ÇANKAYA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

Project Report

CENG 407

Innovative System Design and Development I

#6
GoatCapella

Çağdaş GÜLEÇ
Tarık ŞEN
İrem Beyza AYDOĞAN
Şeref Berkay KAPTAN
Beyza TOZMAN

Serdar ARSLAN
Bilgin AVENOĞLU

Table of Contents

Introduction 5

Literature Review 6

1. Introduction 8

2. Exploration of Fostering Environment Methodologies 9

2.1. Challenge – Based Learning 9

2.2. Gamification in Education 9

2.3. Social Learning with Competition 10

3. Exploration of Available Platforms 11

3.1. LeetCode 11

3.2. CodeWars 14

3.3. CodeChef 15

3.4. SPOJ 15

3.5. Codeforces 16

3.6. Virtual Judge 16

3.7. Exercism 17

3.8. Coderbyte 18

3.9. Codility 18

4. Exploration of Tech Stack and Algorithms 19

4.1. C# Asp.Net Core 19

4.2. Keycloak 19

4.3. Redis 20

4.4. Docker & Docker Compose 21

4.4.1. Docker 21

4.4.2. Docker Compose 21

4.5. Container Orchestration Tools: Docker Swarm vs Kubernetes 22

4.6. AI Based Algorithm 23

4.6.1. Data Collection and Analysis: 23

4.6.2. Data Preprocessing: 23
4.6.3. Machine Learning Model Selection: 24
4.6.4. Model Training and Evaluation: 25
4.6.5. Adaptive Challenge Mechanism: 25
4.7. GraphQL 25
5. Challenges and Limitations 26
5.1. Technical Constraints 27
6. Future Directions and Recommendations 28
6.1. Mobile App 28
6.2. Problem-Solving Platform 28
7. Conclusion 28
Software Requirements Specification (SRS) 30
1. Introduction 31
1.1. Purpose 31
1.2. Scope 32
1.3. Definitions, Acronyms, and Abbreviations 33
1.4. Overview 34
2. Overall Description 36
2.1. Product Perspective 36
2.1.1. User Interfaces 36
2.1.2. Hardware Interfaces 37
2.1.3. Software Interfaces 38
2.1.4. Communication Interfaces 41
2.2. Product Functions 42
2.3. User Characteristics 44
2.4. Constraints 44
2.5. Assumptions and Dependencies 46
3. Specific requirements 47

3.1. External Interfaces 47
3.2. Functional Requirements 48
3.3. Non-Functional Requirements 54
3.3.1. Security 54
3.3.2. Usability 54
3.3.3. Maintainability 55
3.3.4. Compatibility 55
3.4. Performance Requirements 55
3.5. Logical Database Requirements 57
3.6. Design Constraints 57
3.7. Software System Attributes 58
4. Use Cases 60
Software Design Document (SDD) 91
1. Introduction 91
1.1. Purpose 92
1.2. Scope 92
1.3. Definitions, Acronyms, and Abbreviations 93
1.4. Overview 94
1.5. Version History 94
2. System Design 95
2.1. Architectural Design 95
2.2. Decomposition Description 95
2.3. System Modeling 99
2.4. Database Design 109
3. User Interface Design 110
Conclusion 113
References 115

Introduction

This document consists of Literature Review (LR), Software Requirements Specification (SRS) and Software Design Description (SDD) for the AI integrated GoatCapella platform. The literature review explores the foundational knowledge required to develop GoatCapella, including external platforms its communicating with (LeetCode, CodeForces etc.), the tech stack, various fostering environment methodologies and future problems with possible solutions. The software requirement specification document provides a comprehensive outline of the system's functional and non-functional requirements, details the system interfaces, and includes detailed use case diagrams to visually represent the interactions between users and the system. Following the SRS, the software design description document provides a comprehensive explanation of the platform's architecture using UML diagrams. It presents a basic decomposition of the design, accompanied by an Entity Relationship Diagram (ERD), as well as activity and sequence diagrams to illustrate key events. Additionally, it presents detailed user interface designs focusing on the critical aspects of the platform. By including all the documents, this project report delivers a comprehensive overview of the GoatCapella.

ÇANKAYA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

Literature Review

CENG 407

Innovative System Design and Development I

#6
GoatCapella

Çağdaş GÜLEÇ
Tarık ŞEN
İrem Beyza AYDOĞAN
Şeref Berkay KAPTAN
Beyza TOZMAN

Serdar ARSLAN
Bilgin AVENOĞLU

Abstract

This literature review explores the foundational knowledge required to develop GoatCapella, a platform designed to enhance coding skills through challenges, leveraging methodologies like social learning and gamification. GoatCapella will aggregate algorithmic questions and achievements from coding platforms like LeetCode and CodeForces using their public APIs to create a unified and highly customizable challenge management system. It will have the ability to retrieve questions and user progress, solution tracking and feedback. The review covers fostering environment methodologies, the technical capabilities of coding platforms, and the technologies and algorithms to support these features. Foreseeable challenges, limitations, and future directions for GoatCapella's development are also discussed.

1. Introduction

The need and desire to solve algorithmic questions and compete with other computer enthusiasts while doing it led to the emergence of endless coding platforms like LeetCode, CodeWars and CodeForces throughout the last decade. These platforms offer a place for anyone who wants to move forward on coding and algorithms, containing thousands of algorithmic questions, challenges, and competitions, supporting a large set of coding languages.

GoatCapella aims to take this one level further. GoatCapella's core mission is to support individuals to achieve their personal goals, leveraging methodologies like 'challenge-based learning', 'gamification' and 'social learning'. In doing so, it integrates existing platforms that meet the requirements through their public APIs into a highly configurable challenge management system. For a coding platform these requirements correspond to two matters. The first one is retrieving available questions, and the second one is retrieving the knowledge of whether a user solved that specific question or not. All the other data offered by the platform API can also be used to create and configure a challenge, but these two are the minimum. If the platform offers the solution given by an individual to a specific question, it enables even more capabilities like announcing user solutions to challenge creators and making them grade and comment on those solutions. Individual platforms and their abilities will be further investigated in the next chapters.

This literature review document covers the knowledge required to develop GoatCapella. It starts with ‘Exploration of Fostering Environment Methodologies’ which contains different methodologies needed to offer a challenging, social and productive environment. It continues with ‘Exploration of Available Platforms’ that investigates what each platform offers through their respective APIs. It then covers the ‘Technology Stack and Algorithms’ that may be used throughout the process. Next, foreseeable ‘Challenges and Limitations’ are examined. The document continues with ‘Future Directions’ and ends with a brief conclusion section.

2. Exploration of Fostering Environment Methodologies

2.1. Challenge – Based Learning

Challenge Based Learning (CBL) is a framework designed for learning through solving real-world challenges. It was created by Apple to identify the essential design principles of a learning environment. It helps participants become more engaged learners by enhancing their ability to recognize real-life challenges and developing the skills needed to create and pursue effective solutions [1].

CBL includes three phases: Engage, Investigate and Act. Each phase involves activities and exploratory cycles to prepare participants to move to the next phase. Adding challenges to learning environments fosters urgency, passion, and deep understanding. It enables learners the ability to transfer their knowledge across contexts, ensuring their skill sets remain relevant in an ever-changing world.

We hope to bring all the benefits of CBL to our project by making challenge creation a core part of our system, helping developers assess and improve their problem-solving and algorithmic skills to stay relevant in the tech world.

2.2. Gamification in Education

The gamification content shows that users' experience with feedback, focus, and challenge in the gamified content significantly correlates with their learning. Additionally, their focus on the game and the absence of distractions in gamification, along with the presence of clues embedded in the game to assist them in solving challenges and understanding new concepts at their own pace, all contribute to their enhanced learning experience. This issue shows the importance of paying attention to these factors in the design of educational games. These findings are consistent with the expert's opinion that instant feedback is the second key principle in gamification. Also, it has been stated that extended feedback increases the participant's experience. Points, badges, and scoreboards are all critical elements of feedback that encourage users to cooperate, engage, and interact more [2].

Which can be explained by 4 main subtopics:

- **Assessment:** For all activities, instant feedback of correctness or incorrectness, explanation of wrong options, earning or points, points of each solution, total points, scoreboard of the successful passage of the challenge were provided.
- **Challenge:** Challenge/Quest refers to a specific task that a player-controlled character or group undertakes to earn a reward. In quest-based games, players engage in interrelated activities that usually involve movement across different action points.
- **Leaderboards:** Leaderboards visually rank players based on their achievement considering the educational context, a typical leaderboard usually displays information related to their ranking, which is defined by their learning progress.
- **Rewarding:** Reward systems are based on motivation: they act as incentives for players or to alleviate potential disappointment.

2.3. Social Learning with Competition

In the complex landscape of human interaction, the dynamics of competition often shape behaviors and decision-making processes. Social learning allows individuals to observe and learn from the actions of others. Social learning is a process by which individuals acquire knowledge, skills, and behaviors by observing others. This learning mechanism is particularly significant in competitive situations [3].

Mechanisms of Social Learning

- **Modeling:** People often look to role models who excel in their competitive spheres. By emulating the behaviors and strategies of these successful individuals, learners can increase their chances of achieving similar outcomes. This modeling is a critical aspect of adapting to competitive pressures.
- **Feedback and Adaptation:** Social learning is not a one-time event; it involves continuous feedback. Individuals learn from both their own actions and the consequences faced by others. Positive outcomes reinforce specific strategies, while negative experiences may prompt individuals to reassess their tactics and adopt new approaches.
- **Competitiveness of Others:** The degree of competitiveness exhibited by peers can significantly shape one's own behaviors. For instance, if an individual observes a highly aggressive competitor, they may feel compelled to adopt a similar approach

to avoid being outperformed. This dynamic creates an environment where competitiveness is both modeled and reciprocated [4].

3. Exploration of Available Platforms

The primary purpose of all coding platforms discussed in this section is to provide computer enthusiasts tools to enhance their algorithmic skills, prepare for technical job interviews, and ultimately land their dream jobs. Doing so, most of the platforms offer an interactive web-based coding editor and a vast collection of coding questions / challenges designed to sharpen problem-solving skills.

When integrating a platform into GoatCapella, three core capabilities of the platform API are examined. Full integration is possible if all three capabilities are supported, while the platform can still be partially integrated if it supports the first two. These key capabilities are:

- The ability to retrieve the platform's available questions.
- The ability to retrieve information on whether a user has solved a particular question or not.
- The ability to retrieve the user's solution for a specific question.

The following sections provide an in-depth analysis of individual researched coding platforms. Each section begins with a description of the platform, followed by its key capabilities. Next, three core features previously mentioned are examined along with their corresponding API requests where applicable. The section continues with any other platform specific capabilities that may be used for challenge management and concludes by evaluating how suitable the platform is for integration into GoatCapella.

3.1. LeetCode

LeetCode is one of the most popular online coding platforms. It offers more than 3500 coding questions to its hundreds of thousands of active users. It is not only used by candidates but also by companies to identify technically talented candidates. It supports 14 different coding languages at its core and organizes weekly and bi-weekly online challenges regularly [5].

LeetCode distinguishes itself from rival platforms through several unique features, including:

- Allowing users to publish their accepted answers, which others can rate and comment on.

- Providing comparisons between accepted solutions to evaluate memory efficiency and runtime performance.
- Offering official solutions for some questions.
- Tagging questions with over 50 topic-based labels; premium users can also access company-specific tags.
- Offering Study Plans and Crash Courses.
- Rewarding consistent usage with App Coins.
- Providing a GraphQL API that mirrors most web-based operations.

LeetCode's GraphQL API fully supports the three essential capabilities needed by GoatCapella:

1. **Retrieving Questions:** The API provides a straightforward method to fetch questions. The request in **Error! Reference source not found.** retrieves the f irst 10 questions without filtering (for brevity, some attributes are omitted).

```
query problemsetQuestionList {
  questionList(categorySlug: "", limit: 10, filters: {}) {
    data {
      content
      difficulty
      title
}}}
```

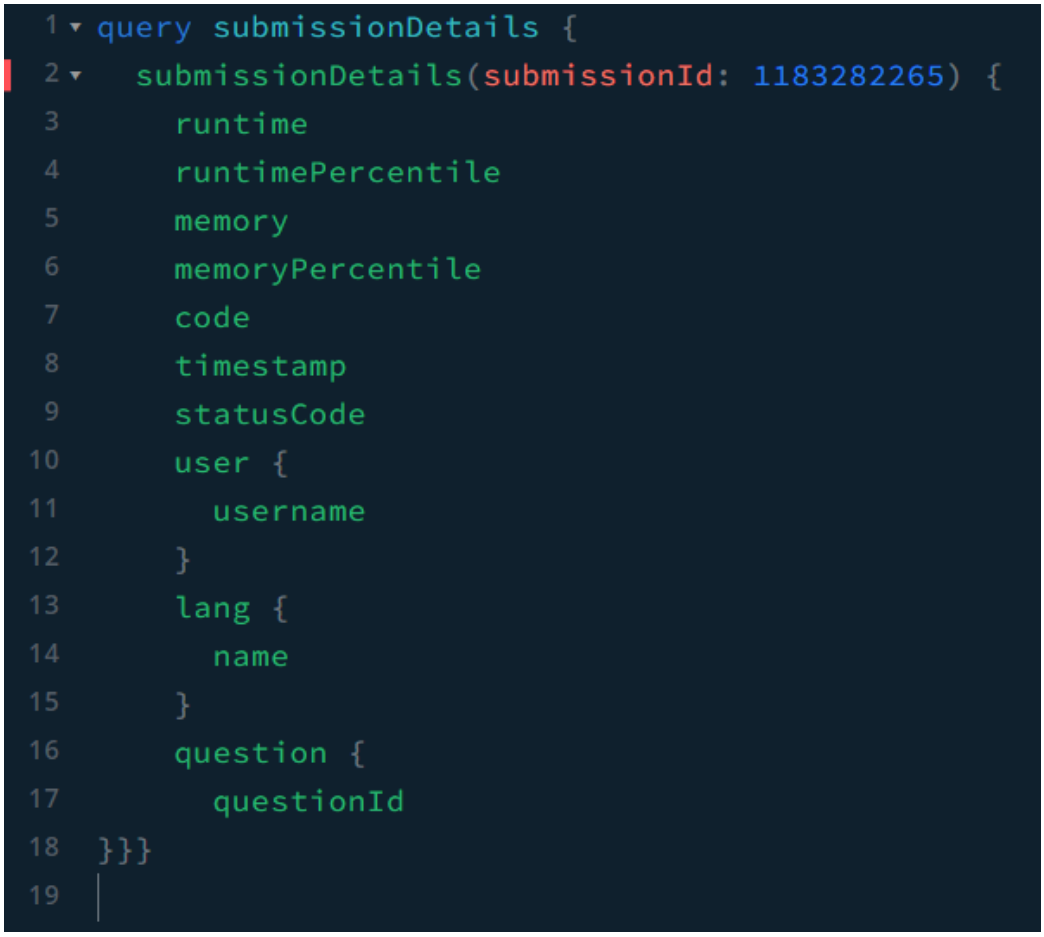
Figure 1. Query of Retrieving Questions

2. **Retrieving Solution Status**: While the API only returns the user's 20 most recent solutions, this can be handled by caching, allowing verification of earlier questions as needed. **Error! Reference source not found.** demonstrates r etrieving the last 10 solutions for the user 'sentarik'.

```
query recentAcSubmissions {
  recentAcSubmissionList(username: "sentarik", limit: 10) {
    id
    title
}}
```

Figure 2. Query of Retrieving Solution Status

3. **Retrieving User Solutions**: User solution data is accessible through authenticated requests. Although the authentication does not need to belong to the user, it must come from an active account. GoatCapella can therefore use a single account to retrieve solution data. **Error! Reference source not found.** s hows how to fetch a solution using an ID obtained from the previous request (with certain attributes and authentication cookies omitted for brevity).



```
query submissionDetails {
  submissionDetails(submissionId: 1183282265) {
    runtime
    runtimePercentile
    memory
    memoryPercentile
    code
    timestamp
    statusCode
    user {
      username
    }
    lang {
      name
    }
    question {
      questionId
}}}
```

Figure 3. Query of Retrieving User Solutions

LeetCode not only supports the three core capabilities but also many other achievements like user ranking, badges, skills, language statistics and the number of questions solved per difficulty level. Each can be accessed through various GraphQL requests, enhancing the diversity and quality of challenges.

In conclusion, LeetCode is highly suitable for integration into GoatCapella due to its extensive GraphQL API, which grants access to almost all data available on its web interface. Additionally, its broad range of achievements, metrics, and user insights can significantly enhance the challenge creation and management processes within GoatCapella.

3.2. CodeWars

CodeWars is a prominent coding platform based on challenges and community-driven problem-solving. The platform offers more than 7500 katas categorized by difficulty, topic and language, with stable support for over 25 different programming languages [6][7]. CodeWars harbors a unique interpretation at its core, based on traditional Japanese martial arts, originating from Dave Thomas, co-author of the book The Pragmatic Programmer. In this system, each kata represents a coding question, and users are ranked based on kyus and dans. Such an approach especially appeals to beginners, promoting skill development incrementally [8].

CodeWars API has limited functionality, with only four endpoints [9]:

- **User Information:** (api/v1/users/{user}): Provides user details such as id, username, honor, clan, ranking, skills and total number of questions solved.
- **Completed Challenges :**(api/v1/users/{user}/code-challenges/completed): Lists every challenge completed by a specific user, including the challenge id, name, slug-name, completion date and completed languages.
- **Authored Challenges:** (api/v1/users/{user}/code-challenges/authored): Lists every challenge authored by a specific user, including challenge id, name, description, rank, tags and languages.
- **Challenge Information:** (api/v1/code-challenges/{challenge}): Retrieves a specific challenge's details, including id, name, category, description, tags, languages, rank, creator, creation and approval dates, and number of times it is completed.

While CodeWars API cannot be integrated directly into GoatCapella, partial integration is achievable using the above API endpoints with some tricks.

- **Retrieving Questions:** Although CodeWars API lacks direct support for retrieving platform questions, it allows access to kata details if the ID is known. Using the second and third API endpoints, GoatCapella can obtain these IDs and retrieve question information indirectly.
- **Retrieving Solution Status**: CodeWars lists every question solved by the user with the second API request, enabling indirect extraction of solution status.
- **Retrieving User Solutions**: Currently, CodeWars API does not support such a functionality, making the system only partially integrable.

Although full integration is not possible, the first endpoint provides many user achievements, enabling highly customized challenge creation and management. Since the platform also supports community created katas, integrating it into GoatCapella is also quite important.

3.3. CodeChef

CodeChef is a web-based online platform that provides opportunities to learn coding and participate in competitions. Offering various interactive courses in languages such as Python, C, C++, Java, and JavaScript, the platform also includes numerous questions on programming languages, data structures, algorithms, SQL, and web development. The difficulty levels of the questions range from basic problems to complex ones [10].

Additionally, there are skill tests available to prove abilities in specific technologies. Every Wednesday at 8:00 PM (IST), 2-hour programming competitions are held, which also increase users' scores on the platform. CodeChef Pro offers premium features such as video solutions, AI-powered assistance, training courses, and certification.

Since CodeChef has discontinued the use of its API, we are unable to access it, and therefore cannot integrate it into our platform.

3.4. SPOJ

SPOJ is an online assessment platform that provides opportunities for practicing algorithm development and problem-solving skills. With over 20,000 available problems, tasks on this platform are either prepared by the SPOJ community of problem setters or sourced from past programming competitions. In addition to popular languages like Python, C++, and Java, it supports more than 40 programming languages, offering users a flexible experience.

The platform evaluates user submissions automatically through the Sphere Engine, enabling users to quickly verify the accuracy and efficiency of their code. With content available in multiple languages besides English, SPOJ appeals to a broad user base. Users can organize contests according to their own rules and invite others to participate. Additionally, SPOJ offers a forum for discussing specific problems, where users can assist one another and share alternative solution strategies [11].

The technology behind SPOJ, provided by Sphere Engine, offers API support; however, this service is paid. The Sphere Engine API is a paid solution for developers seeking to integrate code execution and evaluation features into their own applications. Due to budget constraints, our use of SPOJ on our platform is limited [12].

3.5. Codeforces

Codeforces, established in 2009 by Mikhail Mirzayanov, is known as one of the most rooted coding platforms. Despite its somewhat dated interface, Codeforces remains highly active and regularly organizes challenges that still attract participants from around the world. The platform offers more than 10,000 questions and supports over 20 different coding languages, with various versions and configurations, providing users with a flexible choice of coding environment [13][14].

Codeforces API offers 16 different endpoints. Most of these can be accessed anonymously, allowing only the public data of users to be available via the API. Below, the three previously mentioned requirements for integrating a coding platform into GoatCapella are examined individually [15].

- **Retrieving Questions:** Codeforces API directly supports retrieval of available questions on the platform from the endpoint: "/api/problemset.problems".
- **Retrieving Solution Status:** While Codeforces does not have a direct endpoint to query whether a user has solved a specific question, it does provide a list of questions solved by a user, which allows GoatCapella to access the information it needs. The relevant API endpoint is: "api/user.status?handle={username}".
- **Retrieving User Solutions:** Unfortunately, Codeforces API does not support querying the solutions of users.

Overall, Codeforces meets the first two requirements that GoatCapella seeks, making it only partially integratable into GoatCapella. Despite this limitation, Codeforces remains an invaluable resource for GoatCapella.

3.6. Virtual Judge

Virtual Judge is a web-based platform for competitive programming, encompassing logic and programming problems from a range of well-known online judges like Codeforces,

LeetCode, POJ, AtCoder, and many others. Acting as a central hub, it enables users to solve problems across various platforms, track their progress, and compare results with a global user base, all within a single interface. The platform is particularly popular among students and competitive programming enthusiasts who seek to prepare for contests or enhance their problem-solving skills across various types and difficulty levels without switching between multiple online judges [16].

A unique aspect of Virtual Judge is its support for multi-judge integration. This feature allows users to access problems from different sources and submit their solutions directly on Virtual Judge. The platform then forwards submissions to the original online judge, handles the results, and provides feedback. This centralized submission system lets users practice seamlessly without needing multiple accounts or navigating different interfaces. Additionally, Virtual Judge offers custom contests and team features, enabling users to create personalized contests or practice sessions by selecting problems from among the integrated judges. This flexibility makes it a preferred choice for organizing programming practice, especially in academic or group training environments.

Since Virtual Judge lacks official documentation and a supported API, we are relying on an unofficial API shared by a GitHub user to access user solution history. While this API facilitates integration with our system, some endpoints are outdated or undefined, which may limit access to certain features.

3.7. Exercism

Exercism is a free, open-source platform designed to help people learn and improve their coding skills. It offers exercises and learning tracks for over 65 programming languages, with a focus on both beginners and more advanced learners. It provides a collaborative environment for developers by enabling users to review each other's code and give helpful tips and suggestions. Users can also track their progress and earn badges. Exercism stands out by offering a mentorship program, where users can volunteer as mentors. Mentors provide short descriptions of their expertise and proficient languages, allowing students to find and connect with them for guidance. This feature aims to support the growth of both mentors and students [17].

Due to its API being private, direct integration with our platform isn’t feasible.

3.8. Coderbyte

Coderbyte is a coding platform developed with the purpose of helping individuals grow their coding skills. It offers algorithm and data structure problems with difficulty levels ranging from easy to hard and supports 17 programming languages. It also includes challenges in other areas such as front-end, back-end, and database, helping users improve their skills across various aspects of software engineering.

Coderbyte is the only platform where the runtime analysis for users' solutions is generated and expressed in Big-O notation, making it a preferable choice for users interested in code efficiency. It also offers a feature called 'Interview Kits' to help developers prepare for technical interviews with expert guidance. However, access to the 'Interview Kits' and much of the challenge content requires a paid subscription [18].

The platform's API has limited access, with an API key available only through a paid plan. Due to this limitation, integration of Coderbyte into this project will not be pursued.

3.9. Codility

Codility is a web-based platform used in technical recruitment to assess programming skills through structured coding challenges and tests. Designed to help companies effectively screen developers for problem-solving abilities and coding efficiency, Codility streamlines the hiring process for technical roles. The platform offers extensive features, including Live Collaborative Coding through its CodeLive feature, which enables live coding sessions during technical interviews. In these sessions, candidates and interviewers can work together on coding problems in real time, simplifying the assessment of problem-solving approaches and communication skills directly within the interview setting [19].

Codility also provides an API with advanced capabilities, such as retrieving coding questions, user data, and solution histories, making it well-suited for technical projects needing data integration. However, this API is part of Codility's paid offerings, which limits its direct use in our project due to budget constraints [20].

4. Exploration of Tech Stack and Algorithms

4.1. C# Asp.Net Core

ASP.NET Core is an open-source, cross-platform web development framework by Microsoft. Key features include:

- **Performance and Modularity**: Enables high-performance, lightweight applications by including only necessary components.
- **Cross-Platform Compatibility**: Runs on Windows, macOS, and Linux.
- **Support for MVC and Razor Pages**: Provides MVC for larger projects and Razor Pages for simpler, page-based applications.
- **RESTful API Development**: Ideal structure for building fast and reliable APIs.
- **Cloud and Container Support**: Compatible with cloud deployment and container technologies like Docker.
- **Advanced Debugging**: Includes powerful tools to quickly troubleshoot issues.

These features make ASP.NET Core a strong choice for building modern, scalable, and maintainable applications [21].

4.2. Keycloak

Keycloak is an open-source Identity and Access Management solution that enables applications to provide authentication and authorization without the application having to handle these functions directly. It incorporates advanced authentication standards such as OAuth2, OpenID Connect, and SAML 2.0, making it a flexible tool for securing applications, particularly in cloud environments and distributed systems. Keycloak supports Single Sign-On (SSO), allowing users to log in once and access multiple applications within the same realm. This feature is especially beneficial for large systems with multiple services that require seamless access for users. It is also particularly useful for organizations looking to offer social login through providers like Google or Facebook, while integrating with corporate directories such as LDAP and Active Directory [22].

A key feature of Keycloak is its centralized management of users and security configurations across various applications. This enables developers to configure roles, permissions, and security policies in a unified environment rather than embedding them into

each application individually. Additionally, Keycloak supports identity brokering, allowing authentication to be delegated to external identity providers, making it ideal for organizations managing both internal and external users across diverse platforms. Keycloak also excels in fine-grained authorization, offering detailed permission structures for resource-based access control [23]. These capabilities position Keycloak as a robust tool for managing secure access in modern application architectures like microservices, web applications, and APIs [24].

4.3. Redis

Redis (Remote Dictionary Service) is an open source, in-memory, NoSQL database known for its exceptional speed and versatility. As a key-value store that operates in memory, Redis enables rapid data retrieval, making it ideal for use cases that require low-latency responses, like caching, session management, and real-time analytics. Beyond simple key-value storage, it supports a wide range of data structures such as strings, hashes, lists and sets. This allows for highly atomic operations on complex data structures, differentiating it from other key-value databases.

One of Redis's unique traits is that it combines the benefits of an in-memory data store with persistence options. It can save snapshots of its data to disk through RDB (Redis Database Backup) files or maintain an append only log (AOF) for data durability. This hybrid approach gives Redis a high read and write speed while limiting the dataset to memory size, which keeps the internal complexity lower, and the memory footprint smaller compared to on-disk databases. For larger datasets, Redis labs offers a "Redis on Flash" solution, which allows data to reside on both RAM and flash memory.

Redis has expanded to support a variety of use cases, including document-based storage and vector databases. It can efficiently store JSON and other document-like formats, making it valuable for applications with unstructured or semi-structured data. Moreover, its vector database capabilities allow it to handle high-dimensional vector data, which is crucial for AI and machine learning applications, especially those requiring similarity search, embeddings, or nearest-neighbor queries. This makes Redis well-suited for retrieval augmented generation (RAG), where it can store and rapidly retrieve embeddings and contextual data to enhance AI-driven responses. Redis also supports advanced functionality like pub/sub messaging, Lua scripting for complex operations, and various memory optimization options. Developers can

set memory limits and eviction policies to prevent Redis from consuming excess resources, and it can run on multi-core systems by sharding data across multiple instances [25].

In our case, we'll use it to reduce the load on our main database by caching frequently requested data. This will minimize unnecessary API calls and prevent constant access to the database when the data is available in the cache, improving performance and reducing response times.

4.4. Docker & Docker Compose

4.4.1. Docker

Docker is an open-source platform for developing, shipping, and deploying applications, allowing them to run in isolated environments called containers. This isolation supports efficient and rapid testing and development. Docker uses a client-server architecture where the Docker client communicates with the Docker daemon. The daemon manages the heavy lifting of building, running, and distributing containers through a REST API over UNIX sockets or a network interface. This model allows containers to run concurrently on a host without the overhead of a hypervisor, making them lightweight and secure [26].

4.4.2. Docker Compose

Docker Compose streamlines the management of multi-container applications through simple YAML configuration file, defining services, networks, and volumes. This declarative approach allows developers to control complex setups with ease [27]. Docker Compose enables a robust yet simplified method to manage complex containerized applications, promoting efficiency and scalability across all stages of development and deployment. Some of the key features:

- **Service Definitions**: Each service represents a containerized application, specifying its image, configurations, ports, and dependencies.
- **Network Management**: Compose automatically sets up isolated networks for seamless inter-service communication using container names.
- **Persistent Storage**: Volumes help retain data across container restarts.

- **Single Command Orchestration**: Commands like docker-compose-up and docker-compose-down simplify starting, stopping, and cleaning up multi-container applications.
- **Versatile Environments**: Compose is suitable for development, testing, staging, and production, maintaining consistency across workflows. It integrates seamlessly with CI/CD pipelines, enabling automated processes in production steps.

4.5. Container Orchestration Tools: Docker Swarm vs Kubernetes

Container orchestration tools like Docker Swarm and Kubernetes automate container management, networking, deployment and scaling of containers, allowing developers to get rid of challenges of managing multiple containers manually. Container orchestrators address this challenge by automating various tasks like; load balancing, container failover, resource allocation and scaling, ensuring high availability [28].

Two popular container orchestrators are Docker Swarm and Kubernetes. Docker Swarm is an open-source container orchestrator with native support of Docker. Main features of Docker Swarm include:

- Integration with Docker Engine, making it compatible with Docker tools like Docker Compose.
- Easy to install, learn and use.
- Swarm API support.
- Built-in load balancing.
- Limited functionality compared to Kubernetes.

Kubernetes has become the industry standard over the past decade. It was originally developed by Google and thereafter maintained by the Cloud Native Computing Foundation; it is now open source. Key features of Kubernetes include:

- Wide range of functionalities like auto-scaling, service discovery, rollouts, rollbacks, ingress, load balancing and job execution.
- Intensive set of API support.
- Quite a steep learning curve.
- Heavier system requirements for basic setups.

When these two orchestration tools are compared, Docker Swarm looks like a preferable choice for small to medium projects, not requiring intense workflows. On the other hand, as the project and its workflow get complex, transitioning from Docker Swarm to Kubernetes may be a good idea [29].

4.6. AI Based Algorithm

One of the objectives for our project is to develop an AI model that generates personalized question sets and challenges based on a user's previous question-solving performance. This model will serve as an adaptive learning system that analyzes the user's strengths, weaknesses, and problem-solving habits to dynamically tailor question sets, thereby promoting effective learning. The system will rely on user data like time taken to solve questions, accuracy rates, difficulty levels, types of questions tackled, and the complexity of challenges handled. Such a system requires a multi-phase development process, involving data collection and preprocessing, model selection, training, and iterative performance evaluation.

4.6.1. Data Collection and Analysis:

In the initial phase, we plan to collect and analyze various aspects of users' question-solving performance, including solving time, accuracy rate, question difficulty, types of questions, complexity levels, and user interaction metrics like total questions solved and regularity in practice sessions. Together, these data points will help create a comprehensive user profile, forming a foundation for personalized and adaptive question generation. Leveraging these features, our model may be able to understand and predict user performance trends over time, supporting its adaptive function.

4.6.2. Data Preprocessing:

Once data is collected, it would undergo preprocessing to ensure compatibility with the AI model. Key steps may include:

- **Handling Missing Values:** Addressing any incomplete data entries, such as unanswered or skipped questions, to provide the model with a fuller understanding of each user's performance.
- **Normalization:** Scaling continuous variables, such as solving time, to a 0–1 range to enable consistent processing across different question types and difficulty levels.

- **Labeling Difficulty Levels:** Assigning categorical labels to question difficulty (easy, medium, hard) to provide clear input for supervised learning models.
- **Feature Engineering:** Extracting additional features, such as average solving time per difficulty level, correct-to-incorrect ratio across categories, and interaction regularity, to potentially enhance model accuracy.

4.6.3. Machine Learning Model Selection:

1. Supervised Learning

The initial component of the model is planned to be Supervised Learning, specifically aimed at predicting a user's likelihood of solving questions accurately across various difficulty levels. This approach involves creating a classification model to predict question outcomes, forming the basis for determining appropriate question sets.

We are considering algorithms like Logistic Regression for simple prediction tasks, primarily to predict whether a user might correctly answer a question based on past performance. For more complex datasets with varying difficulty levels, Random Forest and Gradient Boosting algorithms may also be suitable. These models are effective at handling nonlinear relationships and complex interactions in data, making them well-suited for detecting subtle trends in user performance and predicting optimal difficulty levels accordingly [30].

2. Reinforcement Learning (RL)

To dynamically adjust the difficulty of questions and generate adaptive challenges, Reinforcement Learning (RL) techniques will be integral. By implementing Deep Q-Networks (DQN), the system can learn which question difficulties provide the right level of challenge based on real-time feedback, thus optimizing user engagement and progress [31]. For instance, the model can respond to a user's performance by gradually increasing question difficulty when correct answers are consistently provided or lowering it if incorrect answers dominate.

Another unique aspect of RL in this project is the use of Bayesian Optimization to further fine-tune question selection based on the user’s response history. This method leverages the likelihood of correct versus incorrect answers to select questions that offer the optimal challenge level for each user, thus creating a responsive, dynamic challenge structure [32].

3. Recommendation Systems

Lastly, Recommendation Systems will support the generation of diverse question sets. Using Collaborative Filtering, the model can suggest new questions based on user profiles similar to the target user, utilizing historical data on what worked for other users to create targeted question recommendations. Additionally, Content-based Filtering will allow the model to recommend questions based on features of previously solved questions, such as topic, complexity, and difficulty level [33]. This dual recommendation approach ensures that users receive question sets that are both challenging and relevant.

4.6.4. Model Training and Evaluation:

After data preprocessing and model selection, the next phase could involve model training. The training and evaluation might be structured around performance metrics, such as accuracy, precision, recall, and F1-score, to assess effectiveness based on the chosen model. For the reinforcement learning component, a cumulative reward metric could be used to track how well the system adapts over time.

4.6.5. Adaptive Challenge Mechanism:

The AI model will generate challenges adaptively, adjusting to user progress:

- **Difficulty Adjustment**: Based on performance metrics, question difficulty will be automatically increased or decreased.
- **Reward Mechanisms**: Users will be rewarded with badges or points for completing challenges, motivating continued engagement and improvement.

This project integrates AI methodologies to enhance personalized challenge experiences, utilizing Supervised Learning, Reinforcement Learning, and Recommendation Systems to optimize question selection and challenge generation. Each AI technique brings unique strengths, allowing the system to adapt to users' skill levels and challenge preferences in real-time. By leveraging these methods, the project aims to deliver a tailored challenge environment that evolves in sync with users' progress, making coding practice more engaging and effective.

4.7. GraphQL

GraphQL is a query language used for APIs. It operates independently of any specific database or storage engine, enabling data querying by leveraging existing code and data, and it can be developed without dependence on particular programming languages. This flexibility

allows GraphQL to adapt seamlessly to diverse infrastructures, offering a highly dynamic and adaptable usage [34].

One of the most significant features of GraphQL is its ability to retrieve data from a server via a single endpoint. In contrast to traditional REST APIs, which may require multiple endpoints for various data types, GraphQL provides access to all data through a single endpoint. Additionally, it empowers users to retrieve only the data they need, reducing data redundancy. This way, users can avoid unnecessary data downloads and processing, ensuring access to only the specific information required. For example, while a REST API call might retrieve an excess of data, GraphQL enables access solely to the specified fields [35].

In summary, GraphQL introduces flexibility and efficiency to data exchange. It goes beyond the limitations of REST APIs, allowing users to determine precisely what and how much data is retrieved. These capabilities enhance performance on both client and server sides, making the development process faster, more efficient, and user friendly.

5. Challenges and Limitations

The development of GoatCapella presents a range of challenges and limitations that must be addressed to ensure a stable, secure, and scalable platform. These challenges stem from dependencies on external platforms, scalability demands, and the need for effective user engagement strategies, each posing specific technical and operational constraints. Key considerations include:

- **API Accessibility and Limitations**: Many coding platforms, like Exercism and Codility, have limited API access, which may not cover all the features GoatCapella needs. Some platforms may not provide any API, requiring the team to implement web scraping, which can be unreliable and legally restrictive.

- **Dependency on External Platforms**: GoatCapella relies on third-party coding platforms. If these services experience downtime or limit API access, it will disrupt the system's functionality, affecting challenge creation and participation. Changes to third-party APIs or terms of service may break functionality, necessitating regular updates and maintenance.

- **Scalability and Performance**: The platform might struggle with performance as the user base grows. Without scalable infrastructure, high user traffic could cause slowdowns or crashes. Managing concurrent users, especially in time-sensitive challenges, requires optimized backend processing and load balancing.

- **Complexity of Challenge Management**: The system's flexibility in creating public, semi-public, and private challenges with configurable rules introduces complexity, especially in ensuring consistent, fair evaluations. Multiple admin roles and hierarchies add complexity in access control, requiring meticulous management to prevent unauthorized changes.

- **Security and Privacy Concerns**: Handling user authentication, authorization, and sensitive data storage introduces security risks, including data breaches. Strong data encryption and security practices will be required to protect user information and challenge data.

- **User Engagement and Continuity**: Ensuring active user participation and daily streaks may require constant incentive mechanisms and prompt content, which can be challenging to maintain without an established user base. Managing a digital currency for rewards requires a secure, fair, and user-friendly transaction and balance tracking system.

5.1. Technical Constraints

- **Data Caching and Latency Management:** To handle high user volumes and frequent interactions, GoatCapella needs efficient caching strategies for commonly accessed data. Implementing Redis or a similar in-memory caching service can alleviate some server load, but maintaining cache consistency is challenging, particularly for real-time challenge results.
- **Cross-Platform Compatibility and Responsiveness:** GoatCapella would, over time, want to reach its users on more varied devices, meaning it will have to support more varieties of devices. This means the use of responsive design practices to provide a seamless experience irrespective of device types. Optimizing front-end components for consistency in user experience will take care of handling features particular to devices and testing on different screen sizes and operating systems.

6. Future Directions and Recommendations

As GoatCapella evolves, two primary future goals have been set to enhance user experience and platform independence: developing a dedicated mobile app and creating our own challenge-solving platform, like LeetCode, where we can manage challenges directly.

6.1. Mobile App

With a mobile app, instead of relying solely on web or email for tracking, users would receive daily notifications and reminders, be able to create new challenges, get notified about created or recommended challenges, track weekly and daily progress, and monitor current scores and competitors' progress. The mobile experience is expected not only to increase user engagement but also to keep them consistently connected to their goals, regardless of device or platform.

6.2. Problem-Solving Platform

A dedicated website is envisioned to host GoatCapella-specific questions and a problem-solving environment. This step aims to reduce dependency on third-party platforms and their APIs. With our own platform, we can create content that aligns more closely with our mission of personal and challenge-based learning, offering unique content tailored to users' needs and preferences. The ability to customize content quality and consistency on this platform is expected to enhance the overall user experience and enable GoatCapella to operate as an independent learning and challenge platform.

7. Conclusion

In this report, we conducted an in-depth exploration of key elements essential to our project’s development.

The report begins with an introduction to fostering environment methodologies, specifically focusing on three approaches: Challenge-Based Learning, Gamification in Education, and Social Learning with Competition. These methodologies were chosen for their potential to enhance user engagement and learning outcomes within our platform. Each method is analyzed for its relevance to our project goals, outlining how it can contribute to creating a dynamic and collaborative learning space. Following this is a section exploring several existing

platforms like our project concept, assessing ten platforms in total. For each platform, we detail its unique features, strengths, and limitations, and evaluate its potential for integration with our project. The report then moves on to a technical assessment, examining the tech stacks and algorithms that will form the backbone of our platform. Key technologies considered include ASP.NET Core for backend development, Keycloak for authentication, Redis for caching, Docker and Docker Compose for containerization, and container orchestration tools. A preliminary plan for integrating AI algorithms is also presented in this section, detailing how they could enhance the user experience through features like adaptive challenge generation. Additionally, potential challenges and limitations that could impact the project are addressed, and the report concludes with a discussion of future directions, outlining technology enhancements as our platform grows.

In summary, this report provides a comprehensive foundation for our project by exploring competitor platforms, relevant methodologies, and technical considerations while anticipating potential obstacles and future opportunities. Through careful implementation and ongoing refinement, we are confident that our platform will effectively address the needs of our target audience and make a meaningful impact in a competitive market.

ÇANKAYA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

Software Requirements Specification (SRS)

CENG 407

Innovative System Design and Development I

#6
GoatCapella

Çağdaş GÜLEÇ
Tarık ŞEN
İrem Beyza AYDOĞAN
Şeref Berkay KAPTAN
Beyza TOZMAN

Serdar ARSLAN
Bilgin AVENOĞLU

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 [36].

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 [36].
- Windows Server Documentation [37].

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 [38].

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 [39].

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 [40].

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 [41].

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 [42].

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 [43].

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 [44].

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:

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.

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.

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 [45].
- **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 [46].

- **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 [47].

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 [48].

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 [49].

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

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

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

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

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

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

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 [50].
- **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 [41].

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 [43]. 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:

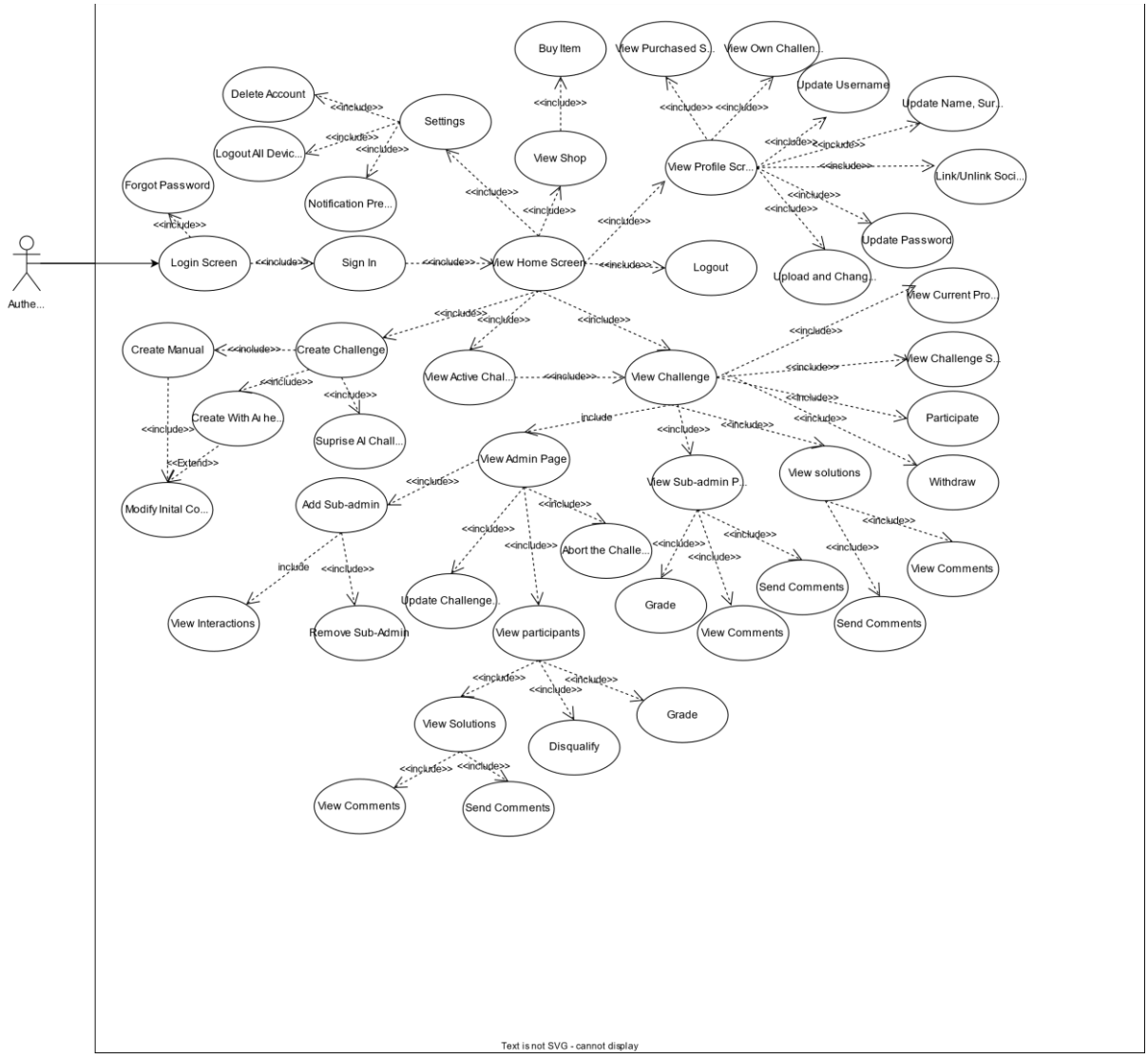
- Compatibility with both Linux and Windows servers.
- Independence from platform-specific features in the application code.

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



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	User opens the application User chooses from Login or Forgot Password options. System successfully directs the user to the related page.
Alternate Flows	

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	User opens the applications. Users input their login credentials. User clicks on the login button. If the credentials are correct, system successfully directs the user to the related page.
Alternate Flows	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	

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.

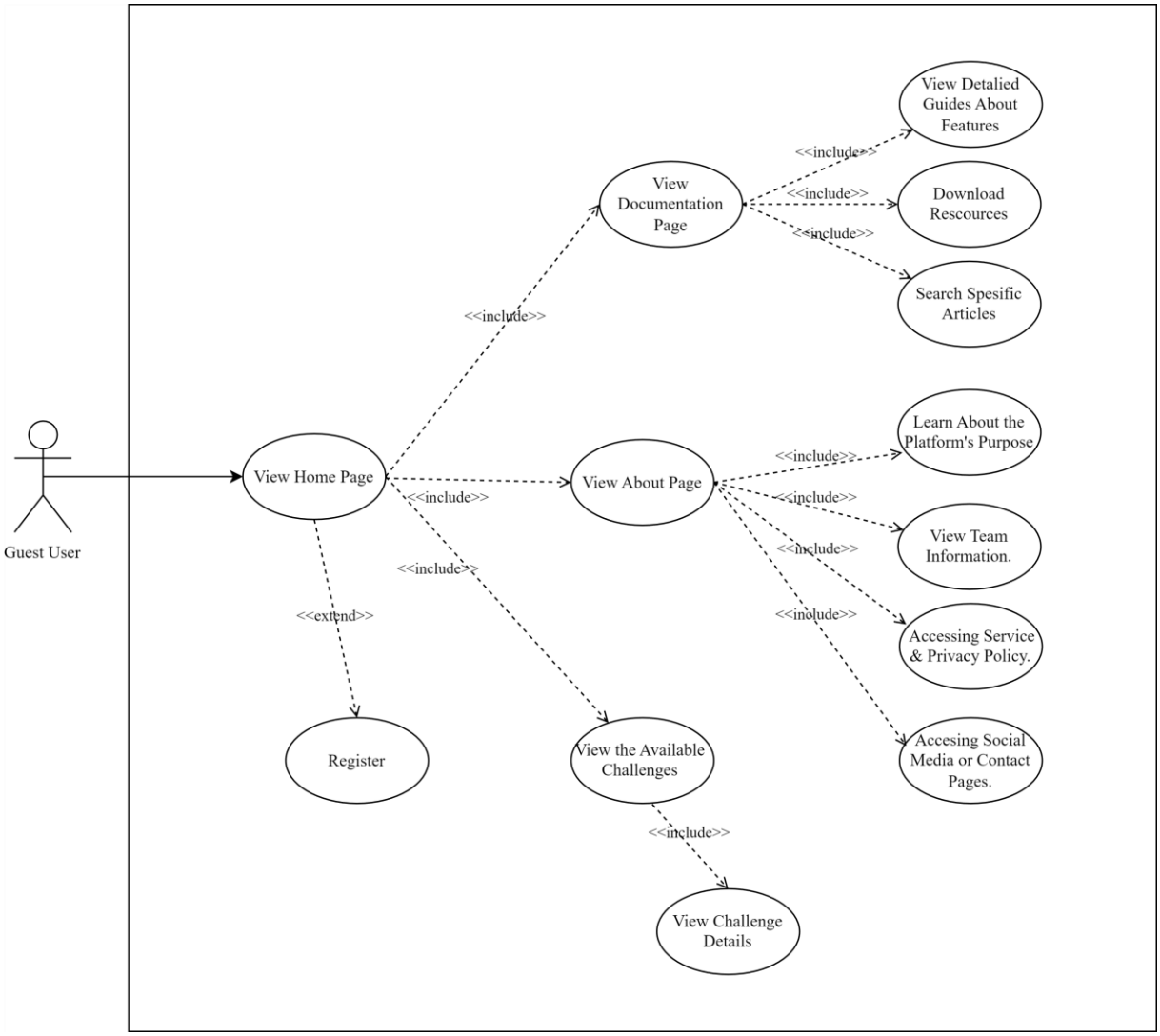
Use Case Id	11
Use Case Name	View shop
Actor	Authenticated User
Description	This use case allows the user to browse and purchase items in the shop using app coins as the only currency.
Precondition	The user must be logged into the system and have a sufficient balance of app coins.
Related Use Cases	2

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



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	User opens the platform. 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	User clicks on the "About" link. System displays the about page
Alternate Flows	If "About" link is not available, display an error message.

Use Case Id	15
Use Case Name	View Available Challenges
Actor	Guest User

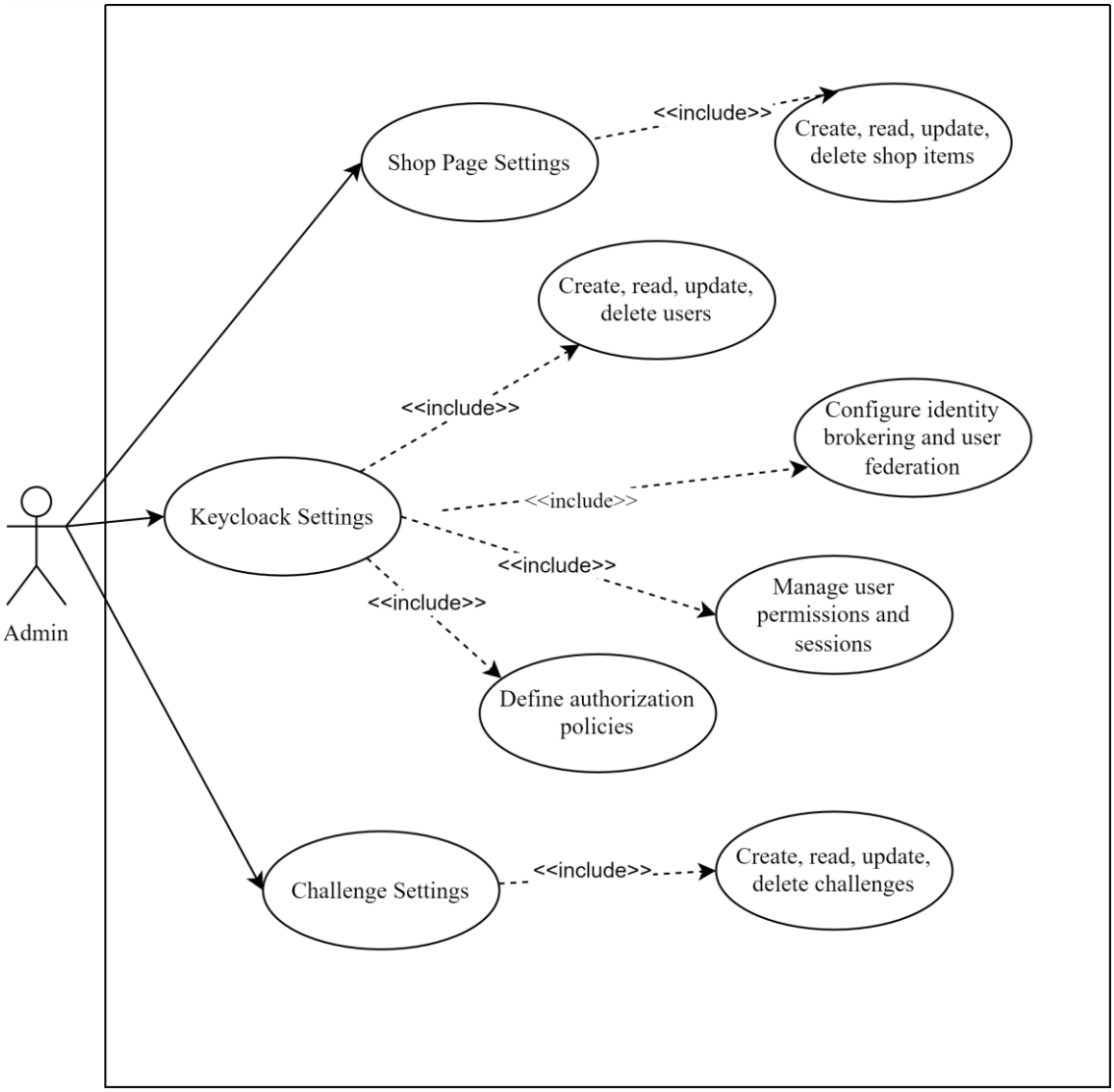
Description	Allows the user to view a list of available challenges on the platform.
Precondition	User is accessing the platform
Related Use Cases	13
Postcondition	User sees the list of available challenges.
Main Flow	User navigates to the "Challenges" section. System displays the list of available challenges.
Alternate Flows	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	User clicks on the "Documentation" link. System displays the documentation page.

Alternate Flows	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	The guest user clicks on the "Register" button on the platform. The system displays the registration form. The user fills in the required details The system validates the provided details.
Alternate Flows	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



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.

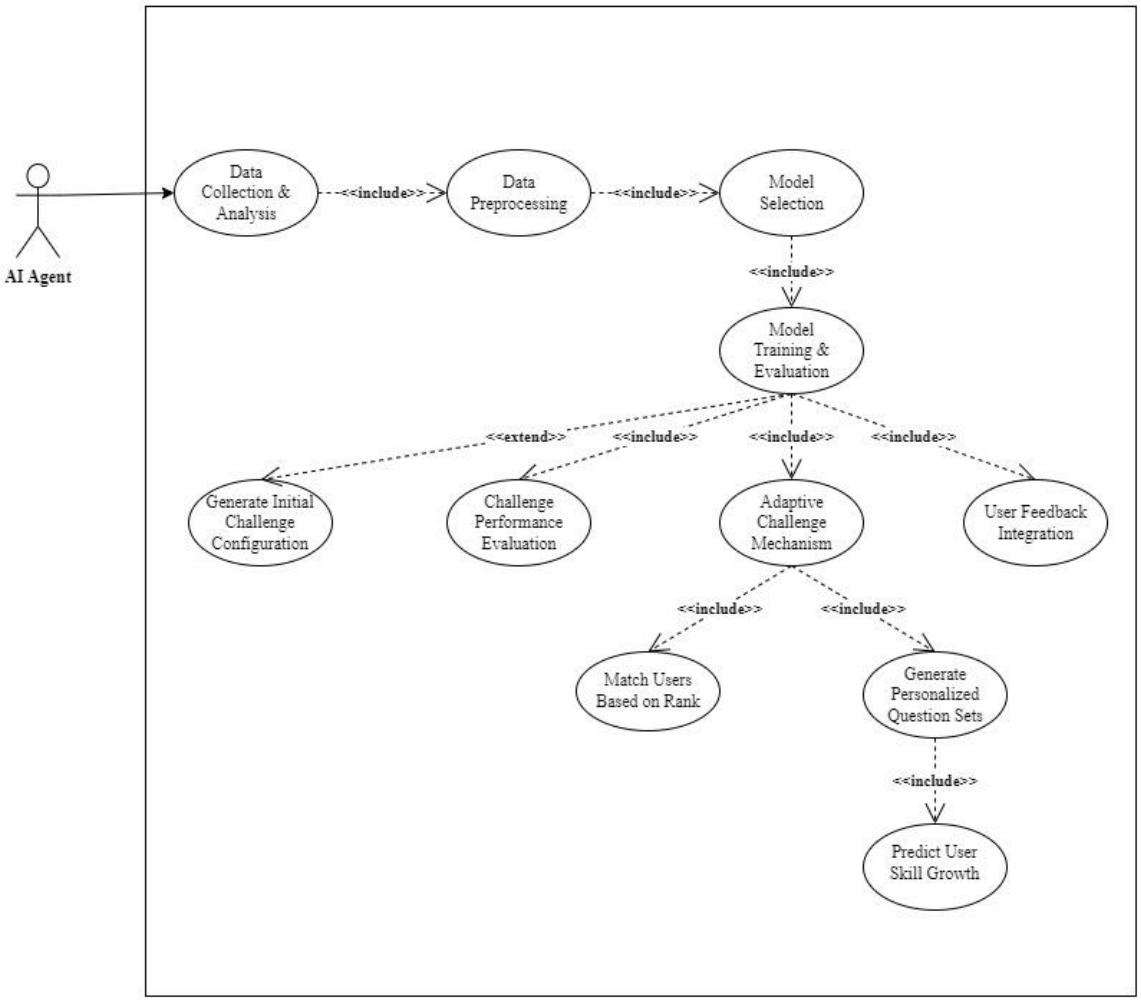
Related Use Cases	None
Postcondition	Shop items are successfully added, updated, or removed, and changes reflect on the user-facing shop page.
Main Flow	Admin navigates to the "Shop Page Settings" section. Admin views the list of existing shop items. Admin selects to create, edit, or delete a shop item. 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	Admin navigates to the "Keycloak Settings" section.

	The admin modifies the necessary settings using the Keycloak Admin Panel and submits the changes. System updates the Keycloak server with the new configuration.
Alternate Flows	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	Admin navigates to the "Challenge Settings" section. Admin views the list of current challenges. Admin selects to create, edit, or delete a challenge. 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



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	The AI Agent collects detailed data from user interactions, including solving time, accuracy rates, and question difficulty, to build a comprehensive performance profile. The system calculates aggregate metrics such as the total number of questions solved, regularity of participation, and performance trends to provide meaningful insights. The collected data is securely stored in a structured format, ensuring it is ready for preprocessing in subsequent steps.
Alternate Flows	If required user data is unavailable, the system logs the missing data, notifies relevant modules, and skips analysis for the incomplete entries while continuing with the available data.

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.

ÇANKAYA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

Software Design Document (SDD)

CENG 407

Innovative System Design and Development I

#6
GoatCapella

Çağdaş GÜLEÇ
Tarık ŞEN
İrem Beyza AYDOĞAN
Şeref Berkay KAPTAN
Beyza TOZMAN

Serdar ARSLAN
Bilgin AVENOĞLU

1. Introduction

This Software Design Document (SDD) presents a technical overview of the GoatCapella project, an innovative platform aimed at elevating the experience of competitive programming and algorithmic problem-solving. Drawing upon the insights and requirements laid out in the Software Requirements Specification (SRS) and the educational methodologies explored in the Literature Review, this SDD brings together architectural decisions, system decompositions, user interface considerations, and database design strategies. GoatCapella seeks to consolidate challenges and data from multiple existing coding platforms—such as LeetCode, CodeWars, and CodeForces—into a single, immersive environment where learners can both hone their technical skills and engage with peers. In doing so, GoatCapella incorporates a variety of learning and motivational frameworks, including challenge-based

learning, gamification, and social learning methodologies. These frameworks, described in detail in the Literature Review, help shape the platform's core features and ensure that it serves as an engaging, enjoyable, and educational ecosystem for diverse user groups.

1.1. Purpose

The purpose of this Software Design Document (SDD) is to define the architectural and design approach for developing GoatCapella, a web-based platform for coding challenges and competitive programming. This document provides the technical details required for developers, testers, and stakeholders to understand the system's design and ensure its implementation aligns with defined goals. GoatCapella integrates various coding platforms (e.g., LeetCode, CodeWars) and introduces features like gamification, challenge-based learning, and social collaboration to enhance user experience.

1.2. Scope

GoatCapella's scope extends beyond merely aggregating challenges from existing coding platforms. Its mission is to offer a unified and enhanced problem-solving environment where users can interact with multiple challenge sources, create customized challenges, track their own progress, and receive AI driven recommendations. Key functionalities include:

- **API Integrations:** Securely connecting to and extracting challenge-related data from third party platforms (LeetCode, CodeWars, CodeForces, etc.).
- **Challenge Management**: Enabling users and administrators to create, join, or monitor various types of challenges (public, private, speed-run, or adaptive challenges).
- **Adaptive Learning & AI Components:** Analyzing user performance to adjust question difficulty and recommend new challenges based on collaborative filtering, reinforcement learning, or content-based models.
- **Gamification Mechanics:** Introducing badges, coins, daily streaks, and leaderboards to foster ongoing engagement and reflect progress.
- **Community & Feedback Features:** Offering commentary tools, solution reviews, and collaborative spaces for each challenge, thus promoting the social learning methodology highlighted in the Literature Review.

Potential future additions to the platform's roadmap, as hinted at in both the SRS and Literature Review, could include exploring the development of a mobile app to improve accessibility and considering the creation of a proprietary problem-solving system to possibly reduce reliance on external platforms.

1.3. Definitions, Acronyms, and Abbreviations

Although many of the key terms used in GoatCapella (e.g., SRS, CBL, Gamification) have been defined in the Software Requirements Specification (SRS), the Software Design Document (SDD) introduces a few additional, design-focused terms that warrant clarification:

- **Architectural Diagram:** A high-level representation of the system's structure, illustrating how various components (modules, services, third-party integrations) interact. This diagram helps stakeholders visualize the "big picture" of the application and understand major communication pathways.
- **ER Diagram (Entity-Relationship Diagram):** A graphical illustration of the database's entities (e.g., Users, Challenges, Solutions) and the relationships among them. In the SDD, ER Diagrams help convey how data is stored and interconnected in relational databases (e.g., PostgreSQL).
- **Sequence Diagram:** A type of UML (Unified Modeling Language) diagram that shows how processes interact through time. In this document, sequence diagrams map out the flow of messages or events between objects (e.g., the application server, the database, the user's browser) in a specific scenario, such as creating a new challenge or fetching user progress.
- **Activity Diagram:** A type of UML diagram that represents the flow of activities or tasks in a system. It is used to visualize the sequence and conditions of workflows, such as a user registering for the platform, submitting a challenge, or receiving feedback. Activity diagrams are particularly useful for identifying decision points and parallel processes, providing a detailed view of the system's functional behavior.
- **Use Case Diagram:** A UML diagram that depicts the interactions between the system and its various actors (e.g., regular users, sub-admins, main admins). Use case diagrams help define functional requirements by illustrating the distinct actions each actor can perform.
- **Class Diagram:** Another UML construct used to represent the static structure of the system by showing its classes, attributes, operations, and the relationships among them. In the SDD, class diagrams clarify how the application's main objects (e.g., User, Challenge, Submission) are modeled in code.
- **UML (Unified Modeling Language):** A standardized modeling language that offers a variety of diagrams (such as Sequence Diagrams, Class Diagrams, and Use Case Diagrams) for describing the design, structure, and behavior of a software system.

These design-oriented terms are referenced throughout the SDD to ensure clarity around how GoatCapella's architecture, data models, and workflows are documented and communicated.

1.4. Overview

This document is structured to offer a clear progression from high-level design objectives to more granular details.

A. Introduction

Provides the high-level background and motivations behind GoatCapella, referencing the SRS and Literature Review for context. It outlines the purpose, scope, and key terminology relevant to the project's implementation.

B. System Design

- **Architectural Design:** Discusses the system architecture, highlighting the division of responsibilities among modules.
- **Decomposition Description:** Breaks down the solution into smaller components or services, detailing how each module communicates within the broader ecosystem.
- **System Modeling:** Shows activity diagrams and sequence diagrams to illustrate the operational lifecycle of features (e.g., challenge creation, AI-based question suggestion, user progress tracking).
- **Database Design:** Explains the rationale for choosing both a relational database (PostgreSQL) for structured data and a NoSQL store (MongoDB) for unstructured or flexible data needs.

C. User Interface Design

Presents the primary page flows (landing page, challenge overview, challenge creation, profile page, admin dashboard, etc.).

1.5. Version History

A summary of the document's revision history ensures transparency and traceability:

Version	Date	Description
1.0	27.12.2024	Initial Release

2. System Design

2.1. Architectural Design

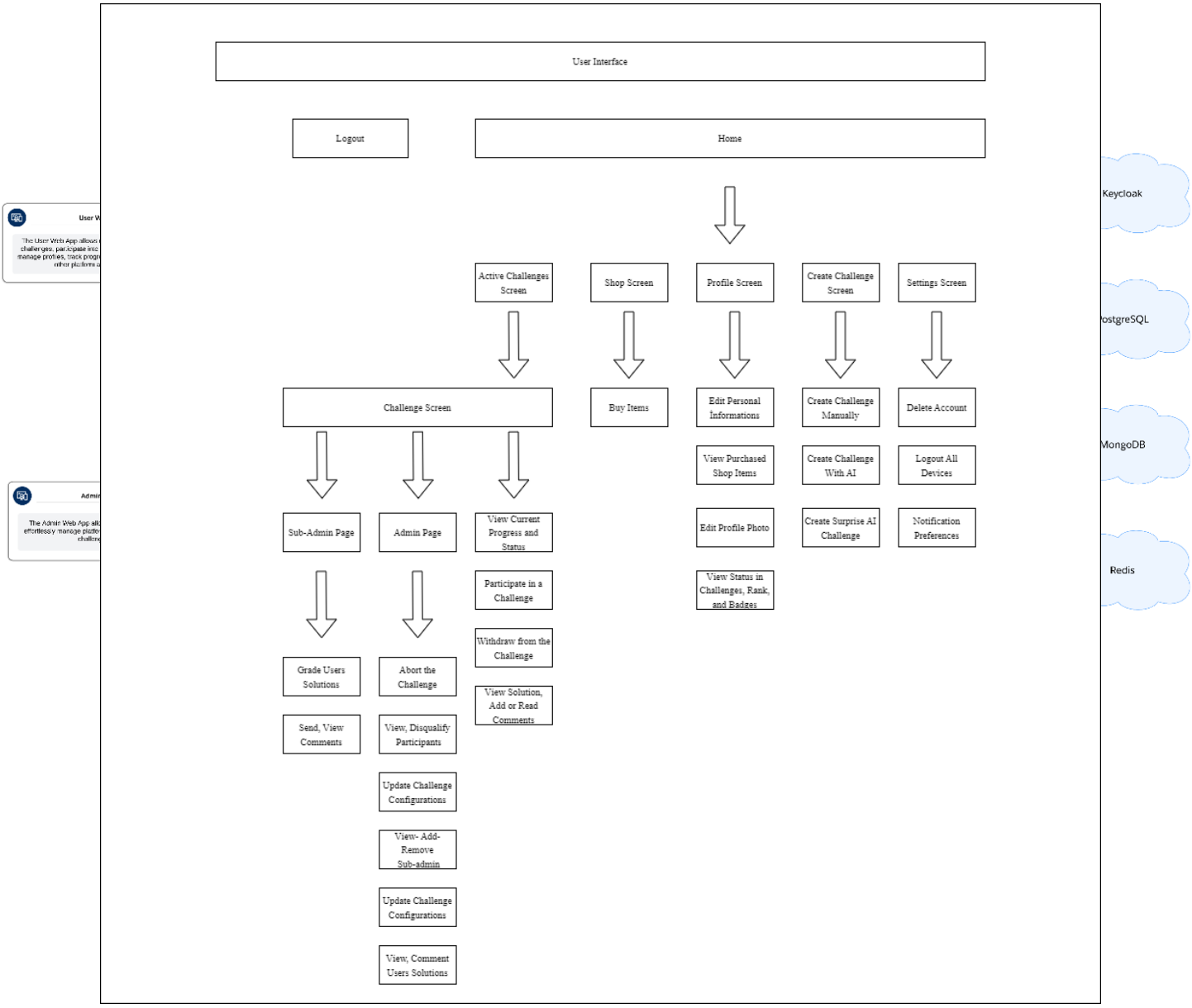


Figure 4. Layered Architecture Design for System

2.2. Decomposition Description

1. User Interface

Main Screen:

- It is the section where users can access the shop, profile, active challenges, participated challenges, and settings.

- Facilitate navigation between different challenges and features.

View Components:

- Displays challenges, leaderboards, and user achievements.
- Includes screens for initiating, participating in, and tracking challenges.

2. **Analytics and Personalization**

AI Data Analytics:

- It collects and analyzes user data to understand the user's preferences and interactions.

AI-Driven Personalization Manager:

- Utilizes artificial intelligence to perform data analysis, creating new challenges or recommending existing challenges based on the user's previous participation in challenges.

3. **Shop**

Shop Manager:

- Provides an interface where users can view items such as challenge boosters, extra attempts, or special content.
- Handles purchase transactions exclusively using in-app coins and updates the coin balance accordingly.

Item Catalog:

- Stores prices (in coins) and descriptions of challenge-related items.

4. **Settings**

User Preferences:

- Allows users to configure their application preferences.
- Provides options to enable or disable specific types of notifications.

Account Manager:

- Allows users to permanently delete their accounts.
- Enables users to log out from all active sessions on multiple devices.

5. **Security and Privacy**

Data Protection:

- Advanced encryption techniques are used to secure user data.
- Privacy and security are maintained during data transmission and storage.

Privacy Controls:

- It provides privacy settings that allow users to control their personal information and manage how it is shared.

6. **Challenge Process**

Manual Challenge Creation:

- Challenge admins can create their own challenges by defining settings, duration, difficulty, and rewards.
- Challenge admins can add a sub-admin and invite other users to participate in the challenge.

AI-Assisted Challenge Creation:

- Challenge admins can customize a few settings and let AI generate the rest for optimal user participation and engagement.

Surprise AI-Generated Challenges:

- The AI Agent analyzes the user's preferences, historical performance data, and activity patterns to create a challenge configuration tailored to the user's needs.

Active Challenges:

- Users can view a list of active challenges they are currently participating in and track their progress.
- They can see details of existing challenges, including their names, durations, participants, and rewards.

Features:

- Users can filter or search for challenges based on criteria such as type, duration, or rewards.

7. **Challenge Management Responsibilities**

Admin Responsibilities:

- Create or abort challenges.
- View and manage participants, assign/remove sub-admins.
- Monitor progress and grade submissions.

Sub-Admin Responsibilities:

- Grade submissions, manage comments, and track progress.

Collaboration:

- Admins oversee sub-admin actions to maintain control and accountability.

8. **Notifications and Alerts**

   Notification Manager:

   - Manages in-app notifications and real-time alerts for challenge updates, deadlines, or results.
   - Customizes notifications based on user preferences.

   Event Triggers:

   - Sends notifications for significant events such as the addition of a new challenge, deadline reminders, and results of completed challenges.

9. **User Engagement and Social Features**

   Engagement Tools:

   - Provides features like leaderboards and badges to motivate users.

   Social Interaction:

   - Competing with other users also increases motivation and engagement.

2.3. System Modeling

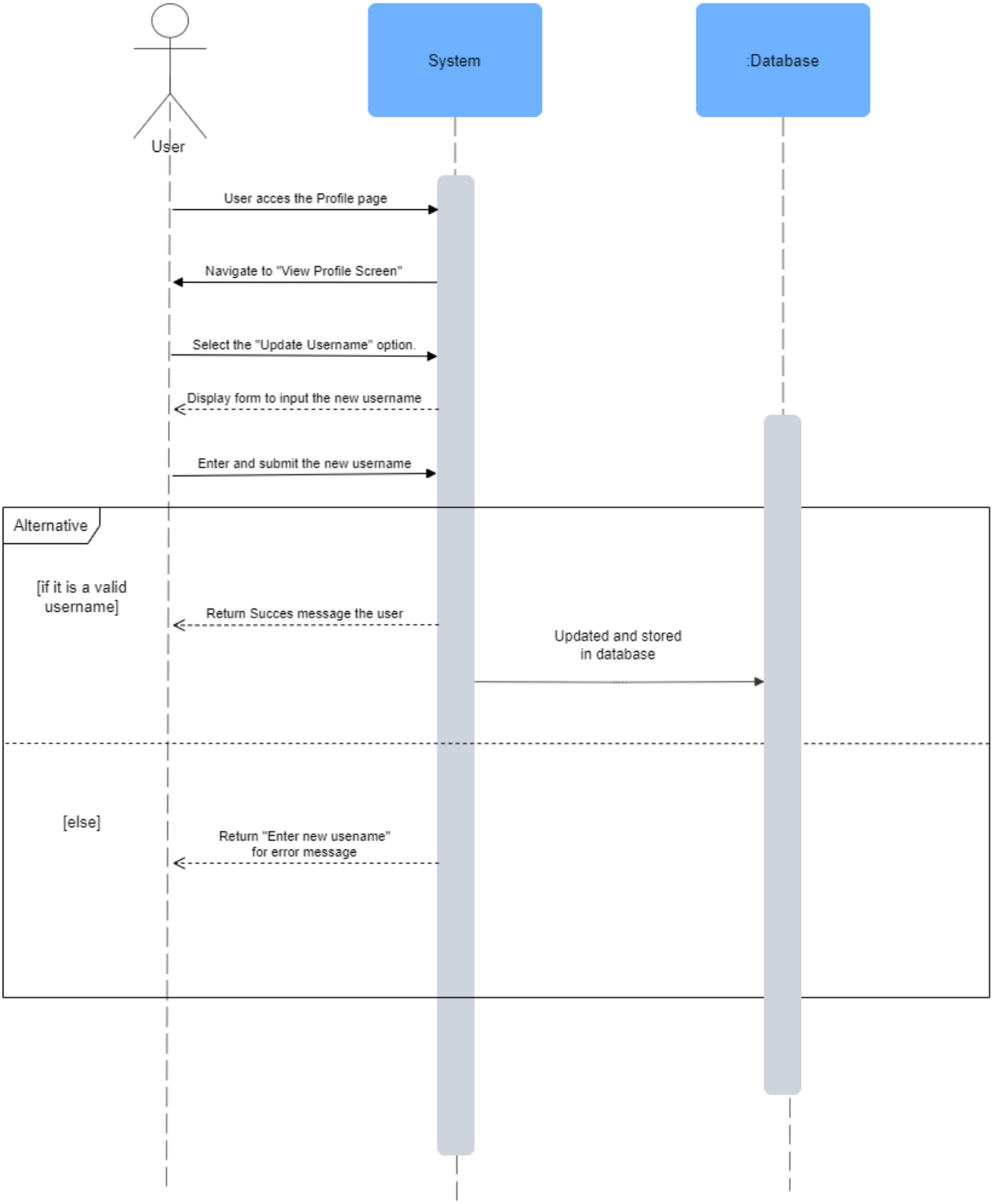


Figure 5. Sequence Diagram of Update Username

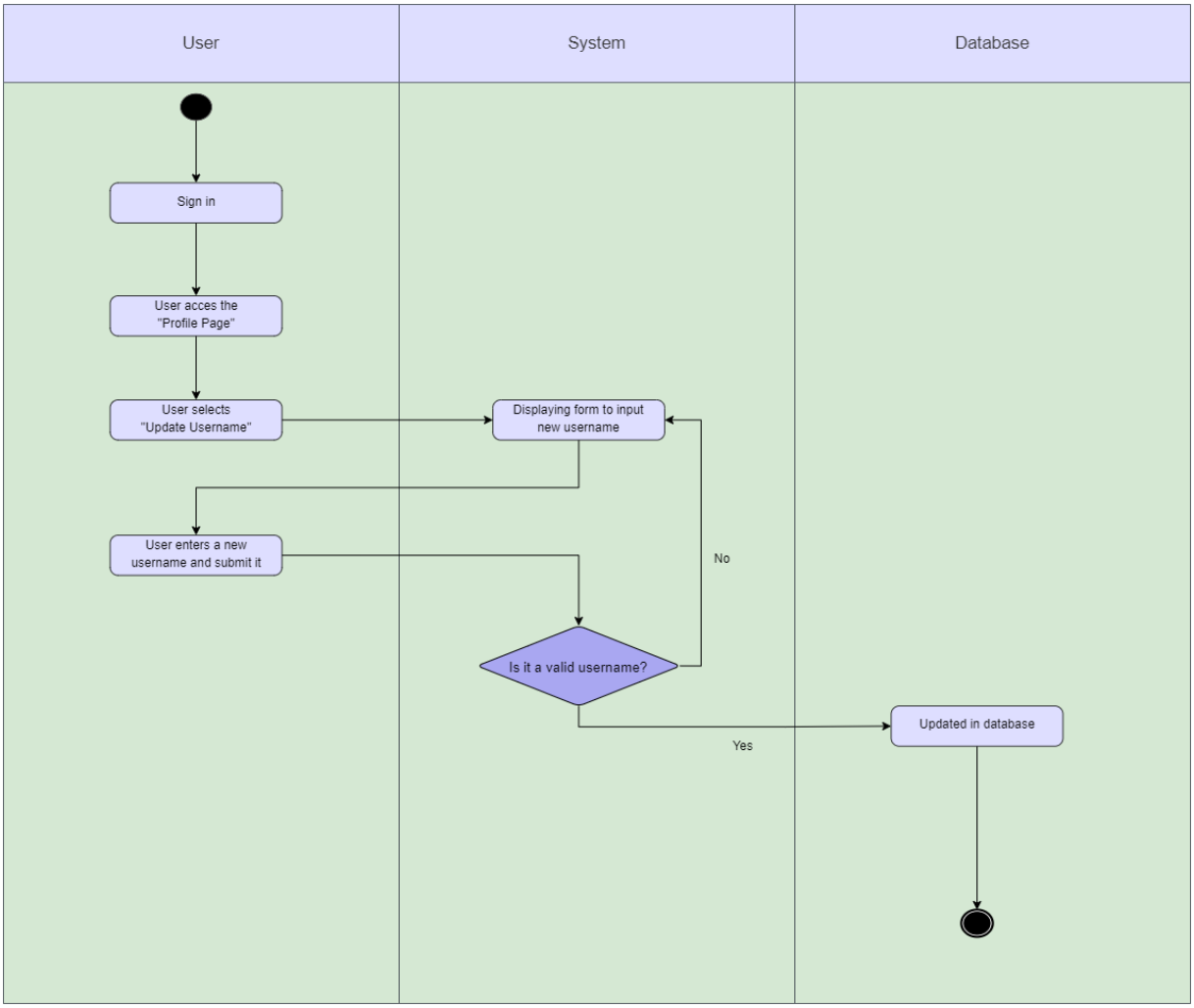


Figure 6. Activity Diagram of Update Username

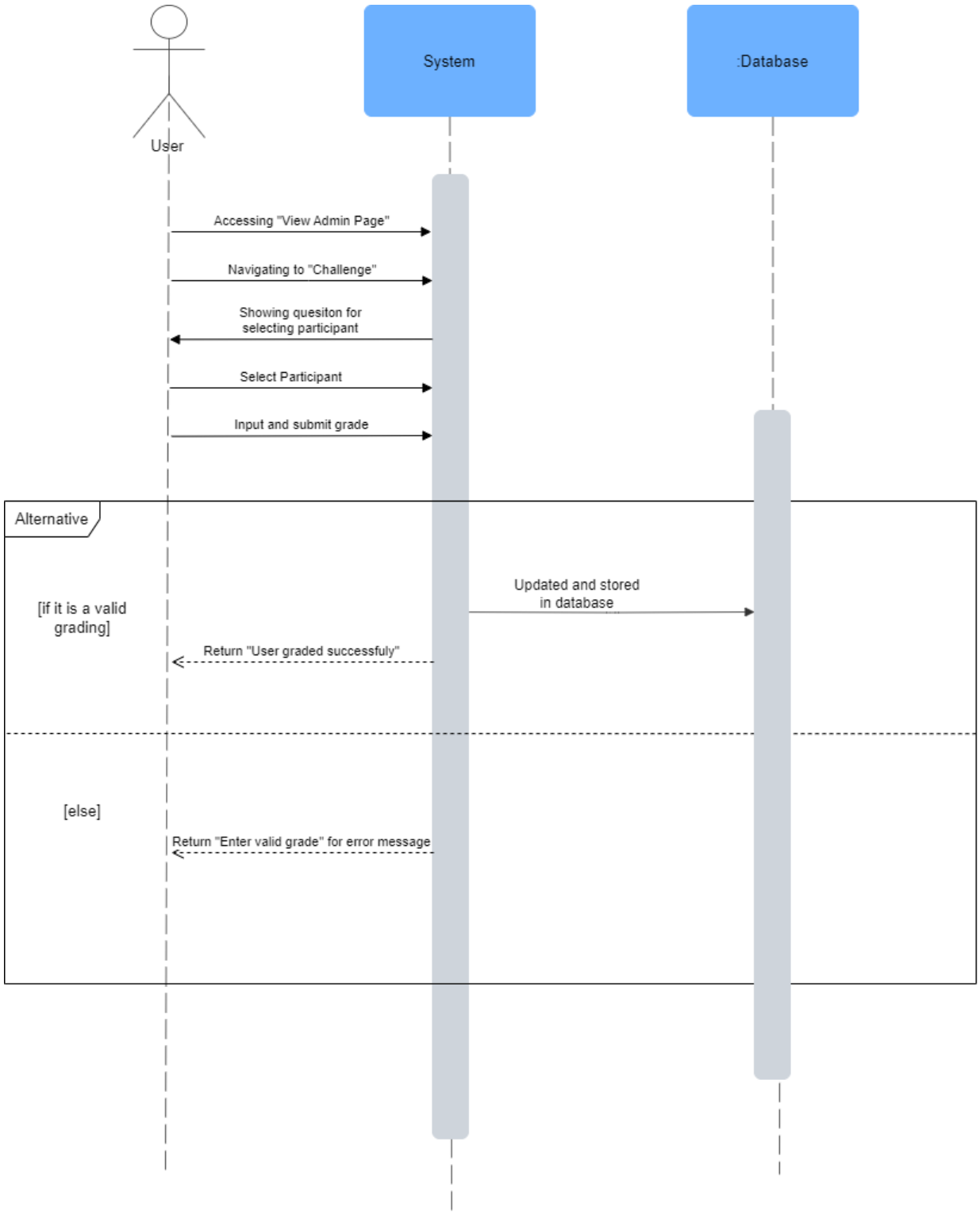


Figure 7. Sequence Diagram of Admin Grading

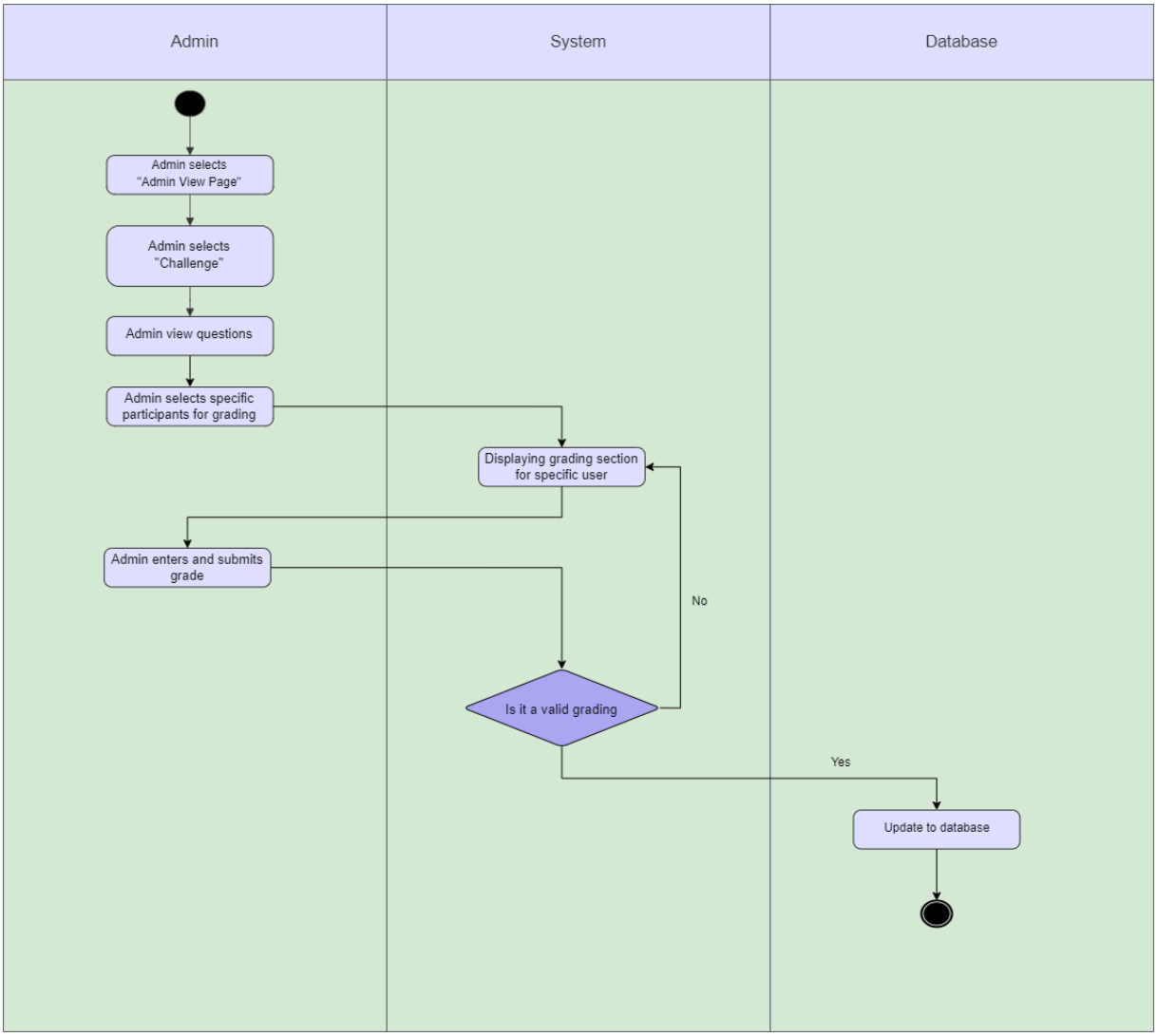


Figure 8. Activity Diagram of Admin Grading

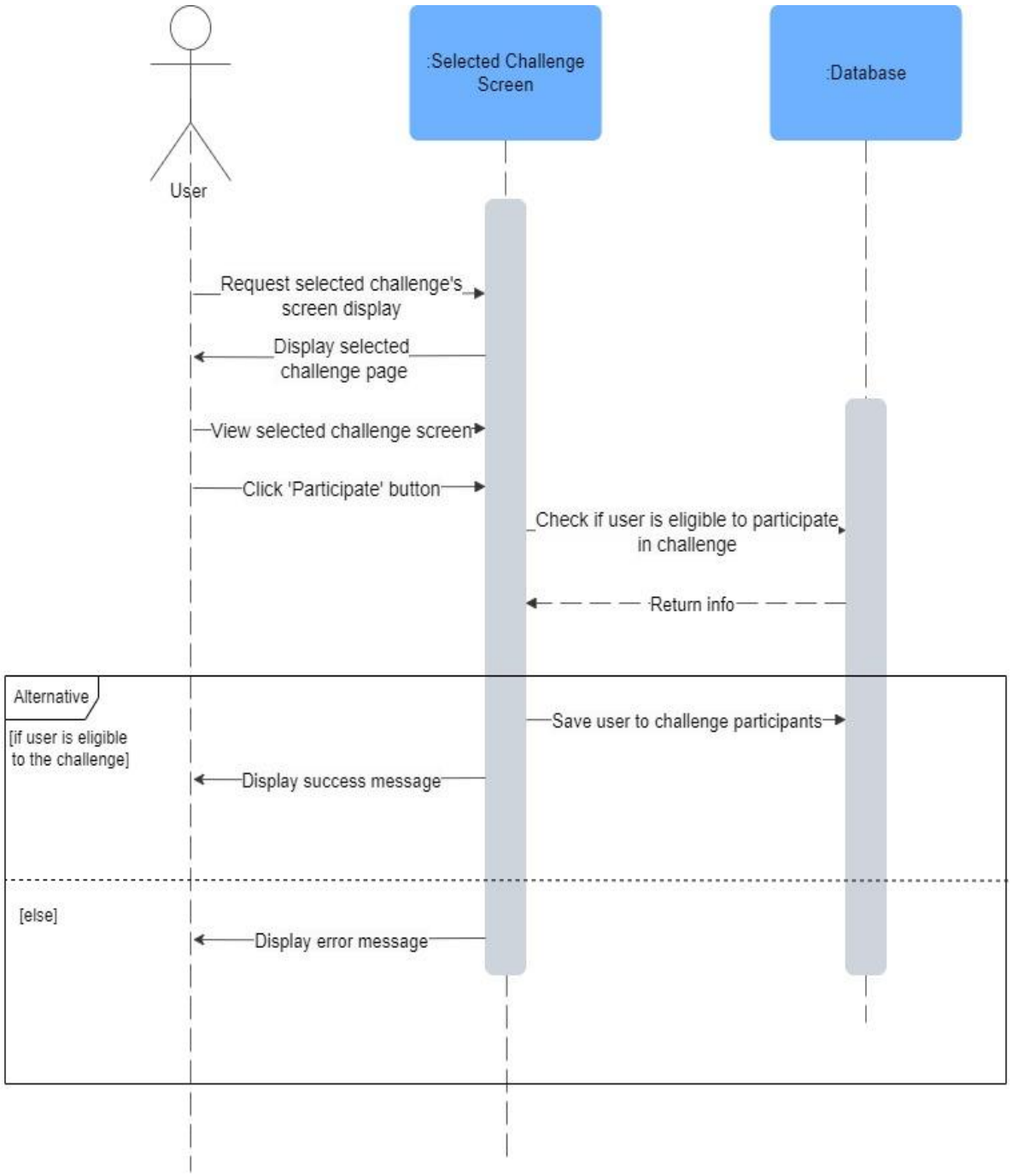


Figure 9. Sequence Diagram of Participant Challenge

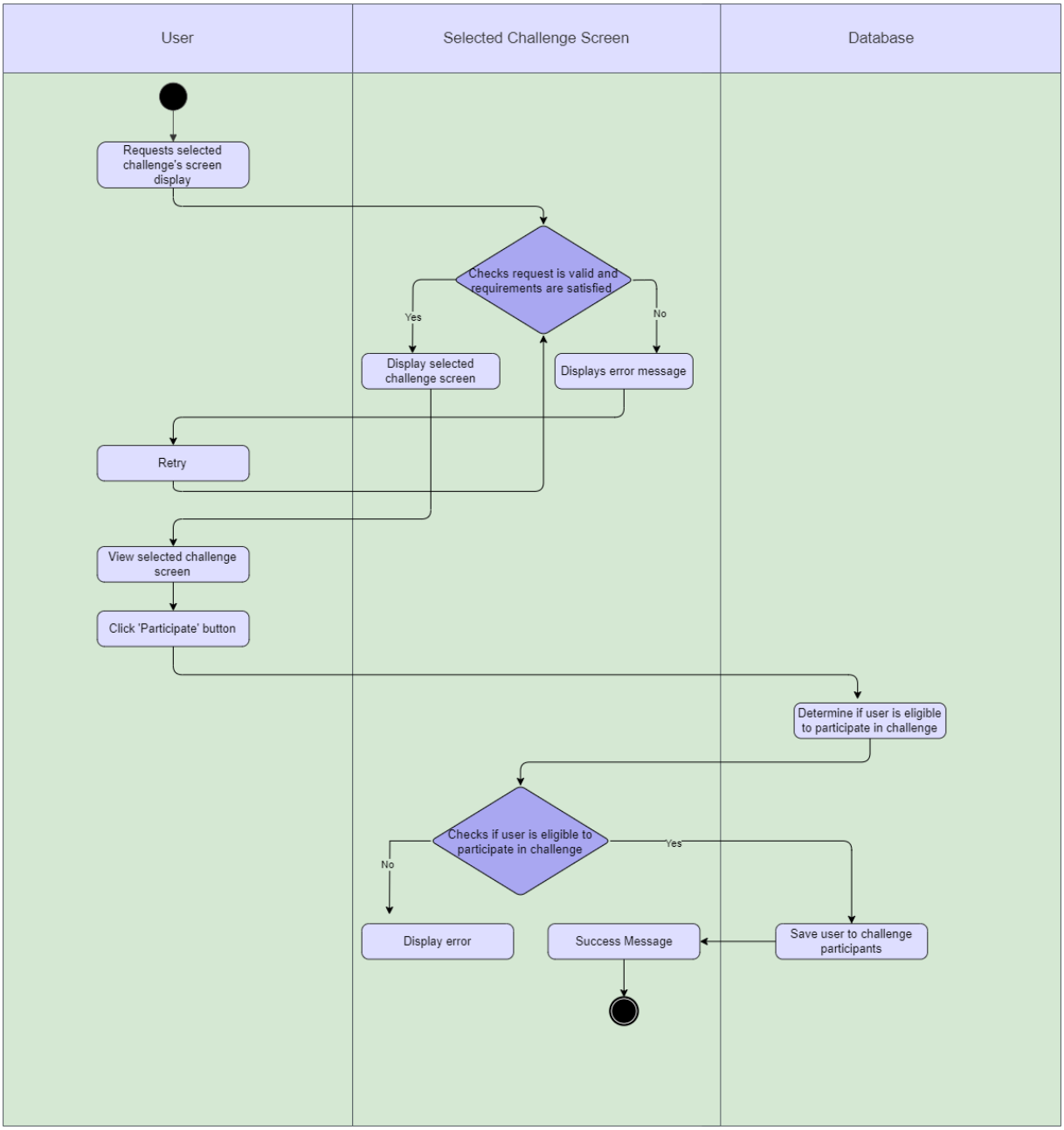


Figure 10. Activity Diagram of Participant Challenge

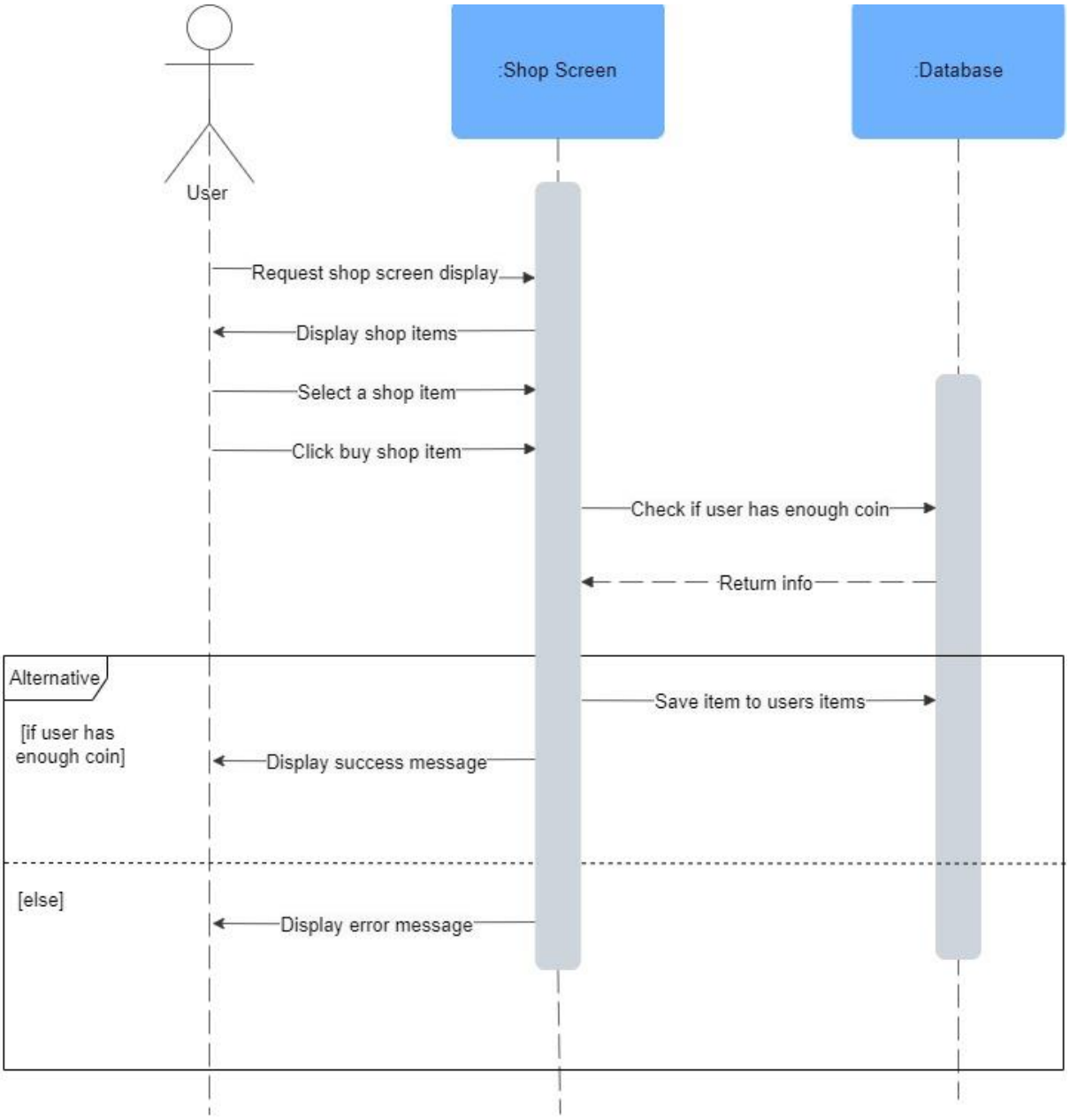


Figure 11. Sequence Diagram of Shop Page

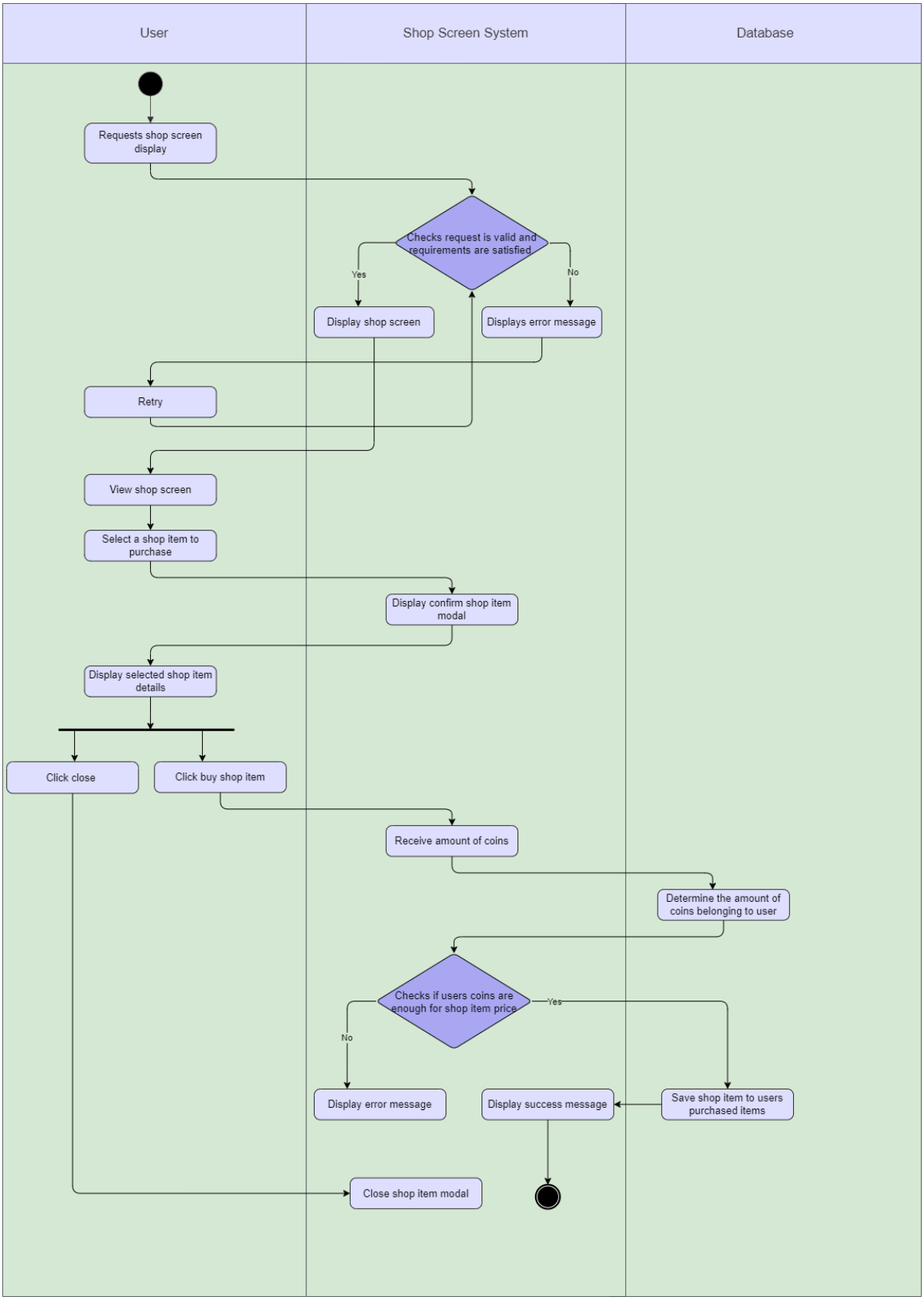


Figure 12. Activity Diagram of Shop Page

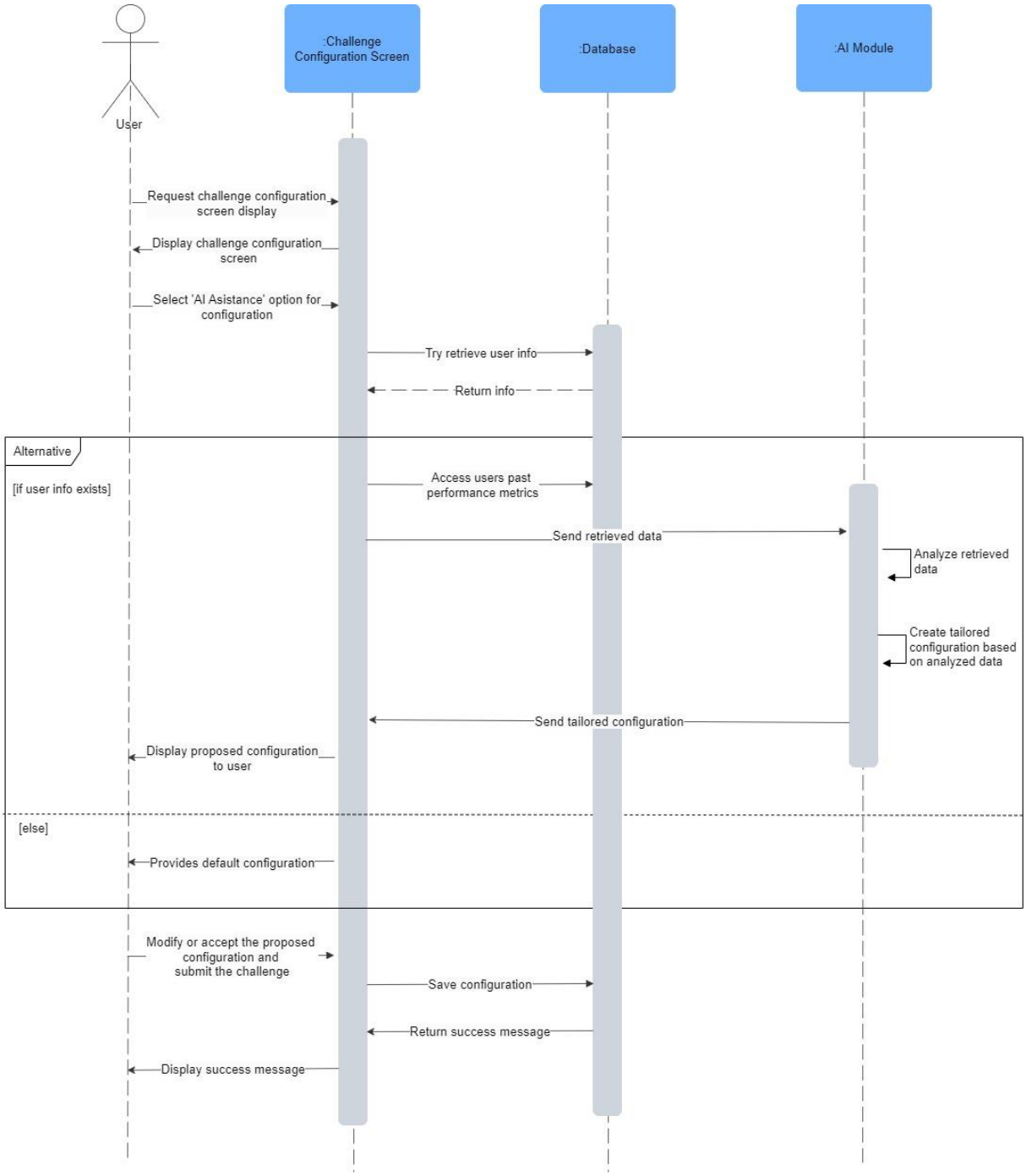


Figure 13. Sequence Diagram of AI Create Challenge

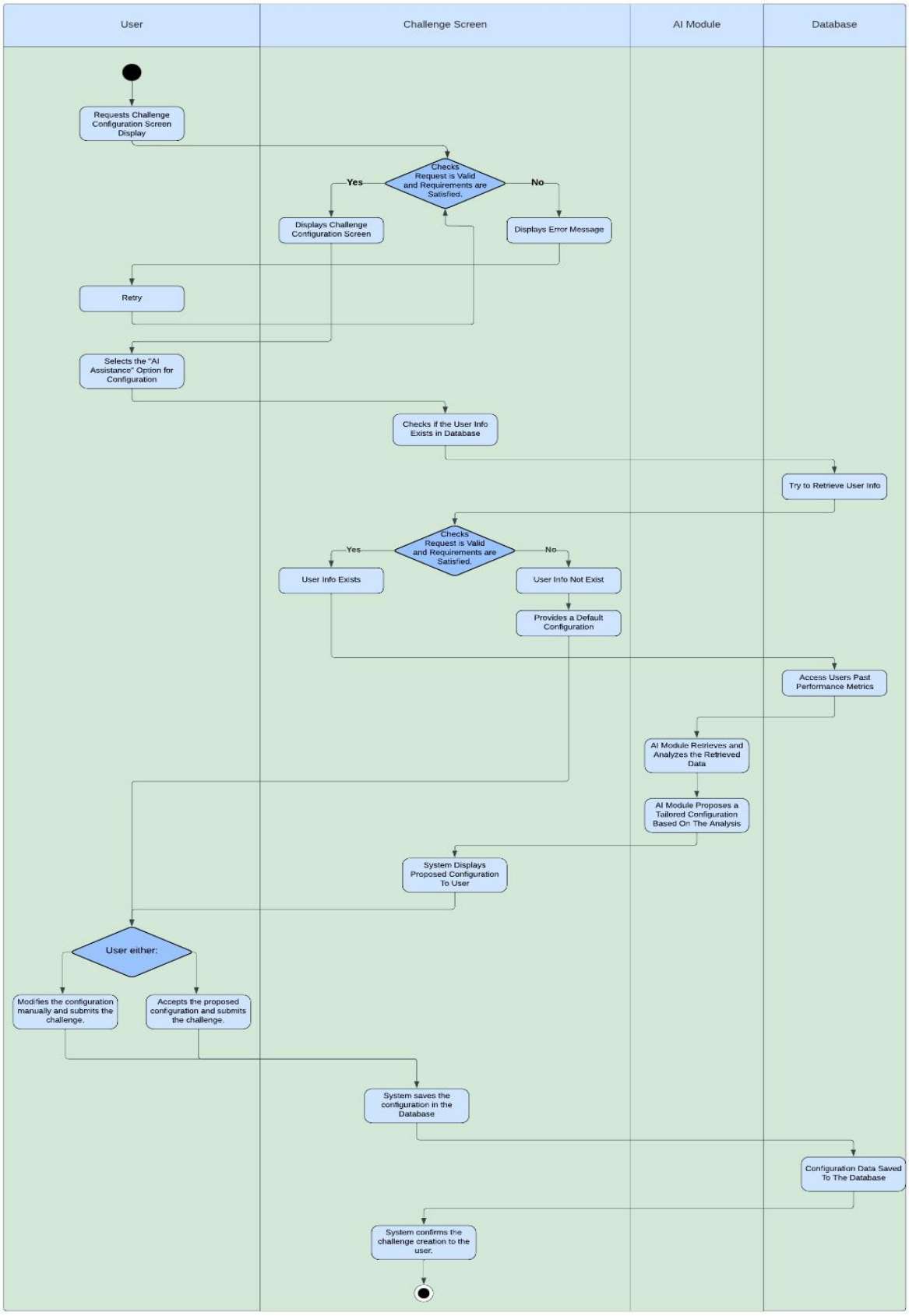


Figure 14. Activity Diagram of AI Create Challenge

2.4. Database Design

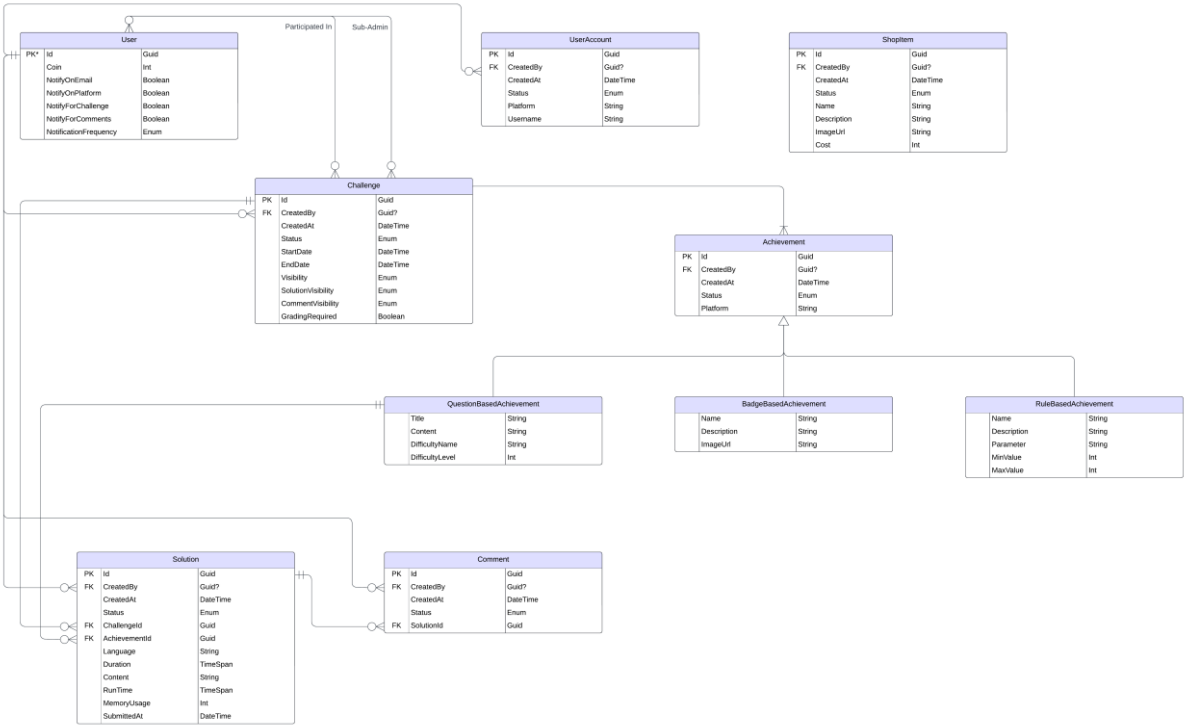


Figure 15. ER Diagram of Database Design

3. User Interface Design

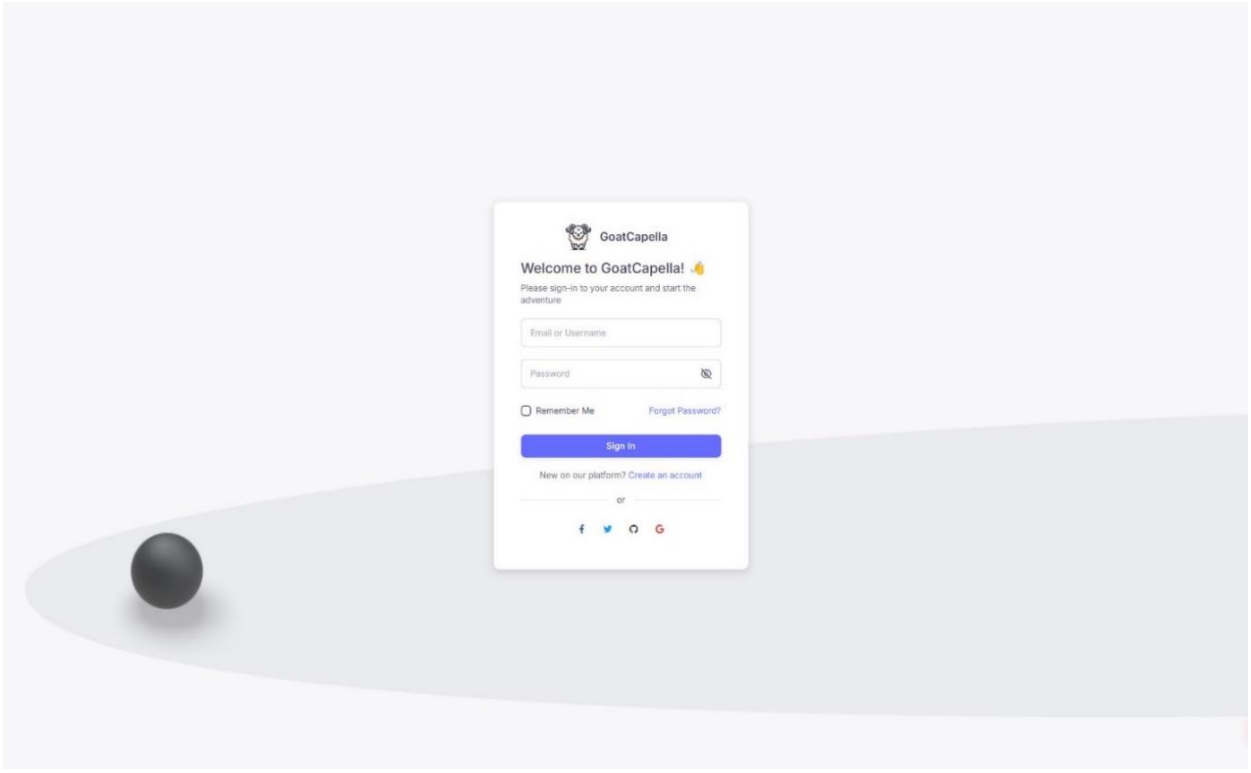


Figure 14. Sign Up Page

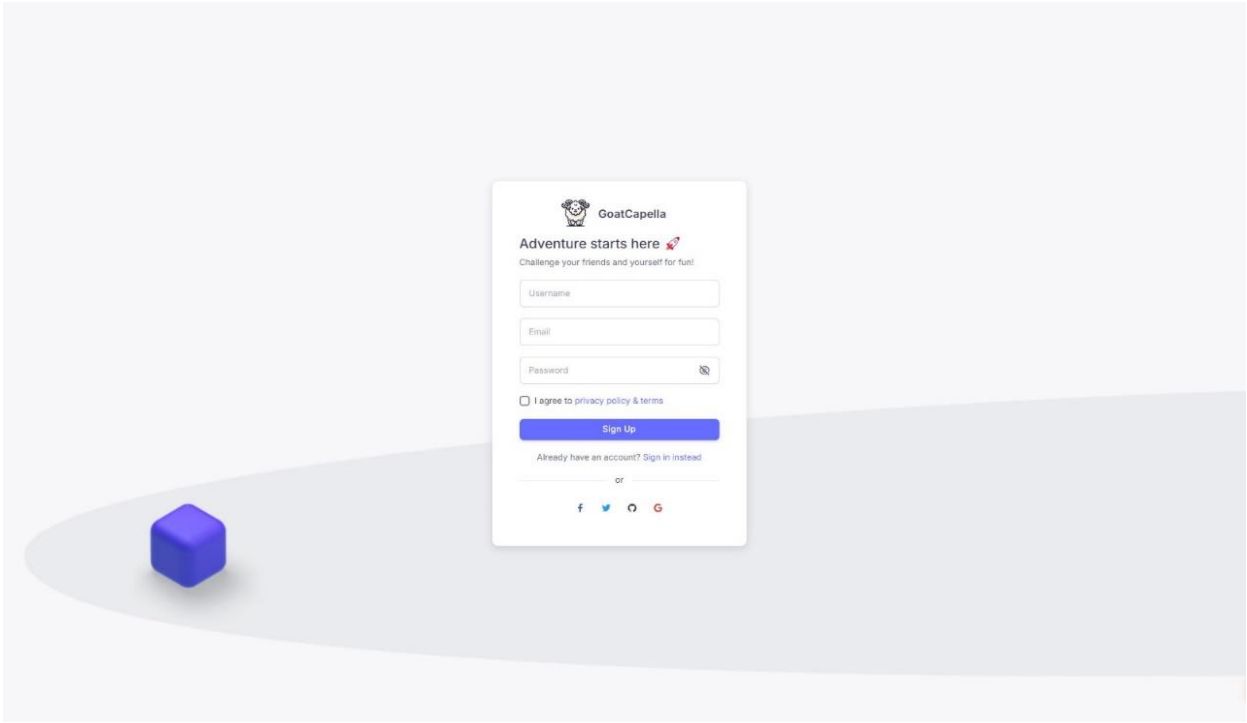


Figure 15. Sign In Page

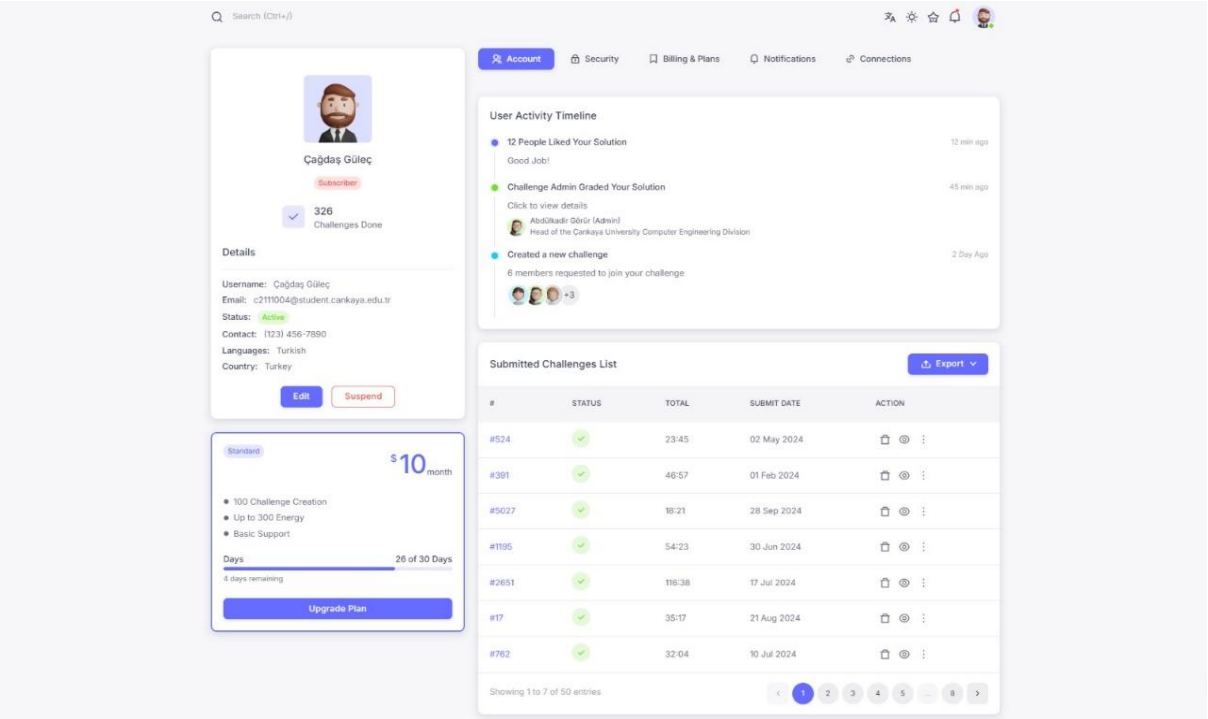


Figure 16. User Profile Page

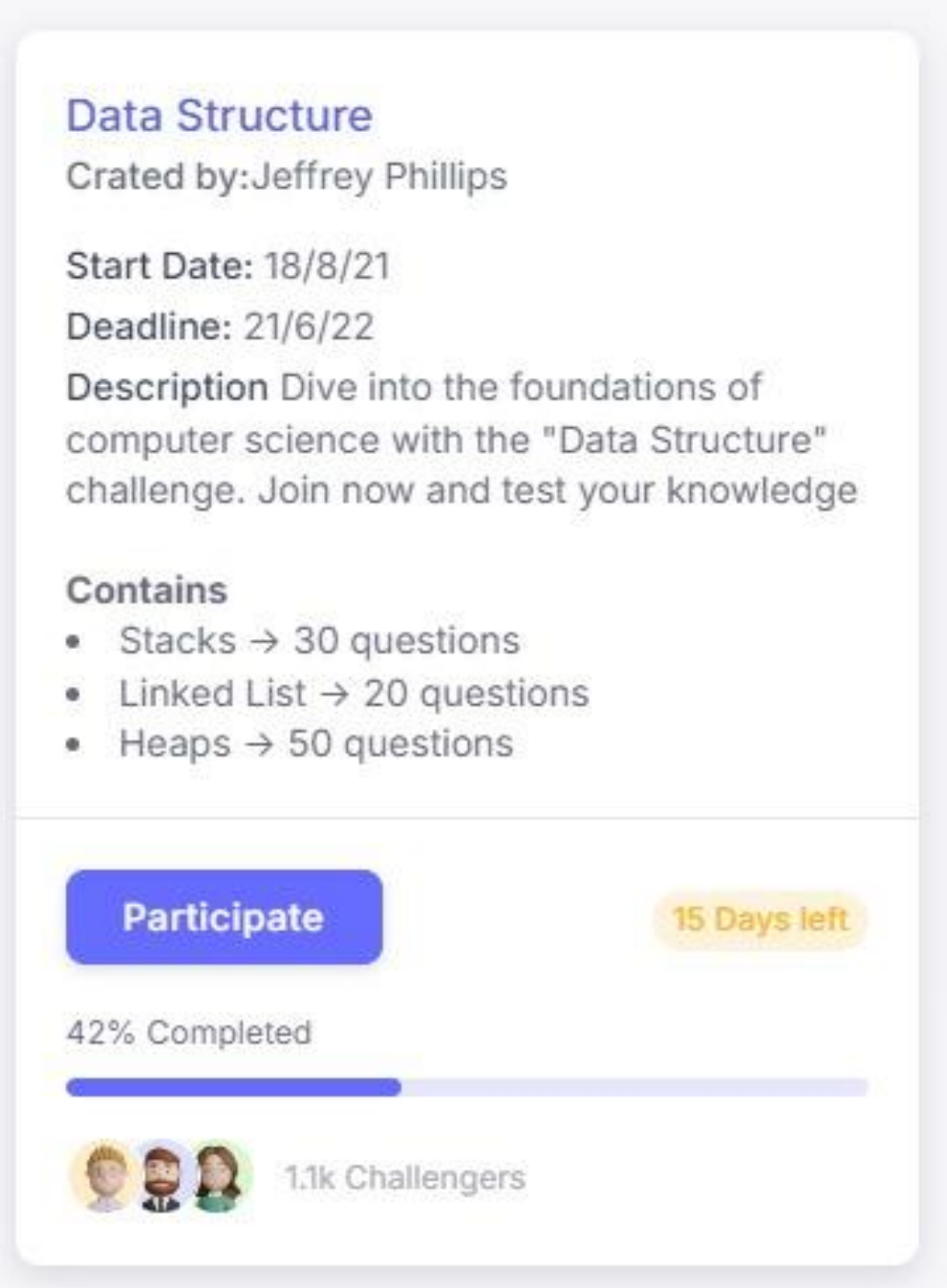


Figure 17. Participate Challenge Page

Conclusion

This report provides a detailed examination of the fundamental components of the GoatCapella project, including the Literature Review (LR), Software Requirement Specification (SRS), and Software Design Document (SDD). Each section describes the conceptual, technical, and functional aspects of the project comprehensively.

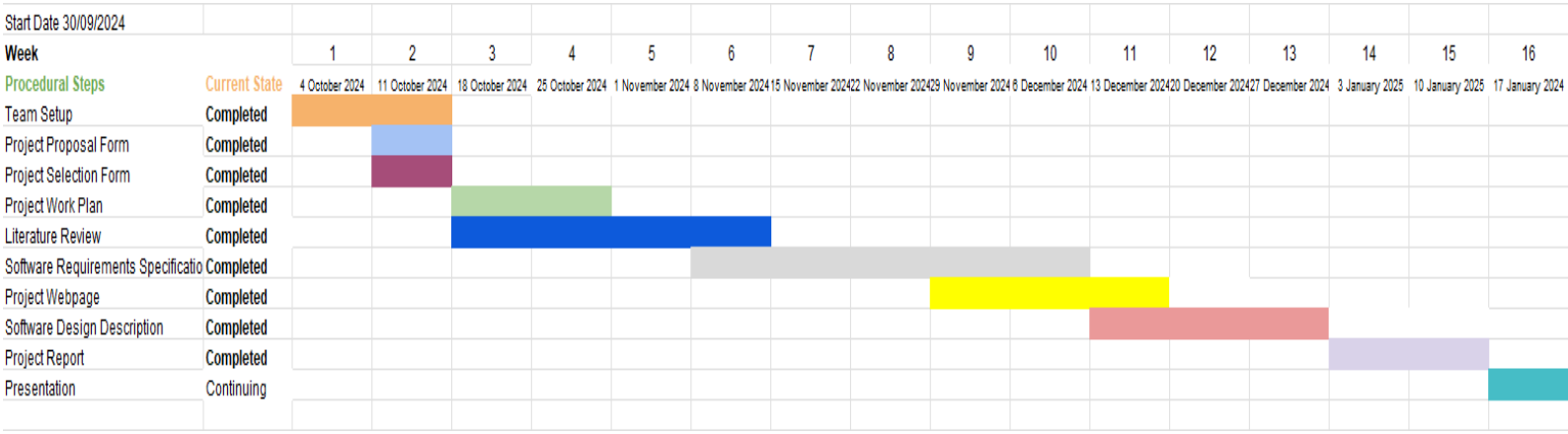
In addition to conducting research on existing applications for GoatCapella and making decisions to develop innovative features, and establishing the foundations of the application, detailed information that is analyzed and in line with the objectives of the project, as well as following technological developments, is also stated in the Literature Review section.

The point that GoatCapella aims to reach is explained in the Software Requirement Specification, where the system is compatible with target user expectations along with critical elements such as performance criteria, usability, scalability, security and privacy.

The SDD document transforms the requirements into an architectural design, detailing the structure, modules, data flow, and technical specifications of the system. This section includes diagrams and interfaces to visualize the design, providing a framework for the development process.

In conclusion, this report provides a comprehensive perspective that brings together theoretical and practical elements for the development of GoatCapella and guides future development. The review of platforms, together with related methodologies and technical evaluations, provides a comprehensive foundation for GoatCapella. The combination of requirements specification and a solid design foundation is aimed at the successful implementation of GoatCapella.

Project Work Plan



References

1. "Challenge-based learning," Wikipedia, Oct. 3, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Challenge-based_learning. [Accessed: Nov. 3, 2024].
2. M. Author et al., "Article Title," Journal Name, vol. x, no. x, pp. xxx–xxx, Oct. 2024. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC10611935. [Accessed: Nov. 3, 2024].
3. M. Author and M. Author, "Asymmetric Adaption in Social Learning: Understanding the Dilemma of Competition and Cooperation," Research Publication, Oct. 2024. [Online]. Available: https://www.researchgate.net/publication/383199409_Asymmetric_Adaption_in_Social_Learning_Understanding_the_Dilemma_of_Competition_and_Cooperation. [Accessed: Nov. 3, 2024].
4. M. Author et al., "A Comparative Evaluation of the Effect of Social Comparison, Competition, and Social Learning in Persuasive Technology on Learning," Research Publication, Jul. 2021. [Online]. Available: https://www.researchgate.net/publication/353113397_A_Comparative_Evaluation_of_the_Effect_of_Social_Comparison_Competition_and_Social_Learning_in_Persuasive_Technology_on_Learning. [Accessed: Nov. 3, 2024].
5. "LeetCode," LeetCode, Oct. 2024. [Online]. Available: https://leetcode.com. [Accessed: Nov. 3, 2024].
6. "Finding Kata," Codewars Documentation, Oct. 2024. [Online]. Available: https://docs.codewars.com/getting-started/finding-kata. [Accessed: Nov. 3, 2024].
7. "Languages," Codewars Documentation, Oct. 2024. [Online]. Available: https://docs.codewars.com/languages. [Accessed: Nov. 3, 2024].
8. "Kata Concepts," Codewars Documentation, Oct. 2024. [Online]. Available: https://docs.codewars.com/concepts/kata. [Accessed: Nov. 3, 2024].
9. "Introduction," Codewars Developer, Oct. 2024. [Online]. Available: https://dev.codewars.com/#introduction. [Accessed: Nov. 3, 2024].
10. "About Us," CodeChef, Oct. 2024. [Online]. Available: https://www.codechef.com/aboutus. [Accessed: Nov. 3, 2024].
11. "SPOJ," Wikipedia, Oct. 3, 2024. [Online]. Available: https://en.wikipedia.org/wiki/SPOJ. [Accessed: Nov. 3, 2024].
12. "Sphere Engine," SPOJ, Oct. 2024. [Online]. Available: https://www.spoj.com/sphereengine. [Accessed: Nov. 3, 2024].

13. M. Mirzayanov, "5 Years of Codeforces," Codeforces Blog, Jun. 2015. [Online]. Available: https://codeforces.com/5years#:~:text=Codeforces%20was%20launched%20by%20me. [Accessed: Nov. 3, 2024].

14. M. Author, "Blog Entry," Codeforces Blog, Oct. 2024. [Online]. Available: https://codeforces.com/blog/entry/121114. [Accessed: Nov. 3, 2024].

15. "Codeforces API Methods," Codeforces Documentation, Oct. 2024. [Online]. Available: https://codeforces.com/apiHelp/methods. [Accessed: Nov. 3, 2024].

16. "About VJudge," VJudge, Oct. 2024. [Online]. Available: https://vjudge.net. [Accessed: Nov. 3, 2024].

17. "About Exercism," Exercism, Oct. 2024. [Online]. Available: https://exercism.org/about. [Accessed: Nov. 3, 2024].

18. "Developers," Coderbyte, Oct. 2024. [Online]. Available: https://coderbyte.com/developers. [Accessed: Nov. 3, 2024].

19. "About Codility," Codility, Oct. 2024. [Online]. Available: https://www.codility.com. [Accessed: Nov. 3, 2024].

20. "API Documentation," Codility, Oct. 2024. [Online]. Available: https://codility.com/api-documentation/#overview. [Accessed: Nov. 3, 2024].

21. "What is ASP.NET Core," Microsoft Learn, Oct. 2024. [Online]. Available: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core. [Accessed: Nov. 3, 2024].

22. "Keycloak," Keycloak, Oct. 2024. [Online]. Available: https://www.keycloak.org. [Accessed: Nov. 3, 2024].

23. "Baeldung," Baeldung, Oct. 2024. [Online]. Available: https://www.baeldung.com. [Accessed: Nov. 3, 2024].

24. S. Brady, "Scott Brady," Scott Brady, Oct. 2024. [Online]. Available: https://www.scottbrady91.com. [Accessed: Nov. 3, 2024].

25. "Getting Started with Redis," Redis Documentation, Oct. 2024. [Online]. Available: https://redis.io/docs/latest/get-started/. [Accessed: Nov. 3, 2024].

26. "Docker Guides," Docker Documentation, Oct. 2024. [Online]. Available: https://docs.docker.com/guides/. [Accessed: Nov. 3, 2024].

27. "Docker Compose," Docker Documentation, Oct. 2024. [Online]. Available: https://docs.docker.com/compose/. [Accessed: Nov. 3, 2024].

28. "What is Container Orchestration," Red Hat, Oct. 2024. [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-container-orchestration. [Accessed: Nov. 3, 2024].

29. "Docker Swarm vs Kubernetes," IBM, Oct. 2024. [Online]. Available: https://www.ibm.com/think/topics/docker-swarm-vs-kubernetes. [Accessed: Nov. 3, 2024].

30. L. Breiman, "Random Forests," University of California, Berkeley, Oct. 2001. [Online]. Available: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf. [Accessed: Nov. 3, 2024].

31. M. Author, "Deep Q-Networks," METU Resources, Spring 2017. [Online]. Available: https://user.ceng.metu.edu.tr/~emre/resources/courses/AdvancedDL_Spring2017/DQN_Muhammed.pdf. [Accessed: Nov. 3, 2024].

32. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," NeurIPS Proceedings, 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf. [Accessed: Nov. 3, 2024].

33. F. Ricci, L. Rokach, and B. Shapira, "Recommender Systems Handbook," Springer, Oct. 2024. [Online]. Available: https://www.cse.iitk.ac.in/users/nsrivast/HCC/Recommender_systems_handbook.pdf. [Accessed: Nov. 3, 2024].

34. "GraphQL," GraphQL, Oct. 2024. [Online]. Available: https://graphql.org/. [Accessed: Nov. 3, 2024].

35. "Learn GraphQL," GraphQL, Oct. 2024. [Online]. Available: https://graphql.org/learn/. [Accessed: Nov. 3, 2024].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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