Homework 7 - TicketBoss

T25 - Software Systems Architecture - M.EIC010

Rita Kiss - Laura Lumijärvi - Yana Peycheva - Juan José Osorio - Amanda Oliveira - Tiago Marques

Introduction

Our task was to design a possible architecture for TicketBoss, a ticket-sales service. TicketBoss is a system designed to handle ticket sales for various entertainment events. In this report, we present the Logical View and the Process View of the system through UML diagrams. Furthermore, we describe two use cases in detail and discuss the important quality attributes of TicketBoss, along with our plan to address them.

Logical View: Components and Connectors Diagram

The logical view consists of four subsystem components: the "User Layer," "Application Layer," "Data Access Layer," and the "External Services Layer." The User Layer encompasses four role components: Seat Selection, Event Viewer, Checkout, and Search Engine. Within the Application Layer, Seat Reservation, Event Reservation, and Execute Order handle the system's core functionalities. Seat Reservation forwards seat selections to the Event Component, which in turn sends event reservations to the Execute Order Component. The Data Access Layer comprises Data Retrieval and Data Storage, interconnected by an assembly connector to manage data operations. Lastly, the External Services Layer includes Payment Processing and Email Confirmation components.

In the User Layer, the Seat Selection component receives available seat information from Seat Reservation, while the Event Viewer retrieves available events from the Event Reservation component. The Checkout component forwards orders to the Execute Order Component within the Application Layer.

Within the Application Layer, both Seat Reservation and Event Reservation interact with the Data Retrieval component in the Data Access Layer to obtain necessary information. The Execute Order component interfaces with the Data Storage component to store orders in the database. Additionally, Execute Order sends payment details to the Payment Processing component in the External Services Layer and receives email confirmations from the Email Confirmation component within the same layer.

These architectural decisions were made to ensure a clear separation of concerns and modularity. The layered architecture pattern facilitates scalability and maintainability by allowing components to be developed, tested, and modified independently. Moreover, the use of role components within each layer promotes encapsulation and reusability, enhancing the system's flexibility. Interactions between layers are carefully orchestrated to ensure efficient data flow and communication, resulting in a robust and reliable ticket sales system.

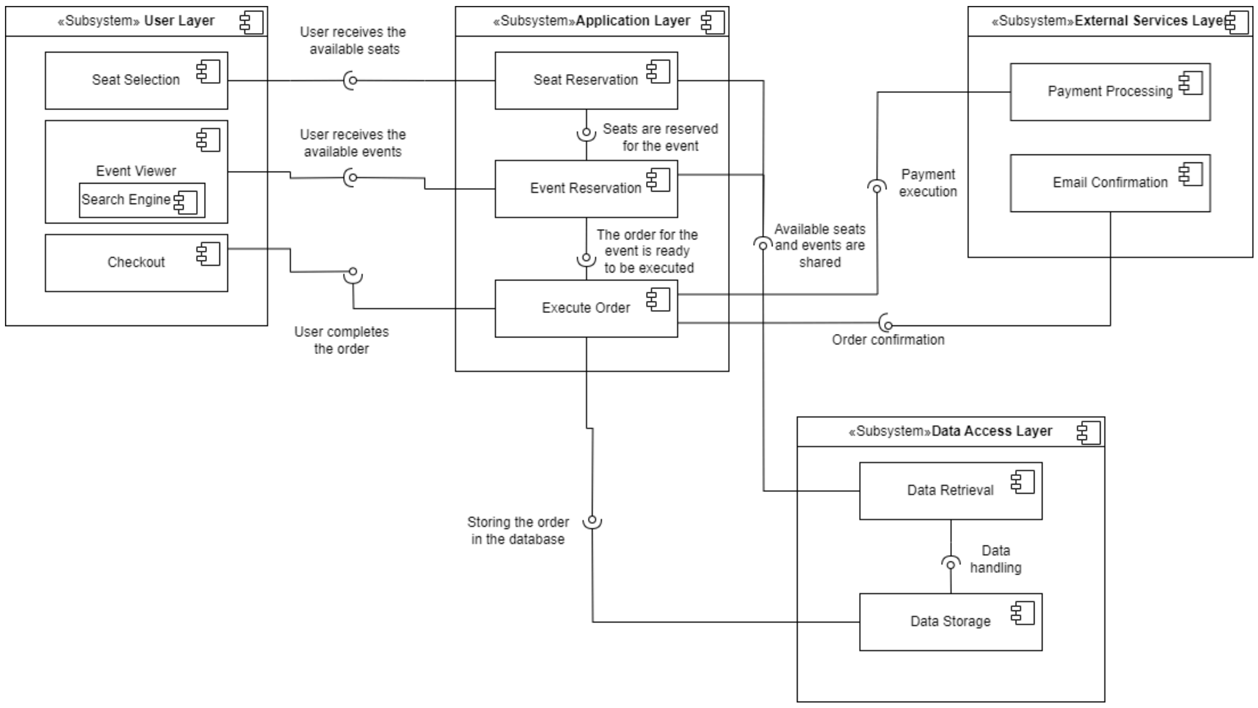
«Subsystem» User Layer
Seat Selection
Event Viewer
Search Engine
Checkout
User receives the available seats
User receives the available events
User completes the order
«Subsystem»Application Layer
Seat Reservation
Seats are reserved for the event
Event Reservation
The order for the event is ready to be executed
Execute Order
«Subsystem»External Services Layer
Payment Processing
Email Confirmation
Payment execution
Available seats and events are shared
Order confirmation
Storing the order in the database
«Subsystem»Data Access Layer
Data Retrieval
Data handling
Data Storage

Figure 1. Logical View

Process view: Sequence Diagram - Interaction Among Components

Our UML sequence diagram (Fig. 2.) provides a high-level, yet detailed overview of the interactions and message flow between the participants of our system, namely the user, User Interface, Application Server, Database, and External Services during the ticket purchasing process in the TicketBoss system.

It contains 4 objects and one actor. The actor is the customer of the TicketBoss system, and the 4 objects are: User Interface, Application Server, Database and External services, that correspond to the User Layer, Application Layer, Data Access Layer, and the External Services Layer components of the Logical View. The order of typical events and messages that are passed between elements of the ticket sales system is as follows:

1. User searches for an event

   1.1. The user interacts with the User Interface to search for an event
   1.2. The User Interface sends a search event request to the Application Server
   1.3. The Application Server retrieves event data from the Database based on the provided search criteria, such as artist, venue, date etc.
   1.4. The Application Server sends the event data back to the User Interface to display it to the user

2. User selects an event

   2.1. The user selects an event from the displayed list on the User Interface
   2.2. The User Interface sends an event selection request to the Application Server
   2.3. The Application Server retrieves detailed event information from the Database, which contains information about the artist, venue, images, textual description, and importantly the <u>seating arrangement and available seats</u>, etc.
   2.4. The Application Server sends the event information back to the User Interface for display

3. User selects seat(s)

   3.1. The user selects a seat for the chosen event through the User Interface
   3.2. For each selected seat, the User Interface sends a seat selection request to the Application Server
   3.3. The Application Server retrieves seat information from the Database, which contains the precise location (sector, name etc.), <u>price</u> of the selected seat and additional attributes (e.g. VIP, disabled etc.). With this action the App Server also updates the seat availability in the Database.
   3.4. The Application Server sends the seat information back to the User Interface for display
   3.5. The user can repeat the seat selection process if they want to reserve multiple seats, but a user is only able to select a maximum of 10 seats at a time, for 10 minutes, which begins after the selection of the first seat

4. User deletes seat(s) from selection

   Deleting or deselecting a seat involves the same logic as seen in seat selection. This allows the users to change their minds, discover the different prices. The user is able to delete the previously chosen seats, then select (new) seats again etc.

5. Checkout

   5.1. The user proceeds to checkout through the User Interface
   5.2. The User Interface sends a checkout request to the Application Server
   5.3. The Application Server stores the order information in the Database
   5.4. The Application Server forwards a payment request to the payment processing External Service
   5.5. If the transaction is successful, the Application Server sends a confirmation email to the user through an external service and updates the order status in the Database. Otherwise, it updates the seat information in the Database, so that the seat becomes available again, updates the order status to reflect an unsuccessful purchase and displays an error message to the user

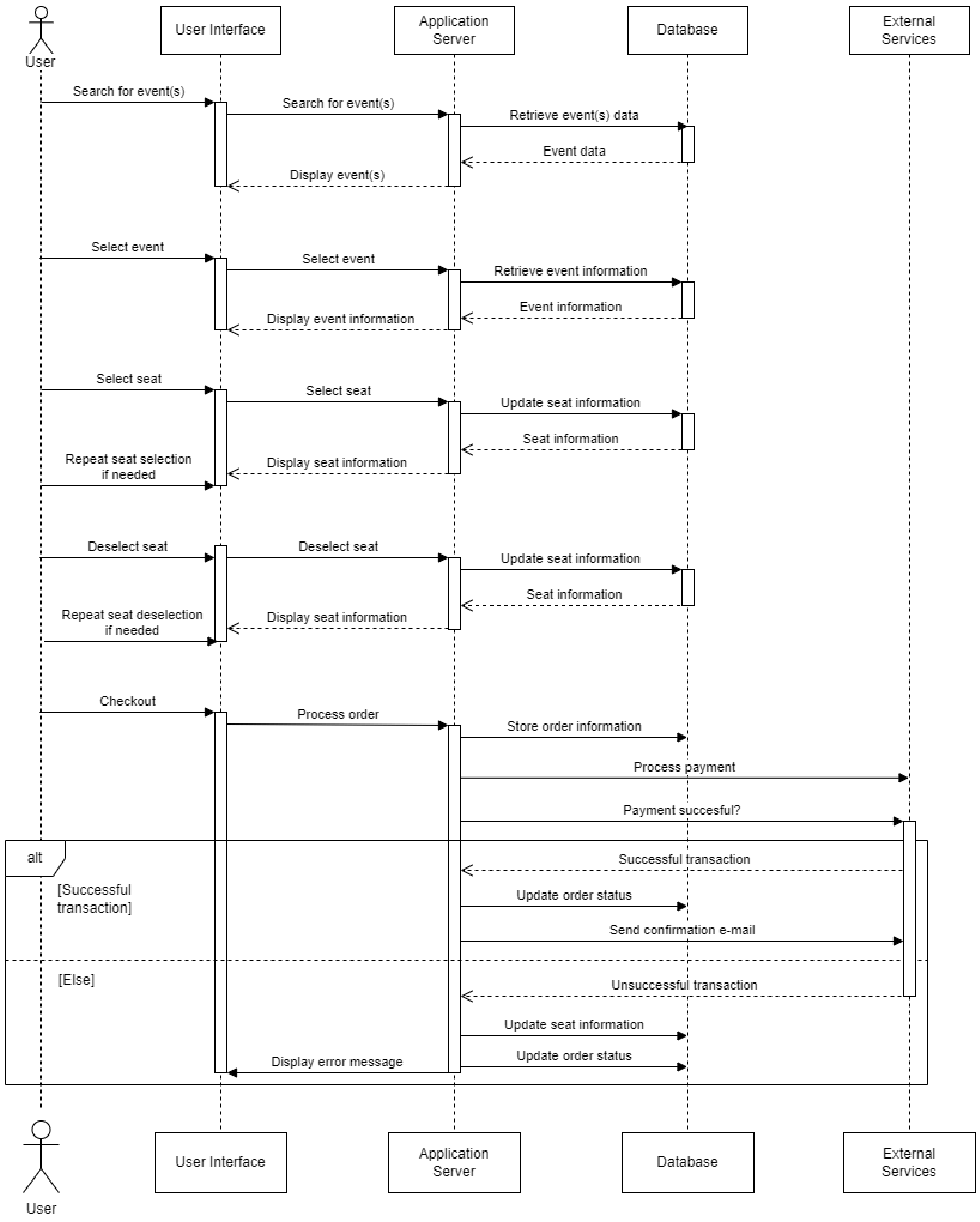

Figure 2. Process View

Use Cases 1 and 4

Use Case 1 (User selects seats and buys them)

- **Primary Actor:** User
- **Secondary Actors:** TicketBoss System, Payment Gateway
- **Preconditions:**
    - User is logged into the TicketBoss system.
    - User has navigated to the page displaying available seats for a specific entertainment event.
- **Main Flow:**
    - User selects the desired seats from the available options.
    - TicketBoss system holds the selected seats for the user for ten minutes.
    - User proceeds to checkout.
    - User enters credit card information and other personal information.
    - User confirms the purchase of the selected seats.
    - TicketBoss system processes the payment through the Payment Gateway.
    - Upon successful payment, TicketBoss system confirms the purchase and sends a confirmation email to the user.
    - Finally, TicketBoss marks the seats as occupied (no longer available for purchase).
- **Alternate Flow:**
    - If the user does not complete the purchase within ten minutes, the held seats are released for other users to buy them.
    - TicketBoss informs the current user that he/she has reached the 10 minutes timeout to purchase the seats.
    - TicketBoss redirects the user to the page that displays the available seats.

Use Case 4 (Two or more people select seats for the same event at the same time (within ten minutes of each other.) There is no conflict.)

- **Primary Actors:** User1, User2 (and potentially more users)
- **Secondary Actors:** TicketBoss System, Payment Gateway
- **Preconditions:**
    - User1 and User2 are logged into the TicketBoss system.
    - Both users have navigated to the page displaying available seats for a specific entertainment event. It is the same event for both of them.
- **Main Flow:**
    - User1 and User2 independently select the desired seats from the available options.
    - TicketBoss system holds the selected seats for each user for ten minutes.
    - Both users proceed to checkout independently.

    - Both users enter their respective credit card information and other personal information.
    - Both users confirm the purchase of their seats independently.
    - TicketBoss system processes the payments for both users simultaneously through the Payment Gateway.
    - Upon successful payment processing for each user, TicketBoss system confirms the purchases and sends confirmation emails to both users.
    - Finally, TicketBoss marks the seats bought by the two users as occupied (no longer available for purchase).
- **Alternate Flow:**
    - If any user does not complete the purchase within ten minutes, the held seats are released for other users to buy, but it does not affect the purchase process of the other user. Each purchase process is independent per user.

Addressing quality attribute requirements

We chose four of the most important quality attributes to inspect more in detail. These four are availability, reliability, performance and usability. Focusing on specific quality attributes helps prioritizing and addressing key aspects of the system. Though we could analyze other QAs as well (e.g. security and maintainability), for the sake of this assignment's length, we chose these four as they are crucial for the application's success and user satisfaction.

Availability means how long the system is up and running and how many people can access it. For TicketBoss, we find it acceptable if the system stops working for a few minutes during the late hours at night as it's less probable that someone will try to reserve a seat at that time. In addition, it is acceptable that the app doesn't handle all the users who want to access the platform at the same time. However, the waiting time shouldn't compromise the purchasing experience for the customer, meaning they should still ideally be able to buy a ticket even though the waiting takes a while. Importance of availability is moderate to high. With a limited amount of tickets and busy servers, it's unfortunately probable that not all customers are able to buy the tickets they wish. However, the architecture should enable as many users as possible to purchase tickets in a timely manner.

Reliability addresses how long it takes for the system to recover from errors. It answers the question how reliable the information and the actions are that take place in the application. For instance, when a user chooses a seat, are they really reserving the exact seat? Possibly it is already reserved by someone else and this should be indicated clearly to the user. TicketBoss should recover from errors only in a few minutes to avoid compromising the seat reservations. In the application, the seats are reserved for 10 minutes so the user shouldn't lose them even in case of an error. Even though there can be small errors, such as processing delays or network issues, the user has to get the seat they purchase. Otherwise, the customer may experience inconvenience, time and money loss and trust for TicketBoss.

Performance regards response times of each functionality/component of TicketBoss. For example, what is the speed of its search engine or how fast do seat prices load? For each functionality the response time is ideally under 1 second. The maximum response time should be a couple seconds and a loading sign should be presented to the user, so they know their request is being handled. The importance of performance is relatively high, but it's not the most important QA. Bad performance could lead to long waiting times in each function. This could leave the user dissatisfied and annoyed and the user wouldn't use this service again. However, there are different methods to take into account when designing the performance of TicketBoss. For example, compressing images in the application reduces the file size of images, which in turn leads to faster loading times and reduced data consumption. In addition, for instance pagination limits the amount of data fetched and displayed at once. Different indexing for the database speeds up database queries. Caching stores frequently accessed data temporarily in memory or disk, reducing the need to fetch data from the original source multiple times. These all reduce load times and resource usage, especially for large datasets, resulting in faster data retrieval and improved overall system performance. To balance out the workload of the system, multiple servers should be used. Another efficient way is to direct users to wait in a virtual queue in case of an immense amount of requests (imagine people purchasing tickets for a Taylor Swift concert). Lastly, it could be useful to apply a broker/load balancer to manage multiple requests. In other words, check which server is least busy and redirect requests.

Usability refers to how easy and efficient the system is to use. This means for example how comprehensible it is for the user to access and utilize all the features within the system. In the case of TicketBoss this could also regard how smooth the system's navigation flow is. Failures shouldn't be common or affect the overall experience, but e.g. small lagging is acceptable. The crucial parts, such as purchasing itself, shouldn't fail. After reserving the seat, the process should go right efficiently in the first try. Usability can be addressed in many ways. The user interface should be self-explanatory, TicketBoss should provide a help section and it should cater to various users, e.g visually impaired users. In other words, usability brings attention to all customer needs and we see it has high importance. Poor usability can lead to lack of trust and customer satisfaction. Clients may discontinue use due to insufficiently satisfactory and weak navigation flow.