ÇANKAYA UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

CENG 407

Software Requirements Specifications

GoatCapella

Tarık Şen

Çağdaş Güleç

Beyza Tozman

İrem Beyza Aydoğan

Şeref Berkay Kaptan

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this Software Requirements Specification (SRS) document is to define the objectives, functionalities, and constraints of GoatCapella, a web-based platform designed to take algorithmic problem-solving and competitive programming to the next level. GoatCapella aims to support individuals in achieving their personal goals by leveraging methodologies such as challenge-based learning, gamification, and social learning.

In the last decade, platforms like LeetCode, CodeWars, and CodeForces have become popular spaces for algorithmic practice, offering thousands of problems and fostering competitive environments. GoatCapella differentiates itself by providing a unique challenge management system that integrates with these existing platforms via public APIs. Its core mission is to create a productive, engaging, and collaborative learning environment where users can:

- **Access Challenges Across Platforms:** Aggregate algorithmic problems from multiple platforms, retrieving essential data such as available questions and whether a user has solved a specific question.
- **Enhance Engagement:** Introduce new capabilities like public or private challenges, solution grading, and commentary, enhancing the experience for educators, learners, and coding enthusiasts.
- **Foster Community Learning:** Leverage social learning by allowing users to view, discuss, and learn from each other’s solutions.

This document was created to provide a structured and detailed guideline for the development of GoatCapella. It defines the platform's objectives and technical requirements, serving as a reference point for its design, implementation, and evaluation. For the readers, this SRS offers clarity about the system's purpose, capabilities, and expected outcomes. It ensures a shared understanding among all involved parties, enabling a coordinated approach toward building GoatCapella.

1.2. Scope

GoatCapella is a web-based platform designed to centralize, enhance, and personalize the experience of solving algorithmic challenges and participating in competitive programming. By integrating multiple coding platforms like LeetCode, CodeWars, and others through their public APIs, GoatCapella provides users with an engaging environment for creating, participating in, and managing challenges. The platform aims to foster a collaborative and competitive learning environment by incorporating methodologies such as gamification, social learning, and challenge-based learning.

GoatCapella supports features that empower users to create and participate in challenges tailored to their needs. Challenges can be configured as public, semi-public, or private and may be time-based or condition-based. These challenges can include one or more questions aggregated from various platforms. The system also supports speed-run challenges, where participants compete on speed and accuracy.

Administrators play a key role in GoatCapella, with a hierarchical admin structure allowing for management of challenges. Admins can view participant answers, provide comments, and grade solutions, with final rankings and awards reflecting these evaluations. After a challenge concludes, participant answers can optionally be reviewed publicly, fostering a learning environment where users can share and discuss solutions.

The platform also introduces system-generated challenges, where GoatCapella's AI creates challenges based on user preferences, performance, and activity. These challenges can be assigned to users automatically or upon request, and users have the flexibility to customize their configurations.

GoatCapella integrates gamification elements to enhance user engagement. Participants and challenge creators earn platform coins based on their performance and the popularity of their challenges. Daily streaks are rewarded to encourage regular participation, and users can spend these coins in the platform's shop to purchase items such as streak freezers or experience accelerators.

Future developments include the launch of a mobile application to improve accessibility and user continuity, as well as the creation of an independent coding platform to reduce dependency on third-party systems and better cater to specific user needs.

By combining these features into a single, unified system, GoatCapella bridges the gaps in existing platforms, providing an engaging solution for coding education and competition.

1.3. Definitions, Acronyms, and Abbreviations

- **API (Application Programming Interface):** A set of functions and protocols enabling integration and communication between different software platforms.
- **CBL (Challenge-Based Learning):** A learning methodology that emphasizes solving real-world challenges to foster engagement and skill development.
- **Gamification:** The use of game design elements such as leaderboards, badges, and rewards to enhance user motivation and engagement in learning and activities.
- **Keycloak:** An open-source identity and access management system used for authentication and authorization, supporting Single Sign-On (SSO).
- **Redis:** An in-memory NoSQL database utilized for caching frequently accessed data to reduce database load and improve system performance.
- **Docker:** A platform that allows the development and deployment of applications in lightweight, portable containers.
- **Docker Compose:** A tool for defining and managing multi-container Docker applications using a YAML configuration file.
- **GraphQL:** A query language for APIs that enables efficient and flexible data retrieval through a single endpoint, reducing data redundancy.
- **Streak Freezer:** A shop item in GoatCapella that helps users preserve their daily challenge streaks during periods of inactivity.
- **DQN (Deep Q-Network):** A reinforcement learning algorithm used in GoatCapella's AI to dynamically adjust question difficulty based on user performance.
- **Collaborative Filtering:** A recommendation algorithm that suggests items (challenges) based on similarities in user behavior or preferences.
- **Content-Based Filtering:** A recommendation algorithm that suggests items based on features or attributes of previously selected items.
- **SAML (Security Assertion Markup Language):** A standard for secure single sign-on and identity federation, supported by Keycloak.
- **CI/CD (Continuous Integration/Continuous Deployment):** Practices used to automate and streamline the development and deployment of software applications.

- **RL (Reinforcement Learning):** A type of machine learning that optimizes decision-making by learning from feedback in dynamic environments.
- **Bayesian Optimization:** A technique used to fine-tune AI models by iteratively improving predictions based on probabilistic evaluations.

1.4. Overview

This Software Requirements Specification (SRS) document is designed to outline the objectives, functionalities, and constraints of GoatCapella. It serves as a structured guideline for the development, testing, and deployment of the platform, ensuring alignment between the goals of the system and the expectations of its users. Each section of the document is crafted to provide clarity about the system's design, features, and technical requirements.

The SRS is organized as follows:

1. **Introduction:**

This section introduces the purpose, scope, and key terminologies used throughout the document. It provides an understanding of GoatCapella's mission and how this document supports its development.

- **Purpose:** Defines why GoatCapella exists and the goals it aims to achieve.
- **Scope:** Explains the breadth of the platform's capabilities, user interactions, and future directions.
- **Definitions and Acronyms:** Lists technical terms and abbreviations to ensure clarity.
- **Overview:** Summarizes the document’s structure and its role in guiding the project lifecycle.

2. **Overall Description:**

This section provides a description of the system's context, features, and interactions. It lays the foundation for understanding GoatCapella's architecture and design choices.

- **Product Perspective:** Describes how GoatCapella integrates with existing platforms and fits into the larger ecosystem of coding and algorithmic learning tools.
- **System Interfaces:** Details the communication pathways between GoatCapella and external systems, such as coding platforms and APIs.

- **User Interfaces:** Explains how users interact with the platform, including challenge creation, grading, and participation workflows.
- **Hardware and Software Interfaces:** Specifies the technical environment needed to support the platform, including system requirements and integration tools like Docker and Redis.
- **Communications Interfaces:** Discusses the protocols and standards used for data exchange within the system and with external APIs.
- **Product Functions:** Summarizes the platform's features, such as configurable challenges, adaptive learning, and gamification elements.
- **User Characteristics:** Defines the target audience, including learners, educators, and competitive programmers, along with their needs and technical proficiency.
- **Constraints:** Highlights limitations like API dependency, scalability challenges, and security considerations.
- **Assumptions and Dependencies:** Lists external factors and dependencies, such as third-party API availability and compliance with platform-specific rules.

3. Specific Requirements:

This section outlines the technical and functional requirements needed to implement GoatCapella effectively.

- **Functional Requirements:** Defines the platform’s core functionalities, including challenge creation, user authentication, leaderboard management, and AI-driven recommendations.
- **Non-Functional Requirements:** Establishes performance expectations, such as response time, scalability, reliability, and security measures.
- **Logical Database Requirements:** Describes the structure and relationships of the data stored within the platform, including user profiles, challenges, and performance metrics.
- **Design Constraints:** Specifies constraints such as technology stack choices (e.g., ASP.Net Core, Redis, Keycloak) and adherence to industry standards.
- **Software System Attributes:** Discusses qualities like maintainability, usability, portability, and adaptability.

2. Overall Description

2.1. Product Perspective

2.1.1. User Interfaces

1. Home Page

This is the landing page for all users. It displays ongoing challenges for both authenticated and guest users.

2. Selected Challenge Page

This page appears after a user selects a specific challenge, presenting customized views tailored to the roles and responsibilities of each type of user. The functionalities provided depending on the user type are as follows:

Guest Users:

- View the description of the selected challenge, including basic details such as the challenge creator and the number of participants.

Authenticated Users:

- View challenge details and status
- Track progress
- See comments and solutions (if the challenge is configured accordingly)
- Join and withdraw
- Make comments and publish solutions (if the challenge is configured accordingly)

Challenge Creators (Admins):

- View participant solutions
- Grade solutions
- Update challenge configurations
- Add sub-admins
- Cancel the challenge

Sub-Admins:

- Grade the participant’s solutions and make comments.

3. **Challenge Creation Page**

Authenticated users can create challenges and configure their details on this page. Once created, the user becomes the administrator of the challenge.

4. **Shop Page**

Authenticated users can purchase items (e.g., boosters) that assist with their challenges on this page.

5. **Profile Page**

Authenticated users can view their completed challenges, ongoing challenges, rank, badges, and purchased shop items.

6. **Account Page**

Authenticated users can update their personal information.

7. **Login Page**

This page allows existing users to log into the application.

8. **Register Page**

New users can register for the application here.

9. **Admin Page**

This page allows administrators to manage application-level settings, including configuring shop items and managing challenges.

2.1.2. Hardware Interfaces

This section describes the hardware interfaces required for the system to function effectively. The system is designed to interact with servers, client devices, and other physical components to support the functionality of the challenge-based application [2].

Deploy Environment Hardware

Type: Cloud-based or dedicated servers.

Purpose: The server will host the application backend, manage user authentication, process challenges, and store user-related data (e.g., challenge progress, submissions).

CPU: Quad-core 3.0 GHz or higher.

RAM: 16 GB or higher.

Storage: SSD with at least 1 TB of capacity.

Network: High-speed internet connection (1 Gbps or higher) to handle multiple user requests.

Client Hardware

Supported Devices: The application should run on the following client devices: Windows 10/11, macOS 11.0+, Linux (Ubuntu 20.04+), Android 10.0+ and iOS 14.0+.

CPU: Dual-core 1.5 GHz or higher.

RAM: 2 GB or higher.

Network: Devices must have stable internet access (minimum 10 Mbps).

Development & Testing Environment Hardware

Type: Local or virtual servers.

Purpose: Used for system development, testing, and debugging.

CPU: Dual-core 2.5 GHz or higher (virtualized processors acceptable).

RAM: 8 GB minimum (16 GB preferred for smoother multitasking during testing).

Storage: SSD with at least 512 GB capacity.

Network: Stable internet connection (100 Mbps or higher) to facilitate collaboration, code repository access, and integration with CI/CD pipelines.

2.1.3. Software Interfaces

This section describes all the software interfaces required for GoatCapella to function effectively, focusing on how and why it interacts with them. These interfaces enable functionalities like authentication, database management, caching and integration with external systems. For each interface, details about its name, source, version, purpose, and communication methods are outlined, including references to documentations of the interfaces.

Operating System

Source: Open source or proprietary, depending on the chosen OS (Ubuntu or Windows).

Version: Ubuntu 22.04 LTS (Linux) / Windows Server 2022.

Purpose: Underlying environment for running docker, which in turn hosts GoatCapella and its supporting containers. Ubuntu (Linux) is preferred for production environments due to its stability, and wide adoption in containerized workloads, while Windows may be used for local development.

Communication: Primarily indirect communication due to the abstraction provided by Docker Engine and Docker APIs.

Documentations:

- Ubuntu Official Documentation [1].
- Windows Server Documentation [2].

ASP.NET Core

Source: Open source, maintained by Microsoft.

Version: ASP.NET Core 8.0.

Purpose: Core backend framework for the system.

Communication: JSON over HTTP/HTTPS for internal API communication.

Documentation: Official Documentation [3]

MongoDB

Source: Open source.

Version: MongoDB 8.0.

Purpose: NoSQL database for storing unstructured or semi-structured data, such as challenges.

Communication: C# Driver and Entity Framework Core Provider.

Documentation: Official Documentation [4].

PostgreSQL

Source: Open source.

Version: PostgreSQL 17.2.

Purpose: Primary relational database for structured data storage.

Communication: ORM with Entity Framework Core.

Documentation: Official Documentation [5].

Keycloak

Source: Open source, maintained by Red Hat.

Version: Keycloak 26.0.

Purpose: Authentication, authorization and Single Sign On (SSO) solution.

Communication: OAuth 2.0 and OpenID Connect protocols over HTTPS.

Documentation: Official Documentation [6].

Redis

Source: Open source, maintained by Redis Ltd.

Version: Redis 7.4.

Purpose: In-memory data store used for caching to enhance application performance, reduce load on external APIs and databases.

Communication: StackExchange.Redis and NRedisStack libraries.

Documentation: Official Documentation [7].

Docker

Source: Open source, maintained by Docker Inc.

Version: Docker 24.0.5 or newer.

Purpose: Containerization technology to package GoatCapella's applications with their dependencies, ensuring consistency.

Communication: Docker CLI.

Documentation: Official Documentation [8].

Docker Compose

Source: Open source, maintained by Docker Inc.

Version: Docker Compose 2.30 or newer.

Purpose: Multi-container application management from one configuration file.

Communication: YAML configuration file.

Documentation: Official Documentation [9].

2.1.4. Communication Interfaces

The Communication Interfaces section describes the mechanisms, standards, and technologies that enable communication between the components of the GoatCapella platform and its external dependencies. These interfaces facilitate secure data exchange, maintain system performance, and ensure compatibility between systems. GoatCapella's communication design is critical to supporting key functionalities such as challenge creation, participant interactions, API integrations with third-party coding platforms, and data synchronization.

The communication framework is built on protocols and standards to ensure reliability, scalability, and security. It emphasizes the use of modern authentication and authorization mechanisms, enabled by Keycloak, which implements OpenID Connect for authentication and OAuth 2.0 for authorization. This ensures an identity management system for both internal and external communications. The following are the key types of communication interfaces employed in GoatCapella:

1. API Communication

GoatCapella integrates with multiple third-party coding platforms using public APIs to fetch challenges, track user progress, and manage challenge outcomes.

- **Approach:** REST is primarily used for API interactions due to its simplicity and widespread adoption. For platforms offering advanced querying capabilities, GraphQL is utilized to fetch precise data efficiently.
- **Security:** Interactions with external APIs are secured using OAuth 2.0, which provides token-based authorization for secure and controlled access.
- **Data Format:** JSON is the standard format for data exchange, ensuring ease of use and compatibility across different systems.

2. Client-Server Communication:

This interface supports communication between users and the GoatCapella backend, ensuring interaction and feature accessibility.

- **Protocols:** HTTP/HTTPS is used for secure and standardized web-based interactions between clients and the server.
- **Authentication and Authorization:** GoatCapella employs OpenID Connect for user authentication and OAuth 2.0 for access control, both implemented via Keycloak to provide seamless and secure identity management.

3. Internal Subsystem Communication:

GoatCapella's internal architecture consists of modular components that communicate with one another to handle user requests, manage data, and execute challenge-related processes.

- **Approach:** REST APIs are used for efficient inter-service communication between the platform's internal modules.
- **Authentication:** Inter-service calls are secured with JSON Web Tokens (JWT) issued by Keycloak, ensuring that only authenticated services can interact with critical resources.
- **Data Caching and Optimization:** Redis is used as a caching layer to optimize frequently accessed data and reduce the load on the main database.

2.2. Product Functions

Functional Req. ID #	Functional Requirement Name	Functional Requirement Description
FR 1	Register Page	The system must allow new users to register by providing necessary information such as username, email, and password.
FR 2	Log-in Page	The system must allow existing users to log into their accounts using their credentials.
FR 3	Home Page	The system must display the landing page with ongoing challenges for both authenticated and guest users.
FR 4	Selected Challenge Page	The system must allow authenticated users to view details of selected challenges, track progress, and interact with others.

FR 5	Challenge Creation Page	Authenticated users must be able to create new challenges by providing challenge details and configuring challenge settings.
FR 6	Shop Page	Authenticated users must be able to purchase items, such as boosters, to assist them with challenges.
FR 7	Profile Page	The system must allow authenticated users to view their completed challenges, ongoing challenges, rank, badges, and purchased items. Users should also be able to update their profile information.
FR 8	Admin Page (GoatCapella)	The system must allow administrators to manage application-level settings, such as configuring shop items, user roles, and challenge management.
FR 9	AI Challenge Creation	The system must allow the AI to create personalized challenges for users based on their data, preferences, and previous challenges.
FR 10	Challenge Participation	The system must allow users to participate in challenges, with options to comment on solutions, track progress, and withdraw if needed.
FR 11	Sub-Admin Management	The system must allow sub-admins to manage challenge details and configurations based on permissions granted by the main admin.
FR 12	User Progress Tracking	The system must track user progress across challenges and display the status of ongoing challenges.
FR 13	Challenge Commenting	The system must allow users to comment on challenges and other users' solutions if they have participated in the challenge.

FR 14	User Withdrawal	The system must allow users to withdraw from a challenge if they no longer wish to participate.

2.3. User Characteristics

This section describes the general characteristics of the intended users of the GoatCapella application. Understanding these characteristics is essential for tailoring the application's features and interface to meet the specific needs and preferences of its user base.

- **Competitive Challengers:** These individuals thrive in competitive environments and are motivated by constant self-testing and challenging. Various challenges and competitions encourage their participation by continuously testing their skills and pushing their limits.
- **Casual Participants:** They prefer to engage in challenges that are easy to understand and are designed for fun and personal enjoyment. They may also seek social interactions through these more relaxed and instructional challenges.
- **Users with an Interest and Basic Knowledge in Coding:** These users have an interest in coding and possess basic programming skills. However, using the application does not require high levels of technological knowledge, allowing a broader audience to access and interact with the application effectively.

2.4. Constraints

The Constraints section identifies the limitations and restrictions that influence the development, deployment, and operation of GoatCapella. These constraints stem from technical challenges, operational limitations, external dependencies, and compliance requirements.

1. **Technical Constraints:**

Third-Party API Dependency:

GoatCapella integrates with external coding platforms (e.g., LeetCode, CodeWars) through public APIs to fetch challenge data and user progress. However:

- Limited or discontinued API access might reduce the platform's ability to offer consistent services.

- Differences in API structures and capabilities across platforms add complexity to the integration process.

System Performance:

- Inefficient handling of high user concurrency could lead to slow response times or system crashes.

Scalability and Infrastructure:

- GoatCapella must handle an increasing number of users, challenges, and concurrent activities as it grows. A lack of scalable infrastructure may lead to bottlenecks.

Authentication and Authorization:

- Integration with Keycloak for identity management introduces complexity, requiring proper configuration of roles, permissions, and secure session management.

2. **Operational Constraints:**

Budgetary Limitations:

- The project operates within a constrained budget, limiting access to premium services such as advanced APIs, cloud resources, and enterprise-grade tools.
- Open-source technologies and free-tier services must be prioritized, potentially requiring trade-offs in performance or features.

Time Constraints:

- The development timeline is bound by the academic term, restricting the time available for feature development, rigorous testing, and iteration.

Hardware Resource Constraints:

- Initial development and deployment will rely on limited hardware, which may not support extensive traffic or computationally intensive tasks.

3. **Legal and Compliance Constraints**

API Usage Policies:

- Interactions with external platforms must comply with their terms of service to avoid potential legal issues or service bans.

4. Security Constraints

Sensitive Data Handling:

- All user data, including challenge progress and credentials, must be encrypted during transmission and storage.

Authentication and Token Management:

- Secure generation, storage, and validation of JWT tokens are critical to prevent unauthorized access.

2.5. Assumptions and Dependencies

Assumptions

- **Availability of Third-Party APIs:** It is assumed that the external APIs for platforms like LeetCode, CodeWars, and others will remain available and accessible during the course of the project. The system assumes that these APIs will continue to provide challenge data and user progress tracking without significant changes or disruptions.
- **Infrastructure Scalability:** It is assumed that the infrastructure (both on-premise or cloud) will scale effectively to handle increasing numbers of users and challenges as the system grows [10].
- **Authentication System:** It is assumed that the Keycloak integration for identity management and role-based access control will function without significant issues, and that users will be able to log in securely with minimal disruptions.

Dependencies:

- **Third-Party APIs**: The system depends on the external APIs from coding platforms (e.g., LeetCode, CodeWars) for fetching challenge data, tracking user progress, and fetching results.
- **Keycloak for Authentication**: The platform depends on Keycloak for secure authentication and authorization. Any changes or interruptions in Keycloak services, such as server downtime or configuration issues, will impact the system's ability to authenticate users [11].

- **Database Technology:** The system relies on specific database technologies (e.g., SQL Server or PostgreSQL) for storing user data and challenge information. Changes in database systems or configurations could require significant changes to the data access layer.
- **Open-Source Technologies:** The system depends on several open-source technologies for core functionality (e.g., Dapper for data access, React for the frontend). Any changes in the support or availability of these technologies may require adapting or replacing them.

3. Specific requirements

3.1. External Interfaces

This section includes the external platform interfaces that the system communicates with. Regarding each interface, name, source, version, purpose and communication methods are detailed along with references to documentations, highlighting their roles within the platform.

LeetCode API

Source: Proprietary, maintained by LeetCode.

Version: Not publicly documented.

Purpose: Retrieving available achievements, achievements of a specific user and solutions of a user to acquire these achievements.

Communication: HTTPS requests using HttpClient in ASP.NET Core

Documentation: Unofficial API documentation [12].

CodeWars API

Source: Public API, maintained by CodeWars.

Version: API version 1.0.

Purpose: Retrieving available achievements and achievements of a specific user.

Communication: HTTPS requests using HttpClient in ASP.NET Core.

Documentation: Official API Documentation [13].

CodeForces API

Source: Public API, maintained by CodeForces.

Version: Not publicly documented.

Purpose: Retrieving available achievements and achievements of a specific user.

Communication: HTTPS requests using HttpClient in ASP.NET Core.

Documentation: Official API Documentation [14].

3.2. Functional Requirements

FR1 – Sign Up Page

Name	FR1 Sign-Up Page
Purpose/Description	This function allows new users to register for the platform by providing necessary information such as username, email, and password.
Inputs	Username, email, password, and consent to terms of service.
Processing	The system validates the input data, creates a new user account, and stores the information in the database.
Outputs	Successful registration message and redirect to the login page or error message if validation fails.

FR2 – Log in Page

Name	FR2 Log-in Page

Purpose/Description	This function allows existing users to log into their accounts by providing credentials (username and password).
Inputs	Username and password.
Processing	The system validates the provided credentials and grants access if they match.
Outputs	Authentication success (access to the platform) or error message if credentials are incorrect.

FR3 – Home Page

Name	FR3 Home Page
Purpose/Description	This function displays the landing page for all users, showcasing ongoing challenges. Both authenticated users and guest users can view the challenges available on the platform.
Inputs	None (only the system's challenge data is displayed).
Processing	The system fetches the list of ongoing challenges and displays them to users.
Outputs	A list of ongoing challenges that users can interact with, with options to view or participate depending on the user type.

FR4 – Selected Challenge Page

Name	FR4 Selected Challenge Page
Purpose/Description	This function allows authenticated users to view the details of a selected challenge, track their progress, and interact with other users' solutions. Users can comment on solutions if they have participated, withdraw from the challenge, or see challenge status. Challenge creators and sub-admins have additional management capabilities.
Inputs	Challenge participation status, user actions (comment, withdraw).
Processing	The system retrieves challenge details, user progress, and comment history. It checks if the user is allowed to comment or withdraw based on participation status.
Outputs	Challenge details, user comments, solution status, options to participate or withdraw, and management options for admins and sub-admins.

FR5– Challenge Creation Page

Name	FR5 Challenge Creation Page
Purpose/Description	This function allows authenticated users to create new challenges by providing details

	such as the challenge name, description, and rules. Once created, the user becomes the administrator of the challenge.
Inputs	Challenge name, description, rules, and configuration details.
Processing	The system validates the challenge details and saves them in the database. The user is assigned as the administrator of the challenge.
Outputs	A new challenge created and visible to users for participation, with the creator having administrative control.

FR6– Shop Page

Name	FR6 Shop Page
Purpose/Description	This function enables authenticated users to purchase items such as boosters, which can help them progress in their challenges.
Inputs	Item selection, payment details.
Processing	The system processes the payment for the selected items, deducts the user's balance, and adds the purchased items to their profile.
Outputs	Confirmation of the purchase, updated user profile with new items, and boosters.

FR7– Profile Page

Name	FR7 Profile Page
Purpose/Description	This function allows authenticated users to view their completed challenges, ongoing challenges, rank, badges, and purchased shop items. Users can also update their personal information on this page.
Inputs	User's profile data and personal information updates.
Processing	The system retrieves the user's profile data and completed challenge information. It updates the profile when new information is submitted.
Outputs	A user profile with challenge history, rank, badges, and personal information, along with options to update the profile.

FR8– Admin Page (GoatCapella)

Name	FR8 Admin Page (GoatCapella)
Purpose/Description	This function allows administrators to manage application-level settings, including shop item configuration, user roles, and challenge management.
Inputs	Admin actions for configuration (add/edit/remove items, manage user roles).

Processing	The system processes admin actions, updates configurations, and manages user roles and challenge settings.
Outputs	Updated configurations, user role assignments, and challenge management updates.

FR9 – AI Challenge Creation

Name	FR9 AI Challenge Creation
Purpose/Description	This function allows the AI system to create personal challenges for users based on their data, such as previous challenges and user preferences.
Inputs	User data, challenge creation criteria.
Processing	The AI system analyzes the user's data, identifies patterns, and generates customized challenges.
Outputs	A new personalized challenge that is available for the user to participate in.

3.3. Non-Functional Requirements

3.3.1. Security

- **Data Protection:** The GoatCapella application adheres to the General Data Protection Regulation (GDPR) principles to ensure the protection of user data. GDPR, a law established in the European Union for the protection of personal data, includes fundamental principles such as data transparency, minimization, and security. The application collects

users' personal data only for specified and lawful purposes and takes all necessary technical and administrative measures to secure this data. This ensures that users have control over their data and that their information is processed securely [15].

- **Access Control:** For user authentication and authorization processes within the application, Keycloak will be used. Keycloak is an open-source identity and access management solution that offers extensive features for secure login procedures in applications. It is equipped with features such as Single Sign-On (SSO), social media login, two-factor authentication, and the ability for users to define their own access policies. Keycloak will be used to centrally manage access control and manage user permissions in the application [16].

3.3.2. Usability

GoatCapella application is designed to allow users to interact easily and effectively. Usability is a priority integrated into every aspect of the application, focusing on the following key features:

- **Easy Navigation**: An intuitive navigation structure is established to ensure that users can quickly and easily access the information they need. Menus and buttons are clearly labeled and placed in easily accessible locations.
- **Feedback and Support**: Users can report issues they encounter within the application and provide feedback. Additionally, features such as a Frequently Asked Questions (FAQ) section and live support are available to assist users.
- **Customization Options**: Users can adjust the settings within the application according to their personal preferences. This allows for personalization of the interface and tailoring the application to best suit their needs.

3.3.3. Maintainability

- **Modular and Layered Architecture:** The application will follow a modular architecture to separate concerns into independent, well-defined modules.
- **Readable and Well-Documented Code:** The project will adhere to widely accepted coding standards (e.g., C#/.NET conventions) to improve readability and consistency.
- **Version Control and Code Review:** All code changes will be version-controlled using GitHub, ensuring proper tracking and rollback capabilities if needed.

Testing:

- **Unit Testing:** Automated tests will be implemented for core functions to ensure they work as expected.
- **Integration Testing:** Testing will verify the seamless interaction between different modules.
- **End-to-End Testing:** Simulating real-world user scenarios to test the system holistically.

3.3.4. Compatibility

GoatCapella application is being developed using Docker. Docker allows applications to run in independent and portable containers. These containers provide a lightweight, portable, and isolated environment that contains everything needed for the application to function [8]. This ensures that the application can run consistently across different operating systems. Docker container technology allows the application to perform with the same functionality and efficiency on various platforms such as Windows, Linux, and macOS. This approach ensures that the application is accessible and usable by various systems.

3.4. Performance Requirements

The performance requirements of the system are defined as follows:

User Capacity

- **Peak Load**: The system shall support 2,000 concurrent users during peak times without significant performance degradation.
- **Normal Load**: The system shall handle 10,000 daily active users on average.

Response Time

- The system shall ensure that 90% of all API requests (e.g., challenge retrieval, grading submissions) respond within 1 second.
- User-facing pages (e.g., challenge creation, leaderboards) shall be loaded within 3 seconds on average under normal conditions.

Throughput

- The platform shall process at least 500 API requests per second during normal operations and scale up to 5,000 requests per second during peak usage.

Scalability

- The system shall scale horizontally to accommodate growth in users, data volume, and traffic spikes.
- Cloud-based auto-scaling mechanisms shall ensure resource allocation adjusts dynamically during peak loads.

Data Handling

- The database shall efficiently manage up to 10 million challenge records and 1 million user profiles, with scalability to support future growth.
- Caching (e.g., Redis) shall handle frequent queries such as user challenge history and leaderboard data to reduce database load and ensure sub-second response times.

Availability

- The platform shall maintain 99.9% uptime, allowing no more than 43 minutes of downtime per month.
- Critical systems, including user authentication and challenge submissions, shall be designed for high availability with failover mechanisms.

Reliability

- In the event of a system crash, the platform shall recover within 10 minutes, restoring the latest state with minimal disruption.

Monitoring and Optimization:

- Monthly performance audits shall ensure the system adheres to defined benchmarks and remains optimized as user demands evolve.

3.5. Logical Database Requirements

The logical design of the database is built on a flexible and extensible structure where all data entities are derived from a common Base Entity. This approach ensures consistency, scalability, and ease of management while allowing the system to accommodate diverse data models seamlessly. The system is designed to leverage both PostgreSQL and MongoDB as database solutions, enabling the use of relational and non-relational data storage where appropriate. Below are the key aspects of this design:

- **Logical Design Explanation:** The Base Entity serves as a foundation for all data models. Any class that implements the Base Entity can be persisted in the database. The Base Entity serves as a foundation for all data models. Any class that implements the Base Entity can be persisted in the database.
- **Migration and Schema Management:** Database schemas can be automatically generated based on classes derived from the Base Entity (e.g., using the Entity Framework Code-First approach).
- **Flexibility in Data Storage:** The system allows storing any class that implements the Base Entity, without being constrained by specific data types. Examples include:

  User: Stores user-related information (e.g., name, email, roles).

  Challenge: Stores information and configurations related to challenges.

  Participation: Tracks user participation in challenges.

  Reward: Stores user rewards and earning history.

  Shop Item: Represents items or features available for purchase

3.6. Design Constraints

- **Scalability Constraint:** The system architecture must be designed to scale up and down using Docker containers. Container orchestration must be achieved using Docker Swarm or Kubernetes, based on ease of setup and long-term scalability requirements.
- **API Rate Limitation Constraint:** External dependent platform APIs imposes a rate limit on API requests. For example, Codeforces limits its API to one request every 2 seconds. This limitation must be accounted for system design, especially for the challenge management and configuration. The system must ensure that it does not overload the external dependent platforms.
- **Open-Source Constraint:** The system must exclusively utilize open-source software (excluding any external dependent platforms).
- **Cross Platform Compatibility Constraint:** The system must be cross platform compatible, supporting both Linux (Ubuntu) and Windows environments. This constraint ensures that the system can be deployed in various configurations, including production environments (where Linux is preferred) and local development environments (where windows may be used).
- **Technology Constraint:** The system must utilize the following technologies for their respective functionalities: Redis for caching; Keycloak for authentication, authorization and

identity management; PostgreSQL and MongoDB for structured and unstructured data storing, respectively.

- **Secure Communication Constraint:** All communication between the system and external services must be conducted over secure protocols. HTTPS will be enforced for all external API requests and user interfaces.
- **Data Privacy Constraint:** The system must comply with the General Data Protection Regulation (GDPR) to ensure the protection of user data and uphold the privacy rights of individuals.

3.7. Software System Attributes

The system will be evaluated based on five different software system attributes. For every attribute, detailed requirements will be outlined to define the expected standards with corresponding verification methods to ensure these standards are objectively assessed and maintained.

Reliability

Requirements:

- The system must handle 99.9% of operations without errors.
- The system must decrease functionality and maintain consistency when external dependencies (e.g. Codeforces) are unavailable.

Verifications:

- Stress testing using automated tools to simulate high traffic and evaluate system responses.
- Analyzing logs over a large period (3 months).
- Simulating external API downtime to ensure the system remains consistent.

Availability

Requirements:

- The system must ensure 99.9% availability, which equates to no more than 43.8 minutes of downtime per month.
- Automated failover mechanisms for critical services.

Verifications:

- Testing failover mechanisms by simulating server or container crashes.

- Analyzing logs over a large period (3 months).

Security

Requirements:

- All logs must be maintained for the next 90 days.
- Keycloak must be used for secure authentication, authorization and identity management.

Verifications:

- Analyzing logs over a large period (3 months).

Maintainability

Requirements:

- The system must adopt a modular architecture with well-defined and documented interfaces between components.
- Detailed documentation for APIs, configuration settings, and deployment processes.

Verifications:

- Evaluating the ease of modifying components during simulated maintenance tasks.
- Reviewing documentation to ensure it is clear and comprehensive.

Portability

Requirements:

- Compatibility with both Linux and Windows servers.
- Independence from platform-specific features in the application code.

Verifications:

- Deploying the system in two distinct environments (Ubuntu Linux and Windows).
- Analyzing the codebase to confirm the absence of platform-specific features.

4. Use Cases

Authenticated User Use Case

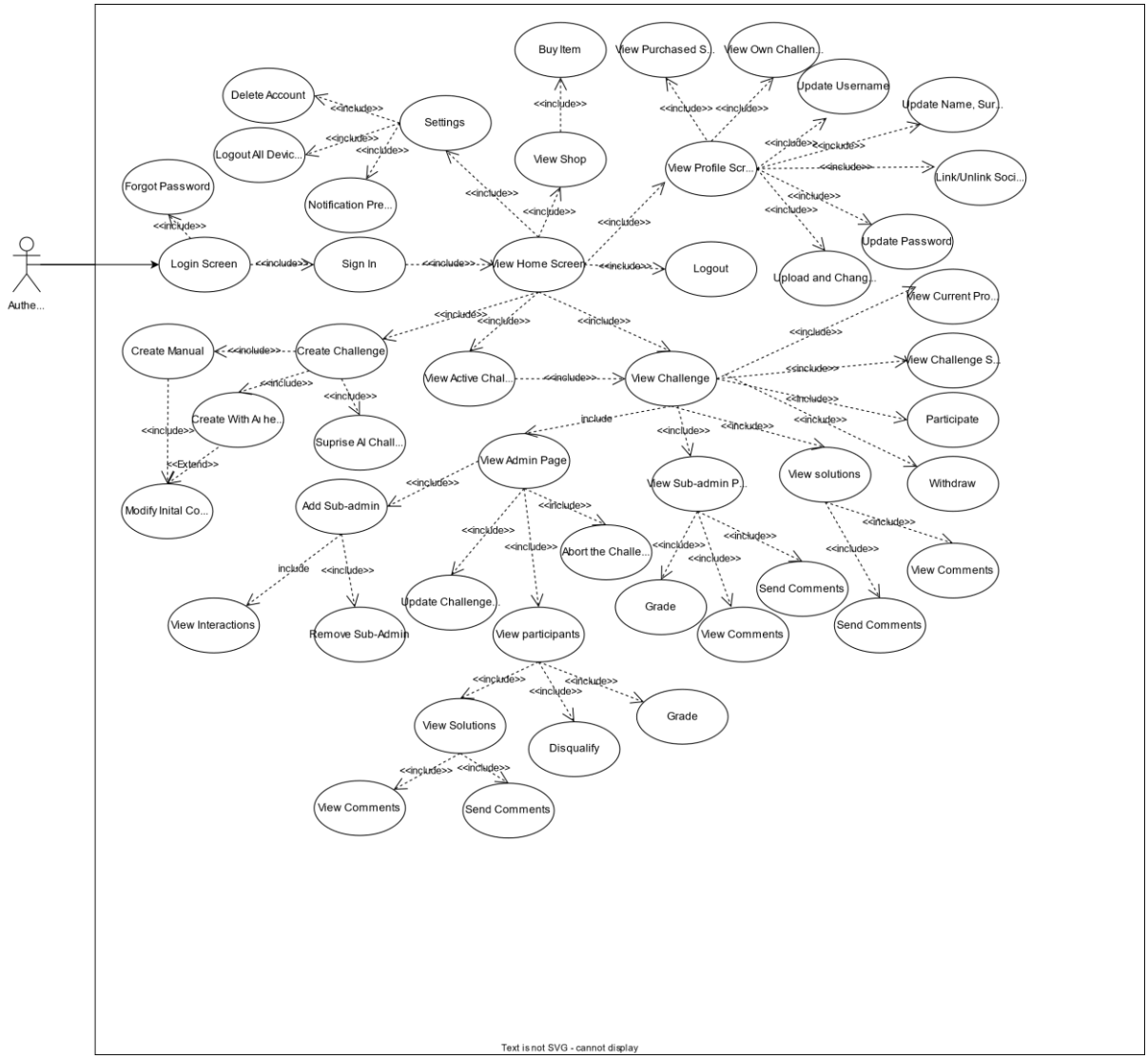

Figure 1 Use Case Diagram of Authenticated User

Use Case Id	1
Use Case Name	Login Screen
Actor	User
Description	The interface that greets the unlogged user when the user first opens the application.

Precondition	User must have a registered account in the application.
Related Use Cases	
Postcondition	User is successfully directed to the related page.
Main Flow	1. User opens the application 2. User chooses from Login or Forgot Password options. 3. System successfully directs the user to the related page.
Alternate Flows	None.

Use Case Id	2
Use Case Name	Sign In
Actor	Authenticated User
Description	Allows user to log in to the application
Precondition	User must have a registered account in the application.
Related Use Cases	1,3
Postcondition	User is successfully logged into the application.
Main Flow	1. User opens the applications. 2. Users input their login credentials. 3. User clicks on the login button.

	4. If the credentials are correct, system successfully directs the user to the related page.
Alternate Flows	A. If user clicks on "Forgot Password", redirect to password reset flow.

Use Case Id	3
Use Case Name	Forgot Password
Actor	Authenticated User
Description	Allows the user to reset their password if they have forgotten it, by verifying their identity and creating a new password.
Precondition	User must have a registered account in the application.
Related Use Cases	1
Postcondition	User successfully resets their password and can log in with the new password.
Main Flow	1. User selects the "Forgot Password" option. 2. The system prompts the user to enter their registered email address. 3. Users submit their email. 4. The system sends a password reset link or code to the provided email. 5. User clicks the reset link or enters the code to access the password reset page. 6. User creates a new password and submits it. 7. The system updates the user's password and displays a success message.

Alternate Flows	A. If the user enters an unregistered email, the system displays an error message. B. If the reset link or code expires, the system prompts the user to request a new one. C. If the new password does not meet validation criteria (e.g., too short), the system displays an error and asks for correction.

Use Case Id	4
Use Case Name	Home Page
Actor	Authenticated User
Description	It is the main interface where the user can access various features after logging in
Precondition	User must be logged in to the application.
Related Use Cases	1,2
Postcondition	Users successfully navigate to the desired feature or action from the home page.
Main Flow	1. User opens the application and logs in. 2. The system directs the user to the home page. 3. User selects an option (e.g., Profile, Challenges, Shop, Settings). 4. The system navigates the user to the corresponding feature or interface.
Alternate Flows	None.

Use Case Id	5
Use Case Name	View Challenge
Actor	Authenticated User
Description	Allows the user to browse and view details of available challenges, including their status, progress, and participants.
Precondition	Users must be logged in to the application.
Related Use Cases	4
Postcondition	User successfully views the challenge details.
Main Flow	1. User navigates to the Home Page. 2. User selects the "View Challenges" option. 3. The system displays a list of available challenges. 4. User selects a challenge to view its details. 5. The system displays detailed information about the selected challenge (e.g., description, status, participants).
Alternate Flows	A. If there are no available challenges, the system displays a message such as "No challenges available at the moment."

Use Case Id	6
Use Case Name	Create Challenge
Actor	Authenticated User

Description	Allows the user to create a new challenge by specifying its details.
Precondition	Users must be logged in to the application.
Related Use Cases	4
Postcondition	A new challenge is successfully created and added to the list of available challenges.
Main Flow	1. User navigates to the Home Page. 2. User selects the "Create Challenge" option. 3. System displays a form for entering challenge details (e.g. title, description, rules, start/end dates). The user can create the settings AI can produce them for user, or user can specify a few specific requests and leave the rest to AI. 4. User/AI determines the title and description, selects start/end dates, filters and selects challenge topics, specifies who can participate (public/private), sets rewards and scoring, selects the difficulty level, and the sub-admin can also make adjustments or additions. 5. User fills in the required fields and submits the form. 6. The system validates the inputs and creates the challenge. 7. The user receives a confirmation message that the challenge has been successfully created.
Alternate Flows	A. If mandatory fields are left empty, the system prompts the user to complete them. B. If the challenge details do not meet validation criteria (e.g., invalid dates), the system displays an appropriate error message.

Use Case Id	7

Use Case Name	Admin page
Actor	Authenticated User
Description	The admin page allows the platform admin to manage challenges and assign sub-admins who can assist.
Precondition	At least one challenge must be created.
Related Use Cases	5,6
Postcondition	Admin successfully manages challenges and assigns sub-admins to assist in moderation and management tasks.
Main Flow	1. Admin can only view the challenge they created. They cannot perform actions on other challenges. 2. Admin can modify the title, description, and rewards of the challenge they created. 3. Admin can delete the challenge they created, which removes it permanently from the system. 4. Admin can assign a sub-admin to the challenge they created. Sub-admins can work with the admin in managing the challenge. 5. Admin can view the users participating in the challenge they created and the questions they have solved. 6. Admin can comment on the solutions submitted by participants. 7. Admin can view comments on participants' solutions. 8. Admin can grade the solutions submitted by participants. 9. Admin can disqualify participants from the challenge based on their performance or actions. 10. Admin can end the challenge, marking it as completed. 11. Admin can cancel the challenge, removing it from active challenges.

Alternate Flows	A. If the admin tries to modify the details of a challenge that has already been marked as completed, the system displays an error message indicating that modifications are not allowed for completed challenges. B. The system requires the admin to view the solution before assigning a grade and displays a message prompting them to review the solution.

Use Case Id	8
Use Case Name	Sub-Admin Page
Actor	Authenticated User
Description	Sub-admin can only comment on, read comments, and rate solutions for challenges they are assigned to.
Precondition	The sub-admin must be logged in and assigned to a challenge.
Related Use Cases	5,6,7
Postcondition	Sub-admin has commented on, rated, and read comments on the challenge solutions.
Main Flow	1. Sub-admin views the challenge assigned to them. 2. Sub-admin reads the solutions submitted by participants. 3. Sub-admin comments on the solutions. 4. Sub-admin rates the solutions submitted by participants. 5. Sub-admin reads comments made by participants.
Alternate Flows	A. If the sub-admin makes an incorrect comment, the admin can correct it.

Use Case Id	9
Use Case Name	Logout
Actor	Authenticated User
Description	This use case represents the action of logging out from the system, ending the user's session.
Precondition	User must be logged in to the application.
Related Use Cases	2
Postcondition	The user is logged out, and the session is terminated.
Main Flow	1. User clicks on the "Logout" button. 2. The system processes the logout request. 3. System ends the user's session and redirects to the login page.
Alternate Flows	A. If the user's session expires, they are automatically logged out and redirected to the login page.

Use Case Id	10
Use Case Name	Profile Screen
Actor	Authenticated User
Description	This use case allows the user to view and update their profile information, such as name, email, password, and profile picture.
Precondition	The user must be logged into the system.
Related Use Cases	2,4
Postcondition	The user's profile is updated successfully, or changes are discarded.

Main Flow	1. User navigates to the "Profile" section. 2. The system displays the user's current profile details. 3. User updates the desired information (e.g., name, email, password, or profile picture). 4. User can add/remove social media links 5. User saves the changes. 6. The system validates the inputs and updates the profile details. 7. The system confirms the successful update to the user.
Alternate Flows	A. If the user provides invalid inputs (e.g., incorrect email format), the system shows an error message and prompts the user to correct it. B. If the user cancels the update, no changes are saved, and the profile remains unchanged. C. If a technical issue occurs, the system notifies the user and retains the existing profile information.

Use Case Id	11
Use Case Name	View shop
Actor	Authenticated User
Description	This use case allows the user to browse and purchase items in the shop using app coins as the only currency.
Precondition	The user must be logged into the system and have a sufficient balance of app coins.
Related Use Cases	2
Postcondition	The user successfully purchases an item, and the app coin balance is updated.

Main Flow	1. User successfully purchases an item, and the app coin balance is updated. 2. The system displays the list of available items, including their names, descriptions, prices in app coins, and stock availability. 3. User selects an item to view more details. 4. User chooses to purchase the item using app coins. 5. System verifies that the user has enough app coins. 6. If sufficient coins are available, the system deducts the item's cost from the user's app coin balance. 7. System confirms the successful purchase and updates the inventory and coin balance.
Alternate Flows	A. If the user does not have enough app funds, the system displays an error message B. If the selected item is out of stock, the system notifies the user and suggests browsing other items. C. If an error occurs during the transaction, the system cancels the purchase, restores the user's coin balance, and notifies the user to try again later.

Use Case Id	12
Use Case Name	Settings
Actor	Authenticated User
Description	This use case allows the user to manage account settings, including deleting their account, logging out from all devices, and updating notification preferences.
Precondition	The user must be logged into the system.

Related Use Cases	
Postcondition	The user's account settings are updated or left unchanged if no action is taken.
Main Flow	1. User navigates to the "Settings" section. 2. The system displays options: Delete Account, Logout All Devices, and Notification 3. User selects "Delete Account". 4. The system asks for confirmation. 5. If confirmed, the account is permanently deleted. 6. User selects "Logout All Devices". 7. The system logs the user out from all devices and confirms the action. 8. User selects "Notification Preferences". 9. The system displays available notification options. 10. User enables or disables notifications as desired. 11. User saves the changes, and the system updates the settings.
Alternate Flows	A. If the user cancels the deletion process, the account remains active. B. If the system encounters an error, the user is notified, and the action is not completed. C. If the user enters an invalid input, the system prompts for corrections before saving changes. D. If the user cancels changes, the notification settings remain unchanged.

Guest User Use Case

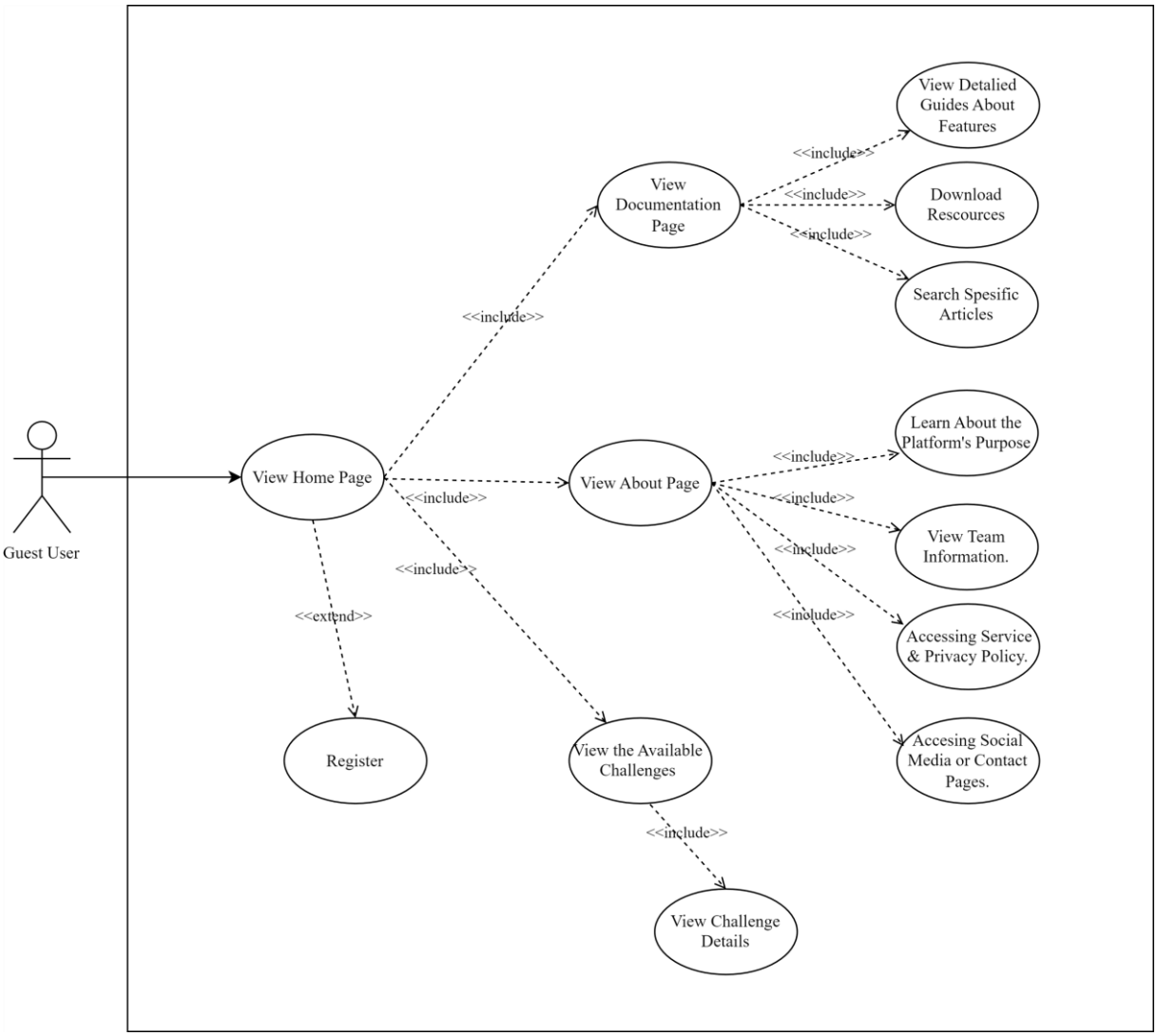

Figure 2 Use Case Diagram of Guest User

Use Case Id	13
Use Case Name	View Home page
Actor	Guest User
Description	Allow the user to view the main page of the platform.
Precondition	Users are accessing the platform.

Related Use Cases	None.
Postcondition	Users see the home page.
Main Flow	1. User opens the platform. 2. The system displays the home page.
Alternate Flows	None.

Use Case Id	14
Use Case Name	View About Page
Actor	Guest User
Description	Allow the user to view information about the platform.
Precondition	Users are accessing the platform.
Related Use Cases	13
Postcondition	User sees the about page.
Main Flow	1. User clicks on the "About" link. 2. System displays the about page
Alternate Flows	A. If "About" link is not available, display an error message.

Use Case Id	15
Use Case Name	View Available Challenges

Actor	Guest User
Description	Allows the user to view a list of available challenges on the platform.
Precondition	User is accessing the platform
Related Use Cases	13
Postcondition	User sees the list of available challenges.
Main Flow	1. User navigates to the "Challenges" section. System displays the list of available challenges.
Alternate Flows	A. If there are no available challenges, display a message indicating that no challenges are currently available.

Use Case Id	16
Use Case Name	View Documentation Page
Actor	Guest User
Description	Allow the user to view the documentation for the platform.
Precondition	Users are accessing the platform.
Related Use Cases	13
Postcondition	Users see the documentation page.
Main Flow	1. User clicks on the "Documentation" link. 2. System displays the documentation page.

Alternate Flows	A. If the "Documentation" link is not available, display an error message.

Use Case Id	17
Use Case Name	Register
Actor	Guest User
Description	Allows a guest user to register and become a registered user of the system.
Precondition	The user is accessing the platform as a guest.
Related Use Cases	13
Postcondition	The user is registered the system
Main Flow	1. The guest user clicks on the "Register" button on the platform. 2. The system displays the registration form. 3. The user fills in the required details 4. The system validates the provided details.
Alternate Flows	A. If the guest user chooses not to register, they remain as a "Guest User" and can access limited features such as the home page.

Admin Use Case

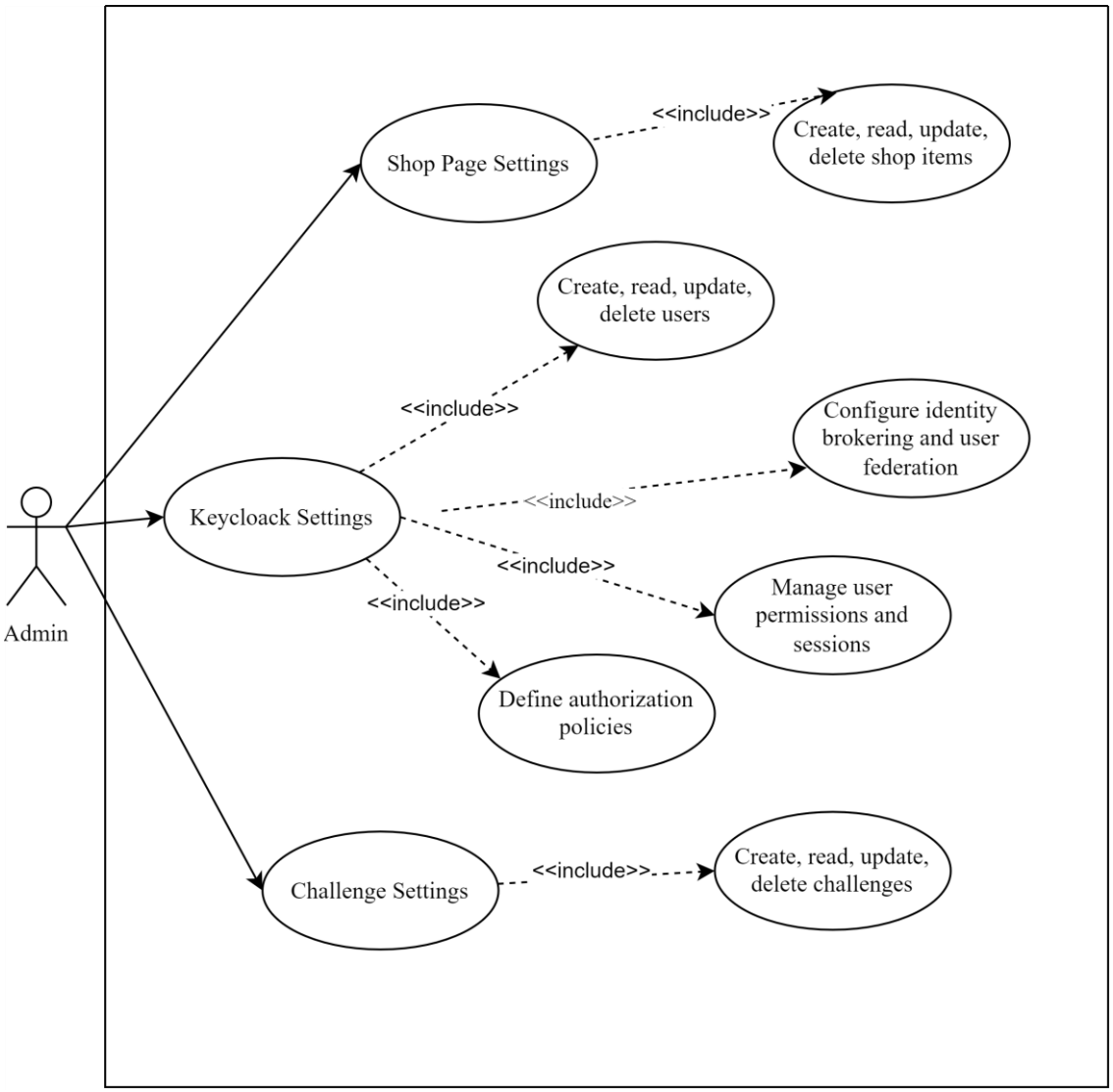

Figure 3 Use Case Diagram of Admin

Use Case Id	18
Use Case Name	Manage Shop Page Settings
Actor	Admin
Description	Admin can create, read, update, or delete items in the shop page settings to customize the platform's store offerings.
Precondition	The admin is logged into the system with proper authorization.

Related Use Cases	None
Postcondition	Shop items are successfully added, updated, or removed, and changes reflect on the user-facing shop page.
Main Flow	1. Admin navigates to the "Shop Page Settings" section. 2. Admin views the list of existing shop items. 3. Admin selects to create, edit, or delete a shop item. 4. System updates the shop settings based on admin actions.
Alternate Flows	A. If an admin tries to delete an item that doesn't exist, the system notifies the admin and logs the error.

Use Case Id	19
Use Case Name	Keycloak Settings Management
Actor	Admin
Description	Admin configures identity brokering, user federation, and user permissions through Keycloak integration.
Precondition	The Keycloak server is running and configured.
Related Use Cases	
Postcondition	Keycloak settings are successfully updated, ensuring user authentication and authorization work as intended.
Main Flow	1. Admin navigates to the "Keycloak Settings" section.

	2. The admin modifies the necessary settings using the Keycloak Admin Panel and submits the changes. 3. System updates the Keycloak server with the new configuration.
Alternate Flows	A. If the Keycloak server is unavailable, the system notifies the admin and retries when the server is online.

Use Case Id	20
Use Case Name	Manage Challenges
Actor	Admin
Description	Admin can create, read, update, or delete challenges to maintain the system's database of tasks for users.
Precondition	The admin has proper authorization and the database is accessible.
Related Use Cases	None
Postcondition	Challenges are updated and accessible for users.
Main Flow	1. Admin navigates to the "Challenge Settings" section. 2. Admin views the list of current challenges. 3. Admin selects to create, edit, or delete a challenge. 4. The system processes the changes and updates the database.
Alternate Flows	A. If the admin tries to update a challenge that is locked, the system prevents the update and displays an error message.

AI Agent Use Case

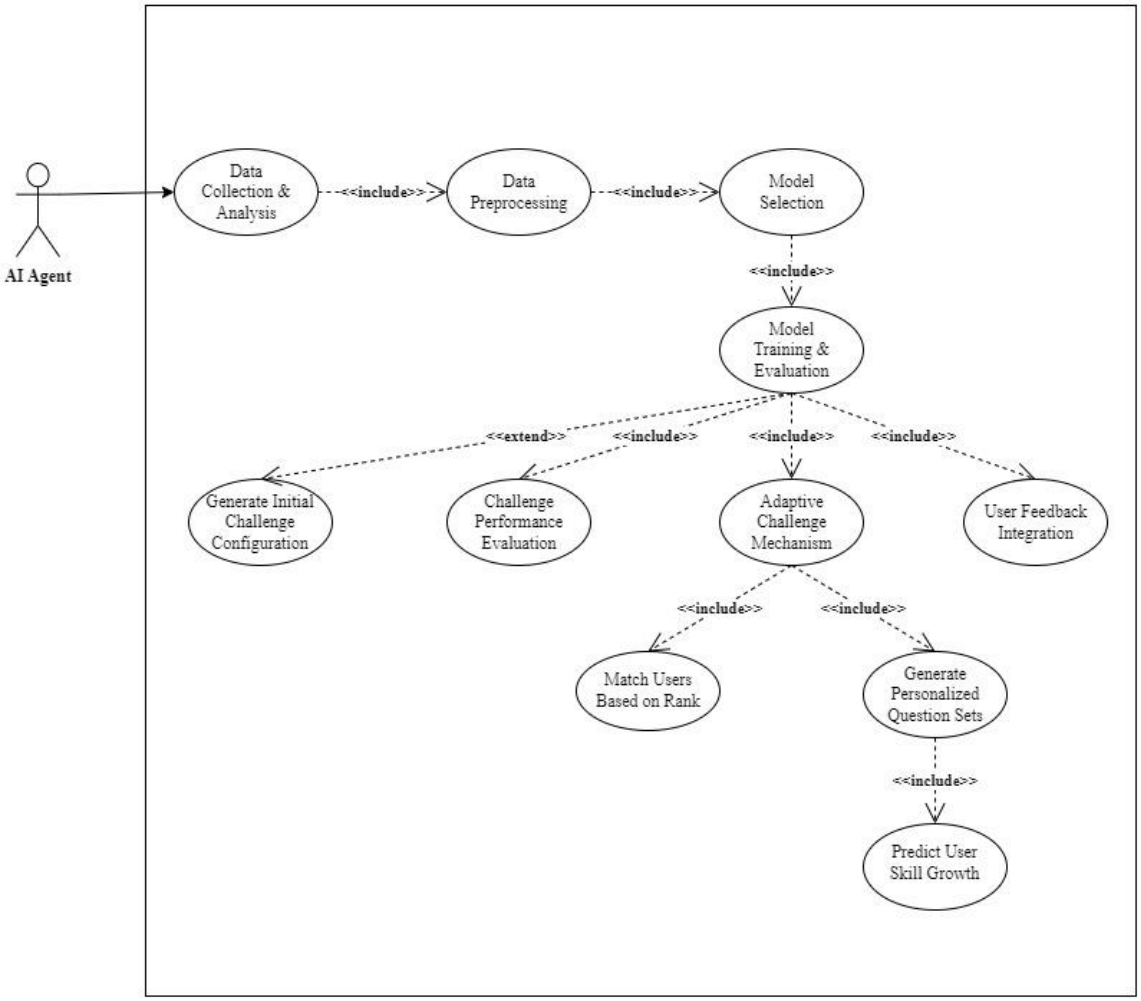

Figure 4 Use Case Diagram of AI Agent

Use Case Id	21
Use Case Name	Data Collection & Analysis
Actor	AI Agent
Description	Collects user performance data to build a profile for adaptive learning and challenge generation.
Precondition	User data and interaction metrics are accessible.

Related Use Cases	22
Postcondition	User data is collected and stored in a structured format.
Main Flow	1. The AI Agent collects detailed data from user interactions, including solving time, accuracy rates, and question difficulty, to build a comprehensive performance profile. 2. The system calculates aggregate metrics such as the total number of questions solved, regularity of participation, and performance trends to provide meaningful insights. 3. The collected data is securely stored in a structured format, ensuring it is ready for preprocessing in subsequent steps.
Alternate Flows	None

Use Case Id	22
Use Case Name	Data Preprocessing
Actor	AI Agent
Description	Processes raw data into a suitable format for AI model consumption.
Precondition	Data has been collected from users.
Related Use Cases	21,23
Postcondition	Data is cleaned, normalized, and labeled for further processing.

Main Flow	1. The AI Agent addresses missing data, such as skipped or unanswered questions, to ensure the dataset is as complete and reliable as possible for analysis. 2. The system normalizes continuous variables, such as solving time, by scaling them to a consistent range (e.g., 0–1), ensuring uniformity and comparability across different data points. 3. The AI assigns difficulty labels (e.g., easy, medium, hard) to each question based on predefined criteria and calculates additional features, such as the average solving time per difficulty level and the accuracy ratio, to enrich the dataset for model training.
Alternate Flows	A. If data normalization fails, the system automatically retries the process with adjusted thresholds to achieve consistency. B. If the proportion of missing data exceeds an acceptable threshold, the AI skips the affected data batch and logs the issue for further review.

Use Case Id	23
Use Case Name	Model Selection
Actor	AI Agent
Description	Selects the appropriate algorithm for personalized challenge generation.
Precondition	Preprocessed data is available.
Related Use Cases	22, 24

Postcondition	The suitable algorithm is selected and configured, ensuring it is optimized and ready for the training phase to achieve accurate results.
Main Flow	1. The AI Agent evaluates the dataset's complexity by analyzing patterns, feature distributions, and relationships to determine the appropriate modeling approach. 2. Based on the dataset's characteristics, the system selects supervised learning models, such as Gradient Boosting, to accurately predict user performance and success rates. 3. Reinforcement learning models, like Deep Q-Networks, are chosen to dynamically adjust challenge parameters, such as difficulty levels, based on user feedback.
Alternate Flows	A. If the dataset complexity is low, the system defaults to simpler models, such as Logistic Regression, to efficiently handle the prediction task while maintaining accuracy.

Use Case Id	24
Use Case Name	Model Training & Evaluation
Actor	AI Agent
Description	Trains the selected models and evaluates their performance using historical data.
Precondition	Models are selected, and historical data is accessible.
Related Use Cases	23, 25, 26, 72, 28

Postcondition	Models are trained, evaluated, and ready for deployment.
Main Flow	1. The system divides the collected data into training and validation sets to ensure a balanced evaluation of the model's performance and prevent overfitting. 2. The AI Agent trains the model using the training set and evaluates its performance with metrics such as accuracy, precision, recall, and F1-score to measure its effectiveness. 3. Feedback mechanisms are employed to iteratively refine the model by incorporating insights from performance evaluations and user interactions.
Alternate Flows	A. If the model underperforms during evaluation, the system adjusts hyperparameters, such as learning rate or regularization strength, and retrains the model to improve accuracy. B. If the training data is insufficient, the system prioritizes collecting additional data from user interactions or historical records before initiating retraining to enhance model robustness.

Use Case Id	25
Use Case Name	Generate Initial Challenge Configuration
Actor	AI Agent
Description	Generates initial configuration settings for challenges based on user preferences and performance.
Precondition	Models are trained, and user data is available.

Related Use Cases	24
Postcondition	Initial challenge configurations, tailored to the user's preferences and performance data, are generated and suggested to the user.
Main Flow	1. The user initiates a request to create a new challenge, specifying basic requirements such as challenge type or desired outcomes. 2. The AI Agent analyzes the user's preferences, historical performance data, and activity patterns to tailor the challenge configuration to their needs. 3. Based on the analysis, the system suggests optimal configurations, including difficulty levels, a curated question set, and recommended time limits, ensuring the challenge aligns with the user's skill level and goals.
Alternate Flows	A. If user preferences are unavailable or incomplete, the system automatically generates configurations using default settings, ensuring a seamless experience. B. If the model's confidence in the suggested configuration is low, the system notifies the user and provides an option for manual adjustments to refine the challenge settings.

Use Case Id	26
Use Case Name	Challenge Performance Evaluation
Actor	AI Agent
Description	Analyzes user performance in challenges and provides feedback or insights.

Precondition	User performance data from challenges is accessible.
Related Use Cases	24, 30
Postcondition	Performance insights are logged and shared with the user.
Main Flow	1. The AI Agent retrieves detailed user performance metrics from completed challenges, including accuracy rates, response times, and question difficulty levels, to gain a comprehensive understanding of the user's performance. 2. The system analyzes the retrieved data to identify patterns, calculate success rates, and measure other critical metrics such as average solving time and performance consistency. 3. Based on the analysis, the AI Agent generates actionable feedback, highlighting strengths and areas for improvement, and logs these insights for further processing and potential integration into future challenge configurations.
Alternate Flows	A. If performance data is incomplete, the system proceeds with analyzing the available metrics, logs the missing data for tracking purposes, and notifies relevant modules for future data collection efforts.

Use Case Id	27
Use Case Name	Adaptive Challenge Mechanism
Actor	AI Agent

Description	Dynamically adjusts challenge settings based on user performance during participation.
Precondition	Models are trained, and challenges are configured.
Related Use Cases	24, 29, 30
Postcondition	Challenges are adjusted to align with the user's current skill levels, ensuring an optimal balance of difficulty and engagement for an effective learning experience.
Main Flow	1. The AI Agent continuously monitors user performance, tracking metrics such as accuracy, response time, and progress to assess the user's current skill level and engagement. 2. Based on the monitored data, the system dynamically adjusts challenge parameters, such as difficulty level, allotted time, or question type, to maintain an optimal balance between challenge and user capability. 3. The system provides immediate feedback, such as progress updates, hints, or motivational messages, to keep the user engaged and encourage continuous improvement.
Alternate Flows	A. If adjustments fail due to technical issues or insufficient data, the system falls back to preconfigured static challenge settings, ensuring a consistent user experience without interruptions.

Use Case Id	28
Use Case Name	User Feedback Integration
Actor	AI Agent

Description	Incorporates user feedback into the system to improve future challenges and recommendations.
Precondition	User feedback is collected and accessible.
Related Use Cases	24, 30
Postcondition	Feedback is processed and incorporated into system improvements.
Main Flow	1. The user provides feedback on challenges or question sets, sharing insights such as difficulty appropriateness, clarity of questions, or overall experience, to help refine the system. 2. The AI Agent processes the submitted feedback, integrates it with existing data, and updates relevant datasets to ensure the feedback is accurately reflected in future recommendations and challenge adjustments. 3. The system analyzes aggregated feedback trends and uses them to fine-tune challenge generation models, improving their alignment with user expectations and needs.
Alternate Flows	A. If the feedback is unclear or incomplete, the system prompts the user with specific questions or clarification requests to ensure the feedback can be effectively utilized. B. If the feedback conflicts with existing data or trends, the AI flags it for manual review by administrators or moderators to resolve discrepancies and ensure data consistency.

Use Case Id	29
Use Case Name	Match Users Based on Rank

Actor	AI Agent
Description	Matches users with appropriate opponents for competitive challenges based on rank and performance.
Precondition	Rank and performance data are available for all users.
Related Use Cases	21, 27
Postcondition	Users are matched with suitable opponents for the challenge.
Main Flow	1. The AI Agent retrieves rank and performance metrics, such as success rates, recent activity, and skill levels, for all active users to identify potential matches for competitive challenges. 2. The system calculates compatibility scores by comparing user metrics, ensuring that matches are fair and balanced based on similar skill levels and performance histories. 3. The AI Agent pairs users with the highest compatibility scores and sends invitations for the challenge, ensuring an engaging and competitive experience.
Alternate Flows	A. If no suitable match is found, the system notifies the user, provides a detailed explanation, and suggests alternative challenges or modes, such as solo practice or system-generated opponents. B. If a user declines the match invitation, the system dynamically retries by selecting the next best match from the pool of available users, ensuring minimal delay in initiating the challenge.

Use Case Id	30

Use Case Name	Generate Personalized Question Sets
Actor	AI Agent
Description	Creates question sets tailored to user strengths, weaknesses, and learning objectives.
Precondition	Models are trained, and user performance data is available.
Related Use Cases	27, 31
Postcondition	A personalized question set is generated and presented to the user.
Main Flow	1. The AI Agent analyzes the user's skill profile, including strengths, weaknesses, and recent performance trends, to create a detailed understanding of their learning needs and goals. 2. The system selects questions that match the user's current skill level and topic preferences, ensuring relevance and alignment with their learning path, while also considering question difficulty and variety. 3. The AI Agent assembles a personalized question set based on the selected criteria, ensuring a mix of appropriately challenging and engaging questions, and delivers it to the user.
Alternate Flows	A. If there is insufficient data for personalization, the system defaults to a curated set of general recommendations, providing a balanced question set suitable for most users. B. If the user specifies a preference for a particular topic or difficulty level, the system adjusts the selection process to filter questions, accordingly, tailoring the question set to the user's explicit request.

Use Case Id	31
Use Case Name	Predict User Skill Growth
Actor	AI Agent
Description	Forecasts user progress and recommends learning paths based on historical performance.
Precondition	Models are trained, and historical user data is available.
Related Use Cases	24, 30
Postcondition	User receives predictions and recommendations for improving their skills.
Main Flow	1. The AI Agent evaluates the user's historical performance data, including trends in accuracy, solving time, question difficulty, and consistency, to develop a comprehensive performance profile. 2. The system applies advanced predictive models to analyze the data and forecast the user's potential skill growth, identifying areas of improvement and expected progress over time. 3. Based on the predictions, the AI Agent recommends personalized learning paths and specific improvement strategies tailored to the user's strengths, weaknesses, and goals, helping them achieve measurable progress.
Alternate Flows	A. If historical data is insufficient for accurate predictions, the system provides general growth recommendations based on common patterns observed in similar user profiles, ensuring the user receives useful guidance.

	B. If the confidence level in the prediction is low, the system prioritizes additional data collection through ongoing challenges and user interactions to refine the forecast and enhance future recommendations.

References

[1] Ubuntu, “Ubuntu Server documentation,” 26 11 2024. [Online]. Available: https://ubuntu.com/server/docs.

[2] Windows, “Windows Server documentation,” 26 11 2024. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/.

[3] Microsoft, “ASP.NET Core 8.0 Documentation,” 24 11 2024. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0.

[4] “MongoDB Reference,” 24 11 2024. [Online]. Available: https://www.mongodb.com/docs/manual/reference/.

[5] PostgreSQL, “PostgreSQL Documentation,” 24 11 2024. [Online]. Available: https://www.postgresql.org/docs/17/index.html.

[6] Keycloak, “Keycloak Documentation,” 24 11 2024. [Online]. Available: https://www.keycloak.org/documentation.

[7] Redis, “NRedisStack guide (C#/.NET),” 26 11 2024. [Online]. Available: https://redis.io/docs/latest/develop/clients/dotnet/.

[8] Docker, “Docker Reference,” 26 11 2024. [Online]. Available: https://docs.docker.com/reference/.

[9] D. Compose, “Docker Compose,” 26 11 2024. [Online]. Available: https://docs.docker.com/compose/.

[10] QAT, “Writing Assumptions and Constraints”, 01 12 2024. [Online]. Available: https://qat.com/writing-assumptions-constraints-srs/

[11] Keycloak, “Securing Applications and Service Guide”, 01 12 2024. [Online]. Available: https://www.keycloak.org/docs/23.0.7/securing_apps/

[12] akarsh1995, “GitHub,” 26 11 2024. [Online]. Available: https://github.com/akarsh1995/leetcode-graphql-queries.

[13] CodeWars, “CodeWars Documentation,” 26 11 2024. [Online]. Available: https://dev.codewars.com/#introduction.

[14] CodeForces, “CodeForces API Documentation,” 26 11 2024. [Online]. Available: https://codeforces.com/apiHelp

[15] GDPR, “General Data Protection Regulation” 01 12 2024. [Online]. Available: https://gdpr-info.eu/art-5-gdpr/

[16] Keycloak, “Keycloak Documentation,” 25 11 2024. [Online]. Available: https://www.keycloak.org/documentation.