Software Engineering and Testing. BSC Year 2, 2024/2025
(Assignment 3 - 20%)

Assessment 3: Design and Draft Implementation

Submitted by: Shae Mohamed (B00164905)
Sergio Garcia (B00165542)
Robert Melciu (B00166345)

Submission date: 23/03/2025

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Ordinary Degree in Computing in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated.

Author: Shae Mohamed Dated: 10/03/2025

Author: Sergio Garcia Dated: 10/03/2025

Author: Robert Melciu Dated: 10/03/2025

Contents

The Game Day - Event Ticketing Purchase Website

Abstract / Executive Summary:

This document describes the design and functional specs for "The Game Day", the ticket purchasing website. The purpose of this website it to create an online platform for users to purchase ticket for playing in hobby sporting events. The website is designed to have a user-friendly interface that allows an easy event organisation and ticket purchasing. This document will illustrate the function specs, user requirements and application specs for the website. making sure it meets the needs for both the user and the event organisers. the design will be based on a (OOAD) design and will use UML Diagrams for clarity and structure. the website will include features such as event browsing, user account management and ticket validation.

Project Definitions

- Purpose of document: This document is to outline the design and specs for the website "The Game Day" ticketing website including application requirements, functionality and design decisions.
- What is the project? "The Game Day" is an event ticketing website that allows users to browse sporting events close to their current location, view event details, purchase tickets to participate and make social interactions with other users that have a similar interest and manage their bookings.
- Functional Specifications:
    - User registration and login system
    - Event listing and search functionality
    - Secure payment processing
    - User account management
    - Ticket validation system

- Main components of the software system:
    - Front-end: User interface and interactions (HTML, CSS)
    - Back-end: Server-side processing (Database, PHP)
    - Database: Storage of user data, event details, and ticket information

Document Revision

- Rev. 1.0 – Initial version (March 10, 2025)

Methodology

- System Models – UML:

UML helps us understand the structure and behaviour of our system through the iagrams, it does help us to organize and define the relationships between different components so that we don't make any mistakes throughout the building of our project.

- Use of OOAD:

OOAD helps us to give it a structure to the database and project overall that represents real worls entries by breaking down the database into manageable components. This method enhances modularity, reusability, and scalability, making the system easier to maintain and expand.

- Purpose of Using Classes / Class Diagram:

Class Diagram helps us to model real world entities and encapsulate data and functionality. It helps us understand how the different components interact with each other. With this, the code is modular, reusable and easier to maintain.

- Static vs. Dynamic Case Diagrams:

Static case diagrams help us understan the overall structure of our database and the interactions between users and the other components of the database.On the other hand, dynamic case diagrams focus mainly on the order that actions are done and how the database behaves during those interactions, like ticket purchasing for example.

- What is an ERD?

An ERD helps us to model the data structure by representing the entities and their relationships in a diagram. This diagram helps us to design the database efficiently by identifying how different data components, like users, events, and tickets, are related. It ensures that the data is organized and is efficiently stored.

- Purpose of Using Classes:

Classes helps us to define and structure real world entities in the database. By encapsulating related data and behaviors into those classes, we will keep the code organized, modular, and reusable. This will also ensure that each part of the database can be easily modified or extended if it is ever needed.

- Volatile vs. Persistent Storage:

Volatile storage is used for temporary data storage, we use it in PHP variables where data like user session information or temporary ticket selections are stored during interactions. This data is temporarly stored in Ram, and will be erased on by end of the session or when the system shuts down. On the other hand, persistent storage is used to store long term data in our MySQL database, like user profiles, event details, or ticket purchases. This ensures that crucial information is retained even if the system is offline.

- User Interface Template:

The Interface Template helps us as a visual layout and structure of our platform, making sure that our platform is consistent and intuitive.

Requirements

4.1 Use Cases

- Use Case 1: Search and Browse Events
- Use Case 2: View Event Details
- Use Case 3: Select and reserve Tickets
- Use Case 4: Purchase Tickets

- Use Case 5: Cancel or Refund Tickets

4.2 Use Case Specifications

Search and Browse Events

Goal:
Allow any user (guest or registered) to discover available events.
Flow of Events:

- **User** opens the Ticketing website.
- **User** enters search criteria (e.g., event name, category, date, location) or simply browses featured events.
- **System** queries the database for matching events.
- **System** displays a list of events with basic info (name, date, venue, price range).
- **User** selects an event to learn more or proceeds to reserve tickets.

Postconditions:

- The user sees all relevant events that match their criteria.
- The system is ready to show further details on any selected event.

View Event Details

Goal:
Provide detailed information about a selected event.
Flow of Events:

- **User** clicks on a specific event from the search results.
- **System** retrieves the event's full details (date, start time, venue, seat map/pricing tiers, description).
- **System** displays these details, including any special offers or limited-time deals.
- **User** can decide to proceed with ticket selection or return to the list.

Postconditions:

- The user has all the info needed to decide on purchasing tickets.

Select and Reserve Tickets

(Think of this like "Reserve Room" in the hotel example, but for tickets.)
Goal:
Allow a user to choose the number of tickets or specific seats and initiate a reservation in the system.
Flow of Events:

1. **User** selects the desired event.
2. **User** chooses the number of tickets or specific seats (if a seat map is used).
3. **System** verifies seat/ticket availability in the database.
4. **System** places the selected seats/tickets in a "pending" or "cart" state to prevent others from booking them simultaneously.
5. **System** calculates a subtotal (including any service fees or taxes) and displays it to the user.
6. **User** confirms they wish to proceed to payment.

Postconditions:

- The user has a "pending reservation" or items in their cart.
- The system marks those tickets as temporarily reserved.

Purchase Tickets

Goal:

Enable a user to complete payment and finalize the ticket purchase.

Flow of Events:

1. **User** reviews the order summary (ticket quantity, seat info, total cost).
2. **User** provides payment details (credit card, PayPal, etc.). *(Login might be required here, but we're not listing "Login" as its own requirement.)*
3. **System** processes payment via a payment gateway.
4. **System** updates the database to confirm the purchase and finalize the ticket status to "sold."
5. **System** generates an order confirmation (order ID, e-ticket link, or QR code) and sends it to the user (onscreen + email).

Postconditions:

- The user's purchase is confirmed.
- The system stores an order record and marks tickets as sold.

Cancel or Refund Tickets

Goal:

Allow a user to request cancellation or refund if the event's policy allows it.

Flow of Events:

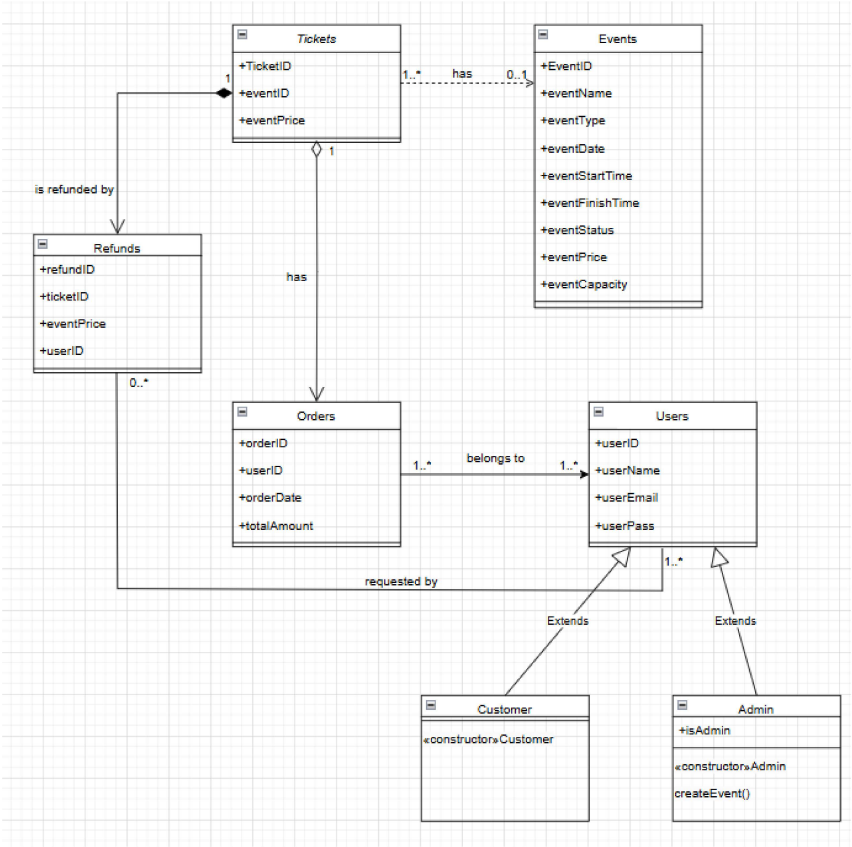
1. **User** navigates to their "My Tickets" or "Order History" page.
2. **User** selects a previously purchased ticket/order and chooses "Cancel/Refund" (if eligible).
3. **System** checks the event's refund policy (time limits, partial refunds, etc.).
4. **System** updates the ticket status to "canceled" if allowed.
5. **System** processes a refund or partial refund, if applicable.
6. **System** displays a confirmation of the cancellation/refund to the user and updates the database records.

Postconditions:

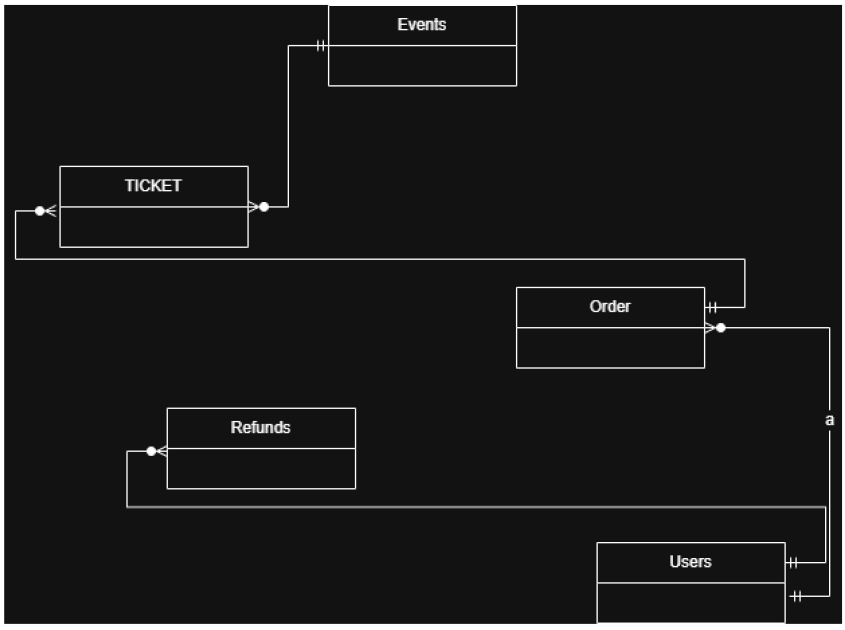
- The user's ticket is canceled.
- The system records any refund transaction.

Case Diagrams

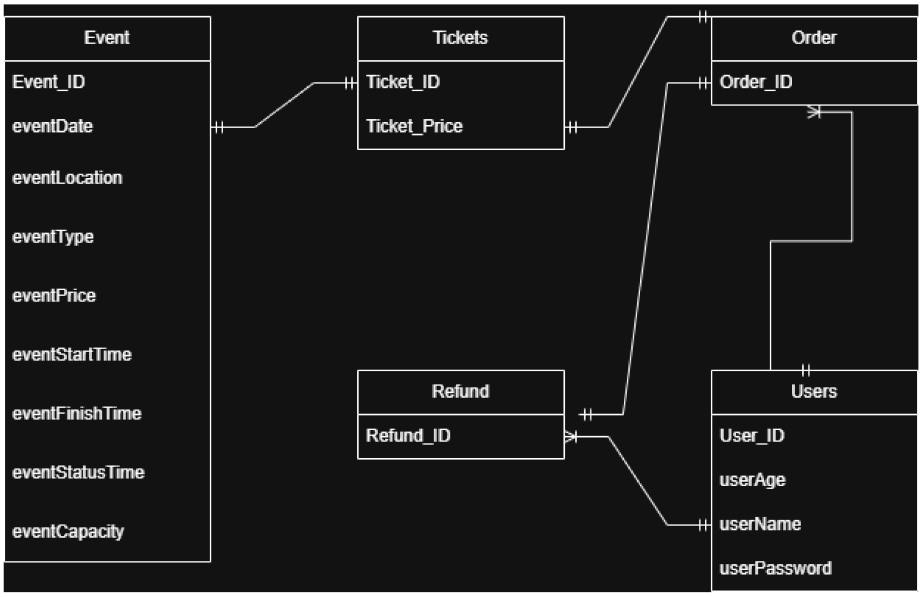
- Class Diagram:



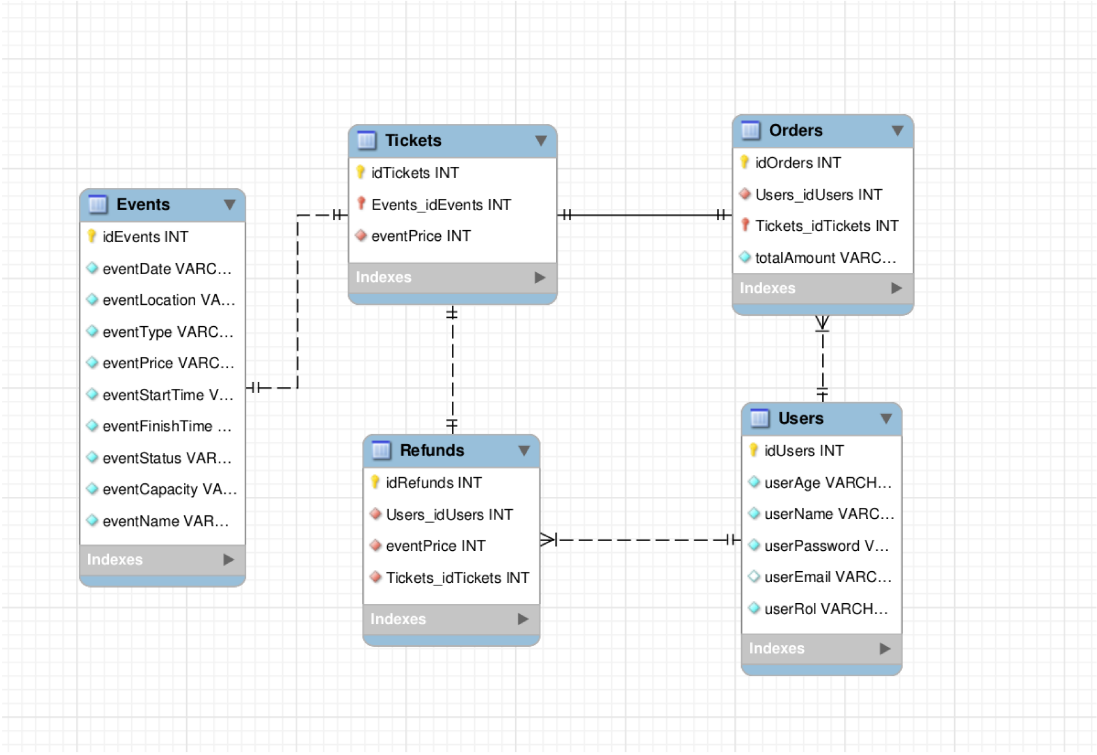
- Entity Relationship Diagram (ERD):
  - Conceptual:



- Logical:



- Physical:



Paragraph explaining design decisions:

Class_diagrams:

The design decisions aim to ensure efficient management of users, events, tickets, and transactions while maintaining clarity and scalability. The User class serves as the base class, with two specialized subclasses: Customer and Admin. This inheritance structure allows for clear differentiation between the roles, where Admin can create and manage events, while Customer can browse, purchase, and track events. The Order class is connected to the Ticket class through aggregation, indicating that an order can contain multiple tickets but tickets can exist independently. The Refund class is composed with Ticket, showing that refunds are tightly coupled to the ticket's lifecycle. The Event class links to Ticket, reflecting the association between specific events and their tickets. The design is modular, ensuring flexibility for future enhancements, and is aligned with the functional requirements of user and event management, ticket tracking, and refund processing.

ERD_diagrams:

The relationships between the core entities (Users, Orders, Tickets, Events, and Refunds) are modeled to reflect real world operations. The Users table stores user information, including personal details, allowing for secure authentication. The Orders table captures user purchases, associating tickets with specific events and users. Tickets are linked to Events, with each event having multiple tickets, reflecting the sales for different events. The Refunds table is connected to both Users and Tickets to facilitate refund requests, ensuring the platform's flexibility in handling user requests. The normalization of data reduces redundancy and ensures the scalability of the system. The use of foreign keys to maintain relationships between tables allows for data integrity, ensuring that events, orders, tickets, and refunds are all correctly associated with their respective users.

Conclusions

The project is progressing well, with most of the core features defined. The design has revealed the need for a more robust payment processing system, which was not fully anticipated in the initial proposal. Additional changes include the introduction of a user roles system (Admin vs. Customer), which was added to handle different levels of access.

Checklist: Is your document complete, and correct?

Content:

- Does the design include all requirements from the customers' needs?
  Yes
- Are you satisfied with all parts of the document?
  Yes
- Do you believe all parts have been implemented?
  Yes
- Have you explained your methodology and design choices?
  Yes
- Have you clearly articulated your understanding of the purpose of all diagrams created ?
  Yes
- What are these diagrams? Why you need them? How were they developed?
  UML Diagrams are used to model our database structure (class diagrams) and interactions (use case and case diagrams). They were developed to organise the database and to ensure a clear design, avoiding possible mistakes as well in that way.

  Our Entity-Relationship Diagram (ERD) is used to model how entities like users, events, and tickets are connected within the database, ensuring data integrity and an efficient structure. Our ERD helps in defining how data will be stored and accessed in the database.

- Is each part of the document in agreement with all other parts?
  Yes
- Does the design create a solution for the initial proposal?
  Yes

Completeness:

- Are all the necessary components specified?
  Yes
- Are the design specifications precise enough?
  Yes
- Are all sections from the document template included – if changed, why?
  Yes, they have not been changed

Clarity:

- Is the design reasonable?
  Yes
- Is the level of details for each design section appropriate?
  Yes
- Is the design written in a language appropriate to the intended audience of software engineering teams?
  Yes
- Are all items clear and unambiguous?
  Yes