



Cairo University
Faculty of Computers and Artificial Intelligence
Computer Science Department



Supervised by:

Dr. Mohamad Abdelwahab

TA: Nada Nasser

Implemented by:

20200136	<i>Hazem Adel Khalel Nabawy</i>
20200359	<i>Omar Mohmed Mostafa</i>
20200268	<i>Doha Abdelbasset Ahmed</i>
20200514	<i>Marwa Ahmed Mohamed Mubarak</i>
20200501	<i>Mahmoud Abdelrady Gad</i>

Graduation Project
Academic Year 2023-2024
Final Year Documentation

Contents

Chapter 1: Introduction	5
1.1 Motivation (Abstract)	5
1.2 Problem Definition.....	5
1.3 Our Solution	5
1.4 Gantt chart of project time plan.....	7
1.5 Development Methodology: Agile	8
1.6 Tools and Technologies Used in the Project.....	9
1.6.1 Back-end	9
1.6.2 Front-end.....	9
1.6.3 Machine Learning.....	9
1.7 Report Organization (summary of the rest of the report)	10
Chapter 2: Related work	11
2.1 Related Work.....	11
2.1.1 Similar Features.....	11
2.1.2 Distinguishing Features of Groofy Code	12
Chapter 3: System Analysis.....	13
3.1 Project specification.....	13
3.1.1 Functional Requirements.....	13
3.1.2 Non-Functional Requirements.....	14
3.2 Use case Diagram.....	15
.....	15
Chapter 4: System Design	16
4.1 System Component Diagram.....	16
4.2 System Class Diagrams.....	17
4.3 Sequence Diagrams	18
4.3.1 Ranked Match.....	18
4.3.2 Accept team invitation	19
4.3.3 Send match invitation to friend	20
Chapter 5: Implementation and Testing	29
5.1 Challenges and Solutions	29
5.1.1. Lack of Hands-on Experience with Used Technologies	29
5.1.2. Integration with Existing Websites like Codeforces.....	29
5.1.3. Transition from Node.js to Java Spring Boot	30

5.1.4. Difficulty of Java Configurations.....	30
5.1.5. Choosing Between Machine Learning Algorithms: Random Forest vs. Clustering	30
5.2. Implementation.....	30
5.2.1. Machine Learning Implementation	30
5.2.2. Back-End Implementation	37
5.2.3. Front-End Implementation.....	41
5.3. Testing.....	41
5.3.1. Back-End Testing	41
5.3.2. Front-End Testing.....	44
References	52

List of Figures

Figure 1 - Gantt Chart.....	7
Figure 2- Use case Diagram	15
Figure 3- System Component Diagram.....	16
Figure 4 - System Class Diagrams.....	17
Figure 5 - Ranked Match Sequence Diagram.....	18
Figure 6 - Accept team invitation Sequence Diagram	19
Figure 7 - Send match invitation to friend Sequence Diagram	20
Figure 8 - ERD Diagram	21
Figure 9 - Home Page	22
Figure 10 - Leader Board	22
Figure 11 - Create Clan	23
Figure 12 - Search Clan.....	23
Figure 13 - Clan Chat.....	24
Figure 14 - Profile	24
Figure 15 - Play Match.....	25
Figure 16 - Game History.....	25
Figure 17 - Solo Match.....	26
Figure 18 - Ranked Match.....	26
Figure 19 - Losing Ranked Match.....	27
Figure 20 - Wining Ranked Match	27
Figure 21 - Opponent has left.....	28
Figure 22 - Actual vs Predicted Target	34
Figure 23 - API Call Flow.....	38
Figure 24 - Swagger.....	42
Figure 25 - Postman	43
Figure 26 - required fields (login)	44
Figure 27 -Invalid credentials	44

Figure 28 - Create Post.....	45
Figure 29 - Delete Post.....	45

List of Tables

Table 1 - required fields (login)	44
Table 2 - Invalid credentials	44
Table 3 – Create Post	45
Table 4 - Delete Post	45
Table 5 - Start Solo Match	46
Table 6 - Submit compiler error	46
Table 7 - Submit wrong answer	47
Table 8 - Start Ranked Match	47
Table 9 - Ranked Match after Matching player.....	48
Table 10 - Wining the match.....	48
Table 11 - Leave the match	49
Table 12 - Search about user.....	49
Table 13 - Send Message.....	50
Table 14 - Send Friend request.....	50
Table 15 - Chat sending message.....	51

List of Abbreviations

ML	Machine Learning
JWT	JSON Web Token
DTO	Data Transfer Object
SCSS	Sassy Cascading Style Sheets
API	Application Programming Interface
JPA	Java Persistence API

Chapter 1: Introduction

1.1 Motivation (Abstract)

In the current landscape of online coding platforms, coding enthusiasts often face a fragmented experience. Many platforms focus primarily on coding challenges, but they lack a cohesive ecosystem that caters to diverse coding interests. Groofy Code emerges as a solution to this problem by integrating challenges, problem-solving activities, and competitive 1 vs 1 matches within a comprehensive framework. Existing platforms often lack an inclusive social aspect, which hinders collaboration and community building among coding enthusiasts.

Our motivation to develop Groofy Code stems from a passion for fostering a collaborative and engaging environment for coding enthusiasts. We aim to provide a unified platform where users can seamlessly transition between challenges, problem-solving activities, and competitive matches while building connections with like-minded individuals through clans and an interactive chat system. Groofy Code is designed to be more than just a coding platform; it is a community where users can grow, learn, and compete together.

1.2 Problem Definition

The current landscape of online coding platforms presents significant challenges due to the lack of a cohesive and all-encompassing solution for coding enthusiasts. Existing platforms often specialize in isolated aspects, such as coding challenges, competitive matches, or collaborative problem-solving, creating a fragmented experience for users. This fragmentation hinders the development of a unified community where individuals can seamlessly transition between various coding activities, showcase their skills, and engage in social interactions.

The absence of an integrated platform poses several challenges for coding enthusiasts:

1. **Fragmented Experience:** Users must navigate multiple platforms to fulfill their coding interests, leading to a disjointed experience.
2. **Limited Social Interaction:** The lack of an inclusive social aspect on existing platforms prevents users from building meaningful connections with other coding enthusiasts.
3. **Inconsistent Skill Development:** With isolated functionalities, users may struggle to find a balanced approach to developing their coding skills across different activities.
4. **Lack of Community Engagement:** The absence of a unified platform makes it difficult to foster a sense of community and collaboration among users.

These challenges highlight the need for a comprehensive and dynamic environment that caters to diverse preferences and provides a seamless user experience.

1.3 Our Solution

Groofy Code redefines the online coding experience by offering a feature-rich platform that addresses the shortcomings of existing platforms. Our solution includes the following key features:

1. **Solver Profile:**

- **Personalized Journey:** Each user has a detailed profile that showcases their achievements, progress, and social connections. This personalized journey helps users track their growth and stay motivated.
- 2. **Problem Solving:**
 - **Diverse Challenges:** Groofy Code offers a wide range of coding challenges that cater to different skill levels and interests, ensuring that users are constantly engaged and challenged.
- 3. **1 vs 1 Challenges:**
 - **Competitive Element:** Users can participate in 1 vs 1 matches, which add a competitive element to the platform. These matches are automatically timed and have a global rating impact, encouraging users to improve their skills and compete at higher levels.
- 4. **Clan System:**
 - **Collaboration and Community Building:** The clan system allows users to form or join clans, fostering collaboration and community building. Clans can participate in group challenges and compete against each other, enhancing the social aspect of the platform.
- 5. **Friends:**
 - **Enhanced Communication:** Users can add friends, making it easier to connect and communicate with other coding enthusiasts.
- 6. **Customizable Unrated Challenges:**
 - **Flexibility:** Users can create and customize their own unrated challenges, providing flexibility in the types of problems they wish to tackle and practice.
- 7. **Global Rating System:**
 - **Performance Evaluation:** The global rating system evaluates both individual and clan performance, encouraging healthy competition and providing users with a clear sense of their standing within the community.
- 8. **Velocity Matches:**
 - **Quick Challenges:** Groofy Code introduces quick 15-minute velocity matches designed to challenge users to solve problems rapidly, testing their coding speed and accuracy under tight time constraints.

Groofy Code's technology stack, featuring React, Java Spring Boot, and machine learning, underscores its commitment to providing a seamless, engaging, and technologically advanced coding environment. By integrating these cutting-edge technologies, Groofy Code ensures that users have a robust and enjoyable experience, making it the go-to platform for coding enthusiasts seeking a comprehensive and dynamic environment.

1.4 Gantt chart of project time plan

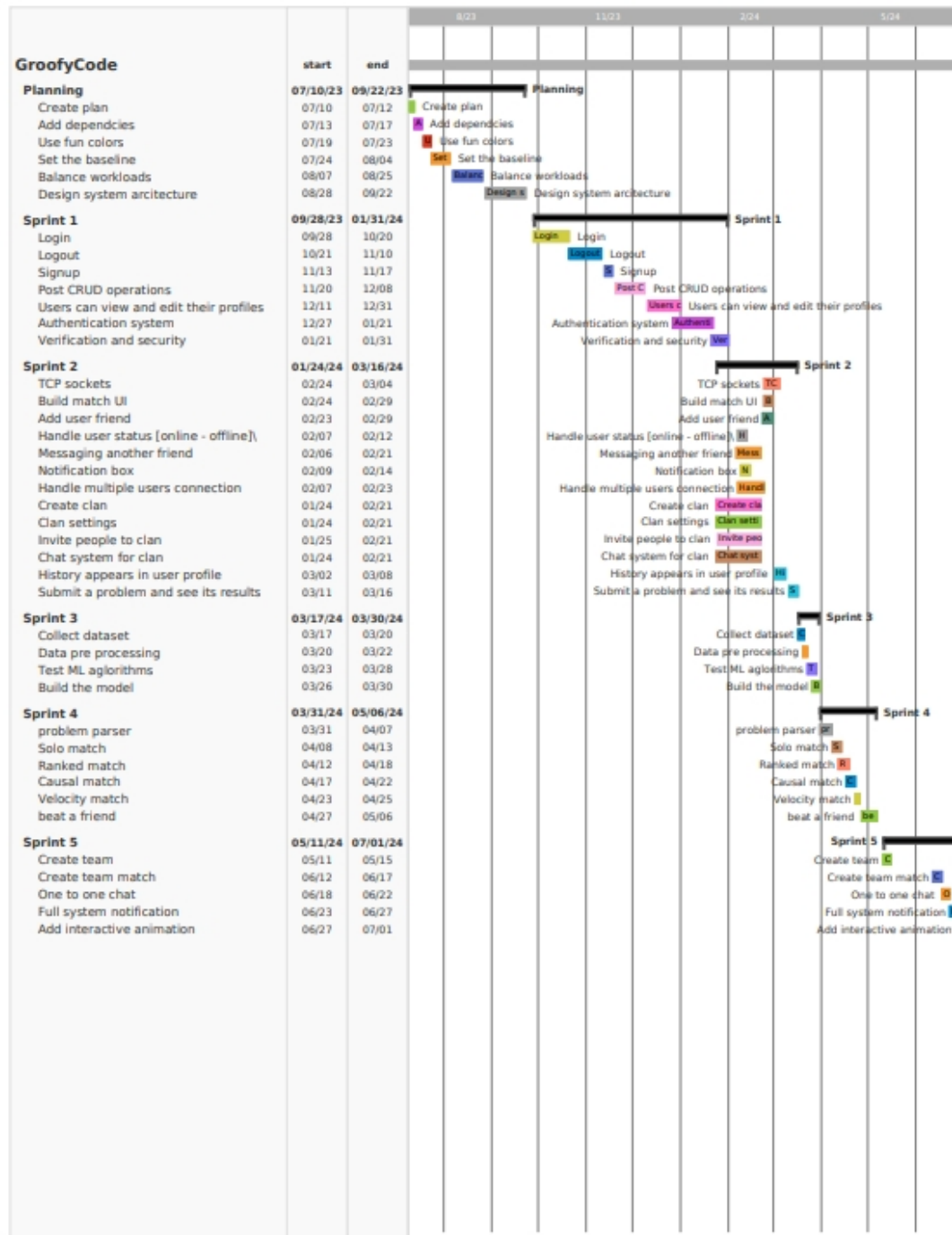


Figure 1 - Gantt Chart

1.5 Development Methodology: Agile

In the software development life cycle, we use the agile methodology, which promotes iterative development, collaboration, and flexibility to adapt to changing requirements. Agile is an appropriate method because it aims to achieve these goals by enabling continuous feedback, early and frequent delivery of functional components, and a focus on customer satisfaction. The entire software development process is divided into iterative cycles called sprints, each resulting in a potentially shippable product increment. The phases in our agile methodology are:

Requirements Gathering

In this phase, we collaborate with stakeholders to gather all the functional and non-functional requirements for the project. The requirements include the main functionality of the platform, such as user profiles, problem-solving interfaces, 1 vs 1 challenges, clan system, and chat system.

Sprint Planning

In each sprint planning session, we prioritize the requirements and create a sprint backlog, which includes the tasks and features to be developed in the upcoming sprint. This planning ensures that the team has a clear understanding of the goals and deliverables for each sprint.

Development

The development phase involves the actual coding and implementation of the features planned for the sprint. We follow best practices, coding standards, and agile principles to ensure high-quality and maintainable code. Development is done in small, manageable increments, allowing for continuous integration and frequent delivery of working software.

Testing

During the testing phase, we conduct thorough testing of the features developed in the sprint to ensure they meet the requirements and function as intended. Automated testing tools and quality assurance techniques are used to verify the accuracy and reliability of the code. Testing is an integral part of each sprint, ensuring that any issues or bugs are identified and addressed promptly.

To ensure the reliability of our API endpoints, we will use **Swagger** for API documentation and testing. Swagger allows us to design, build, document, and test our APIs with ease. This tool will help us ensure that our APIs function correctly and meet the specified requirements by providing a user-friendly interface for testing each endpoint. It also facilitates automated testing and validation of API responses, contributing to a robust and reliable system.

Review and Retrospective

At the end of each sprint, we hold a sprint review meeting to demonstrate the completed features to stakeholders and gather feedback. This feedback is crucial for making any necessary adjustments and planning future sprints. We also conduct a sprint retrospective to

reflect on the sprint process, identify areas for improvement, and implement changes to enhance the team's efficiency and effectiveness.

Maintenance and Continuous Improvement

Once the platform is deployed, it will require ongoing maintenance and updates to ensure it continues to function correctly and meet user needs. We develop a maintenance schedule and plan to address any issues or bugs that arise, as well as implement new features and enhancements based on user feedback and evolving requirements.

By following the agile methodology, we ensure that Groofy Code is developed in a flexible, efficient, and customer-focused manner, delivering a high-quality platform that meets the needs of coding enthusiasts.

1.6 Tools and Technologies Used in the Project

1.6.1 Back-end

- **Java**
- **Java Spring Boot**
- **MySQL**
- **Spring Security**
- **Spring JPA**
- **Spring WebSocket**
- **Spring REST APIs**
- **JWT (JSON Web Token)**
- **Swagger** (for API documentation and testing)
- **Integration with Codeforces** (Third Party)
- **Integration with Firebase** (Third Party)
- **Postman** (for API testing)

1.6.2 Front-end

- **React.js**
- **TypeScript**
- **Redux**
- **SCSS**
- **MathJax**
- **PrimeReact**
- **WebSocket**
- **CodeMirror**
- **Formik**

1.6.3 Machine Learning

- Python programming language: NumPy, Pandas, Sklearn, Matplotlib for visualization.

- Multiple datasets: Combined datasets for training and evaluation, processed using Pandas.
- RandomForestRegressor: For predicting the expected rating of coders.
- Joblib: For model serialization and deserialization.
- StandardScaler: For feature normalization.
- Flask: For building and deploying APIs to serve model predictions.
- Excel files: Used as data sources for training and evaluation.
- Visualization: Plotting actual vs. predicted results using Matplotlib.
- Kaggle environment: For data science and machine learning experiments and competitions.

1.7 Report Organization (summary of the rest of the report)

The rest of the document discusses the different phases that describe and illustrate the characteristics of the Groofy Code project:

Chapter Two: Related Work

- Compares Groofy Code to platforms like Codeforces, AtCoder, and V-Judge.
- Highlights unique features of Groofy Code, such as 1 vs 1 matches, team matches, personalized recommendations, and a dynamic environment with badges and trophies.

Chapter Three: System Analysis

- Outlines functional requirements (user registration, authentication, problem-solving, clan system, chat system, rating system, and matchmaking using machine learning).
- Describes non-functional requirements (scalability, performance, reliability, usability, portability, security).
- Includes a use case diagram showing stakeholders and scenarios.

Chapter Four: System Design

- Presents diagrams describing class and component relationships and data storage/retrieval:

Component Diagram, Class Diagram, Sequence Diagrams, Entity-Relationship Diagram (ERD), System GUI Design

Chapter Five: Implementation and Testing

- Reports on machine learning models for recommendations and user rating predictions.
- Covers backend development challenges and solutions.
- Provides testing scenarios and results to ensure system requirements are met.

Summary

The Groofy Code project aims to create a comprehensive coding platform with challenges, problem-solving activities, and competitive matches. It uses machine learning for personalized recommendations and features a dynamic environment with badges and trophies. The project ensures a scalable, reliable, and user-friendly platform through robust design and thorough testing.

Chapter 2: Related work

2.1 Related Work

In the landscape of online coding platforms, several existing platforms provide coding challenges and competitive programming, much like Groofy Code. This section highlights the similarities and differences between Groofy Code and some of the most notable platforms in this space.

2.1.1 Similar Features

- *Codeforces*

Codeforces is a popular platform known for its regular programming contests and a vast collection of coding problems. It offers features such as:

Solving Problems: Users can solve problems independently to improve their coding skills.

User Interaction: Users can search for other users, send messages, and view profiles.

Group Competitions: Codeforces supports the creation of groups and allows users to compete as teams in contests.

- *AtCoder*

AtCoder is another competitive programming platform that provides regular contests and a large repository of problems. Key features include:

- **Contests:** AtCoder organizes various contests, from beginner to advanced levels, to challenge participants.
- **Problem Practice:** Users can practice problems from past contests.
- **Rating System:** Participants are rated based on their performance in contests.

- *V-Judge (Virtual Judge)*

V-Judge aggregates problems and contests from various online judges, allowing users to practice and compete across different platforms. It offers:

- **Problem Aggregation:** Users can solve problems sourced from multiple online judges.

- **User and Group Interaction:** Similar to other platforms, V-Judge allows users to search for and interact with other users and join groups.

2.1.2 Distinguishing Features of Groofy Code

While the aforementioned platforms share similarities with Groofy Code, there are several key differences that set Groofy Code apart:

Competitive Programming

- **1 vs 1 Matches:** Unlike traditional platforms, Groofy Code emphasizes 1 vs 1 competitive matches, offering a more personalized and direct competition experience.
- **Team vs Team Matches:** In addition to individual matches, Groofy Code supports team-based competitions, enhancing collaborative problem-solving skills.
- **Velocity Matches:** Groofy Code introduces quick 15-minute matches designed to challenge users to solve problems rapidly, testing their coding speed and accuracy under tight time constraints.

Personalized Problem Recommendations

- **Machine Learning:** Groofy Code employs a machine learning model to recommend problems tailored to the user's skill level and learning pace. This ensures that users are consistently challenged but not overwhelmed, providing a customized learning experience.

Dynamic Environment

- **Badges and Trophies:** To foster a sense of achievement and recognition, Groofy Code incorporates badges, rankings, and trophies, motivating users to engage more actively.

Comprehensive Practice

- **Solo Matches:** In addition to competitive matches, Groofy Code offers solo practice sessions, allowing users to improve their skills in a non-competitive environment.

Collaborative Aspect

- **Clan System:** Groofy Code enhances the social aspect by allowing users to form or join clans. This fosters a sense of community and collaboration, encouraging teamwork and shared learning.

Summary

While existing platforms like Codeforces, AtCoder, and V-Judge provide robust features for coding challenges and competitive programming, Groofy Code distinguishes itself with its unique focus on personalized competitive matches, a dynamic and engaging environment, and enhanced collaborative features. These elements create a comprehensive and enriching experience for coding enthusiasts, setting Groofy Code apart from its counterparts.

References

- [Codeforces](#)
- [AtCoder](#)
- [V-Judge](#)

Chapter 3: System Analysis

3.1 Project specification

3.1.1 Functional Requirements

- 1. User Registration and Authentication**
 - Users should be able to register with a unique username and password.
 - Users must be able to log in securely using their credentials.
 - Password recovery/reset functionality should be available.
- 2. Solver Profile Management**
 - Users should be able to create and edit their solver profiles.
 - Profile information should include a display name and optional bio.
 - Users can link their solver profiles to their social media accounts.
- 3. Global Score and Badges**
 - The platform should calculate and display a global score for each solver.
 - Badges should be awarded based on achievements and milestones.
- 4. Problem Solving**
 - Solvers can access a diverse range of coding challenges.
 - Each challenge should specify the allowed programming languages.
 - The platform should support the submission.
 - A submission history and a code editor with syntax highlighting should be available.
 - Use a machine learning model to match users with appropriate problems.
- 5. Clan System**
 - Solvers can create, join, and manage clans.
 - Each clan should have a logo, achievements, and bio.
 - Clans should have a ranking based on collective performance.
- 6. Friends System**
 - Solvers can search for friends and send friend requests.
 - Users can manage their list of friends.
- 7. Chat System**
 - Clan members can communicate through an interactive chat system.
 - The chat system should support messaging, and emojis.
- 8. Rating System**
 - The platform should maintain a global rating system for individual solvers and clans.
 - Ratings should be updated based on performance in rated challenges and challenge difficulty.
- 9. Matchmaking System Using Machine Learning**

- Utilize historical user data, including user rate, types of problems solved, and problems rate, to build a recommendation model that ensures fairness in matches between users.
- Implement collaborative filtering techniques to suggest potential opponents with similar skill levels and problem-solving strategies.
- Recommend problems to the user for practicing by predicting the rate of the problem to be solved and its tags based on the historical data of the user.

3.1.2 Non-Functional Requirements

1. **Scalability**

- The system should scale horizontally to accommodate an increasing number of users.

2. **Performance**

- Response time for key actions, such as code submission and challenge initiation, should be within 2 seconds.

3. **Reliability**

- The platform should reliably track and update user scores and ratings in real-time.

4. **Usability**

- The user interface should be intuitive, with clear navigation and a responsive design for various devices and screen sizes. Aim for an average user satisfaction rating of at least 4 out of 5 in user feedback surveys.

5. **Portability**

- The platform should be accessible and fully functional across major web browsers, including Chrome, Firefox, Safari, and Edge.

6. **Security**

- Utilize secure coding practices to protect user data and prevent unauthorized access. Implement encryption for data transmission and storage, including user passwords.

3.2 Use case Diagram

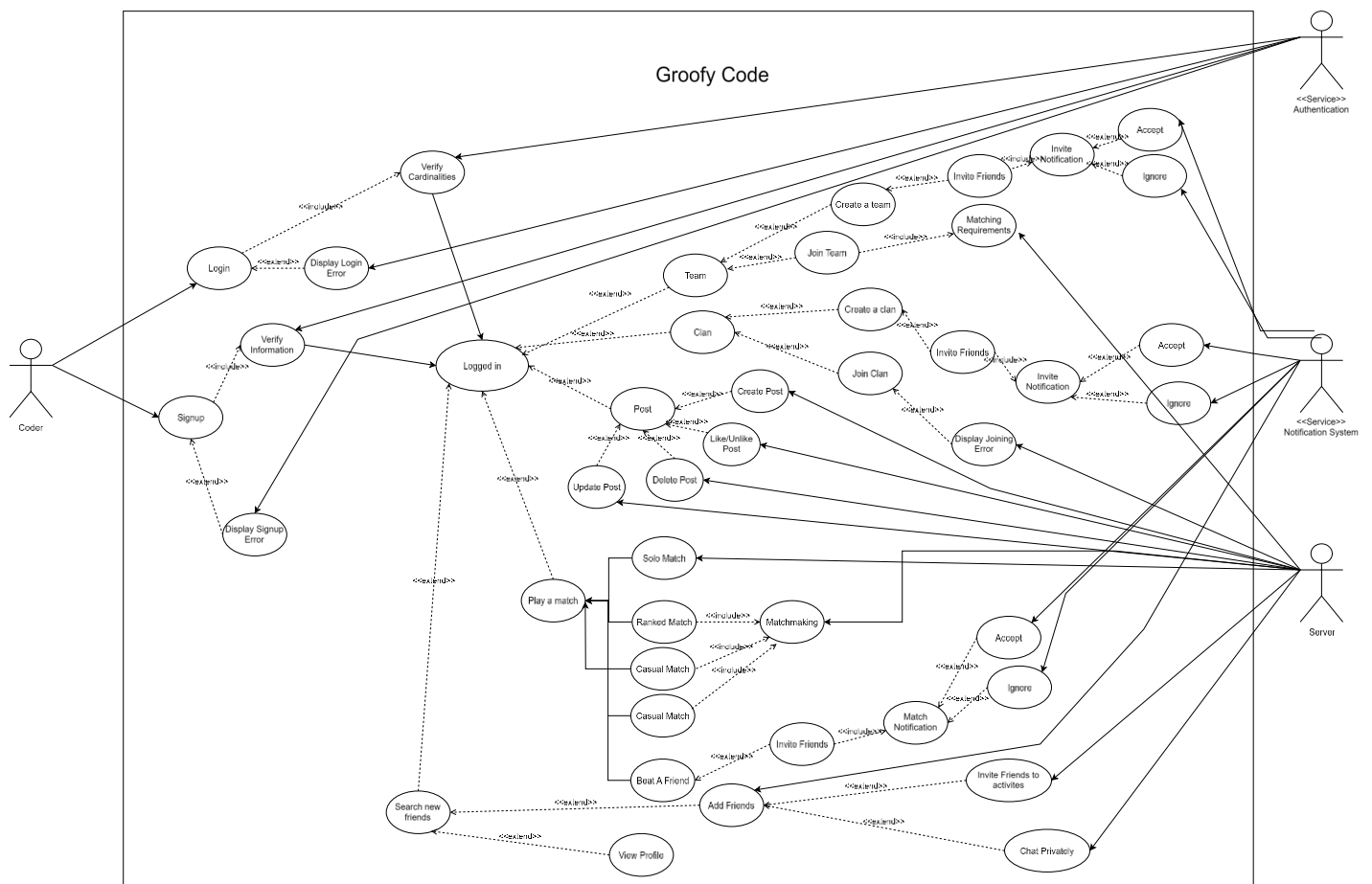


Figure 2- Use case Diagram

Chapter 4: System Design

4.1 System Component Diagram

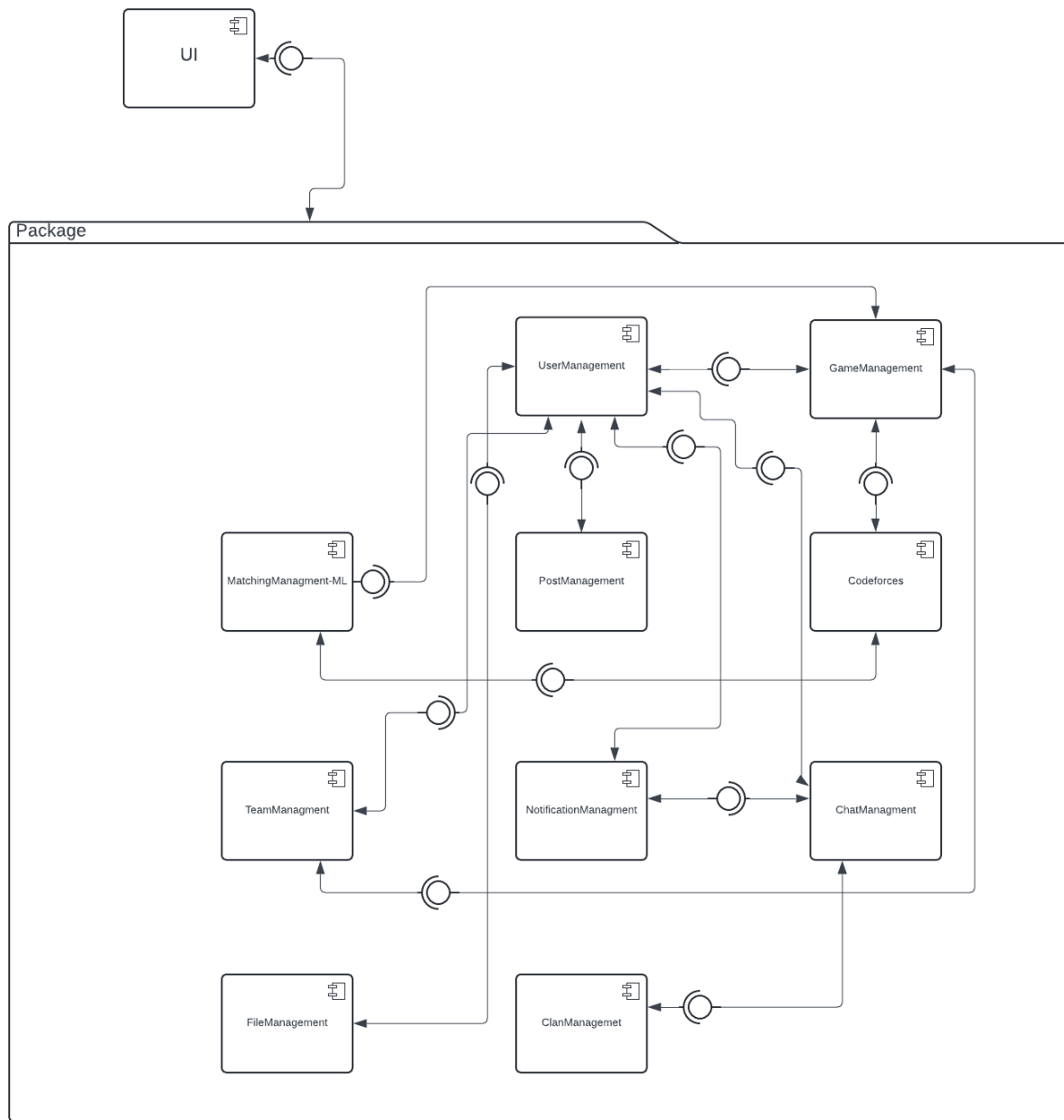


Figure 3- System Component Diagram

4.2 System Class Diagrams

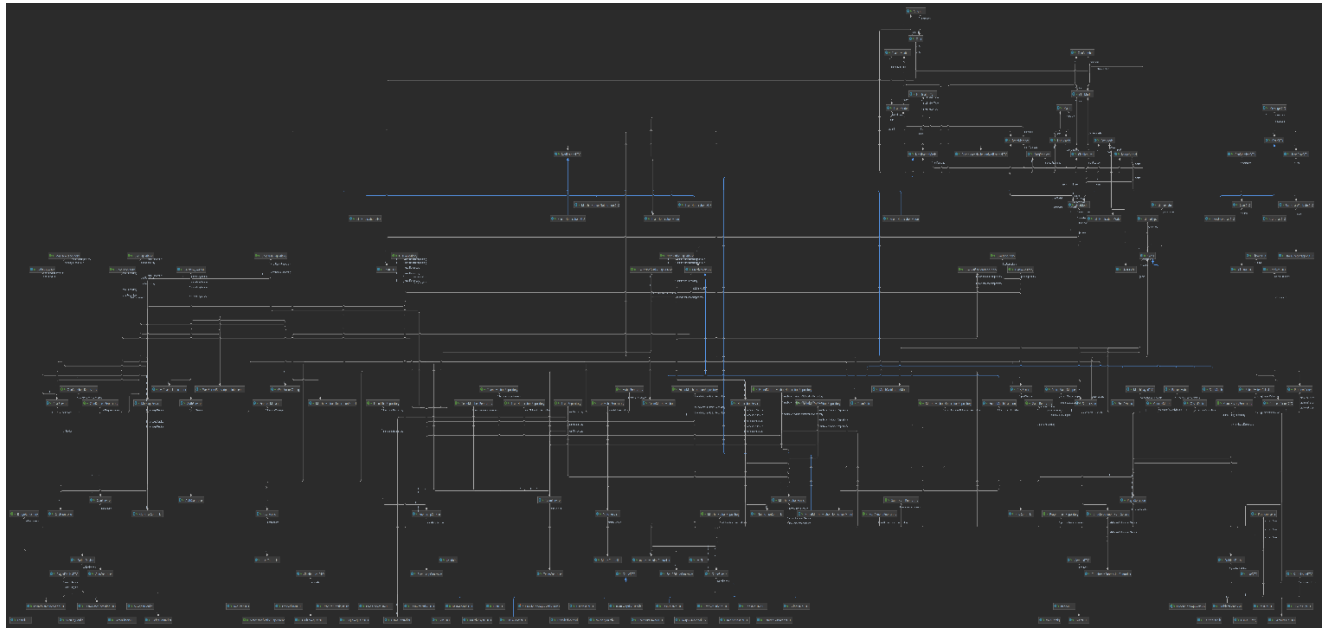


Figure 4 - System Class Diagrams

4.3 Sequence Diagrams

4.3.1 Ranked Match

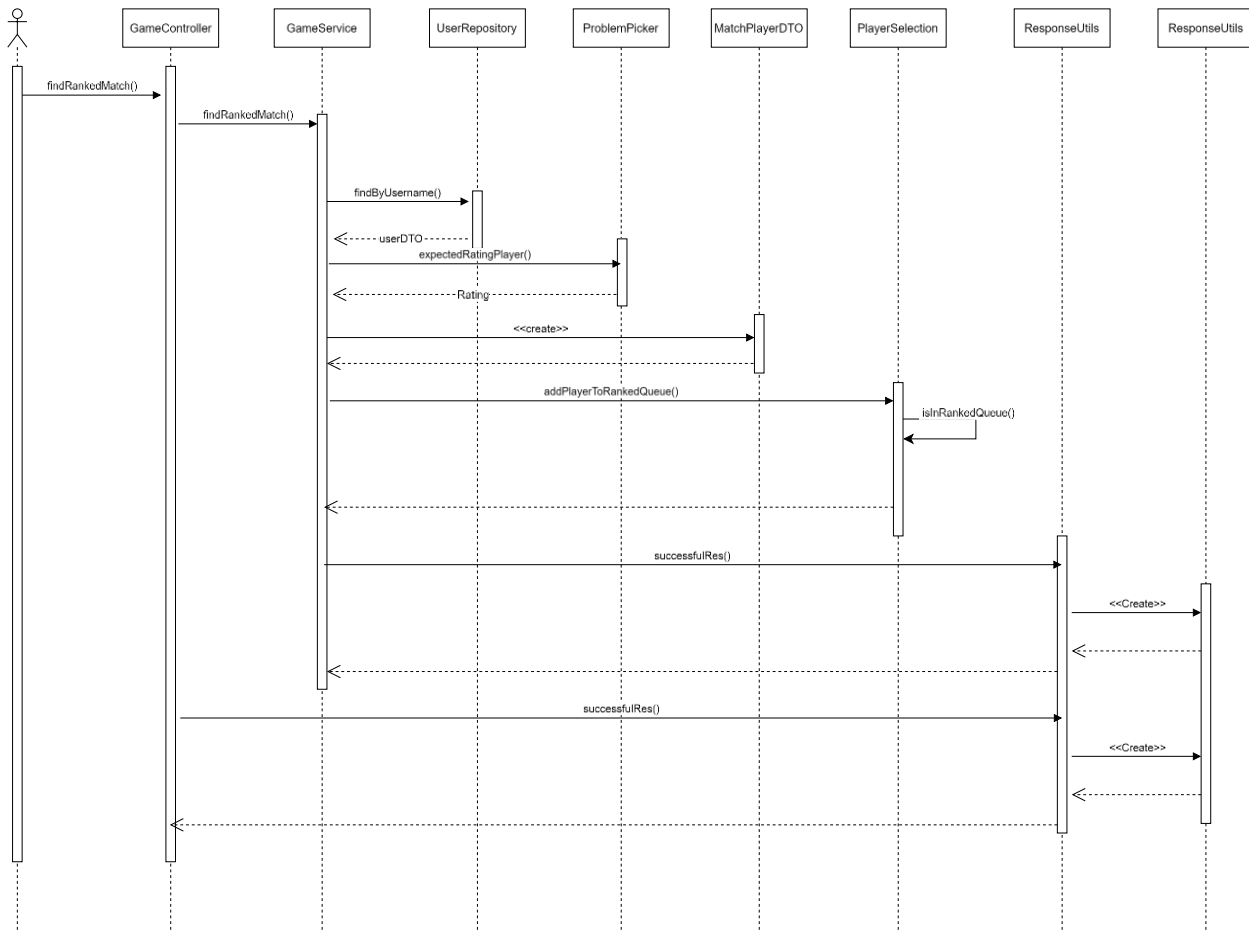


Figure 5 - Ranked Match Sequence Diagram

4.3.2 Accept team invitation.

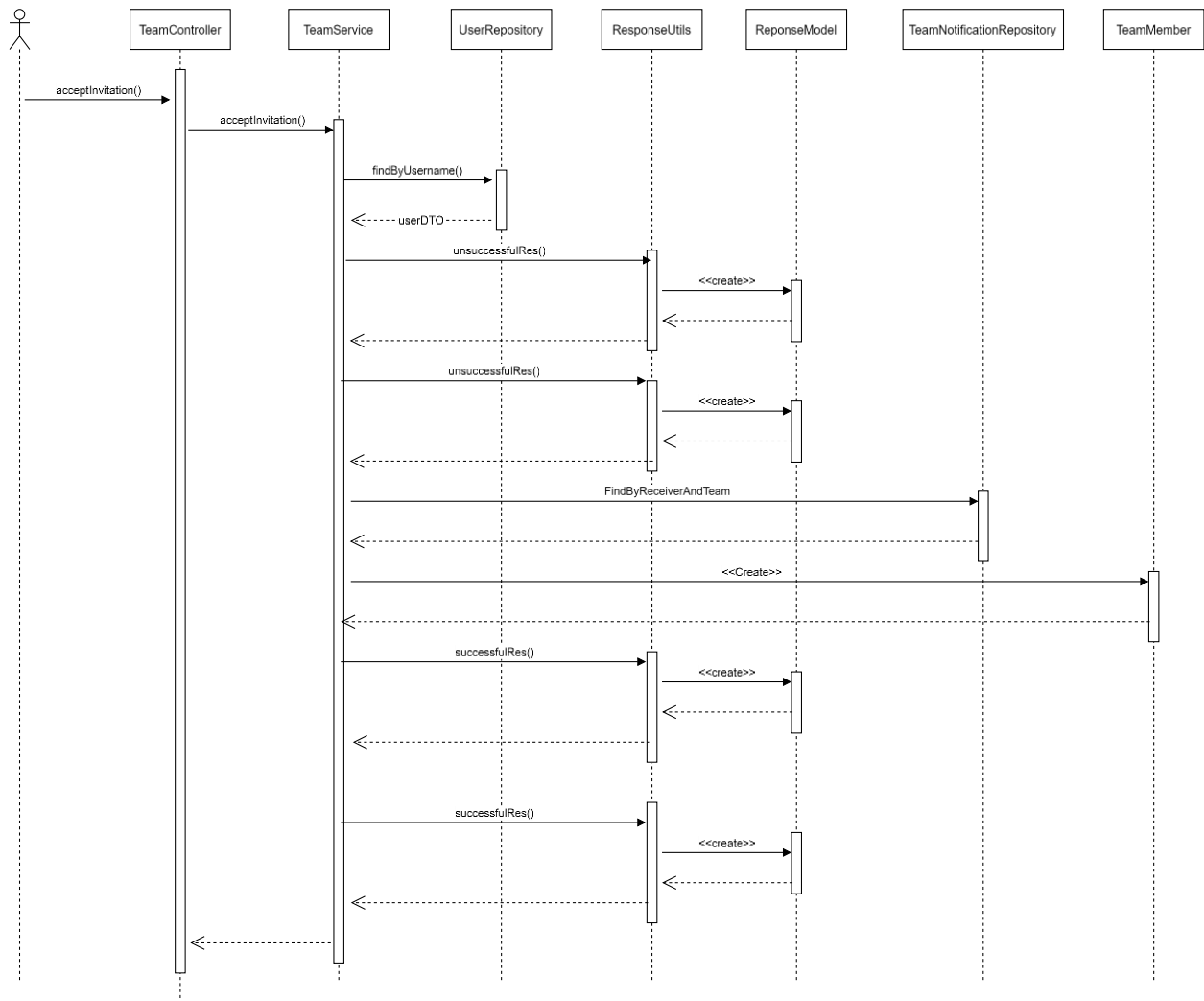


Figure 6 - Accept team invitation Sequence Diagram

4.3.3 Send match invitation to friend.

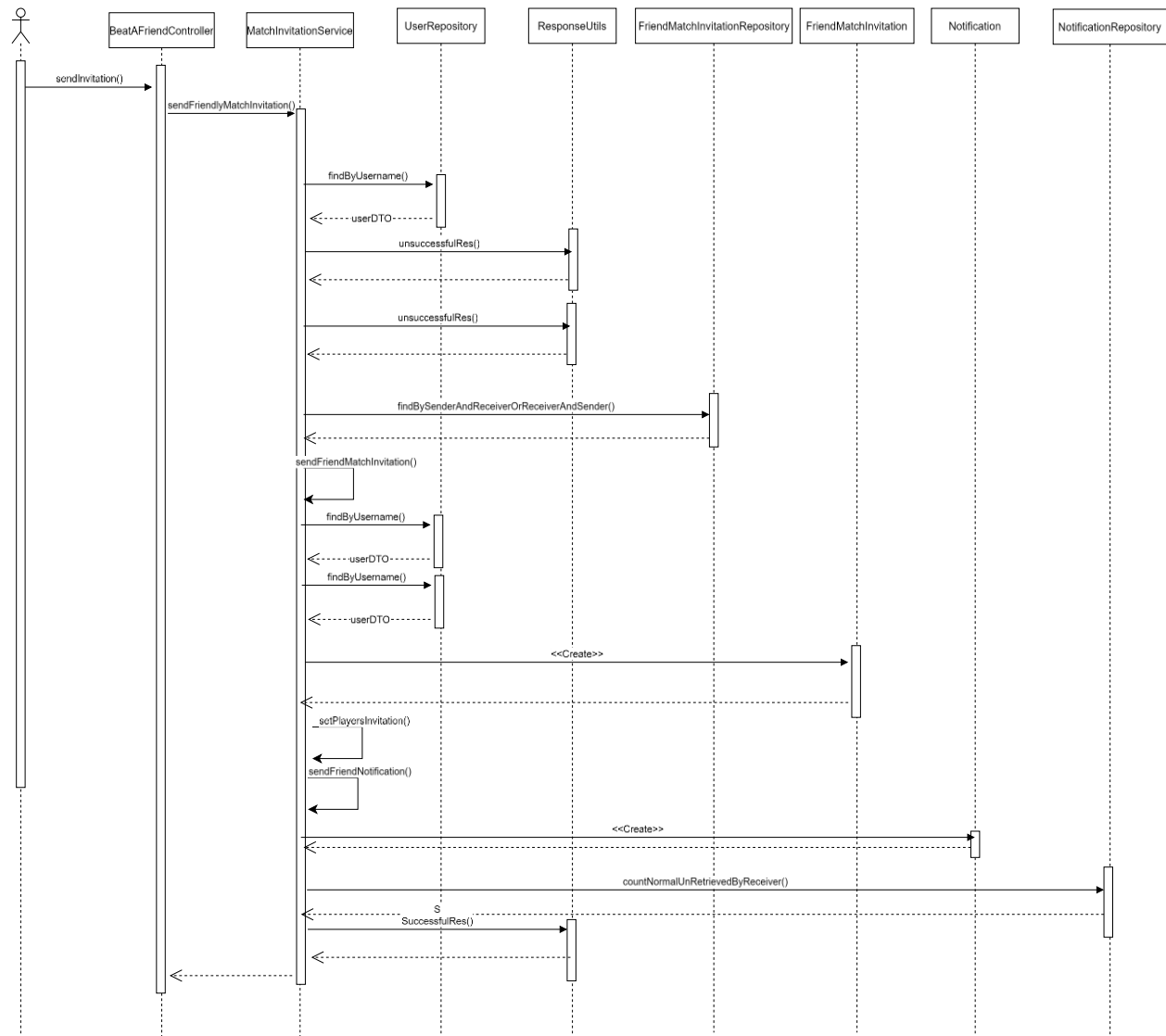


Figure 7 - Send match invitation to friend Sequence Diagram

4.4 Project ERD

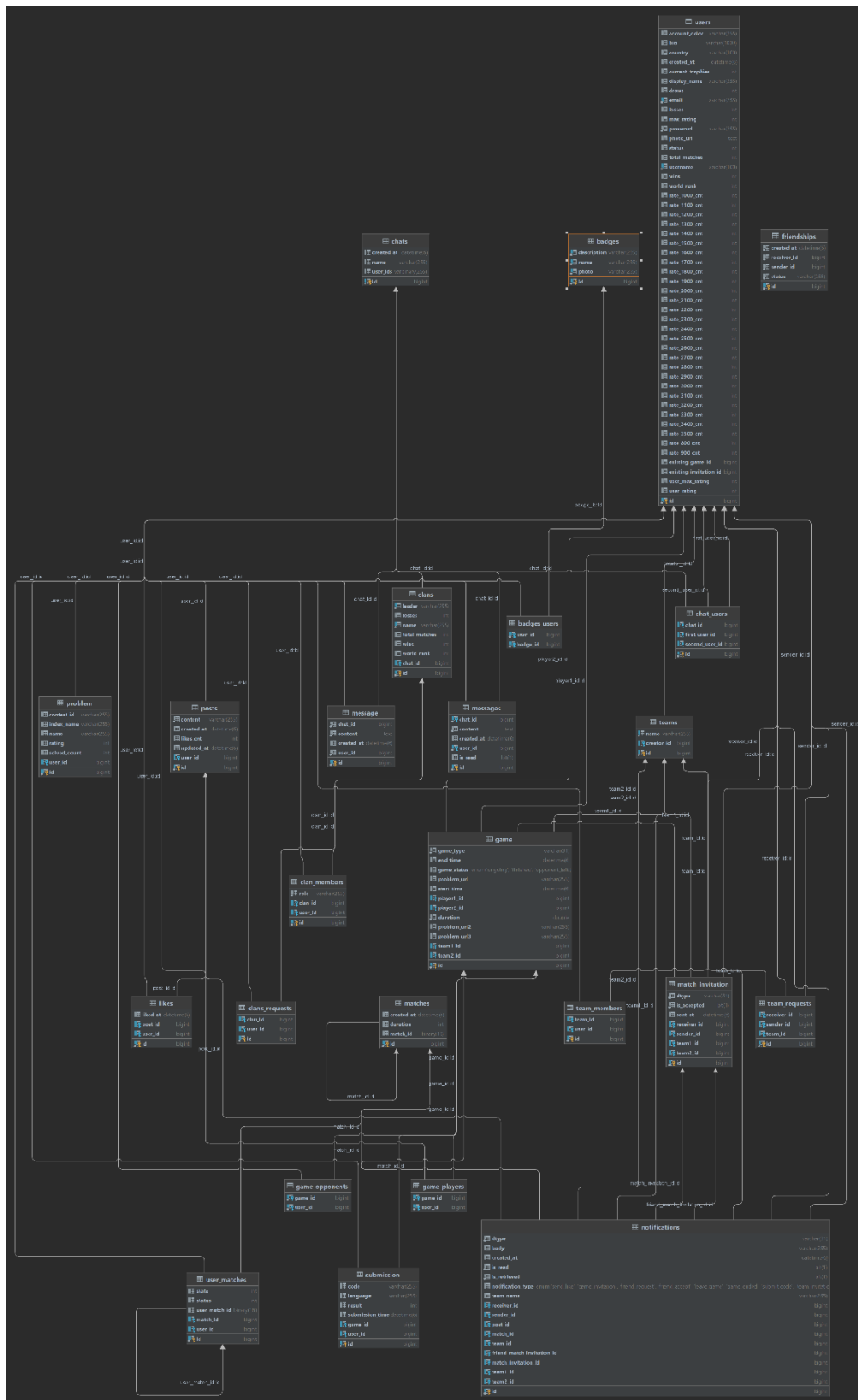


Figure 8 - ERD Diagram

4.5 System GUI Design

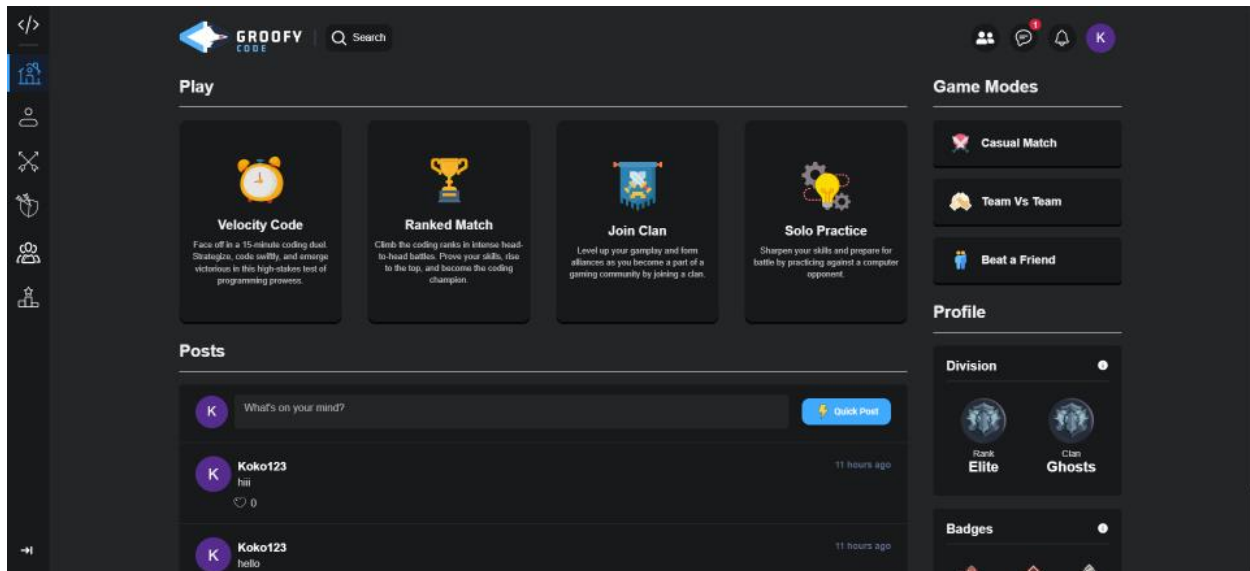


Figure 9 - Home Page

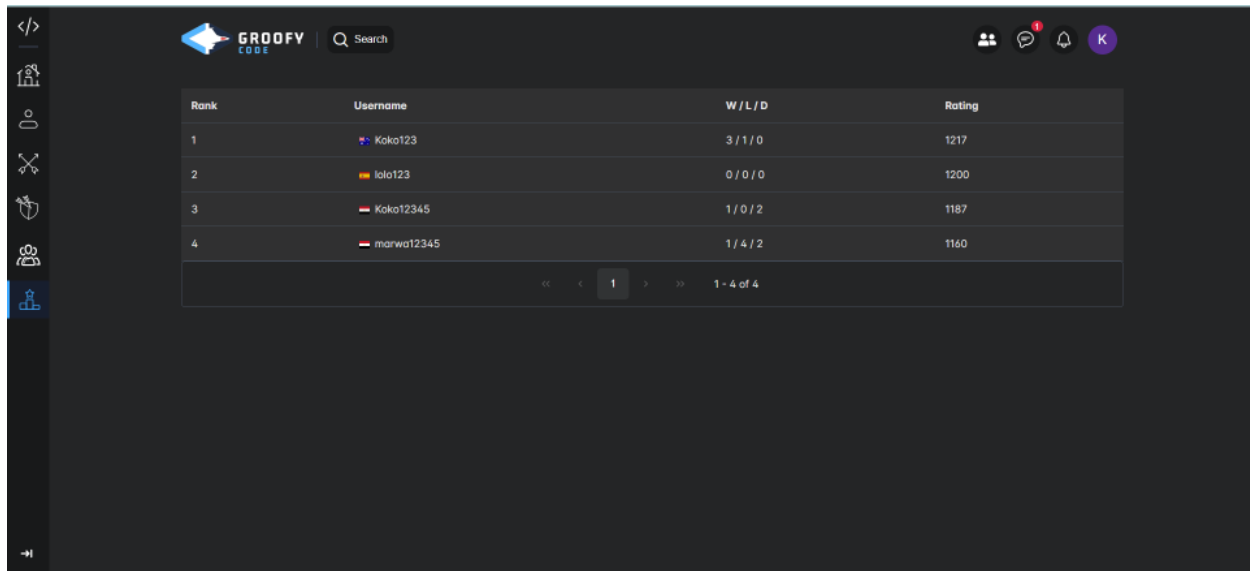


Figure 10 - Leader Board

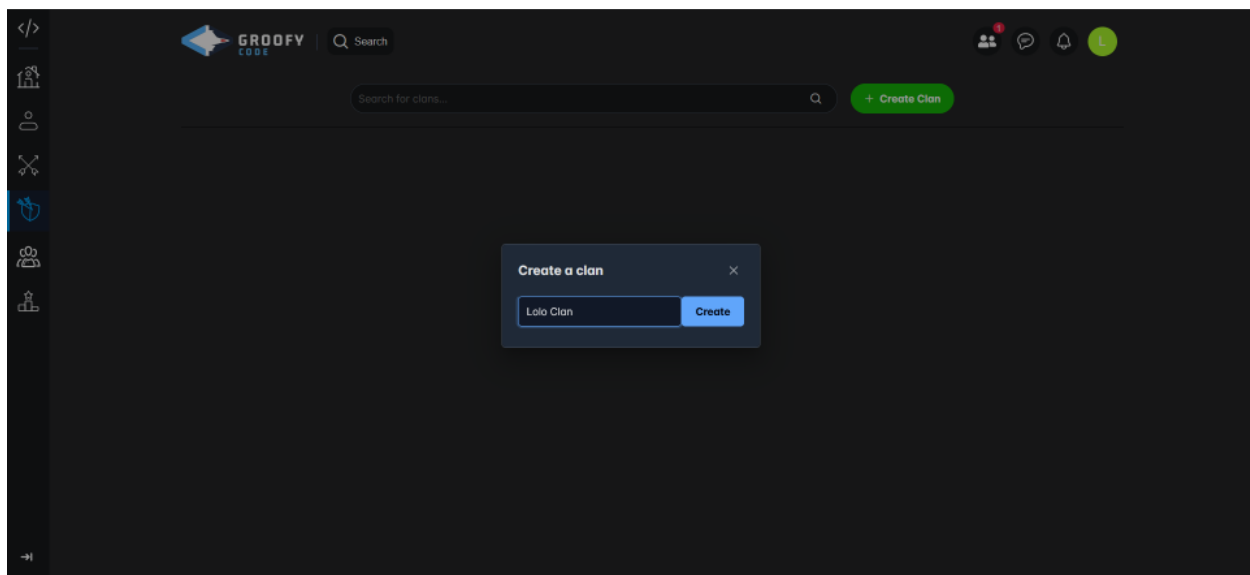


Figure 11 - Create Clan

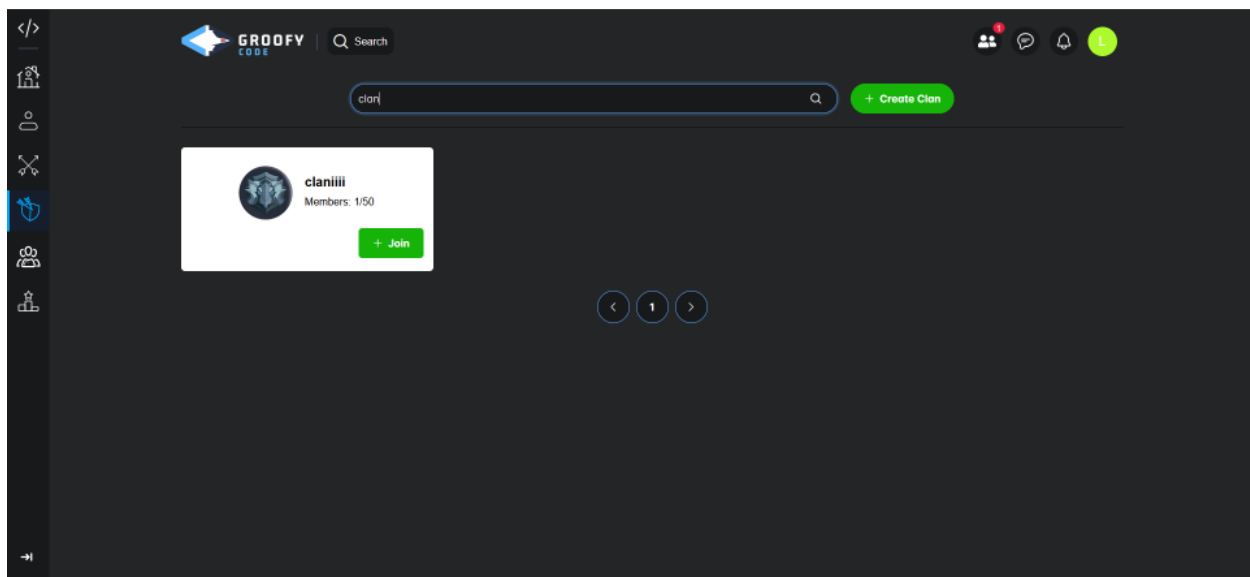


Figure 12 - Search Clan

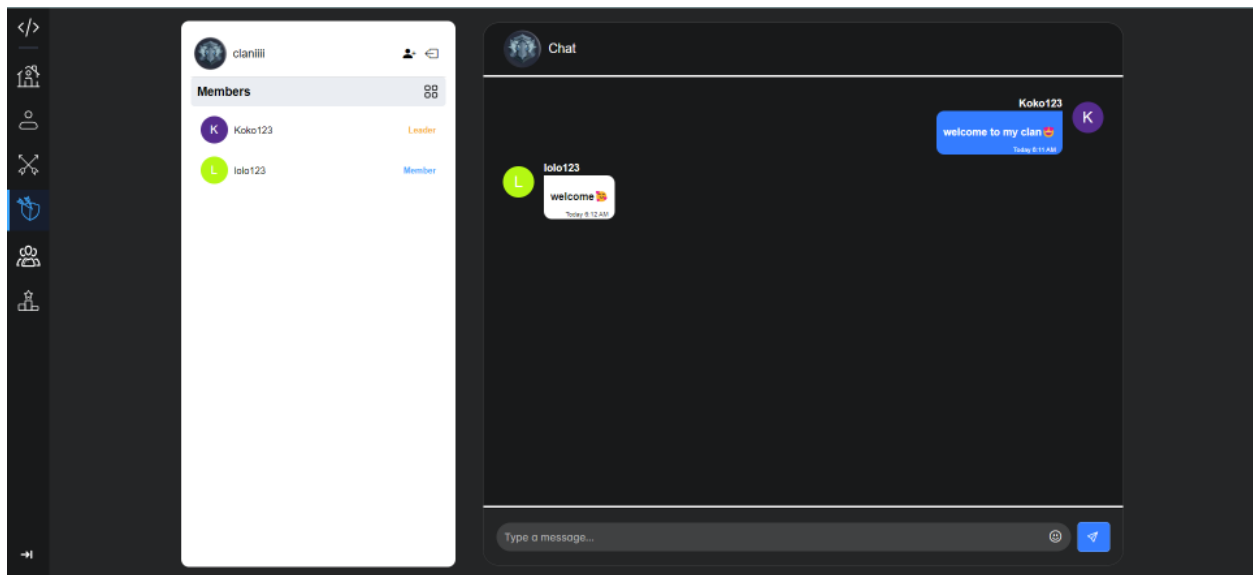


Figure 13 - Clan Chat

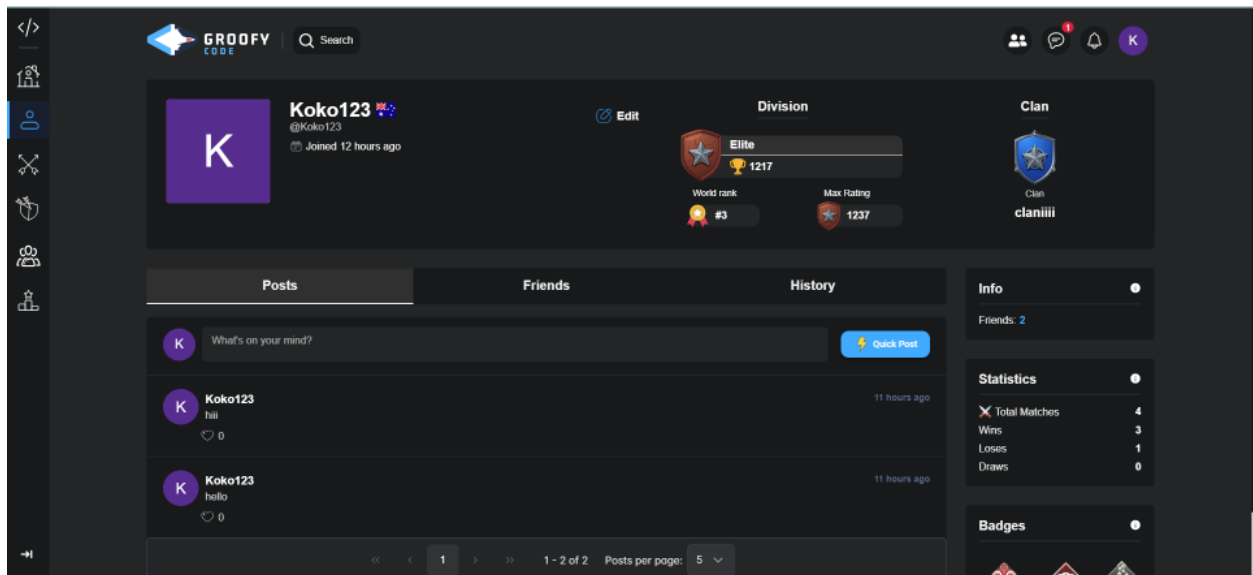


Figure 14 - Profile

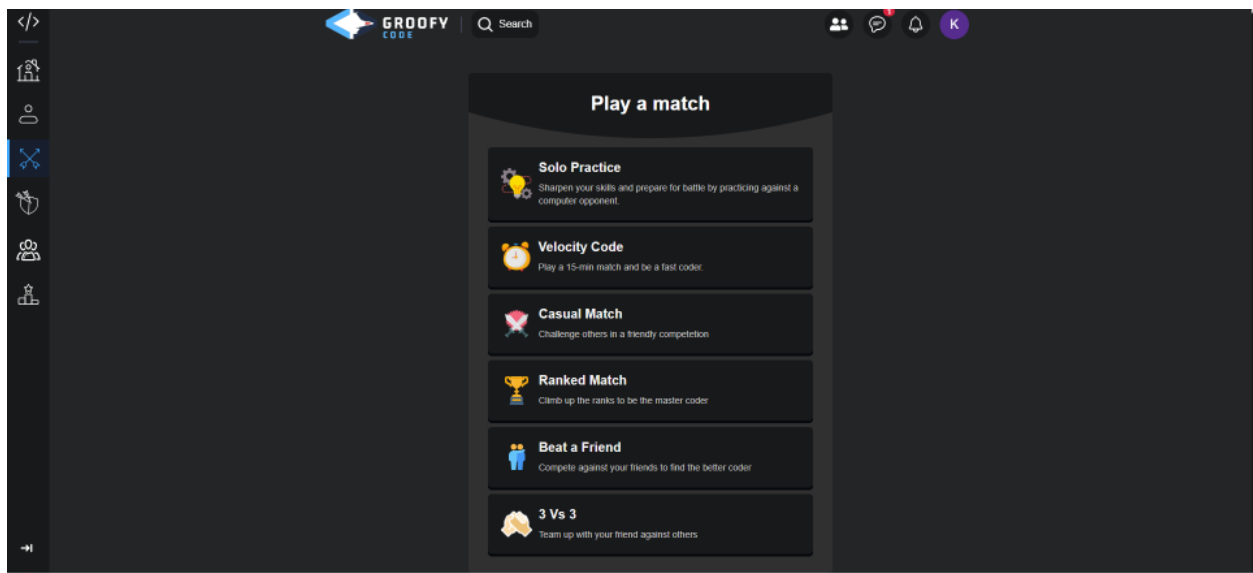


Figure 15 - Play Match

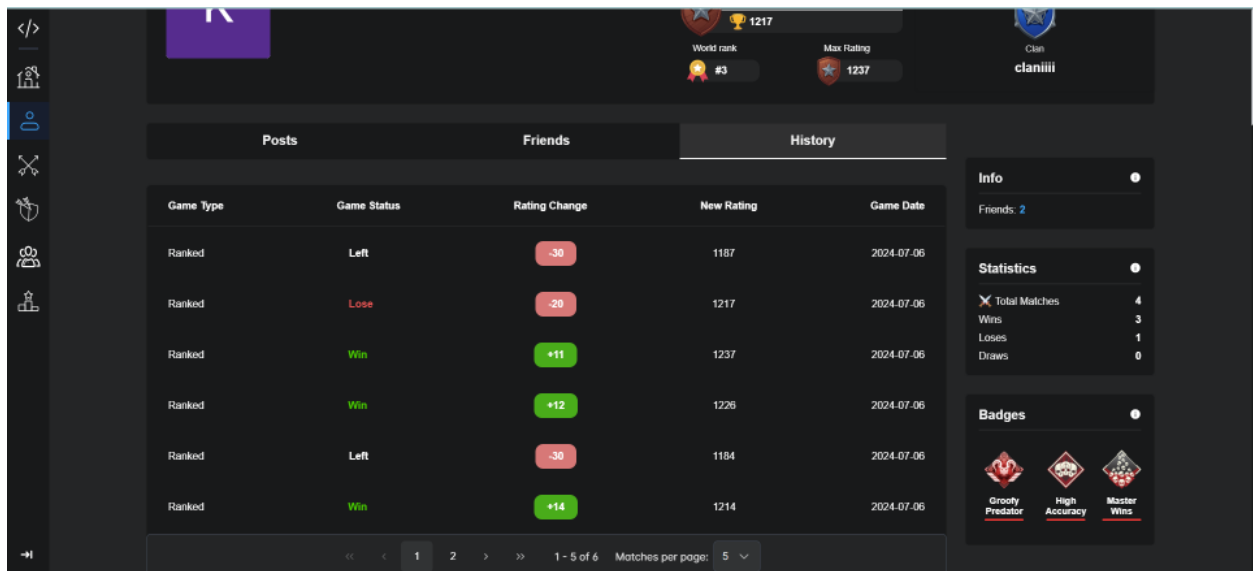


Figure 16 - Game History

Leave

58:14 left

hazemadel2 H

Solo Game | 60 Min

Move Brackets

time limit per test 1 second
memory limit per test 256 megabytes
input standard input
output standard output

You are given a bracket sequence s of length n , where n is even (divisible by two). The string s consists of $\frac{n}{2}$ opening brackets '(' and $\frac{n}{2}$ closing brackets ')'.
In one move, you can choose exactly one bracket and move it to the beginning of the string or to the end of the string (i.e. you choose some index i , remove the i -th character of s and insert it before or after all remaining characters of s).
Your task is to find the minimum number of moves required to obtain regular bracket sequence from s . It can be proved that the answer always exists under the given constraints.
Recall what the regular bracket sequence is:

- '()' is regular bracket sequence;
- If s is regular bracket sequence then $(s + s)$ is regular bracket sequence;
- If s and t are regular bracket sequences then $s + t$ is regular bracket sequence.

For example, "()", "(())", "()" and "()" are regular bracket sequences, but ")", "((" and ")))" are not. You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \leq t \leq 2000$) — the number of test cases. Then t test cases follow.
The first line of the test case contains one integer n ($2 \leq n \leq 50$) — the length of s . It is guaranteed that

Language: C++

```

1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "Hello world!";
7 }
8

```

Submit

Submission Time	Verdict	Language
No submissions found		

Figure 17 - Solo Match

Leave

59:54 left

hazemadel2 H

Ranked Game | 60 Min

hazemadel2 H

VS

hazemadel H

Filling Shapes

time limit per test 1 second
memory limit per test 256 megabytes
input standard input
output standard output

You have a given integer n . Find the number of ways to fill all $3 \times n$ tiles with the shape described in the picture below. Upon filling, no empty spaces are allowed. Shapes cannot overlap.

This picture describes when $n = 4$. The left one is the shape and the right one is $3 \times n$ tiles.

Input

Language: C++

```

1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "Hello world!";
7 }
8

```

Submit

Submission Time	Verdict	Language
No submissions found		

Figure 18 - Ranked Match

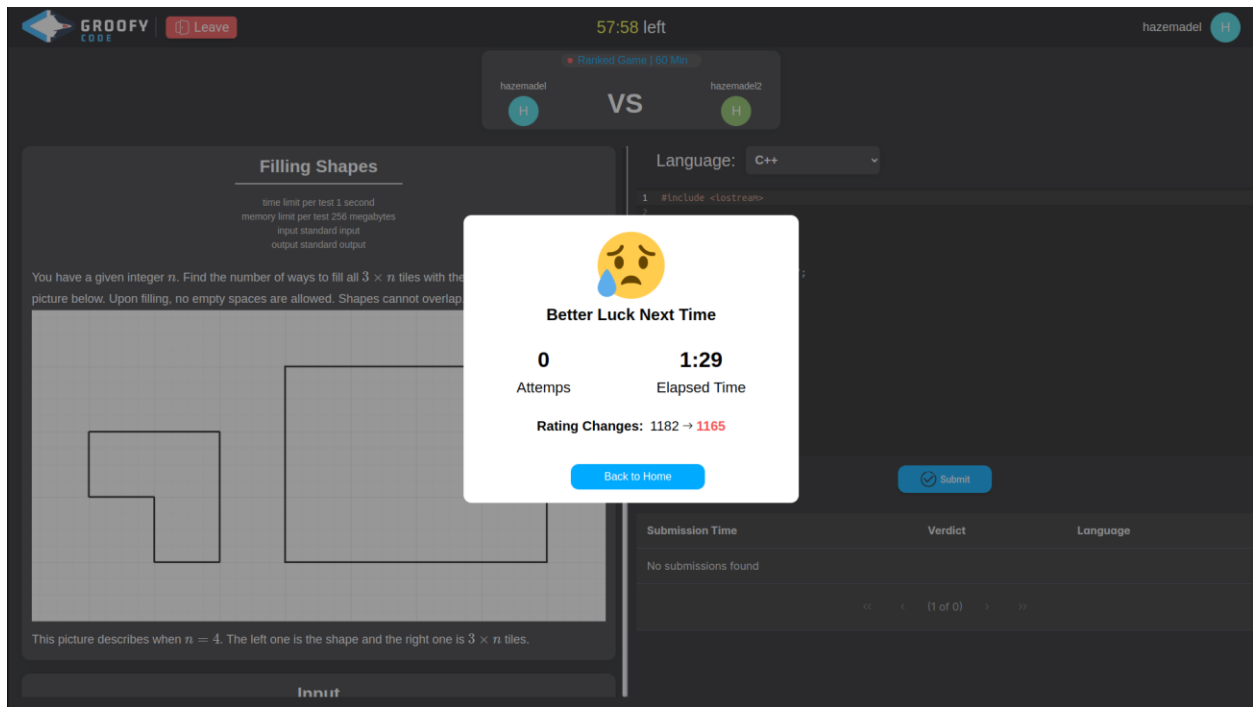


Figure 19 - Losing Ranked Match

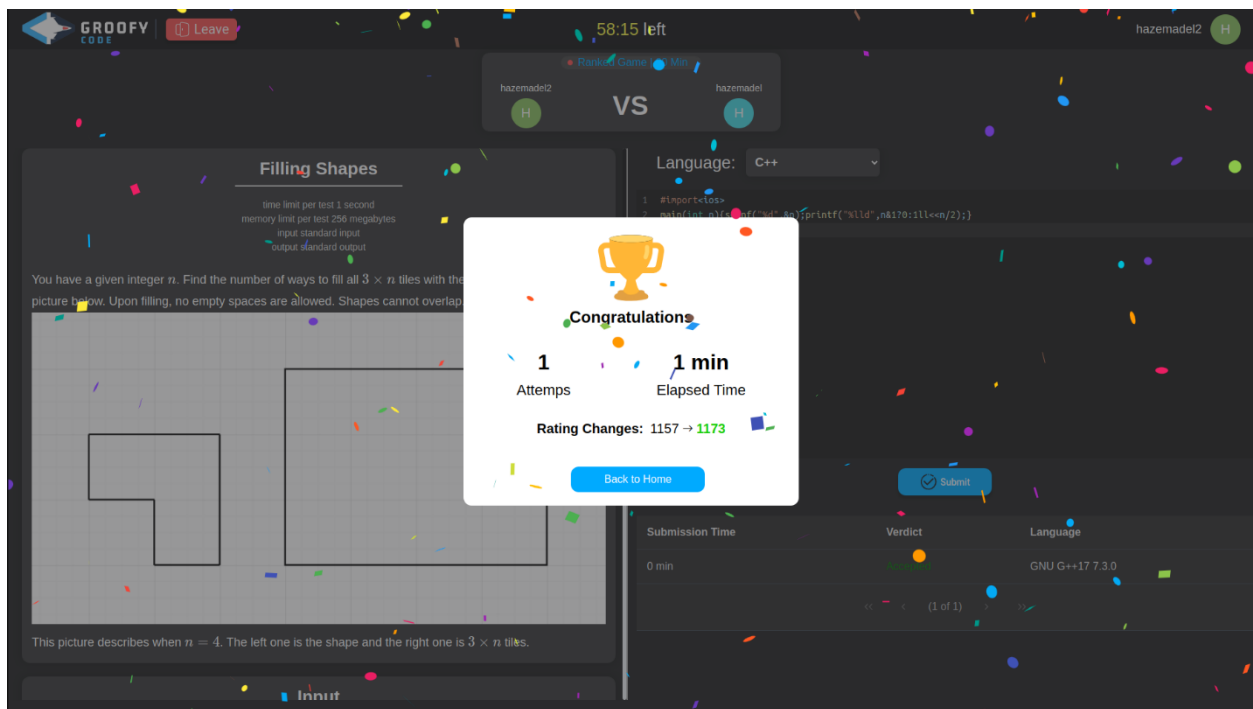


Figure 20 - Wining Ranked Match

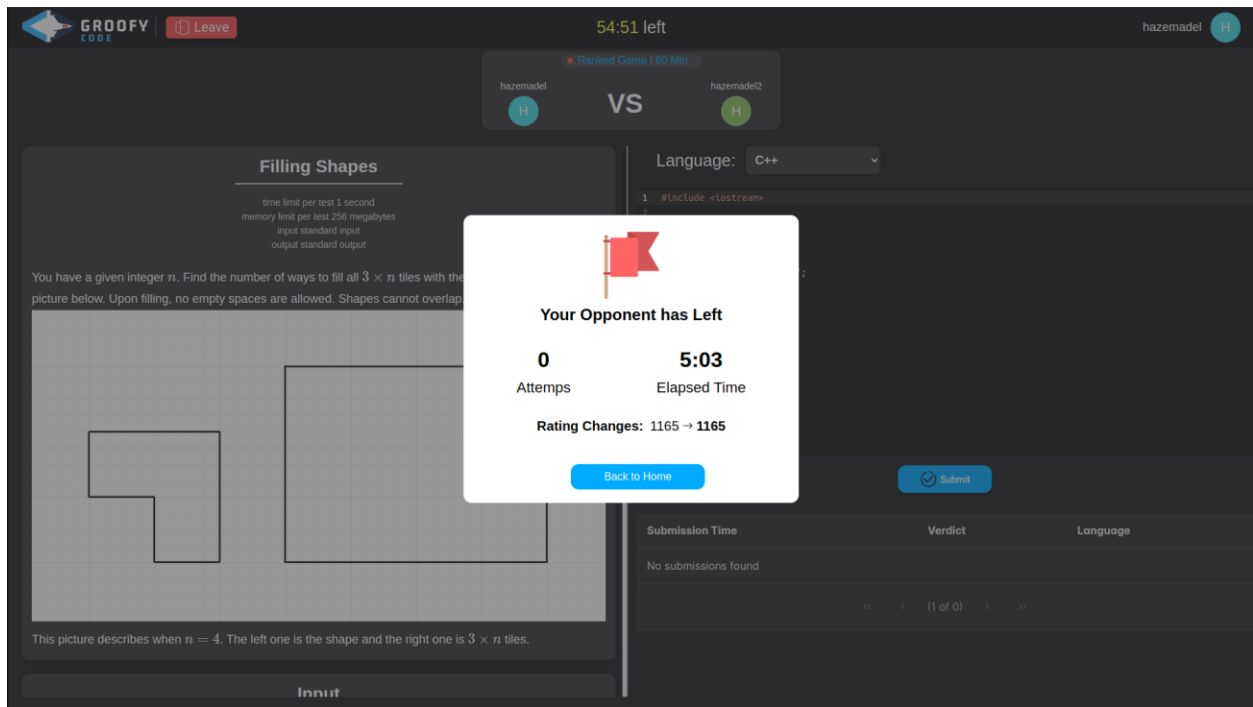


Figure 21 - Opponent has left

Chapter 5: Implementation and Testing

5.1 Challenges and Solutions

Developing Groofy Code has presented a variety of challenges, particularly related to technology, integration, time constraints, and decision-making in the implementation process. Below are the key challenges, their reasons, and the solutions implemented to address them.

5.1.1. Lack of Hands-on Experience with Used Technologies

Reason:

- The development team had limited prior experience with some of the chosen technologies, including Java Spring Boot, React.js, and machine learning algorithms.

Solution:

- **Training and Learning Resources:** The team invested time in self-paced learning through online courses, documentation, and tutorials to build proficiency in these technologies.
- **Collaborative Learning:** Team members shared knowledge and conducted peer review sessions to ensure a comprehensive understanding and to address any knowledge gaps.
- **Incremental Implementation:** Adopting an iterative approach allowed the team to gradually implement and test features, learning from each phase and improving upon the previous efforts.

5.1.2. Integration with Existing Websites like Codeforces

Reason:

- Codeforces limits the number of API requests, leading to issues with request blocking due to spamming.
- Integrating data from Codeforces into the feature model was challenging due to the lack of labels or answers.

Solution:

- **Rate Limiting Script:** Developing a script that waits 20-30 seconds between API requests ensured compliance with Codeforces' rate limits and prevented blocking.
- **Error Handling:** Implementing robust error handling and retry mechanisms to manage failed requests gracefully.
- **Optimized Data Requests:** Aggregating data requests to minimize the number of API calls needed.
- **Combining Unlabeled Data:** A custom algorithm was developed to derive meaningful results by analyzing counts of each rating solved, current rating, max rating, wins, losses, and draws.

5.1.3. Transition from Node.js to Java Spring Boot

The decision to switch from Node.js to Java Spring Boot was driven by the need for robust backend capabilities. Node.js doesn't support classes in straight way, has poor file management, and Java offers better performance and strong community support.

5.1.4. Difficulty of Java Configurations

Configuring Java environments, dependencies, and frameworks can be complex, especially when dealing with conflicts in dependencies and versions.

5.1.5. Choosing Between Machine Learning Algorithms: Random Forest vs. Clustering

Reason:

- Selecting the most appropriate machine learning algorithm to recommend problems and match users required balancing accuracy, complexity, and computational efficiency.

Solution:

- **Algorithm Evaluation:** Comparing the performance of Random Forest and clustering algorithms on historical user data to assess their accuracy and suitability.
- **Chosen Algorithm:** Opting for Random Forest due to its higher accuracy and better handling of complex relationships in the data.
- **Continuous Improvement:** Regularly reviewing and tuning the algorithm based on user feedback and performance metrics to ensure optimal recommendations.

By addressing these challenges methodically, Groofy Code aims to deliver a robust and user-friendly platform for coding enthusiasts.

5.2. Implementation

5.2.1. Machine Learning Implementation

Objective

The goal of this model is to predict the expected rating of users on GroofyCode, an online coding challenge platform, based on various user performance metrics.

Data Preprocessing

Loading Data: Multiple datasets (coders_dataset_1.xlsx to coders_dataset_26.xlsx) are loaded into Pandas DataFrames.

```
# Load multiple datasets
all_dfs = []
for i in range(1, 27): # Assuming datasets are numbered from 1 to 26
    file_path = f'coders_dataset_{i}.xlsx'
    df = pd.read_excel(file_path)
    df.fillna(0, inplace=True) # Replace all NaN values with 0
    df = calculate_expected_rating(df)
    all_dfs.append(df)
```

Handling Missing Values: All NaN values in the datasets are replaced with 0 using

```
df.fillna(0, inplace=True) # Replace all NaN values with 0
```

Combining Data: All DataFrames are concatenated into a single DataFrame (combined_df) using pd.concat.

```
combined_df = pd.concat(all_dfs, ignore_index=True)
```

The size of the data is about (50-60) k records.

Dropping Columns: The user_handle column is dropped from the combined DataFrame.

```
combined_df.drop(columns=['user_handle'], inplace=True)
```

Filling Remaining NaN: Any remaining NaN values are replaced with 0 again to ensure no missing data.

```
combined_df.fillna(0, inplace=True)
```

Features

The features used in this model are:

- user_rating: The current rating of the user.
- user_max_rating: The maximum rating the user has achieved.
- wins: The number of wins the user has.
- draws: The number of draws the user has.
- losses: The number of losses the user has.
- rate_800_cnt to rate_3500_cnt: The count of problems solved by the user at various difficulty levels, ranging from 800 to 3500.

Target Variable

- `expected_rating`: The predicted future rating of the user, calculated using a custom function based on various user performance metrics.

Feature Engineering: `calculate_expected_rating(df)`

The `calculate_expected_rating` function calculates the expected rating for each user in the dataset:

Weighting Difficulty Levels: A dictionary of weights for different difficulty levels is created, ranging from 0.5 to 3.2.

Performance Score Calculation: For each user, a performance score is calculated by summing the counts of problems solved at each difficulty level, weighted by the respective difficulty weights.

```
def calculate_expected_rating(df):
    MIN_DELTA, MAX_DELTA = 100, 900
    difficulty_weights = {
        800: 0.5, 900: 0.6, 1000: 0.7, 1100: 0.8, 1200: 0.9,
        1300: 1.0, 1400: 1.1, 1500: 1.2, 1600: 1.3, 1700: 1.4,
        1800: 1.5, 1900: 1.6, 2000: 1.7, 2100: 1.8, 2200: 1.9,
        2300: 2.0, 2400: 2.1, 2500: 2.2, 2600: 2.3, 2700: 2.4,
        2800: 2.5, 2900: 2.6, 3000: 2.7, 3100: 2.8, 3200: 2.9,
        3300: 3.0, 3400: 3.1, 3500: 3.2
    }
    def calculate_performance_score(row):
        score = 0
        for rating in range(800, 3501, 100):
            column_name = f'rate_{rating}_cnt'
            if column_name in row:
                score += row[column_name] * difficulty_weights[rating]
        return score
    performance_scores = df.apply(calculate_performance_score, axis=1)
```

Adjustment by Wins, Losses, Draws, and Max Rating: The performance score is adjusted based on the user's win-to-loss ratio, draws, and max rating.

```
performance_scores = df.apply(calculate_performance_score, axis=1)

# Incorporate wins, losses, draws, and max rating into the performance score
adjusted_scores = performance_scores * (1 + df['wins'] / (df['losses'] + 1)) + df['draws'] * 0.1
max_rating_influence = df['user_max_rating'] / 3500 # Normalizing the max rating influence
```



```
# Add max rating influence to the adjusted scores
adjusted_scores += adjusted_scores * max_rating_influence
```

Normalization: The adjusted performance scores are normalized to a range between MIN_DELTA (100) and MAX_DELTA (900).

```
min_perf, max_perf = adjusted_scores.min(), adjusted_scores.max()
normalized_performance_scores = (adjusted_scores - min_perf) / (max_perf - min_perf) *
(MAX_DELTA - MIN_DELTA) + MIN_DELTA
```

Expected Rating Calculation: The expected rating is calculated by adding the normalized performance scores to the user's current rating.

```
df['expected_rating'] = df['user_rating'] + normalized_performance_scores.clip(lower=MIN_DELTA,
upper=MAX_DELTA)
return df
```

Model Training

Feature and Target Definition:

```
# Define features and target
X = combined_df.drop('expected_rating', axis=1)
y = combined_df['expected_rating']
```

Data Splitting: The data is split into training and testing sets using an 80-20 split (train_test_split).

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Initialization and Training: A Random Forest Regressor model is initialized with 100 estimators and a random state of 42.

```
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Prediction and Evaluation: Predictions are made on the test set (model.predict). The Mean Squared Error (MSE) and R-squared (R2) score are calculated to evaluate the model's performance.

```
# Predict and evaluate
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
r2_percentage = r2 * 100
```

Model Evaluation

Mean Squared Error (MSE): A measure of the average squared difference between the actual and predicted values.

R-squared (R2) Score: Indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It is expressed as a percentage.

Mean Squared Error: 836.4096933043736

R-squared Percentage: 99.27%

Visualization

A scatter plot of actual vs. predicted ratings is created to visually assess the model's performance. The plot includes a line indicating perfect predictions ($y = x$).

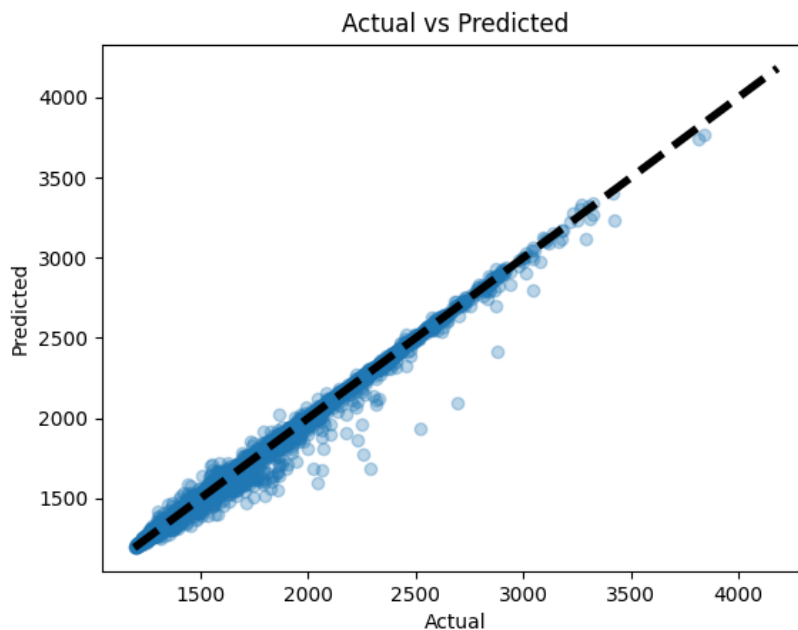


Figure 22 - Actual vs Predicted Target

Model Saving

The trained model is saved to a file (rating_prediction_rf_model.pkl) using `joblib.dump` for future use.

```
joblib.dump(model, 'rating_prediction_rf_model.pkl')
```

Conclusion

The Random Forest Regressor model predicts the expected rating of users on GroofyCode with a certain level of accuracy, evaluated using MSE and R-squared metrics. The model incorporates various user performance metrics and adjusts scores based on the user's historical performance and the difficulty level of problems solved. The model is saved for future predictions.

5.2.1.1 Flask API Implementation:

This Flask API is to provide an endpoint for predicting the expected rating of users on GroofyCode based on various user performance metrics using a pre-trained Random Forest Regressor model.

Flask API Endpoints

Endpoint: /predict

Method: POST

Description: Accepts JSON data containing user performance metrics and returns the predicted expected rating.

Steps:

1. Initialize the Flask App
2. Create an instance of the Flask class.
3. Load the Pre-trained Model
4. Load the model from a file using joblib. Handle the case where the model file is not found.
5. Define the /predict Endpoint.
6. Create a route for the /predict endpoint.
7. Extract the JSON data from the request.
8. Extract the features needed for prediction from the JSON data.
9. Use the pre-trained model to predict the expected rating.
10. Return the predicted rating as a JSON response.

```
app = Flask(__name__)
# Try to load the model, handle error if not found
try:
    model = joblib.load('rating_prediction_rf_model.pkl')
except FileNotFoundError:
    print("Model file not found. Please check the file path.")
    model = None
```

```

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json() # This assumes JSON data which includes the features

    # Extract features from the JSON data
    features = [
        data['user_rating'],
        data['user_max_rating'],
        data['wins'],
        data['draws'],
        data['losses'],
        data['rate_800_cnt'],
        data['rate_900_cnt'],
        data['rate_1000_cnt'],
        data['rate_1100_cnt'],
        data['rate_1200_cnt'],
        data['rate_1300_cnt'],
        data['rate_1400_cnt'],
        data['rate_1500_cnt'],
        data['rate_1600_cnt'],
        data['rate_1700_cnt'],
        data['rate_1800_cnt'],
        data['rate_1900_cnt'],
        data['rate_2000_cnt'],
        data['rate_2100_cnt'],
        data['rate_2200_cnt'],
        data['rate_2300_cnt'],
        data['rate_2400_cnt'],
        data['rate_2500_cnt'],
        data['rate_2600_cnt'],
        data['rate_2700_cnt'],
        data['rate_2800_cnt'],
        data['rate_2900_cnt'],
        data['rate_3000_cnt'],
        data['rate_3100_cnt'],
        data['rate_3200_cnt'],
        data['rate_3300_cnt'],
        data['rate_3400_cnt'],
        data['rate_3500_cnt']
    ]

    prediction = model.predict([features])
    return jsonify({'expected_rating': prediction.tolist()[0]})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

```

Conclusion

This Flask API provides a convenient way to predict the expected rating of users on GroofyCode based on various user performance metrics. The model is pre-trained and saved using joblib, and the API handles the prediction logic based on input features provided in JSON format.

5.2.2. Back-End Implementation

Groofy Code's backend is built using Java Spring Boot, following the Model-View-Controller (MVC) architecture. This structure promotes a clean separation of concerns, making the codebase more manageable and scalable.

5.2.2.1. Project Structure

The backend project is organized into several key packages:

- **Model:** Contains the data models representing entities in the database.
- **DTO (Data Transfer Object):** Contains classes used for data transfer between different layers of the application.
- **Service:** Contains the business logic and service layer.
- **Repository:** Contains the data access layer, interacting with the database.
- **Controller:** Contains the RESTful endpoints, handling incoming HTTP requests.
- **Configurations:** Contains configuration classes for the application.
- **Utilities:** Contains utility classes and helper functions.

5.2.2.2 API Call Flow and Package Interaction

When an API request is made to the Groofy Code backend, it follows a well-defined flow through various packages to ensure a structured and efficient processing of the request. Here's a step-by-step overview of the process:

1. **Controller Package**
 - **Initial Handling:** The request first hits the controller, which handles the HTTP endpoint.
 - **Example:** A request to /users is handled by UserController.
2. **DTO (Data Transfer Object) Package**
 - **Data Mapping:** The controller converts incoming JSON data to DTOs and vice versa for outgoing responses.
 - **Example:** UserDTO is used to transfer user data between the client and the server.
3. **Service Package**
 - **Business Logic:** The controller calls the service layer, where the main business logic is executed.

- **Example:** UserService handles operations like fetching user details, saving new users, etc.
- 4. **Repository Package**
 - **Database Interaction:** The service layer interacts with the repository to perform CRUD operations on the database.
 - **Example:** UserRepository interacts with the database to fetch, save, or delete user data.
- 5. **Model Package**
 - **Entity Representation:** The repository uses model classes to represent database tables.
 - **Example:** User class is mapped to the users table in the database.
- 6. **Configurations Package**
 - **Application Settings:** Configuration settings are applied, such as security configurations, WebSocket settings, etc.
 - **Example:** SecurityConfig manages the security aspects of API endpoints.
- 7. **Utilities Package**
 - **Helper Functions:** Utility classes provide common functionalities used across the application.
 - **Example:** JwtTokenUtil is used for JWT token generation and validation during authentication processes.

Example API Call Flow:

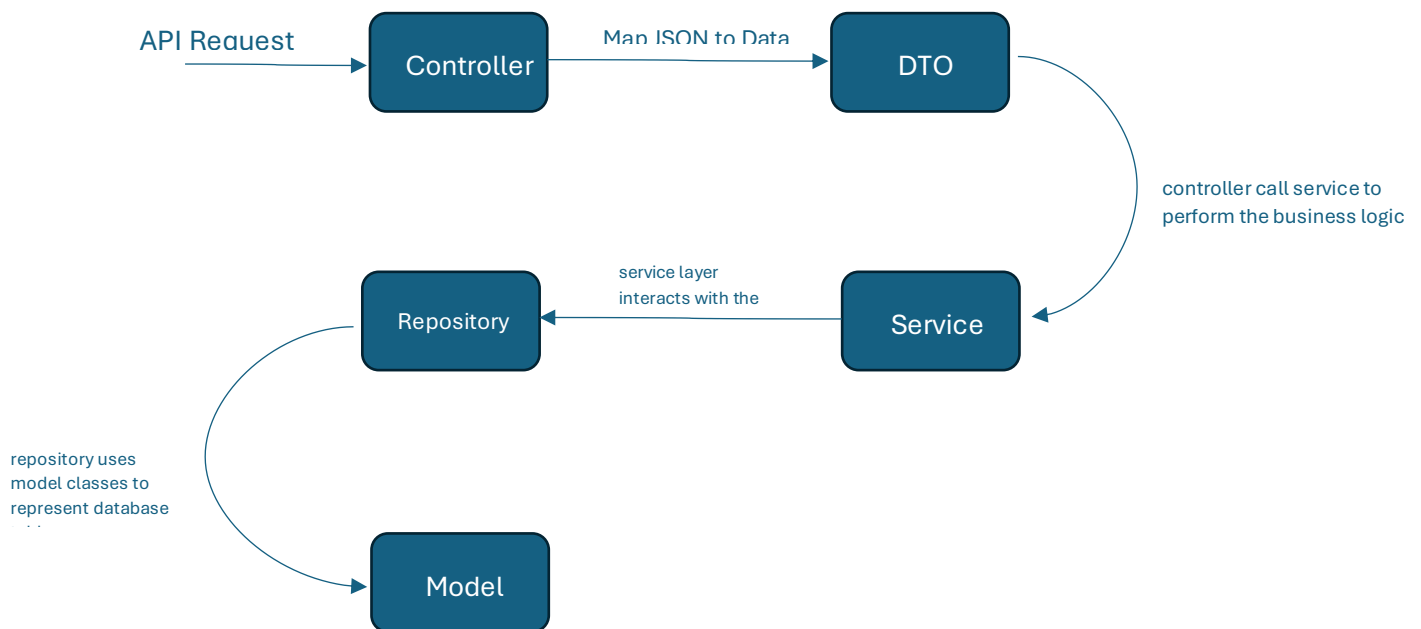


Figure 23 - API Call Flow

5.2.2.3 Part of the code Example of getAllUsers API:

1. UserController:

```
@RestController
@SecurityRequirement(name = "bearerAuth")
public class UserController {
    private final UserService userService;

    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping
    public ResponseEntity<Object> getAllUsers() throws Exception {
        return userService.getAllUsers();
    }
}
```

2. UserService

```
@Service
public class UserService implements UserDetailsService {
    @Autowired
    private final UserRepository userRepository;

    @Autowired
    private final ModelMapper modelMapper;
    @Autowired
    private final PasswordEncoder passwordEncoder;

    public ResponseEntity<Object> getAllUsers() throws Exception {
        try {
            List<UserModel> users = userRepository.findAll();
            List<UserDTO> userDTOS = users.stream().map(user -> modelMapper.map(user,
UserDTO.class)).toList();

```

```

        return ResponseEntity.ok(ResponseUtils.successfulRes("Users retrieved successfully",
userDTOS));
    } catch (Exception e) {
        throw new Exception(e);
    }
}
}

```

3. UserRepository

```

@Repository
public interface UserRepository extends JpaRepository<UserModel, Long> {
}

```

4. UserModel

```

@Entity
@Table(name = "users")
@Getter
@Setter
public class UserModel implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false, length = 100)
    private String username;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(nullable = false)
    private String password;

    private String displayName;

    @Column(length = 100)
    private String country;

    @Column(length = 1000)
    private String bio;
}

```


Conclusion

This structured flow ensures that each layer in the Groofy Code backend has a specific role, promoting separation of concerns and maintainability. By following this organized flow, your project can handle requests efficiently, making the platform robust and scalable, which is crucial for a dynamic and interactive coding platform like Groofy Code.

5.2.3. Front-End Implementation

5.2.3.1 Project Structure:

- Public folder: Contains all the assets of the website (images, SVG, etc.).
- Components: Contains different UI parts of the system (like buttons, input fields, etc.).
- Pages: Contains the main pages of the website, utilizing the components in its structure.
- Utilities: Contains shared CSS styling, interface types, etc.
- Redux Store: Contains slices and actions. Slices are the memory-stored items used for displaying dynamic content. Actions are functions that interact with backend endpoints.

5.2.3.2 Frontend Flow:

- When a button is clicked, a page is refreshed or navigated to, or any interaction occurs, a Redux action is called.
- The Redux action takes the request parameters (if any) and sends a request to the required endpoint.
- When a response is returned, it is stored in Redux slices, as it will be used on different pages of the website.
- When a Redux slice is updated, this reflects on the pages, showing the required items from the slice that was updated with the endpoint response data.

5.3. Testing

5.3.1. Back-End Testing

Testing the backend thoroughly is crucial for ensuring that all APIs function correctly under various scenarios. Using tools like Postman and Swagger can streamline this process. Below, I'll outline a structured approach to creating a Postman collection and using Swagger for comprehensive backend testing.

Swagger tests register API:

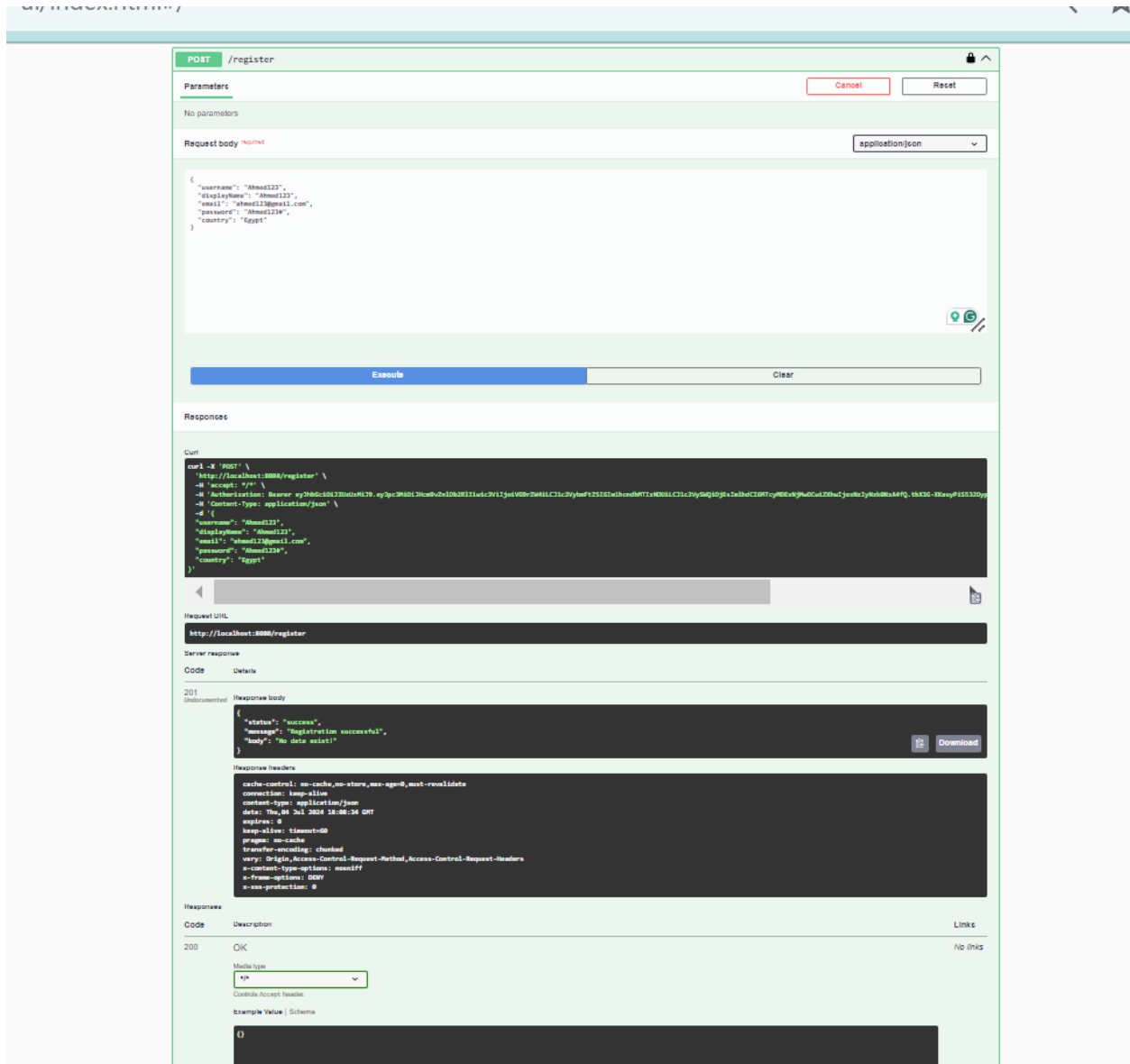


Figure 24 - Swagger

Postman tests Login API:

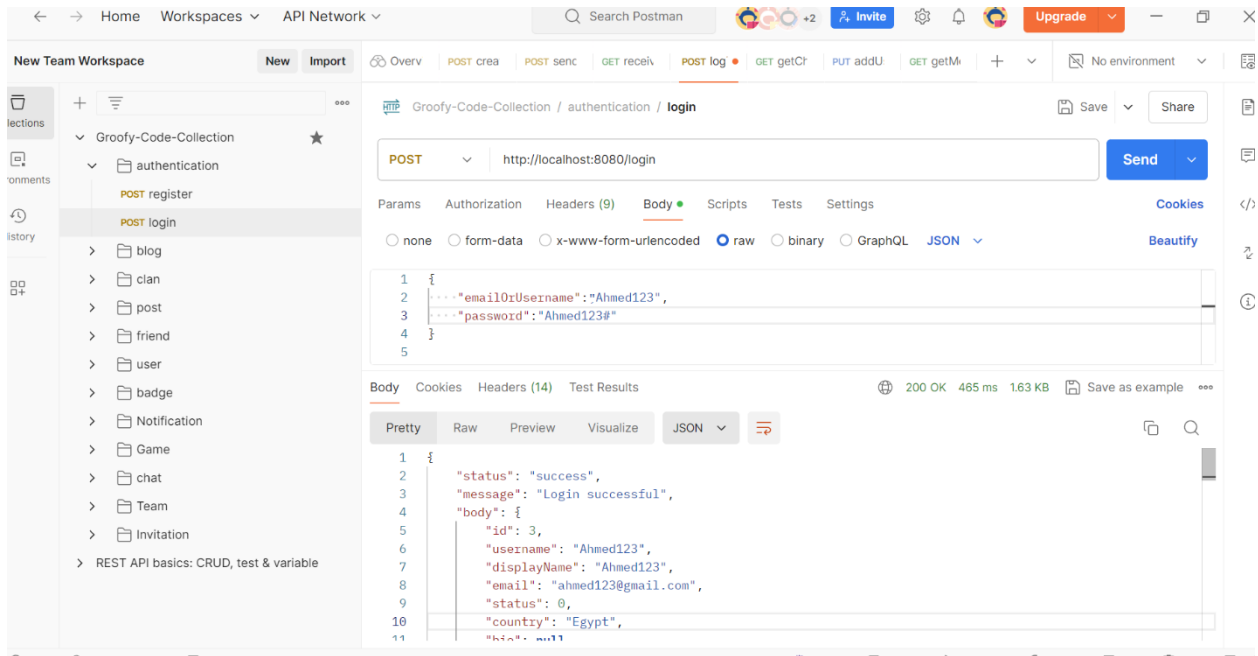


Figure 25 - Postman

5.3.2. Front-End Testing

1. Login when username is empty, or password is empty

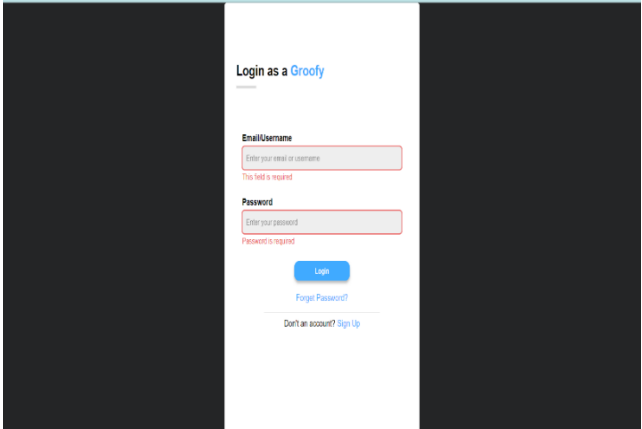
Action	Expected Output	Actual Output	Screenshot from the website
Press login button	Error message	Error message (This field is required)	 <p>The screenshot shows a login form titled 'Login as a Groofy'. It has two input fields: 'Email/Username' and 'Password'. Both fields have red borders and red error messages below them: 'This field is required' for the email/username field and 'Password is required' for the password field. There is a blue 'Login' button and links for 'Forgot Password?' and 'Don't an account? Sign Up'.</p> <p>Figure 26 - required fields (login)</p>

Table 1 - required fields (login)

2. Login in when username or password is Invalid:

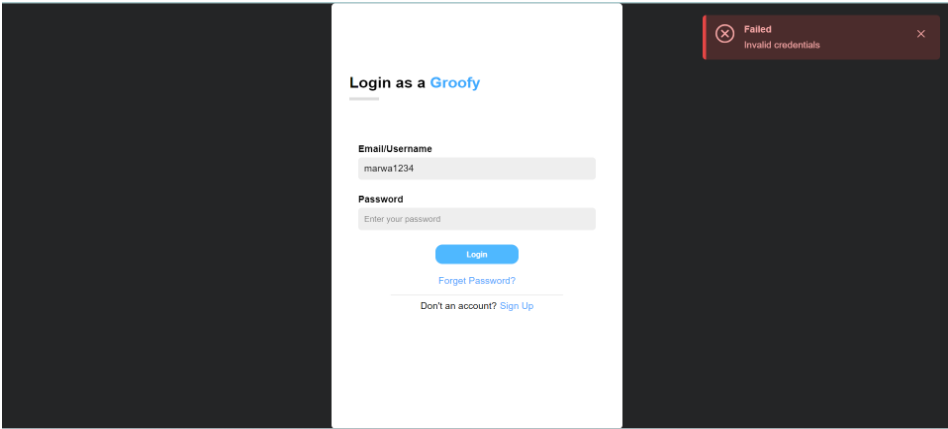
Action	Expected Output	Actual Output	Screenshot from the website
Press login button	Error message	Error message (Failed Invalid credentials)	 <p>The screenshot shows the same login form as Figure 26. The 'Email/Username' field now contains the text 'manwa1234'. A red error message 'Failed Invalid credentials' is displayed at the top right of the form area. The 'Password' field is empty. The 'Login' button and other links are still visible.</p> <p>Figure 27 -Invalid credentials</p>

Table 2 - Invalid credentials

3. Create Post

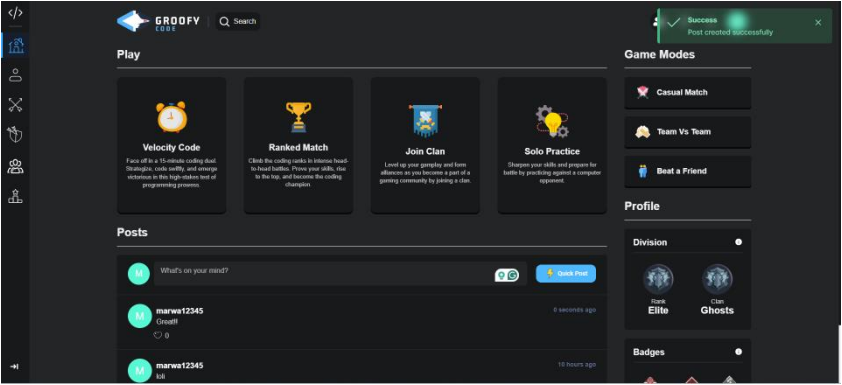
Action	Expected Output	Actual Output	Screenshot from the website
Press Quick Post button	Post Created Successfully	Success Message (Success Post Created successfully)	 <p>Figure 28 - Create Post</p>

Table 3 – Create Post

4.Delete Post

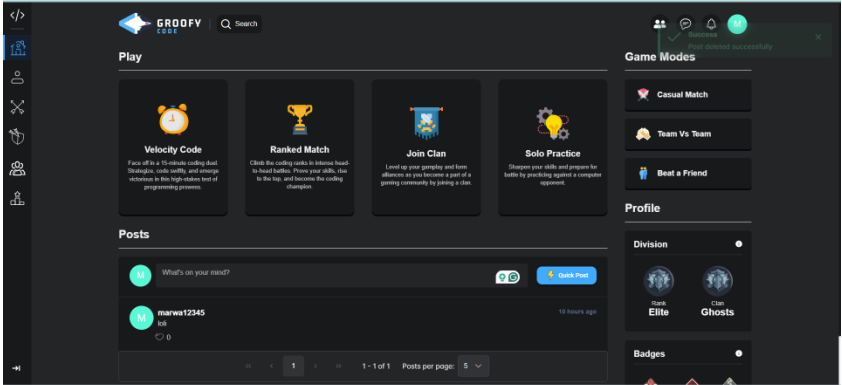
Action	Expected Output	Actual Output	Screenshot from the website
Press Delete button	Post Deleted Successfully	Success Message (Success Post Deleted successfully)	 <p>Figure 29 - Delete Post</p>

Table 4 - Delete Post

5. Start Solo Match

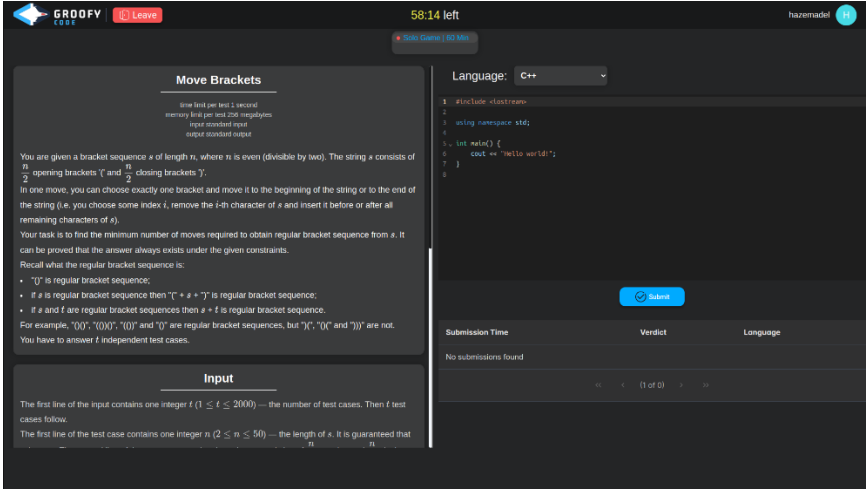
Action	Expected Output	Actual Output	Screenshot from the website
Press Solo Practice button	Start Solo Practice Match	Success (Start Solo Match successfully)	

Table 5 - Start Solo Match

6.Submit compiler error

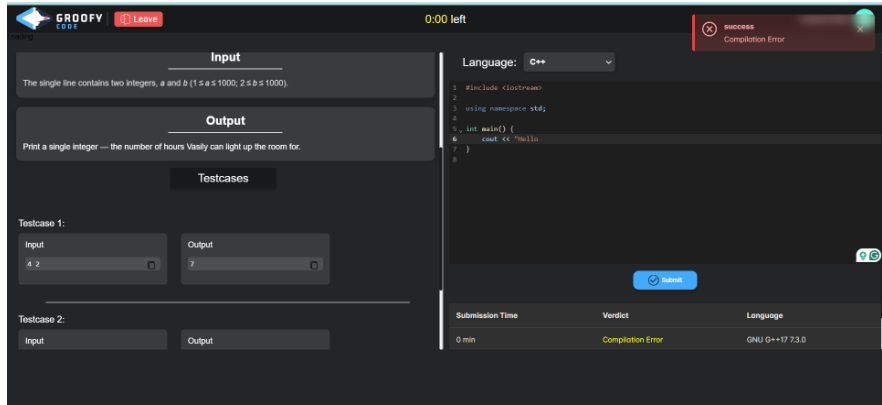
Action	Expected Output	Actual Output	Screenshot from the website
Press submit button with missing semicolon	Return Compiler error message	Return message (Compiler Error)	

Table 6 - Submit compiler error

7. Submit wrong answer

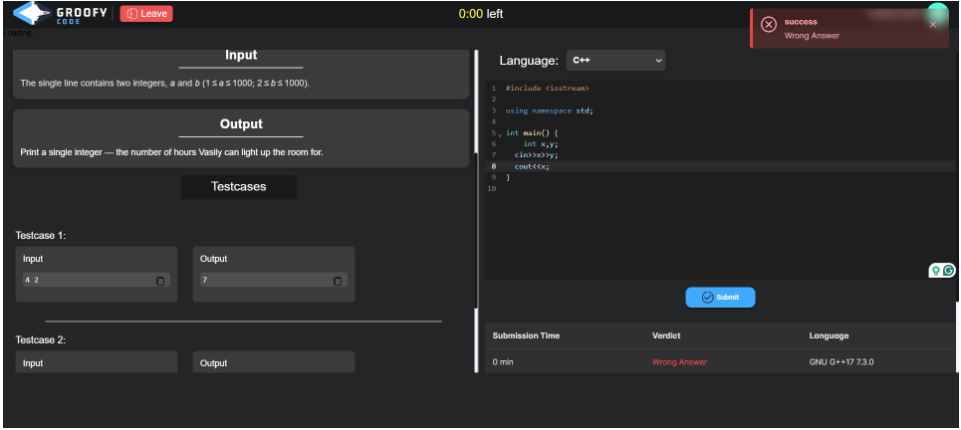
Action	Expected Output	Actual Output	Screenshot from the website
Press submit button without taking any input	Return wrong answer message	Return message (Wrong Answer)	

Table 7 - Submit wrong answer

8. Start Ranked Match

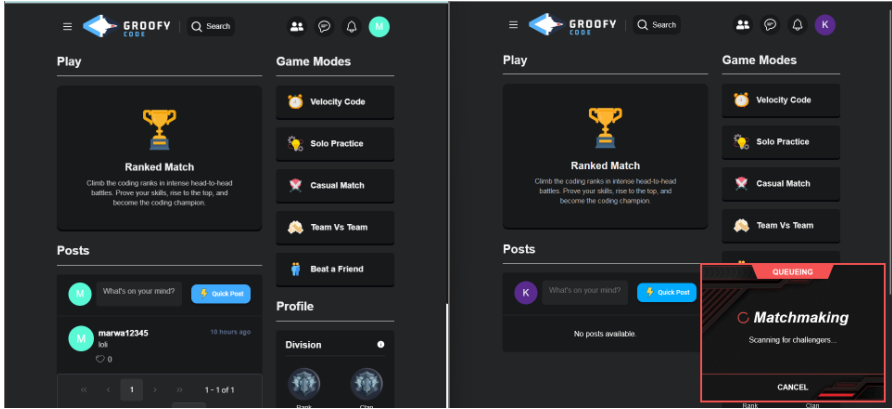
Action	Expected Output	Actual Output	Screenshot from the website
Press Ranked Match button	Start Matching player	Return Dialog Queuing and waiting to match you	

Table 8 - Start Ranked Match

9.Game Started

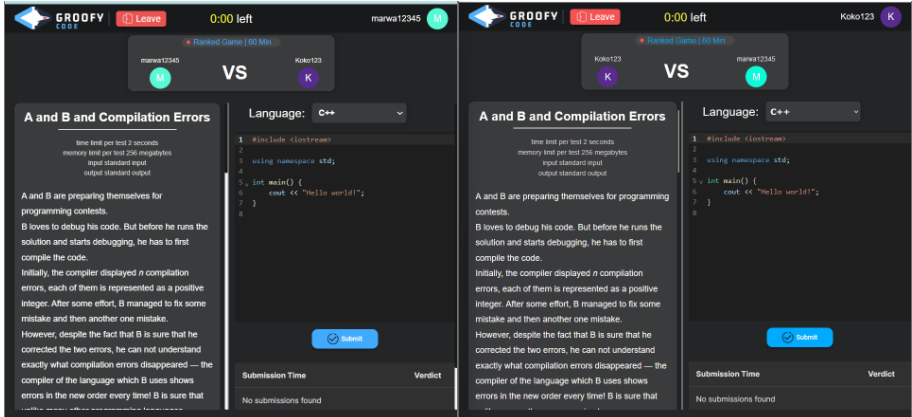
Action	Expected Output	Actual Output	Screenshot from the website
When Matching a player	Start the Ranked Match	Successfully start the match	

Table 9 - Ranked Match after Matching player

10.Wining the match

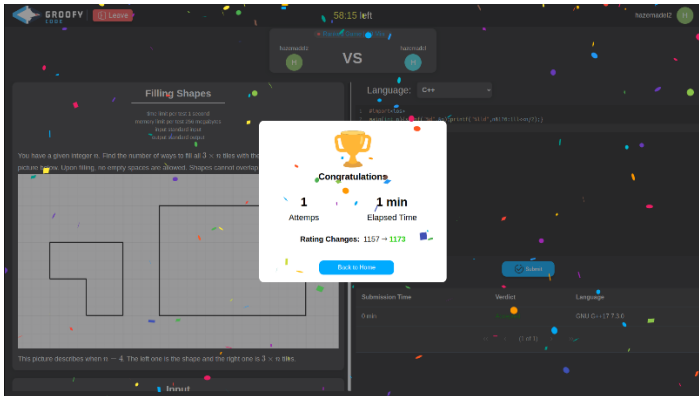
Action	Expected Output	Actual Output	Screenshot from the website
the player win the match	Another player loses the match	Return Dialog tell another player that he loses the match	

Table 10 - Wining the match

11. Leave the match

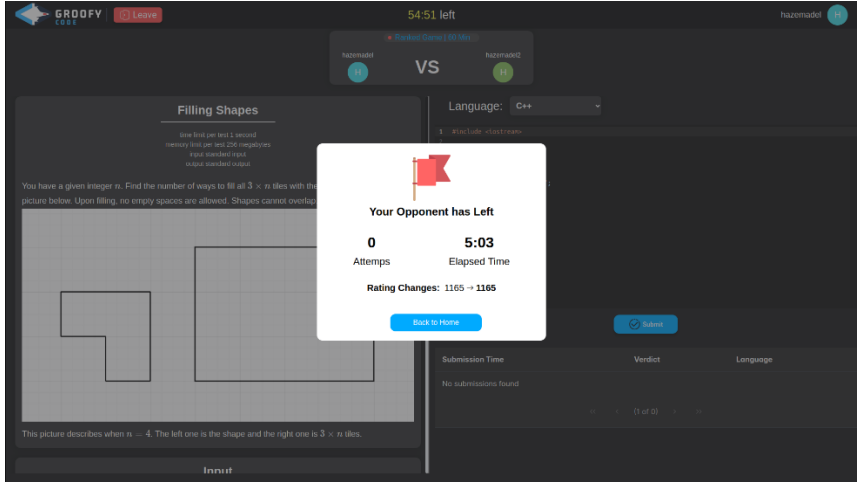
Action	Expected Output	Actual Output	Screenshot from the website
One of them leave the match	The game ends for both and the player that left the game lose rate	Return Dialog tell another player that the other player left the match	

Table 11 - Leave the match

12. Search about user

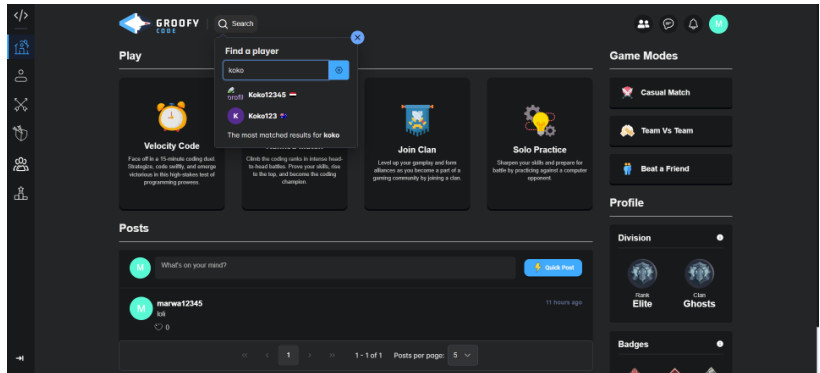
Action	Expected Output	Actual Output	Screenshot from the website
Enter the username and press search button	Find the users that start by this search word	Return all the users that its prefix as the search word	

Table 12 - Search about user

13.Send Message

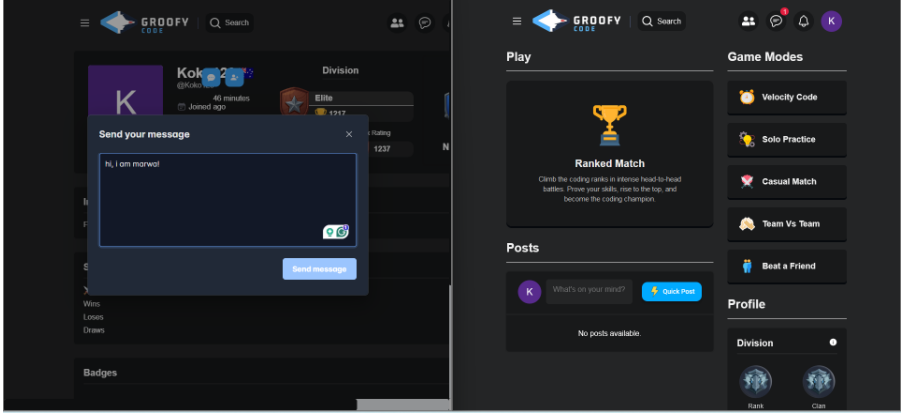
Action	Expected Output	Actual Output	Screenshot from the website
Press chat button and write message then send	The receiver user receives notification of the message	The receiver user receives notification of the message (Notification icon increases)	

Table 13 - Send Message

14.Send Friend request

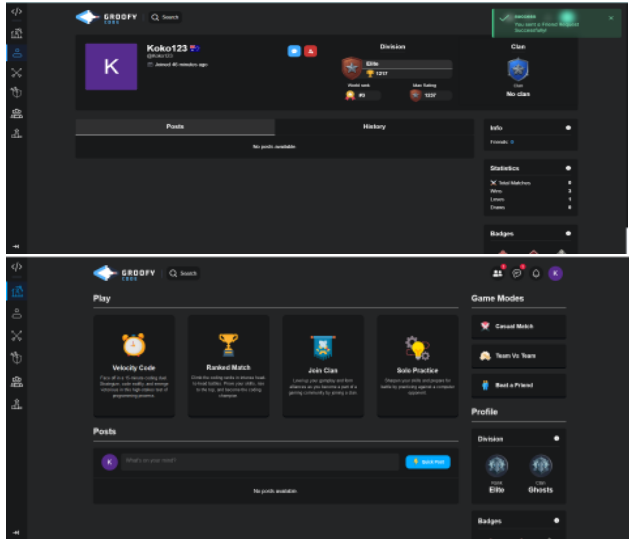
Action	Expected Output	Actual Output	Screenshot from the website
Press sends request button	The receiver user receives notification of the friend's icon	The receiver user receives notification of the request (Notification icon increases)	

Table 14 - Send Friend request

15.Chat sending message

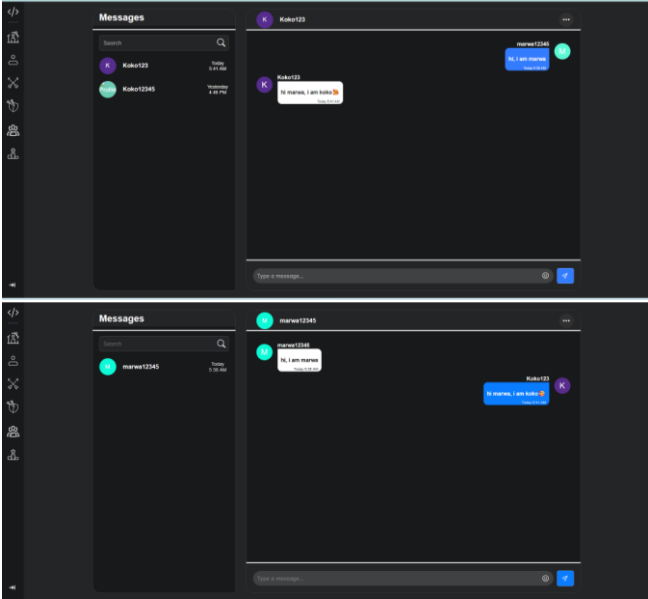
Action	Expected Output	Actual Output	Screenshot from the website
Send message	The receiver user receives the message without any loading	The message appears at the receiver chat	

Table 15 - Chat sending message

References

- https://en.wikipedia.org/wiki/Elo_rating_system (Elo System)
- <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html> (Random Forest Regressor)
- <https://codeforces.com/apiHelp> (Collecting Data)
- [https://en.wikipedia.org/wiki/Normalization_\(statistics\)](https://en.wikipedia.org/wiki/Normalization_(statistics)) (Normalization)
- <https://primereact.org/> (Prime React for Components)
- <https://redux.js.org/> (Redux)
- <https://www.npmjs.com/> (npm packages)
- <https://react.dev/reference/react> (React Documentation)
- <https://rapidapi.com/judge0-official/api/judge0-ce> (Judge0 for the Editor)
- <https://www.svgrepo.com/> (Collecting SVGs & Images)
- <https://colorhunt.co/> (Choosing Colors & Palette)
- <https://spring.io/guides/gs/messaging-stomp-websocket/> (messaging - web socket)
- <https://www.baeldung.com/websockets-spring> (spring web socket)
- <https://spring.io/guides/gs/securing-web> (spring security)
- <https://spring.io/projects/spring-data-jpa> (spring data JPA)