Software Requirements Specification (SRS) for Helwan Tournaments

Table of Contents

1. Introduction

The Software Requirements Specification (SRS) document outlines the requirements and specifications for the development of the Helwan Tournaments platform. This document serves as a guide for the design, development, and implementation of the platform, providing a comprehensive overview of its functionalities, features, and scope.

1.1 Purpose of the Document

The purpose of this document is to define the functional and non-functional requirements of the Helwan Tournaments platform, including its key features, user interactions, system architecture, and deployment considerations. It serves as a reference for stakeholders, developers, designers, and testers involved in the project, ensuring a clear understanding of the project objectives and scope.

1.2 Scope of the Project

The scope of the Helwan Tournaments platform encompasses the development of a web-based application that facilitates competitive gaming events for titles like Valorant. The platform allows users to register, participate in tournaments, schedule matches, manage teams, and track performance metrics. It aims to provide a seamless and secure experience for players, teams, and organizers, fostering a vibrant community of gaming enthusiasts.

1.3 Overview of Helwan Tournaments Platform

The Helwan Tournaments platform is designed to streamline the organization and management of gaming tournaments, catering to the needs of players, teams, and organizers. By leveraging modern technologies such as Java Spring Boot, React.js, and MySQL, the platform offers a robust and scalable solution for hosting competitive gaming events. With features such as user

authentication, tournament creation, match scheduling, and real-time notifications, the platform aims to enhance the gaming experience and foster a sense of community among gamers.

1.4 Definitions, Acronyms, and Abbreviations

SRS: Software Requirements Specification

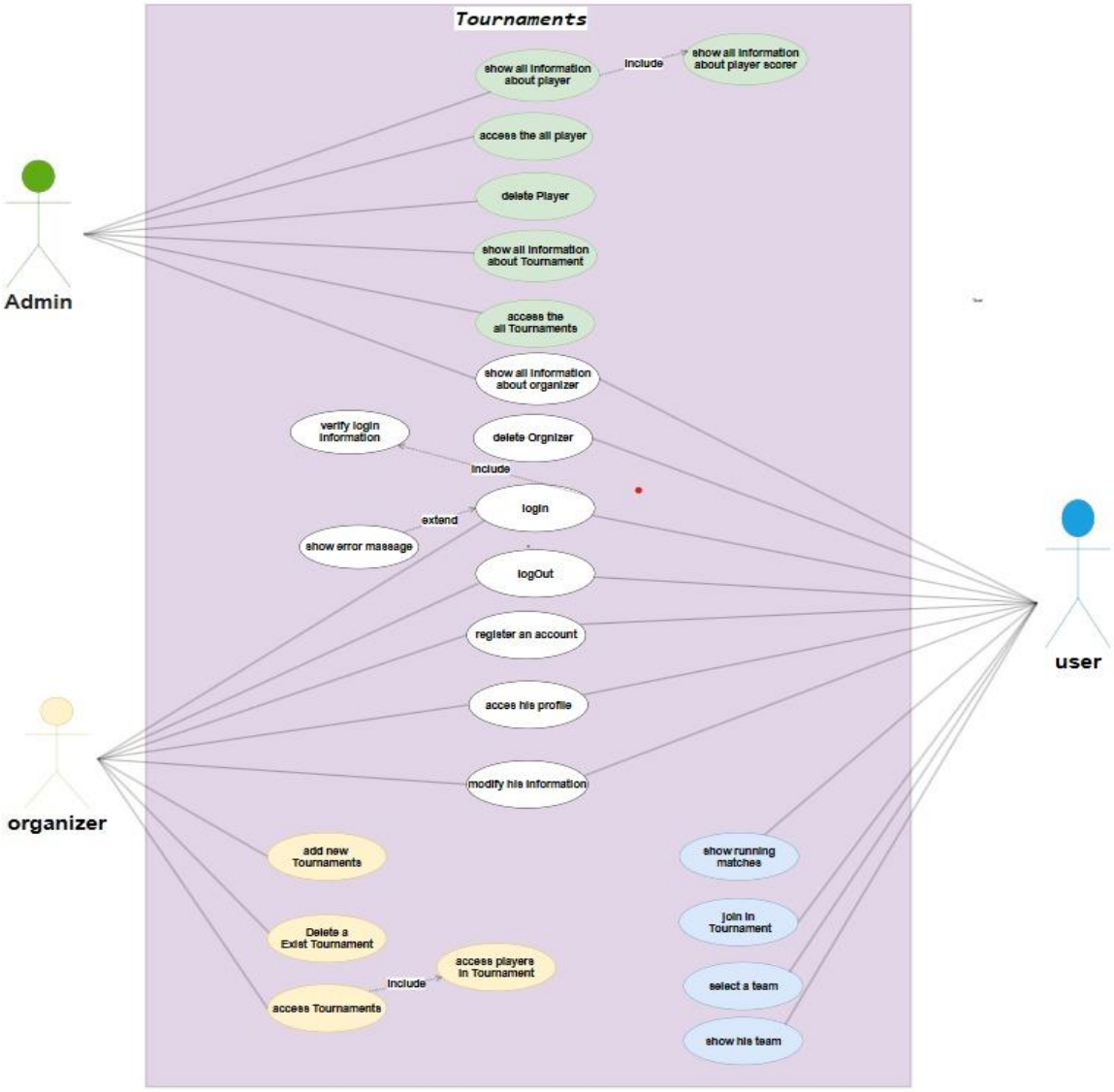
API: Application Programming Interface

JWT: JSON Web Token

CRUD: Create, Read, Update, Delete

2. Functional Requirements

The functional requirements of the Helwan Tournaments platform define the specific features and capabilities that the system must provide to its users. These requirements describe the interactions between users and the system, outlining the tasks and functionalities supported by the platform.



2.1 User Authentication

User Registration: Users should be able to create an account on the platform by providing necessary information such as username, email, and password.

User Login: Registered users should be able to log in to their accounts using their credentials.

Password Recovery: Users should have the ability to recover their password in case they forget it, through email verification or security questions.

2.2 Tournament Management

Create Tournament: Organizers should be able to create new tournaments by specifying details such as title, description, game title, start date, end date, and registration deadline.

Manage Tournament: Organizers should have access to a dashboard to manage tournaments, including updating tournament details, managing registrations, and closing registrations.

Join Tournament: Players should be able to join tournaments by registering their teams or as individual players, depending on tournament rules and regulations.

View Tournament Details: Users should be able to view detailed information about upcoming tournaments, including schedule, participating teams, rules, and prize pool.

2.3 Match Management

Schedule Match: Organizers should be able to schedule matches within tournaments, specifying match date, time, participating teams, and match format.

Record Match Results: After a match is completed, organizers or team captains should be able to record the match results, including scores, winner, and any relevant details.

View Match History: Users should be able to view past match results, including scores, participating teams, and match details.

2.4 Team Management

Create Team: Players should be able to create new teams or join existing teams on the platform.

Manage Team: Team captains should have access to a dashboard to manage team members, invite new members, and remove existing members.

View Team Details: Users should be able to view details about teams, including team name, members, matches played, and performance statistics.

2.5 User Profile

View Profile: Users should be able to view their own profile information, including username, email, profile picture, and performance statistics.

Update Profile: Users should have the ability to update their profile information, including email, password, and profile picture.

2.6 Notification System

Email Notifications: Users should receive email notifications for important events such as tournament registration confirmation, match schedules, and match results.

In-App Notifications: Users should receive in-app notifications for real-time updates on tournament status, match schedules, and other relevant information.

2.7 Admin Panel

Manage Users: Administrators should have access to a panel to manage user accounts, including user registration, account activation, and account suspension.

Manage Tournaments: Administrators should be able to manage tournaments, including creating, editing, and deleting tournaments as needed.

View Reports: Administrators should have access to reports and analytics on user activity, tournament participation, and platform usage metrics.

3. Non-Functional Requirements

The non-functional requirements of the Helwan Tournaments platform define the quality attributes and constraints that govern the system's operation and

performance. These requirements specify how the system should behave in terms of reliability, performance, security, usability, and other key aspects.

3.1 Performance Requirements

Responsiveness: The platform should respond to user interactions within 2 seconds under normal load conditions.

Scalability: The system should be able to handle a minimum of 1000 concurrent users without significant degradation in performance.

Throughput: The platform should support a minimum of 1000 requests per minute for user authentication, tournament registration, and match scheduling.

3.2 Security Requirements

Data Encryption: User passwords and sensitive data should be stored securely using encryption techniques to prevent unauthorized access.

Authentication: User authentication should be performed securely using industry-standard protocols such as JWT (JSON Web Tokens) or OAuth 2.0.

Authorization: Access to sensitive operations and administrative functions should be restricted to authorized users based on their roles and permissions.

Data Privacy: The platform should comply with data privacy regulations such as GDPR (General Data Protection Regulation), ensuring that user data is protected and handled responsibly.

3.3 Reliability Requirements

Availability: The platform should have an uptime of at least 99.9% to ensure continuous availability to users.

Fault Tolerance: The system should be resilient to failures and errors, with mechanisms in place to handle exceptions gracefully and recover from failures quickly.

Backup and Recovery: Regular backups of the database should be performed to prevent data loss, and a recovery plan should be in place to restore the system in case of disasters.

3.4 Usability Requirements

Intuitive User Interface: The user interface should be intuitive and user-friendly, with clear navigation and easy-to-understand workflows.

Accessibility: The platform should be accessible to users with disabilities, following accessibility standards such as WCAG (Web Content Accessibility Guidelines).

Multilingual Support: The platform should support multiple languages to cater to a diverse user base.

3.5 Compatibility Requirements

Cross-Browser Compatibility: The platform should be compatible with modern web browsers such as Chrome, Firefox, Safari, and Edge.

Responsive Design: The user interface should be responsive and adapt to different screen sizes and devices, including desktops, tablets, and mobile phones.

3.6 Performance Requirements

Documentation: Comprehensive documentation should be provided for developers, administrators, and end-users, covering installation instructions, user guides, and API documentation.

Training and Support: Training materials and support resources should be available to help users understand and utilize the platform effectively.

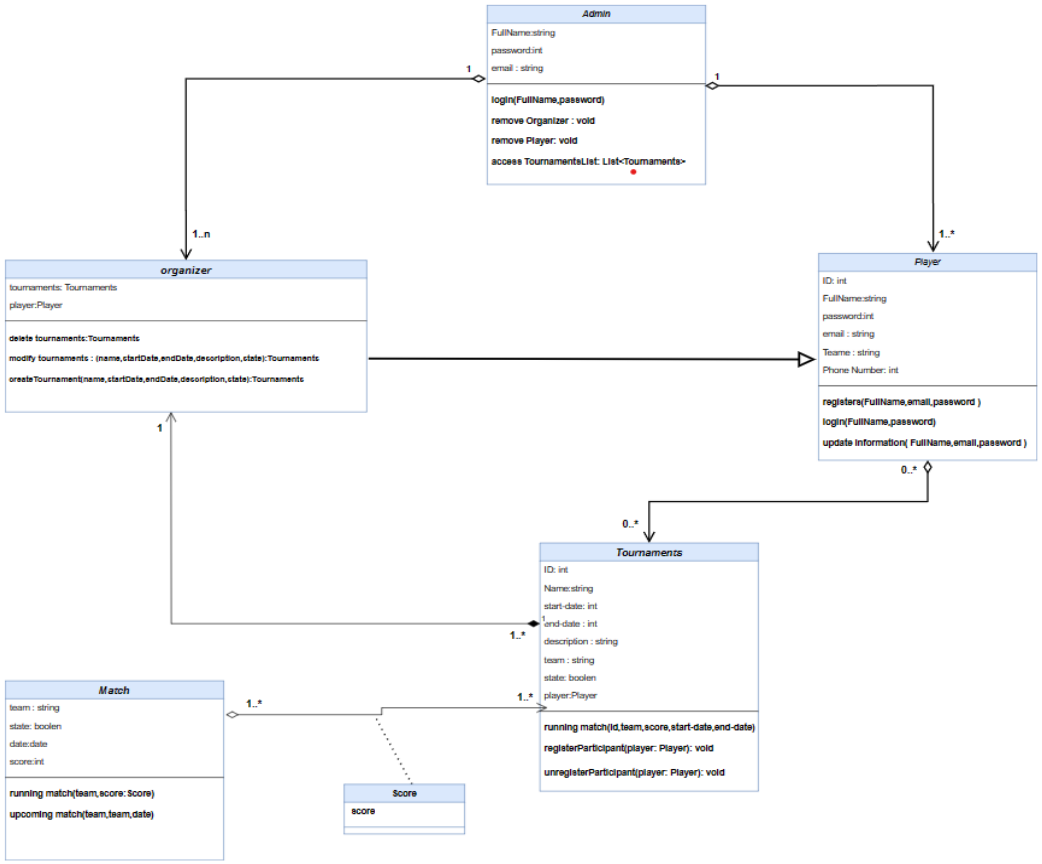
Community Engagement: The platform should foster a vibrant community of users, with forums, discussion groups, and social media channels for user engagement and feedback.

4. System Architecture

The system architecture of the Helwan Tournaments platform defines the structure, components, and interactions of the various modules and layers that make up the system. It provides a high-level overview of how the platform is designed and organized to meet its functional and non-functional requirements.

4.1 Overview

The Helwan Tournaments platform follows a modern, layered architecture that separates concerns and promotes modularity, scalability, and maintainability. It consists of the following key components:



Presentation Layer: The presentation layer handles user interactions and interfaces, including web-based interfaces for users, administrators, and organizers. It is built using React.js, a popular frontend JavaScript library, to create responsive and interactive user interfaces.

Application Layer: The application layer contains the business logic and application services that orchestrate the interactions between the presentation layer and the data layer. It is implemented using Java Spring Boot, a lightweight framework for building enterprise applications.

Data Layer: The data layer manages the storage and retrieval of data, including user information, tournament details, match results, and other relevant data. It utilizes a MySQL relational database to store structured data and ensure data integrity and consistency.

Security Layer: The security layer provides mechanisms for user authentication, authorization, and data protection. It integrates with Spring Security, a powerful authentication and access control framework, to enforce security policies and protect sensitive information.

Integration Layer: The integration layer facilitates communication and integration with external services, APIs, and third-party platforms. It enables features such as email notifications, payment processing, and social media integration.

4.2 Architectural Components

Frontend: The frontend component consists of React.js components, responsible for rendering user interfaces, handling user inputs, and communicating with the backend services via RESTful APIs.

Backend: The backend component comprises Java Spring Boot services, implementing business logic, data processing, and interaction with the database. It exposes RESTful APIs to the frontend for performing CRUD operations and other actions.

Database: The database component is a MySQL relational database, storing persistent data such as user profiles, tournament details, match results, and configuration settings.

Security: The security component includes Spring Security configurations and filters, ensuring secure authentication, authorization, and protection against common security threats such as cross-site scripting (XSS) and SQL injection.

4.3 Interaction Flow

The interaction flow within the system follows a typical client-server model, where the frontend interacts with the backend via RESTful API calls. Users interact with the frontend interface to perform actions such as user registration, login, tournament creation, match scheduling, and viewing match results. The frontend communicates with the backend services to process these actions, which in turn interact with the database to fetch or update data as needed.

4.4 Scalability and Extensibility

The system architecture is designed to be scalable and extensible, allowing for future growth and expansion. It can accommodate increasing user loads by horizontally scaling the backend services and database instances. Additionally, new features and functionalities can be easily added by extending the existing modules or integrating with external services through well-defined APIs.

5. User Interface Design

The user interface design of the Helwan Tournaments platform focuses on creating intuitive, visually appealing, and responsive interfaces that enable users to interact with the system efficiently and effectively. This section outlines the key principles, components, and design considerations that govern the user interface design.

5.1 Principles

The user interface design follows the following principles to ensure a positive user experience:

Simplicity: The interface is designed to be simple and easy to understand, with intuitive navigation and clear visual hierarchy.

Consistency: Consistent design elements, layout patterns, and interaction behaviors are maintained across the platform to provide a cohesive experience.

Accessibility: The interface is designed to be accessible to users of all abilities, following accessibility standards such as WCAG (Web Content Accessibility Guidelines).

Responsive Design: The interface is responsive and adapts seamlessly to different screen sizes and devices, including desktops, tablets, and mobile phones.

Feedback: Users receive immediate feedback for their actions, such as success messages, error alerts, and loading indicators, to enhance usability and transparency.

5.2 Components

The user interface consists of the following key components:

Navigation Bar: A persistent navigation bar at the top of the screen provides access to main sections of the platform, including Home, Tournaments, Matches, Teams, and User Profile.

Dashboard: Organizers and administrators have access to a dashboard that displays relevant information and actions related to their roles, such as creating tournaments, managing matches, and viewing reports.

Forms and Input Fields: Forms and input fields are used for user registration, login, tournament creation, match scheduling, and other data entry tasks. They include validation messages and error handling to guide users and prevent input errors.

Lists and Tables: Lists and tables are used to display information such as tournament schedules, match results, team members, and user profiles. They support sorting, filtering, and pagination for easy navigation and data exploration.

Modals and Dialogs: Modals and dialogs are used for displaying messages, notifications, confirmation prompts, and other contextual information without disrupting the main interface flow.

Buttons and Icons: Buttons and icons are used for triggering actions such as submitting forms, navigating between pages, and performing CRUD operations. They provide visual cues and affordances for interaction.

Responsive Layout: The layout of the interface adjusts dynamically based on screen size and device orientation, ensuring optimal viewing and interaction experiences across different devices and screen resolutions.

5.3 Design Considerations

In addition to the principles and components mentioned above, the user interface design considers the following aspects:

Branding and Theming: The interface incorporates branding elements and theming to reflect the identity and aesthetics of the Helwan Tournaments platform.

Localization: The interface supports multiple languages and locales to cater to a diverse user base, allowing users to switch between languages based on their preferences.

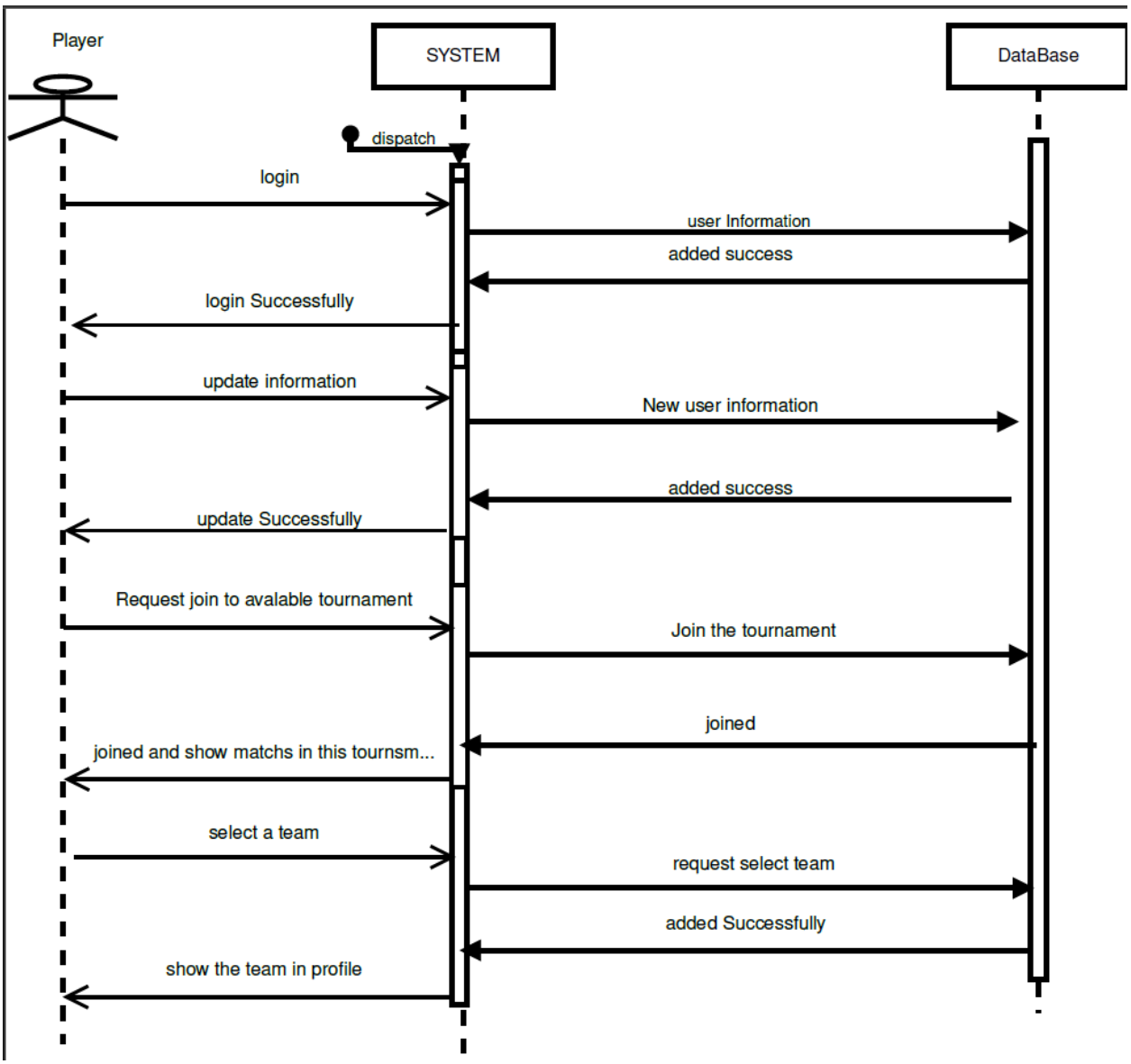
Performance Optimization: The interface is optimized for performance, with minimal loading times, efficient data fetching, and lazy loading of resources to enhance responsiveness and user experience.

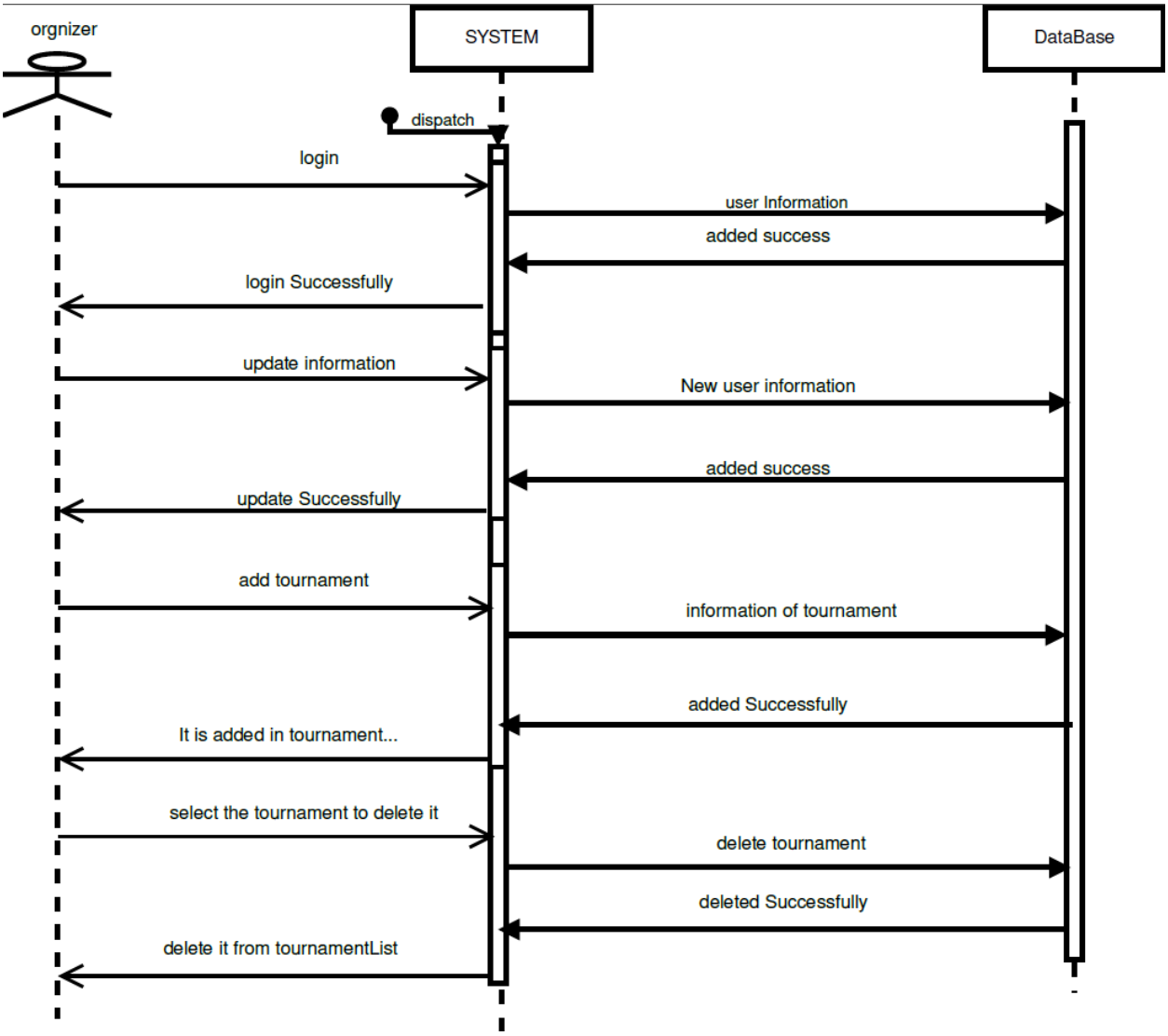
Cross-Browser Compatibility: The interface is tested and optimized to ensure compatibility with modern web browsers such as Chrome, Firefox, Safari, and Edge, providing a consistent experience across different platforms.

User Feedback and Iteration: User feedback is collected and incorporated into the design through usability testing, user interviews, and feedback forms. The interface undergoes iterative improvements based on user insights and observations.

6. Other Digrams

6.1 Sequence Diagrams




orgnizer
SYSTEM
DataBase
dispatch
login
user Information
added success
login Successfully
update information
New user information
added success
update Successfully
add tournament
information of tournament
added Successfully
It is added in tournament...
select the tournament to delete it
delete tournament
deleted Successfully
delete it from tournamentList

Object Constraint Language

i. Player

C1: context Player inv: email.matches('^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,})

C2 : context Organizer inv : username.matches (^[a-zA-Z0-9]+$)

C3: context Player inv: (username <> null) and (email <> null)

and (password <> null)

C4: context Player:: login (username: String, password: Int)

pre: username <> null and password <> null

post: result = (username = username //in registers

and password = password //in registers

C5: context Player inv: (name <> null) and (email <> null) and(password <> null) and (phoneNum<> null)

C6: context Player:: updateInformation (name: String, email: String,

password: int,phoneNum:int)

post:

self.name = name and

self.email = email and

self.password = password and

self.phoneNume = phoneNum

C7: context Player::JoinForTournament(tournament: Tournament)

pre: tournament.status = true // "Available"

post: player joined in tournament.

---

ii. Organizers

C8: context Organizer inv: email.matches('^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,})

C9 : context Organizer inv : username.matches (^[a-zA-Z0-9]+$])

C10: context Organizer inv: (username <> null) and (email <> null)

and (password <> null)

C11: context Organizer:: login (username: String, password: Int)

pre: username <> null and password <> null

post: result = (username = username //in registers

and password = password //in registers

C12: context Organizer inv: (name <> null) and (email <> null) and(password <> null) and (phoneNum<> null)

C13: context Organizer:: updateInformation (name: String, email: String,

password: int,phoneNum:int)

post:

self.name = name and

self.email = email and

self.password = password and

self.phoneNume = phoneNum

C14: context organizer ::createTournment (name, description,startDate,endDate,state)

pre: Tournament = null //not created

post : Tournament created success

C15: context Organizer

def: canEditTournament(organizer: Organizer, tournament: Tournament): Boolean =

organizer.tournaments->includes(tournament)

def: canDeleteTournament(organizer: Organizer, tournament: Tournament): Boolean = canEditTournament(organizer, tournament)

def: canCreateTournament(organizer: Organizer):

Boolean = true

---

iii. Admin

C16: context Admin:: login (username: String, password: Int)

pre: username <> null and password <> null

post: result = (username = username //in registers

and password = password //in registers

C19: context Admin:: removePlayer() : Boolean
post: self.removePlayer(player:Player)

C20: context Admin:: removeOrganizer () : Boolean
post: self.removeOrganizer (organizer: Organizer)