Faculdade de Engenharia da Universidade do Porto

Software Systems Architecture

Homework #07 - TicketBoss Architecture

Team T23:
Anete Pereira (up202008856)
Bárbara Carvalho (up202004695)
David Fang (up202004179)
Milena Gouveia (up202008862)
Pedro Correia (up202006199)

1. Introduction

The demand for convenient and efficient ticketing solutions for entertainment events is greater than ever. The TicketBoss system provides users with a platform to browse events, select seats and purchase tickets with ease. This report looks at the architecture, functionality and quality attributes of the TicketBoss system, aiming to provide a comprehensive understanding of its design and capabilities. From the logical view of components and connectors to the process view of user interactions, every aspect of system operation is examined to highlight its strengths and address potential challenges. In addition, key quality attributes such as scalability, reliability and ease of use are analyzed to ensure that TicketBoss meets the expectations of users and stakeholders.

2. System Requirements

2.1. Functional Requirements

- **Display Entertainment Events**: Users can view a list of available entertainment events.
- **Show Available Seats**: Users can see seat availability for a selected event.
- **Hold Seats for 10 Minutes**: Seats are reserved for a user for 10 minutes while they complete the purchase process.
- **Purchase Tickets**: Users can buy tickets by providing payment information.
- **Send Confirmation Emails**: Users receive confirmation emails after completing a successful purchase.
- **Concurrent Access to the Same Event**: Multiple users can access and select seats for the same event simultaneously.
- **Limit of 10 Seats at a Time**: Users are restricted to holding a maximum of 10 seats during the purchase process.

2.2. Non Functional Requirements

- **Scalability**: The system should be able to handle multiple concurrent users and events.
- **Reliability**: Seats should be accurately held and released within the specified time frame.
- **Performance**: Response time should be fast, especially during seat selection and purchase.
- **Security**: User information and transactions should be securely handled.
- **Availability**: The system should be available for use during peak ticket sale times.
- **Maintainability**: The architecture should be modular and easy to maintain and update.

3. Ambiguities and Decisions

When designing the Ticket System Architecture, we identified multiple ambiguities in the problem's specification and in order to address these uncertainties within the system, we made several decisions based on our understanding of the project requirements. Below are the aforementioned ambiguities along with the decisions we made in order to address them.

3.1. Data Types

- **Event**: Represents an entertainment event available for ticket purchase. It includes attributes such as event name, date, venue, available seats, and ticket prices.
- **Seat**: Refers to individual seats or tickets available for an event. Each seat may have attributes such as seat number and availability status.
- **User**: Represents the person using the system. Each user has an username, a name and a password.
- **Shopping Cart**: Represents a shopping cart associated with a user. Each shopping cart has a timer and the tickets the user hasn't bought.
- **Ticket**: Represents a ticket purchased by a user for attending an event. Each ticket has a purchase date, a status (bought, not bought) and the seat associated with it.

3.2. System Usage

The TicketBoss system is primarily accessed and utilized through a web-based interface, commonly referred to as the TicketBoss web page. Users access the system using standard web browsers on their desktop computers, laptops, tablets, or smartphones. The web page provides a user-friendly interface for browsing available events, selecting seats, and purchasing tickets. It allows users to search for events based on criteria such as event type, date, location, or artist, and provides detailed information about each event, including available seats and pricing.

When entering an event, the system automatically creates one ticket for the user. If no seats are available the event won't show up. After entering an event, the user can buy more tickets for the event by selecting other seats. The tickets are added to their shopping cart, where they are temporarily held for a predefined period (e.g., 10 minutes) while users complete the checkout process. If the checkout process is not completed within the allocated time, the system automatically releases the held seats back into the available pool for other users to purchase. Additionally, the system may notify the user of the expiration of their held seats and prompt them to refresh the page or restart the ticket selection process if they wish to continue. During checkout, users provide payment information, such as credit card details, to finalize the ticket purchase. Upon successful completion of the transaction, users receive confirmation emails containing details of their ticket purchase and event information.

4. Architecture

For our architecture we had in mind the following architecture patterns:
- **Layered Architecture**: The system is divided into logical layers, each responsible for different functionalities. We decided on three layers, a presentation layer that deals with the user interface, a logic layer for the main logic of the application and a data management layer responsible for the requests to the database.
- **Client-Server Architecture:** The client (web-browser) interacts with the server-side components to perform various actions such as browsing events, selecting seats, and purchasing tickets. The server handles requests from clients, processes business logic and communicates with external systems such as the Payment System and the Contact System.
- **Microservices Architecture**: The system is decomposed into a collection of loosely coupled services, each responsible for a specific task, allowing the system to be more flexible and for easier sustenance.

4.1. Logical View

4.1.1. Context Diagram

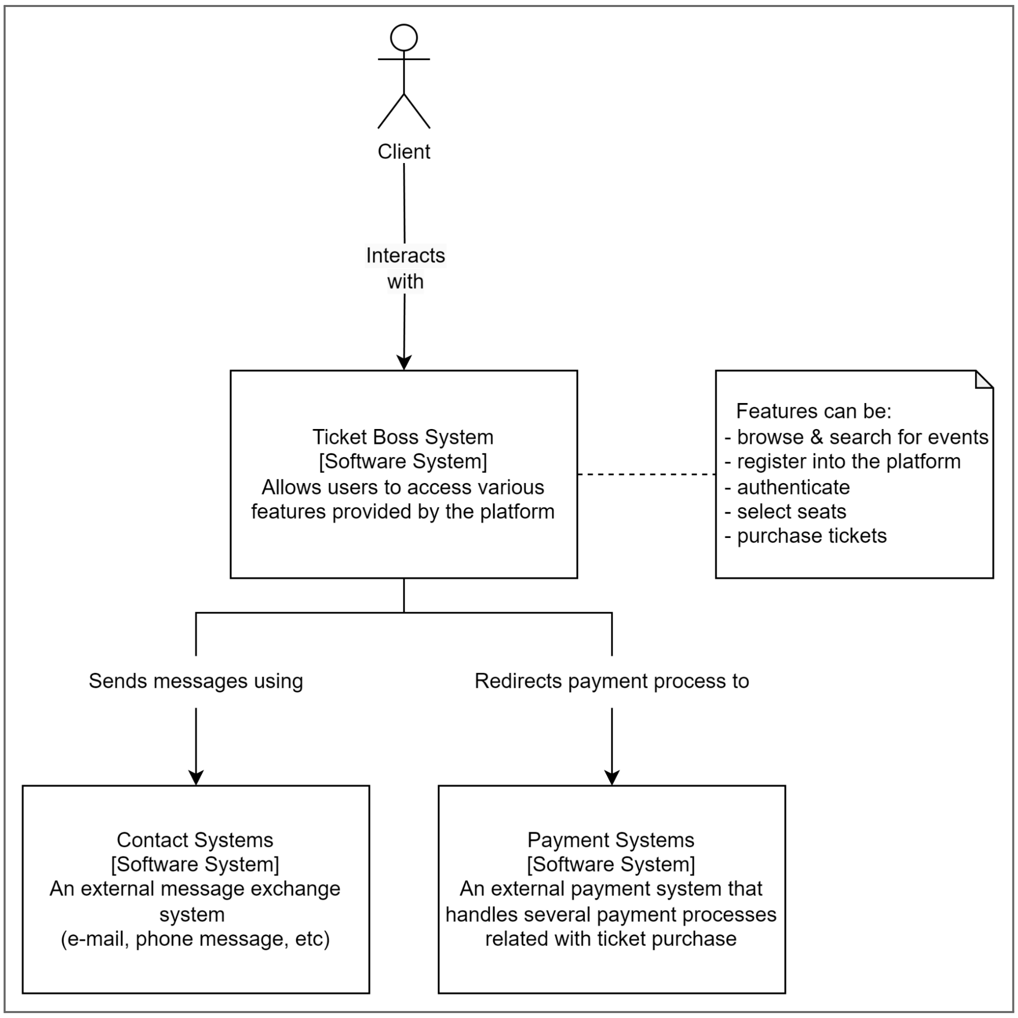

Client
Interacts with
Ticket Boss System
[Software System]
Allows users to access various features provided by the platform
Features can be:
- browse & search for events
- register into the platform
- authenticate
- select seats
- purchase tickets
Sends messages using
Redirects payment process to
Contact Systems
[Software System]
An external message exchange system
(e-mail, phone message, etc)
Payment Systems
[Software System]
An external payment system that handles several payment processes related with ticket purchase

Fig. 1: Context diagram

The Context diagram provides a broad overview of the system architecture that allows for a comprehensive understanding of the system's scope. In this diagram, the systems are represented as a box, surrounded by its users and other interacting systems. The emphasis lies on identifying actors, roles, as well as software systems, rather than delving into technical intricacies such as technologies and protocols. This zoomed-out view offers a high-level perspective suitable for communication with non-technical audiences.
For this case, we have an actor, the client, that interacts with the Ticket Boss System. It can browse events, purchase tickets for the events etc. The system is further linked to two other systems, the Contact System and the Payment System. Both systems are external to the system.

4.1.2. Class Diagram

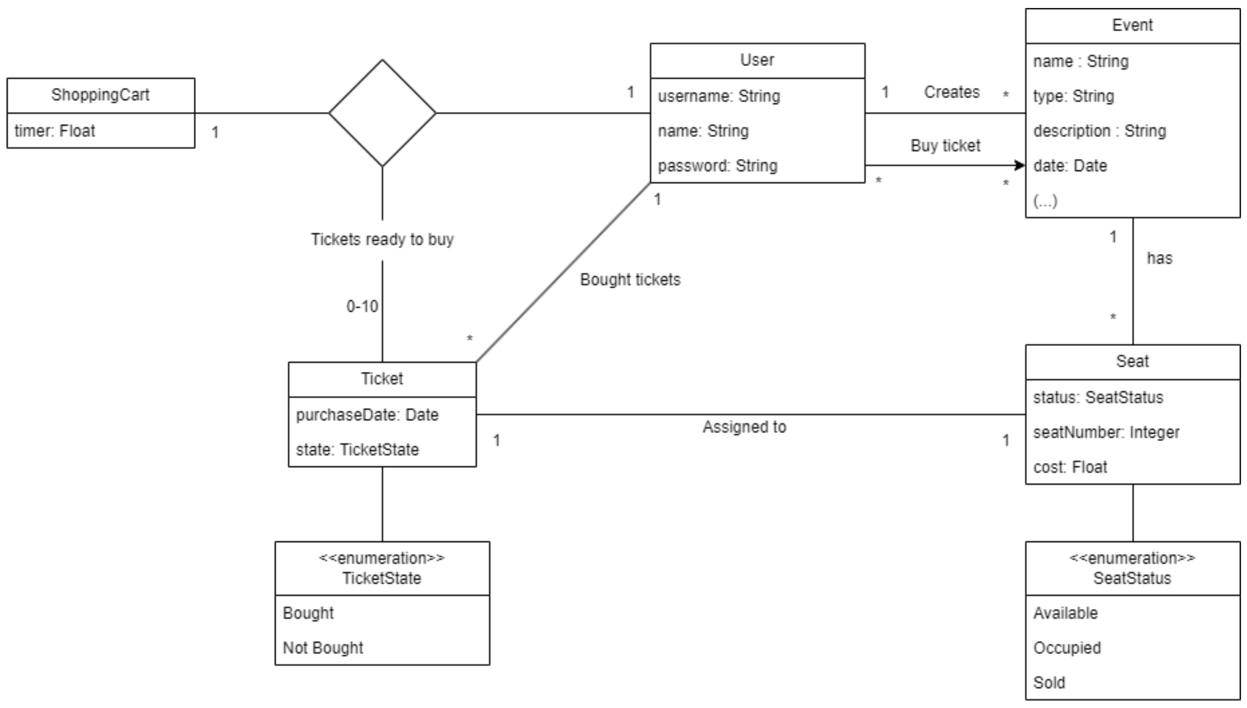


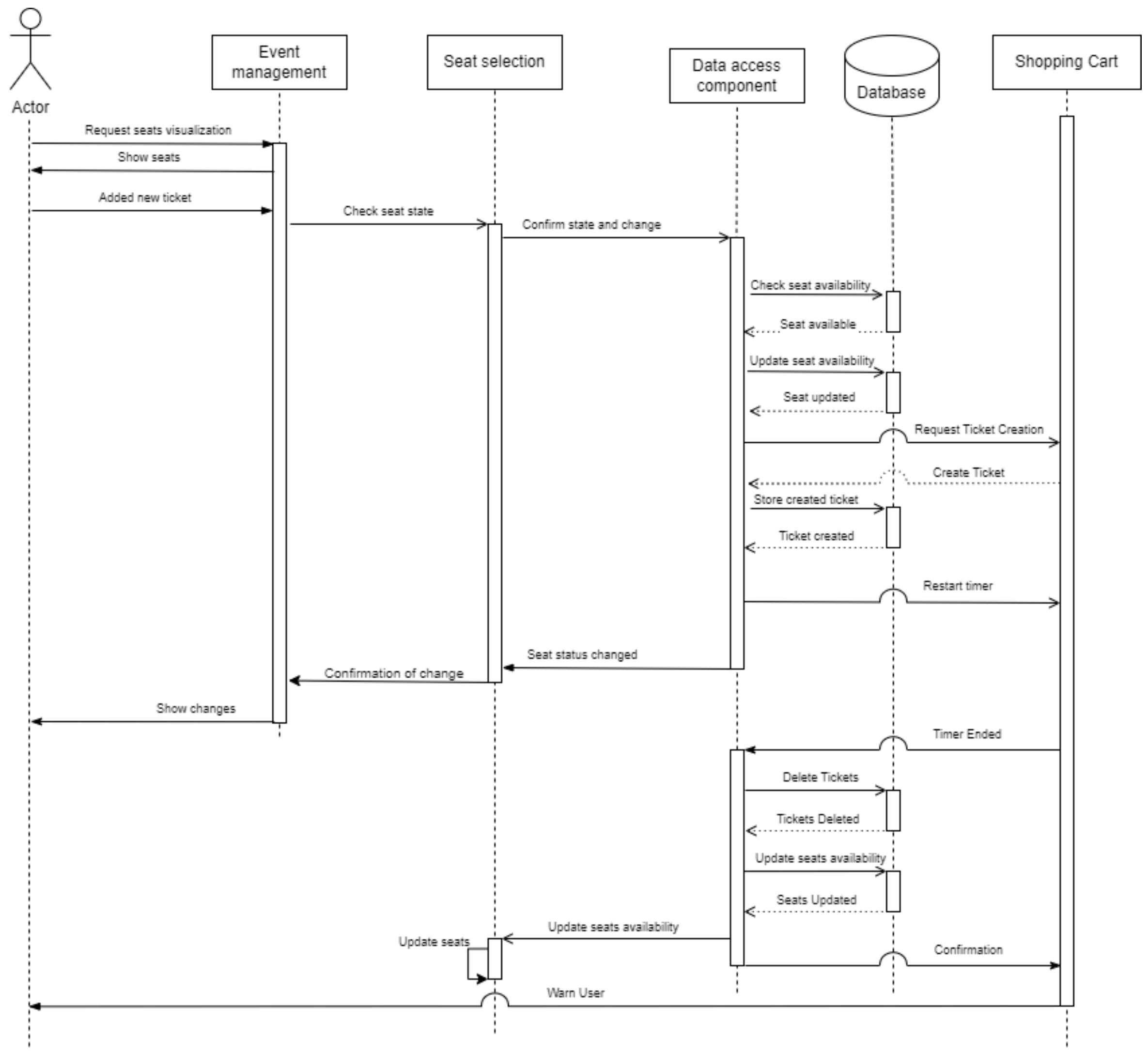
Fig. 2: Class Diagram

The Class Diagram illustrates the key classes and their relationships, providing a structural overview of the system's data model:

- **Ternary Relation between ShoppingCart, User, and Tickets**: Represents the association between a user, their shopping cart, and the tickets they intend to purchase. A user can have one shopping cart that contains multiple tickets for purchase.
- **Ticket with Seat**: Represents the association between a ticket and the seat it corresponds to for an event. Each ticket is associated with a specific seat within the venue.
- **Event with Seats**: Represents the association between an event and the seats available for the event. Each event has a set of seats where attendees can be seated.
- **User with Events (Creates)**: Represents the association between a user and the events they create or manage. Each user can be associated with multiple events that they organize or participate in.
- **User with Events (Buy tickets):** Represents the action that a user can make in an event. The user can hold multiple tickets for an event, but as said before, can only buy 10 at a time.
- **User with Tickets**: Represents the association between a user and the tickets they purchase. Each user can purchase multiple tickets for various events.

4.2. Process View

4.2.1. Sequence Diagrams



Name	Element	Description
Synchronous message symbol	⟶	This symbol is used when a sender must wait for a response to a message before it continues.
Asynchronous message symbol	⟶	Asynchronous messages don't require a response before the sender continues.
Asynchronous return message symbol	⇠	Represented by a dashed line with a lined arrowhead.
Reply message symbol	⇠	Represented by a dashed line with a lined arrowhead, these messages are replies to calls.

Fig. 3: Sequence Diagram 1

Figure 3 represents the normal flow of the actions that happen when a user decides to enter an event, after he/she is already logged into the system. As said before, entering an event means that a ticket is automatically created for the user, meaning that there must be at least one seat available for purchase. If a seat is available a ticket is created and the cart timer starts counting down. The user is only allowed to buy the tickets on his shopping cart if the timer hasn't ended. After a payment is done, an email is then sent to the user with the information.

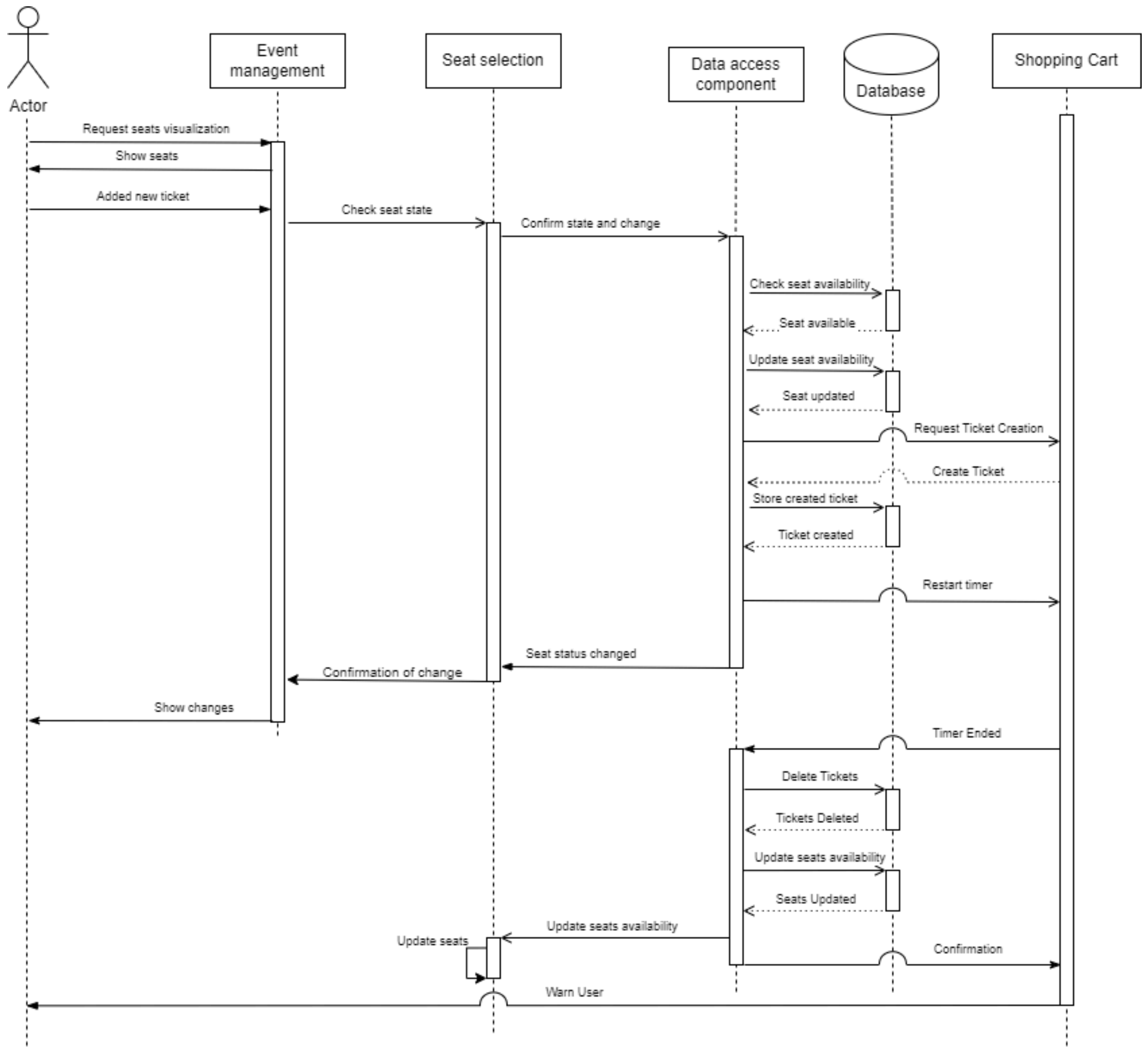
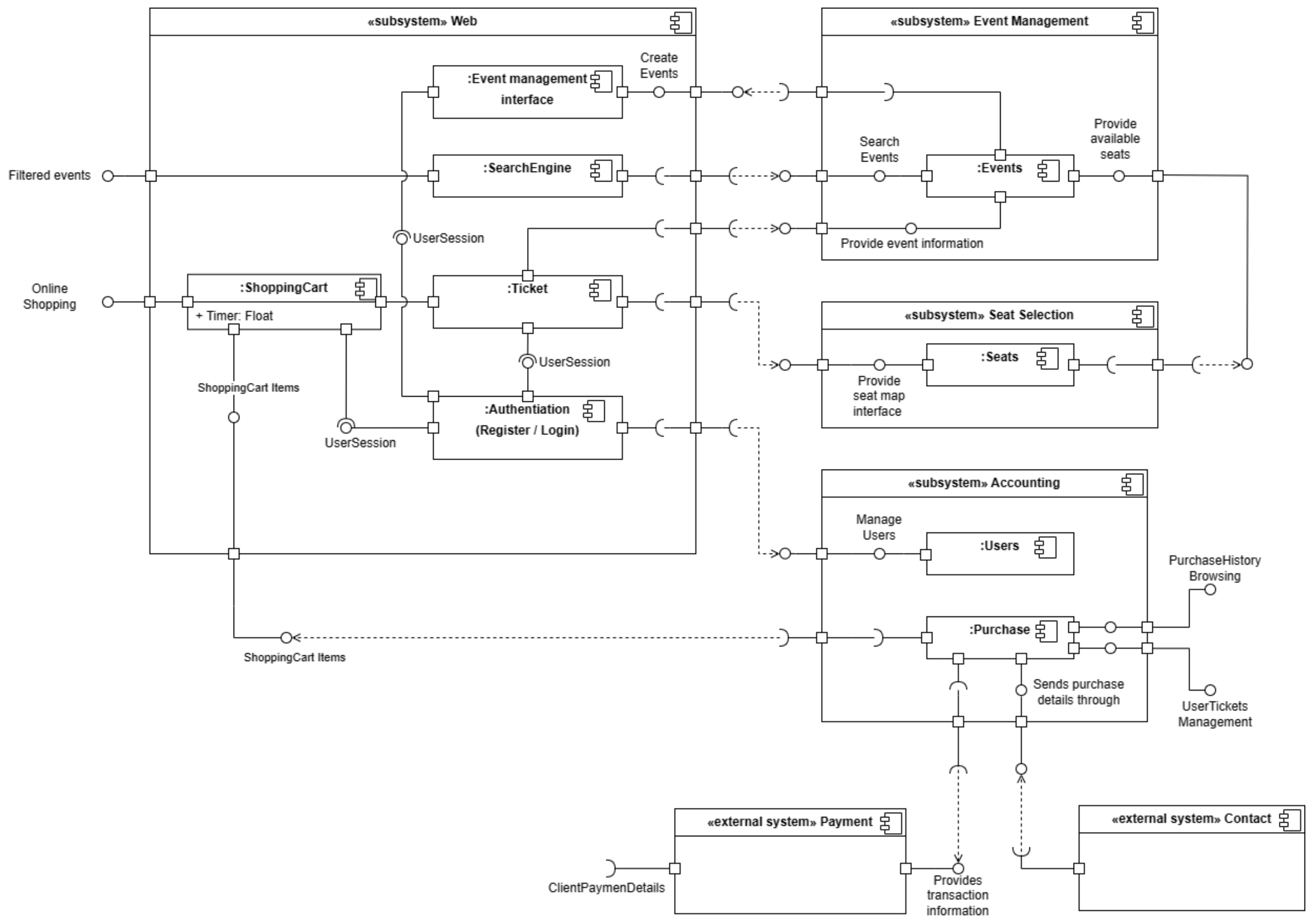


Fig. 4: Sequence diagram 2

The diagram above represents the case of when a user wants to add a new ticket but he doesn't buy the tickets in time. As shown above, when a user buys a new ticket, the shopping cart timer is restarted. If the timer runs out before the purchase is completed, the tickets are deleted and a warning message is displayed to the user.

4.3. Deployment / Implementation View

4.3.1. Component Diagram



Name	Element	Description
Port	□	A port is often used to help expose required and provided interfaces of a component.
Require Interface	——(	Required Interface symbols with only a half circle at their end (a.k.a. sockets) represent an interface that the component requires (in both cases, the interface's name is placed near the interface symbol itself).
Provided Interface	——○	Provided interface symbols with a complete circle at their end represent an interface that the component provides - this "lollipop" symbol is shorthand for a realization relationship of an interface classifier.
Dependency	------->○	A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation.

Fig. 5: Component Diagram

A component diagram provides a high-level view of a system’s architecture by illustrating the organization of system components and their dependencies. In our case, we’ll focus on six major systems: four subsystems within our application and two external systems.

Systems & Subsystems

- **Web System**: Responsible for the user-facing web application. Provides a user interface to handle user interactions with the system, including an authentication module.
- **Event Management System**: Manages event-related functionalities such as creation and modification of events and distribution of event-related information to other subsystems.
- **Seat Selection System**: Handles operations related to seats. It can be seat selection, allocation based on the user interaction or modification at the database.
- **Accounting System**: Provides user-related services for both login and registration of new users. Additionally, checkout operations are also available in this system, which are useful for buying the tickets, accessing the purchase history and managing user tickets.

External Systems

- **Payment System**: Represents the external payment gateways and integration which requires the user information to perform a purchase of tickets.
- **Contact System**: Manages communication with users (via e-mail, phone message, other), sending the purchase details, which are bought tickets and the total price paid.

4.4. Physical View

4.4.1. Deployment Diagram

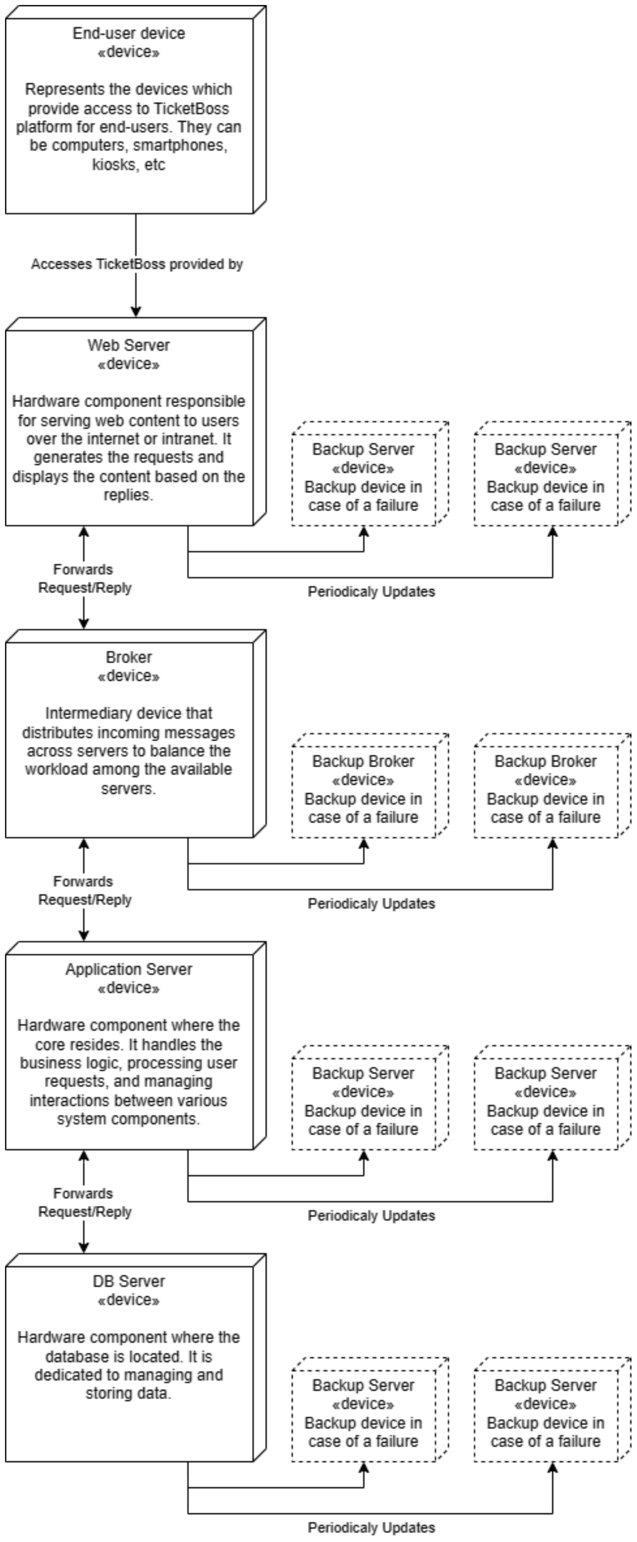
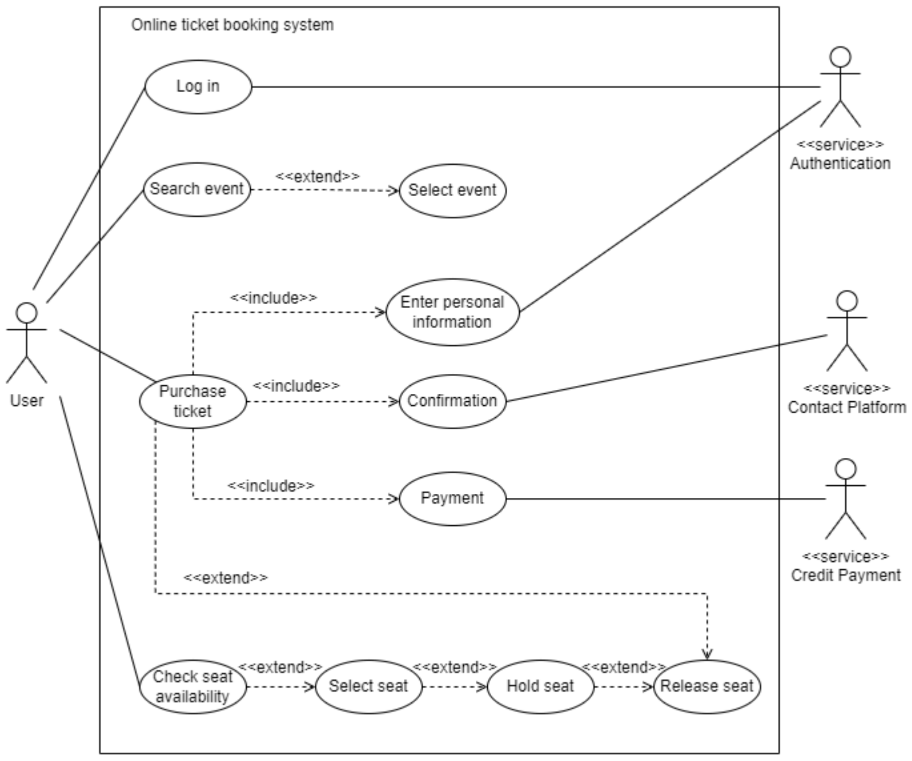

End-user device
«device»
Represents the devices which provide access to TicketBoss platform for end-users. They can be computers, smartphones, kiosks, etc
Accesses TicketBoss provided by
Web Server
«device»
Hardware component responsible for serving web content to users over the internet or intranet. It generates the requests and displays the content based on the replies.
Backup Server
«device»
Backup device in case of a failure
Backup Server
«device»
Backup device in case of a failure
Forwards Request/Reply
Periodicaly Updates
Broker
«device»
Intermediary device that distributes incoming messages across servers to balance the workload among the available servers.
Backup Broker
«device»
Backup device in case of a failure
Backup Broker
«device»
Backup device in case of a failure
Forwards Request/Reply
Periodicaly Updates
Application Server
«device»
Hardware component where the core resides. It handles the business logic, processing user requests, and managing interactions between various system components.
Backup Server
«device»
Backup device in case of a failure
Backup Server
«device»
Backup device in case of a failure
Forwards Request/Reply
Periodicaly Updates
DB Server
«device»
Hardware component where the database is located. It is dedicated to managing and storing data.
Backup Server
«device»
Backup device in case of a failure
Backup Server
«device»
Backup device in case of a failure
Periodicaly Updates

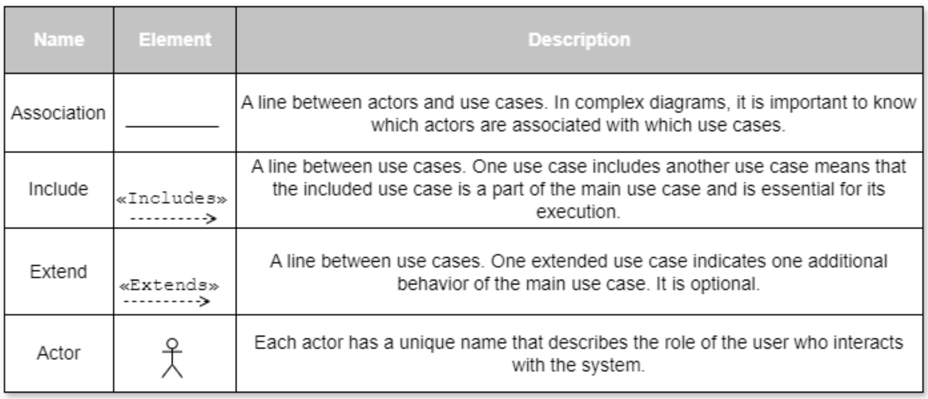
Fig. 6: Deployment Diagram

The TicketBoss platform is deployed across several hardware components illustrated in the diagram above.

4.5. Use-Case View

4.5.1. Use Case Diagram





Name	Element	Description
Association	———	A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
Include	«Includes» - - - - ->	A line between use cases. One use case includes another use case means that the included use case is a part of the main use case and is essential for its execution.
Extend	«Extends» - - - - ->	A line between use cases. One extended use case indicates one additional behavior of the main use case. It is optional.
Actor	웃	Each actor has a unique name that describes the role of the user who interacts with the system.

Fig. 7: Use case diagram

The use case view provides a detailed overview of key interactions between users and the TicketBoss system, including functionalities such as seat selection, purchase transactions, and conflict resolution. By comprehensively describing these use cases, it is possible to gain insight into how users interact with the system and can ensure that system requirements are effectively captured and addressed.

Use Case 1: User selects seats and buys them

The primary objective of this use case is to allow users to select and purchase tickets for entertainment events. Upon accessing the system, users are presented with a list of available events. After selecting an event of interest, the system displays available seats, allowing users to choose their preferred seats and initiate the purchase process. Once seats are selected, the system holds them for ten minutes to allow users to complete the transaction. Users then enter their credit card information and billing address, and upon successful payment processing, they receive a confirmation email. If the purchase is not completed within the ten-minute timeframe, held seats are released for others to buy.

Use Case 4: Two or more people select seats for the same event, with a conflict

This use case addresses situations where multiple users attempt to select seats for the same event simultaneously, leading to potential conflicts. In such scenarios, the system detects and resolves conflicts by preventing simultaneous selection of the same seats by different users. Upon detecting a conflict, the system may prompt users to choose alternative seats or notify them of the conflict, ensuring fair seat allocation. In this representation, “Select seats” is the action where conflicts may occur. "Hold Seats" is responsible for holding the selected seats for a certain period, and "Release seats" is involved in releasing held seats if a conflict arises. While conflicts themselves are not directly represented in the use case diagram, it illustrates how the system manages conflicts when they occur.

4.6. Quality attributes enforcement

4.6.1. Scalability

- Having the system broken into independent scalable components.
- Implement database sharding or replication to distribute database load and ensure high availability.
- Utilize compressing mechanisms to reduce transmission channel load and improve response times.
- Offload non-essential tasks to background processes or message queues for asynchronous processing.

4.6.2. Reliability & Availability

- Deploy redundant components, such as backup servers, databases, and network connections, to mitigate the impact of hardware failures or network outages.
- Implement failover mechanisms and automatic recovery procedures to handle failures gracefully.
- Incorporate failover mechanisms and automated recovery processes to effectively manage system failures.
- Develop and test disaster recovery plans to minimize downtime and data loss.

4.6.3. Performance

- Optimized database speed regardless of queries complexity and data volume.
- Write efficient database queries and utilize indexing for faster data retrieval.
- Distribute traffic across multiple servers to prevent overloading and ensure consistent performance.

4.6.4. Security

- Implement encryption for payment transactions to ensure data confidentiality.
- Implement access controls and encryption techniques to protect client information storage.
- Employ robust authentication mechanisms, such as multi-factor authentication, to prevent unauthorized access to user accounts.
- Conduct regular security audits and vulnerability assessments to identify and address potential weaknesses.

4.6.5. Maintainability

- The independent microservices allow for separate development, deployment and maintenance.
- Maintain detailed documentation for each microservice, aiding understanding and collaboration among teams.
- Continuously improve each microservice through refactoring
- Employ decentralized monitoring and logging solutions for each microservice to track performance metrics and troubleshoot issues independently.

5. Conclusion

In conclusion, the architecture crafted for the TicketBoss system represents a solution carefully designed to cater all the requirements of managing ticket sales for various entertainment events. Each architectural decision for TicketBoss included considerations about quality attributes such as performance, reliability, and scalability.

Furthermore, the strategic adoption of architectural patterns, such as client-server and microservices architecture, emphasizes TicketBoss's commitment to modularity, flexibility, and future scalability. These patterns not only facilitate the separation of concerns but also empower the system to adapt and evolve in response to changing user needs and technological advancements.