SPL-2 Software Requirement Specification and Analysis, 2022

TestCube

Cloud Based Test Case Generation Tool for Java Source Code

SE 505 : Software Project Lab II

Submitted by

Piash Mahmud

BSSE Roll No. : 1121

Saad Sakib Noor

BSSE Roll No. : 1122

Supervised by

Dr. Kazi Muheymin-Us-Sakib

Designation: Professor

Institute of Information Technology

Institute of Information Technology

University of Dhaka

10-02-2022

Supervisor's Signature: _____

Index

TestCube

1. Introduction

This chapter is a part of our software requirement specification for the project 'TestCube'. In this chapter we will focus on the intended audience for this project.

1.1 Purpose

This document briefly describes the Software Requirement Analysis of **TestCube**. It contains the functional, non-functional and the supporting requirements and establishes a requirement's baseline for the development of the system. The requirements contained in the SRS are independent, uniquely numbered and organized by topics. 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 report is intended for audiences including the users(programmers), project managers, developers and testers.

● The users will use this SRS to verify that the developer team has created a product that is acceptable to the customer.

● 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.

2. Inception of TestCube

In this chapter, the inception part of the SRS will be discussed briefly.

2.1 Introduction

TestCube is a cloud based test case generator that will generate and execute test cases for given java source code. It will store the result and a user can view and download the test cases and result again.

2.2 Inception Procedure

At the beginning of our project, we entered the inception stage. This stage includes how the project will be started and its scope and limitations. The main goal of this phase is to identify the requirements, demand and establish some sort of mutual understanding between the software team and the stakeholders of TestCube. In order to make this phase effective we took the following steps:

- Identifying the client of our project
- Icebreaking
- Identifying the stakeholders of TestCube
- Identifying the multiple viewpoints of stakeholders

2.3 Identify The Clint of Our Project

Any programmer wanting to test their code with ease is our potential client. An inexperienced programmer who does not know much about generating test cases and testing in general can be identified as a client of this project.

2.4 Icebreaking

Icebreaking refers to the fact that to diminish the communication barrier between two people. It is a crucial part since it denotes the acceptance of our proposal. We started this phase by talking with the stakeholders with context free languages. Their behavior, responding to our question, impacted the whole system.

2.5 Identify The Stakeholders of TestCube

Stakeholder refers to any person or group who will be affected directly or indirectly by the system. Stakeholders include end-users who interact with the system and everyone else in an organization who may be affected by its installation. TestCube has a limited number of stakeholders. They are:

- Users

2.6 Identify The Multiple Viewpoints of The Stakeholders

Different stakeholders expect different benefits from the system as every person has his own point of view. So, we have to recognize the requirements from multiple viewpoints. Different viewpoints of the stakeholders about the expected software are given below:

Users' Viewpoint

- Easy and fast interface.
- A user-friendly interface.
- Cloud based software.
- Availability to use everywhere.
- Store information about users.
- Store information about inputted code and its corresponding output.
- Report section to see the whole warnings and generated test cases.
- Easy to use.

2.7 Conclusion

The primary goal of this project is to model and design software for the programmers including beginners to generate unit test cases for their given code

and make it easy to use. For this reason, the software needs to be simple. The software will be designed in such a way as it takes very little time to get started. And also clearing out facts about what users want, how the software will work, how it can be more convenient, how it will save time and energy.

3. Elicitation of TestCube

After discussing the Inception phase, we need to focus on the 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 for the stakeholders, limited communication with stakeholders due to shortage of time and volatility of the stakeholders. 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 the Question and Answer (Q&A) approach in the previous chapter, where the inception phase of requirement engineering has been described. The main task of this phase is to combine the elements of 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 of Work Products

3.2.1 Collaborative Requirements Gathering

We have talked with some novice programmers in the inception phase. These discussions created an indecisive state for us to elicit the requirements. To solve this problem, we have met with some stakeholders (who are playing a vital role in the whole process) more than once to elicit the requirements.

3.2.2 Quality Function Deployment

Quality Function Deployment (QFD) is a process and set of tools used to effectively define customer requirements and convert them into detailed engineering specifications and plans to produce the products that fulfill those requirements. QFD is used to translate customer requirements Voice of the Customer (VOC) into measurable design targets and drive them from the assembly level down through the sub-assembly, component and production process levels. QFD methodology provides a defined set of matrices utilized to facilitate this progression.

3.2.2.1 Normal Requirements

Normal requirements reflect objectives and goals stated for a product or system during meetings with the customer. Those are the basic requirements that fulfill client satisfaction. The Normal requirements we came up with are the following:

- Login and Signup for users
- User data will be stored into database
- The tool will generate test cases for given codes
- The tool will be able execute the test cases

3.2.2.2 Expected Requirements

These requirements are so obvious that the customer need not explicitly state them. Their absence can create significant dissatisfaction.

3.2.2.3 Exciting Requirements

These requirements are beyond the user's expectations. Following are the exciting requirements of our system:

- User can set range on the test cases to be generated
- Users can check if their code behaves according to the expected behavior that they provide
- Machine learning algorithms will be used to generate test cases, along with a reliable open source tool.

3.3 Usage Scenario

Introduction

TestCube is a cloud based test case generator that will generate and execute test cases for given java source code. It will generate test cases using Randoop and a machine learning based solution developed by us. It will store the input codes given by users and the results generated for it.

Sign Up

To start using our tools, users first have to sign up. In order to sign up, users have to provide their Full-Name, Username, valid Email and a valid password. Both the username and the email must be unique for each account. Passwords must be 6 letters and must have at least one special character in it. Upon giving a unique email id, the users will be sent an OTP to verify their account creation.

Log in

In order to login, users have to provide their account's username or email and their password.

Upload Code

Users will upload the code they want to generate test cases for. They can also upload the expected behavior of the code in JSON format, if they want to reveal whether the method behaves as expected for generated test cases.

Set Test Criteria

Users can set various test criteria when they are using TestCube. They can choose which tool(s) they want to use among our machine learning based solutions and randoop. Users can also set the ranges of the test cases to be generated. They can choose to turn off auto execution using the generated test cases, which will be turned on by default.

Test Case Generation

After taking the input, **TestCube** will use the selected tool(s) and generate test cases for the given java source code. If a user sets a range of test cases to be generated, the tool(s) will only show test cases within that range.

Test Case Execution and Report Generation

By default **TestCube** will automatically execute the test cases it generates. However, users can choose to turn this feature off. If auto-execution is not disabled, TestCube will execute the generated test cases and it will add the execution details in a report. The report will contain information about the executions. It will show if there was any error or warnings during execution. The test cases triggering the error/warning will be specified. If expected behavior of the code is given, the report will contain test cases that cause the code to not work as expected. However, if the user turns auto execution of test cases off, then **TestCube** will not execute the test cases, it will only output the user generated test cases in the report.

View and Download Results

After generating test cases and executing them (if auto-execution is not disabled), TestCube will output the users all the generated test cases and reports as result. Users can view the test cases and reports and they will also have the option to download them.

Store and Show Generated Cases

All the input codes, generated test cases and reports will automatically be stored in the database as usage data. In the event that any user wants to view previous codes and all the results associated with it, they will have the option to do so. For that, users have to select which code they want to view and the server will bring all the information stored about that code from the database. Users can download their codes and generated test cases, reports associated with the codes whenever they want.

Re-run The Model

Our machine learning based model will be trained using an existing dataset in the beginning. After the tool starts being used, the existing dataset will be updated with all the source codes and test cases that are stored in the database. Then the dataset will be used to re-train the machine learning model. The model will train automatically using the provided source codes and generated test case in a previously fixed day. After training each time, it will set the next re-training day.

4. Scenario Based Modeling

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 diagram

A Use Case Diagram 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 make 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 the 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.

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

4.3 Use Case Diagrams

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

Use Case Diagram Level : 0

Name: TestCube
Primary Actors: User, Server

Secondary Actor: Database, Email

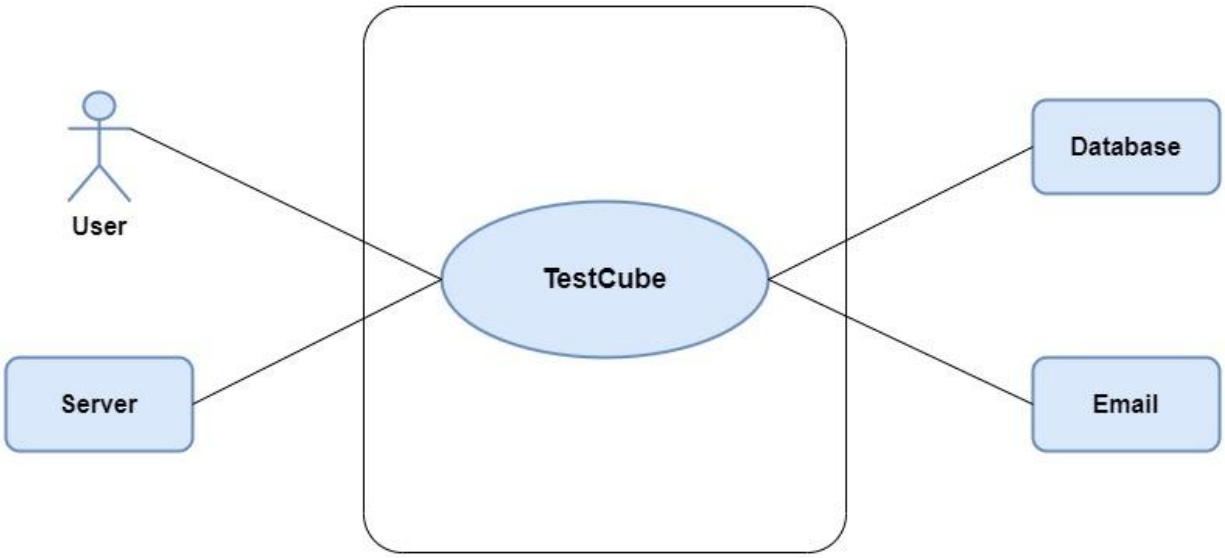


Figure 1: **TestCube** (Usecase-0)

Description of Use Case Level-0: The use case diagram "TestCube" at level 0 shows the top most view of the project and all the actors, namely:

1. User
2. Server
3. Database
4. Email

Use Case Diagram Level: 1

Name: TestCube (Detailed)
Primary Actor: User, Server
Secondary Actor: Email, Database

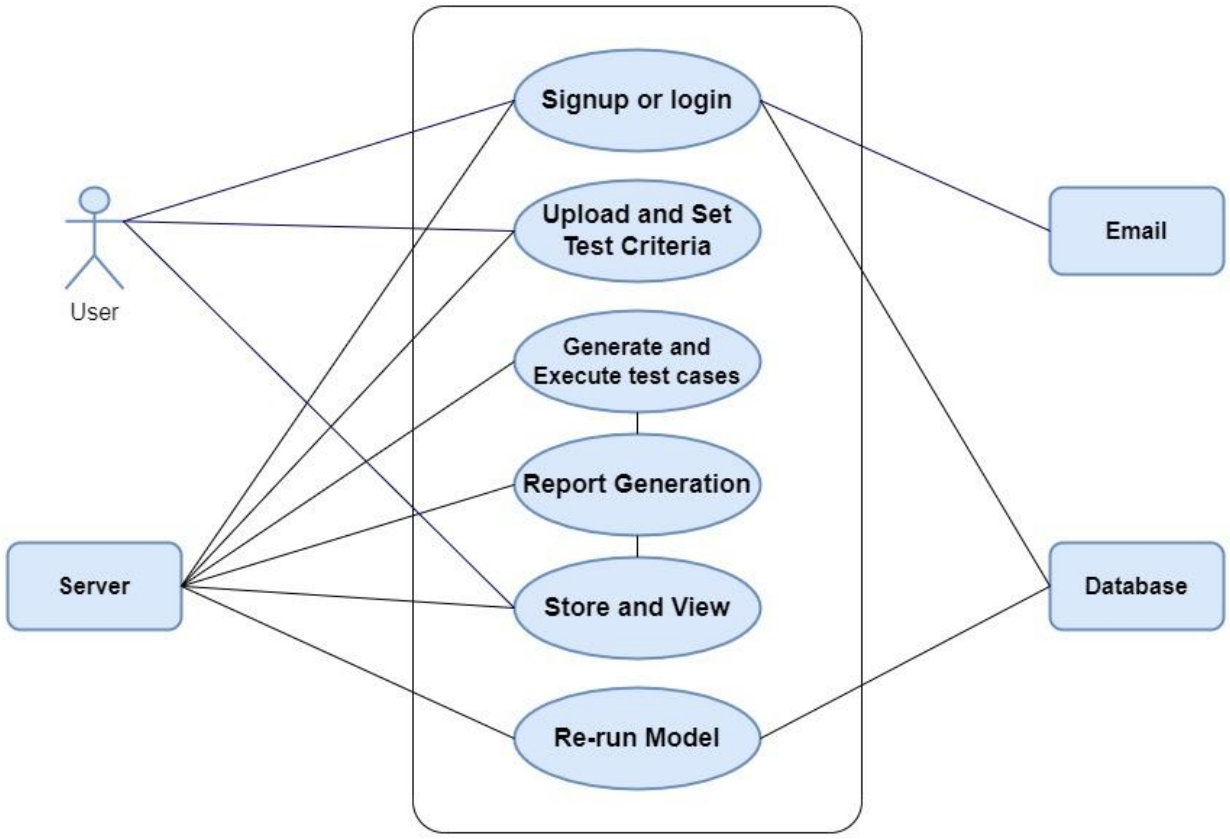


Figure 2: (usecase-1)

Description of Use Case Level-1:

- **Signup or login:** Users have to login to use TestCube. In order to login, users have to provide their account's username/email and password. If a user does not already have an account, then he/she has to sign up. Users have to provide their Full-name, username, email and password. User's account creation will be verified during sign up.
    - **Action:** user provides necessary information to the system
    - **Reply:** By verifying the information given by the user, an account will be created for the user.
    - **Action:** User provides his/her username/email and password to log into the system.
    - **Reply:** User logs into TestCube.

- **Upload and Set Test Criteria:** To generate test cases for a code, users have to upload the code. Additionally, they can upload expected behavior of the given code in JSON format. After uploading, users will have the option to set some criterias on the test cases. They can set ranges in which the test cases will be generated. They can also choose which tool(s) they want to use. If a user wants to turn off execution of the generated test cases, they can do so.
    - **Action:** User uploads codes and expected behavior
    - **Reply:** User is asked if he/she wants to set test criteria.

    - **Action:** User sets test criteria
    - **Reply:** Input is sent for processing

- **Generate and Execute Test Cases:** TestCube will generate test cases for the given code using the selected tools. If the user sets some range of for the test cases, then TestCube will show test cases within that range. After generating test cases, TestCube will automatically execute the test cases (if the user does not turn this feature off)
    - **Action:** TestCube generates test cases.
    - **Reply:** Test cases are sent for execution.

    - **Action:** TestCube executes generated test cases.
    - **Reply:** Execution status is kept as report

- **Report Generation:** Upon execution, TestCube will check if any test cases caused the program to face any error or warning. Those cases will be mentioned in the report. If the expected behavior is provided, then TestCube will also check for cases where the code fails to behave according to the expected behavior.
    - **Action:** TestCube generates report
    - **Reply:** Report is sent to make result
- **Store and View:** After the report is ready, users can view the report. Server will store the input and the report in the database. Users can view or download their inputs and reports whenever they want.
    - **Action:** TestCube stores results to database

- **Reply:** Result is stored in the database in user's usage document

- **Action:** User wants to view result
- **Reply:** result will be brought from database

- **Re-run Model:** The machine learning model will train itself with all the user inputs and outputs in a previously set day. After training, the date of next training will be set.
  - **Action:** Training day has arrived
  - **Reply:** Model is trained again.

Use Case Diagram level: 1.1

Name: Sign up or log in
Primary Actor: User, Server
Secondary Actor: Email, Database

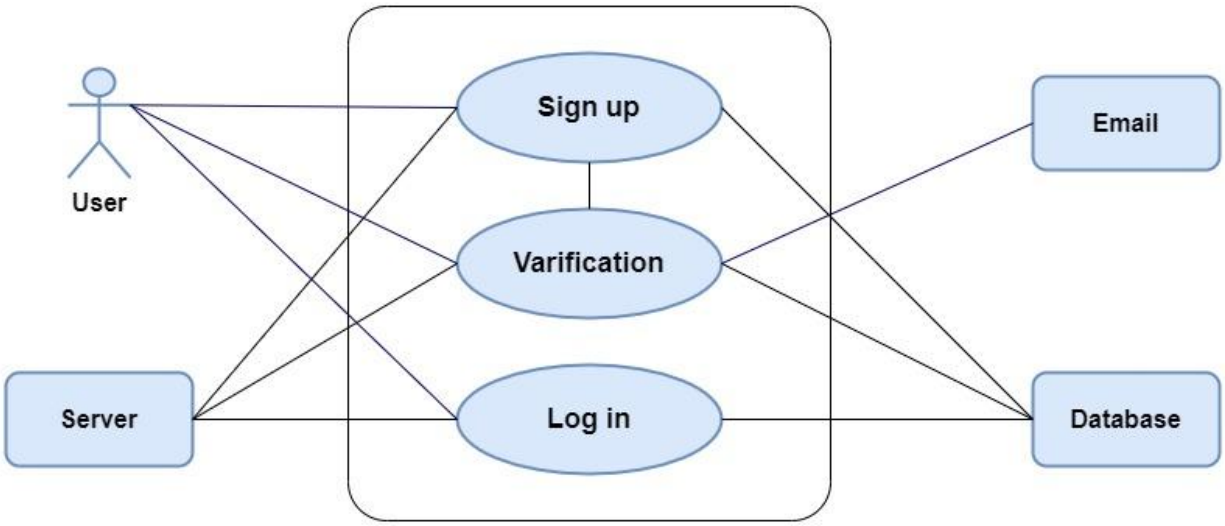


Figure 3: **Signup or login** (usecase-1.1)

Description of Use Case Level-1.1: This module handles user login and signup features.

- **Sign up:** This sub module will handle account sign up. Users will provide their full-name, username, email address and password to sign up.
    - **Action:** User provides information to sign up
    - **Reply:** wait for verification.

- **Verification:** This sub module will verify user's sign up**.** Upon requesting sign up, users will receive an OTP to verify their sign up attempt.
    - **Action:** OTP sent to user to verify account
    - **Reply:** User receives OTP

- **Log in:** This sub module will be used for logging in. Users have to login to the system using their account's username/email and password.
    - **Action:** User will provide mail address and password.
    - **Reply:** If valid, the user will be logged in.

Use Case Diagram Level: 1.2

Name: Upload and Set Test Criteria
Primary Actor: User, Server

Secondary Actor: Database

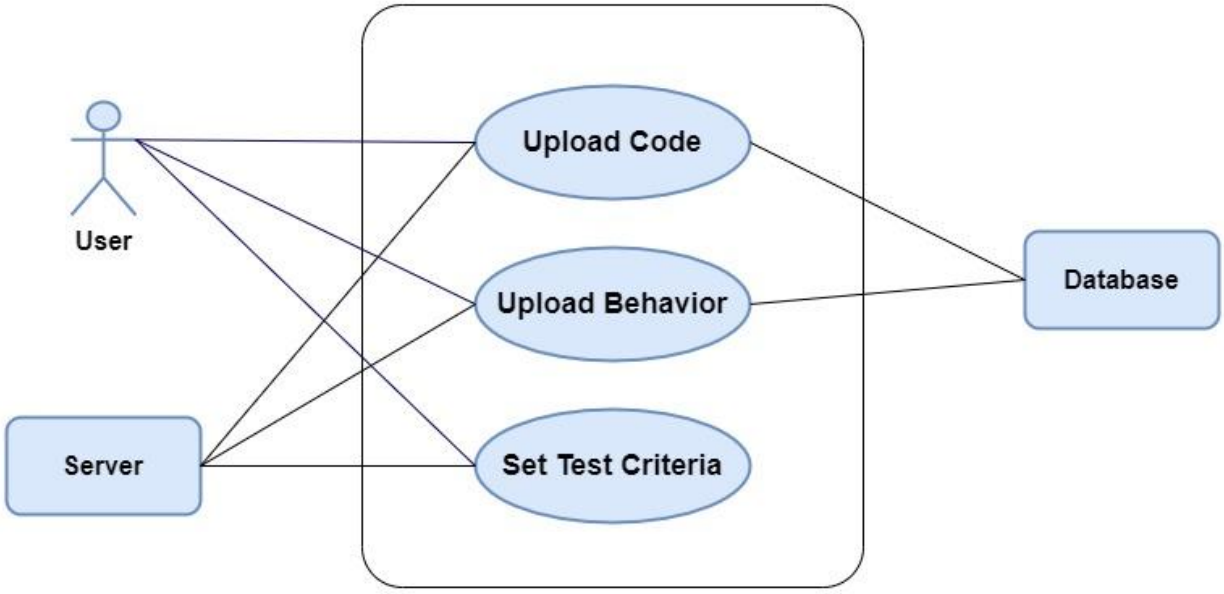


Figure 4: **Upload and Set Test Criteria** (usecase-1.2)

Description of Use Case Level-1.2: This module will handle input upload and setting test criteria

- **Upload Code:** This sub module will let users upload their code that they want to generate test cases for. Users can write their code in a text box or directly upload a file.
    - **Action:** Users will upload their codes.
    - **Reply:** Input received.

- **Upload Behavior:** This sub module will let users upload the behavior of the input code. Users can choose not to upload any behavior too. Users can write their behavior in a text box or directly upload a file. Behavior has to be in JSON format.
    - **Action:** User will upload expected behavior.
    - **Reply:** Input received.

- **Set Test Criteria:** This sub module will let users set some test criterias on their usage. After uploading, users can set ranges in which the test cases will

be generated. They can also choose which tool(s) they want to use. If a user wants to turn off execution of the generated test cases, they can do so.

- **Action:** User sets test criterias
- **Reply:** Input is sent for processing

Use Case Diagram Level: 1.3

Name: Generate and Execute Test Cases
Primary Actor: User, Server

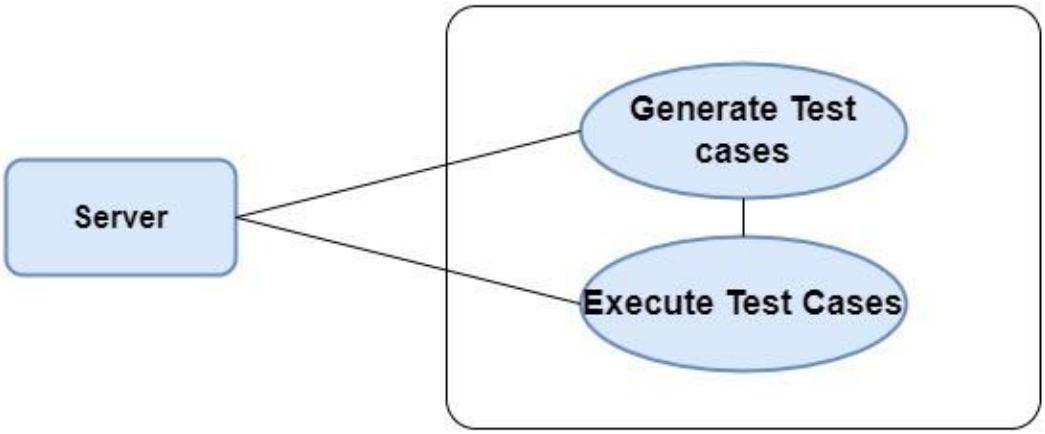


Figure 5: **Generate and Execute test cases** (usecase-1.3)

Description of Use Case Level-1.3: This module handles test cases generation and their execution.

- **Generate Test Cases:** This sub-module is responsible for generating test cases. TestCube will generate test cases for the given code using the selected tools.
  - **Action:** TestCube generates test cases.
  - **Reply:** Test cases are sent for execution.
- **Execute Test Cases:** This sub-module is responsible for executing the generated test cases if the user does not turn off execution of test cases.
  - **Action:** TestCube executes generated test cases.

- **Reply:** Execution status is kept as report

Use Case Diagram Level: 1.4

Name: Report Generation
Primary Actor: Server

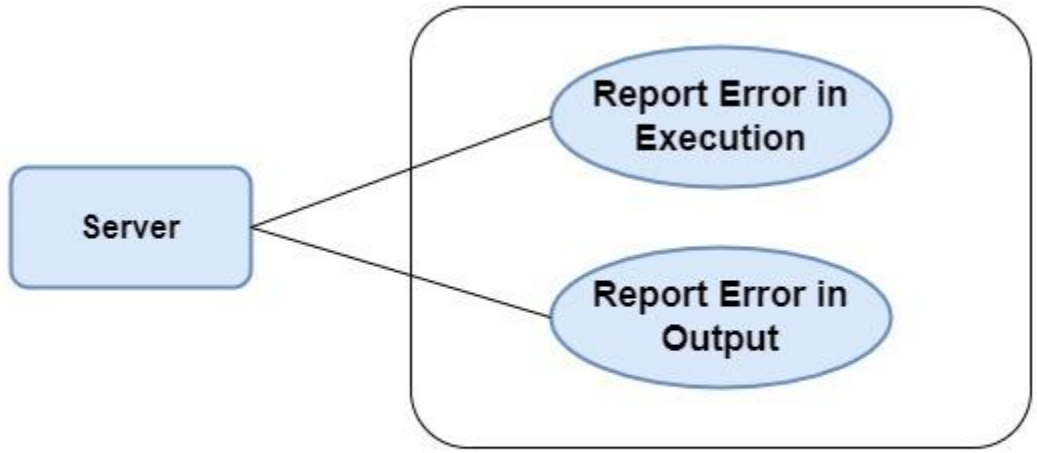


Figure 6: **Report Generation** (usecase1.4)

Description of Use Case Level-1.4: This module is responsible for generating the report. In general, the report will contain all the generated test cases.

- **Report Error in Execution:** This sub-module will add errors/ warnings or unhandled exceptions that are faced during execution of the test cases to the report.
  - **Action:** TestCube finds error in execution.
  - **Reply:** Report error in execution.

- **Report Error in Output:** This sub-module will check for any mismatch in output of provided behavior and test case execution, if the behavior is provided.

- **Action:** TestCube finds error in output.
- **Reply:** Report error in output.

Use Case Diagram Level: 1.5

Name: Store and view
Primary Actor: User
Secondary Actor: Server, Database

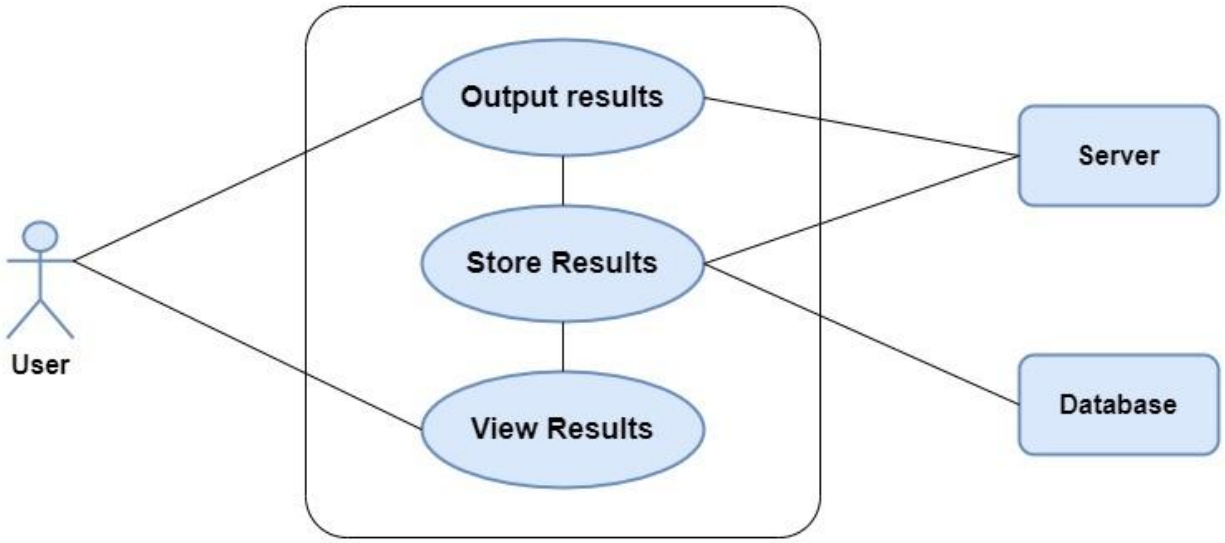


Figure 7: **Store and View** (usecase-1.5)

Description of Use Case Level-1.5: This module handles output store and view.

- **Output Results:** This sub-module will output the generated report to the user.
  - **Action:** TestCube shows output to the user
  - **Reply:** User sees output

- **Store Results:** This sub-module will make the server store the inputs and reports in the database.
  - **Action:** TestCube stores results to database

- **Reply:** Result is stored in the database in user's usage document

- **View Results:** This sub-module will help users view all their previous inputs and reports
  - **Action:** User wants to view result
  - **Reply:** result will be brought from database

Use Case Diagram level: 1.6

Name: Re-Train Model
Primary Actor: Server
Secondary Actor: Database

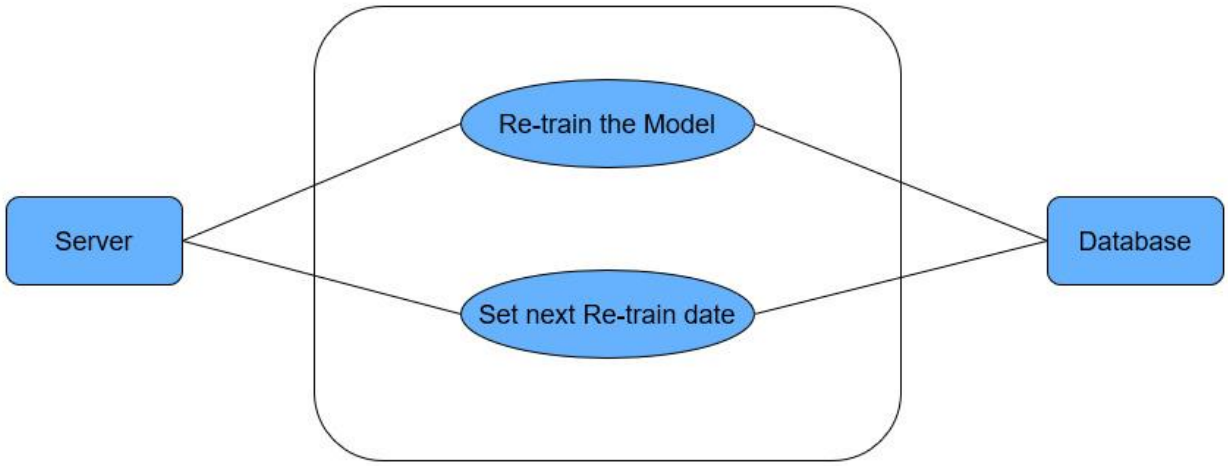


Figure 8: Re-run the Model (usecase-1.6)

Description of Use Case Level-1.6: This module will handle re-run of the machine learning model. The model will be trained again using user's codes and test cases in a previously set day.

- **Action:** Training day has arrived
- **Reply:** Model is trained again.

4.4 Activity Diagram

Definition of Activity Diagram:

Activity diagram is an important behavioral diagram in UML diagram to describe dynamic aspects of the system. Activity diagram is essentially an advanced version of flowchart that models the flow from one activity to another activity.

Activity Diagram level: 1

Name: TestCube (Detailed)
Reference: Use Case Level 1

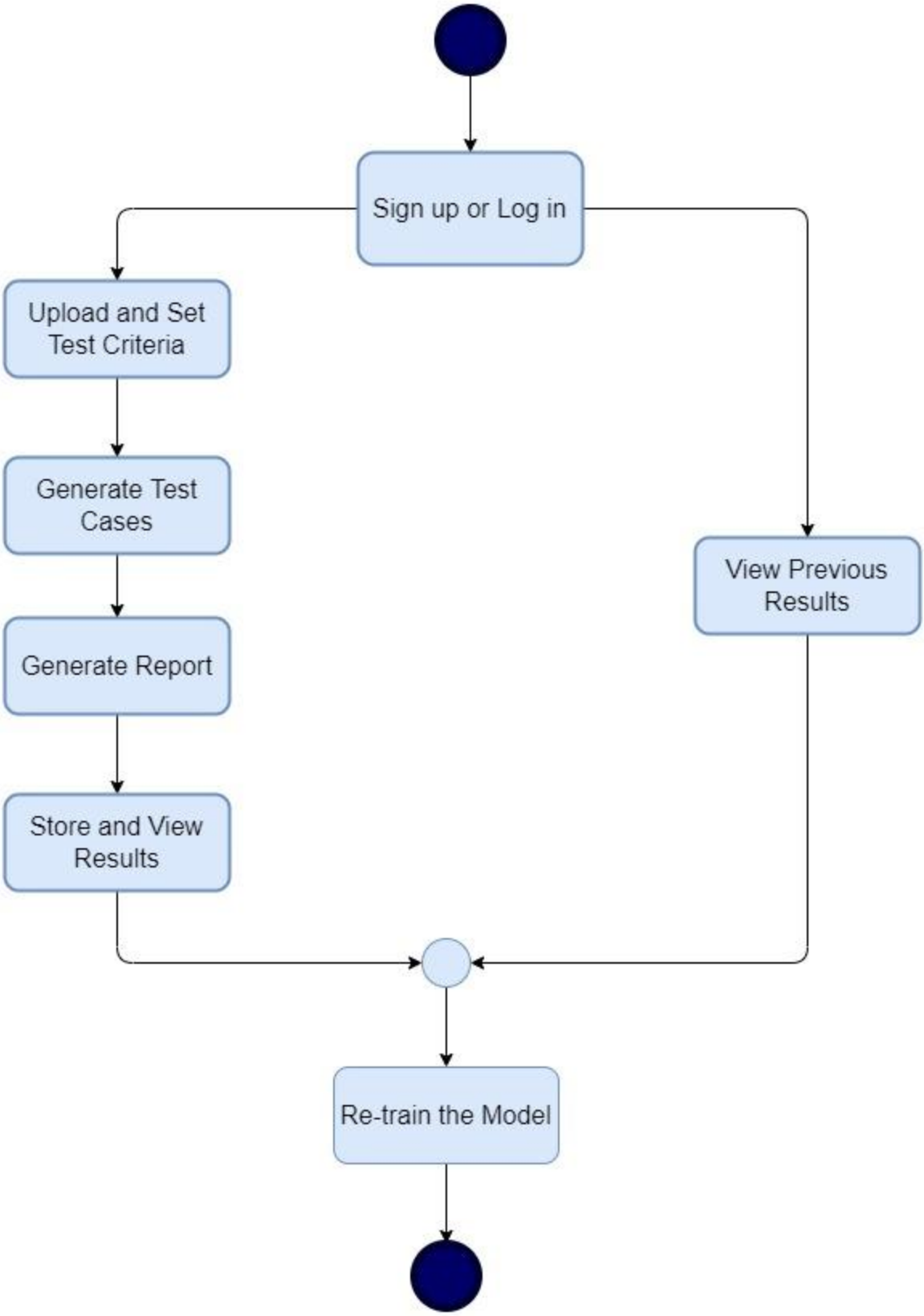


Figure 9: **TestCube** (activity-1)

Activity Diagram level: 1.1

Name: Sign up or Log in
Reference: Use Case Level 1.1

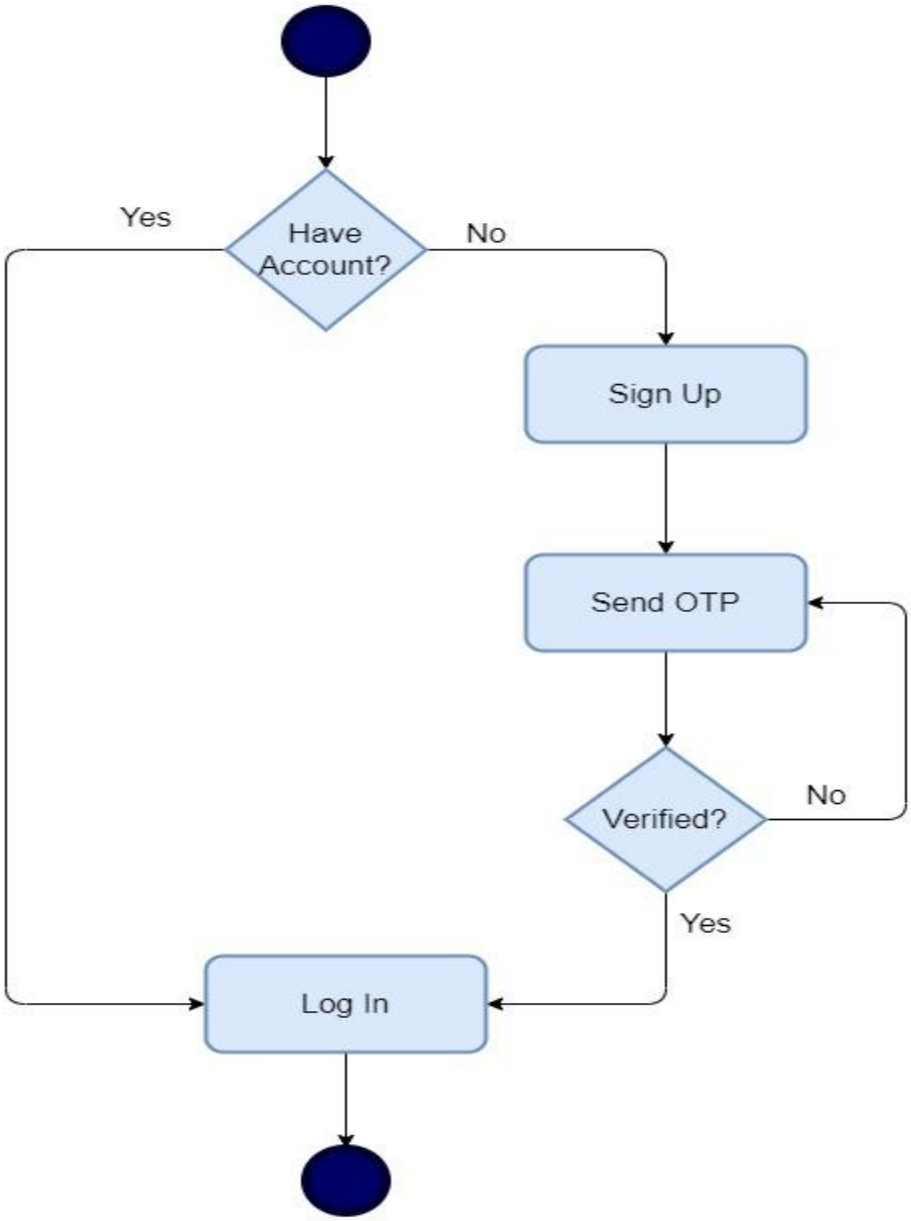


Figure 10: **Login or Signup** (activity-1.1)

Activity Diagram level: 1.2

Name: Upload and Set Test Criteria
Reference: Use Case Level 1.2

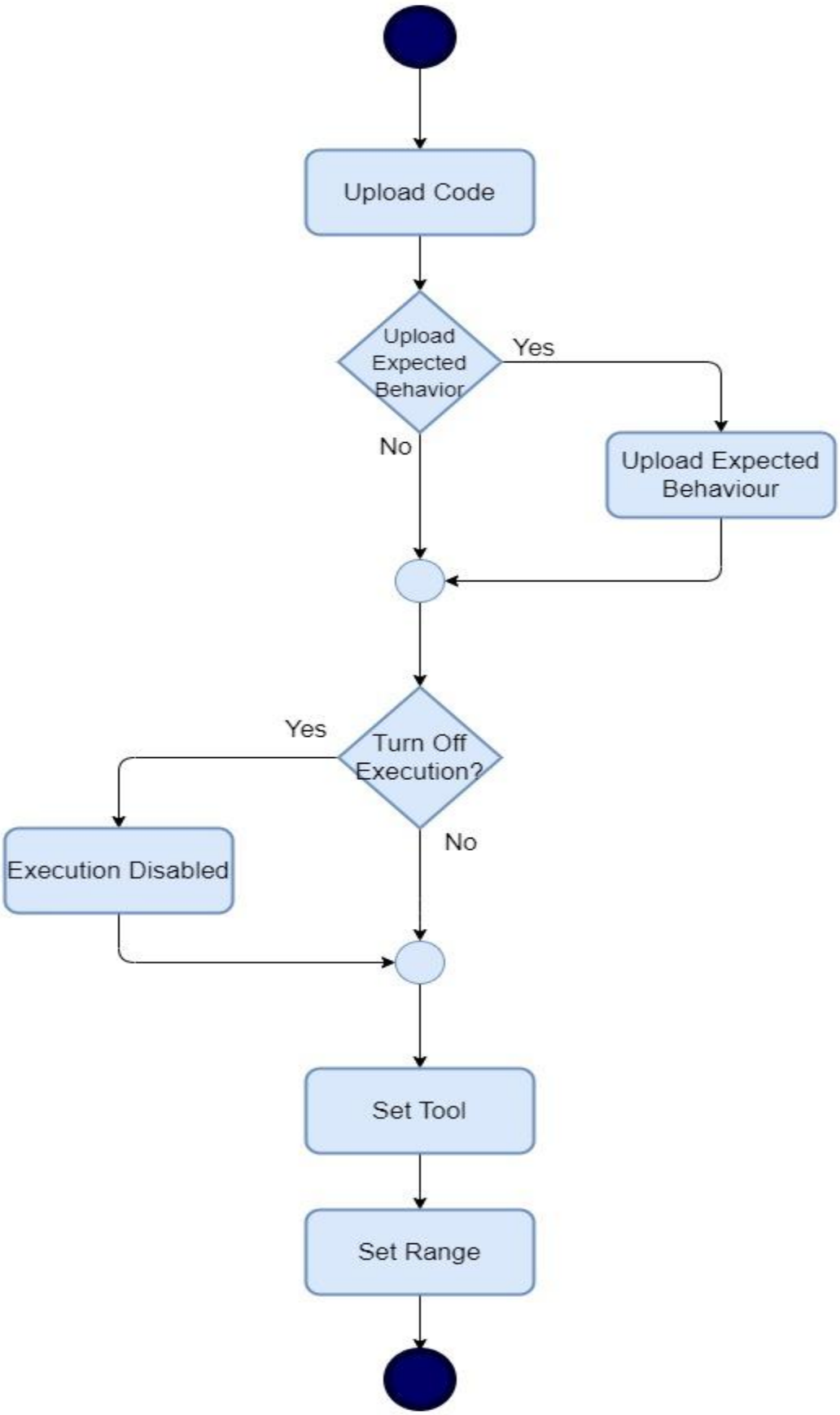


Figure 11: **Upload and Set Test Criteria** (activity-1.2)

Activity Diagram level: 1.3

Name: Generate and Execute Test Cases
Reference: Use Case Level 1.3

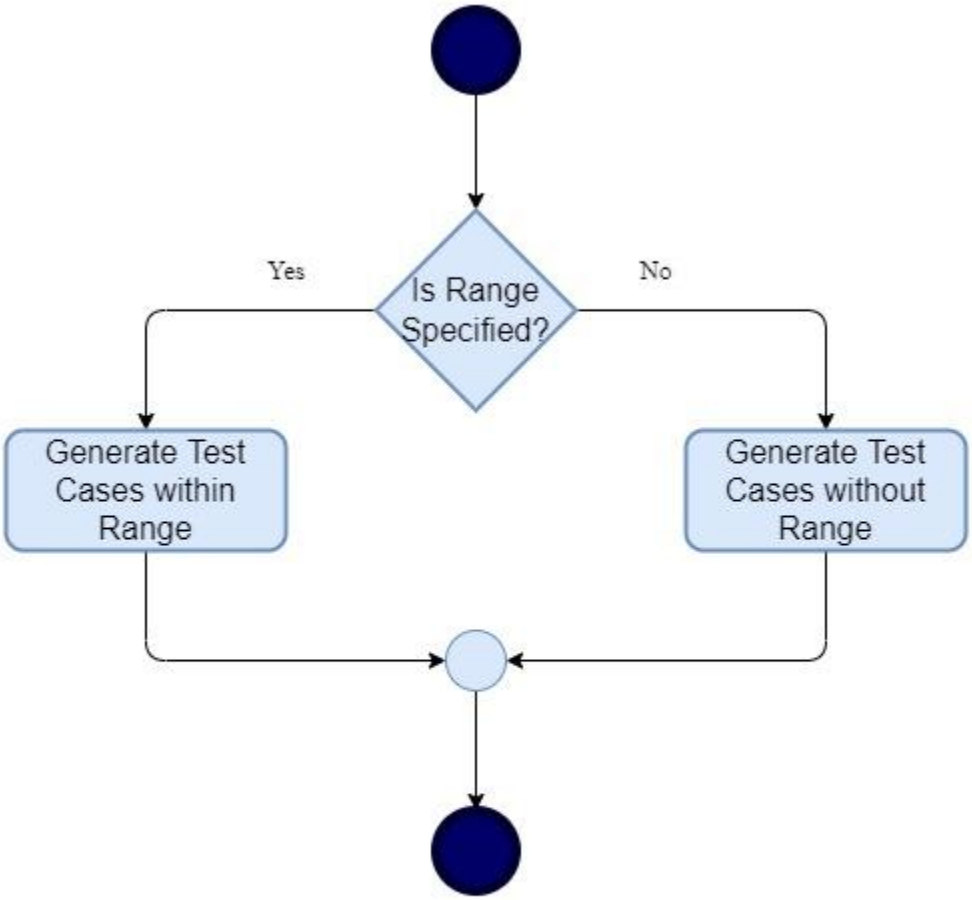


Figure 12: **Generate Test Case** (activity-1.3)

Activity Diagram level: 1.4

Name: Report Generation
Reference: Use Case Level 1.4

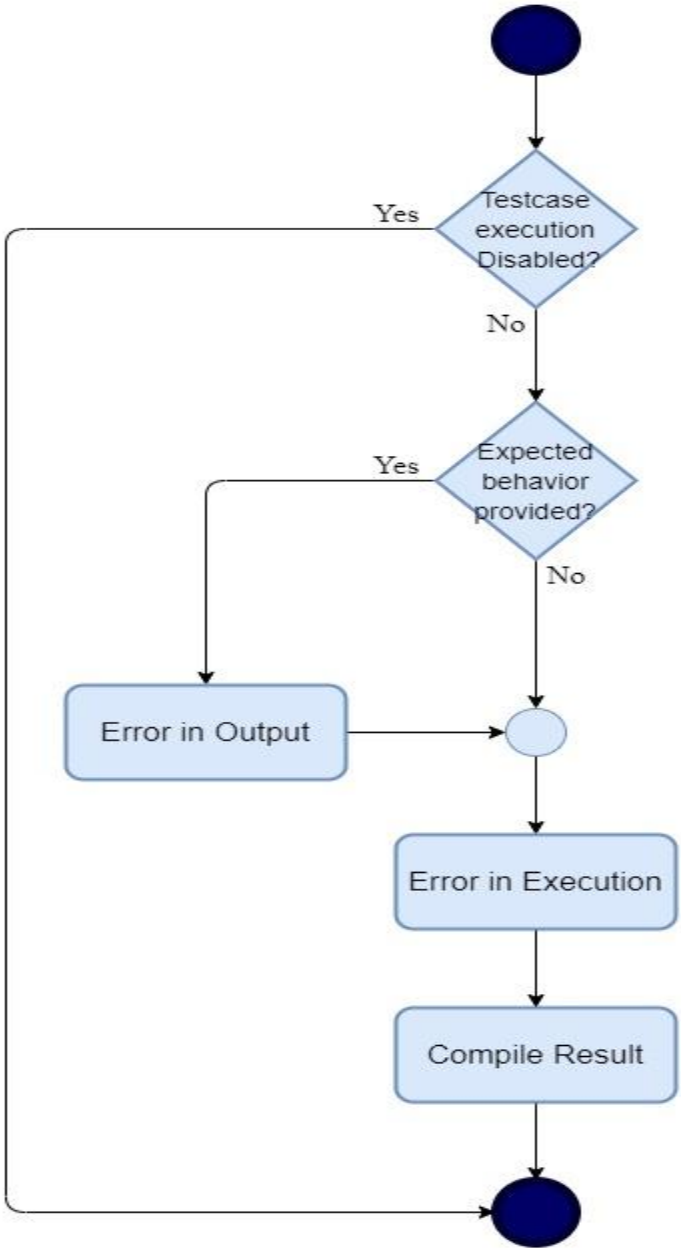


Figure 13: **Execute Test Case and Compile Result** (activity-1.4)

Activity Diagram level: 1.5

Name: Store and View Result
Reference: Use Case Level 1.5

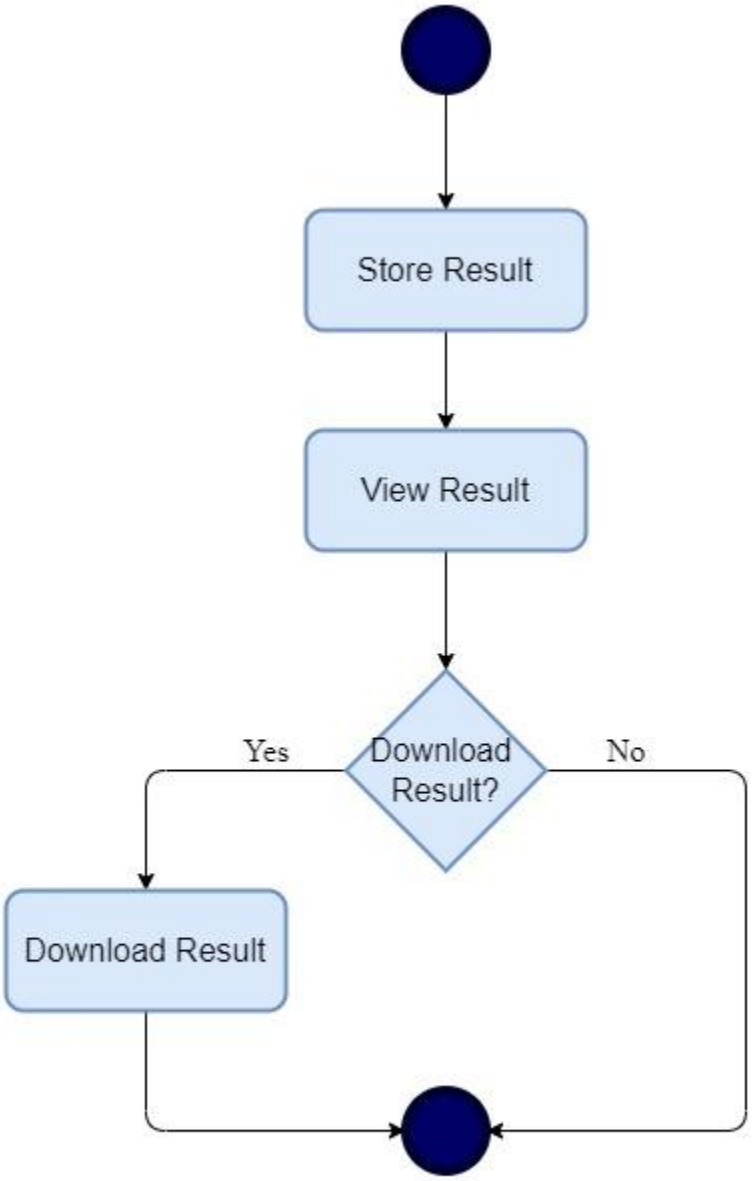


Figure 14: **Store and View Result** (activity-1.5)

Activity Diagram level: 1.6

Name: Re-Train Model
Reference: Use Case Level 1.6

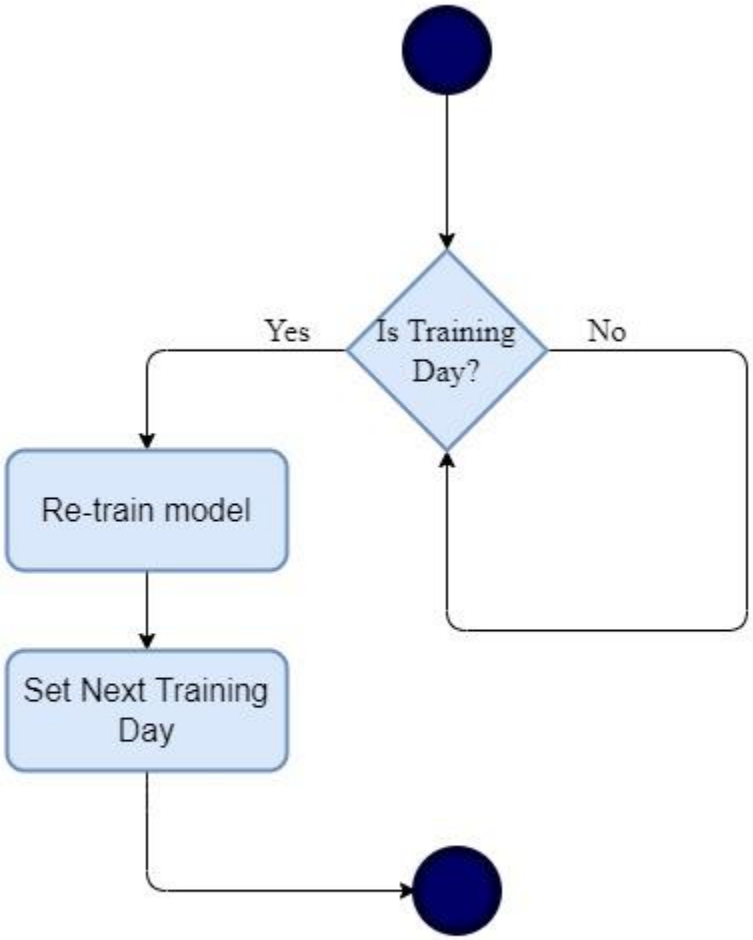


Figure 16: **Re-train Model** (activity-1.6)

4.5 Swimlane Diagram

Definition of Swimlane Diagram:

A swimlane diagram is a type of flowchart that delineates who does what in a process. Using the metaphor of lanes in a pool, a swimlane diagram provides clarity and accountability by placing process steps within the horizontal or vertical “swimlanes” of a particular employee, workgroup, or department. It shows connections, communication and handoffs between these lanes, and it can serve to highlight waste, redundancy and inefficiency in a process.

SID (Swimlane ID): 1
Name: TestCube(Detailed)
Reference: Use Case and Activity Diagram level 1

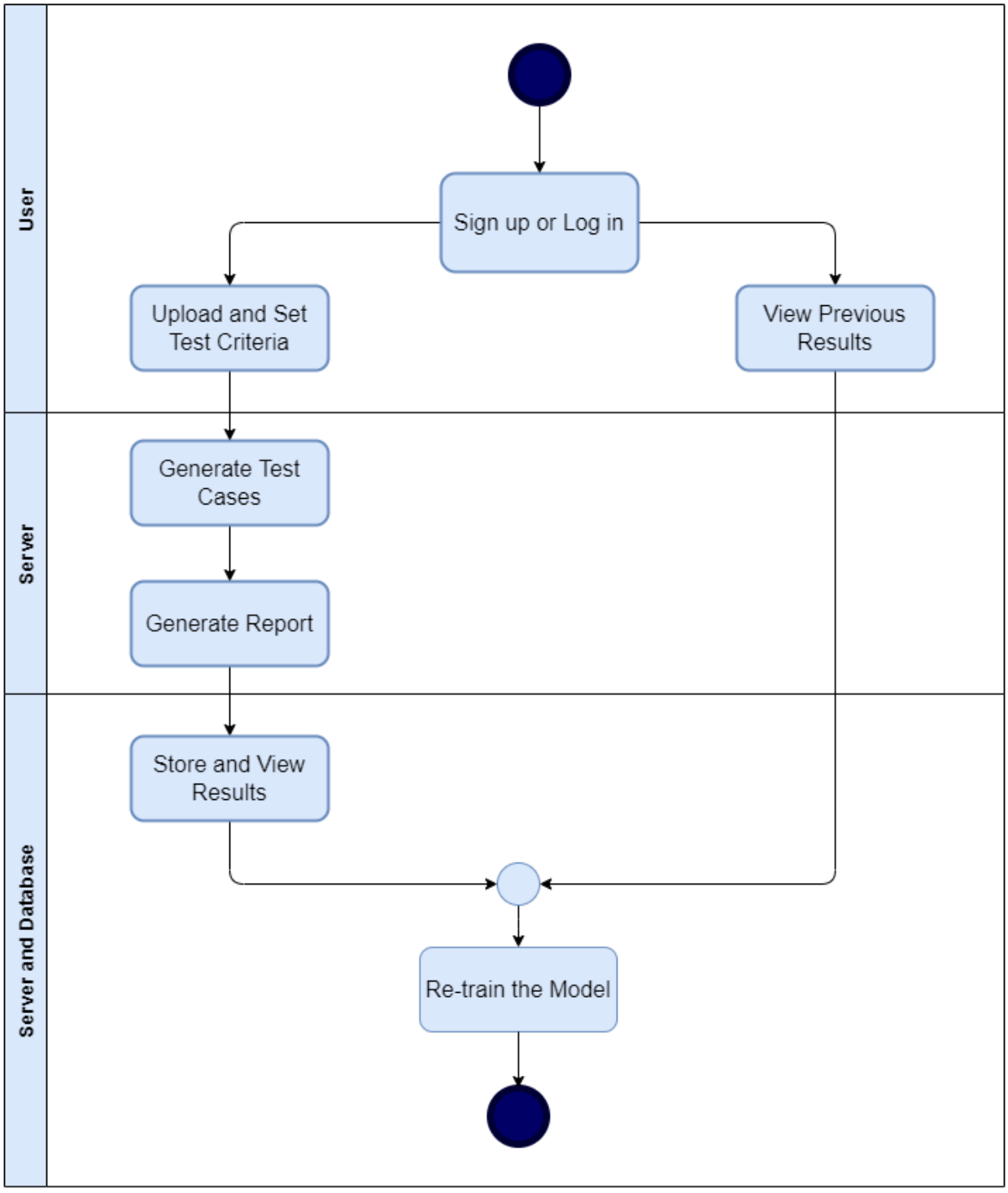


Figure 17: **TestCube** (swimlane-1.1)

SID (Swimlane ID): 1.1
Name: Sign up or Log in
Reference: Use Case and Activity Diagram level 1.1

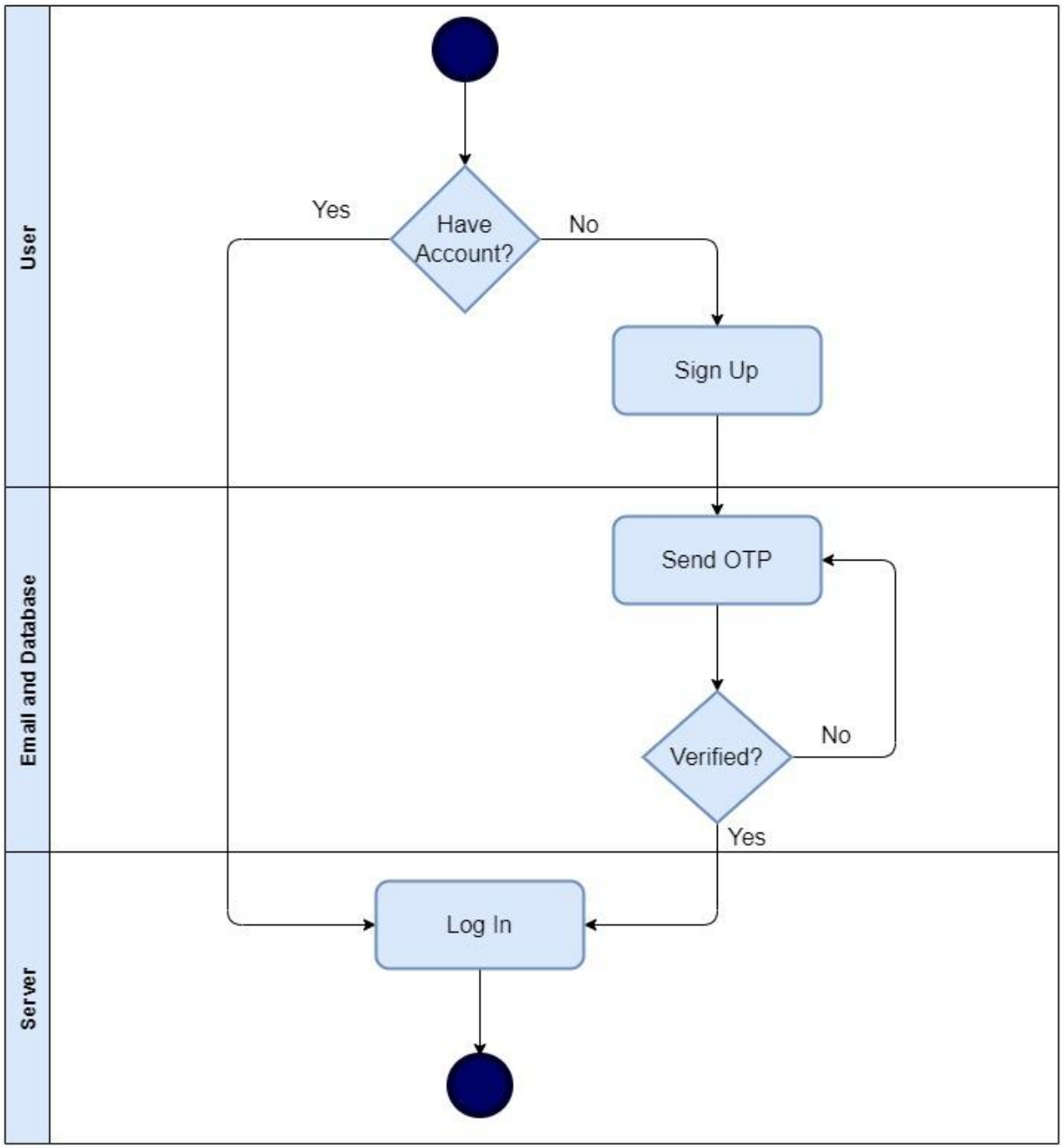


Figure 18: **Login or Signup** (swimlane-1.1)

SID (Swimlane ID): 1.2
Name: Upload Code and Test Criteria
Reference: Use Case and Activity Diagram level 1.2

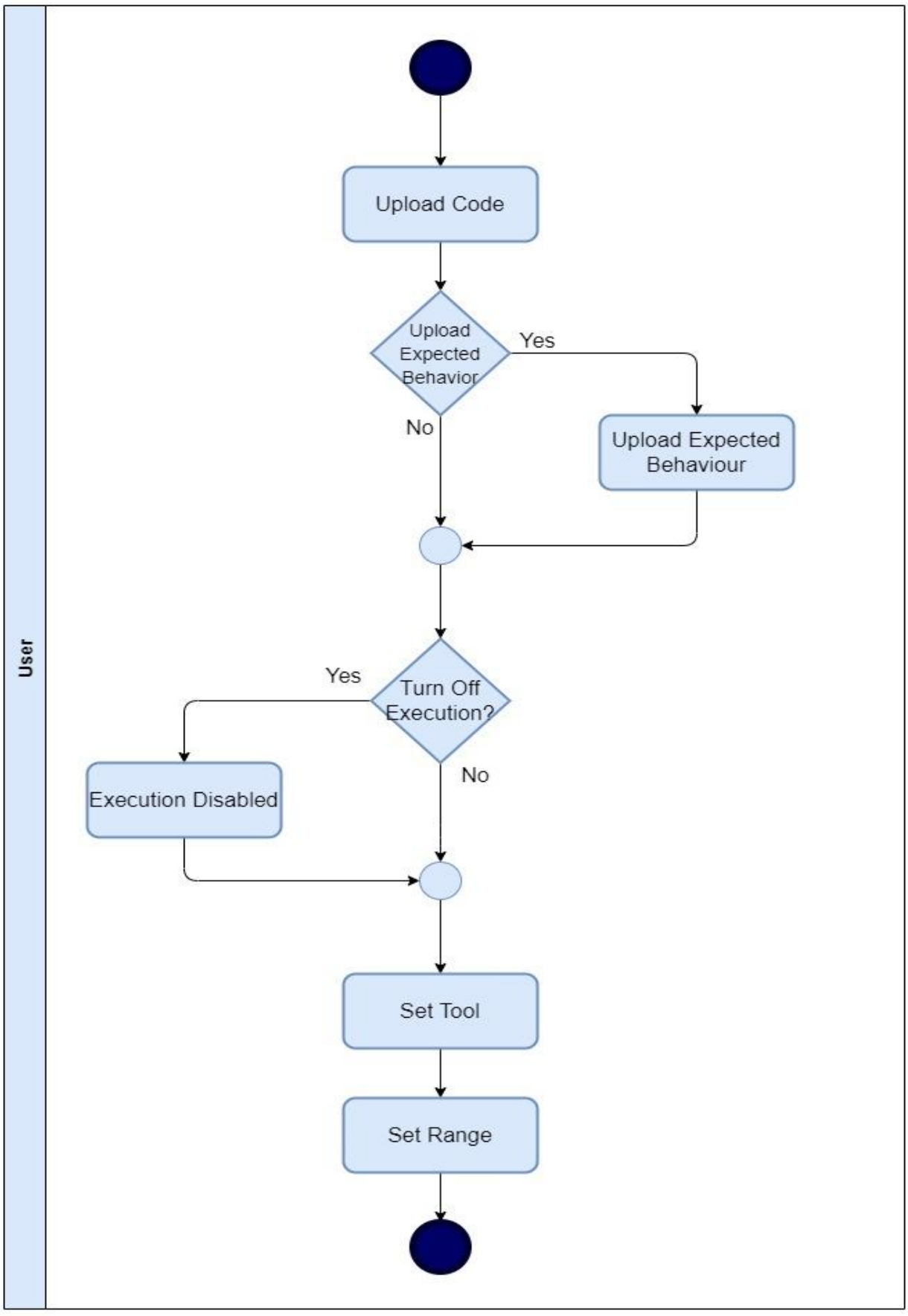


Figure 19: **Upload and Set Test Criteria** (activity-1.2)

SID (Swimlane ID): 1.3
Name: Generate Test Cases
Reference: Use Case and Activity Diagram level 1.3

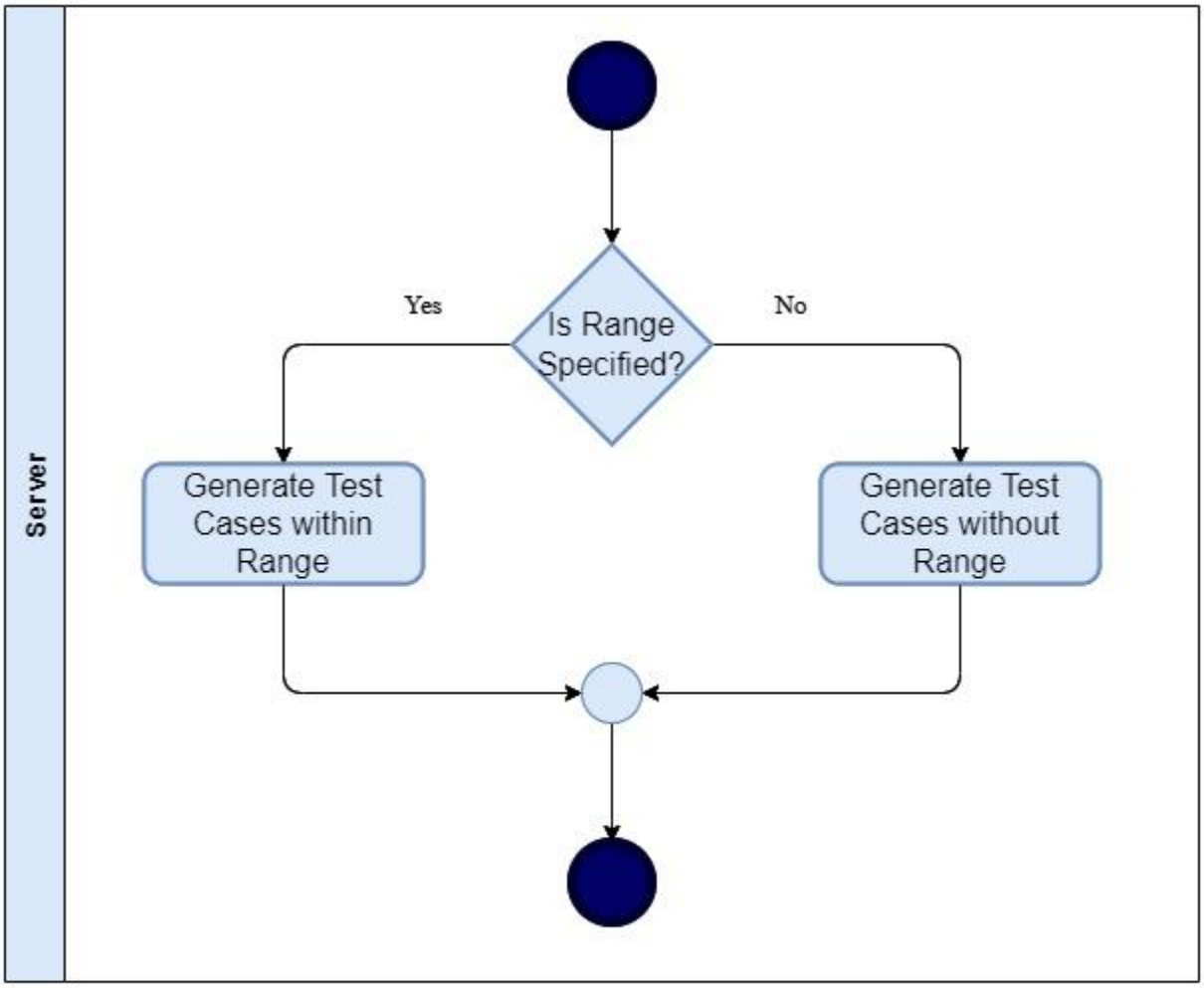


Figure 20: **Generate Test Case** (swimlane-1.3)

SID (Swimlane ID): 1.4
Name: Report Generation
Reference: Use Case and Activity Diagram level 1.4

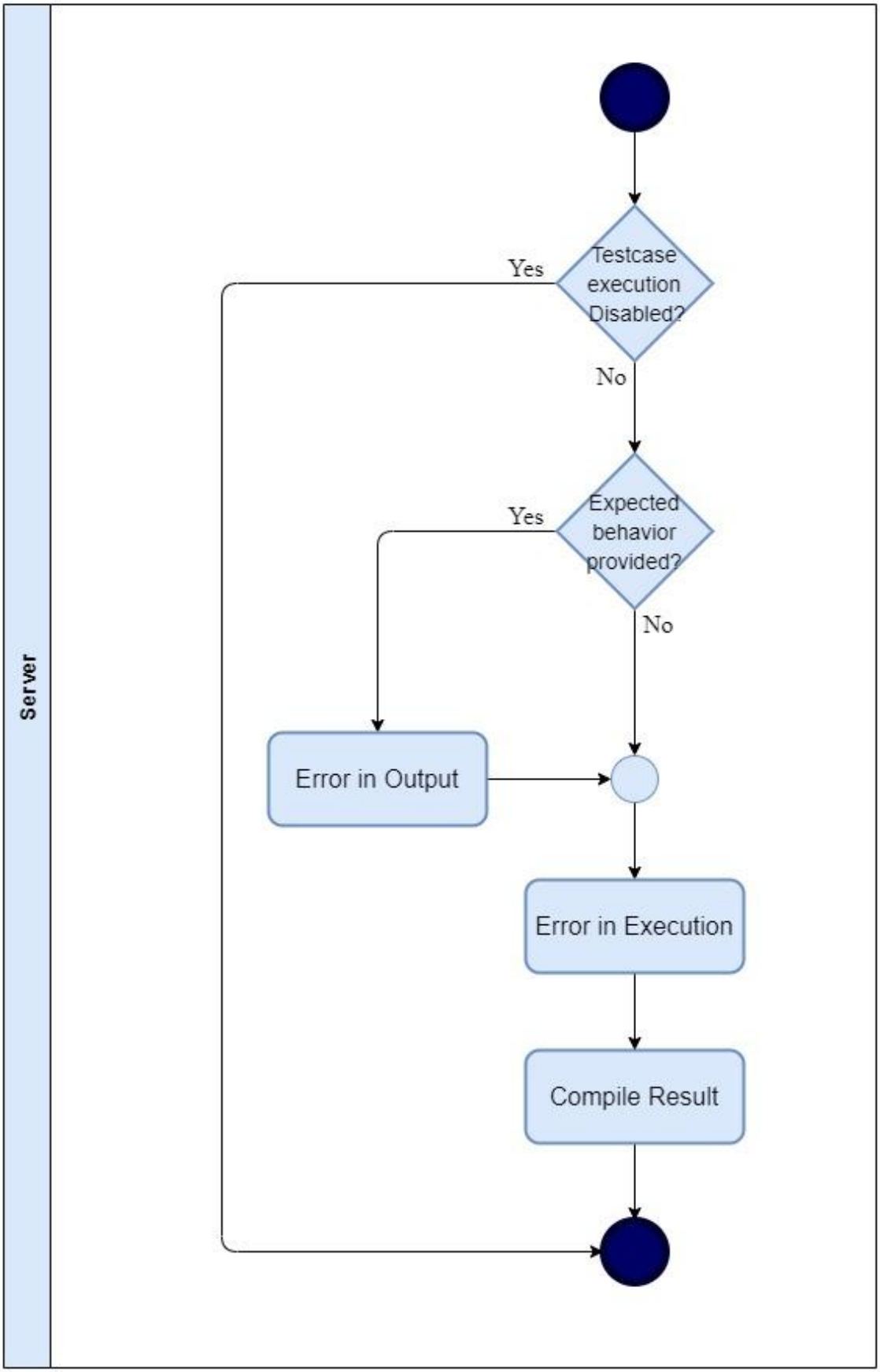


Figure 21: **Execute Test Case and Compile Result** (swimlane-1.4)

SID (Swimlane ID): 1.5

Name: Store and View Result

Reference: Use Case and Activity Diagram level 1.5

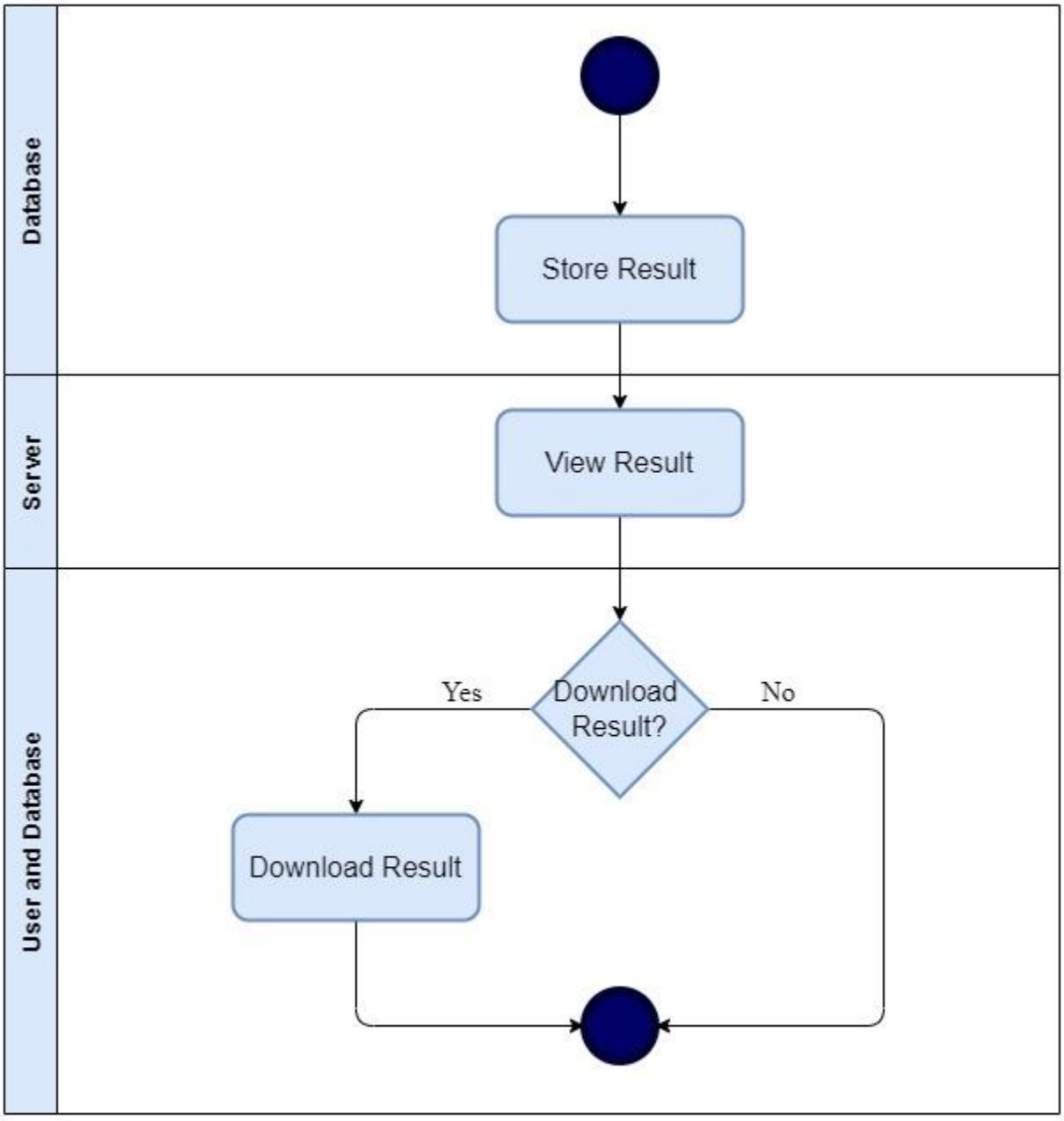


Figure 22: **Store and View Result** (swimlane-1.5)

SID (Swimlane ID): 1.6
Name: Re-Train Model
Reference: Use Case and Activity Diagram level 1.6

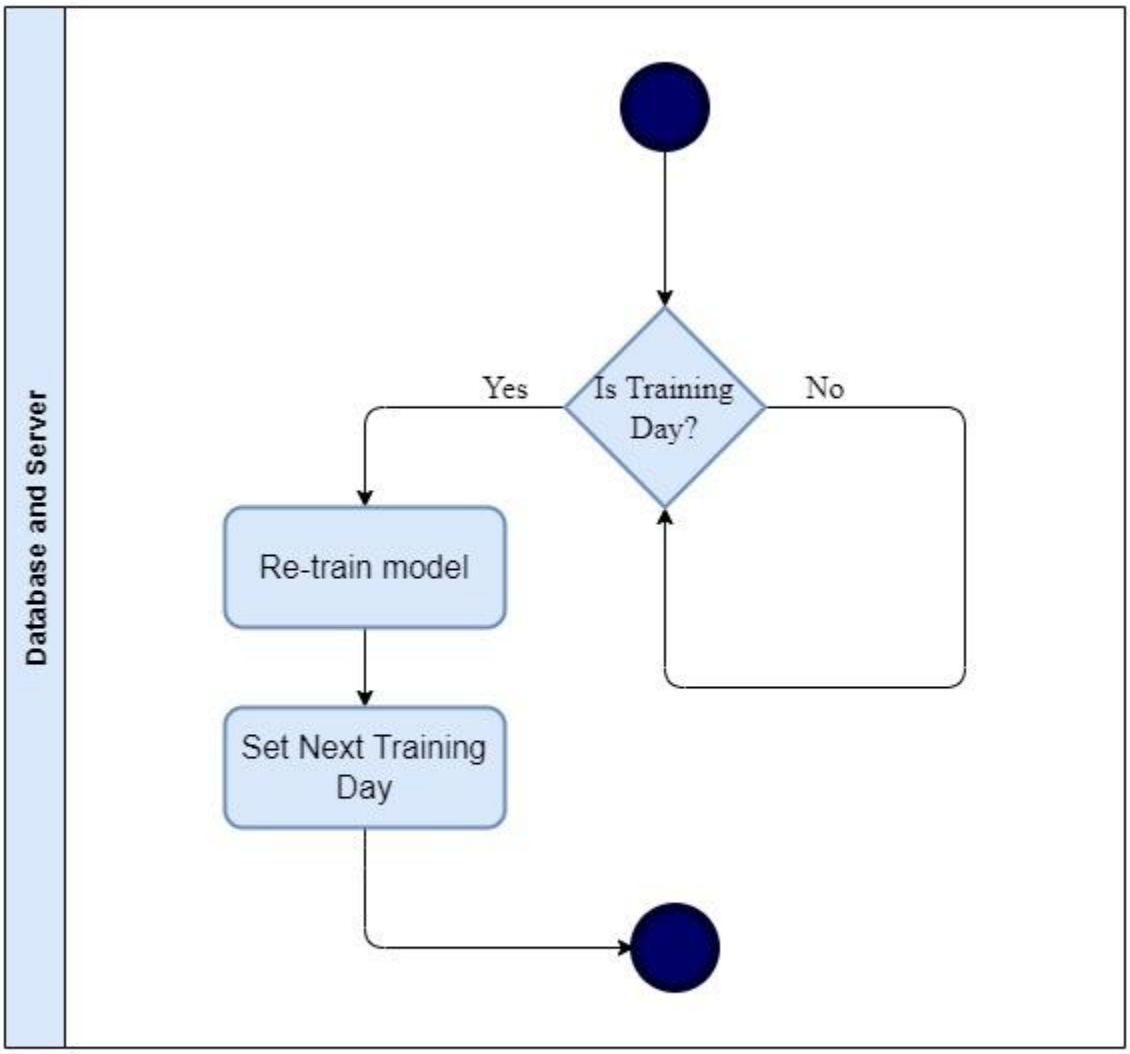


Figure 23: **Re-train Model** (swimlane-1.6)

5.
6. Data-Based Modeling of TestCube

5.1 Data Modeling Concept

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) identifies 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 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 Data Object Identification

We identified all the nouns whether they are in problem space or in solution space from our story:

No.	Noun	Problem/ Solution Space	Attributes	Remark
1	TestCube	p		Operation
2	Test cases	s		Stored Data
3	Randoop	p		Operation
4	Machine learning based solution	p		Operation
5	Input codes	s		Stored Data

6	Input id	s		Stored Data
7	**User**	s	10, 11, 12, 13	Stored Data
8	Result	p		Operation
9	Tools	p		Operation
10	Full-Name	s		Stored Data
11	Username	s		Stored Data
12	Email	s		Stored Data
13	Password	s		Stored Data
14	OTP	p		Operation
15	Account	p		Operation
16	**Input**	s	5, 6, 17, 18, 19	Stored Data
17	Expected Behavior	s		Stored Data
18	Selected tools	s		Stored Data
19	Range of test cases	s		Stored Data
20	Auto Execution Status	s		Stored Data
21	**Report**	s	6, 2, 23, 24	Stored Data
22	Auto Execution	p		Operation
23	Errors	s		Stored Data
24	Warning	s		Stored Data
26	Re-training day	p		
27	dataset	p		

5.2.2 Selected Data Objects

1. User
2. Input
3. Report

5.2.3 Analysis

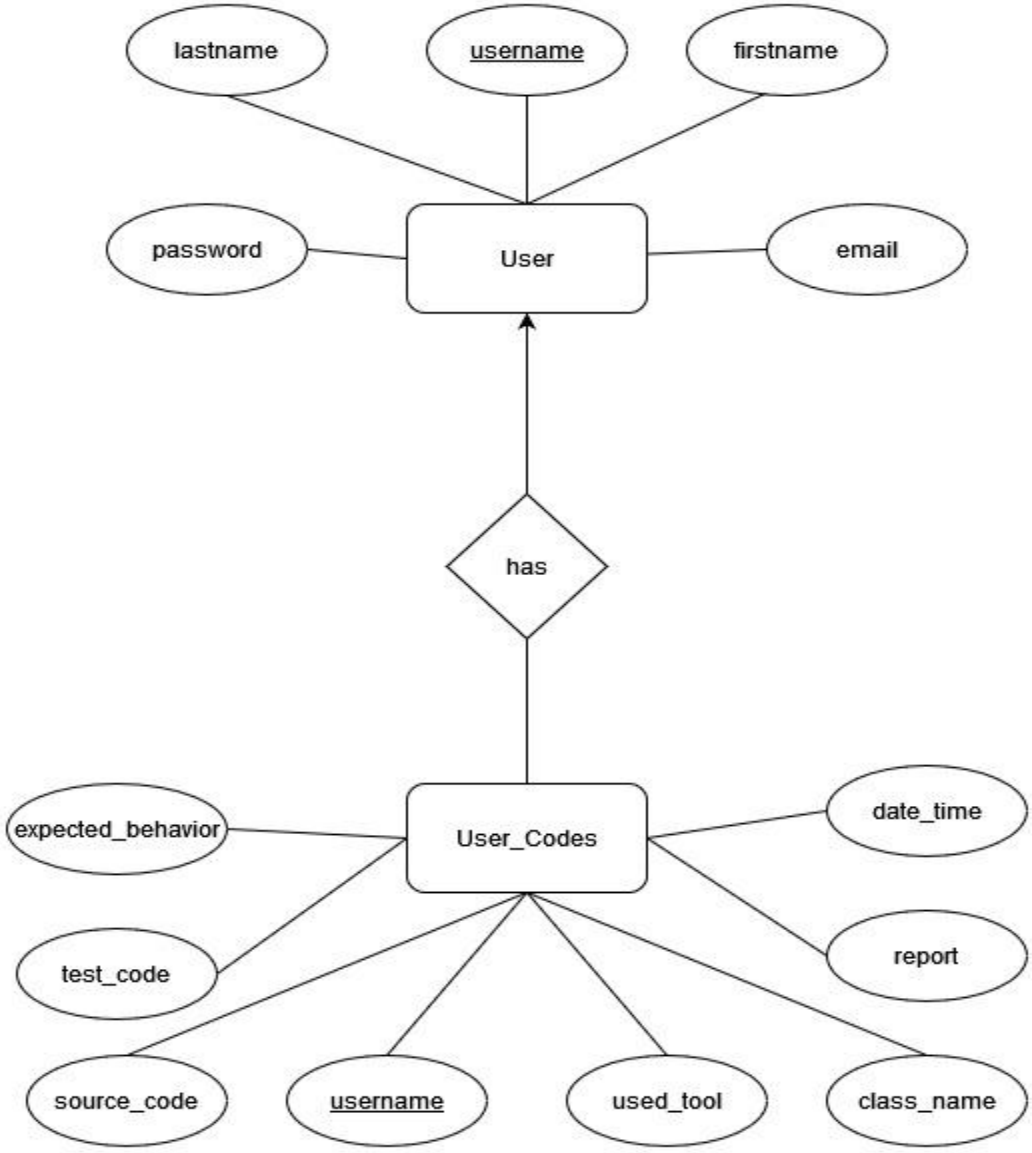
In our program, input and report have a one-to-one relationship and we always have to show input and report together. In order to reduce query cost, we can merge input and report and make a new object called “Usage”.

That leaves us with final data objects:

1. User
2. Usage

Since we are using Sqlite, the database design will be as follows:

Database Design



Schema Diagram:

Data Object	Attribute	Type	Size
User	-username -firstname -lastname -email -password	CHARACTER CHARACTER CHARACTER TEXT TEXT	20 20 20
User_codes	-username -class_name -source_code -test_code -expected_behavior -report -date_time -used_tool	CHARACTER CHARACTER TEXT TEXT TEXT TEXT DATE CHARACTER	20 20 20 20

6. Class Based Modeling of TestCube

This chapter is intended to describe class based modeling of TestCube

6.1 Class Based Modeling Concept

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

Potential Nouns to become a class

Serial	Noun
1.	TestCube
2.	Test cases
3.	Machine learning based solution

4.	Input code
5.	**User**
6.	**Result**
7.	**Tools**
8.	**Test case Generator**
9.	**MLTestcaseGenerator**
10.	**Email**
11.	**Randoop**
12.	**Account**
13.	**Input**
14.	Expected Behavior
15.	Selected tools
16.	Range of test cases
17.	**Auto Execution Status**
18.	**Report**
19.	Auto Execution
20.	**Errors**
21.	**Warning**
22.	**Re-training day**
23.	**Dataset**

Verb List of TestCube:

Serial	**Verb**
1.	Generate Test cases

2.	Execute Test cases
3.	Store input
4.	Store output
5.	Create account
6.	Send OTP
7.	Verify account
8.	Upload code
9.	Upload expected behavior
10.	Set test criteria
11.	Set test range
12.	Turn off auto execution
13.	Generate report
14.	Show report
15.	View report
16.	View output
17.	Download output
18.	Select tool
19.	Re-train the model
20.	Set next train date

6.2 General classification

Candidate classes were then characterized in seven general classes. The seven general characteristics are as follows:

1. External entities
2. Things
3. Events
4. Roles
5. Organizational units
6. Places
7. Structures

Serial	**Noun**	**General Classification**
1.	TestCube	2
2.	Test cases	2,3
3.	Machine learning based solution	2,3
4.	Input code	3
5.	**User**	**4,5,7**
6.	**Result**	**4,5,7**
7.	**Tools**	**2,4**
8.	**Test case Generator**	**4,5,7**
9.	**MLTestcaseGenerator**	**4,5,7**
10.	**Email**	**4,5,7**
11.	**Randoop**	**2,4**
12.	**Account**	2
13.	**Input**	**4,5,7**
14.	Expected Behavior	2
15.	Selected tools	2
16.	Range of test cases	2
17.	**Auto Execution Status**	**2,4**
18.	**Report**	**4,5,7**
19.	Auto Execution	2

20.	**Errors**	2
21.	**Warning**	2
22.	**Re-training day**	**4,5,7**
23.	**Database**	**2,4,7**

6.3 Selection Criteria

The candidate classes are then selected as classes by six Selection Criteria. A candidate class generally becomes a class when it fulfills around three characteristics.

1. Retain information
2. Needed services
3. Multiple attributes
4. Common attributes
5. Common operations
6. Essential requirements

Potential general classified nouns to become a class after selection criteria:

Serial	**Noun**	**Selection Criteria**
1.	User	1,2,3,4,5,6 (selected)
2.	Result	1,2,3,4,5,6 (selected)
3.	Tools	
4.	Test case Generator	1,2,3,4,5,6 (selected)
5.	MLTestcaseGenerator	1,2,3,4,5,6 (selected)
6.	Email	6(selected)
7.	Randoop	6(selected)

8.	Input	1,2,3,4,5,6 (selected)
9.	Auto Execution Status	
10.	Report	1,2,3,4,5,6 (selected)
11.	Re-training day	
12.	Database	1,2,3,4,5,6(selected)

6.4 Selected Classes:

1. User
2. Input
3. Result
4. Report
5. TestCaseGenerator
6. MLTestcaseGenerator
7. Randoop
8. Email
9. Database

6.5 Analysis:

As we are working on a cloud based web server application, we will need some controller classes for handling the http requests from the front end to back end. Also we find some common attributes and methods of Report class and Result class. So, we merge it with the Output class. Again TestCaseGenerator will be an abstract class and MLTestCaseGenerator and Randoop will have a “is-A” relation with this class.

So, final classes are:

1. User
2. Input
3. InputController
4. Output

5. OutputController
6. Database
7. TestCaseGenerator
8. MLTestCaseGenerator
9. Randoop
10. EmailSender

6.6 Attribute & Method Identification:

Serial	Class Name	Attribute	Method
1.	User	-fullname -username -email -password	+login() +create_account() +add_user() +checkDuplicateEmail() +checkDuplicateUserna me()
2	InputController	-database	+isValidInput() +parseInput() +storeInput() +storeCriteria()
3.	Input	-input_id -input_code -expected_behavior -selected_tools -range_of_test_cases	+getter() +setter()
4.	OutputController	-database	+generateResult() -storeResult() +showResult()
5.	Output	-test_cases -input -auto_execution_stat	+getter() +setter()

		us	
6.	Database		+connect() +isConnected() +update() +read() +write() +delete() +verifyInDatabase()
7.	TestCaseGenerator		+generateTestCases() +executeTestCases() +reportGeneration()
8.	MLTestcaseGenerator	-next_training_day	+generateTestCases() +executeTestCases() +reportGeneration() +checkTrainingDay() +trainModel() +setTrainingDay()
9.	Randoop		+generateTestCases() +executeTestCases() +reportGeneration()
10.	EmailSender	-email_id	-sendOTP() -varifyAccount()

6.7 CRC Card:

Serial	Class Name	Responsibility	Collaborator
1.	User	• Create account • Log in • Check duplicate username • Check duplicate email	Input, Output, EmailSender, Database

2.	InputController	• Check input validity • Perse input • Store test criteria • Store input	Input, Database
3.	Input	• Bind data to InputController	InputController
4.	OutputController	• Generate result • Store result • Show result	Output, Database
5.	Output	• Bind data to OutputController	OutputController
6.	Database	• Read data • Write data • Update data • Delete data • Verify in database	InputController, OutputController, User, MLTestcaseGenerator
7.	TestCaseGenerator	• Generate test cases • Execute test cases	InputController, OutputController
8.	MLTestcaseGenerator	• Generate test cases • Execute test cases • Set next training day • Train the model	TestCaseGenerator, Database
9.	Randoop	• Generate test cases • Execute test cases	TestCaseGenerator
10.	EmailSender	• Send OTP • Verify user account	User

6.8 Class Card:

User	
Attribute	**Method**
-fullname -username -email -password	+login() +create_account() +add_user() +checkDuplicateEmail() +checkDuplicateUsername()
Responsibility	**Collaborator**
● Create account ● Log in ● Check duplicate username ● Check duplicate email	Input, Output, EmailSender, Database

InputController	
Attribute	**Method**
-database	+isValidInput() +parseInput() +storeInput() +storeCriteria()
Responsibility	**Collaborator**
● Check input validity ● Perse input ● Store test criteria ● Store input	Input, Database

Input	
Attribute	**Method**
-input_id -input_code -expected_behavior -selected_tools -range_of_test_cases	+getter() +setter()
Responsibility	**Collaborator**
● Bind data to InputController	InputController

OutputController	
Attribute	**Method**
-database	+generateResult() -storeResult() +showResult()
Responsibility	**Collaborator**
● Generate result ● Store result ● Show result	Output, Database

Output	
Attribute	**Method**
-test_cases	+getter()

-input -auto_execution_status	+setter()
Responsibility	**Collaborator**
• Bind data to OutputController	OutputController

Database	
Attribute	**Method**
	+connect() +isConnected() +update() +read() +write() +delete() +verifyInDatabase()
Responsibility	**Collaborator**
• Read data • Write data • Update data • Delete data	InputController, OutputController, User, MLTestcaseGenerator

TestCaseGenerator	
Attribute	**Method**
	+generateTestCases() +executeTestCases() +reportGeneration()

Responsibility	**Collaborator**
• Generate test cases • Execute test cases	InputController, OutputController

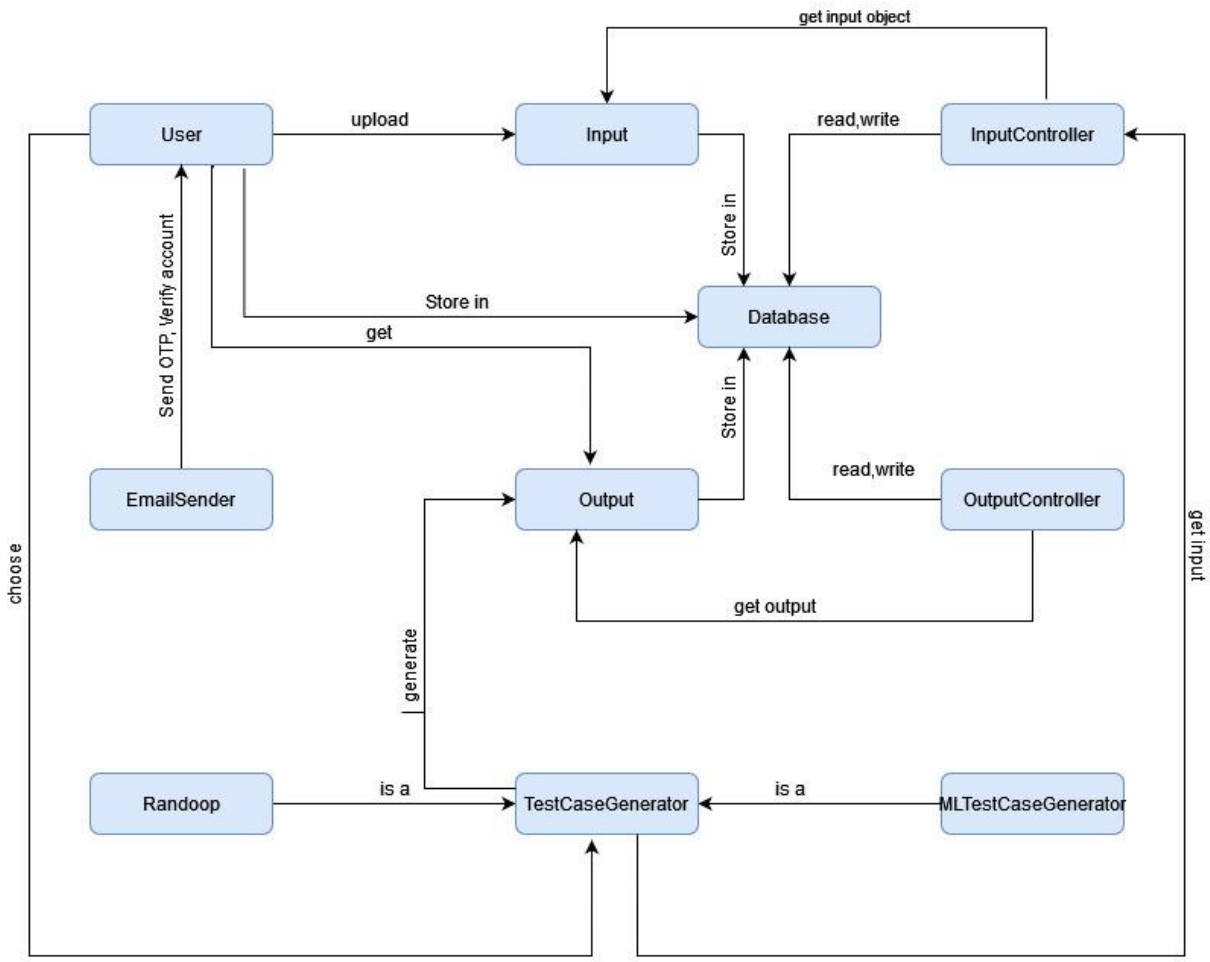
MLTestCaseGenerator	
Attribute	**Method**
-next_training_day	+generateTestCases() +executeTestCases() +reportGeneration() +checkTrainingDay() +trainModel() +setTrainingDay()
Responsibility	**Collaborator**
• Generate test cases • Execute test cases • Set next training day • Train the model	TestCaseGenerator, Database

Randoop	
Attribute	**Method**
	+generateTestCases() +executeTestCases() +reportGeneration()

Responsibility	**Collaborator**
• Generate test cases • Execute test cases	TestCaseGenerator

EmailSender	
Attribute	**Method**
-email_id	-sendOTP() -verifyAccount()
Responsibility	**Collaborator**
• Send OTP • Verify user account	User

6.9 Class Diagram:



7. Behavioral Modeling

The behavioral model indicates how software will respond to external events or stimuli. In the context of behavioral modeling, two different characterizations of states must be considered: (1) the state of each class as the system performs its function and (2) the state of the system as observed from the outside as the system performs its function.

7.1 State Transition Diagram

One component of a behavioral model is a UML state diagram that represents active states for each class and the events (triggers) that cause changes between these active states.

7.1.1 Identifying Events

Serial	Event	State Name	Initiator	Collaborator	Associated Method
1	Signing up to the system	Sign_Up	User	Database, EmailSender	+create_account()
2	Adding user to database	Add_User	User	Database	+addUser()
3	Checking duplicate username	Check_Duplicate_Username	User	Database	+checkDuplicateUsername() +verifyInDatabase()
4	Checking duplicate email	Check_Duplicate_Mail	User	Database	+checkDuplicateEmail() +verifyInDatabase()
5	Sending OTP	SendOTP	EmailSender	User	-sendOTP()
6	Verify email	VerifyMail	User	EmailSender	-verifyAccount()
7	Logging in to the system	Login	User	Database	+login()
8	Binding input to inputController	BindInput	InputController	Input	+getter() +setter()

9	Checking input validity	CheckInputValidity	InputController	Input	+isValid()
10	Parsing input	ParseInput	InputController	Input	+parseInput()
11	Storing input	StoreInput	InputController	Input, Database	+storeInput() +storeCriteria()
12	Generating test cases	GenerateTestCase	TestCaseGenerator	OutputController, Input, Output	+generateTestCases()
13	Executing test cases	ExecuteTestCase	TestCaseGenerator	OutputController, Input, Output	+executeTestCases()
14	Binding output to output controller	BindOutput	OutputController	Output	+getter() +setter()
15	Generating results	GenerateResult	OutputController	TestCaseGenerator, Output	+reportGeneration() +generateResult()
16	Showing results	ShowResult	OutputController	Output, Database	+showResult()
17	Storing results	StoreResult	OutputController	Output, Database	+storeResult()
18	Checking if training day	CheckTrainingDay	MLTestCaseGenerator		+checkTrainingDay()
19	Training model	TrainModel	MLTestCaseGenerator	Database	+trainModel()
20	Setting Training Day	SetTrainingDay	MLTestCaseGenerator	Database	+setTrainingDay()

21	Updating info	Update	Database	User, InputController, OutputController	+update()
22	Deleting from table	Delete	Database	User, InputController, OutputController	+delete()
23	Adding to database	Write	Database	User, InputController, OutputController	+write()
24	Reading from database	Read	Database	User, InputController, OutputController	+read()

7.1.2 State Transition Diagrams:

ID-1:
Name: User

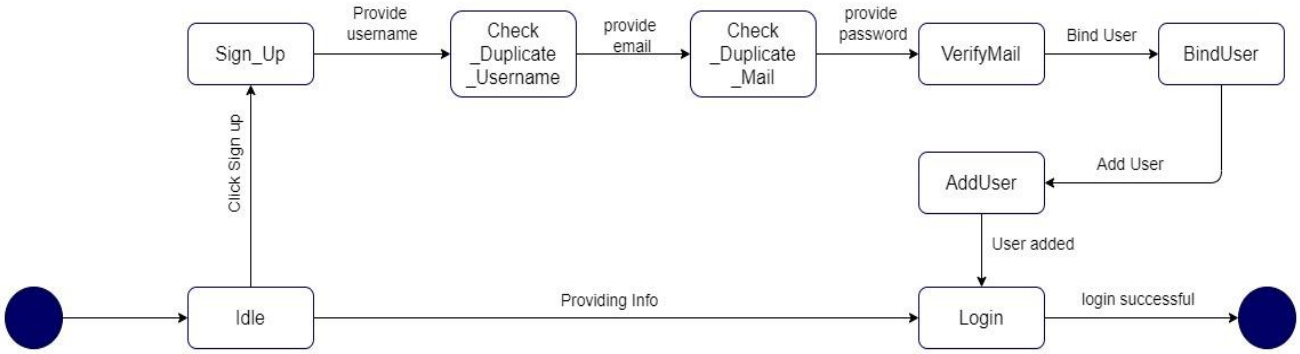


Figure 24: User

ID-2:
Name: EmailSender

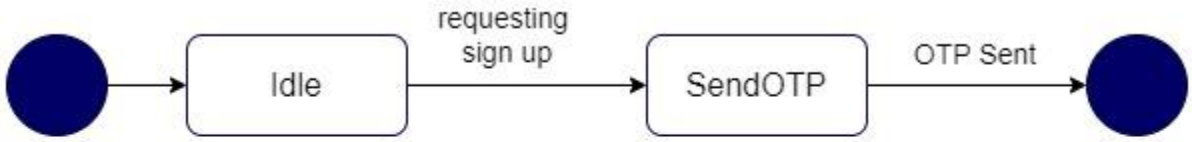


Figure 25: EmailSender

ID-3:
Name: InputController

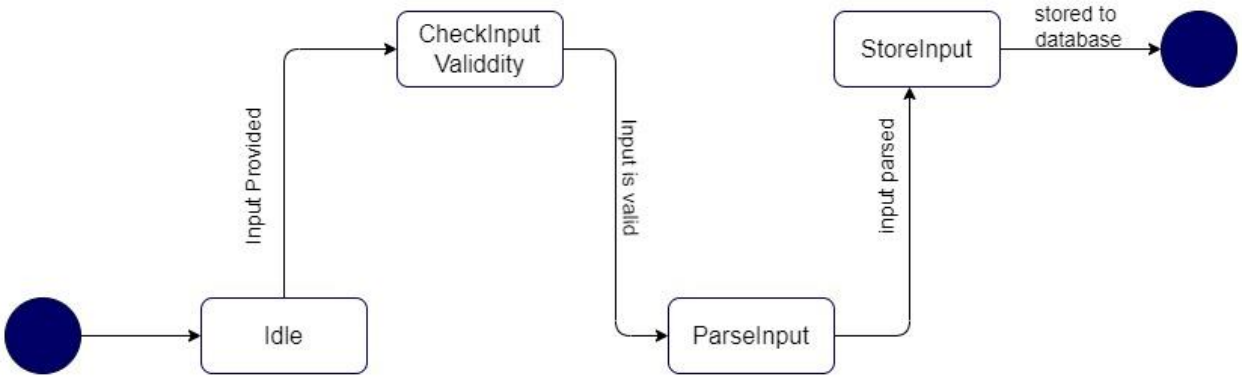


Figure 26: InputController

ID-4:
Name: OutputController

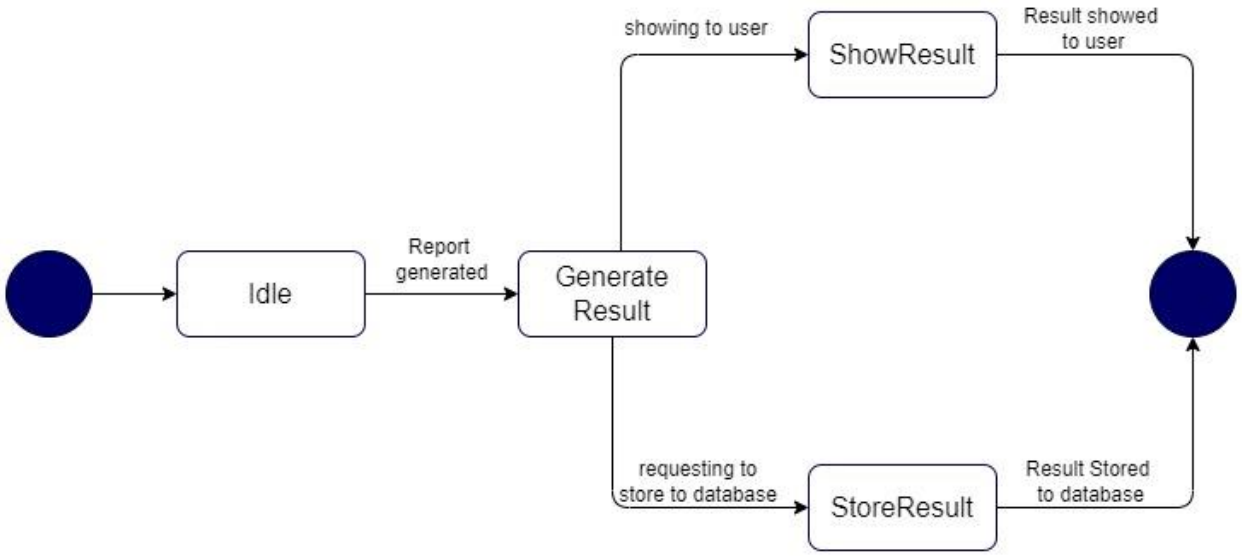


Figure 27: OutputController

ID-5:
Name: TestCaseGenerator

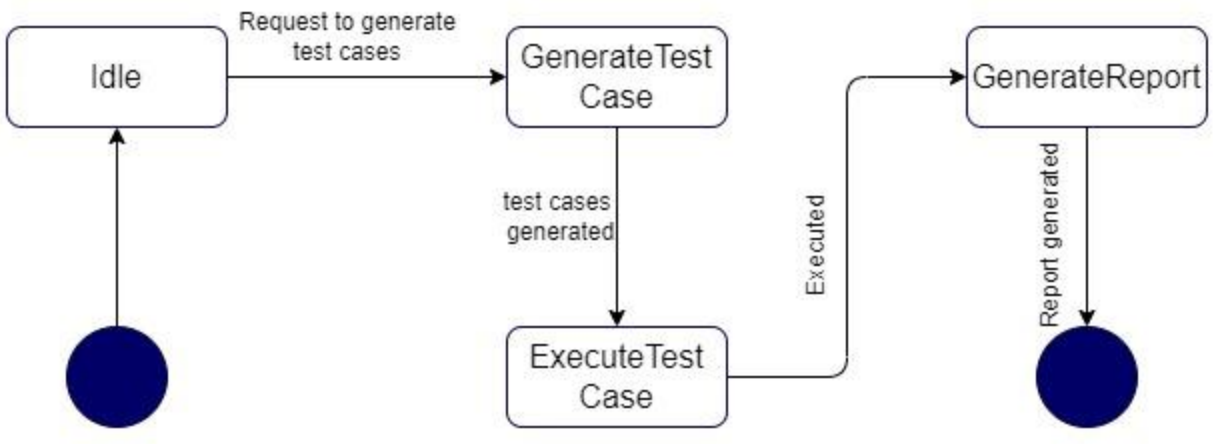


Figure 28: TestCaseGenerator

ID-6:
Name: MLTestCaseGenerator

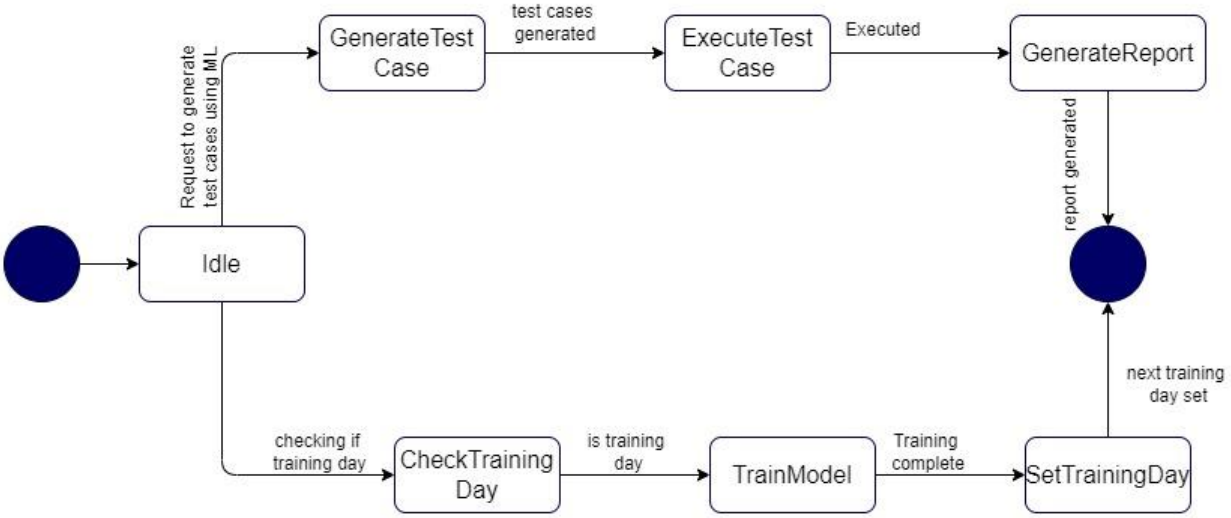


Figure 29: MLTestCaseGenerator

ID-7:
Name: Randoop

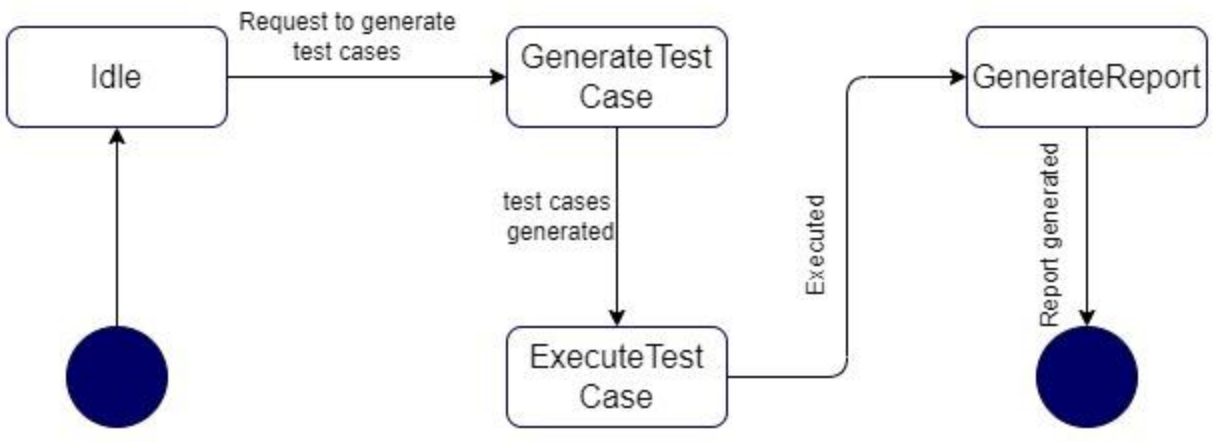


Figure 30: Randoop

ID-8:
Name: Database

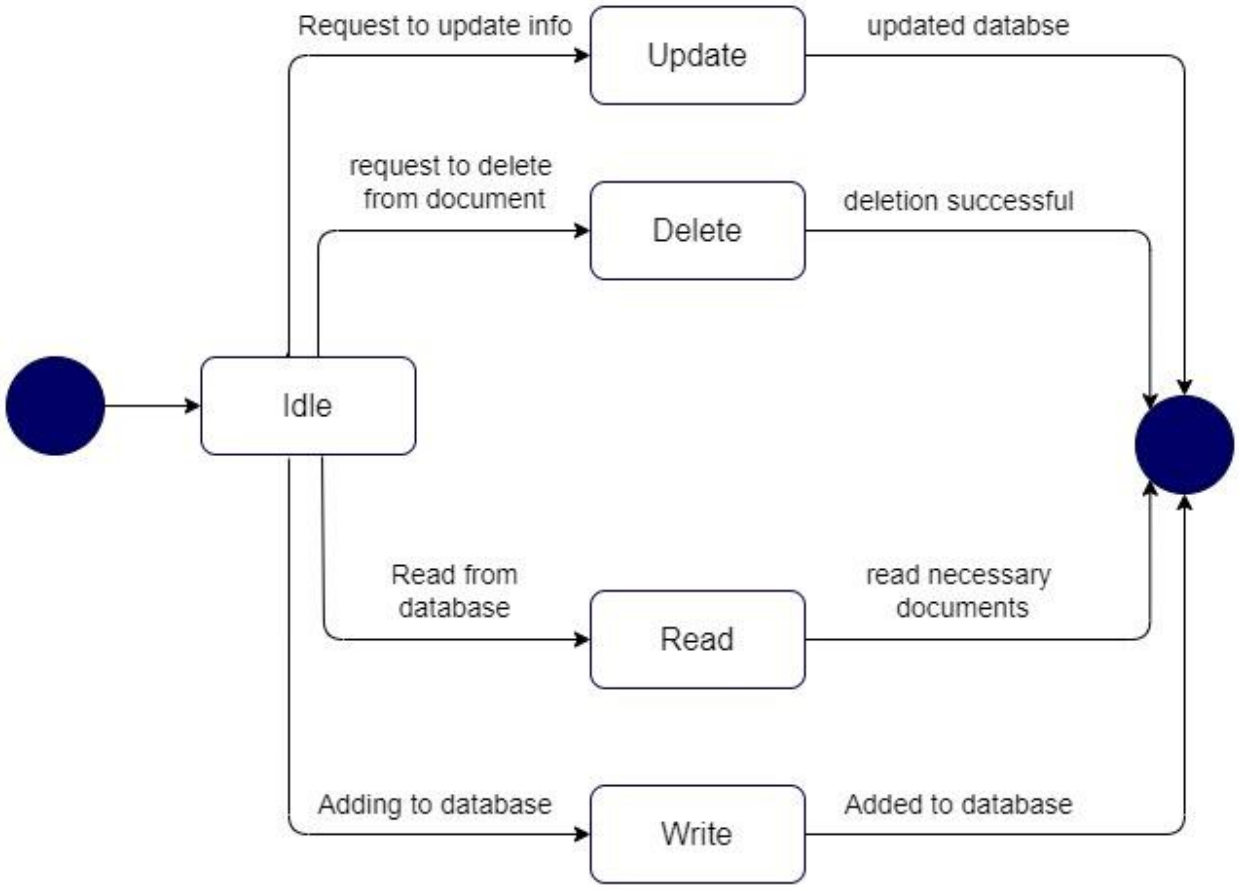


Figure 31: Database

7.2 Sequence Diagram:

ID-1:

Sign up and add user

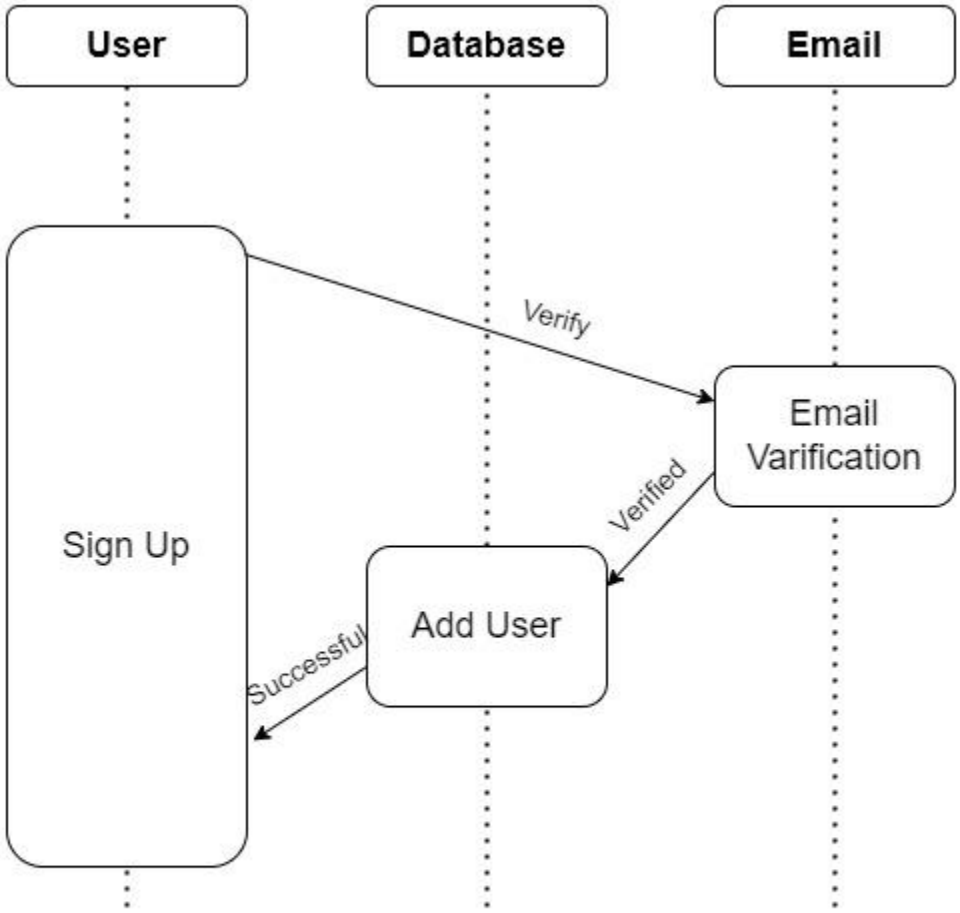


Figure 32: Sign up and Add User

ID-2:

Login

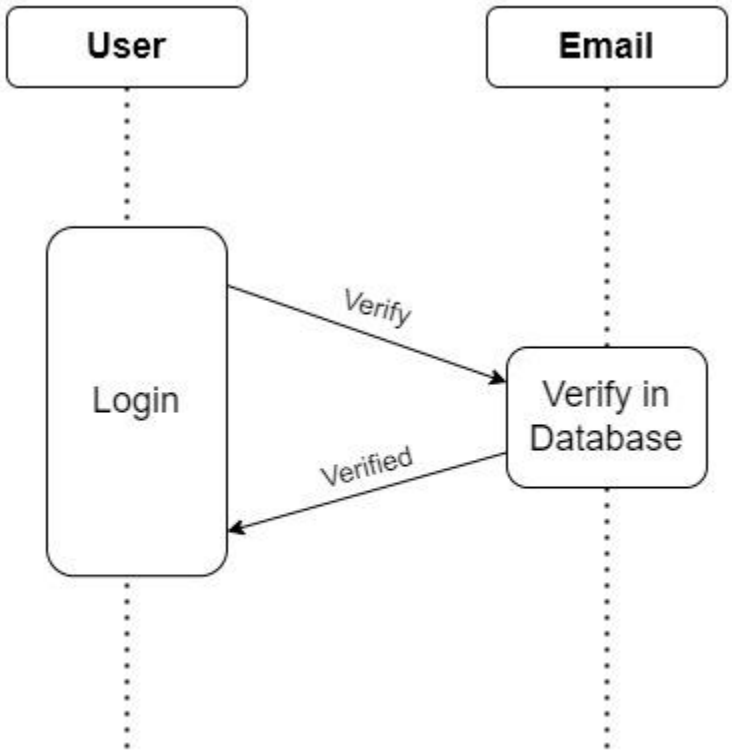


Figure 33: Login

ID-3:

Input processing

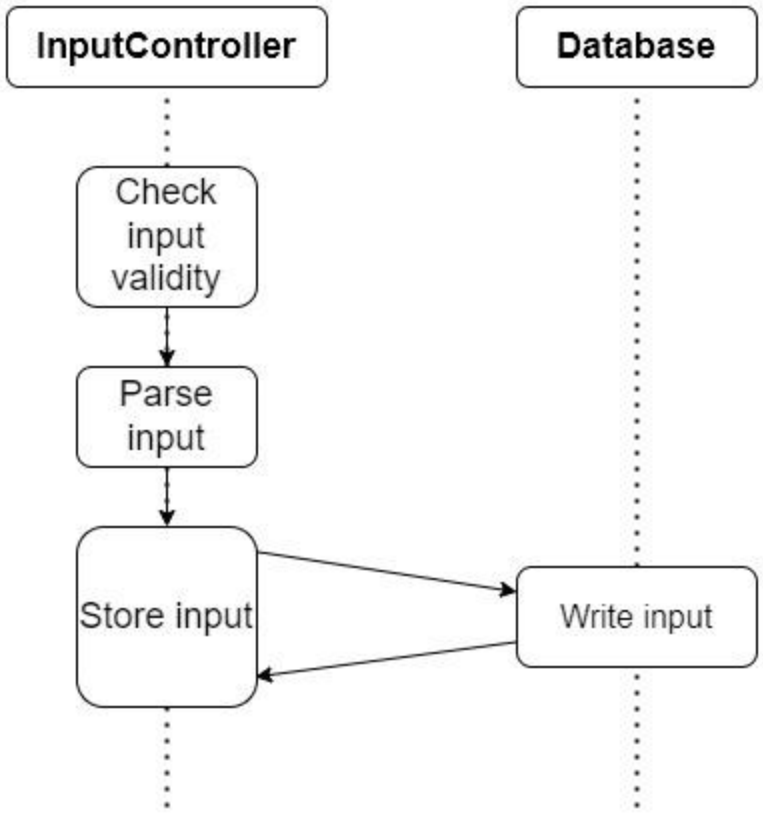


Figure 34: Input processing

ID-4
Result generation

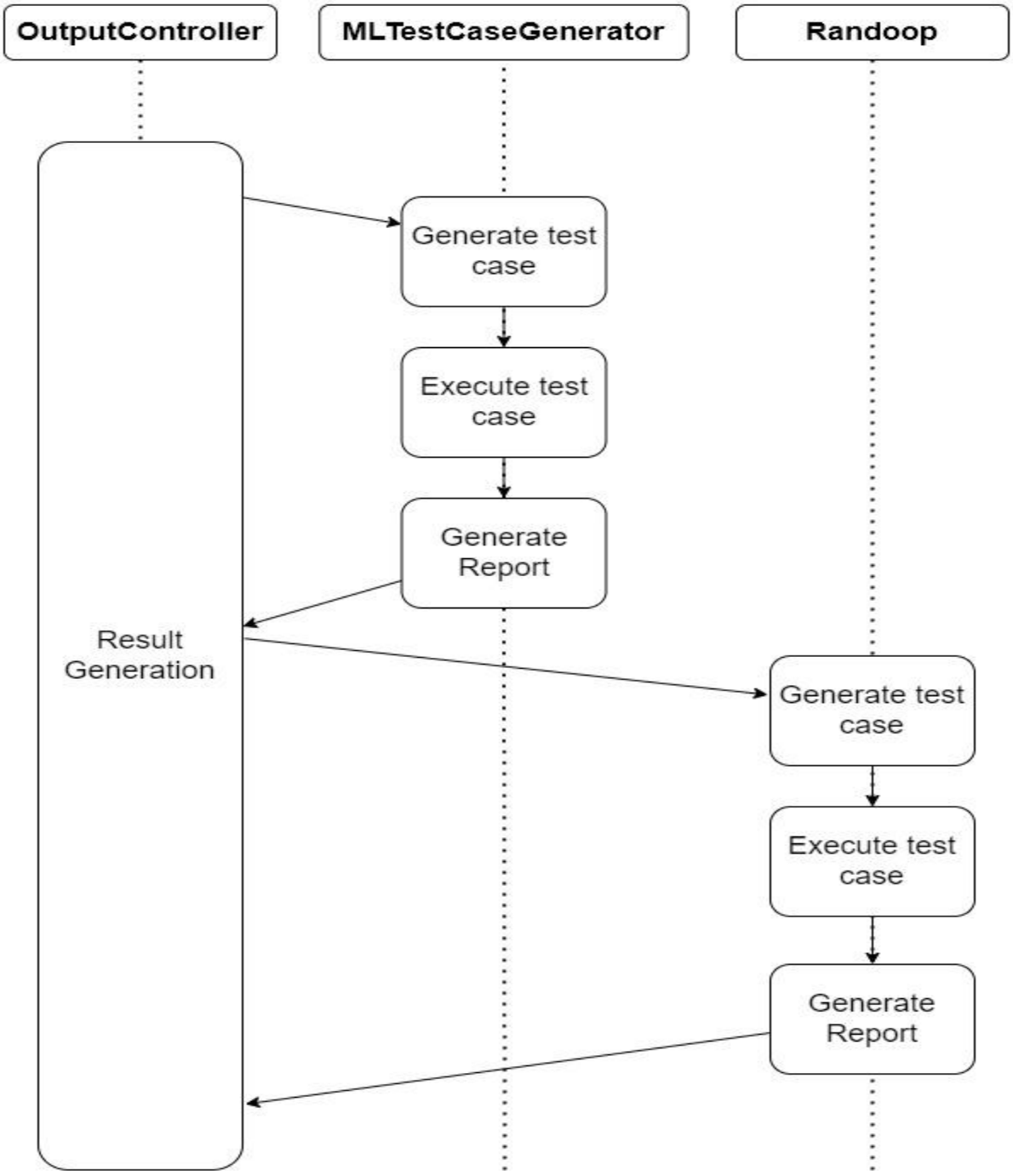


Figure 35 : Result Generation

ID-5:

Store and View Result

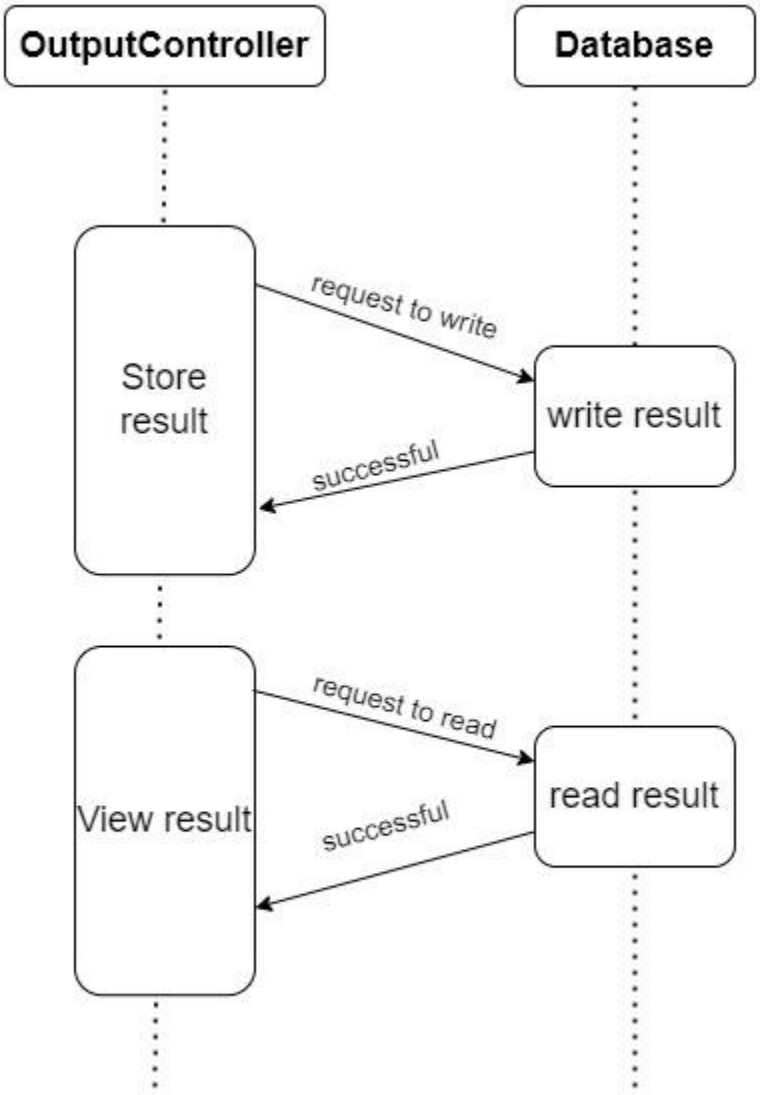


Figure 36: Store and View result

ID-6:
Re-Train Model

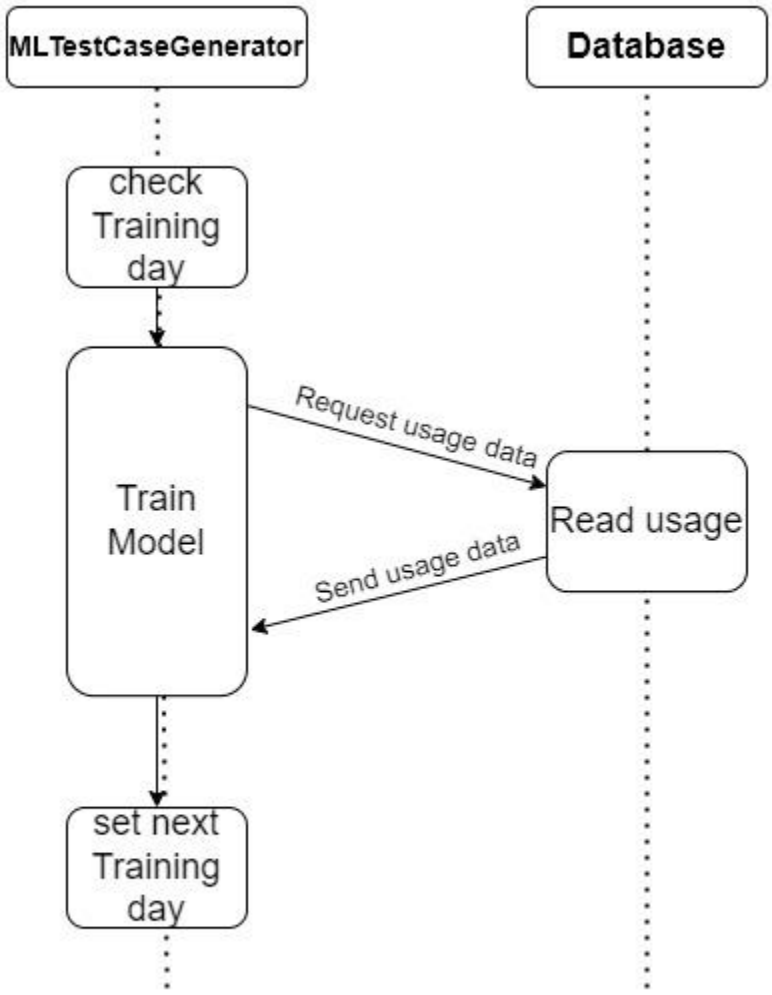


Figure 36: Re-Train Model