Arquitetura de Sistemas de Software
2023/2024

Homework #07

“TicketBoss”

Team 21

Members:
up202008569@edu.fe.up.pt - Ana Rita Carneiro
up202302747@edu.fe.up.pt - Artur Freitas
up202005097@edu.fe.up.pt - Pedro Balazeiro
up202007544@edu.fe.up.pt - Sérgio Carvalhais

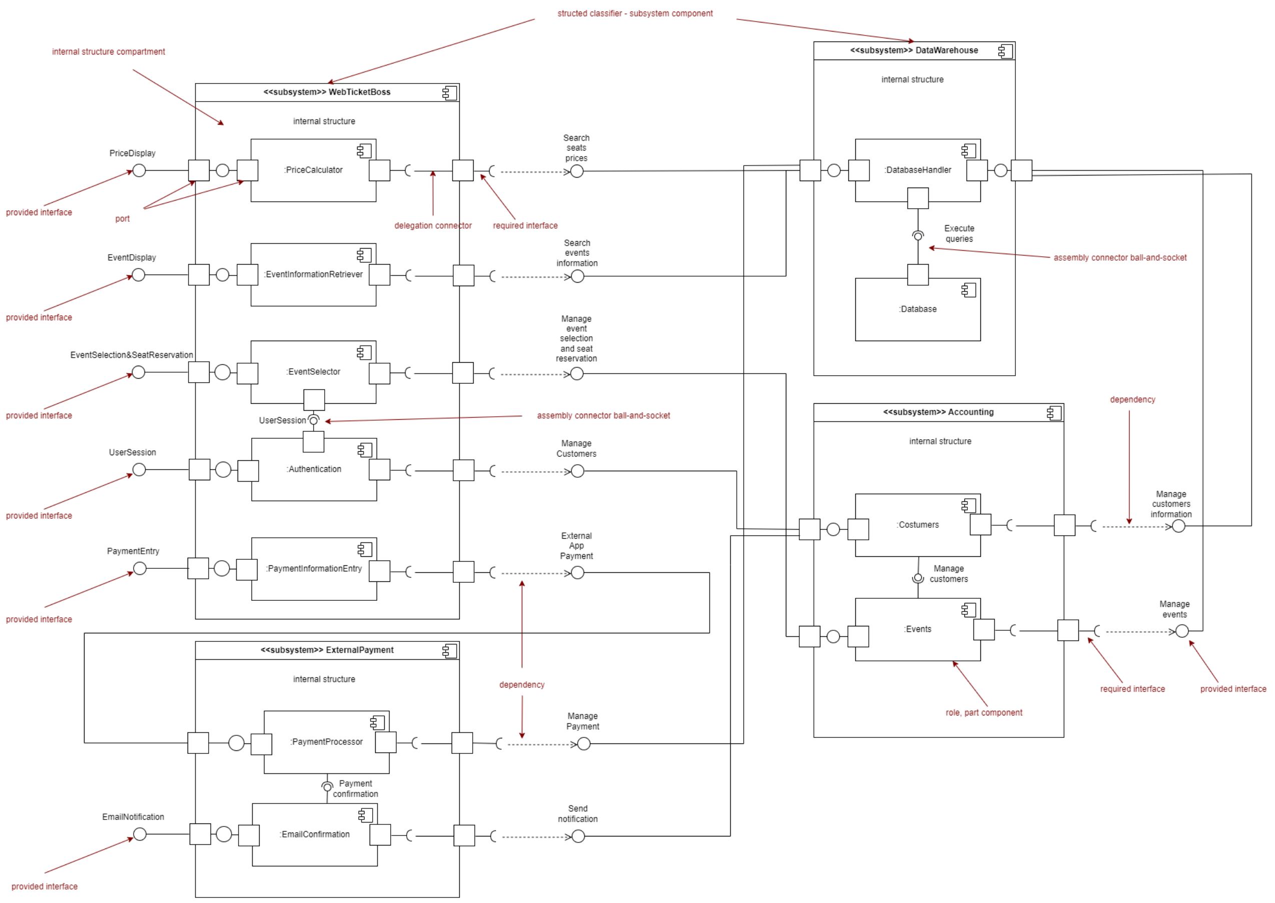
Introduction

The TicketBoss system represents a comprehensive solution for managing ticket sales across a diverse range of entertainment events. Developed to cater to the dynamic needs of both event organizers and ticket buyers, TicketBoss offers a seamless and efficient platform for browsing, selecting, and purchasing tickets for various entertainment experiences.

In response to the evolving demands of the ticketing industry, this report presents a detailed architectural design for the TicketBoss system. By examining key architectural views, including logical, process, and use-case perspectives, this report elucidates the underlying structure and functionality of TicketBoss. Additionally, important quality attribute requirements, such as portability and usability, are analyzed to ensure that the system meets the highest standards of performance and user experience.

Through collaborative design efforts and rigorous architectural considerations, the TicketBoss system aims to provide a robust, scalable, and user-friendly ticketing solution. By delving into the intricacies of its architecture and design rationale, this report lays the foundation for the successful implementation and deployment of TicketBoss in real-world scenarios.

Logical view



To ensure clarity, conciseness, and consistency in our components and connectors diagram, we've chosen to reduce the original number of components and connectors. Additionally, we've enhanced documentation to provide thorough explanations of each component and connector, making the diagram easier to understand and avoiding confusion.

In this finalized version, we've organized our system into four essential subsystems, each representing a collection of closely related components:

Accounting Subsystem (representing the physical aspect):

- Customers: Represents users of the system, responsible for interacting with TicketBoss (selecting events, seats, etc.) and managing customer information in the database.
- Events: Represents entertainment events, managing event details, seat information, and interactions with the database.

Data Warehouse Subsystem (essentially the data storage):

Initially, we included a router component in the TicketBoss subsystem to analyze incoming requests and route them to the appropriate database handler. However, in this final version, we have consolidated this functionality into the database handler itself.

- Database: Stores information related to customers and events.
- Database Handler: Interacts directly with the database, executing queries, updates, inserts, or deletes based on received instructions.

WebTicketBoss Subsystem (representing the user interface):

- PriceCalculator: Calculates and displays prices associated with selected seats (it retrieves pricing information from the database based on the selected seats' location, category, or other relevant factors and presents it to the user for review before completing the purchase).

Initially we separated this next component in two: seat availability and event display…

- EventInformationRetriever: Retrieves and presents information from the database about available entertainment events to users, including event details and seat availability.

The EventSelector component initially had various components but we compacted all in one…

- EventSelector: Manages event selection and seat reservation processes, enforcing limits on the number of seats (max. 10) per user and handling seat release after a specified duration (10 min.).
- Authentication: Manages customers and their authentications.
- PaymentEntry: Facilitates the transition to the external payment application for payment processing.

ExternalPayment Subsystem (external app for payment):

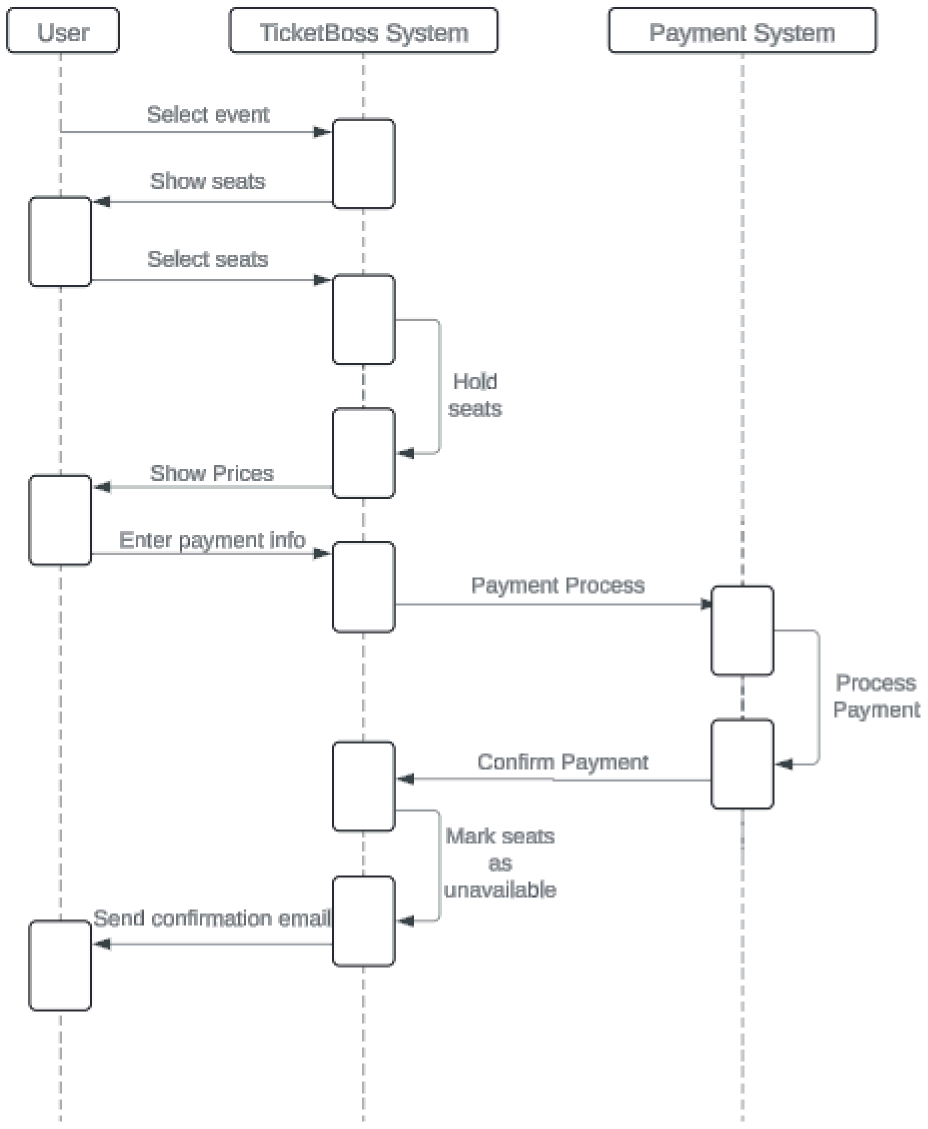
- PaymentProcessor: Handles credit card payments, integrating with external payment processing services or applications.
- EmailConfirmation: Generates and sends confirmation emails to users after successful ticket purchases, including event details and transaction summaries.

These components work together to provide a comprehensive ticket sales system, with clear responsibilities and interactions outlined for each.

Process view

Now, we're going to show how things happen step by step in TicketBoss. We made some sequence diagrams of certain situations to explain how users and the TicketBoss system work together when buying tickets, picking seats, and paying for them. These diagrams help us understand how everything happens in the system in a clear way.

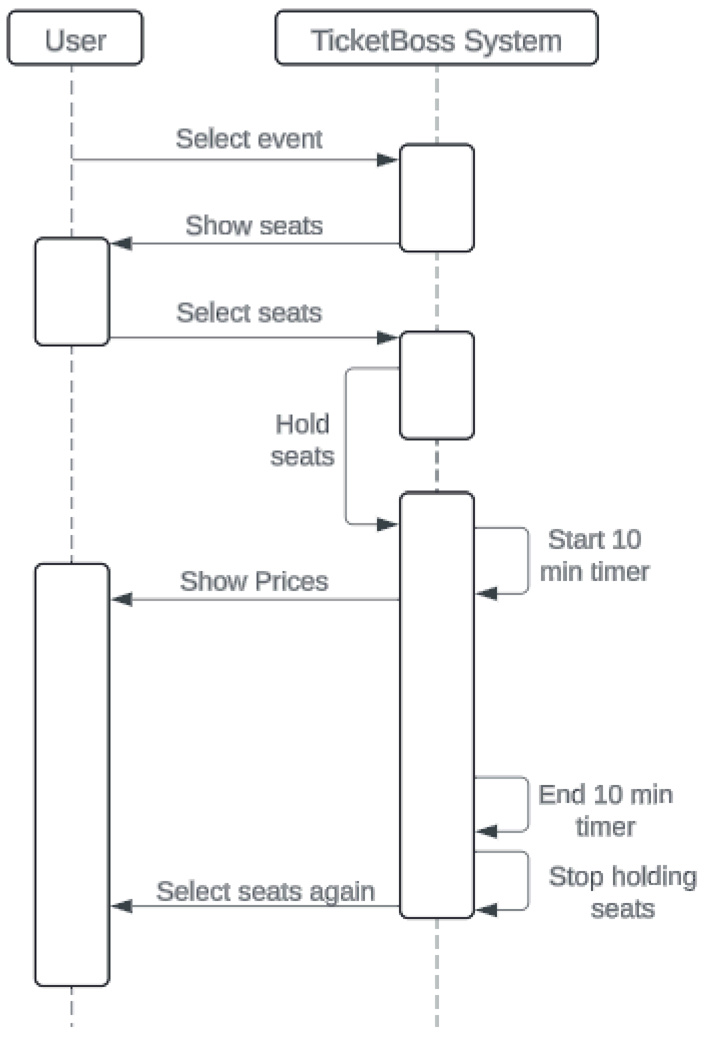
1. **User purchases tickets for an event.**



The TicketBoss user journey is depicted in this sequence diagram, which breaks down the ticket buying procedure into easy-to-follow segments. Users first browse the events that are offered and choose what they want to attend. Users select their seats and the system reserves them for ten minutes so they may think them over. Seat costs are shown throughout this period, giving users the information they need to make an informed decision before paying.

After making a decision they are happy with, consumers safely enter their payment information into the system. After that, TicketBoss communicates with outside payment processors to confirm and handle the transaction. When everything is finished successfully, the system verifies the transaction and saves the selected seats for good. Users also receive email confirmations, which serve to reassure them and validate their ticket purchases.

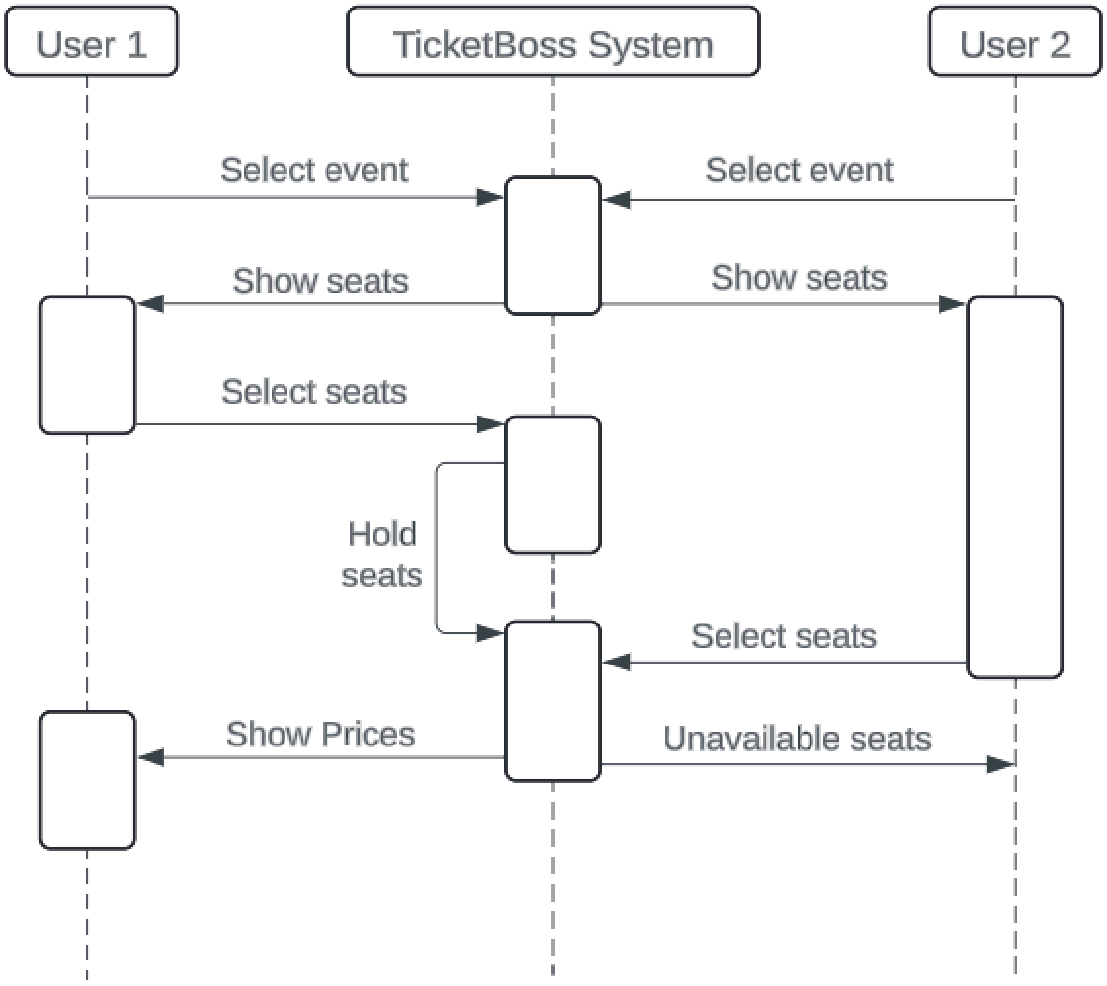
2. **User abandons seat selection.**



In this example, a user interacts with TicketBoss by choosing an event and selecting from the available seats. After seats are chosen, the system holds them while showing the user the associated costs. This gives the customer more time to decide whether or not to buy the ticket.

A 10-minute countdown timer starts at the same moment. The system will automatically release the held seats and ask the user to choose their preferred seats again if they don't finish the purchase within this window of time. This system makes sure that seats aren't kept reserved for longer than necessary, which makes it possible to allocate available inventory more effectively and improves the user experience.

3. **Two users try to select the same seats.**



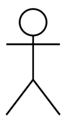
This is the case where two users log into TicketBoss at the same time and choose the same event. As a result, the same sets of available seats are shown to both users. Still, the system places a temporary hold on the first user's selected seats because of how quickly they are able to select their preferred seats.

Unaware of the ongoing reservation, the second user chooses the same seats as they see them as available, but the TicketBoss system quickly informs them that they are not. The reason for this disparity is that the seats were already put on hold for the initial user's consideration in the meantime.

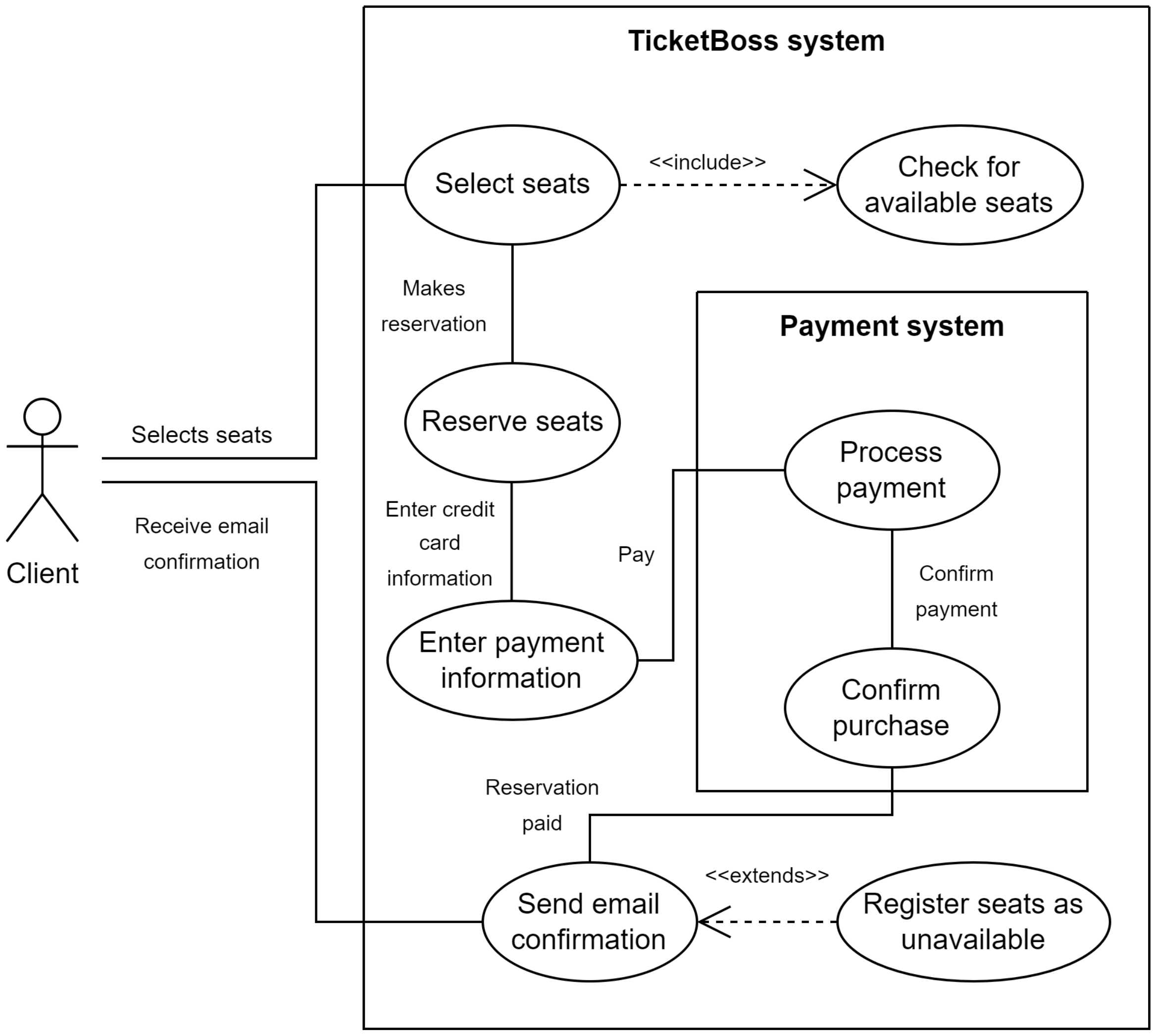
Use-case view

The use-case view of the TicketBoss system offers a comprehensive exploration of the system's functionality from the perspective of its users. Use cases provide a structured approach to defining system requirements and capturing user interactions with the system. By analyzing specific use cases, stakeholders can gain insights into how users interact with the system and the expected behavior in various scenarios.

Before writing the use cases and presenting the use-case diagrams, here is a quick description of each component within the diagrams:

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		An actor represents a role of a user that interacts with the system that you are modeling. The user can be a human user, an organization, a machine, or another external system.

Use-case 1: “User selects seats and buys them”



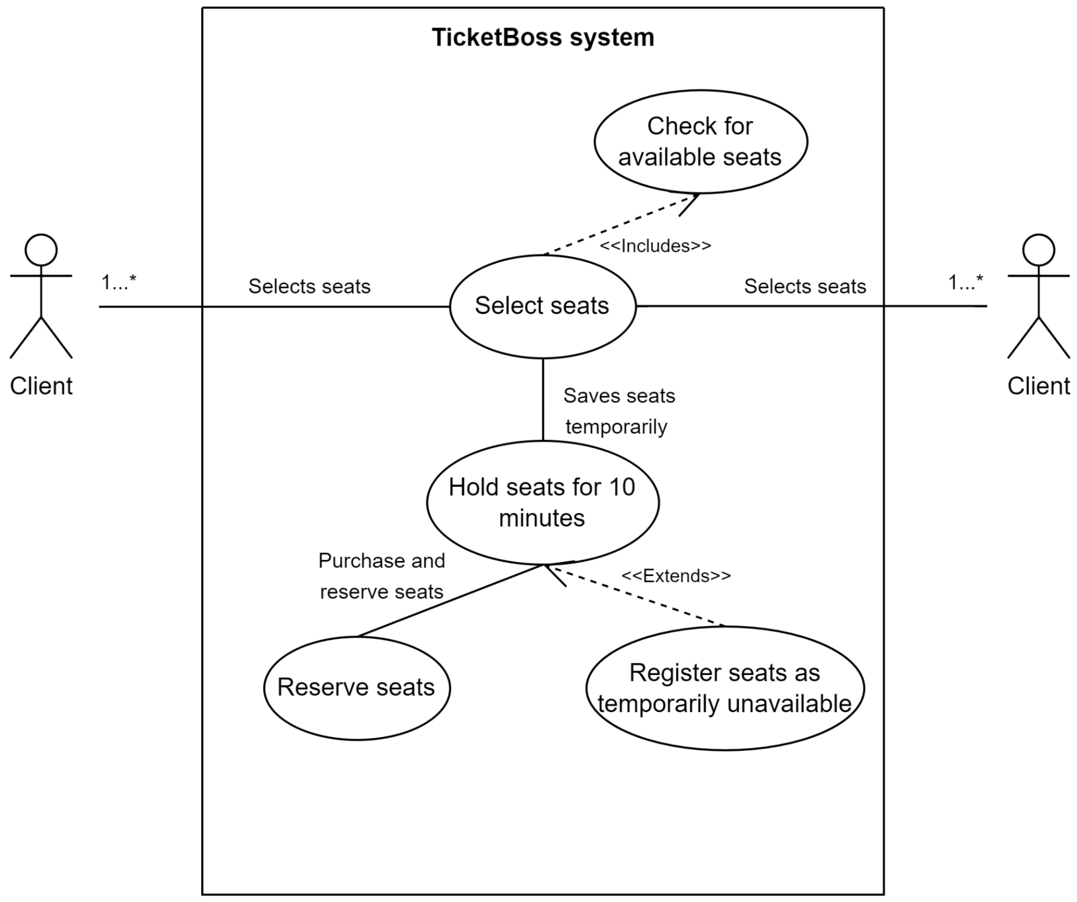
The use-case diagram for the "User selects seats and buys them" scenario in the TicketBoss system depicts the actors, use cases, and interactions involved in the ticket purchasing process. At the center of this process is the primary actor, the "User," who initiates the sequence of interactions by selecting seats for an event. This action triggers a series of events facilitated by the system's functionalities and external actors.

Once the user selects seats, the system temporarily reserves them, ensuring that they are available for purchase. Subsequently, the user is prompted to enter their payment information, including credit card details and billing address. This step initiates the "Enter Payment Information" use case, where the user provides the necessary payment details.

Following the input of payment information, the system interfaces with the external "Payment System" actor to process the payment. This interaction ensures the validation and secure processing of the user's payment details. The "Process Payment" use case handles this step, facilitating communication between the TicketBoss system and the payment processing service.

Upon successful payment processing, the system confirms the purchase to the user, marking the completion of the transaction. Additionally, the system sends an email confirmation to the user, providing them with details about their purchased tickets. These final steps are encapsulated in the "Confirm Purchase" and "Send Email Confirmation" use cases, respectively.

Use-case 2: “Two or more people select seats for the same event at the same time (within ten minutes of each other.) There is no conflict.”



This use-case diagram illustrates a scenario where multiple users engage with the TicketBoss system to select seats for the same event concurrently, within a ten-minute timeframe, without encountering conflicts. Let's delve into a detailed explanation of the diagram's components and interactions for inclusion in the report:

At the heart of the scenario lies the primary actor, the user, who initiates the seat selection process by triggering the "Select Seats" use case within the TicketBoss system. This action signifies the user's intent to reserve seats for a specific entertainment event, setting the stage for subsequent interactions.

Upon receiving the user's request, the TicketBoss system proceeds to verify the availability of the selected seats through the "Check Seat Availability" use case. This step ensures that the seats chosen by the user are currently unreserved and can be held for potential purchase.

If the selected seats are indeed available, the system moves forward with the "Hold Seats for 10 minutes" use case, temporarily reserving them for the user's session. This temporary hold ensures that the selected seats remain allocated to the user for a limited duration, typically ten minutes, allowing ample time for the user to complete the reservation process.

Crucially, the scenario emphasizes the absence of conflicts during this concurrent seat selection process. This highlights the system's capability to manage simultaneous seat selections from multiple users without overlaps or inconsistencies. The TicketBoss system effectively coordinates these interactions, ensuring that each user's seat selections are managed independently and without interference from others.

Architecture Patterns

Regarding the system we are proposing for the TicketBoss software, we can identify some architecture patterns that may be considered when building this application.

Client-Server Architecture

The users' interaction with the system may generate new data in a data storage, which is typical in a client-server architecture. The client end, being either a web app or a mobile app, communicates with the server-side application logic. The server-side logic oversees the business processes, database transactions, and also integrates other systems like payment gateways, for example.

- **Client-side:** handles user interactions like browsing events, selecting seats, and initiating the purchase process, and communicates with the server-side using APIs (RESTful APIs most likely) to fetch data and perform actions.
- **Server-side:** interacts with the database, manages user authentication and authorization, handles seat reservations, processes payments (potentially using a separate payment gateway service), and generates confirmations/tickets.

Model-View-Controller (MVC)

When considering a web application as the client-side interface for TicketBoss, the client-server architecture with Model-View-Controller (MVC) can be traced. The following is an example implementation:

- ### Model

To speak in more technical terms, it is a representation of the data layer located on the server. It stands as the interface between business logic and the storage layer, ensuring an ideal separation of concerns. The model works with application logic on the server side to:

- Receive event details (containing maps of seat locations and their prices);
- Control seat bookings (examining the capacity, making reservations);
- Conclude consumer transactions (connecting with payment gateways);
- Give data to the client.

- ### View

The user interface elements that can be held accountable for presenting the information and catching the user's response are known as front-end technologies. In a web application, these would be the HTML, CSS, and JavaScript code which are used to design the web pages. The View components may encompass various features, such as:

- Event listings with their full particulars and seat selection alternatives;
- User account managing sections;
- Shopping cart, as well as the checkout forms;
- Verification pages and tickets to view.

- ### Controller

An intermediary plays the role of being between the View and the Model; it takes user input from the View components and works with the Model to perform necessary actions. The Controller in a web application can be developed with a web framework that manages routing requests to the right functions. The Controller would be responsible for:

- Accepting users' requests for event browsing, seat selection, and initiate purchases;
- Calling the Model's functions to fetch data, manage reservations, and process purchases;
- Updating View with read values or processing results (like showing seat availability or confirmations);

Database

The system may utilize a relational database to store information about:

- **Events:** The particularities of an event such as its name, date, time, venue, seat chart, pricing information, and so forth.
- **Seats:** Details about every seat like the seat identification number (seat ID), section where it's situated, row number of that seat, and price category for that seat.

- **Users:** User accounts contain such information as name, email, address, and billing details.
- **Reservations:** A table to manage seat reservations, including information such as seat ID, user ID, reservation timestamp, and expiration time.
- **Purchases:** Purchase info including all relevant details such as the event ID, seat IDs, user ID, price paid, confirmation number, and any other necessary information about the purchase.

Reservation System

The reservation system is one of the features of the TicketBoss application that prevents a seat from being sold to another user during a predefined period (10 minutes) while the current user is finalizing the purchase. This could be implemented in a few ways, such as:

- **Data-based approach:** As presented in the previous topic, a solution to this problem would be the creation of a table in the database where information about reservations is stored, such as IDs of seats, timestamps when they were reserved, and user IDs.
- **Distributed caching:** A distributed cache can also be used to store reservation information with expiration times. Seats are released from the cache when the reservation expires.

Asynchronous communication

The best way of implementing email confirmation of purchases is by introducing asynchronous communication. Since the main transaction flow and sending of the email are separated, it ensures that the user does not experience any disruptions and minimizes possible delays caused by email delivery.

To manage seat reservations and confirmations, the system would possibly benefit from using **event-driven architecture**. If the user selects a seat, then one of the events that might occur is the starting of the reservation timer. In case there is an update in seat availability due to either the expiration of the reservation or the acceptance of a purchase, then one more event could be triggered for that and other notifications.

Software Aspects (Quality Attributes)

Following the exercise that was proposed in the practical class, we considered it important for the TicketBoss software to guarantee **security**, **availability**, **usability** and **maintainability**.

Security

- **What does it mean?**

  In the context of this TicketsBoss system, security refers to the protection of customers' personal and financial information, as well as the integrity of the ticketing system from unauthorized access, data breaches, and fraudulent activities. This includes aspects like data protection, authentication, secure transactions authentication, secure infrastructure, fraud prevention and compliance.

- **Acceptance level?**

  The acceptance level of security refers to how well this Tickets system meets these security requirements and how customers perceive the security measures implemented by the company. It's important for this Tickets system to achieve a high acceptance level of security because of customer trust, protecting reputation, legal and regulatory compliance and business.

- **How important?**

  In summary, this is a very important and fundamental quality attribute because it influences the system's success and sustainability as a trusted platform. By prioritizing this the users data can be protected, fraud prevention is done, and trust and confidence is ensured in their services.

Availability

- **What does it mean?**

  The availability of the system, as a term, indicates the ability of the system to stay on and accessible to users. This measure is usually expressed as a percentage of time that the system is available for use. It combines elements such as system uptime, reliability, fault tolerance, and responsiveness towards user requests without slowing down performance.

  Availability is essential in TicketBoss to ensure that users can access it at any time when they wish to buy tickets for their favorite entertainment events. Considering that ticket sales are bound to deadlines and visitors would certainly want to secure good places at famous shows, even a small delay or unavailability of the system could lead to serious revenue loss and customer discontent

- **Acceptance level?**

  TicketBoss system should deliver a high level of availability, allowing its users to access the system reliably and complete ticket purchases without interruption.

  To achieve this availability, the software should implement redundancy at many levels (servers, databases, network connections) to reduce failures impact. Also load balancing would help to distribute the traffic for the multiple servers, avoiding overloading a single component. Some fault tolerance measures, like automatic failover, ensure seamless even when facing failures. Additionally, cashing data that is commonly accessed could reduce the strain on backend servers and enhance the system´s performance.

- **How important?**

  As it affects the user's experience, ability to make purchases in real time, revenue loss, competition, among others, the availability of the application is a key element that TicketBoss must pay close attention to. The users are always demanding that they be able to access their purchase transactions without fail at any time, and therefore downtime could lead to loss of sales as well as generating negative customer sentiment. Moreover, in a market with competitors, continuous availability is crucial for attracting more users and keeping up its good name. High availability through strong architecture and infrastructure is essential for TicketBoss as it helps them fulfill their goal effectively regarding client expectations and strategic planning.

Usability

- **What does it mean?**

  Usability is a quality attribute that refers to how easy a system is to learn, how efficient it is to use and if it provides a satisfying user experience. It's a quality attribute that is very related to the user interface and evaluates if it's consistent in terms of design, intuitive to the user, has clear instructions and if it's responsive to the user interactions.

- **Acceptance level?**

  The acceptable level depends a lot on the type of user we are targeting. There are two main types of users we can identify, which are the non-technical users and the technical users. For the non-technical, maybe usability is a key attribute because they value easy and intuitive interactions with the software. On the other hand, for the technical users, the acceptance level could be different because usability requirements may be more nuanced, focusing on efficiency and workflow optimization.

- **How important?**

  Usability is very important and significant to TicketBoss because the user satisfaction depends on it, meaning that the majority of users tend to engage more with user-friendly, clear and intuitive interfaces, leading to positive perceptions from their side and increasing their satisfaction. Also, by having easy to learn software, the user's training time is reduced (their learning curve reduces), minimizing the need for lots of documentation and clarification. Finally, having an intuitive design and clear feedback mechanisms is a key aspect to minimize and prevent the errors made by the users, also mitigating their frustration while using the software.

Maintainability

- **What does it mean?**

  Maintainability is the term which relates to how the system can be altered and updated without significant efforts and risk.

  The fact that TicketBoss is maintainable is of great importance to make the ticketing process flexible to accommodate new features, bug fixes, and requirements without any distractions to the purchasers.

- **Acceptance Level?**

  The acceptance level refers to the extent to which the stakeholders (clients, users and developers) believe the system is in accordance with maintainability standards.

  High acceptance level will provide the stakeholders with an increased level of satisfaction with the system's maintainability, ensuring the decrease of resistance to the maintenance activities and increase the system's sustainability.

- **How Important?**

  This assesses how maintainability can have a significant impact on the overall success of the TicketBoss project.

  Considering the challenging nature of entertainment events and the growing demand of users, the maintainability is crucial for the TicketBoss to keep up with the pace of market, the market needs, and the efficiency in the long run.