CodeSphere: A Comprehensive Contest and Learning Platform for IIT

1. Introduction

In the evolving landscape of technical education and competitive programming, CodeSphere emerges as a cutting-edge platform designed to revolutionize the way contests are conducted and learning is approached within the Indian Institutes of Technology (IIT). This Software Requirement Specification (SRS) document delves into the detailed requirements for CodeSphere, highlighting its commitment to fostering a dynamic, interactive, and inclusive environment for institutional students (contestants), problem setters, and administrators (admins).

1.1 Purpose

The purpose of this SRS document is to outline the functional and nonfunctional requirements, features, and system design of CodeSphere. It serves as a formal agreement between stakeholders and the development team, ensuring a shared understanding of the system's objectives and specifications. By detailing requirements for registration, authentication, problem repositories, contest participation and hosting, knowledge sharing, notifications, and user profiles, this document aims to guide the development process toward creating a platform that meets the needs of all users effectively.

1.2 Intended Audience

This document is intended for:

1. Institutional Students (Contestants): To understand the features and services available for enhancing their learning and competitive programming skills.
2. Problem Setters: To grasp the functionalities provided for creating, submitting, and managing problems within the platform.

3. Administrators: To comprehend the oversight, management, and maintenance capabilities of the platform.
4. Development Team: To have a clear set of requirements for designing, developing, and testing the platform.
5. Others: To ensure the platform aligns with the strategic goals and educational objectives of IIT.

1.3 Conclusion

CodeSphere aims to become a cornerstone in the technical education ecosystem of IIT, offering a robust platform for contests, learning, and community engagement. This SRS document lays the groundwork for developing a system that is not only functional and reliable but also enriches the educational journey of its users.

2. System Overview

CodeSphere integrates various modules to create a comprehensive environment that supports the growth and development of its users through competitive programming and learning opportunities.

2.1 Registration and Authentication System
Facilitates user onboarding with distinct pathways for admins (requiring a pin code) and contestants, including problem setters, with email verification to ensure authenticity.

2.2 Problem Repository
Acts as a central hub for storing, accessing, and learning from past contest problems, complete with solutions and educational resources.

2.3 Contest Module
Enables users to participate in or host contests, with functionalities supporting problem setting, contest registration, live leaderboards, and detailed contest information dissemination.

2.4 Knowledge Nexus
Encourages knowledge sharing through blogs and solution visualization tools, aiding in the understanding of complex problems and coding strategies.

2.5 Notification System
Ensures timely communication regarding contests, platform updates, and personalized notification preferences for an enhanced user experience.

2.6 User Profile
Provides a detailed overview of user activity, achievements, team participation, and problem-setting history, alongside personal and account settings management.

2.1. Stakeholders

CodeSphere caters to the diverse needs of:

1. **Institutional Students (Contestants)**: Engage in contests, access learning resources, and contribute to the community.
2. **Problem Setters**: Contribute problems to contests, enriching the platform's content and offering unique challenges to contestants.
3. **Administrators**: Oversee platform management, content moderation, and user support to maintain a high-quality, secure environment.

2.2. Identifying The Multiple Viewpoints Of The Stakeholder

The development of CodeSphere involves multiple stakeholders, each with their unique perspectives and expectations from the platform. Understanding these diverse viewpoints is crucial for tailoring the platform to meet the varied needs of its users. Below are the primary stakeholders and their respective viewpoints:

1. Institutional Students (Contestants)

- Learning and Growth: Contestants seek a platform that offers a rich repository of problems and learning resources to help them understand complex concepts and improve their problem-solving skills.

- Competitive Spirit: They value the competitive aspect of the platform, desiring transparent, fair contests with real-time leaderboards to gauge their performance against peers.
- Community Engagement: A desire for a collaborative environment where they can share knowledge, discuss problems, and learn from each other.

2. Problem Setters

- Creative Expression: Problem setters view the platform as a creative outlet where they can design challenging problems that test various aspects of programming and algorithmic thinking.
- Recognition and Impact: They seek recognition for their contributions and want to see their problems engage and challenge the community, contributing to the learning process.
- Ease of Use: A user-friendly problem submission and management interface that simplifies the process of setting problems and receiving feedback.

3. Administrators

- Platform Integrity and Security: Administrators prioritize maintaining a secure, fair, and cheat-free environment for contests and interactions on the platform.
- User Management: They require efficient tools for managing user accounts, including resolving disputes, handling misconduct, and ensuring compliance with platform rules.
- Performance and Scalability: A focus on the platform's ability to handle large volumes of users and data efficiently, ensuring high uptime and smooth operation during peak usage times.

3. Elicitation of CodeSphere

For the CodeSphere project, a comprehensive requirements elicitation process is crucial to ensure the platform meets the expectations of its diverse stakeholders, including institutional students (contestants), problem setters, and administrators.

This process involves collaborative requirements gathering, quality function deployment (QFD), and the development of usage scenarios to establish a solid foundation for the platform's features and functionalities. The elicitation process for CodeSphere has led to the identification of normal, expected, and exciting requirements, as detailed below:

3.1 Collaborative Requirements Gathering

Stakeholder meetings were conducted with institutional students, problem setters, and administrators to understand their needs, challenges, and expectations from the CodeSphere platform. These discussions facilitated a mutual understanding of the problems and potential solutions, allowing for the negotiation of requirements and the specification of preliminary solution elements.

3.2 Quality Function Deployment

QFD was employed to align the platform's development with the specific needs and wants of its users. This approach ensured that CodeSphere's functionalities directly responded to the stakeholders' expectations, enhancing user satisfaction and engagement.

3.2.1 Normal Requirements

The fundamental requirements for CodeSphere, derived from stakeholder meetings, include:

- **User Authentication:** Secure login and sign-up processes for contestants, problem setters, and administrators.
- **Problem Submission and Management:** A streamlined process for problem setters to submit problems, with support for attaching resources and test cases.
- **Contest Registration and Participation:** Easy-to-use interfaces for contestants to register for and participate in contests.
- **Real-Time Leaderboards:** Dynamic leaderboards updating in real-time during contests to reflect standings.

- **User Profiles:** Customizable profiles displaying user achievements, problem-solving history, and participation records.

3.2.2 Expected Requirements

- **Intuitive User Interface**: A design that's consistent across different screen sizes.
- **Responsive Design:** Ensure the platform is accessible and navigable on smartphones, tablets, and desktops.
- **Notification System**: Alert users about contest registrations, upcoming contests, and new forum replies.
- **Privacy Controls for User Profiles:** Users can control which parts of their profile are public, including hiding certain contest performances or personal information.
- **Dynamic Filtering in Problem Repository:** Enable dynamic filtering options in the problem repository for users to filter problems by difficulty, most solved, and whether they have been attempted or solved.
- **Real-time Feedback on Submissions:** Provide immediate feedback on submissions during contests, indicating whether the solution is correct or incorrect.

3.2.2 Exciting Requirements

- **Solution Visualization:** Implement a feature allowing users to visualize their solution's execution flow against provided test cases, highlighting where in the code specific inputs are processed and outputs are generated.
- **Custom Problem Settings:** Enable users to create custom problems.
- **Team Registration with unified team ID:** Implement a feature where teams can register for contests as a single entity, with each team member's ID linked to a unique team ID. This ensures that all submissions by the team are tracked under one identifier, but individual contributions are also recognized

3.3 Usage Scenario

1. Registration and Authentication System:

- User Registration
- User Login

User Registration:

Actor: New user (Admin, Contestant)

New users begin registration by visiting the platform's registration page and get two options: admin, contestant. If choosing admin, a pincode is required for further processing. For contestants, register with a valid email, password and username. Problem-setter is a contestant. The system validates username uniqueness and email format compliance. Upon successful validation, an email verification link is sent to the provided email address. Users confirm their email by clicking the link and are redirected to the login page.

User Login:

Actor: (Admin, Contestant)

Registered users log in by entering their username or email and password on the login page. The system verifies this information against stored data. Upon successful validation, users gain access to their account dashboard.

2. Problem Repository:

Actor: Contestant

The Problem Repository serves as a vital resource within the competition ecosystem, providing a structured platform for storing and accessing past contest problems. Through this repository, contestants gain the invaluable opportunity to revisit and engage with problems from previous competitions, enabling them to reinforce their understanding of core concepts and develop advanced problem-solving strategies. Moreover, the inclusion of solutions alongside the problems facilitates a comprehensive learning experience, allowing contestants to

analyze different approaches and techniques employed to tackle each problem effectively.

3. Contest:

- Participating
- Hosting

Participating in Contest :

Actor: Contestant

Contestants navigate to the Contest section of the platform to select from upcoming contests. After choosing a contest, they register individually or as part of a group. Once registered, contestants await the contest start. During the contest, they access the contest interface and submit solutions. The leaderboard updates in real-time, reflecting contestant standings based on submissions and penalties.

Hosting Contest:

Actor: Problem-Setter, Admin

- Problem setting
- Contest details submission
- Launch contest

Problem Setting:

Actor: (Problem-setter)

Users can access the Problem Setting sub-module within the Contest module. In the Problem Setting interface, users have the option to submit new problems for contests by providing detailed problem statements, title, constraints, and sample test cases. Additionally, users are required to submit their proposed solutions along with test cases to validate the correctness and efficiency of the problem solutions. Once submitted, the proposed problem undergoes a verification process conducted by platform administrators.

The contest hosting module facilitates the seamless organization and execution of contests following the approval of custom problems. This module encompasses essential steps such as setting the contest start and end time, configuring problem submission and evaluation settings, and ensuring all necessary setup is complete. Once the technical aspects are finalized and thoroughly tested, the module facilitates the dissemination of contest details to potential participants through email. Information provided includes the contest format, rules, registration process, and any other relevant details of contest schedule for participants to engage effectively.

4. Knowledge Nexus:

- Blog
- Solution visualization

Blog:

Actor: Contestant

Users, including problem-setters, and contestants, can access the Learning section of the platform to contribute insightful blog posts. The interface provides user-friendly tools for creating posts covering programming, algorithms, data structures, and problem-solving with the title of the content, count of likes and comments. Before publication, a prompt appears, ensuring compliance with the platform's privacy policy. Users are asked if their post violates any privacy policies. If they choose 'no,' the blog is then published for all users to access.

Solution Visualization:

Actor: Contestant

The system generates control flow graphs for test cases which aid users in understanding the flow of code to handle cases effectively, thereby enhancing problem-solving strategies. Users can visualize the flow of code through

corresponding test cases, gaining insights into the execution paths and facilitating comprehensive analysis of the code's behavior.

5. Notification:

- Contest Notification
- Customized Notification Preference

Contest Notifications:

Actor: User, email

Contestants receive email notifications for upcoming contests, results, and announcements. Upon registration, they provide their email addresses, which serve as the primary communication channel. The system automatically sends notifications for scheduled contests, results, and platform updates. Contestants are promptly informed about contest details like start times, duration, and problem sets, as well as rankings. Additionally, they receive updates on platform enhancements, features, and community events.

Customized Notification Preferences:

Users can customize their notification preferences to suit their needs and priorities. In their settings or preferences section, users access the Customize Notification Preferences feature. They select the types of notifications they want, such as contest reminders or blog post updates. Users also choose the frequency of notifications, like immediate updates or daily digests. This customization ensures users receive relevant information according to their preferences, enhancing their experience on the platform.

6. User Profile:

- Profile overview
- Settings
- Team

- Problem setting history
- Admin dashboard

Profile Overview:

Actor: Contestant

Upon visiting the profile page, contestants are presented with a comprehensive overview of their account information, encompassing various aspects of their engagement on the platform. This module includes three key submodules: Personal information, Activity History, Achievements and Badges.

> **Personal information:** This submodule includes username, profile picture, a brief bio, university name, batch, email address, friends, Last seen.
>
> **Activity history:** Users can view their activity history including a log of their recent interactions on the platform such as Submission history including successful solutions and compilation errors, Total number of problems solved, Contest participation, Total number of blog publications.
>
> **Achievements and Badges:** Users can see their achievements and earn badges on the platform. Badges may be awarded for various accomplishments such as solving a certain number of problems, participating in contests, contributing to the community or achieving specific rankings.

Settings:

Actor: Admin, Contestant

Registered users can update their account information by logging in and accessing the profile settings. Here, they can modify details like email address, password, and profile information. After making changes, users submit the updated details for system validation. Upon successful validation, the system updates the user's profile to maintain accurate information.

Team:

Actor: Contestant

The "Team" submodule within the user profile provides a comprehensive glance at the user's involvement within various teams. It offers essential information such as the number of teams the user is currently associated with, along with the names of each team and its respective members. Whether one is a contestant collaborating with teammates or a problem setter coordinating with a contest hosting team, this module streamlines communication and coordination by presenting crucial team details in a concise and accessible format within the user profile.

Problem Setting history:

Actor: Problem-Setter

This module is specifically designed for contestants who actively participate in problem setting. It offers a comprehensive overview including timestamps of the problems they have set and the status of each whether approved or pending. Additionally, it displays the titles or brief descriptions of the problems authored by the problem setter. Moreover, it provides insights into the engagement of contestants with these problems indicating how many contestants have attempted or solved each respective problem.

Admin Dashboard:

Actor: Admin

Admins navigate to the Admin Dashboard section, accessible from the platform's navigation menu. Upon arrival, admins are presented with an overview of user activity, content moderation, and platform performance. Admins have access to tools for managing user accounts, including the ability to view user profiles, reset passwords, and suspend or ban accounts as necessary. The Admin Dashboard provides admins with insights into content moderation, allowing them to review reported content, moderate discussions, and enforce community guidelines. Admins can customize platform settings, manage email notifications, and configure other administrative preferences from the Admin Dashboard.

4. Use case diagrams

Use Case diagrams give the non-technical view of the overall system.

Level - 0

Use Case Name: CodeSphere - Competitive Programming & Learning based Platform
Primary Actors: Admin, Contestant
Secondary Actors: Email

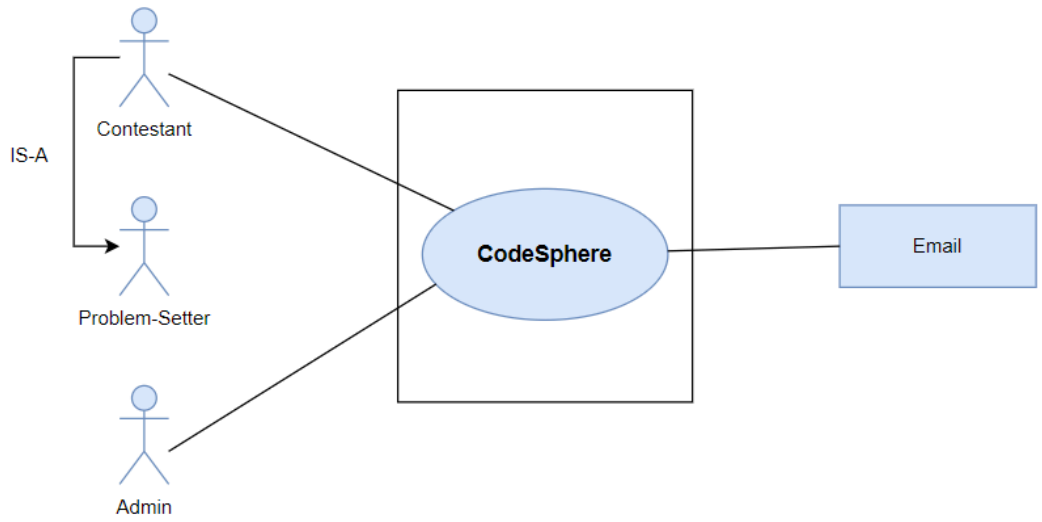


Figure 3: Use case diagram level 0: CodeSphere

Level - 1

Use Case Name: CodeSphere - Competitive Programming & Learning based Platform (detailed)
Primary Actors: Admin, Contestant
Secondary Actors: Email

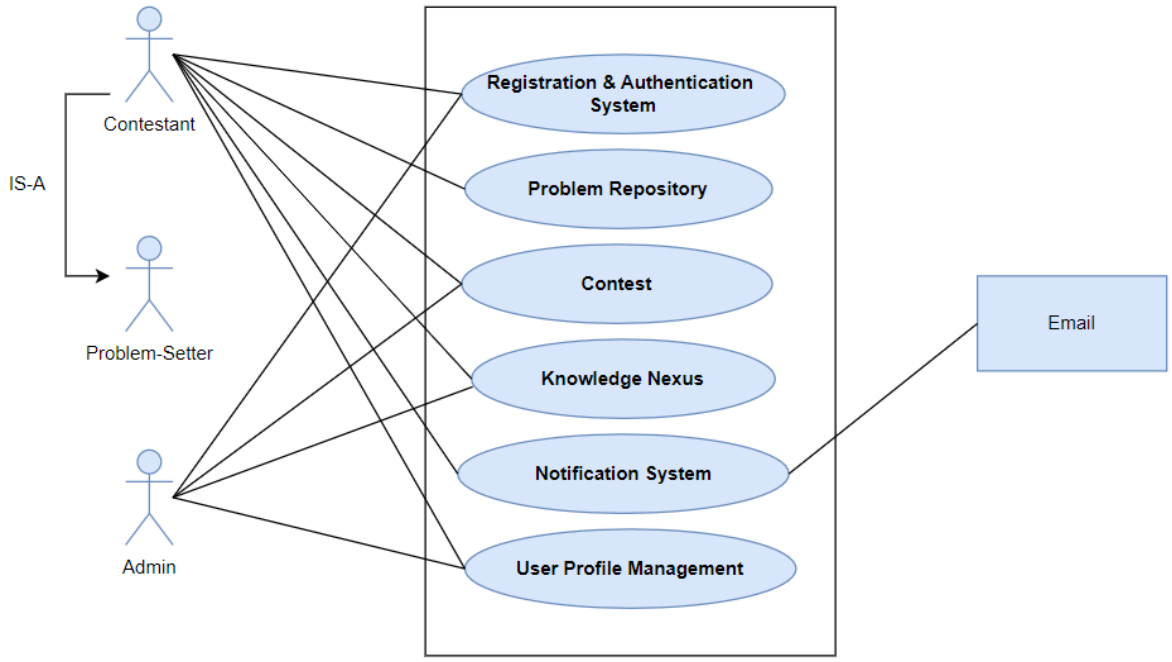


Figure 4: Use case diagram level 1: CodeSphere (Detailed)

Description of Use Case Diagram level 1:

1. **Registration & Authentication System:** New users register by providing username, email, and password. The system checks username uniqueness and email format. After validation, a verification link is sent to the email. Users confirm via the link and proceed to login. Registered users access their account dashboard by entering username/email and password. The system verifies credentials for login.

2. **Problem Repository:** The Problem Repository stores past contest problems, enabling contestants to access and practice them along with their solutions. This allows contestants to hone their problem-solving skills and prepare effectively for future competitions.

3. **Contest:** Contestants participate in upcoming contests by registering and submitting solutions. Problem-setters submit new problems for verification, and organizers host contests by creating custom ones from the platform's problem bank.

4. **Knowledge Nexus:** Users contribute insightful blog posts in the Knowledge Nexus section, covering programming, algorithms, data structures, and problem-solving with code snippets and explanations. Other users can view, react, and comment on these blogs. Additionally, the system generates control flow graphs for test cases, aiding users in understanding code flows to handle cases effectively enhancing problem-solving strategies.

5. **Notification System:** The platform sends email notifications to contestants for upcoming contests, results, and announcements, using their provided email addresses as the primary channel of communication. Contestants are promptly informed about contest details such as start times, duration, problem sets, individual standings, and rankings. Additionally, they receive updates on platform enhancements, features, and community events. Users have the option to customize their notification preferences, selecting the types of notifications they desire and choosing the frequency of updates, thereby ensuring they receive relevant information tailored to their preferences and enhancing their overall platform experience.

6. **User Profile Management:** User profile management module contains these submodules: Profile overview, Settings, Team, Problem setting history, Admin dashboard.

Level - 1.1

Use Case Name: Registration and Authentication System
Primary Actors: Admin, Contestant
Secondary Actors: Email

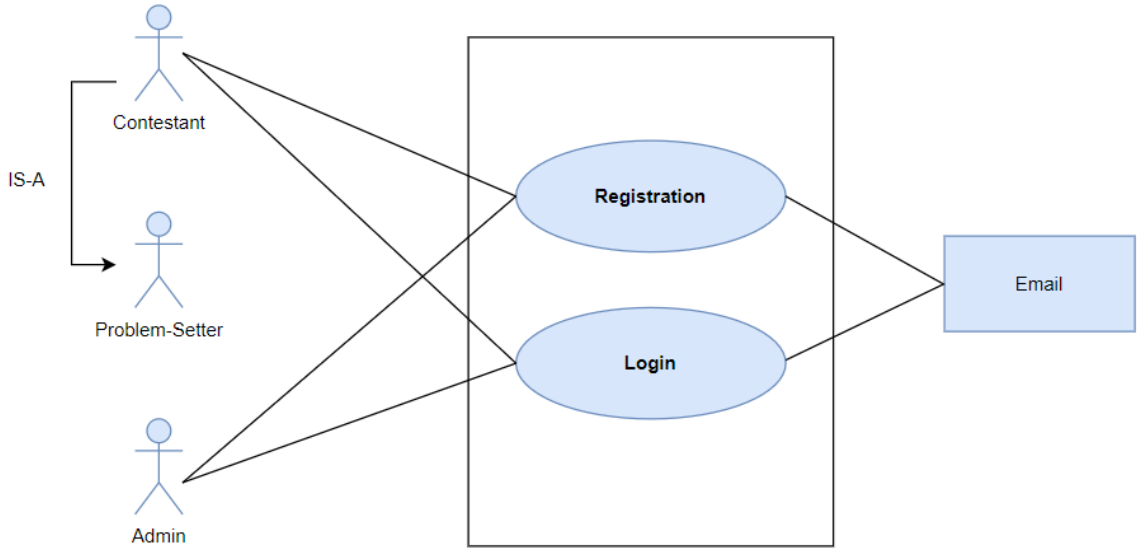


Figure 5: Use case diagram level 1.1: Registration and Authentication System

Description of Use Case Diagram level 1.1:

1. **Registration:** New users are presented with two options: admin, contestant. If choosing admin, a pincode is required for further processing. For contestants, register with a valid email and password. Problem-setter is a contestant. Upon successful validation, an email verification link is sent to the provided email address, allowing users to confirm their email by clicking the link and being redirected to the login page.

2. **Login:** Registered users log in by entering their username or email and password on the login page. The system verifies this information against

stored data. Upon successful validation, users gain access to their respective profile.

Action-Replay:

Action: New users initiate the registration process by selecting one of two options: admin, contestant.

Reply: System stores response.

Action: If selecting admin, the system prompts for a pincode for further processing. Otherwise, by providing valid email and password, a user can register as a contestant.

Reply: Upon successful validation, an email verification link is sent to the provided email address, enabling users to confirm their email by clicking the link and being redirected to the login page.

Action: Registered users access the login page and enter their username or email along with the password.

Reply: The system verifies the entered information against stored data and grants access upon successful validation directing users to their respective profile.

Level - 1.2

Use Case Name: Contest
Primary Actors: Admin, Contestant

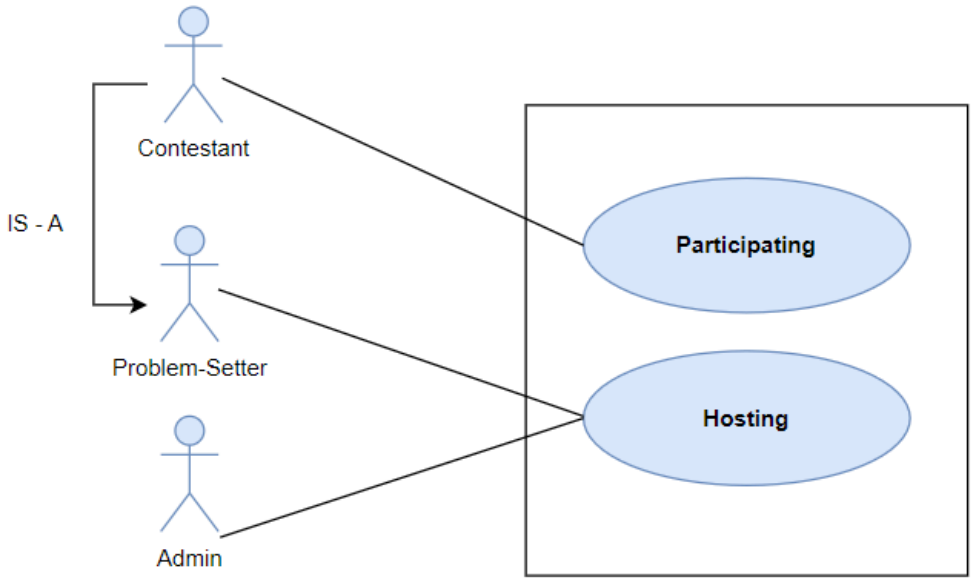


Figure 6: Use case diagram level 1.2: Contest

Description of Use Case Diagram level 1.2:

1. **Participating:** Contestants navigate to the Contest section, registering individually or as a group. After registration, they wait for the contest to start and access the interface to submit solutions during the contest. The leaderboard updates in real-time, reflecting contestant standings based on submissions and penalties.

2. **Problem-Setting:** Problem-Setters access the Problem Setting sub-module within the Contest module, where they can submit new problems with detailed statements, input/output specifications, constraints, and sample test cases. Proposed solutions along with test cases are also required for validation of correctness and efficiency.

3. **Hosting:** The contest hosting module facilitates the seamless organization and execution of contests following the approval of custom problems. This module encompasses essential steps such as setting the contest duration, configuring problem submission and evaluation settings, and ensuring all necessary setup is complete. After submission, the contest undergoes verification by platform administrators. Once the technical aspects are finalized and thoroughly tested, the module facilitates the dissemination of contest details to potential participants through various channels, primarily email. Information provided includes the contest format, rules, registration process, and any other relevant details essential for participants to engage effectively.

Action-Replay:

Action: Contestant accesses the "Contest" section.

Reply: System displays a list of upcoming and ongoing contests.

Action: Contestant chooses to participate in a contest.

Reply: System presents options for registration:

- Individual registration
- Team registration (if applicable)

Action: Contestant selects "Individual" and clicks in registration.

Reply: System confirms registration.

Action: Contestant selects "Team" and creates a new team or joins an existing one.

Reply: System allows creating a new team with a name and inviting members or selecting an existing team to join.

Action: Registered contestant waits for the contest to begin.

Reply: System displays a countdown timer or notification regarding contest start time.

Action: Problem-Setter accesses the "Contest" module and selects "Problem Setting".

Reply: System displays the Problem Setting sub-module.

Action: Problem-Setter creates a new problem.

Reply: System provides a form for entering problem details:

- Problem statement (clear and concise description of the problem)
- Input/output specifications (defining input format and expected output)
- Constraints (limitations on input values or solution approach)
- Specific Programming Language
- Sample test cases (examples of input and corresponding correct output)
- Proposed solution

Action: Problem-Setter select "Create Contest".

Reply: System provides a form for defining contest details:

- Selecting approved problems
- Setting contest duration (start and end time)
- Configuring problem submission and evaluation settings

Action: Problem-Setter submits the contest details.

Reply: System receives the contest details and sends it for admin verification.

Action: Admins receive notification of a new contest submission.

Reply: System displays the proposed contest for review within the admin interface.

Action (Approved): Admins approve the contest.

Reply: System notifies the problem-setter of the approval.

Action (Rejected): Admins reject the contest and provide feedback for improvement.

Reply: System notifies the problem-setter of the rejection with specific reasons and suggestions for revision.

Action: Contest begins.

Reply: System provides access to the contest interface for submitting solutions.
Action: Contestant attempts to solve problems and submits solutions.
Reply: System receives and processes the submitted solutions.
Action: Solutions are evaluated automatically.
Reply: Leaderboard updates in real-time, reflecting contestant standings.
Action: Problem-Setter launches contest.
Reply: System facilitates sending contest notifications through various channels.

Level - 1.2.1

Use Case Name: Hosting
Primary Actors: Admin, Problem-Setter

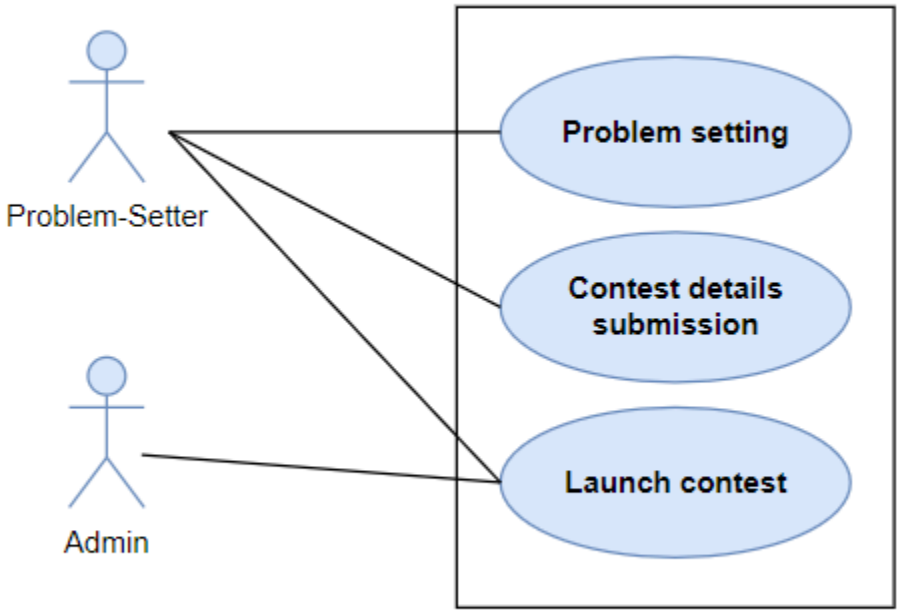


Figure 7: Use case diagram level 1.2.1: Hosting

Level - 1.3

Use Case Name: Knowledge Nexus
Primary Actors: Contestant

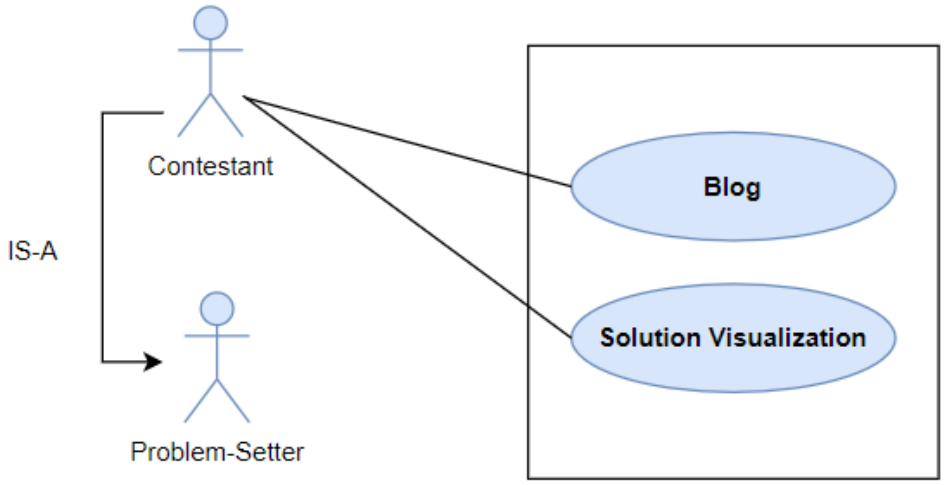


Figure 7: Use case diagram level 1.3: Knowledge Nexus

Description of Use Case Diagram level 1.3:

1. **Blog:** Users from various roles contribute blog posts on programming, algorithms, and problem-solving in the Learning section of the platform. Utilizing user-friendly tools, they craft their content with care and precision. Before publication, a prompt appears, ensuring compliance with the platform's privacy policy. Users are asked if their post violates any privacy

policies. If they choose 'no,' the blog is then published for all users to access and engage with.

2. **Solution Visualization:** The system generates control flow graphs for test cases which aid contestants in understanding the flow of code to handle cases effectively, thereby enhancing problem-solving strategies. Contestants can visualize the flow of code through corresponding test cases, gaining insights into the execution paths and facilitating comprehensive analysis of the code's behavior.

Action-Replay:

Action: User chooses to contribute a blog post.

Reply: System presents the content creation interface with user-friendly tools (text editor, code snippet inserter, formatting options).

Action: User selects the topic category (programming, algorithms, or problem-solving).

Reply: System filters available options within the chosen category for further content selection.

Action: User submits the completed blog post.

Reply: System checks if their post violates any privacy policies or not.

Action: User selects options that his/her post doesn’t violate any privacy policies.

Reply: System publishes the post in the Learning section, notifying the user and making it accessible to all users.

Action: User finds a published blog post of interest.

Reply: System presents the entire post with formatting, images/videos, and hyperlinks as intended by the author.

Action: User reacts to the post with pre-defined emojis (like, dislike, etc.).

Reply: System updates the post's reaction count based on the user's chosen emoji.

Action: User composes a comment on the post.

Reply: System stores the user's comment and displays it alongside the post for other users to see.

Action: Contestant requests to view the control flow graph for the test case.

Reply: The system generates the control flow graph for the test case.

Level - 1.4

Use Case Name: User Profile Management
Primary Actors: Admin, Contestant

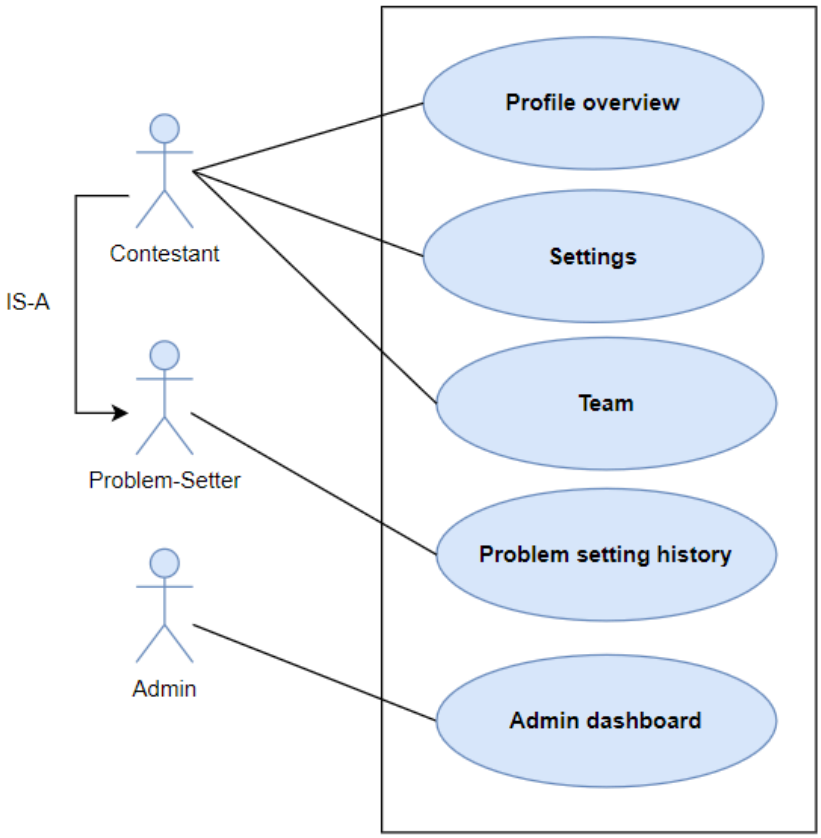


Figure 8: Use case diagram level 1.4: User Profile Management

Description of Use Case Diagram level 1.4:

1. **Profile Overview:** Upon visiting the profile page, contestants are presented with a comprehensive overview of their account information, encompassing various aspects of their engagement on the platform. This consolidated module includes three key submodules:
    - Personal information
    - Activity history
    - Achievements and Badges

2. **Team:** The "Team" submodule within the user profile provides

    - Displays number of teams user is associated with
    - Lists of each team and its members
    - Displays team submissions
    - List of contests participated in as a team
    - List of contests organized as a team

3. **Setting**: Registered users can update their account information by logging in and accessing the profile settings. Here, they can modify details like email address, password, and profile information. After making changes, users submit the updated details for system validation. Upon successful validation, the system updates the user's profile to maintain accurate information.

4. **Problem setting history:** This submodule includes -

    - Displays timestamps of problems set
    - Shows status (approved/pending) of each problem
    - Includes titles or brief descriptions of authored problems
    - Offers insights into contestant engagement

- Indicates number of attempts or solutions per problem

5. **Admin Profile:** Admins access the Admin Dashboard from the platform's navigation menu, overseeing user activity, content moderation and platform performance. They manage user accounts including viewing profiles, resetting passwords and suspending or banning accounts as needed. Admins review reported content, moderate discussions and enforce community guidelines while also customizing platform settings, managing email notifications and configuring administrative preferences.

Action-Replay:

Action: User visits their profile page.

Reply: System displays a comprehensive overview with three submodules:Personal information, Activity history, Achievements and Badges.

Action: User navigates to the "Team" submodule.

Reply: System displays information about the user's teams.

Action: User logs in and accesses profile settings.

Reply: System presents options for updating account information.

Action: User edits information and submits changes.

Reply: System validates the updated details.

Action: Problem-Setter accesses "Problem Setting History".

Reply: System displays a breakdown of the user's problem-setting contributions.

Action: Admin accesses the Admin Dashboard from the navigation menu.

Reply: System displays a comprehensive dashboard for admin.

Level - 1.4.1

Use Case Name: Profile overview
Primary Actors: Contestant

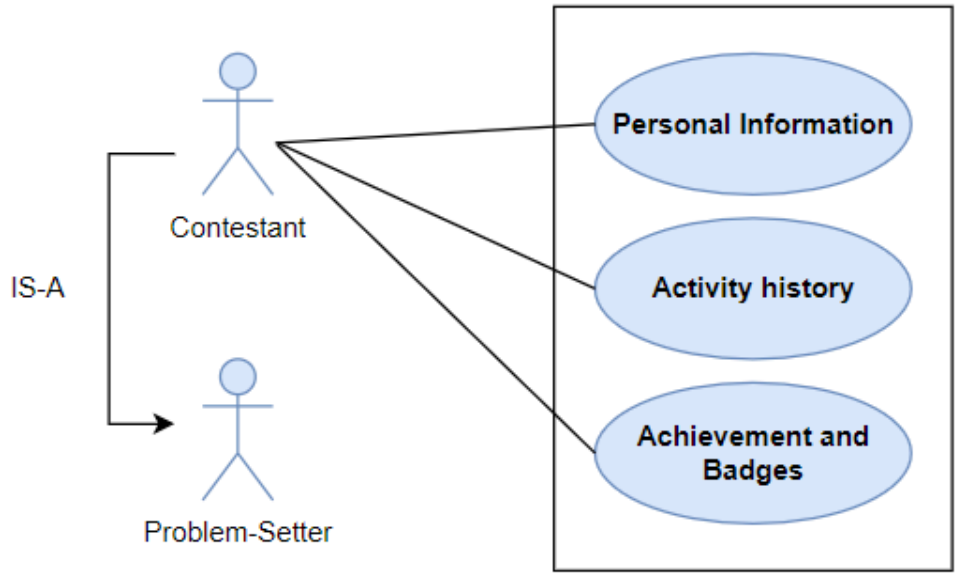


Figure 9: Use case diagram level 1.4.1: Profile overview

Description of Use Case Diagram level 1.4.1:

1. **Personal information:** This submodule includes username, profile picture, a brief bio, university name, batch, email address, friends, Last seen.

2. **Activity history:** Users can view their activity history including a log of their recent interactions on the platform such as Submission history including successful solutions and compilation errors, Total number of problems solved, Contest participation, Total number of blog publications.

3. **Achievements and Badges:** Users can see their achievements and earn badges on the platform. Badges may be awarded for various accomplishments such as solving a certain number of problems, participating in contests, contributing to the community or achieving specific rankings.

Action-Replay:

Action: User visits their profile page.

Reply: System displays the "Personal Information" submodule with details.

Action: User navigates to the "Activity History" submodule.

Reply: System displays a record of user interactions on the platform.

Action: User accesses the "Achievements and Badges" submodule.

Reply: System displays a breakdown of the user's accomplishments.

Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration, and concurrency.

Activity diagram ID: 01

Level 1.1
Name: Registration and Authentication System
Reference: Use case level 1.1 (Figure : use case diagram level 1.1 Registration and Authentication System)

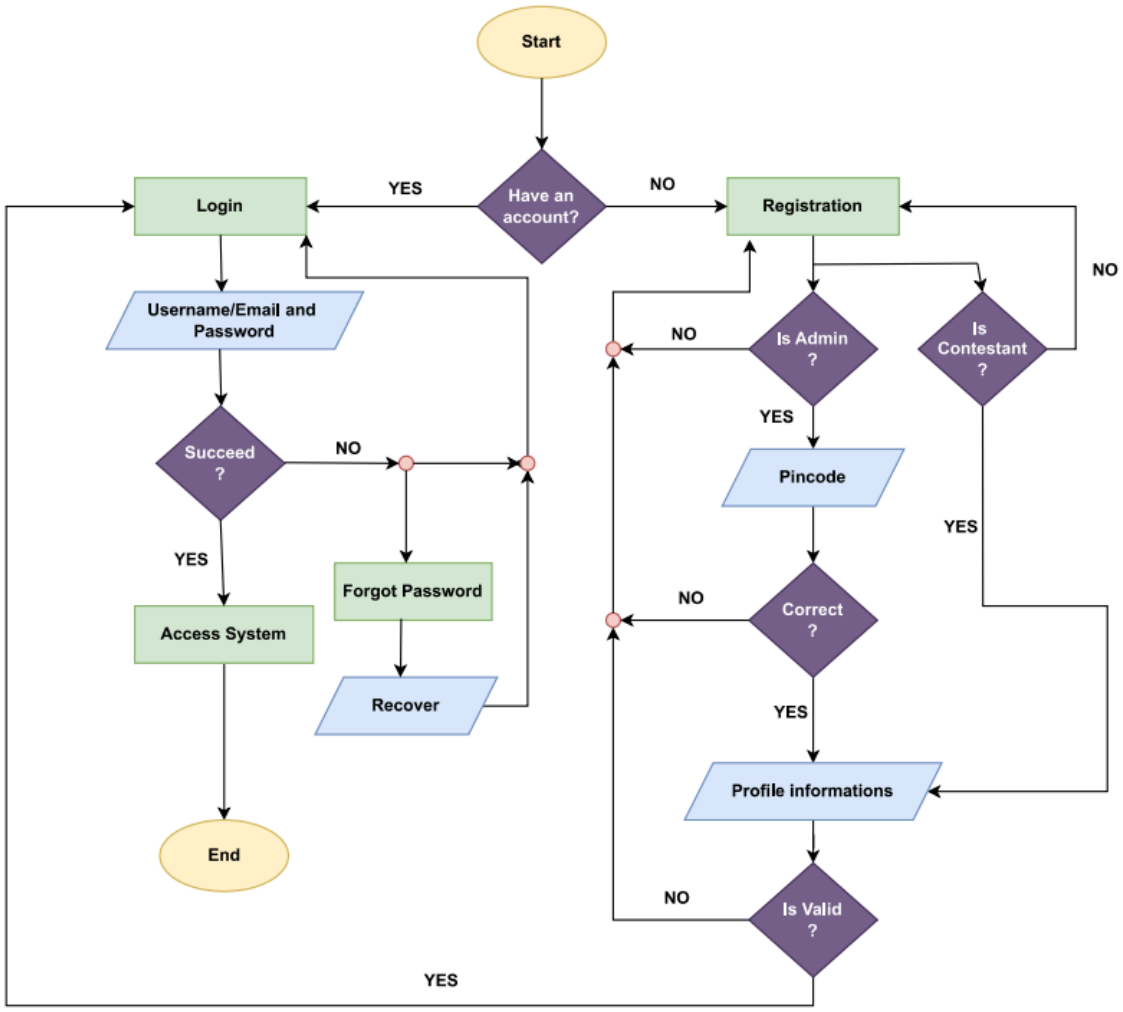


Figure 11: Activity diagram level 1.1: Registration and Authentication System

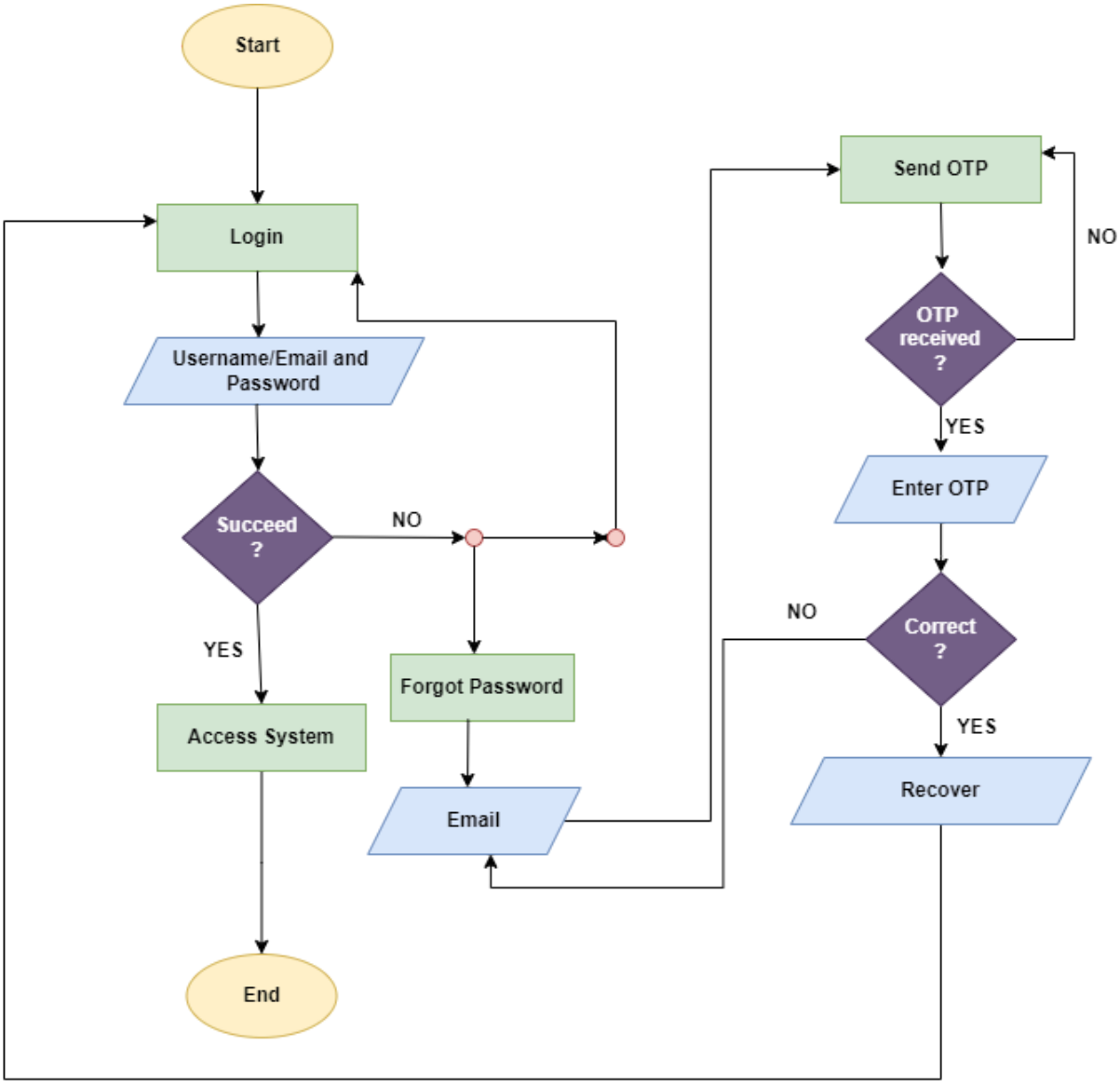


Figure 11: Activity diagram level 1.1.2: Login

Activity diagram ID: 02

Level 1.2
Name: Contest
Reference: Use case level 1.2 (Figure : use case diagram level 1.2 Contest)

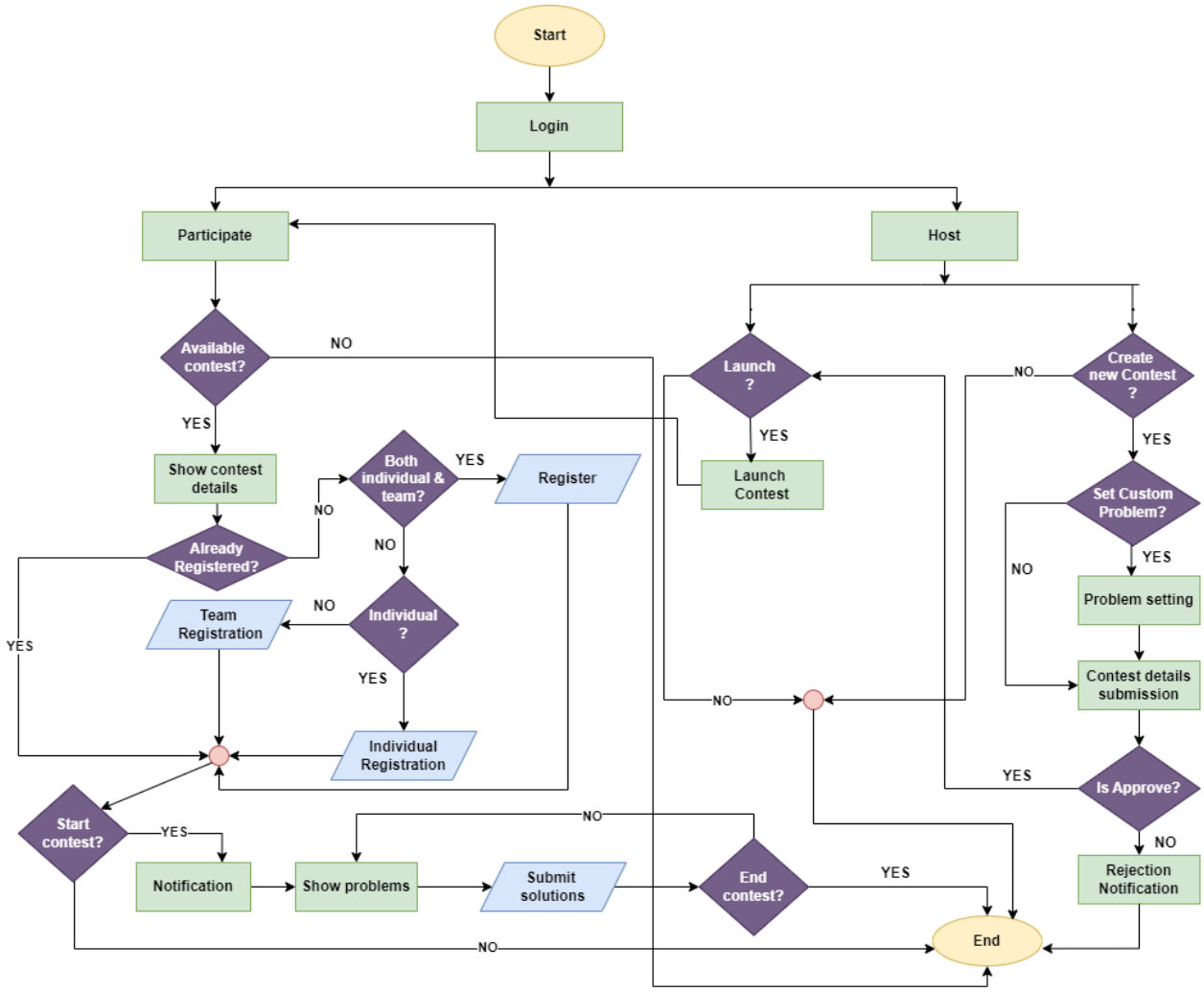


Figure 12: Activity diagram level 1.2: Contest

Activity diagram ID: 03

Level 1.3
Name: Knowledge Nexus
Reference: Use case level 1.3 (Figure : use case diagram level 1.3 Knowledge Nexus)

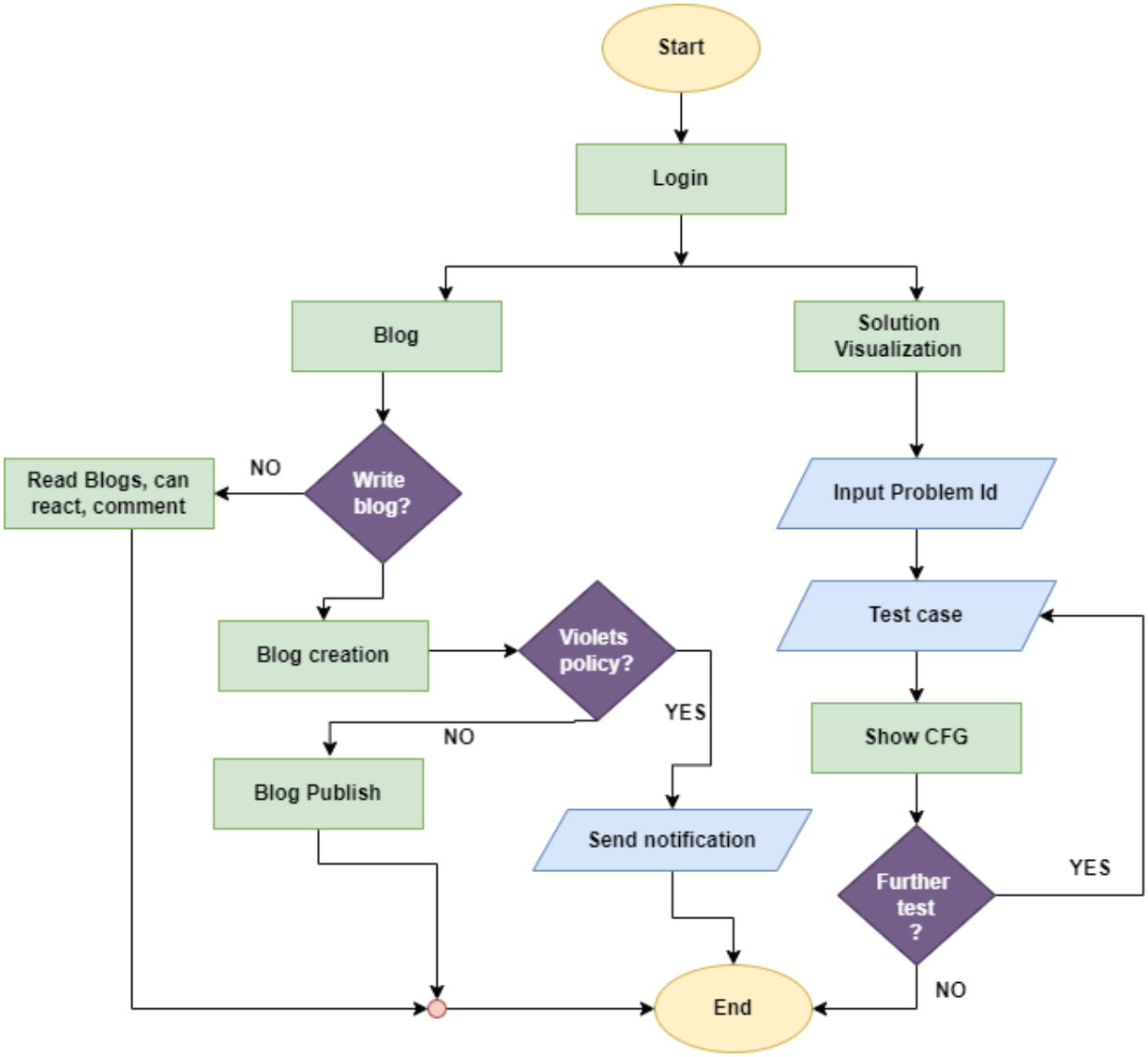


Figure 13: Activity diagram level 1.3: Knowledge Nexus

Activity diagram ID: 04

Level 1.4
Name: User Profile Management
Reference: Use case level 1.4 (Figure : use case diagram level 1.4 User Profile Management)

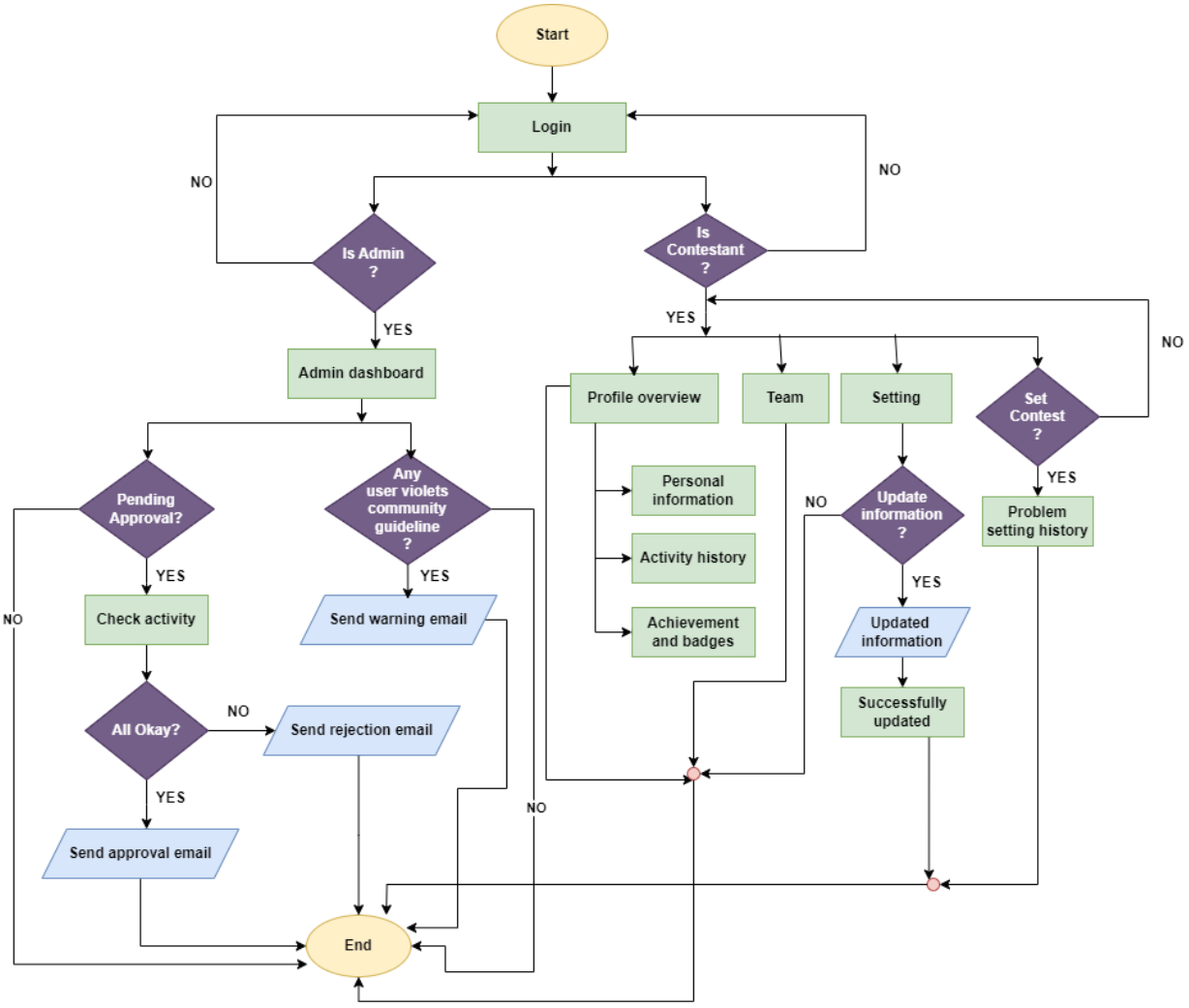


Figure 14: Activity diagram level 1.4: User Profile Management

Activity diagram ID: 5

Level 1.4.1
Name: Setting (Update information)

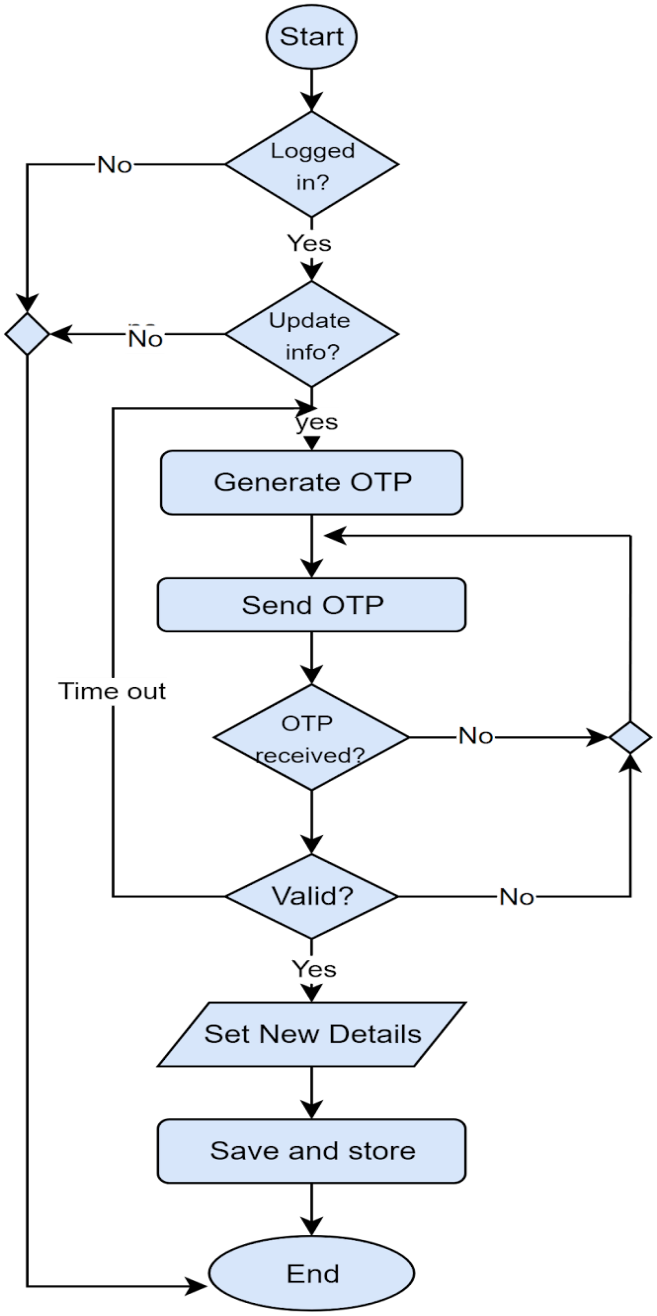


Figure 14: Activity diagram level 1.4.1: Setting

Swim lane diagram

A swimlane diagram is a type of flowchart. Like a flowchart, it diagrams a process from start to finish, but it also divides these steps into categories to help distinguish which departments or employees are responsible for each set of actions. It is based on the analogy of lanes in a pool, as it places process steps within the horizontal or vertical “swimlanes” of a particular department, work group or employee, thus ensuring clarity and accountability.

Swim lane Diagram ID: 01

Level 1.1
Name : Registration and Authentication System
Reference: Activity diagram ID : 01 (Figure : Activity diagram level 1.1 Registration and Authentication System)

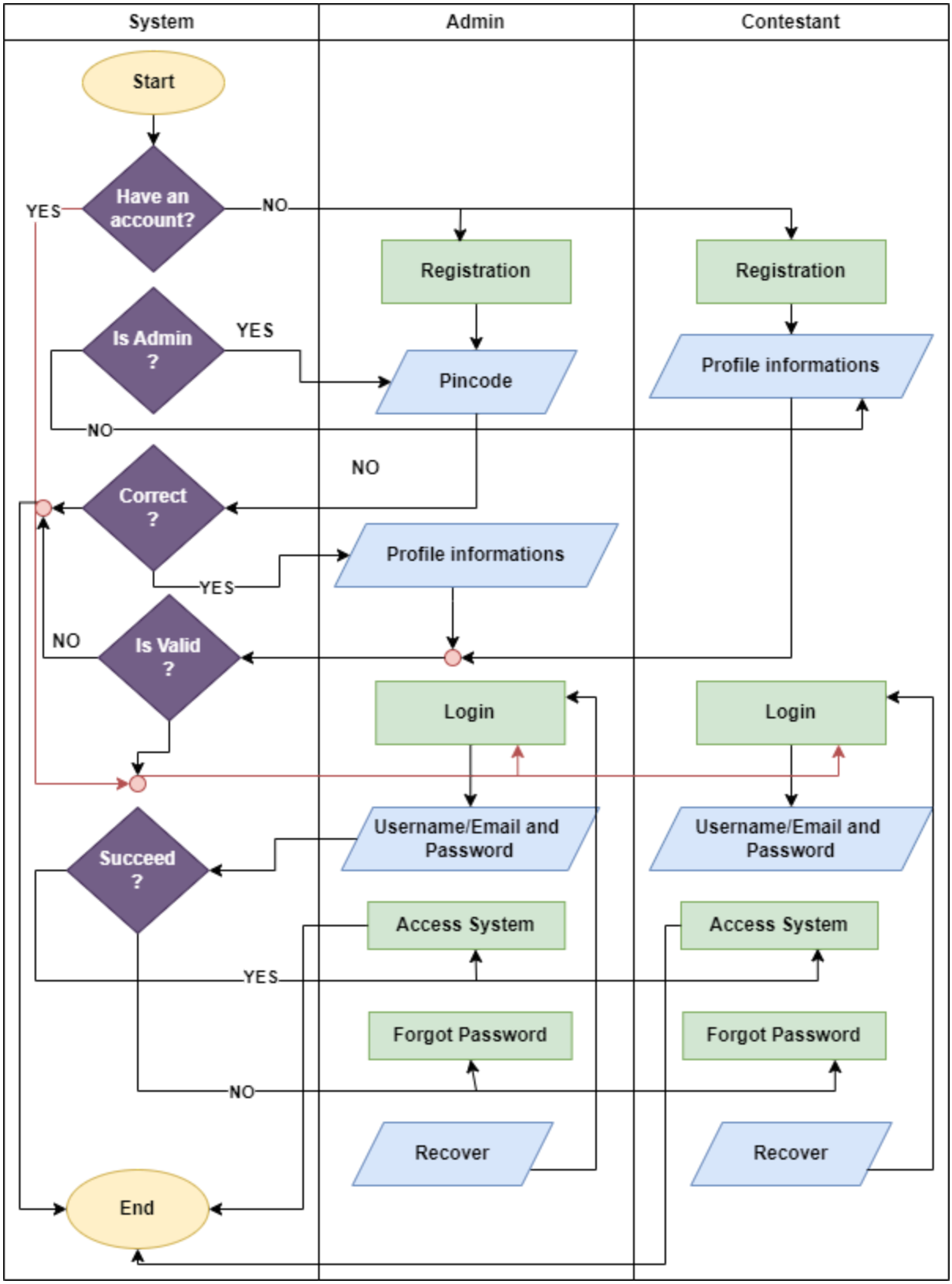


Figure: Swimlane diagram level 1.1: Registration and Authentication

Swim lane Diagram ID: 02

Level 1.2
Name : Contest
Reference: Activity diagram ID : 04 (Figure : Activity diagram level 1.2 Contest)

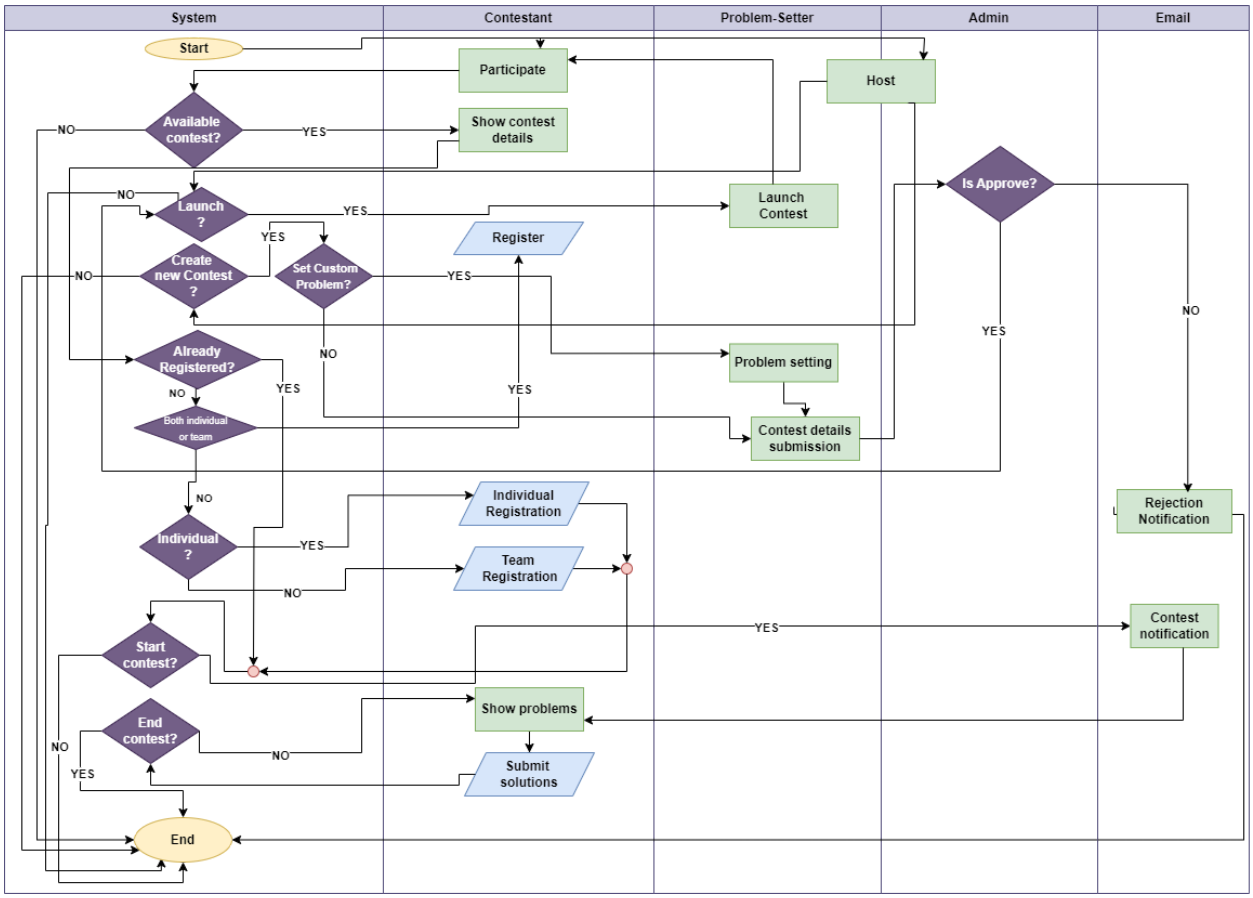


Figure: Swimlane diagram level 1.2: Contest

Swim lane Diagram ID: 03

Level 1.3
Name : Knowledge Nexus
Reference: Activity diagram ID : 05 (Figure : Activity diagram level 1.2 Knowledge Nexus)

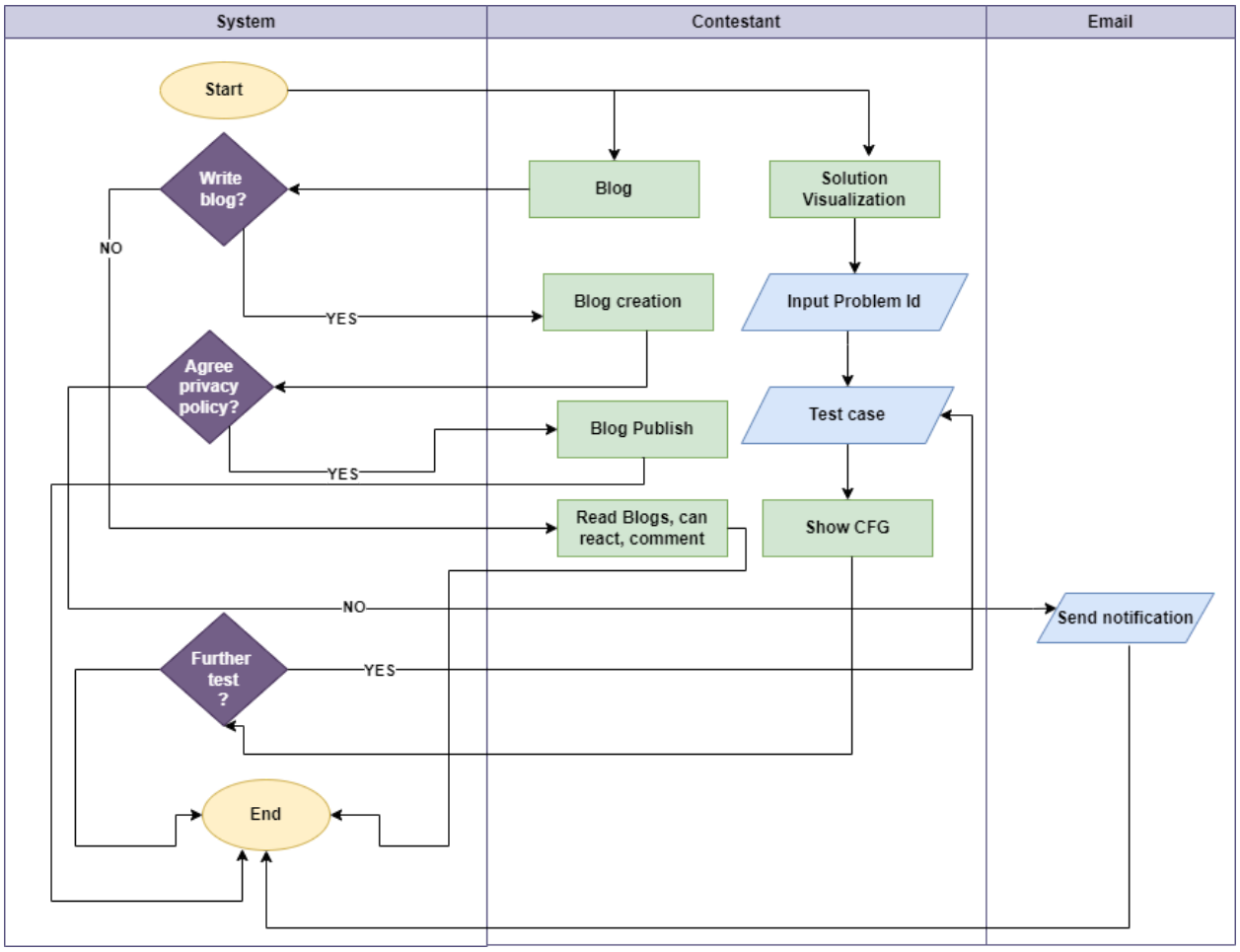


Figure: Swimlane diagram level 1.3: Knowledge Nexus

Swim lane Diagram ID: 04

Level 1.4

Name : User Profile Management
Reference: Activity diagram ID : 06 (Figure : Activity diagram level 1.4 User Profile Management)

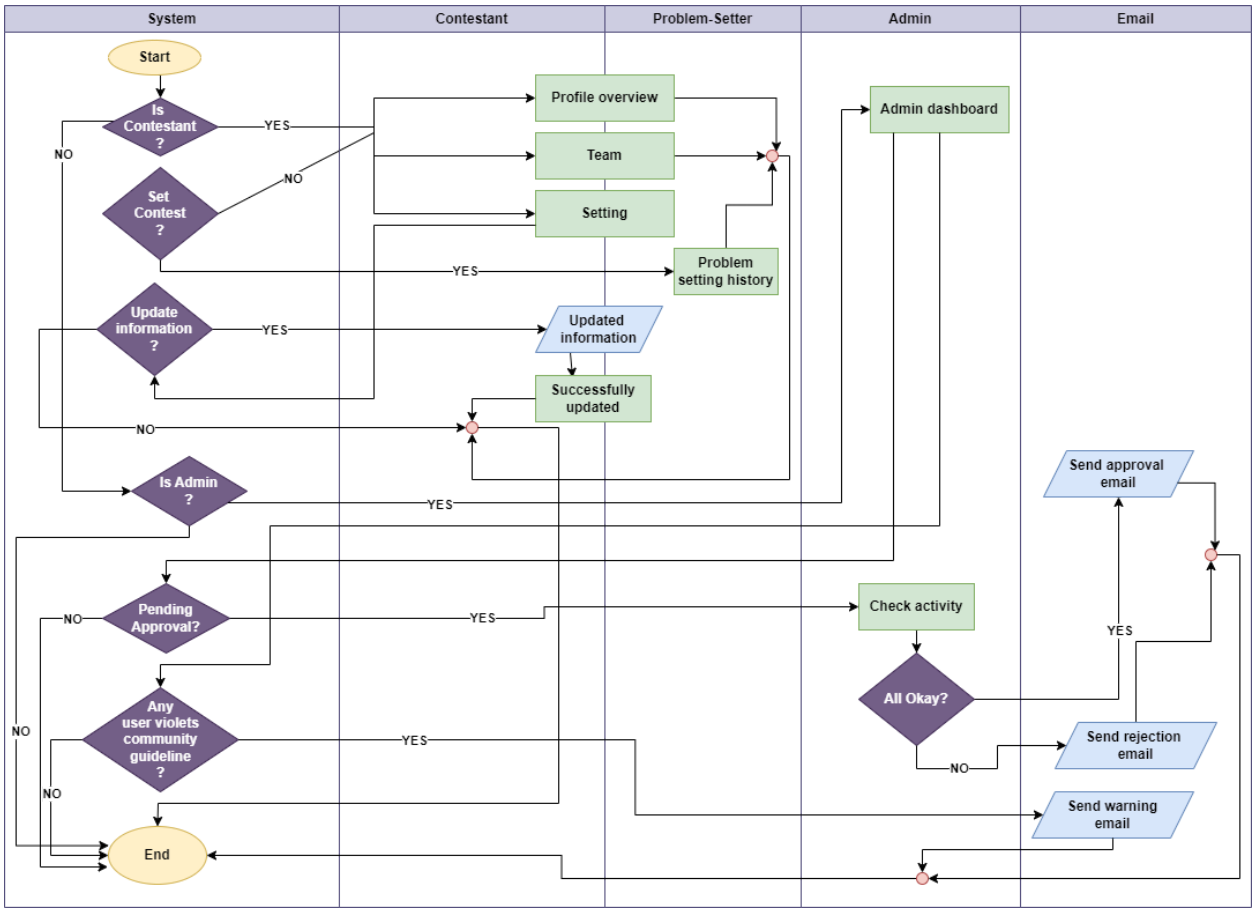


Figure: Swimlane diagram level 1.4: User Profile Management

5. Data based modeling

5.1 Noun Listing

SI	Nouns	Attribute	Problem/Solution space
1.	Registration		P
2.	Authentication		P
3.	System		S
4.	User	Username, Password, Email	S
5.	Account		S
6.	Information		P
7.	Pin Code		S
8.	Platform		P
9.	Page		P
10.	Essential		P
11.	Details		P
12.	Username		S
13.	Email		S
14.	Password		S
15.	Uniqueness		P
16.	Format		P
17.	Compliance		P
18.	Successful		P
19.	Validation		P

20.	Verification		P
21.	Link		P
22.	Provided		P
23.	Address		P
24.	Login page		S
25.	Registered		P
26.	Stored		P
27.	Data		P
28.	Notification		S
29.	Profile		P
30.	Settings		P
31.	Changes		P
32.	Updated		P
33.	Accurate		P
34.	Problem Bank		P
35.	Problem	Title, Test Case, Statement, Constraints, solution	S
36.	Contestant	Username, Password, Email	S
37.	Skills		P
38.	Central		P
39.	Hub		P
40.	Various		P
41.	Sets		P
42.	Interface		P
43.	Easy		P

44.	Navigation		P
45.	Sorting		P
46.	Filtering		P
47.	Options		P
48.	Sessions		P
49.	Criteria		P
50.	Difficulty		P
51.	Popularity		P
52.	Tags		P
53.	Selected		P
54.	Detailed		P
55.	Statement		P
56.	Specification		P
57.	Constraints		P
58.	Efficiently		P
59.	Contest	Title, Description, Start Time, End Time	S
60.	Past		P
61.	Setting		P
62.	Section		P
63.	Upcoming		P
64.	Individually		P
65.	Part		P
66.	Group		P
67.	Once		P

68.	Solutions		S
69.	Leaderboard		S
70.	Real-time		P
71.	Standings		S
72.	Submission		S
73.	Penalties		S
74.	Previous		P
75.	Experience		P
76.	Description		S
77.	Rankings		S
78.	Continuous		P
79.	Learning		P
80.	Development		P
81.	Programming		P
82.	Community		P
83.	Sub-module		P
84.	Module		P
85.	Input		P
86.	Output		P
87.	Sample		P
88.	Test Case		S
89.	Additionally		P
90.	Proposed		P
91.	Correctness		P

92.	Efficiency		P
93.	Submitted		P
94.	Process		P
95.	Admin	Username, Email, Password, Pin Code, Guidelines	S
96.	Organizers		S
97.	External		P
98.	Judges		P
99.	Custom		P
100.	List		P
101.	Available		P
102.	Selection		P
103.	Preferences		P
104.	Setup		P
105.	Accessible		P
106.	Learning		P
107.	Blog	Likes, Comments, Title, Content	S
108.	Visualization		S
109.	Problem Setter	Username, Password, Email	S
110.	Likes		S
111.	Comments		S
112.	Content		S
113.	Title		S
114.	Algorithms		S

115.	Data-Structures		S
116.	Expertize		P
117.	Content		S
118.	Code Snippet		S
119.	Explanation		S
120.	Visual		P
121.	Representation		P
122.	Decision-making		P
123.	Approach		P
124.	Improvements		P
125.			S
126.	Announcements		S
127.	Primary		P
128.	Communication		P
129.	Channel		P
130.	Results		S
131.	Features		S
132.	Events		S
133.	Needs		P
134.	Priorities		P
135.	Reminders		S
136.	Frequency		P
137.	Daily		P
138.	Digest		P

139.	Relevant		P
140.	Statistics		P
141.	Contribution Tracker		P
142.	Participation		P
143.	History		P
144.	Menu		P
145.	Overview		P
146.	Profile picture		S
147.	Bio		S
148.	Number		P
149.	Solved Problem		S
150.	Compilation Error		P
151.	Membership		P
152.	Last seen		S
153.	Timestamp		S
154.	Activity		S
155.	Personal		P
156.	Metrics		P
157.	Category		S
158.	Weakness		P
159.	Summary		P
160.	Forum Comments		P
161.	Feedback		S
162.	Chronological		P

163.	Name		P
164.	Publish Date		S
165.	Performance		P
166.	Final		P
167.	Points		P
168.	Moderation		P
169.	Guidelines		S
170.	Start Time		S
171.	End Time		S
172.	Admin Notification		S

5.2 Potential to be data object

SI	Data Object	Attribute
1.	User	Username, Password, Email
2.	Problem	Title, Test Case, Statement, Constraints, Solution
3.	Contestant	Username, Password, Email
4.	Contest	Title, Description, Start Time, End Time
5.	Admin	Username, Email, Password, Pin Code, Guidelines
6.	Blog	Likes, Comments, Title, Content
7.	Problem Setter	Username, Password, Email

Analysis:

The analysis of the final data objects reveals a streamlined approach to structuring the platform's database. By merging User with Contestant, due to their identical attributes, the system benefits from reduced complexity and increased efficiency. Each class is uniquely tailored to its function within the system, such as managing problems, contests, or blog content.

A primary key has been assigned to each data table, signified by attributes like uid, pid, cid, aid, bid, and sid for Contestant, Problem, Contest, Admin, Blog, and Solution respectively. These primary keys are crucial as they ensure the uniqueness of each record within its table, allowing for precise identification and retrieval of data.

Initially, the Solution was considered an attribute of the Problem object. However, to avoid the pitfalls of multi-valued attributes and to accommodate unique relationships that might arise, the Solution was allocated its own table. This decision not only simplifies the Problem object but also provides a dedicated space to manage the complexities and nuances of solutions—such as different contestant submissions and evaluation statuses.

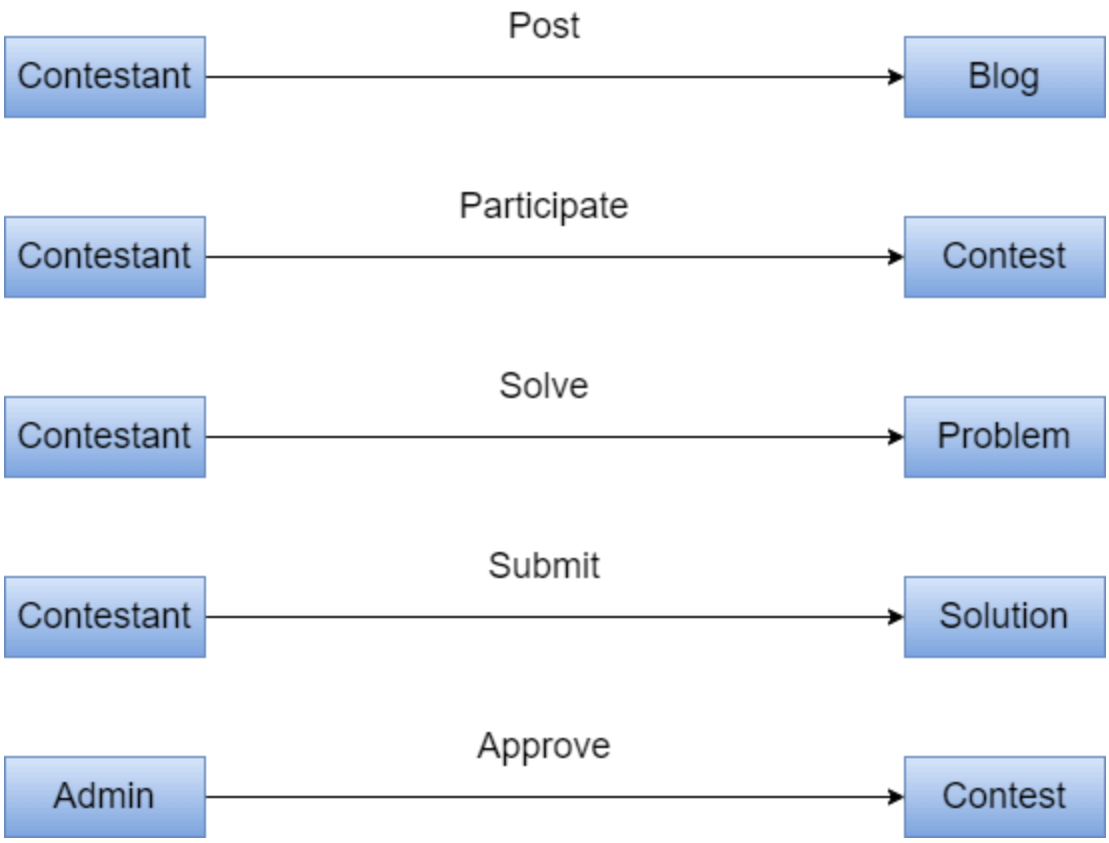
Moreover, the Solution table is characterized as a weak entity; it inherently relies on the Problem entity to have meaning and context. Without the associated problem, a solution lacks identity, because solutions are defined by the problem they aim to resolve. This dependency underlines the relational aspect of the database, ensuring solutions are directly linked to their respective problems, providing clear lineage and allowing for more intricate queries and data analysis.

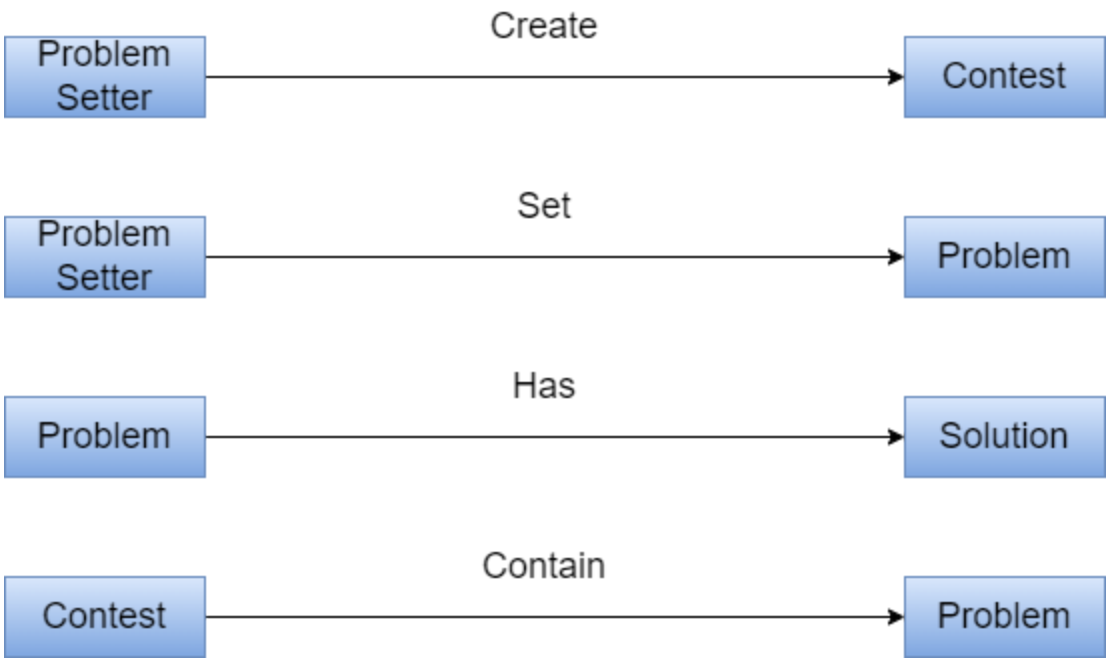
5.3 Final Data Object list

SI	Data Object	Attribute

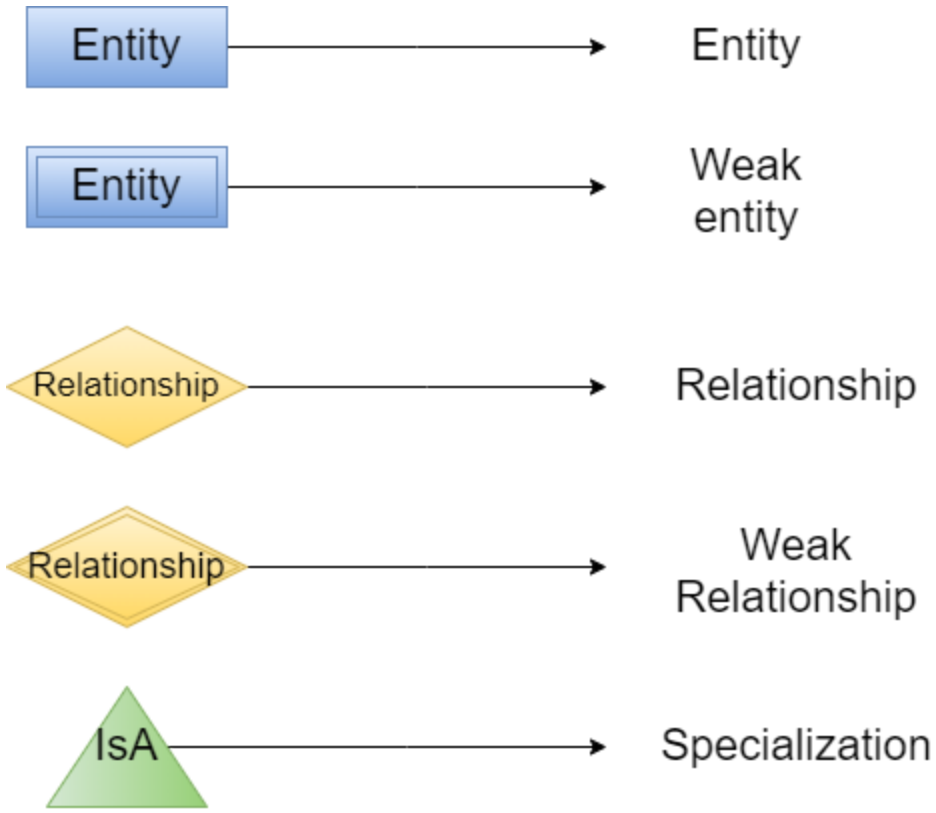
1.	Contestant	uid, username, password, email, registration date
2.	Problem	pid, title, test case, statement, constraints
3.	Contest	cid, title, description, start time, end time
4.	Admin	aid, username, email, password, pin code, guidelines
5.	Blog	bid, likes, comments, title, content
6.	Problem Setter	cid, username, password, email, registration date
7.	Solution	sid

5.4 Relations Between Data Objects


Post
Contestant
Blog
Participate
Contestant
Contest
Solve
Contestant
Problem
Submit
Contestant
Solution
Approve
Admin
Contest



Used ER Diagram symbols & meanings :



5.6 ER DIAGRAM


aid
email
pin code
username
password
guidelines
Admin
Create
Approve
comment count
title
like count
content
Blog
bid
ranking
Contest
start time
end time
cid
description
title
Participate
publish date
Post
Contain
email
username
uid
password
registration date
Contestant
Solve
Submit
IsA
constraint
statement
pid
title
test case
Problem
Has
Problem Setter
Set
Solution
sid

5.7 SCHEMA

Contestant	
Attributes	**Data Types**
uid	Integer
username	String
email	String
password	String
registration date	String

Admin	
Attributes	**Data Types**
aid	Integer
username	String
email	String
password	String
pin code	String

Problem	
Attributes	**Data Types**
pid	Integer
title	String
test case	String
statement	String
constraints	String

Contest	
Attributes	**Data Types**
cid	Integer
title	String
description	String
start time	String
end time	String

Blog	
Attributes	**Data Types**
bid	Integer
like	String
title	String
content	String
comment	String

Problem setter	
Attributes	**Data Types**
uid	Integer
username	String
email	String
password	String
registration date	String

Solution	
Attributes	**Data Types**
sid	Integer
pid	Integer

Post	
Attributes	**Data Types**
bid	Integer
uid	Integer
publish_date	String

Participate	
Attributes	**Data Types**
cid	Integer
uid	Integer
ranking	String

6. Class-based Modeling

The objects that the system will manipulate, the operations (also known as methods or services) applied to manipulate these objects, relationships (some hierarchical) between the objects, and interactions among the defined classes—all are represented by class-based modeling. Classes, objects, attributes, operations, Class-Responsibility-Collaborator (CRC) models, collaboration diagrams, and packages constitute components of a class-based model.

6.1 List of verbs

SI	**Verbs**	**SI**	**Verbs**
1.	Begin	2.	Navigate
3.	Visit	4.	Await

5.	Get	6.	Submit
7.	Choose	8.	Update
9.	Require	10.	Reflect
11.	Register	12.	Have
13.	Validate	14.	Propose
15.	Send	16.	Undergo
17.	Provide	18.	Conduct
19.	Confirm	20.	Host
21.	Click	22.	Organize
23.	Redirect	24.	Execute
25.	Enter	26.	Encompass
27.	Verify	28.	Set
29.	Gain	30.	Evaluate
31.	Access	32.	Ensure
33.	Serve	34.	Complete
35.	Store	36.	Finalize
37.	Revisit	38.	Test
39.	Engage	40.	Contribute
41.	Enable	42.	Create
43.	Reinforce	44.	Post
45.	Understand	46.	Publish

47.	Develop	48.	Ask
49.	Facilitate	50.	Violate
51.	Allow	52.	Generate
53.	Analyze	54.	Aid
55.	Employ	56.	Handle
57.	Tackle	58.	Visualize
59.	Receive	60.	Analyze
61.	Serve	62.	Send
63.	Schedule	64.	Customize
65.	Inform	66.	Suit
67.	Select	68.	Include
69.	Want	70.	View
71.	Present	72.	Earn
73.	Log in	74.	Award
75.	Modify	76.	Achieve
77.	Change	78.	Maintain
79.	Offer	80.	Involve
81.	Associate	82.	Collaborate
83.	Communicate	84.	Co-ordinate
85.	Approve	86.	Design
87.	Attempt	88.	Participate

89.	Moderate	90.	Manage
91.	Reset	92.	Suspend
93.	Ban	94.	Review
95.	Enforce	96.	Configure

6.2 General Classification

1. External entities (e.g., other systems, devices, people) that produce or consume information to be used
by a computer-based system.
2. Things (e.g., reports, displays, letters, signals) that are part of the information domain for the problem.
3. Occurrences or events (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation.
4. Roles (e.g., manager, engineer, salesperson) played by people who interact with the system.
5. Organizational units (e.g., division, group, team) that are relevant to an application.
6. Places (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system.
7. Structures (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects.

SI	Noun	G.C
1.	System	2, 7

2.	User	1,4,7
3.	Account	2
4.	Username	2
5.	Email	2
6.	Password	2
7.	Login page	2,
8.	Notification	2,3,7
9.	Problem	2,3,7
10.	Contestant	1,4,5
11.	Contest	2,3,7
12.	Solutions	2,3,7
13.	Leaderboard	2
14.	Standings	2
15.	Submission	2
16.	Penalties	2
17.	Description	2
18.	Rankings	2
19.	Test Case	2
20.	Admin	1,4,5,7
21.	Organizers	4
22.	Blog	2,3,7

23.	Visualization	2
24.	Problem Setter	1,4,5
25.	Likes	2
26.	Comments	2
27.	Content	2
28.	Title	2
29.	Algorithms	2
30.	Data-Structures	2
31.	Content	2
32.	Code Snippet	2
33.	Explanation	2
34.	Publish Date	2
35.	Start Time	2
36.	End Time	2
37.	Dashboard	2
38.	Announcements	2
39.	Results	2
40.	Features	2
41.	Events	2
42.	Reminders	2
43.	Profile picture	2

44.	Bio	2
45.	Solved Problem	2
46.	Last seen	2
47.	Category	2
48.	Timestamp	2
49.	Activity	2
50.	Feedback	2
51.	Admin Dashboard	2
52.	Database	2,5,7

6.3 Potential Class List

1. User
2. Admin
3. Contestant
4. Problem Setter
5. Notification
6. Contest
7. Problem
8. Solution
9. Dashboard
10. Blog
11. Database
12. Email

6.4 Selection Criteria

1. Retained information: The potential class will be useful during analysis only if information about it must be remembered so that the system can function.
2. Needed services: The potential class must have a set of identifiable operations that can change the value of its attributes in some way.
3. Multiple attributes: During requirement analysis, the focus should be on "major" information; a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity.
4. Common attributes: A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.
5. Common operations: A set of operations can be defined for the potential class and these operations apply to all instances of the class.
6. Essential requirements: External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as classes in the requirements model.

SI	Noun	S.C
1.	User	1, 2, 3, 4, 5, 6
2.	Admin	1, 2, 3, 4, 5, 6
3.	Contestant	1, 2, 3, 4, 5, 6
4.	Problem setter	1, 2, 3, 4, 5, 6
5.	Notification	1, 2, 5, 6
6.	Contest	1, 2, 3, 4, 5, 6
7.	Problem	1, 2, 3, 4, 5, 6
8.	Solution	1, 2, 3, 6
9.	Dashboard	1, 2, 6
10.	Blog	1, 2, 3, 4, 5, 6

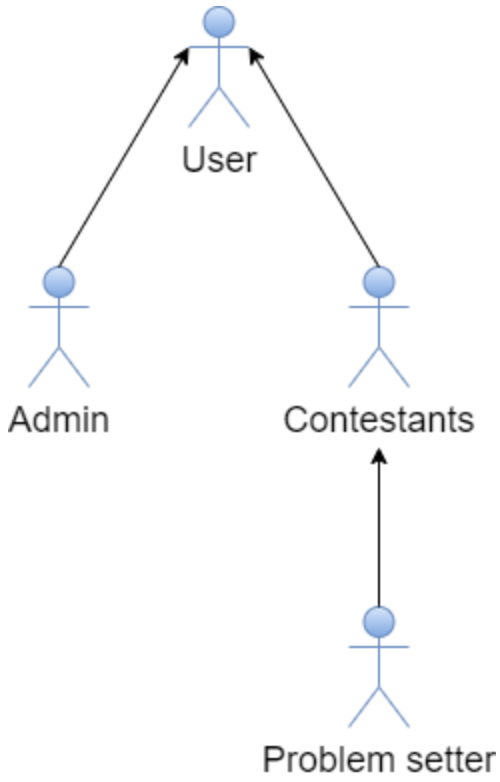
11.	Database	1, 2, 6
12.	Email	1, 2, 6

6.5 Selected Classes

1. User
2. Admin
3. Contestant
4. Problem Setter
5. Notification
6. Contest
7. Problem
8. Solution
9. Dashboard
10. Blog
11. Database
12. Email

Analysis:

The final class list optimizes the system's architecture through strategic merging and a clear inheritance hierarchy. By integrating Email and Dashboard into the Notification, the design centralizes user communications in a single interface, enhancing usability and streamlining backend logic. The inheritance model, with User serving as an abstract base class for Admin and Contestant, and further for Problem Setter inheriting from Contestant, efficiently encapsulates common attributes and functionalities while allowing for role-specific extensions.



This approach reduces redundancy, ensures code reusability, and supports a modular design that facilitates focused development and future scalability. Merging related functionalities and employing a thoughtful inheritance strategy reflects a deep understanding of object-oriented design principles, leading to a system that is both user-centric and maintainable, setting a strong foundation for a comprehensive application.

6.6 Finally Selected Classes

1. User
2. Admin
3. Contestant

4. Problem Setter
5. Notification
6. Contest
7. Problem
8. Solution
9. Blog
10. Database

6.7 Class Cards

1. User

User	
Attribute	**Method**
• user_id • user_name • email • password • registration_date • notification_list • blog_post_list • varification_code • problem_id • solution_id • solution_code	register(), login(), request_authenticate_user(), recover_password(), get_notified(), request_user_info(), modify_user_info(), receive_user_info(), access_general_features(), request_solution_visualization(), react_blog_post(), confirm_email_link(), receive_notification(), receive_new_blog_noti()
Responsibilities	**Collaborator Class**

Manage User Registration	Database
Manage User Login	Database
Password Recovery	Database
General accessibility and Notification	Notification, Database

2. Admin

Admin	
Attribute	**Method**
• pin_code • contest_id • problem_id • contest_scedule_list • approved_contest_list • approved_problem_list • community_guidelines • blog_post_list • removed_user_list • approval_status • requested_contest_list • requested_problem_list	enter_pin(), receive_contest_approval_request(), verify_contest_details(), send_approval_verdict(), schedule_contest(), approve_problem(), manage_user(), manage_content(), enforce_guidelines(), manage_user_account(), remove_blog_post(), validity_test()
Responsibilities	**Collaborator Class**
Admin registration	Database
Contest scheduling	Contest, Database

Manage entire platform operation	Database, Notification

3. Contestant

Contestant	
Attribute	**Method**
• achievements • contest_history • total_problem_solved • total_contests_participated • problem_list • written_blog_list • submitted_blog_list • Favourite_content_list • contest_type • team_list	register_for_contest(), participate_in_contest(), submit_solution(), request_solution_visualization() write_blog_post(), select_problem(), submit_blog_post(), solve_problem(), view_contest_history(), view_learning_materials(), access_contest_list(), update_info(), show_contest_list(), display_achievements(), display_participation_history(), calculate_achievements(), calculate_contest_history(), calculate_solved_problem(), calculate_attempted_problem(), receive_upcoming_contest_noti()
Responsibilities	**Collaborator Class**
Contest participation	Contest, Database

Problem solving	Problem, Database
Posting blog	Blog
Learning content	Blog, Solution
Information Updation	Database

4. Problem setter

Problem setter	
Attribute	**Method**
• contest_setting_list • custom_problem_list • approved_problem_list	set_custom_problem(), create_contest(), submit_test_cases(), custom_problem_solution(), send_contest_approval(), receive_requested_contest_verdict(),
Responsibilities	**Collaborator Class**
Contest creation	Contest
Problem setting	Problem, Database

5. Notification

Notification	
Attribute	**Method**
• notification_list • user_list • all_contest_list • upcoming_contest_list • approved_contest_list • blog_post_list	send_general_notification(), receive_general_notification(), receive_user_info(), send_contest_creation_request(), receive_contest_creation_request(), notify_upcoming_contest(), notify_contest_scheduling(), notify_contest_start(), notify_solution_verdict(), notify_contest_approval(), receive_post_status(), send_post_alert(), send_verification_email(), receive_register_request(), publish_blog_post(), generate_verification_code()
Responsibilities	**Collaborator Class**
User information reception	Database
Sending notification	User

6. Contest

Contest	
Attribute	**Method**
• cid • title • description • start_time • end_time • Duration • problem_list • Partipant_list • Ranking • contest_type • team_list	create_new_contest(), schedule_new_contest(), launch_contest(), check_for_approval(), add_participant(), add_problem(), remove_participant(), remove_problem(), calculate_result(), count_penalties(), get_approval_status(), provide_contest_list(), get_contest_ranking(), show_general_standing(), show_user_submission(), generate_team(), generate_team_id(), set_contest_type()
Responsibilities	**Collaborator Class**
Admin approval for scheduling	Notification
Scheduled contest organization	Notification, Database
Participant Management	Database
Providing real time standing	

7. Problem

Problem	
Attribute	**Method**
• pid • tile • statement • test_cases • constraints • solution_list_by_author • number_solved • author_id • contestant_id • difficulty_level • problem_tag	create_new_problem(), add_new_problem(), modify_existing_problem(), set_problem_statement(), set_constraints(), set_test_cases(), get_problem_statement(), get_constraints(), get_test_cases() define_problem_tag(), define_diffuculty_level(), validate_user_solution(), set_problem_solution(), update_problem_statement(), add_solver()
Responsibilities	**Collaborator Class**
Verifying solution	Solution

8. Solution

Solution	
Attribute	**Method**
• sid • problem_id	evaluate_solution(), visualize_solution(),

• contest_id • status • author_solution_list • contestant_solution_list	send_solution_verdict(), display_visualization(), create_cfg(), set_solution_verdict(), get_problem_solution(), get_contestant_solution(),
Responsibilities	**Collaborator Class**
Solution evaluation	Database, Notification
Visualizing solution	User

9. Blog

Blog	
Attribute	**Method**
• bid • like_count • comment_count • title • Content • author_id • publishing_date	create_new_blog(), set_title(), create_content(), edit_content(), delete_content(), publish_post(), set_accissible(), add_comment(), update_reaction_count(), display_blog_post(), get_title(), get_content(),

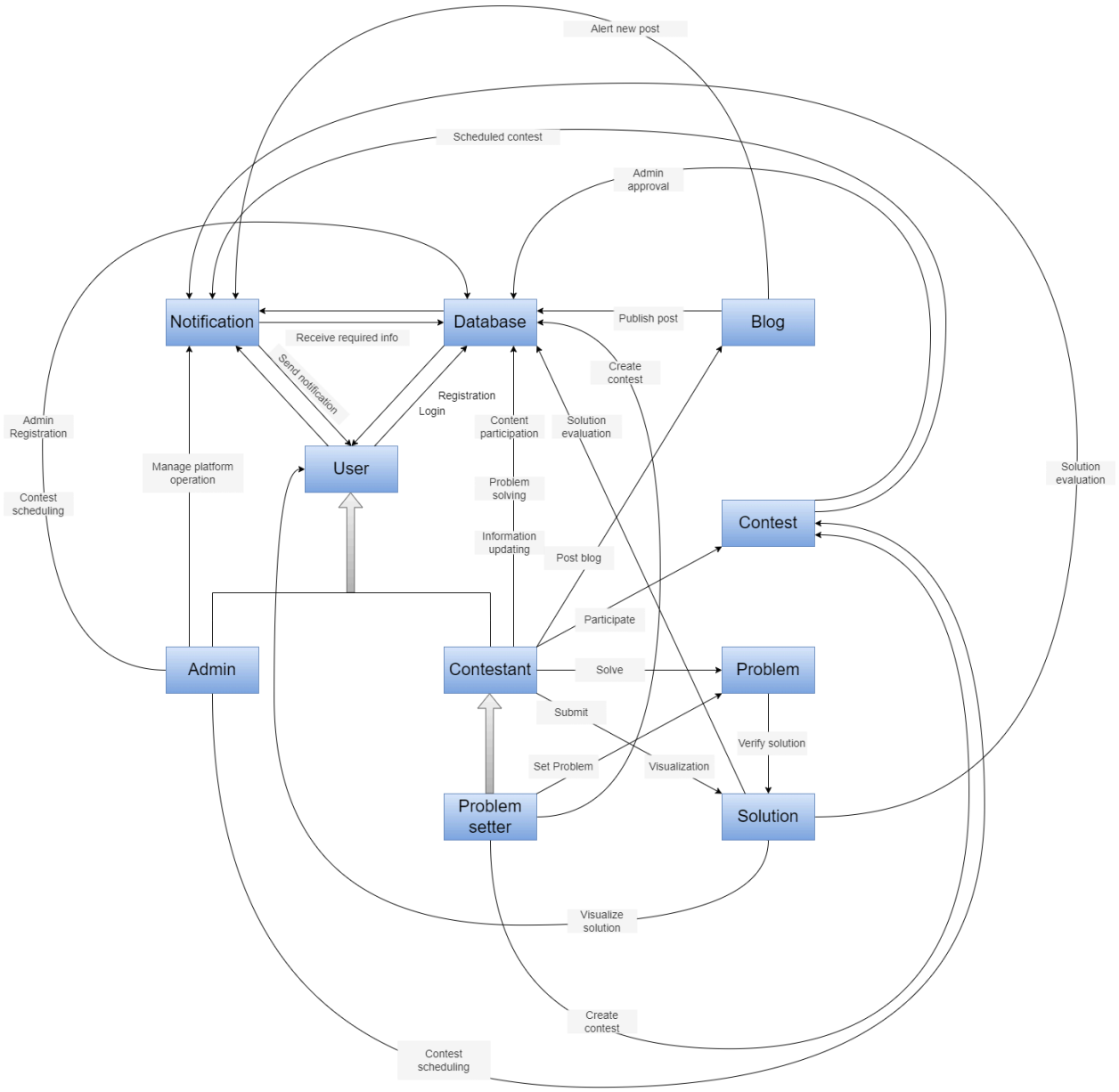
	calculate_comment_count(), calculate_reaction_count()
Responsibilities	**Collaborator Class**
Publishing blog post	Database
Managing blog post	Database
Alerting of new Post	Notification

10. Database

Database	
Attribute	**Method**
• user_list • admin_list • contestant_list • problem_setter_list • contest_participant_list • contest_list • problem_list • contest_ranking_list • blog_post_list • community_guidelines • system_operation_log	save_registered_user(), autheticate_user(), find_user_by_email(), verify_admin_by_pin(), save_scheduled_contest(), save_participant(), remove_participant(), save_blog_post(), update_blog_post(), delete_blog_post(), save_created_problem(), provide_user_info(), update_user_info(), store_info(), validate_info(), update_contest_ranking(), record_register_info(),

	grant_general_feature_access(), analyze_platform_operation(), preview_community_guideline(), update_solution_status()
Responsibilities	**Collaborator Class**
Manage connections	
Store information	User, Contestant, Problem setter, Blog, Admin, Contest, Problem, Solution, Notification
Connect with system database	
Provide required information	

6.8 CRC Diagram for all classes



7. Behavioral Modeling

The Behavior Modeling indicates how the system will behave to external events or stimuli. It is represented as a function of time and event, It describes interactions between objects. It shows how individual objects collaborate to achieve the behavior of the system as a whole. In UML behavior of a system is shown with the help of use case diagram, sequence diagram and activity diagram.
To create behavioral model following things can be considered-
● Evaluation of all use-cases to fully understand the sequence of interaction within the system.
● Identification of events that drive the interaction sequence and understand how these events relate to specific classes.
● Creating sequence for each use case.
● Building a state diagram for the system.
● Reviewing the behavioral model to verify accuracy and consistency

7.1 Event Identification

In the event identification table, events are mentioned in the leftmost column. The initiator class of the event and collaborator classes are mentioned in the following two columns and special cases for each event are mentioned in the rightmost column, if there are any such special cases.

SL	Initiator	Event Name	Collaborator	Related Method
1.	User	Initiate user registration	Database	register(), record_register_info()
2.	Admin	Prompt admin pincode	Notification	enter_pin()
3.	Database	Verify admin pincode	Notification	verify_admin_by_pin( )
4.	Notification	Send email verification link	User, Database	send_verification_ema il(), save_user()
5.	User	Login	Database	login()

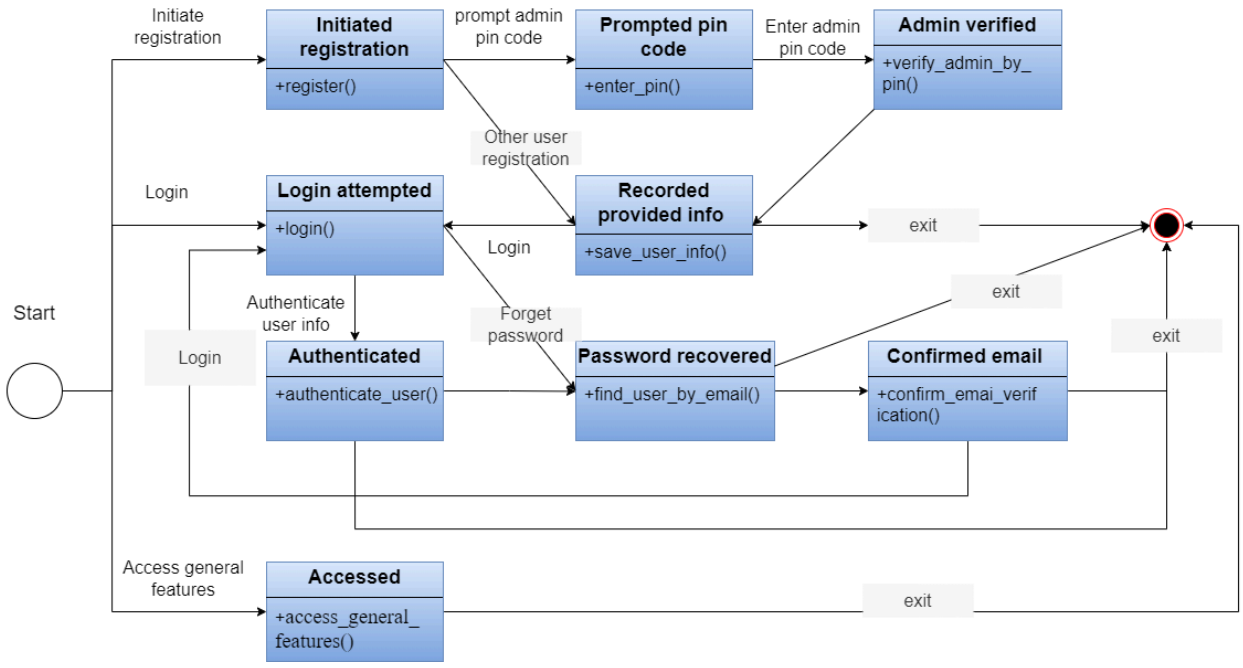
6.	Database	Authenticate user login	Notification	autheticate_user()
7.	User	Update account info	Notification, Database	receive_user_info(),update_user_info()
8.	User	Recover account password	Database	recover_password(), find_user_by_email()
9.	Contestant	Access contest list	Contest	access_contest_list(), provide_contest_list()
10.	Contestant	Confirm contest registration	Contest, Database	register_for_contest(), save_participant()
11.	Notification	Notify contest start	Contestant	notify_contest_start()
12.	Problem setter	Set custom problem	Problem	set_custom_problem()
13.	Problem setter		Contest, Database	submit_test_cases()
14.	Problem setter	Submit custom problem solution	Solution	custom_problem_solution()
15.	Problem setter	Create contest	Contest, Database	create_contest(), contest.check_for_approval()
16.	Notification	Notify contest approval	Database, Admin	notify_contest_approval(), contest_approval_request()
17.	Admin	Validate contest details	Database, Notification	create_new_contest(), schedule_contest(), verify_contest_details(), send_approval_verdict()
18.	Contest	Launch Contest	Notification	launch_contest()

19.	Contestant	Participate in contest	Database, Contest	participate_in_contest()
20.	Contestant	Solve problem	Problem	solve_problem()
21.	Contestant, problem	Submit solution	Solution, Database	submit_solution()
22.	Solution	Evaluate solution	Database, Notification	evaluate_solution(), send_solution_verdict()
23.	Contestant	Request solution visualization	Solution, Problem	request_solution_visualization()
24.	Solution	Visualize solution	Contestant	visualize_solution()
25.	Contest	Update contest ranking	Database	get_contest_ranking()
26.	Contestant	Contribute to blog Post	Blog, Database	write_blog_post()
27.				
28.	Notification	Publish Blog post	User	publish_blog_post()
29.	User	React to blog post	Blog	react_blog_post()
30.	Blog, Database	Update blog reaction count	Notification, User	update_reaction_count(),update_blog_post()
31.	Admin	Manages user account	Database	manage_user_account()
32.	Admin	Remove blog post	Blog, Database	remove_blog_post()
33.	Notification	Schedule contest reminder	Contestant	notify_upcoming_contest()
34.	Contestant	Select problem	Problem, Database	select_problem(), solve_problem()
35.	Contestant	Write blog	Blog	write_blog()
36.				

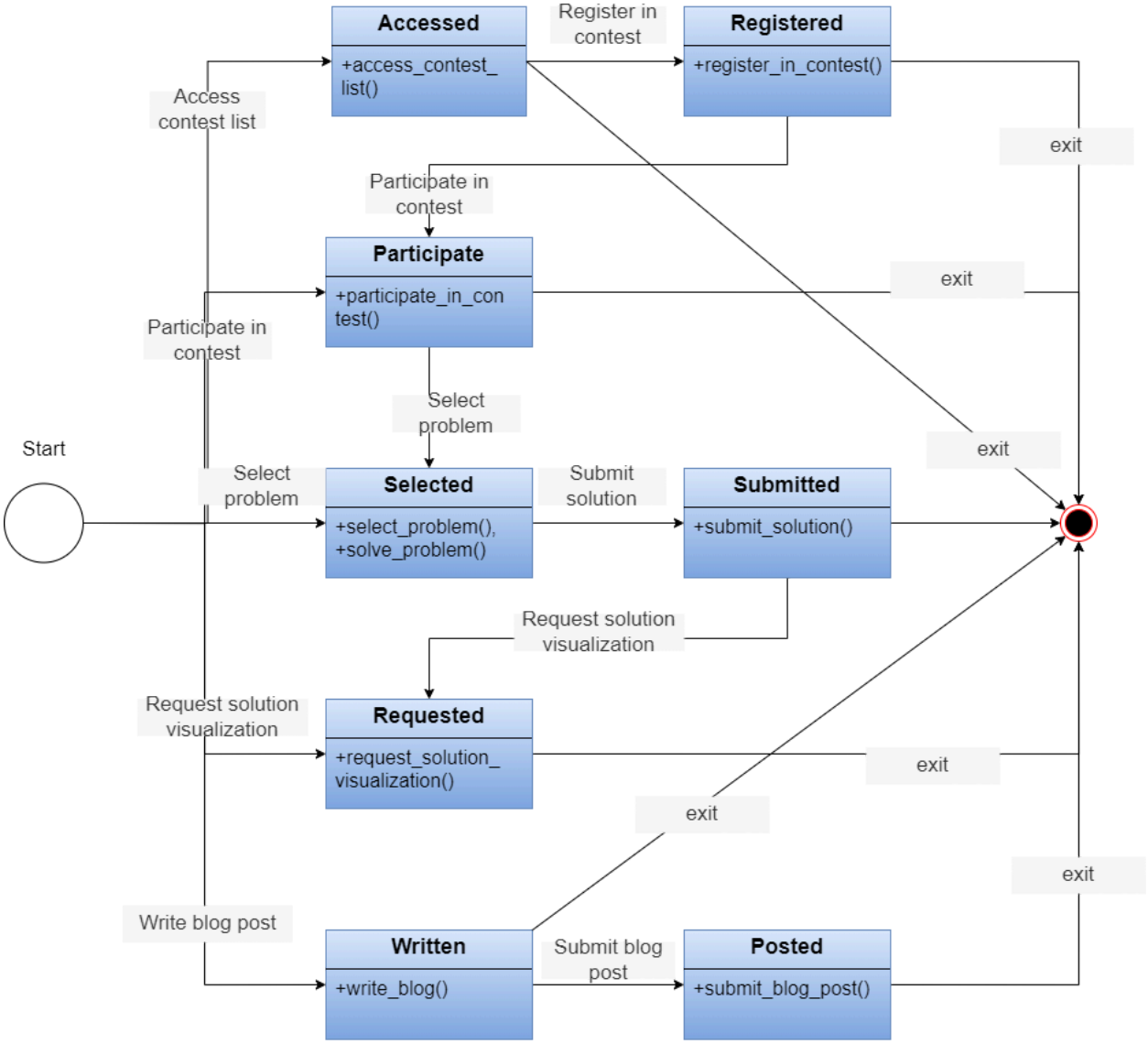
7. 2 State Transition diagrams

State Transition Diagram represents active states for each class of events (triggers). For this we identified all the events, their initiators and collaborators. In the State Transition Diagram the states are shown in boxed texts, and the transition is represented by arrows. It is also called State Chart or Graph. It is useful in identifying valid transitions.

ID: 01
Name: User

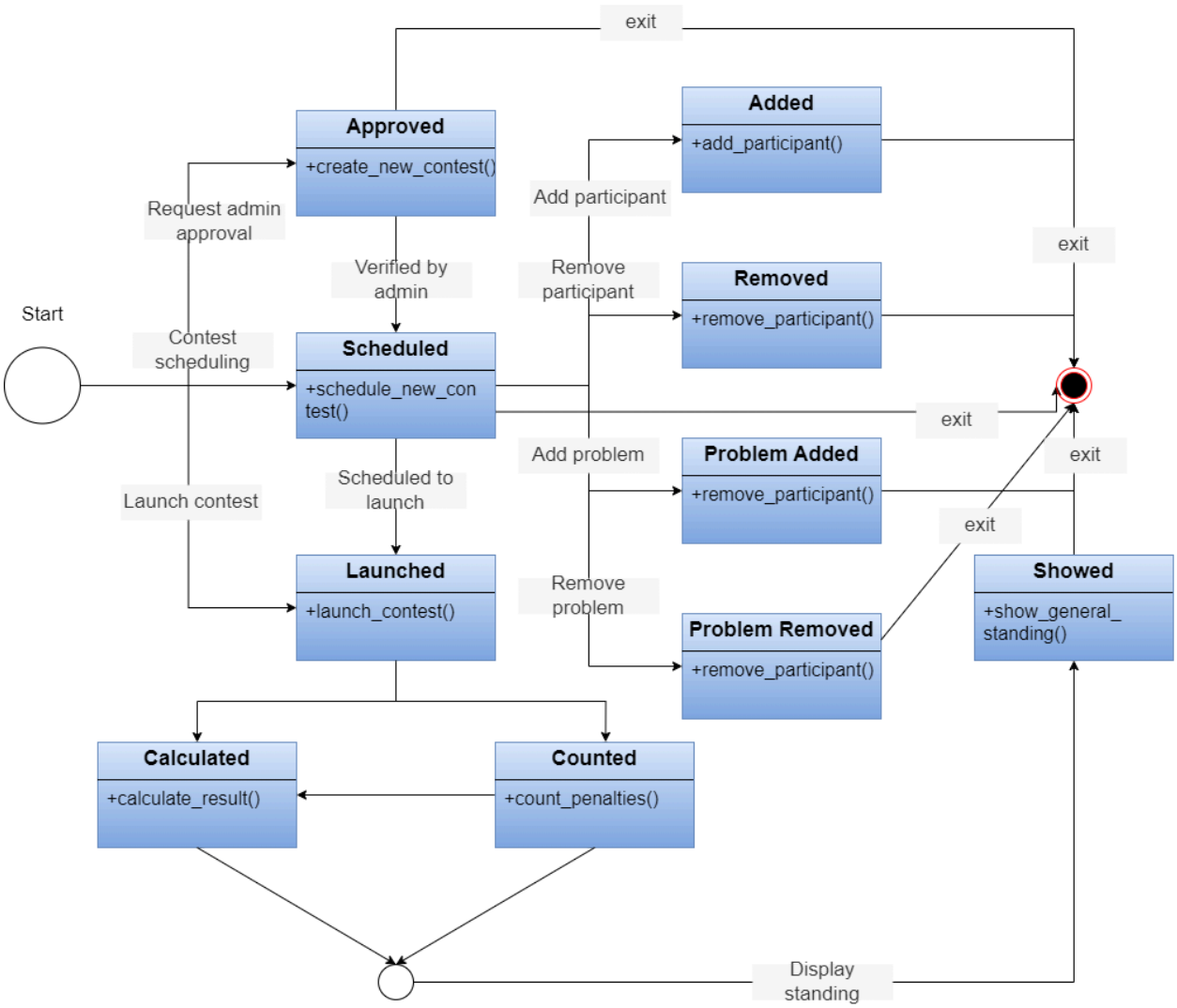


ID: 02
Name: Contestant



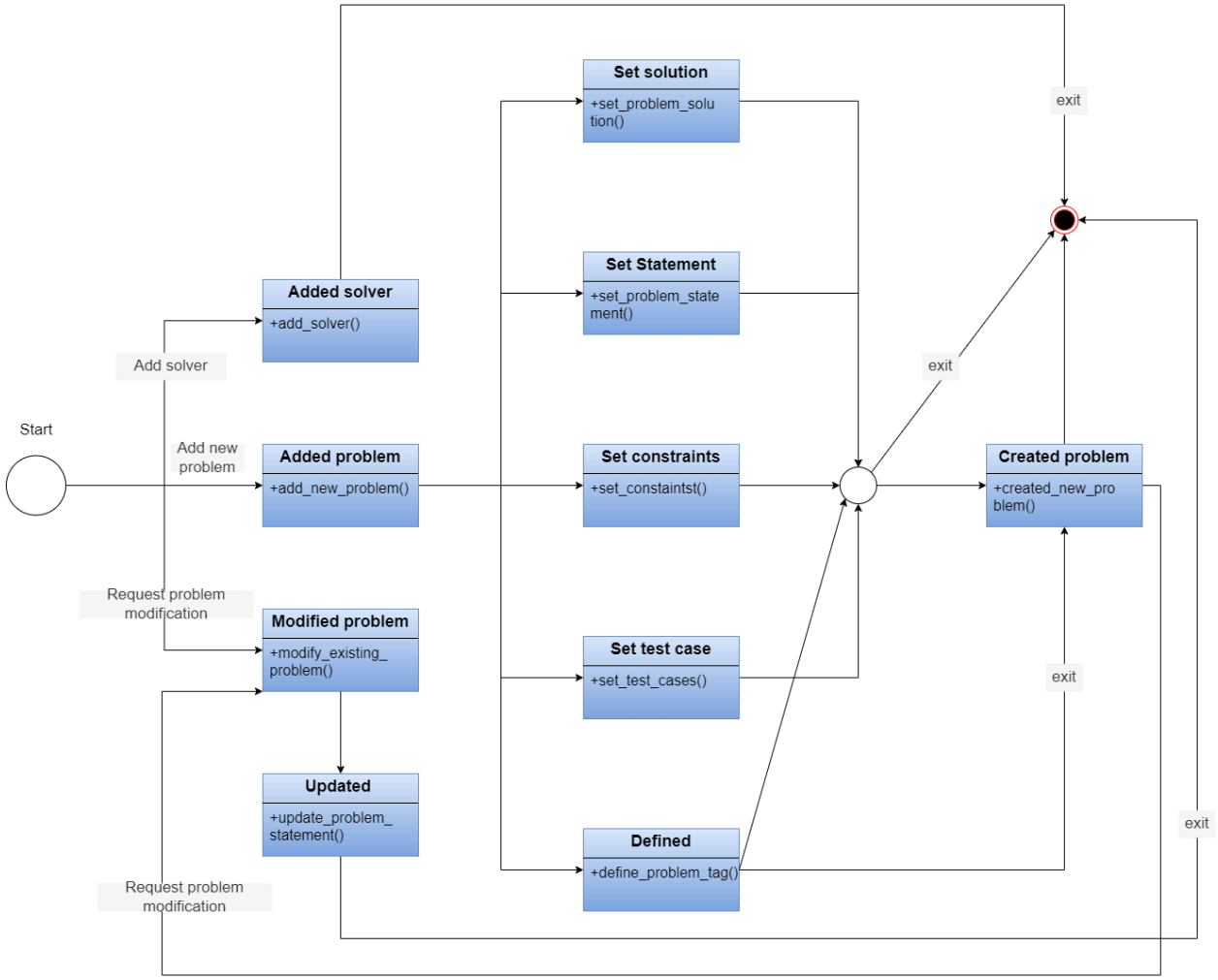
ID: 03

Name: Contest

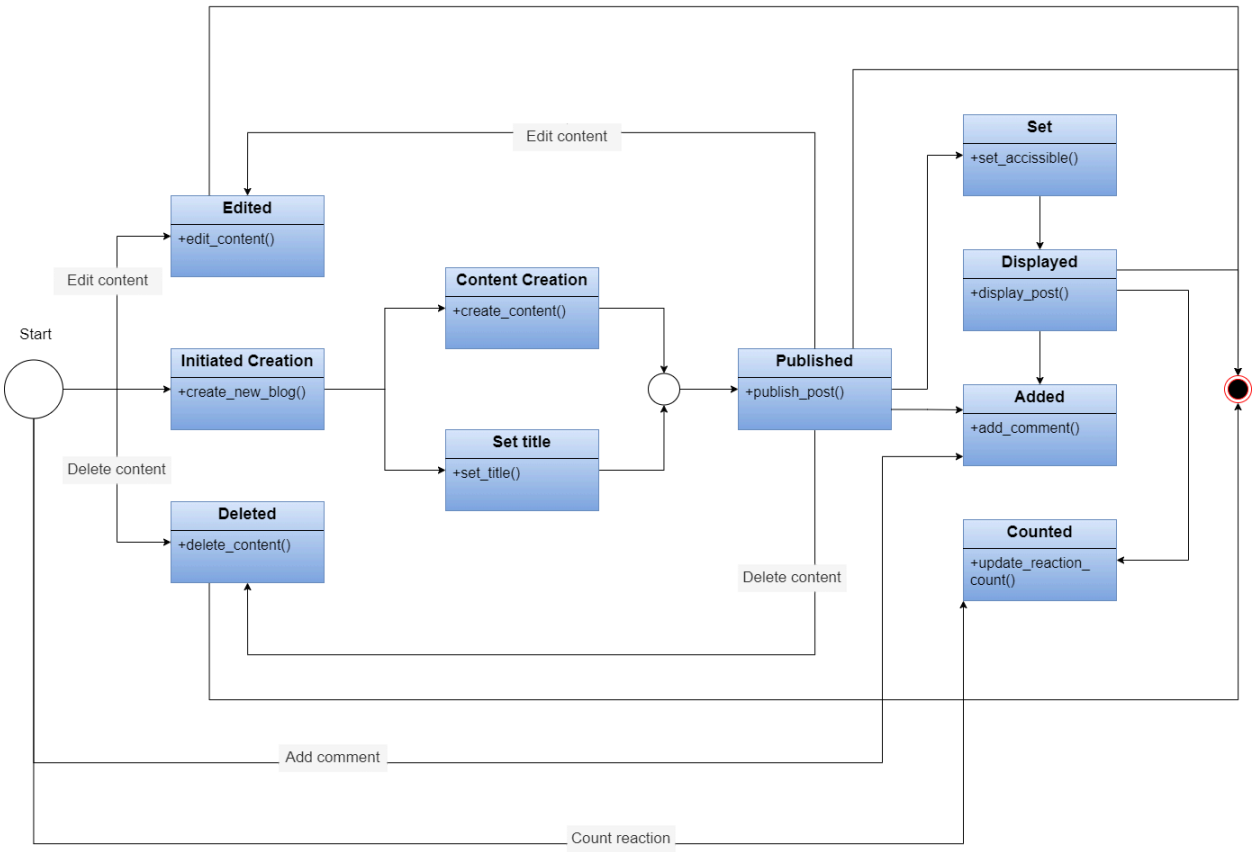


ID: 04

Name: Problem

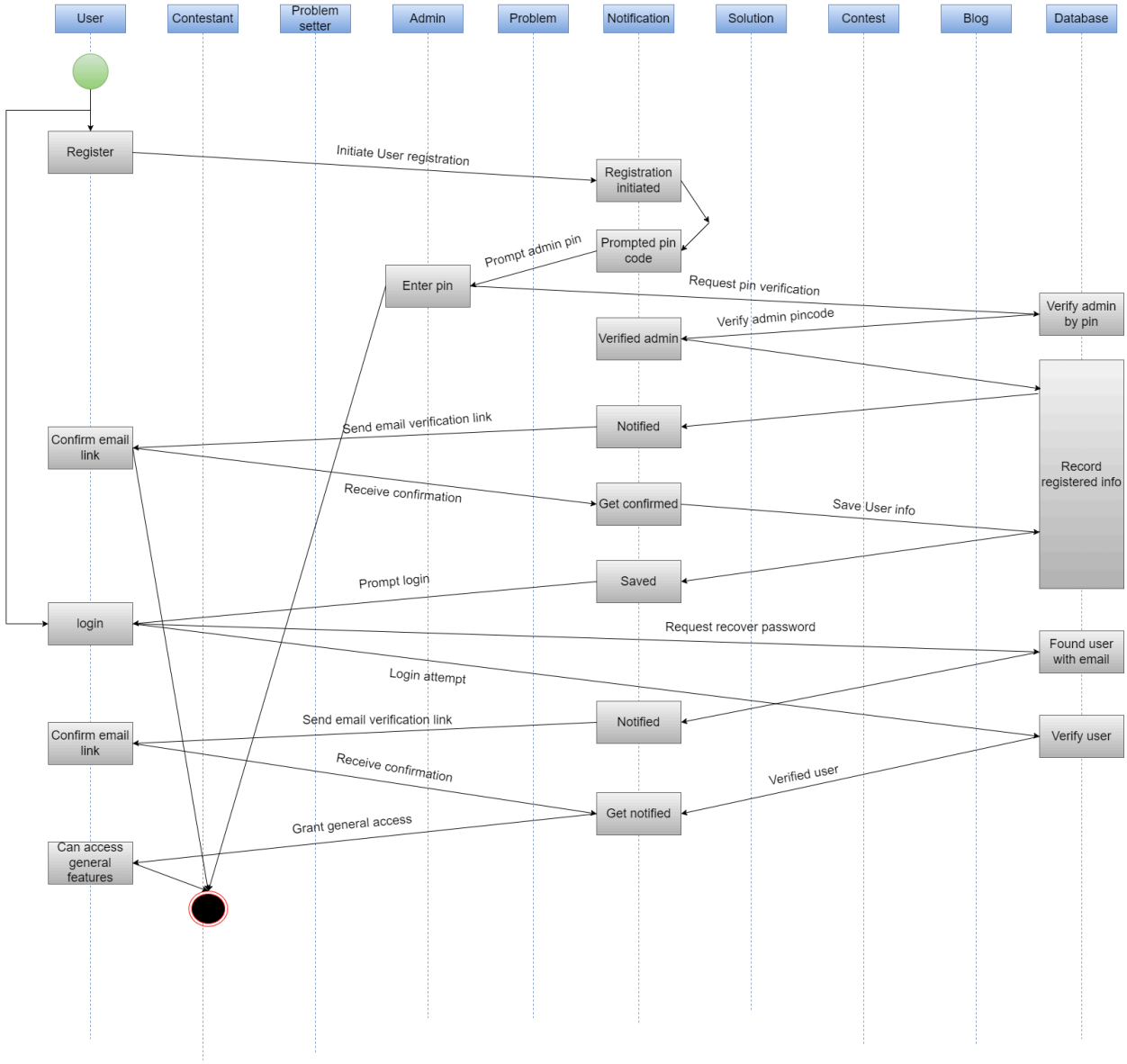


ID: 05
Name: Blog

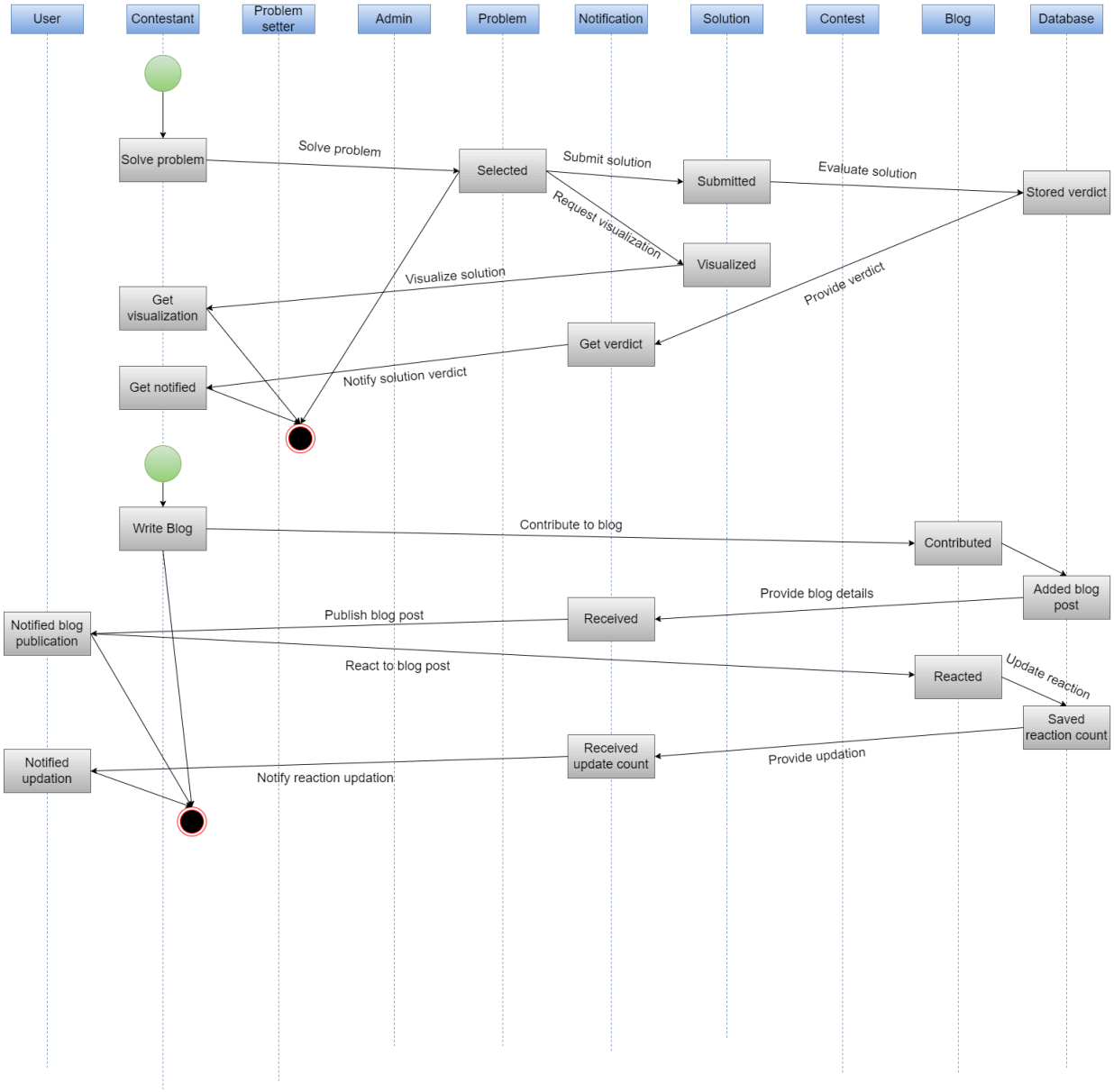

Edit content
Set
+set_accissible()
Edited
+edit_content()
Edit content
Displayed
+display_post()
Content Creation
+create_content()
Start
Initiated Creation
+create_new_blog()
Published
+publish_post()
Added
+add_comment()
Set title
+set_title()
Delete content
Deleted
+delete_content()
Counted
+update_reaction_
count()
Delete content
Add comment
Count reaction

8. Sequence Diagram

Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focused and they show the order of the interaction visually by using the vertical axis of the diagram to represent time, what messages are sent and when.

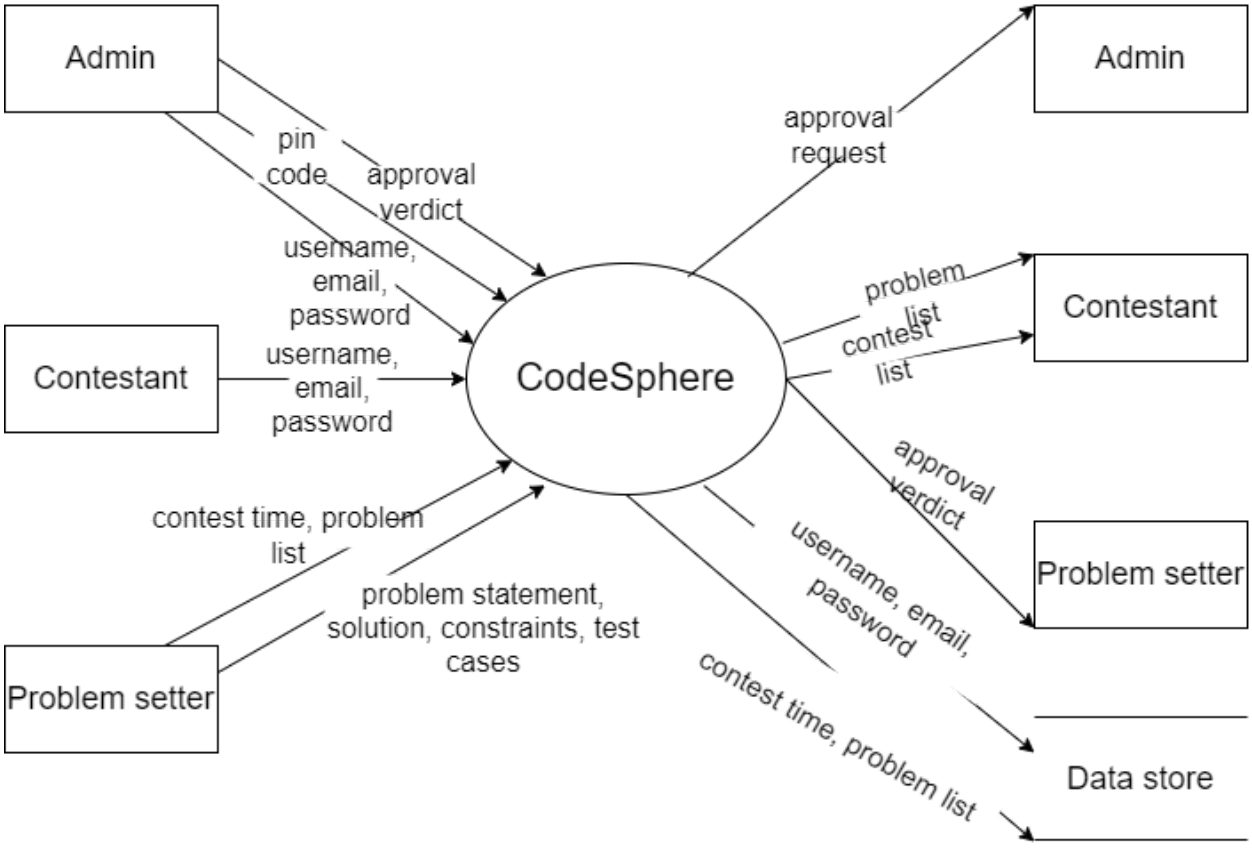

User
Contestant
Problem setter
Admin
Problem
Notification
Solution
Contest
Blog
Database
Register
Initiate User registration
Registration initiated
Prompted pin code
Prompt admin pin
Enter pin
Request pin verification
Verify admin by pin
Verify admin pincode
Verified admin
Notified
Send email verification link
Confirm email link
Receive confirmation
Get confirmed
Save User info
Record registered info
Saved
Prompt login
login
Request recover password
Found user with email
Login attempt
Send email verification link
Notified
Confirm email link
Receive confirmation
Verified user
Verify user
Get notified
Grant general access
Can access general features


User
Contestant
Problem setter
Admin
Problem
Notification
Solution
Contest
Blog
Database
Accessed
Access contest list
Allowed
Provided
Received
Provide upcoming contest
Notify available contest
Get notified
Confirm contest registration
Registered
Participant registered
Get notified
Notify contest start
Received
Provide start time
Participate in contest
Participated
Saved participation
Contest Creation
Request contest create
Generated
Request contest details
Contest details
Set custom problem
Set custom problem
Submit custom problem solution
Submit solution
Provide contest details
Set existing problem
Provide contest details
Get contest details
Received contest approval request
Notify contest approval
Request contest approval
Received
Validate contest details
Saved contest
Add contest
Provide contest details
Launched
Launch contest
Get notified
Notify upcoming contest


User
Contestant
Problem setter
Admin
Problem
Notification
Solution
Contest
Blog
Database
Solve problem
Solve problem
Selected
Submit solution
Submitted
Evaluate solution
Stored verdict
Request visualization
Visualized
Visualize solution
Get visualization
Provide verdict
Get verdict
Get notified
Notify solution verdict
Write Blog
Contribute to blog
Contributed
Added blog post
Provide blog details
Notified blog publication
Publish blog post
Received
React to blog post
Reacted
Update reaction
Saved reaction count
Notified updation
Notify reaction updation
Received update count
Provide updation

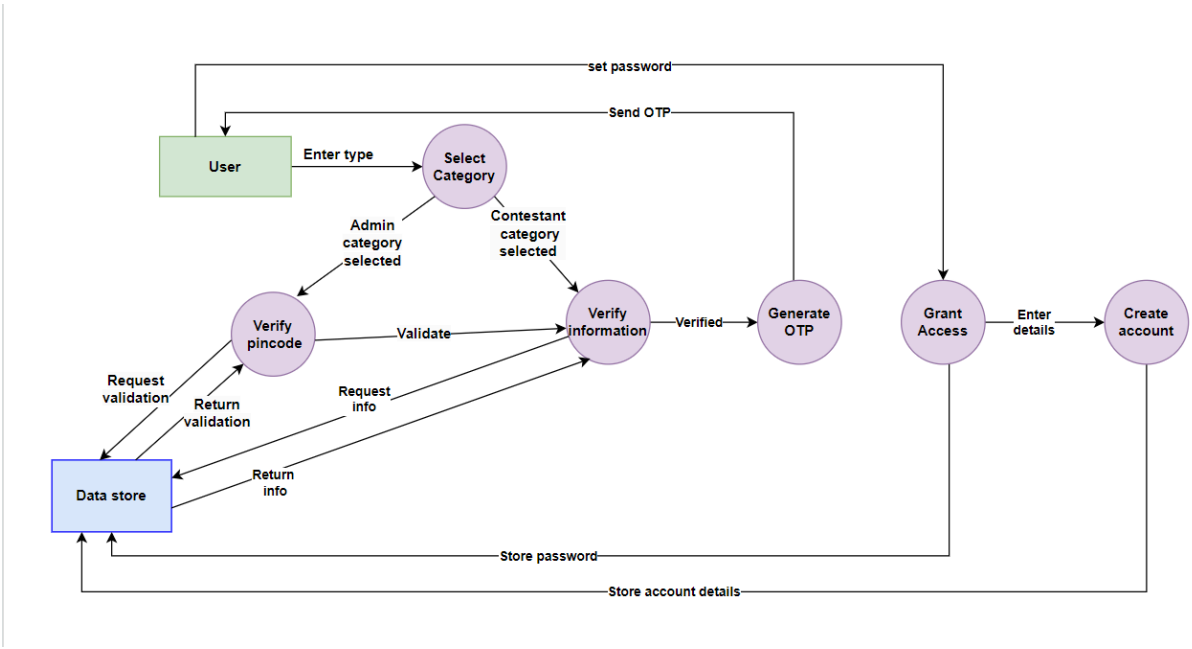
7. Data flow Diagram

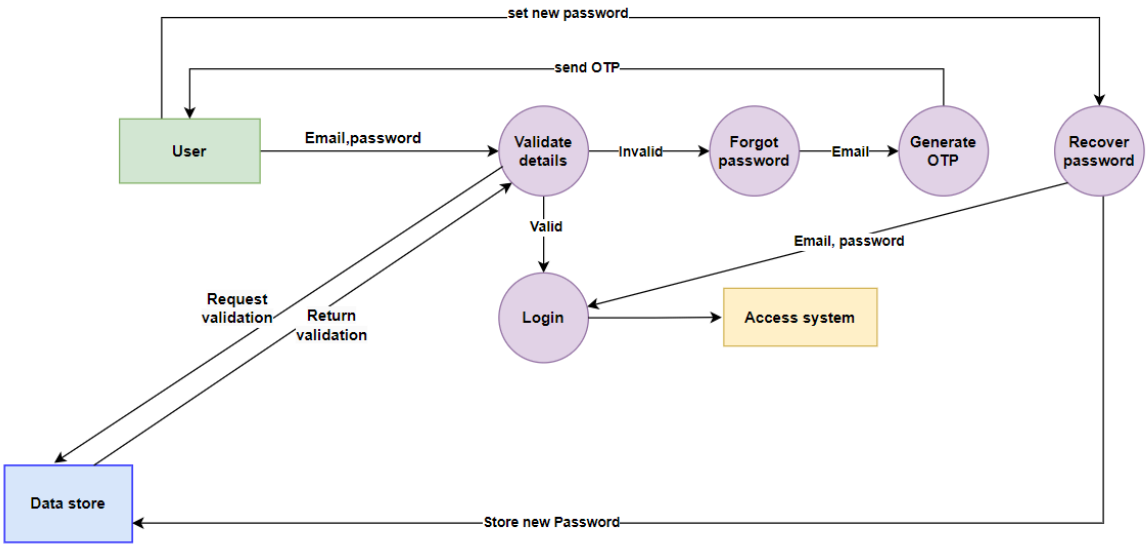
ID: 0
Name : CodeSphere



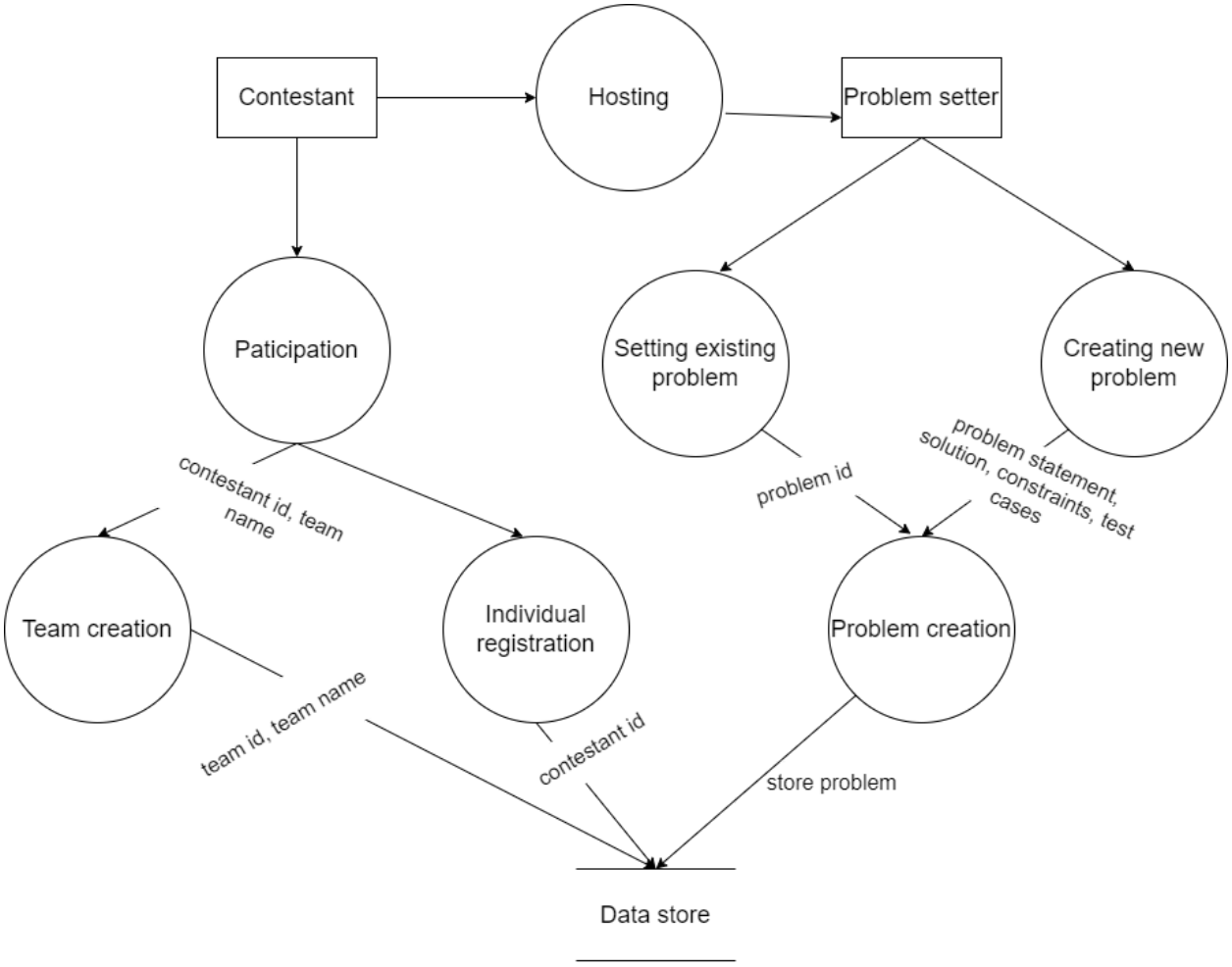
ID: 1.1

Name: Registration and authentication system





ID: 1.2
Name : Contest



ID: 1.3
Name : Knowledge nexus