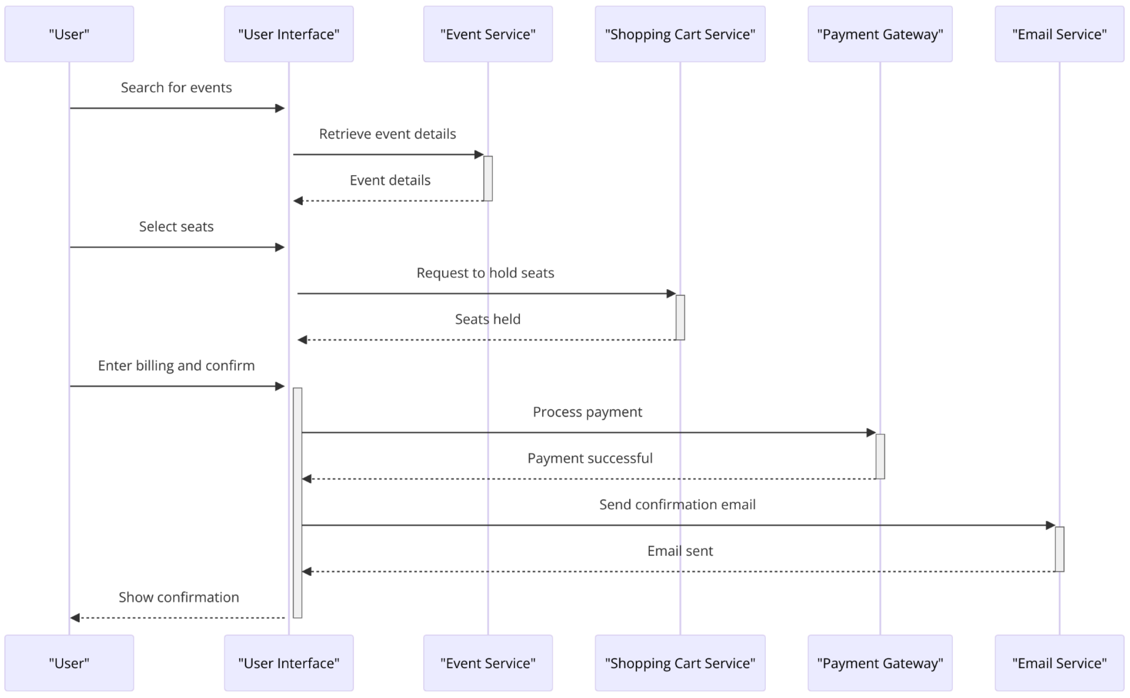
These components will interact through well-defined APIs (Application Programming Interfaces).

- The UI will interact with the Event Service to retrieve event information and the Seat Inventory Service to check seat availability.
- When a user selects seats, the UI will send a request to the Shopping Cart Service to hold the seats.
- The Shopping Cart Service will communicate with the Seat Inventory Service to confirm seat availability and reserve them for the user.
- Upon checkout, the Shopping Cart Service will send the user's information and seat selections to the Payment Gateway for processing.
- If the payment is successful, the Shopping Cart Service will finalize the purchase and send a confirmation email through the Email Service.

3. Process View: Sequence Diagrams

The process view describes the interaction among components through sequence diagrams. This is how the components interact for the key use cases:

User Selects Seats and Buys Them



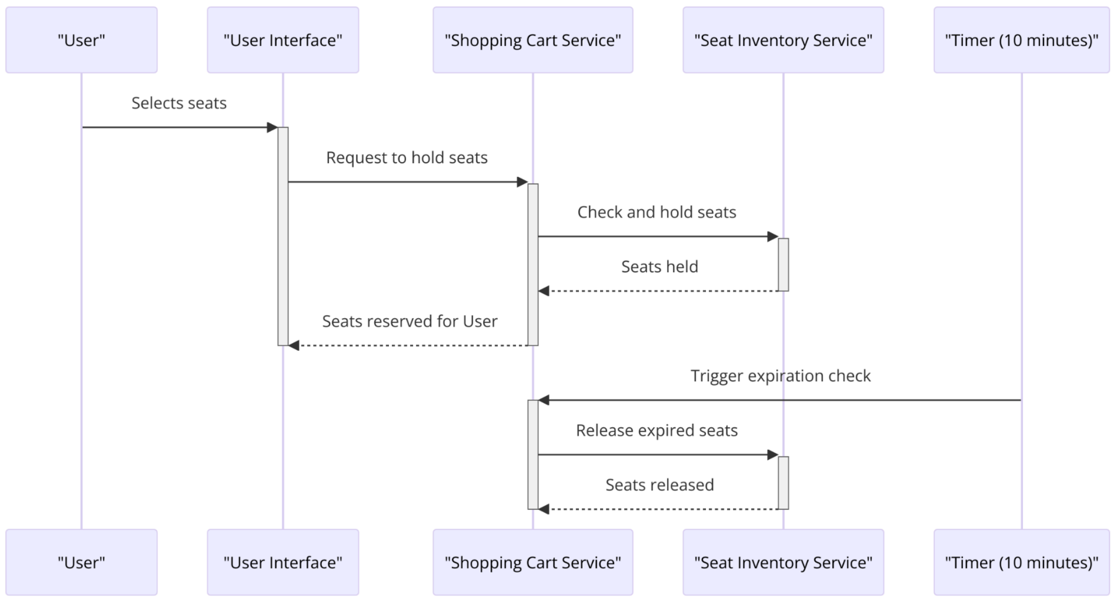
Events overview:

1. User searches for events on the UI.
2. UI retrieves event details from the Event Service.

3. User selects seats on the UI.
4. UI sends a request to the Shopping Cart Service to hold the seats.
5. Shopping Cart Service checks availability with Seat Inventory Service.
6. If available, Shopping Cart Service reserves the seats.
7. User enters billing information and confirms purchase on the UI.
8. UI sends payment information to the Shopping Cart Service.
9. Shopping Cart Service interacts with the Payment Gateway for processing.
10. Payment Gateway sends confirmation back to the Shopping Cart Service.
11. Upon successful payment, Shopping Cart Service finalizes the purchase and marks seats as sold in the Seat Inventory Service.
12. An email confirmation is sent to the user via the Email Service.

User Allows Seats to Expire

The user takes no action after selecting seats. After 10 minutes, a background process:



1. Retrieves held seats from the Shopping Cart Service.
2. Contacts the Seat Inventory Service to release the expired holds.

4. Use-case View

Use Case 1: User selects seats and buys them

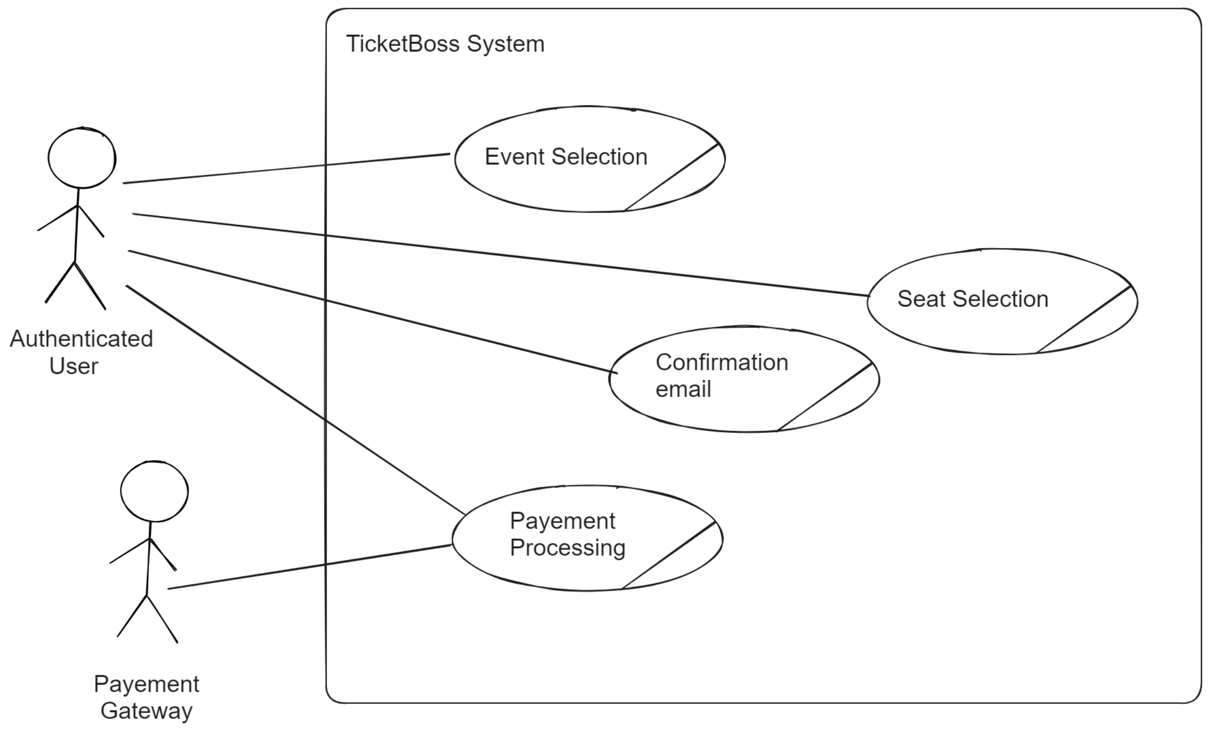
Actors: User, Payment Gateway

Preconditions: User is logged in.

Scenario:

1. User selects an event to attend.
2. User selects seats from the available options.
3. User confirms the selection and proceeds to payment.
4. User enters payment information.
5. Payment gateway takes care of the payment.
6. System sends a confirmation email to the user.

Postconditions: User has successfully purchased seats, and seat availability is updated.



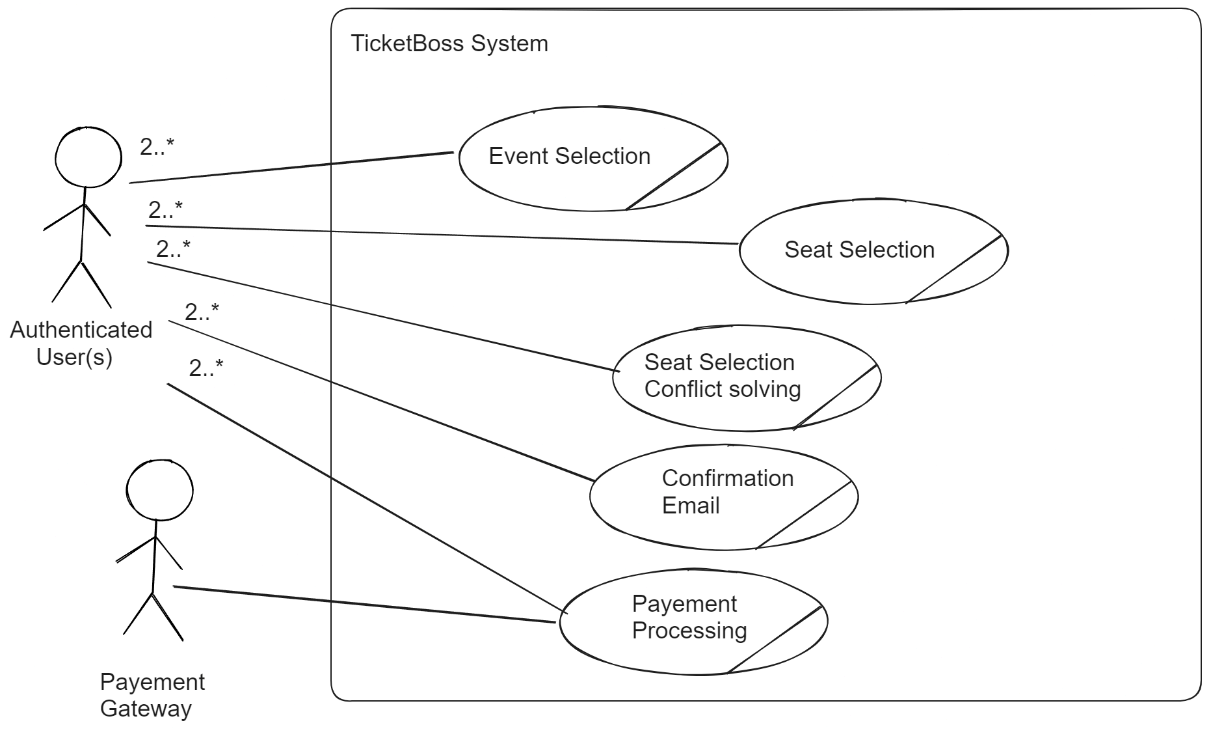
Use Case 4: Two or more people select seats for the same event, with no conflict

Actor: User1, User2

Preconditions: Multiple users are viewing the same event.

Scenario:

1. User1 and User2 select the same event to attend.
2. Both users select seats from the available options.
3. If a seat is selected by another user and held, the system immediately updates the seat status and notifies other users attempting to select the same seat.
4. Each user confirms their selection and proceeds to payment independently.
5. The system processes payments for each user separately.
6. System sends email confirmations to users.
7. System updates the seat availability.



5. Additional Considerations

Quality Attributes

Quality Attribute	**What does it mean?**	**Acceptance level?**	**How important?**
Scalability	Scalability refers to the system's ability to handle increasing user loads by adding resources or distributing the workload across multiple instances of components.	The system should be able to handle an increase in concurrent users without significant loss in performance.	High. As TicketBoss attracts more users and events, the ability to scale efficiently is crucial to maintain a responsive and reliable platform.
Availability	Availability refers to the system's ability to remain operational and accessible to users, even in the case of failures or disruptions.	The system should ensure high availability, aiming to be operational for the majority of the time. This includes maximizing the up time ensuring that any disruptions are kept within acceptable limits.	Critical. Continuous availability ensures uninterrupted access to ticket sales, preventing revenue loss and maintaining customer satisfaction.
Security	Security refers to the measures to protect the confidentiality, integrity, and availability of data and resources within the system.	The system must adhere to standard security practices, such as encryption of sensitive data and input sanitization.	High. Protecting user data and financial transactions is crucial to maintain trust and credibility with customers. Any compromise in security could lead to financial losses and damage to the company's reputation.
Usability	Usability concerns the ease of use and intuitiveness of the system's user interface, ensuring that users can navigate and perform tasks efficiently.	The system's user interface should undergo usability testing with a sample of users, achieving general levels of satisfaction.	Moderate. Usability directly impacts user satisfaction and retention, reducing errors and support requests, which enhances overall efficiency and

			customer experience.
Portability	It's about how easily the system can move between different platforms, environments and devices like mobile smartphones etc.	The system should run on various cloud platforms, phones , other devices and on-premises setups without major changes.	Moderate. It ensures flexibility in deployment, crucial for adapting to different infrastructure needs.

Architectural Patterns

- **Microservices Architecture**: This architectural style has been adopted due to its inherent advantages:
  - **Modularity**: Development is facilitated by the compartmentalization of the system into smaller, independent services.
  - **Deployment Flexibility**: Individual services can be deployed and updated independently, promoting agility.
  - **Scalability**: Specific services can be scaled based on their resource demands, optimizing overall system performance.
- **Event-Driven Architecture**: Asynchronous communication between components is facilitated by an event-driven architecture. This approach allows for non-blocking interactions, such as sending confirmation emails, improving system responsiveness for users.

Architectural Decisions

- **Microservices Architecture**: The microservices architecture was chosen for its ability to promote modular development, independent deployments, and efficient horizontal scaling.
- **Asynchronous Communication**: Asynchronous communication was selected to ensure non-blocking interactions, enhancing system responsiveness during user interactions like seat reservation and email confirmation.