António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

As explained, TicketBoss is a system for managing ticket sales for a variety of entertainment events. As a result, and in accordance with its requirements, we've decided to use a Layered Architecture style to organize the system into several layers that create a separation of concerns.

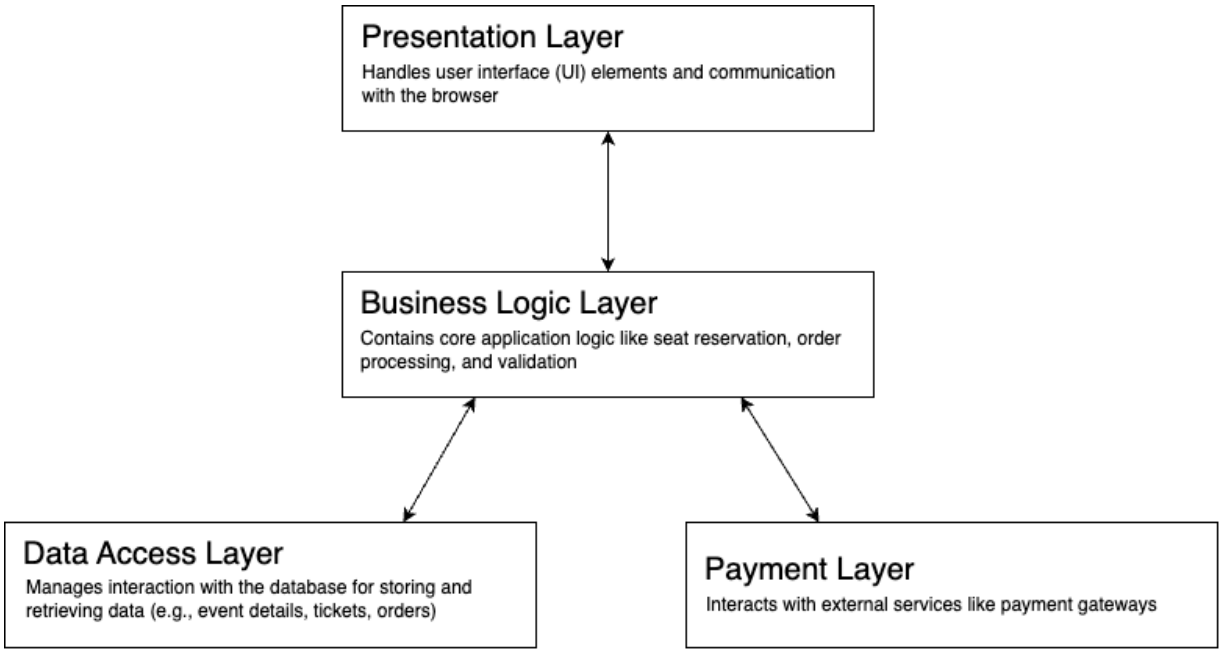

Fig. 1 – Diagram of the Layers of TicketBoss

However, this diagram does not completely describe the system from an architectural aspect. While the layered design provides a useful starting point, an expanded view requires consideration of additional architectural features, which will be discussed in the next sections of this paper.

Logical View

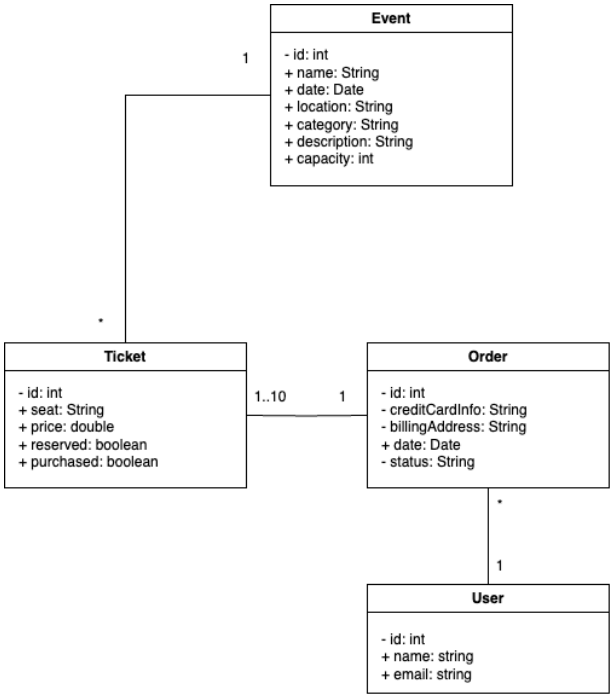

Fig. 2 – TicketBoss's UML Diagram

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Considering the specifications given for this application, we've designed an UML diagram which has the following classes:

- **User:** Stores information about the users who have bought tickets for events through the TicketBoss platform. Since the system doesn't have any authentication logic, as every user is treated as a guest, we'll only store an ID number, their name and email address.
- **Order:** Stores information about an Order a User has made, including its ID, credit card information (of the User), billing address, date the order was made and status
- **Ticket:** Information about a Ticket, including its seat location (row and section within the venue), cost, whether it's reserved or not and whether it's been purchased or not.
- **Event:** Stores information about the event, like its ID, name, scheduled date and time, location, category (if it's a sports event or a concert, for example), description and maximum number of attendees.

In the diagram is also possible to highlight three relationships:

- **One Event can have many Tickets.**
- **One User can have many Orders.**
- **An Order can have up to 10 tickets associated with it.**

These relationships showcase the business rules that are associated with the system, such as, for example, users being able to hold multiple orders, and each order being linked to a single event.

Process View

With the main objective of showcasing the interaction between the components of the system, we've made two sequence diagrams that highlight 3 main characteristics of our design:

- the normal flow of buying a ticket for an event (diagram #1)
- what happens when the time for reserving a seat expires (diagram #1)
- what happens when two or more users try to reserve the same seat, simultaneously (diagram #2)

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Diagram #1

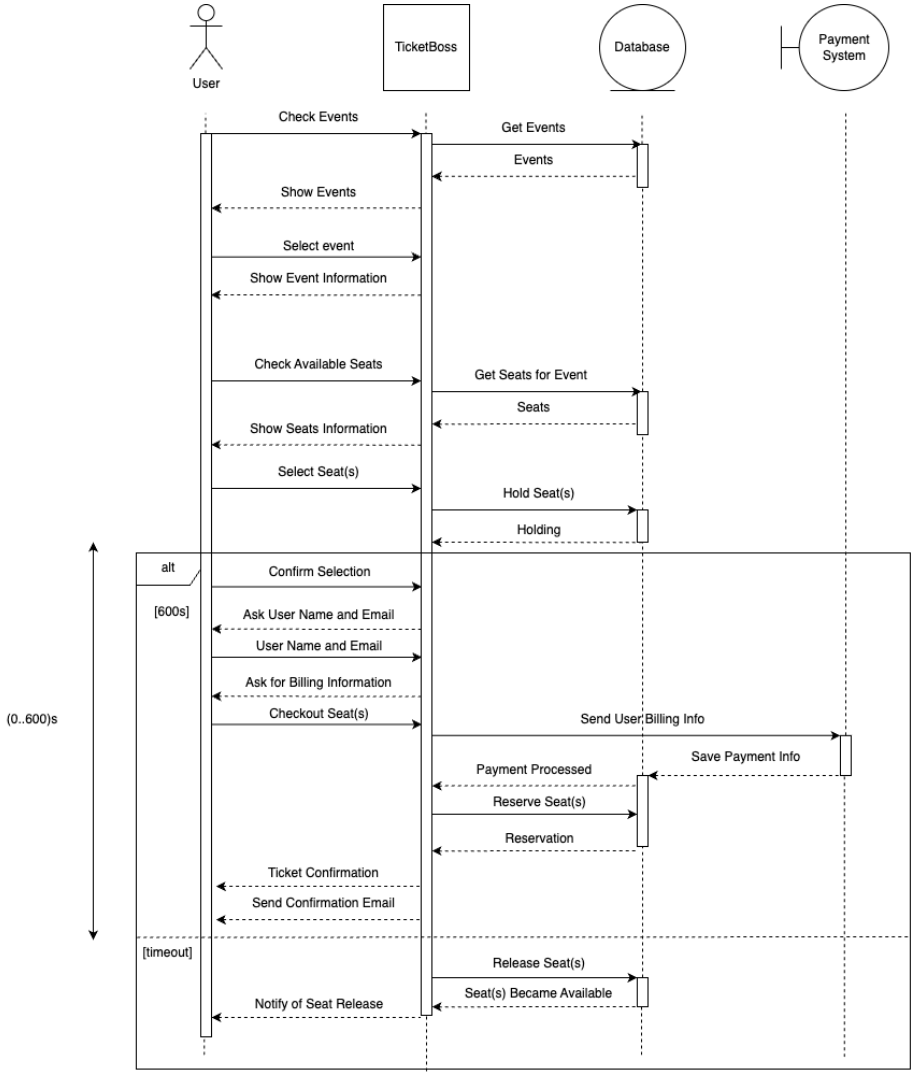

Fig. 3 – TicketBoss's Sequence Diagram #1

As it was said before, this sequence diagram visualizes the flow of buying a seat for an event. More specifically, it showcases the interaction between an user, the TicketBoss system, the database and an external payment system. The diagram is made up of the following steps (presented chronologically):

1. **User Interaction:** The process begins with the User wanting to check the available events at TicketBoss.
2. **Retrieving Events:** TicketBoss requests event data from the Database with a "Get Events" message. Consequently, the Database responds by returning the events and their corresponding information.
3. **Displaying Events:** TicketBoss then displays the events to the User, who selects a certain event to view more information about it.
4. **Checking Seats:** Once the event is selected, TicketBoss requests seat availability from the Database with a "Get Seats for Event" message. The Database provides information about the availability of the seats.

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

5. **Selecting Seats:** The available seats are displayed to the User, who then selects up to 10 seats. After selecting them, TicketBoss processes this selection by sending a "Hold Seat(s)" message to the Database, which holds the seats for a set period (up to 10 minutes)
6. **User Confirmation:** After selecting the seats, the User confirms their selection and provides their name and email, when prompted by TicketBoss. The User is then asked for billing information, to proceed with the purchase.
7. **Processing Payment:** With the User's billing information, TicketBoss then sends a "Send User/Billing Info" message to the external Payment System, to process the payment.
8. **Payment Confirmation:** Once the payment is successfully processed, the Payment System sends a "Payment Processed" message back to TicketBoss. Consequently, the system sends a "Reserve Seat(s)" message to the Database, prompting it to update the seat status to sold.
9. **Ticket Delivery:** A confirmation of the ticket purchase is sent to the User, followed by TicketBoss sending a confirmation email, to the address the User gave previously, through the system's email service.
10. **Timeout (Alternate Path):** After the 10 minutes of ticket holding have passed, where the User doesn't complete the purchase, a timeout occurs. In this scenario, the Database releases the seats with a "Release Seat(s)" message, making them available again. Then, TicketBoss notifies the User of the seat release due to timeout.

Diagram #2

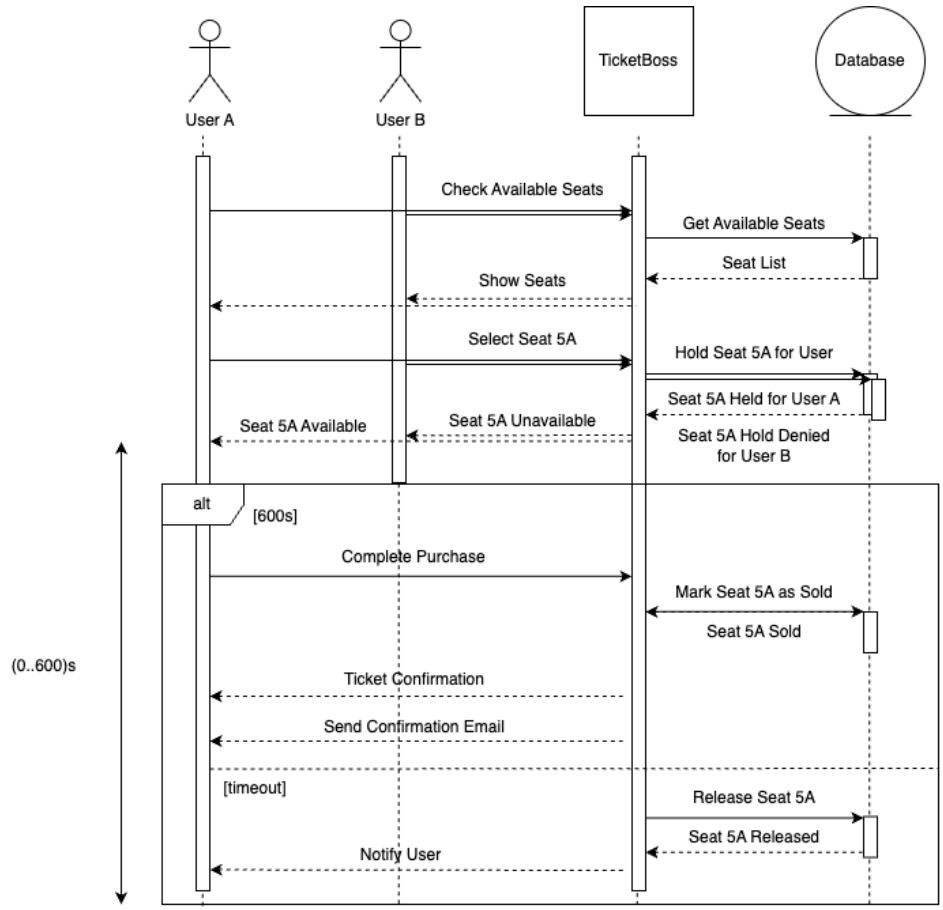

Fig. 4 – TicketBoss's Sequence Diagram #2

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Regarding this sequence diagram, its function is to illustrate how the system deals with concurrency, during the seat selection process. For this system, we decided to apply the "First Write Wins" approach, where the system grants the seat to the User who completed their reservation process first.

The diagram goes as follows:

- Both User A and User B want to reserve Seat 5A, at the same time.
- TicketBoss processes these requests and communicates with the Database.
- The Database locks the seat for the first request that arrives and is processed (in this case, for User A), based on the timestamp in which the request to reserve the seat is received.
- User A's reservation is accepted and written to the Database first.
- When User B's reservation is processed, they'll receive a message denying their request, since the Database detected that the seat in questions is already held by User A.

The key point of this approach is the assumption that the processing of the requests, by the Database, is sequential, as it heavily relies on the accuracy of system clocks. Even if two reservation attempts seem to be happening at the same time to the users, the system serializes them and processes them by the order in which they arrived at the Database. Although this method may not be perceived as the fairest by the end users, especially in high-demand situations where many people are trying to buy tickets for an event, the simplicity and ease of implementation of this approach considerably outweighs the presented disadvantages.

Use Case View

Use Case 1: User selects seats and buys them

Title: Purchase Seats

Description: This use case describes the scenario where a user selects seats for an event and successfully completes the purchase.

Actors: User and Third-Party Payment System

Preconditions:

- The seats selected must be available at the time of selection.

Postconditions:

- The seats are marked as sold.
- The user receives a confirmation of the purchase, via e-mail.

Main Path:

- The user browses the event details.
- The user selects available seats (máx. 10).
- The user proceeds to checkout.
- The system calculates the total cost, including any taxes and fees.

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

- The user enters its e-mail, credit card information and billing address.
- The payment system processes the payment.
- The system confirms the purchase and sends a confirmation to the user, via email.
- The system updates the seat status to sell.

Alternate Path: If the payment fails, the user is prompted to re-enter its credit card information.

Exception Path: If the seats become unavailable while selecting, due to the 10min reserve time, the system notifies the user and redirects him to the selection of available seats screen, now updated.

Priority: High -> This use case is essential for the core functionality of the Ticketboss system, and it directly impacts revenue generation and customer satisfaction.

Frequency of Use: High -> This is one of the most common interactions within the system, especially when there are popular events ocurring.

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

Title: Concurrent Seat Selection with Conflict

Description: This use case describes the scenario where two or more users attempt to select the same seats simultaneously, leading to a conflict.

Actors: User1, User2 (or more users)

Preconditions:

- The seats are initially available.

Postconditions:

- Only one user successfully reserves the seats, receiving its confirmation, via email.
- The other users are notified of the unavailability and redirected to the selection of available seats screen updated.

Main Path:

- User1 selects seats.
- Simultaneously, User2, and potentially others, selects the same seats.
- Both users attempt to finalize their selection.
- The system locks the seats temporarily as the first transaction is processed.
- The system completes the first successful transaction and updates the seat status.
- The system notifies other users that the seats are no longer available.
- Unsuccessful users take the Exception Path

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Alternate Path: Any user can cancel their selection before finalizing, which releases the seats back to availability.

Exception Path: If a system error occurs during processing, all users are notified, rolling back the transaction and redirecting the users to the browse event details screen.

Priority: Medium to High -> While this use case may not occur as frequently as individual purchases, its importance lies in maintaining a good user experience and ensuring fair access to tickets.

Frequency of Use: Medium -> Conflicts in seat selection are less frequent but are likely to occur during the opening sale period of highly anticipated events, where many users are trying to purchase the best available seats simultaneously.

Note: Priority and Frequency of Use were classified in a five-tier classification: Low, Low to Medium, Medium, Medium to High and High.

Physical View

For the case of TicketBoss, a simple client-server model can easily show off how the users interact with the system. With this in consideration, we've designed the following physical diagram:

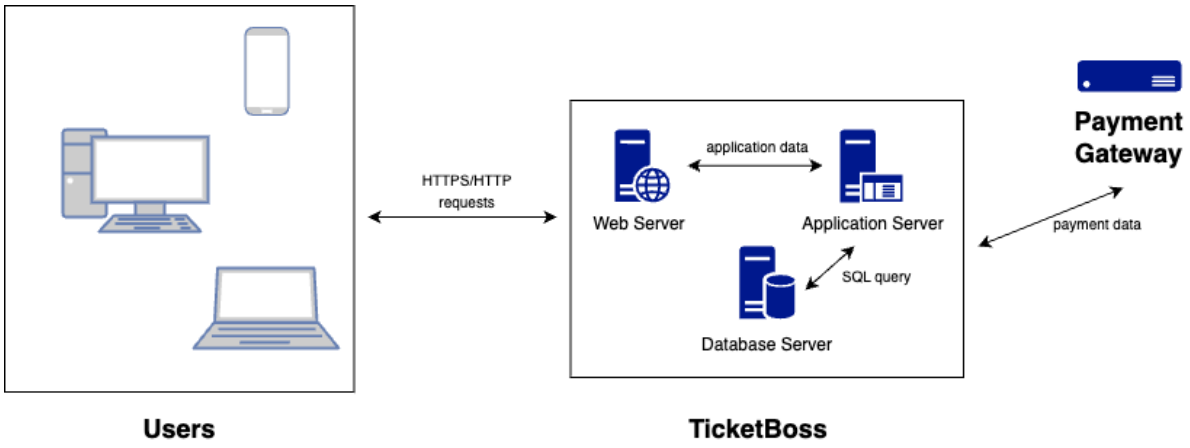

Fig. 5 – TicketBoss's Physical Diagram

The main components of this diagram are:

- **Users:** To access the platform, the users can access it via a web browser while using their device of choice like a smartphone or a desktop computer
- **TicketBoss system:** A self-contained system that holds the three central components:
  - *Web Server:* Where the TicketBoss application is hosted. It handles HTTPS/HTTP requests from the users, allowing them to interact with the system through various web pages.

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

  - *Application Server:* Runs TicketBoss's backend logic. It processes business rules, seat reservations, interactions with the database and communications with other external systems (like the payment gateway).
  - *Database Server:* Stores all persistent information like user details, event information, ticket data and order details.
- **Payment Gateway:** An external system that processes payments. It handles sensitive payment information and transactions.

Quality Attribute Requirements

Quality Attribute	What does it mean?	Acceptable level?	How Important?
Security	Consists of the ability to safeguard critical information, data and applications from unauthorized entities.	Endpoints, data and other critical areas are not available to the public and only accessible to authorized users. Also, ensure that data encryption and authentication are present in the application, to prevent malicious attacks	5 This attribute is crucial since it prevents data leaks and unauthorized accesses, which can lead to the diminishing of user trust
Usability	Related to the ease of use. The flow to buy a ticket must be simple and clear. The user should also be aware of the 10 minutes rule and other important info.	Users can complete the task of buying a ticket >80% of the time, when using the system for the first time. User feedback is also very important (like notifications and confirmation emails)	4 If the system is not understood by the user, then the user can give up on trying to use it or end up doing non intended actions
Portability	Being able to use the system in different types of devices (laptop, mobile phone, etc.)	Availability for the most common browsers (Chrome, Safari, etc.) and most common mobile OS's (iOS, Android)	3 If there is support for at least Android and iOS (if it is an app) or the most common browsers, it is already enough
Scalability	Refers to the ability to handle an increase of users, transactions and data, without any degradation in performance.	The system should be available 99.9% of the time, and all the critical failures should be resolved within one hour.	4 This quality attribute is what ensures the good reputation and high number of sales in an online ticketing platform. If it wasn't considered as important as it is, the

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

			system couldn't be maintained, and the user trust would be lost.
Reliability	The ability of the system to behave as expected and function under the maximum possible load	The application should not only be always available, but also recover quickly and efficiently from failures.	5 This attribute ensures that frequent scenarios, where many people are trying to buy a ticket for a popular event and the platform/app goes down, doesn't happen.

Table 1 - TicketBoss's Quality Attributes

"How Important" Scale: 1 - Not important, 2 - Slightly important, 3 - Moderately Important, 4 - Very Important, 5 - Extremely Important

Architectural Patterns

The architectural patterns found in TicketBoss's architecture are:

- **Layered Architecture -** Suitable for the system in question, due to the clear separation between the Presentation Layer, the Business Logic Layer, Data Access Layer and the Service Layer
- **Client-Server Architecture -** Users will be interacting with the server to book tickets.
- **Model-View-Controller (MVC) Pattern -** It's used to separate the user interface, present in the Presentation layer, from the business logic layer and the data management.
- **Repository Pattern -** Can be applied in the Data Access Layer of the system, in order to abstract the logic that retrieves data from the database.