# CHAPTER-01: INTRODUCTION

This chapter is a part of our software requirement specification for the project “Programming Platform”. In this chapter we focus on the intended audience for this project.

## 1.1 INTRODUCTION

This document briefly describes the Software Requirement Analysis of “Programming Platform”. It contains functional, non-functional and supporting requirements and establishes a requirements baseline for the development of the system. The requirements contained in the SRS are independent, uniquely numbered and organized by topic. The SRS serves as an official means of communicating user requirements to the developer and provides a common reference point for both the developer team and the stakeholder community. The SRS will evolve over time as users and developers work together to validate, clarify and expand its contents.

## 1.2 INTENDED AUDIENCE

This SRS is intended for several audiences including project managers, designers, developers, and testers.

The project managers of the developer team will use this SRS to plan milestones and a delivery date, and ensure that the developing team is on track during development of the system.

The designers will use this SRS as a basis for creating the system’s design. The designers will continually refer back to this SRS to ensure that the system they are designing will fulfill the customer’s needs.

The developers will use this SRS as a basis for developing the system’s functionality. The developers will link the requirements defined in this SRS to the software they create to ensure that they have created a software that will fulfill all of the customer’s documented requirements.

The testers will use this SRS to derive test plans and test cases for each documented requirement. When portions of the software are complete, the testers will run their tests on that software to ensure that the software fulfills the requirements documented in this SRS. The testers will again run their tests on the entire system when it is complete and ensure that all requirements documented in this SRS have been fulfilled.

## 1.3 CONCLUSION

This analysis of the audience helped us to focus on the users who will be using our analysis. This overall document will help each and every person related to this project to have a better idea about the project.

# CHAPTER-02: INCEPTION OF PROGRAMMING PLATFORM

## 2.1 INTRODUCTION

Inception is the beginning phase of requirements engineering. It defines how a software project gets started and what the scope and nature of the problem to be solved is. The goal of the inception phase is to identify concurrent needs and conflicting requirements among the stakeholders of a software project. At project inception, we establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired and the effectiveness of preliminary communication and collaborations between the other stakeholders and the software team. The purpose of the document is to represent a short description of “Programming Platform”

To establish the groundwork we have worked with the following factors related to the inception phases:

* List of stakeholders
* Recognizing multiple viewpoints
* Working towards collaboration
* Requirements questionnaire

### 2.1.1 LIST OF STAKEHOLDERS

Stakeholder refers to any person or group who will be affected by the system directly or indirectly. Stakeholders include end-users who interact with the system and everyone else in an organization that may be affected by its installation. At inception, a list of people who will contribute input as requirements are elicited. To identify the stakeholders we tried to find out those following questions:

* Who is going to use the platform?
* Who is going to manage the platform?
* Who is going to get benefit from the platform?
* Who is going to give feedback to us?

We identified the following stakeholders for our “Programming Platform”.

* Programming Platform Manager: Programming platform manager is a person who will manage the whole system. Such as set a contest, remove contest, add problem, remove problem.
* Contest Regulator: Contest regulator can set, remove contest or problem.
* Problem Solver: Problem solver will be able to see the contests and problems. They can solve a problem and submit the solution code to the system for judgment.

### 2.1.2 RECOGNIZING MULTIPLE VIEWPOINTS

Different stakeholders achieve different benefits from the system. Consequently, each of them has a different view of the system. So we have to recognize the requirements from multiple points of view, as well as multiple views of requirements. Assumptions are given below:

Programming Platform Administrator viewpoints:

* User friendly and efficient system
* Strong Authentication
* Strong security system

Contest Regulator viewpoints:

* User friendly and efficient system
* Efficient contest management system
* Giving live rank of contenders of any contest
* Minimum time to process verdict on solution submission
* Users are supposed to view her/his own source code

Problem Solver viewpoints:

* User friendly and efficient system
* Fascinating outlook
* Minimum time on returning verdict
* User profile statistics of user submission in contest
* Source code reading opportunity of which have been already submitted

### 2.1.3 WORKING TOWARDS COLLABORATION

Every stakeholder has their own requirements. There are some common and conflicting requirements of our stakeholder. That’s why we followed the following steps to merge these requirements-

* Find the common and conflicting requirements
* Categorize them
* List the requirements based on stakeholder’s priority points
* Make final decision about requirements

Common requirements:

* User friendly and efficient platform
* Fascinating outlook
* Minimum time on returning verdict
* User profile statistics of user submission in contest
* Giving live rank of contenders of contests
* Source code reading opportunity of a user’s own source code

Conflicting requirements:

* Limited development time

Final requirements: We finalize the following requirements based on stakeholder’s priority point:

* User friendly and efficient platform
* Fascinating outlook
* Minimum time on returning verdict
* User profile statistics of user submission in contest
* Giving live rank of contenders of contests
* Source code reading opportunity of a user’s own source code

### 2.1.4 REQUIREMENTS QUESTIONNAIRE

We first ask the stakeholder some context free questions to understand the project’s overall performance and goals. These questions are mentioned in section 2.1.1. These questions help us to identify the stakeholders of the project. Then we ask our next set of questions to better understand the problem and take stakeholder’s opinion about the solution.

## 2.2 CONCLUSION

The Inception phase helped us to establish basic understanding about the “Programming Platform”, identify the stakeholders who will be benefited if this system becomes automated, define the nature of the system and the tasks done by the system, and establish a preliminary communication with our stakeholders.

In our project, we have established a basic understanding of the problem, the nature of the solution that is desired and the effectiveness of preliminary communication and collaboration between the stakeholders and the software team. More studies and communication will help both sides (developer and client) to understand the future prospect of the project. Our team believes that the full functioning document will help us to define that future prospect.

# 

# CHAPTER-03: ELICITATION OF PROGRAMMING PLATFORM

After discussing on the inception phase, we need to focus on Elicitation phase. So, this chapter specifies the Elicitation phase.

## 3.1 INTRODUCTION

Requirements Elicitation is a part of requirements engineering that is the practice of gathering requirements from the stakeholders. We have faced many difficulties, like understanding the problems, making questions to the stakeholders, problems of scope and volatility. Though it is not easy to gather requirements within a very short time, we have surpassed these problems in an organized and systematic manner.

## 3.2 ELICITING REQUIREMENTS

We have seen Question and Answer (Q&A) approach in the previous chapter, where the inception phase of requirement engineering has been described. Requirements Elicitation (also called requirements gathering) combines problem solving, elaboration, negotiation and specification. The collaborative working approach of the stakeholders is required to elicit the requirements. We have finished the following tasks for eliciting requirements-

* Collaborative Requirements Gathering
* Quality Function Deployment
* Usage scenarios
* Elicitation work products

### 3.2.1 COLLABORATIVE REQUIREMENTS GATHERING

We have met with the stakeholders of this project in the inception phase such as administrator and problem solvers. Many different approaches to collaborative requirements gathering have been proposed by the stakeholders. To solve this problem we have met with the stakeholders again to elicit the requirements. A slightly different scenario from these approaches has been found.

* The meeting were conducted with an administrator of an online judge and programming problem solvers. They were questioned about their requirements and expectations.
* They were asked about the problems they were facing the current manual system.
* Lastly we selected our final requirement list from the meetings.

### 3.2.2 QUALITY FUNCTION DEPLOYMENT

Quality Function Deployment (QFD) is a technique that translates the needs of the customer into technical requirements for software. It concentrates on maximizing customer satisfaction from the software engineering process. So we have followed this methodology to identify the requirements for the project. The requirements, which are given below, are identified successfully by the QFD.

#### 3.2.2.1 NORMAL REQUIREMENTS

Normal are generally the objectives and goals that are stated for a product or system during meetings with the stakeholders. The presence of these requirements fulfills stakeholders’ satisfaction. The normal requirements of our project-

* First and foremost, a really friendly user interface
* Store the information of all users
* Minimum effort to use the software
* Problem or contest can be added, removed easily without any difficulty.
* Problem solver should get the result of the submitted code immediately.
* Show the rank of a contest

#### 3.2.2.2 EXPECTED REQUIREMENTS

These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction. Below the expected requirements are described

* Authentication process
* Interactive and attractive graphical user interface

### 3.2.3 USAGE SCENERIO

Programming Platform is an automated System for the following purposes-

* Authentication
* Problem Set Management
* Programming Contest Management
* Solution Assessment
* User Profile Management

##### Authentication

Offline programming platform has two types of users.

* Administrator
* Problem solver

At the time of installation of this system an administrator account will be created who will maintain the system. For account creation any type of user has to provide these information

* Username/Email
* Password
* Recovery Pin
* Institute

User’s password can be at most fifteen characters and pin four characters. Only authenticated user can enter to the system.

A user can log into the system entering her/his username/email and password. If provided username/email and password matches, user can enter to the system. Otherwise an error message will be generated.

If a user forget her/his password then she/he can recover her/his account. For user account recovery the system will ask for her/his username/email and pin number. If provided email and pin number matches then she/he will be able to see her/his password. Then using that password she/he can enter into the system.

##### Problem Set Management

One of the basic features of a Programming Platform is to give the programmers the opportunity to practice programming problems to enhance their coding skills. A programming platform is supposed to have some predefined problems which have been set by the administrator.

The problem set will hold the programming problems that have been set in previously arranged contests. This problem set will allow the users to practice contest problems even the contest duration is over.

All the problems will be set by the administrator. Users can surf through the problem set and solve the problems of their choice.

##### Programming Contest Management

Arrangement of contest is the greatest feature of a programming platform or judge. An administrator can arrange programming contests.

To arrange a contest, the administrator needs to set problems. The administrator will set a problem by uploading a problem description file which will be a portable document format (pdf) file. The administrator will also upload a test input file and a solution file corresponding to every problem. The duration of the contest will also be set by the administrator.

The users can participate in any of contest. Users will be ranked based on their performance in the contest.

After the contest, the problems of the contest will also be added to problem set.

##### Solution Assessment

A user can submit the solution of a problem through an input text field which will be underneath that particular problem. At the same time s/he will also select the programming language. The available programming language in this case will be C, C++ and Java.

The submitted solution will be first compiled by the system based on the programming language s/he has selected. Then this solution will be run against the pre-set test input file. The output of the given input set will be matched against the solution file which has also been set by the administrator. If the output matches with the inputted solution file, the solution code will be accepted. Otherwise, the solution will be not accepted. In response to the submitted solution code, the system will provide one of the following verdicts:

* Accepted *: If the output satisfies the provided solution file.*
* Wrong answer : *If the output does not satisfy the provided solution file.*
* Compilation error*: If the system fails to compile the solution code.*
* Time limit exceed: *If the code takes more time than it is expected to execute that code.*

##### User Profile Management

Every user will be provided a dedicated profile against the email he has provided. In her/his profile, a user can see the problems s/he has solved in the contests or after the contests. He can also see in which contest s/he has participated.

A user can also modify her or his profile information (institute, password and recovery pin).

### 3.2.4 ELICITATION WORK PRODUCTS

The work products produced as a sequence of requirements elicitation will vary depending on the size of the system or product to be built. Here, the Elicitation work product includes

* Making a statement of our requirements for the Programming Platform.
* Making a bounded statement of scope of our system
* Making a list of the stakeholders who participated in the requirement elicitation
* A description of the system’s technical environment.
* A list of requirements that are organized by function and domain constraints that apply to each other.
* A set of usage scenario that provide insight into the use of the system

# CHAPTER 4: SCENARIO BASED MODELING

This chapter describes the Scenario Based Model for the Programming Platform.

## 4.1 INTRODUCTION

Although the success of a computer-based system or product is measured in many ways, user satisfaction resides at the top of the list. If we understand how end users (and other actors) want to interact with a system, our software team will be better able to properly characterize requirements and build meaningful analysis and design models. Hence, requirements modeling begins with the creation of scenarios in the form of Use Cases, activity diagrams and swim lane diagrams.

## 4.2 DEFINITION OF USE CASE

A Use Case captures a contract that describes the system behavior under various conditions as the system responds to a request from one of its stakeholders. In essence, a Use Case tells a stylized story about how an end user interacts with the system under a specific set of circumstances. A Use Case diagram simply describes a story using corresponding actors who perform important roles in the story and makes the story understandable for the users.

The first step in writing a Use Case is to define that set of “actors” that will be involved in the story. Actors are the different people that use the system or product within the context of the function and behavior that is to be described. Actors represent the roles that people play as the system operators. Every user has one or more goals when using system.

**Primary Actor**

Primary actors interact directly to achieve required system function and derive the intended benefit from the system. They work directly and frequently with the software.

**Secondary Actor**

Secondary actors support the system so that primary actors can do their work. They either produce or consume information.

## 4.3 USE CASE DIAGRAMS

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

### 4.3.1 Level-0 Use Case Diagram

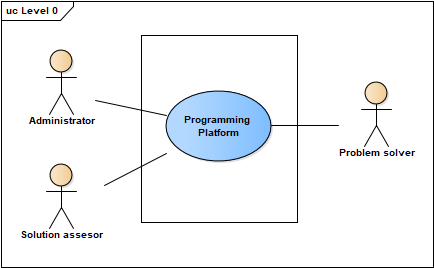


Figure: Level 0 Use Case Diagram- Programming Platform

### 4.3.2 Level-1: Use Case Diagram-Subsystem

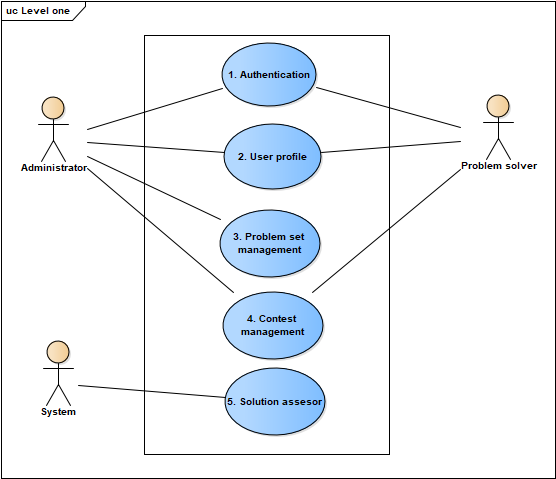


Figure: Level 1 Use Case diagram - Subsystems

**Description of Use Case Diagram Level 1-**

Primary Actor: Administrator, Problem Solver, System.

There are five subsystems in the programming platform. They are:

1. Authentication.

2. User Profile.

3. Problem Set Management.

4. Contest Management.

5. Solution Assessment.

### 4.3.3 Level-1.1: Use Case Diagram-Authentication

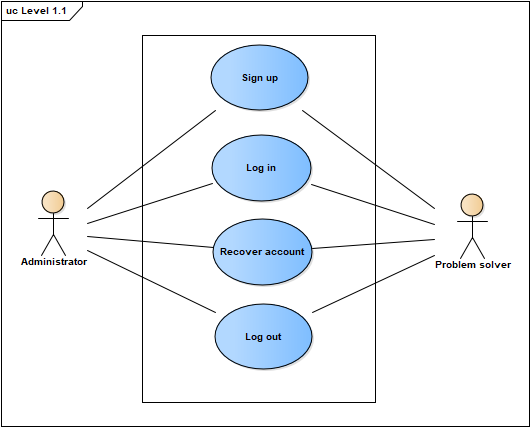


Figure: Level 1.1 Use Case Diagram- Authentication

**Primary Actor:** Administrator, Problem Solver.

The Authentication Subsystem can be divided into four parts:

1. Sign Up
2. Log in
3. Recover Account
4. Log Out

At the time of installation of this system an administrator account will be created who will maintain the system. For account creation any type of user has to provide these information

* Username/Email
* Password
* Recovery Pin
* Institute

User’s password can be at most fifteen characters and pin four characters. Only authenticated user can enter to the system.

A user can log into the system entering her/his username/email and password. If provided username/email and password matches, user can enter to the system. Otherwise an error message will be generated.

If a user forget her/his password then she/he can recover her/his account. For user account recovery the system will ask for her/his username/email and pin number. If provided email and pin number matches then she/he will be able to see her/his password. Then using that password she/he can enter into the system.

### 4.3.3 Level-1.2: Use Case Diagram-User Profile

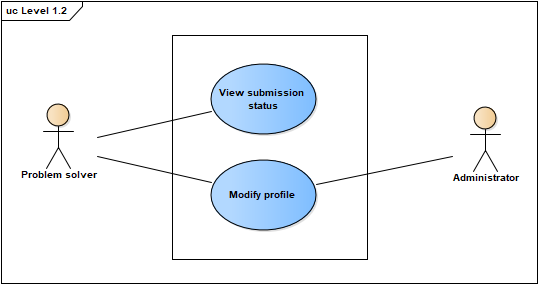


Figure: Level 1.2 Use Case Diagram- User Profile

**Primary Actor:** Problem Solver.

The User Profile Subsystem can be divided into four parts:

1. View submission status
2. Modify profile

Every user will be provided a dedicated profile against the email he has provided. In her/his profile, a user can see the problems s/he has solved in the contests or after the contests. He can also see in which contest s/he has participated.

A user can also modify her or his profile information (institute, password and recovery pin).

### 4.3.4 Level-1.3: Use Case Diagram- Problem Set Management

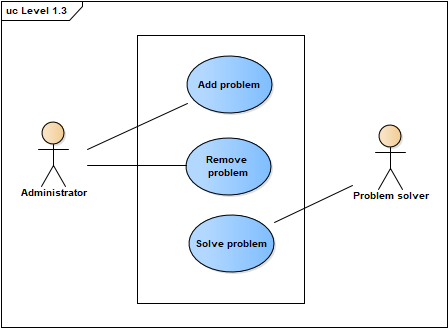


Figure: Level 1.4 Use Case Diagram- Contest Management

**Primary Actors:** Administrator, Problem Solver.

The Problem Set Management Subsystem can be divided into three parts:

1. Set Problem
2. Remove Problem
3. Solve Problem

To arrange a contest, the administrator needs to set problems. The administrator will set a problem by uploading a problem description file which will be a portable document format (pdf) file. The administrator will also upload a test input file, a solution file and time limit corresponding to every problem. The problem will also have a problem name.

The administrator can also remove any problem which has been set previously.

User can solve any problem. For that he needs to select the problem identified by contest id and problem id. Then he needs to provide the programming language name and the solution. The solution will be assessed by the subsystem “Solution assessor”.

### 4.3.5 Level-1.4: Use Case Diagram- Contest Management

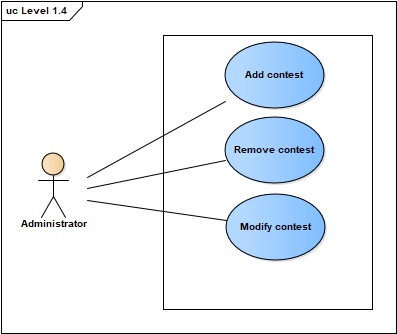


Figure: Level 1.4 Use Case Diagram- Contest Management

**Primary Actors:** Administrator, Problem Solver.

The Contest Management Subsystem can be divided into three parts:

1. Add Contest
2. Remove Contest
3. Modify Contest

To add a contest, an administrator user needs to give contest title, password, starting time, duration. An id will be generated from the system. Then administrator will set problems for the contest which has been described in “Problem Set Management”. The administrator user will also be able to modify the contest information before the contest. S/he can also remove the contest.

## 4.4 ACTIVITY DIAGRAMS OF PROGRAMMING PLATFORM

### 4.4.1 Level-1.1 Activity Diagram

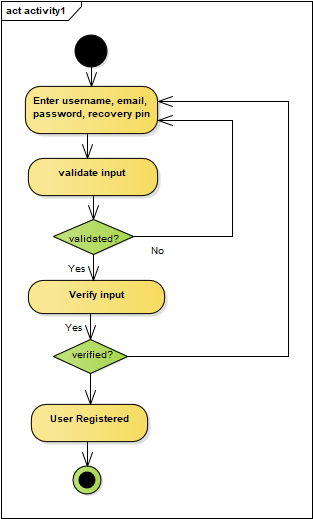


Figure: Activity Diagram 1.1.1 Sign Up

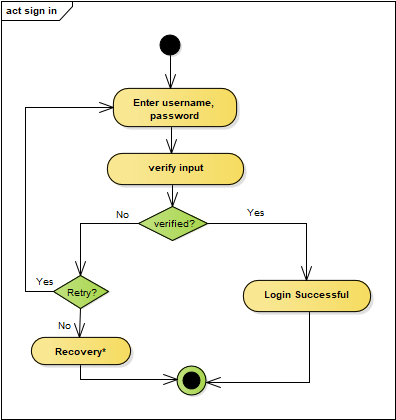


Figure: Activity Diagram 1.1.2 Sign in

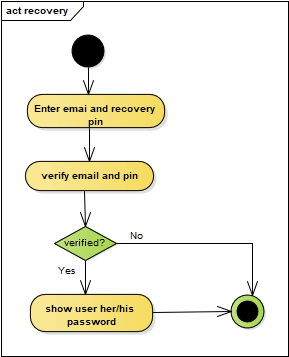


Figure: Activity Diagram 1.1.3 Account Recovery

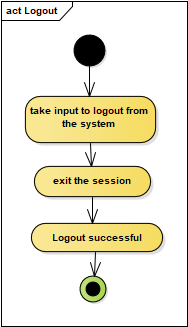


Figure: Activity Diagram 1.1.4 Logout

### 4.4.2 Level-1.2 Activity Diagram

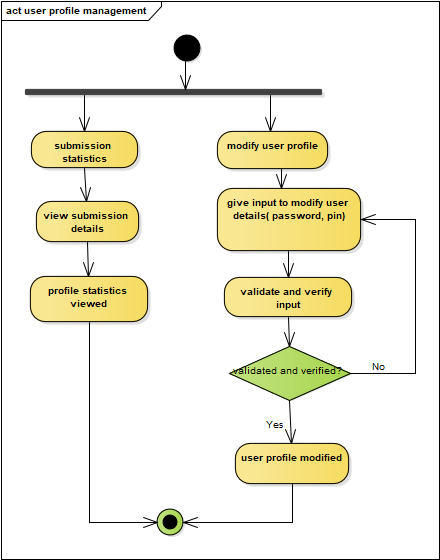


Figure: Activity Diagram 1.2 User Profile

### 4.4.3 Level-1.3 Activity Diagram

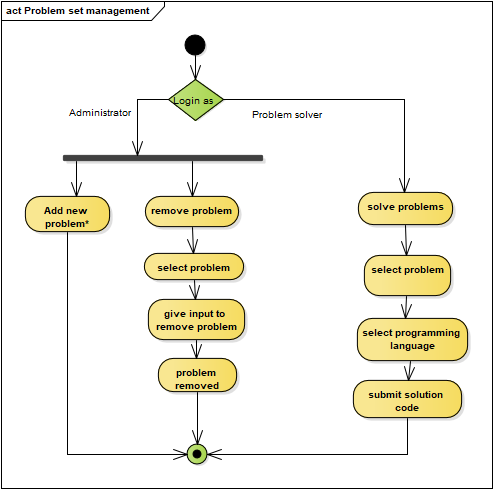


Figure: Activity Diagram 1.3 Problem Set Management

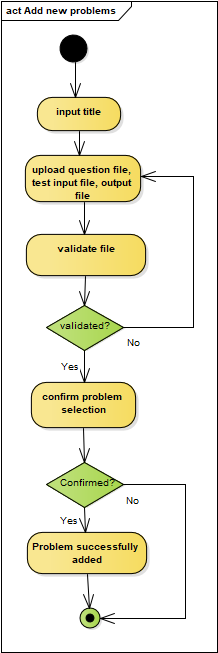


Figure: Activity Diagram 1.3.1 Add Problem

### 4.4.4 Level-1.4 Activity Diagram

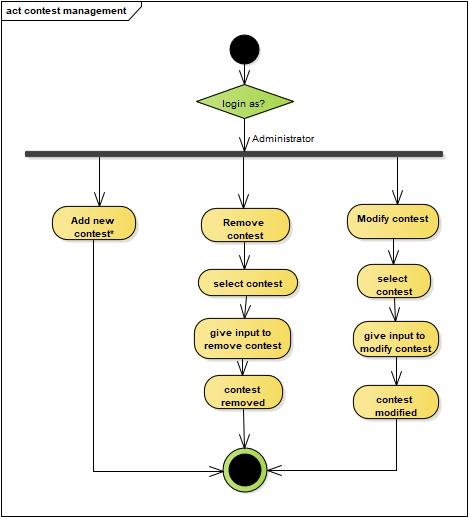


Figure: Activity Diagram 1.4 Contest Management

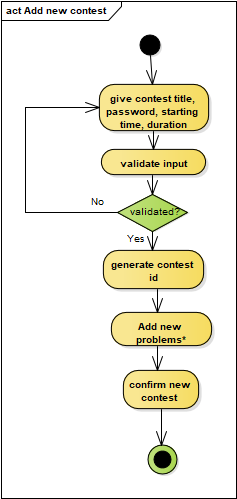


Figure: Activity Diagram 1.4.1 Add Contest

### 4.4.5 Level-1.5 Activity Diagram

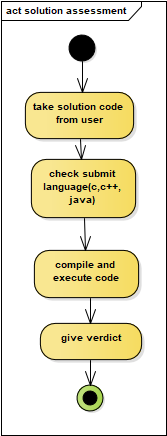


Figure: Activity Diagram 1.5 Solution Assessment

## 4.5 SWIMLANE DIAGRAMS OF PROGRAMMING PLATFORM

### 4.5.1 Level-1.1 Swimlane Diagram

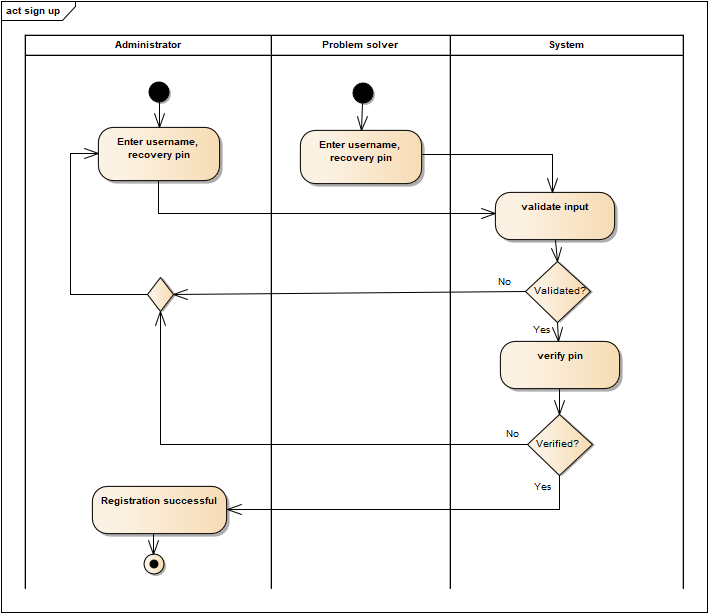


Figure: Swimlane Diagram 1.1.1 Sign Up

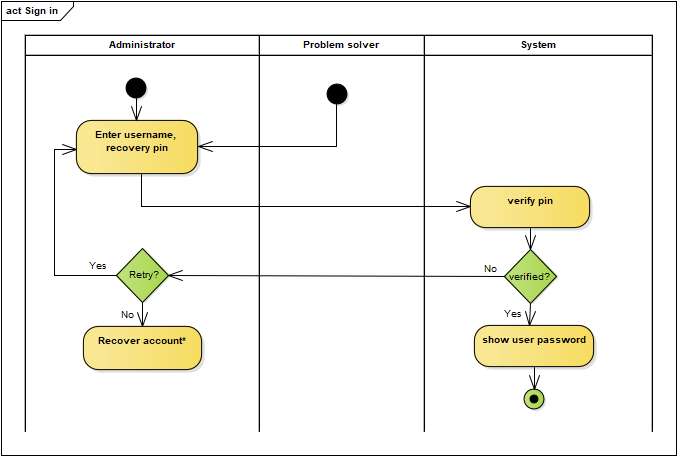


Figure: Swimlane Diagram 1.1.2 Sign in

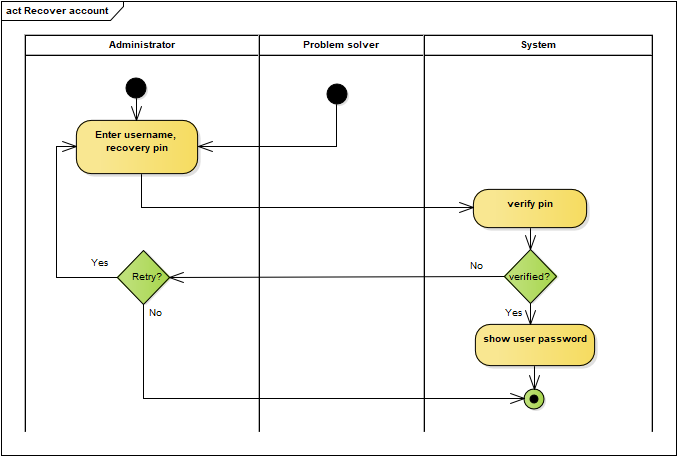


Figure: Swimlane Diagram 1.1.1 Recover Account

### 4.5.2 Level-1.2 Swimlane Diagram

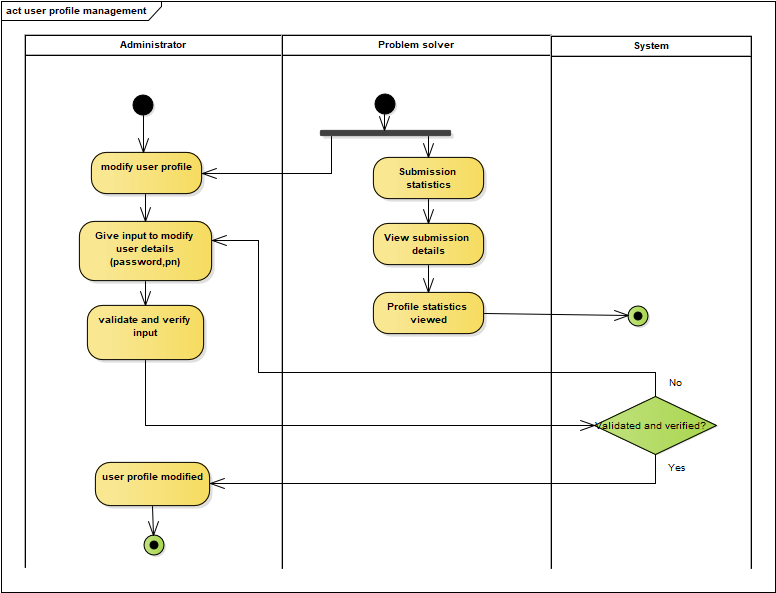


Figure: Swimlane Diagram 1. 2 User Profile

### 4.5.3 Level-1.3 Swimlane Diagram

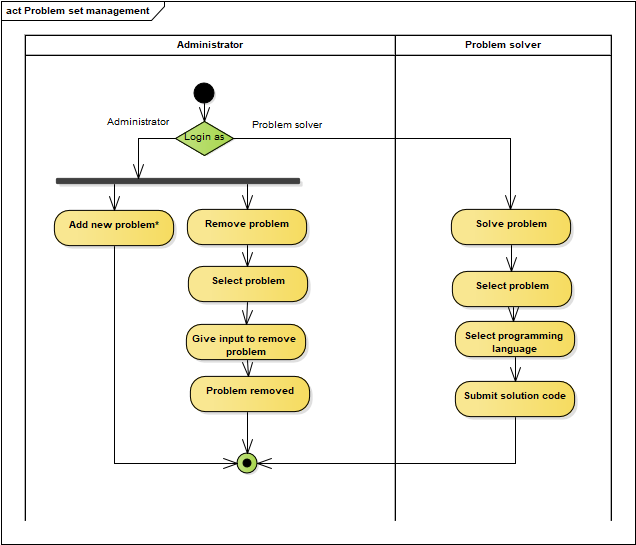


Figure: Swimlane Diagram 1.3 Problem Set Management

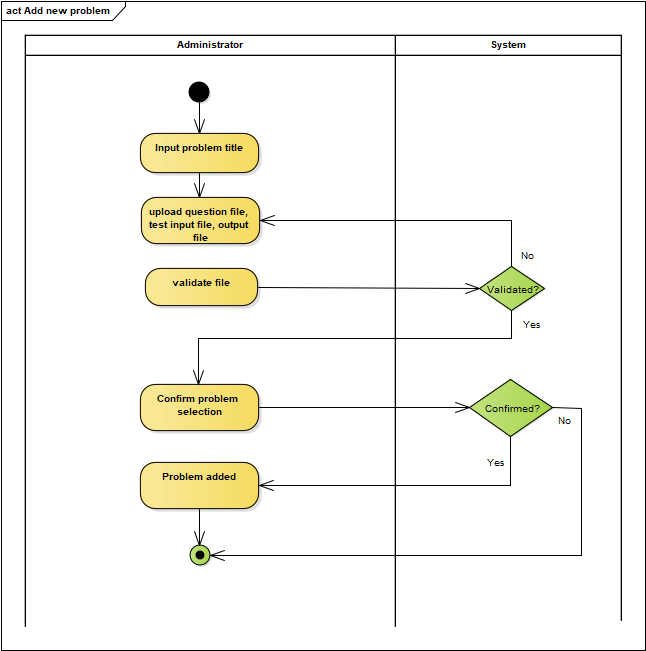


Figure: Swimlane Diagram 1.3.1 Add Problem

### 4.5.4 Level-1.4 Swimlane Diagram

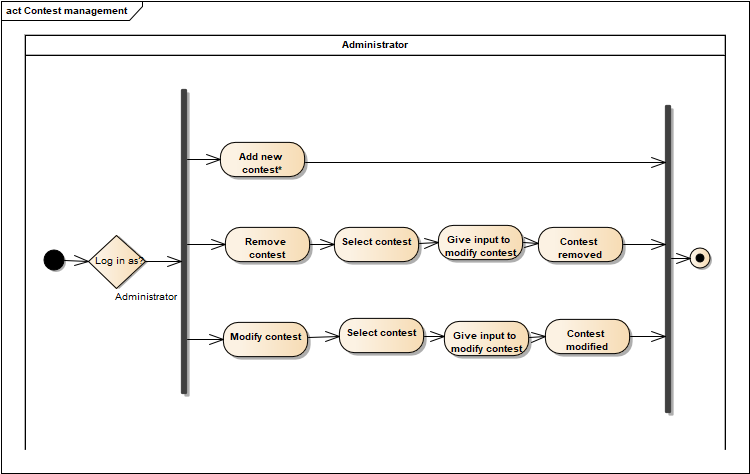


Figure: Swimlane Diagram 1.4 Contest Management

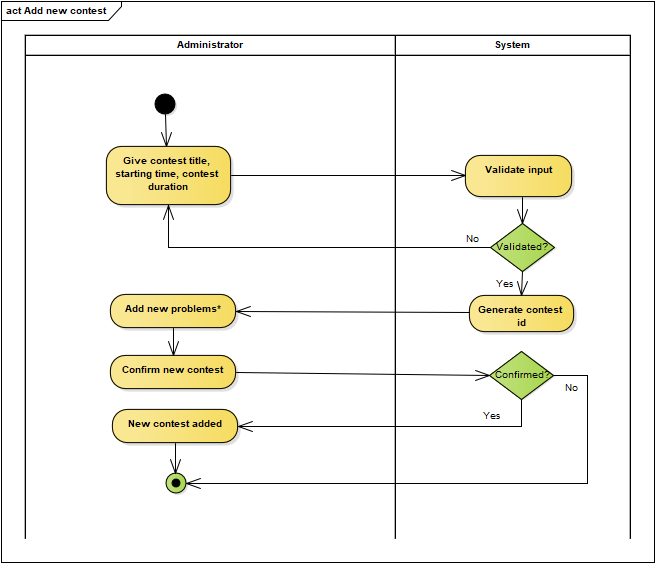


Figure: Swimlane Diagram 1.4.1 Add Contest

### 4.5.5 Level-1.5 Swimlane Diagram

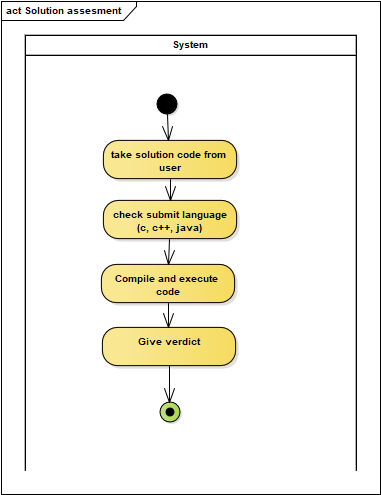


Figure: Swimlane Diagram 1.5 Solution Assessment

# CHAPTER 05: DATA BASED MODELING

## 5.1 Data Modeling Concepts

If software requirements include the necessity to create, extend or interact with a database or complex data structures need to be constructed and manipulated, then the software team chooses to create data models as part of overall requirements modeling. The entity-relationship diagram (ERD) defines all data objects that are processed within the system, the relationships between the data objects and the information about how the data objects are entered, stored, transformed and produced within the system.

## 5.2 DATA OBJECTS

A data object is a representation of composite information that must be understood by the software. Here, composite information means an information that has a number of different properties or attributes. A data object can be an external entity, a thing, an occurrence, a role, an organizational unit, a place or a structure.

### 5.2.1 Noun identification

We identified all the nouns whether they are in problem space or in solution space from our usage scenario.

|  |  |  |  |
| --- | --- | --- | --- |
| Serial No | Noun | S/P | Attributes of Noun |
|  | Programming | P |  |
|  | Platform | P |  |
|  | System | P |  |
|  | Authentication | S |  |
|  | Registration | S |  |
|  | Sign in | S |  |
|  | Sign out | S |  |
|  | Problem set | S |  |
|  | Contest | S | 35,51-55 |
|  | User | S | 16-20 |
|  | Administrator | S | 16-20,22 |
|  | Problem Solver | S | 16-20,21 |
|  | Judge | P |  |
|  | Field | P |  |
|  | Features | P |  |
|  | Username | S |  |
|  | Email | S |  |
|  | Password | S |  |
|  | Recovery pin | S |  |
|  | Institute | S |  |
|  | Registration number | S |  |
|  | Designation | S |  |
|  | Opportunity | P |  |
|  | Message | P |  |
|  | Error | P |  |
|  | Skills | P |  |
|  | Document | P |  |
|  | Installation | P |  |
|  | Programmer | S |  |
|  | Problem | S | 31-36 |
|  | Description file | S |  |
|  | Input file | S |  |
|  | Solution file | S |  |
|  | Assessment | P |  |
|  | Duration | S |  |
|  | Time limit | S |  |
|  | Set | P |  |
|  | Choice | P |  |
|  | Arrangement | P |  |
|  | Language | S |  |
|  | Compilation | P |  |
|  | Verdict | S |  |
|  | Wrong answer | S |  |
|  | Skill | P |  |
|  | Performance | P |  |
|  | Text | P |  |
|  | Code | S |  |
|  | Submission | S | 49-51 |
|  | Submission file | S |  |
|  | Compiler | S |  |
|  | Timestamp | S |  |
|  | Starting time | S |  |
|  | Contest name | S |  |
|  | Rank | S |  |
|  | No of problems solved | S |  |
|  | Profile | S |  |
|  | Problem name | S |  |

### 5.2.2 Potential Data Object

* User [16-20]
* Administrator [16-20],22
* Problem Solver [16-20],21
* Problem [31-36]
* Submission [49-51]
* Contest 35,[51-55]

### 5.2.3 Final Data Object

|  |  |
| --- | --- |
| No | Data Objects |
|  | Administrator: username, email, password, recovery pin, institute, designation |
|  | Problem Solver: username, email, password, recovery pin, institute, registration number |
|  | Contest: contest\_id, contest\_name, starting\_time, duration |
|  | Problem: problem\_id, problem\_name, timelimit, question\_file, input\_file, solution\_file, contest\_id |
|  | Submission: submission\_id, submission\_file, timestamp, verdict, compiler, problem\_id, contest\_id, username |
|  | Participation: username, contest\_id, no\_of\_problems\_solved, timstamp |

## 5.3 DATA OBJECT RELATIONS

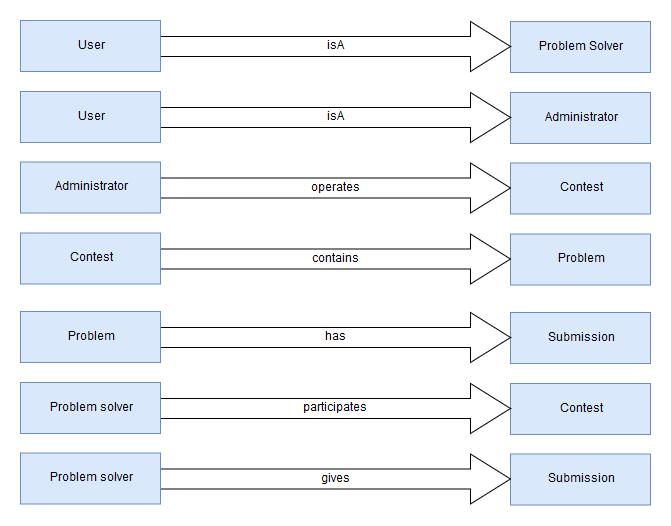


Figure: Data Object Relation

## 5.4 ENTITY RELATIONSHIP DIAGRAM

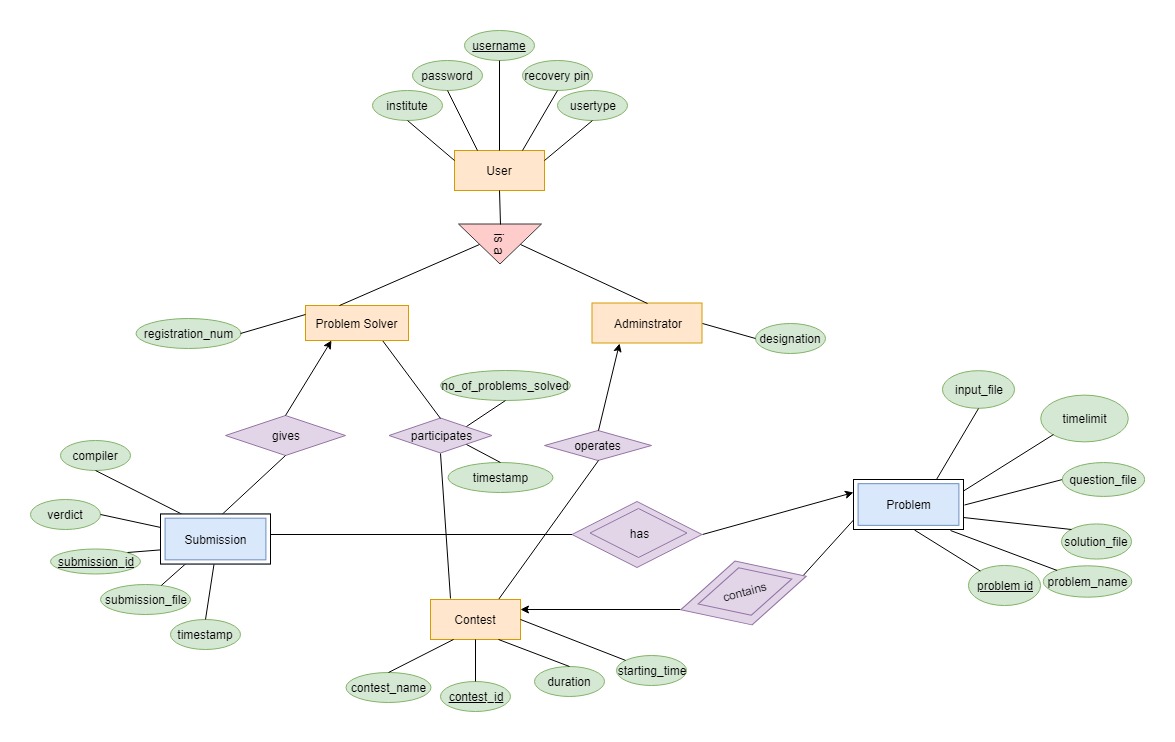


Figure: Entity Relationship Diagram

## 5.5 SCHEMA DIAGRAM

|  |  |  |
| --- | --- | --- |
|  | Administrator |  |
| Attribute | Type | Size |
| userName  password  recoveryPin  institute  userType  Designation | Varchar2  Varchar2  Number  Varchar2  Varchar2  Varchar2 | 50  15  4  50  20  50 |

|  |  |  |
| --- | --- | --- |
|  | Problem solver |  |
| Attribute | Type | Size |
| userName  password  recoveryPin  institute  userType  registrationNumber | Varchar2  Varchar2  Number  Varchar2  Varchar2  Varchar2 | 80  15  4  80  20  50 |

|  |  |  |
| --- | --- | --- |
|  | Problem |  |
| Attribute | Type | Size |
| problemID  contestID  problemName  timeLimit  questionFile  inputFile  solutionFile | Number  Number  Varchar2  Number  Varchar2  Varchar2  Varchar2 | 11  11  80  4  80  80  80 |

|  |  |  |
| --- | --- | --- |
|  | Submission |  |
| Attribute | Type | Size |
| submissionID  problemID  contestID  userName  submissionFile  compiler  timestamp  verdict | Number  Number  Number  Varchar2  Varchar2  Varchar2  Varchar2  TIMESTAMP | 20  11  80  4  80  80  80 |

|  |  |  |
| --- | --- | --- |
|  | Contest |  |
| Attribute | Type | Size |
| contestID  userName  contestName  startingTime  duration | Number  Varchar2  Varchar2 | 11  80  80 |

|  |  |  |
| --- | --- | --- |
|  | Participation |  |
| Attribute | Type | Size |
| userName  contestID  numberOfProblemSolved  timestamp | Varchar2  Number  Number  TIMESTAMP | 80  11  4 |

# CHAPTER 06: CLASS-BASED MODELING

This Chapter is intended to describe class based modeling of “**Programming Platform**”.

## 6.1 CLASS BASED MODELING CONCEPT

Class-based modeling represents the objects that the system will manipulate, the operations that will applied to the objects, relationships between the objects and the collaborations that occur between the classes that are defined.

## 6.2 GENERAL CLASSIFICATION

To identify the potential classes, we have first selected the nouns from the solution space of the story. These were then characterized in seven general classification. The seven general characteristics are as follows

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. Occurrencesorevents(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.

Following are the specifications of the nouns according to the general classifications:

|  |  |  |
| --- | --- | --- |
| Serial No | Noun | General Classification |
|  | Authentication | 3,5 |
|  | Problem set | 2,3,5,7 |
|  | Contest | 2,3,5,6,7 |
|  | User | 4,5,7 |
|  | Administrator | 4,5,7 |
|  | Problem Solver | 1,4,5,7 |
|  | Username |  |
|  | Email |  |
|  | Password |  |
|  | Recovery pin |  |
|  | Institute |  |
|  | Registration number |  |
|  | Designation |  |
|  | Problem |  |
|  | Description file |  |
|  | Input file |  |
|  | Solution file |  |
|  | Duration |  |
|  | Time limit |  |
|  | Language |  |
|  | Verdict |  |
|  | Wrong answer |  |
|  | Code |  |
|  | Submission | 2,5,7 |
|  | Submission file |  |
|  | Compiler |  |
|  | Timestamp |  |
|  | Starting time |  |
|  | Contest name |  |
|  | Rank | 2,7 |
|  | No of problems solved |  |
|  | Profile | 2,7 |
|  |  |  |

## 6.3 SELECTION CRITERIA

The potential classes were then selected as classes by six Selection Criteria. A potential class becomes a class when it fulfills all six characteristics.

1. Retain 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.

|  |  |  |
| --- | --- | --- |
| **No** | **Noun** | **Selection criteria** |
|  | Authentication | 3 |
|  | Problem set | 1, 3-5 |
|  | Contest | 1, 3-5 |
|  | User | 1-5 |
|  | Administrator | 1-5 |
|  | Problem Solver | 1-5 |
|  | Submission | 1, 3-5 |
|  | Rank | 3-5 |
|  | Profile | 1, 3-5 |

## 6.4 ASSOCIATE NOUN AND VERB IDENTIFICATION

We will now identify the nouns and verbs associated with the potential classes to better find out the attributes and methods of each class.

|  |  |  |  |
| --- | --- | --- | --- |
| No | Potential class | Noun | Verb |
|  | Authentication | Problem solver, Administrator | Log in, sign up, recover account and log out. |
|  | Problem set | Problem name, time limit, question file, input file, solution file, problem id. | Set new problem, remove problem, display problem set. |
|  | Contest | Contest id, contest name, duration, starting time. | Set new contest remove contest, modify contest, display contests, show rank of a contest. |
|  | User | User name, password, recovery pin, user type, institute. | Create new user |
|  | Administrator | User name, password, recovery pin, user type, institute, designation. | Managing contest, managing problem |
|  | Problem solver | User name, password, recovery pin, user type, institute, registration number. | Participating in contest, submitting code, check submission status. |
|  | Submission | Submission language, verdict, submission code. | Receiving code, compiling and running code, giving verdict. |
|  | Rank | User, number of problems solved, contest id | Calculating number of problems solved by a user in a contest, calculating rank. |
|  | Profile | User | Show profile and modify profile. |

## 6.5 Attribute Identification

|  |  |  |
| --- | --- | --- |
| No | Name | Attribute |
|  | Authentication | username  password  recoveryPin  userType  institute |
|  | User | username  password  recoveryPin  userType  institute |
|  | Administrator | username  password  recoveryPin  userType  institute  designation |
|  | Problem solver | username  password  recoveryPin  userType  institute  registrationNumber |
|  | Contest | contestId  contestName  startingTime  duration |
|  | Problem | problemName  timeLimit  questionFile  inputFile  solutionFile |
|  | Submission | submissionLanguage  verdict  submissionCode |
|  | Rank | user  numberOfProblemsSolved  contest id |
|  | Profile | user |

## 6.6 Method Identification

|  |  |  |
| --- | --- | --- |
| No | Class | Methods |
|  | Authentication | * signUp() * login() * recoverAccount() * logout() |
|  | Problem Set | * addProblem() * removeProblem() * displayProblem() |
|  | Contest | * showContest() * createContest() * removeContest() * modifyContest() |
|  | User | * createUser() |
|  | Administrator | * prepareContest() * removeContest() * modifyContest() * showResult() * displayProblem() |
|  | Problem solver | * checkSubmission() * submitCode() * participateInContest() |
|  | Submission | * matchWithSolutionFile() * compileCode() * runCode() * giveVerdict() * receiveCode() |
|  | Rank | * showRankOfContest() |
|  | Profile | * modifyProfile() * showProfile() |

## 6.8 Class card

|  |  |
| --- | --- |
| Authentication | |
| Attribute | Method |
| * username * password * recoveryPin * userType * institute | * signUp() * login() * recoverAccount() * logout() |
| Responsibilities | Collaboration |
| * Register a user * Login to the system * Recover a user account * Logout from the system | Validation  DBConnector |

|  |  |
| --- | --- |
| UserProfile | |
| Attribute | Method |
| * Username * password * recovery pin * userType * institute | * modifyProfile() * showProfile() |
| Responsibilities | Collaboration |
| * Showing user profile * Modifying user profile | Validation  DBConnector |

|  |  |
| --- | --- |
| Problem solver | |
| Attribute | Method |
| * Registration number | * checkSubmission() * submitCode() * participateInContest() |
| Responsibilities | Collaboration |
| * Display the submissions * Submit code * Participate in contest | Contest  Submission  DBConnector |

Note: Problem solver will extend UserProfile class.

|  |  |
| --- | --- |
| Contest | |
| Attribute | Method |
| * contestId * contestName * startingTime * duration | * showContest() * prepareContest() * removeContest() * modifyContest() * showRankOfContest() |
| Responsibilities | Collaboration |
| * Showing all contests * Creating new contest * Removing a contest * Updating a contest * Showing rank of a contest | Problem  Validation  DBConnector |

|  |  |
| --- | --- |
| Problem | |
| Attribute | Method |
| * problemName * timeLimit * questionFile * inputFile * solutionFile | * addProblem() * removeProblem() * displayProblem() |
| Responsibilities | Collaboration |
| * Adding a new problem * Removing a problem * Displaying problem of a contest. | Validation  DBConnector |

|  |  |
| --- | --- |
| Validation | |
| Attribute | Method |
|  | * validateInput() |
| Responsibilities | Collaboration |
| * Validate data | DBConnector |

|  |  |
| --- | --- |
| Submission | |
| Attribute | Method |
| * submissionLanguage * verdict * submissionCode | * matchWithSolutionFile() * compileCode() * runCode() * giveVerdict() * receiveCode() |
| Responsibilities | Collaboration |
| * Compiling the submitted code * Running the compiled code * Matching with solution file * Giving a verdict | DBConnector |

|  |  |
| --- | --- |
| DBConnector | |
| Attribute | Method |
|  | * connectToDatabase() |
| Responsibilities | Collaboration |
| * connecting to database |  |

## 6.9 CLASS RESPONSIBILITY COLLABORATOR DIAGRAM

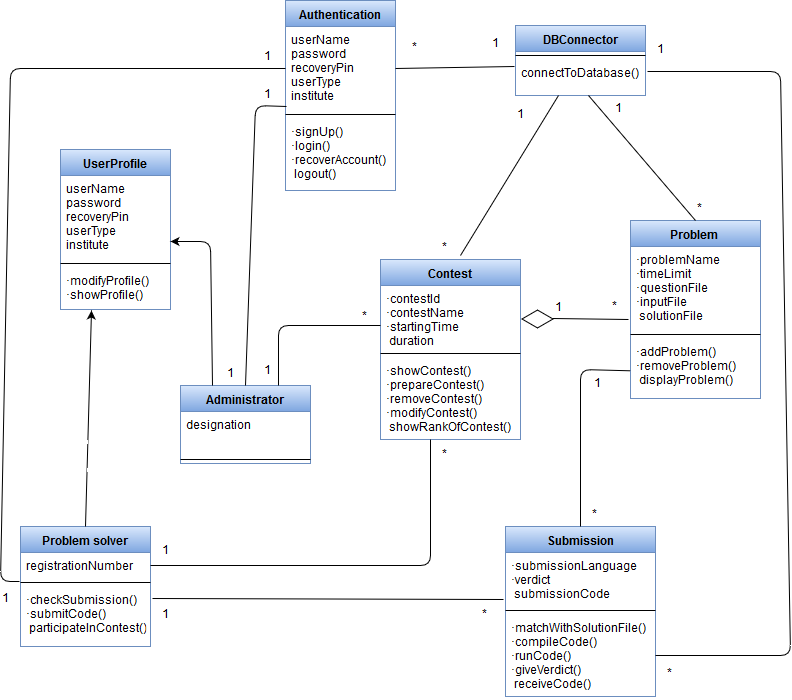


FIGURE: Class Responsibility Collaborator Diagram

# CHAPTER 07: FLOW ORIENTED MODELLING

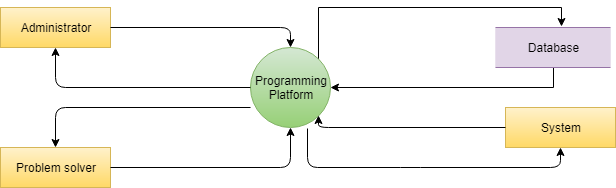
This chapter focusses on the flow oriented modelling.

## 7.1 Introduction

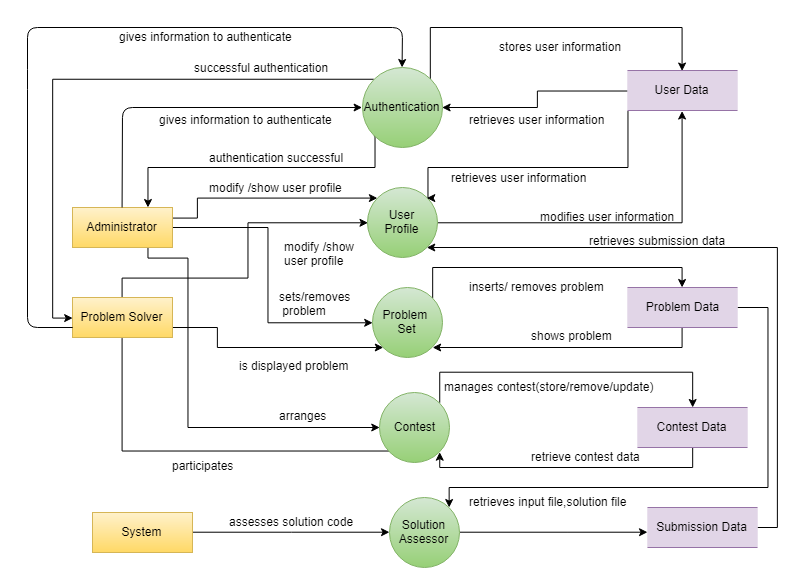
Data flow-oriented modelling is one of the most widely used requirement modelling technique. It provides a clear overview over system requirements and data flow.

## 7.2 Data Flow Diagram (DFD)

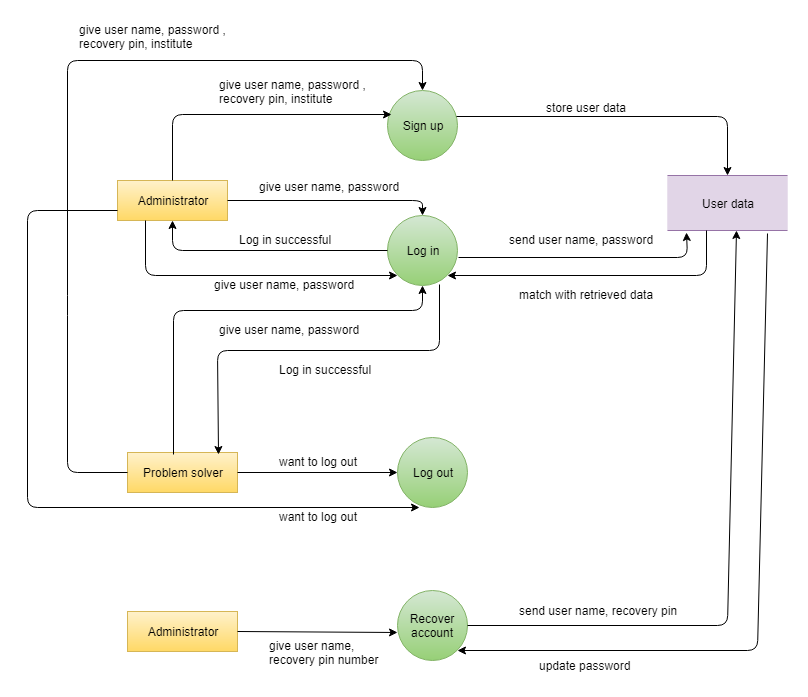
### 7.2.1 Level-0 Data Flow Diagram

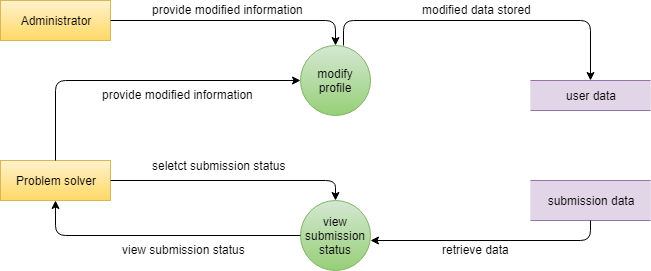


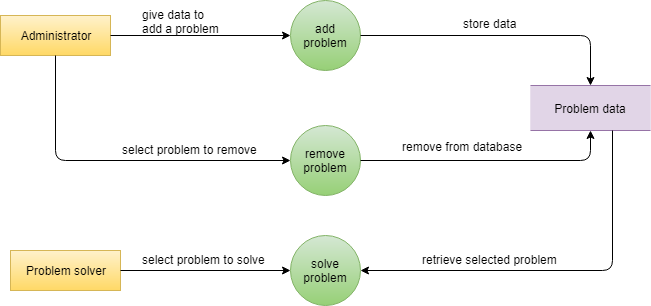
### 7.2.2 Level-1 Data Flow Diagram

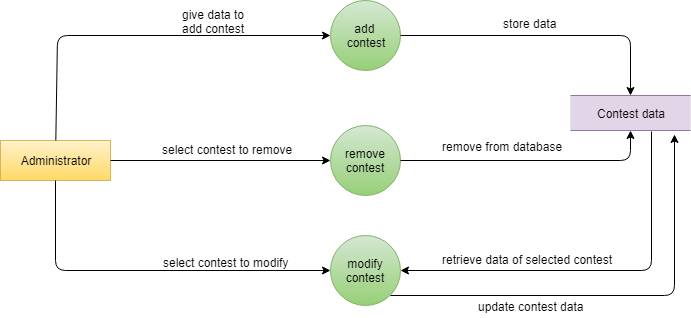


### 7.2.2 Level-2 Data Flow Diagrams









# CHAPTER 08: BEHAVIORAL MODELLING

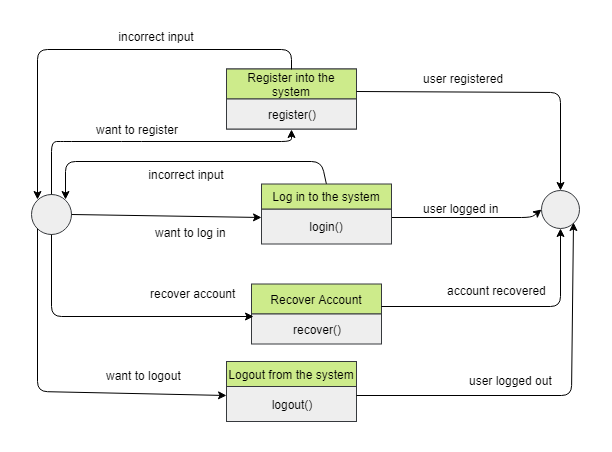
## 8.1 State Transition Diagrams

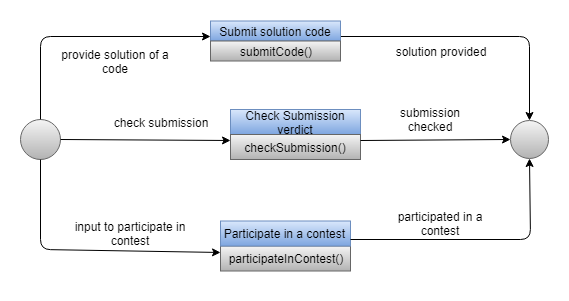
State diagram represents active states for each class the events (triggers). For this we identified all the events, their initiators and collaborators.

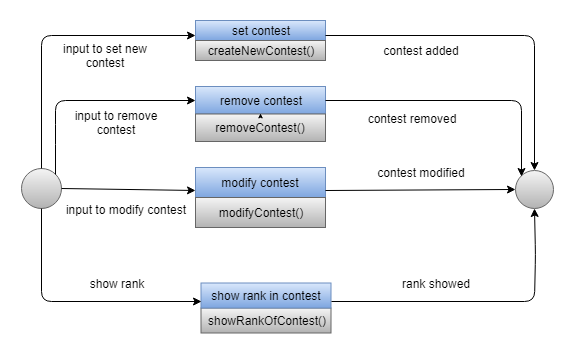
### 8.1.1 Event Identification

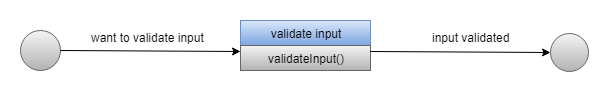
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Event | Primary Object | Collaborator | Invoked Method |
|  | Register into the system | Authentication | User, Database | Register() |
|  | Validate inputted data | System | User, Contest, Problem | Validate() |
|  | Verify inputted data | System | User, Contest, Problem | Verify() |
|  | Login into the system | Authentication | User | Login() |
|  | Recover account | Authentication |  | Recover() |
|  | Log out from the system | Authentication | System | Logout() |
|  | Set contest | Contest | Problem, Database | CreateNewContest() |
|  | Remove contest | Contest | Database | removeContest() |
|  | Participate in contest | Contest |  | participateInContest() |
|  | Modify contest | Contest | Database | modifyContest() |
|  | Set problem | Problem | Database | addProblem() |
|  | Remove problem | Problem | Database | removeProblem() |
|  | Show problem | Problem | Database | showProblem() |
|  | Submit solution code | Submission | System, Database | submitForJudgement() |
|  | Compile code | System |  | compileCode() |
|  | Run code | System |  | runCode() |
|  | Match output with predefined output file | System | Database | matchWithSolutionFile() |
|  | Give verdict | System | User | giveVerdict() |
|  | Show submission status | ProblemSolver | System | showSubmission() |
|  | Show rank in contest | Contest | System, Database | showRankOfContest() |
|  | Modify profile | User | Database | modifyUserProfile() |
|  |  |  |  |  |

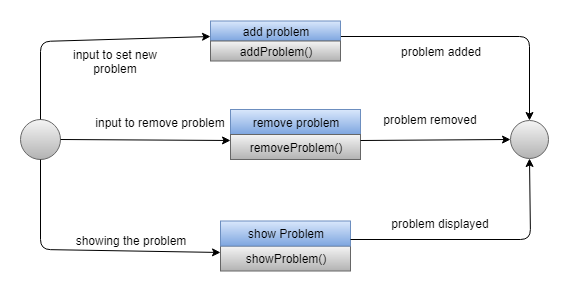
### 8.1.2 State Transition Diagrams

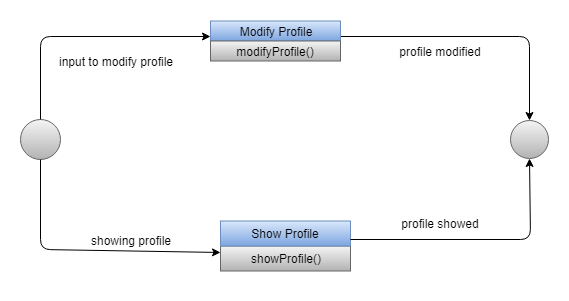


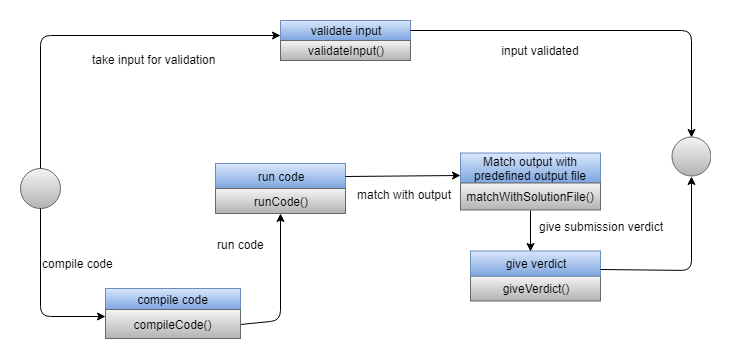






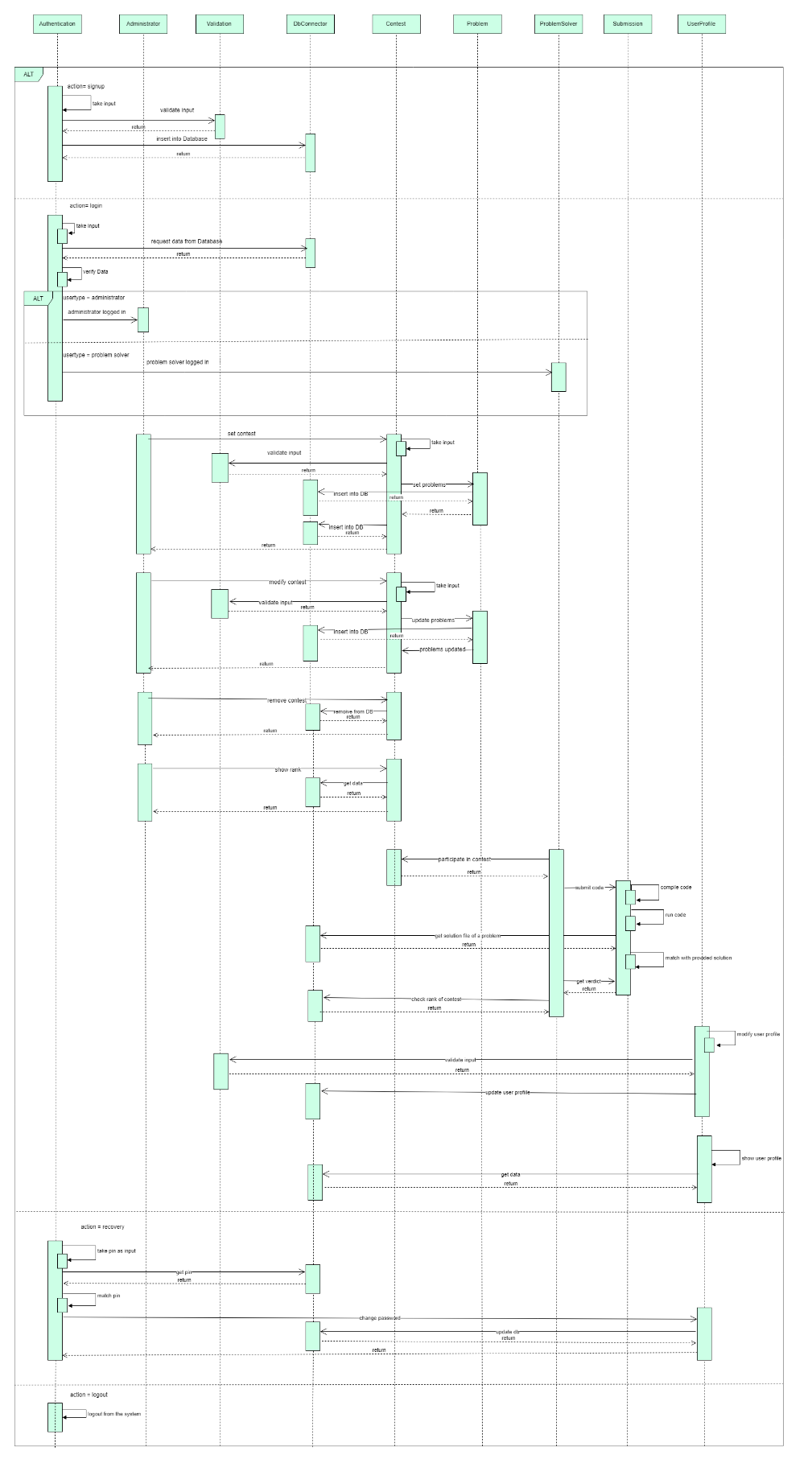






## 8.2 Sequence Diagram

A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order.



# CHAPTER 09: CONCLUSION