# GENERAL TESTING FRAMEWORK FOR LAB PROGRAMS

---


**SUBMITTED BY**

Abhiram Ashok
Aysha Naurin
Febin Nelson
Sreelakshmi K

**GUIDED BY**

Prof. Shibu Kumar

# PROBLEM STATEMENT

Validating student lab programs manually is a time-consuming and error-prone process, making it difficult for faculty to efficiently manage and test submissions. Students often lack immediate feedback on their code, hindering their ability to learn from mistakes and improve.

**Checkio** aims to solve this by providing an automated framework for program validation. Faculty can create and manage test cases, while students can run their programs against these cases to receive instant results, error feedback, and suggestions. The framework supports both command-line and web-based interfaces, offering a seamless and efficient solution for program testing and validation.

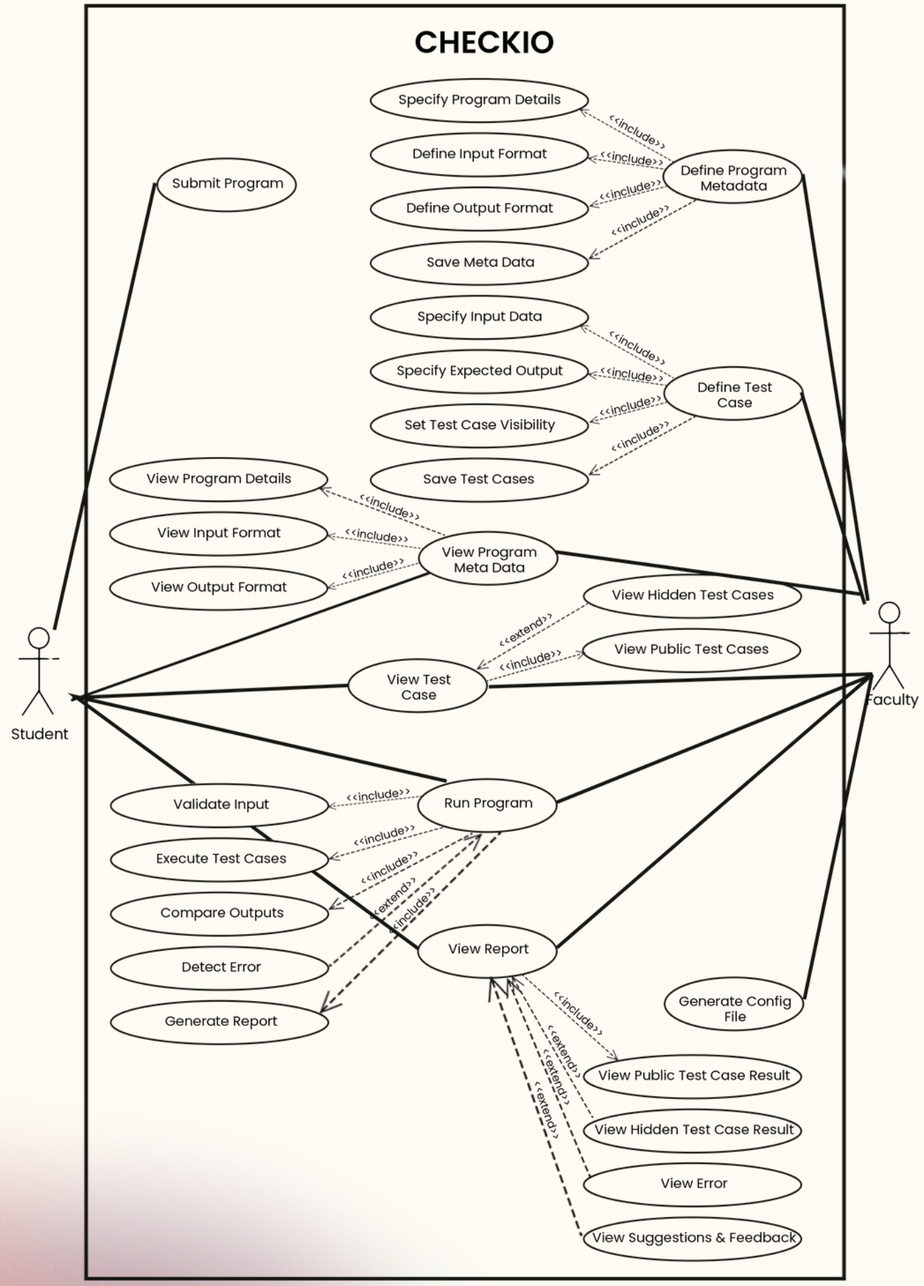
# OBJECTIVE

- Automate program evaluation by testing student submissions against predefined test cases.
- Detect and highlight errors with precision.
- Provide predictions for the cause of errors and suggest potential solutions.
- Reduce faculty workload and enhance the student learning experience.

# SYSTEM DIAGRAMS

# USE CASE DIAGRAM

- Present the user interactions and system functionalities.
- **Actors:**
  - **Faculty**: oversees program operations, maintains full control of the system, has administrative access
  - **Student:** submits programs, executes tests, accesses feedback, views only public test cases



# KEY USE CASES

## Faculty

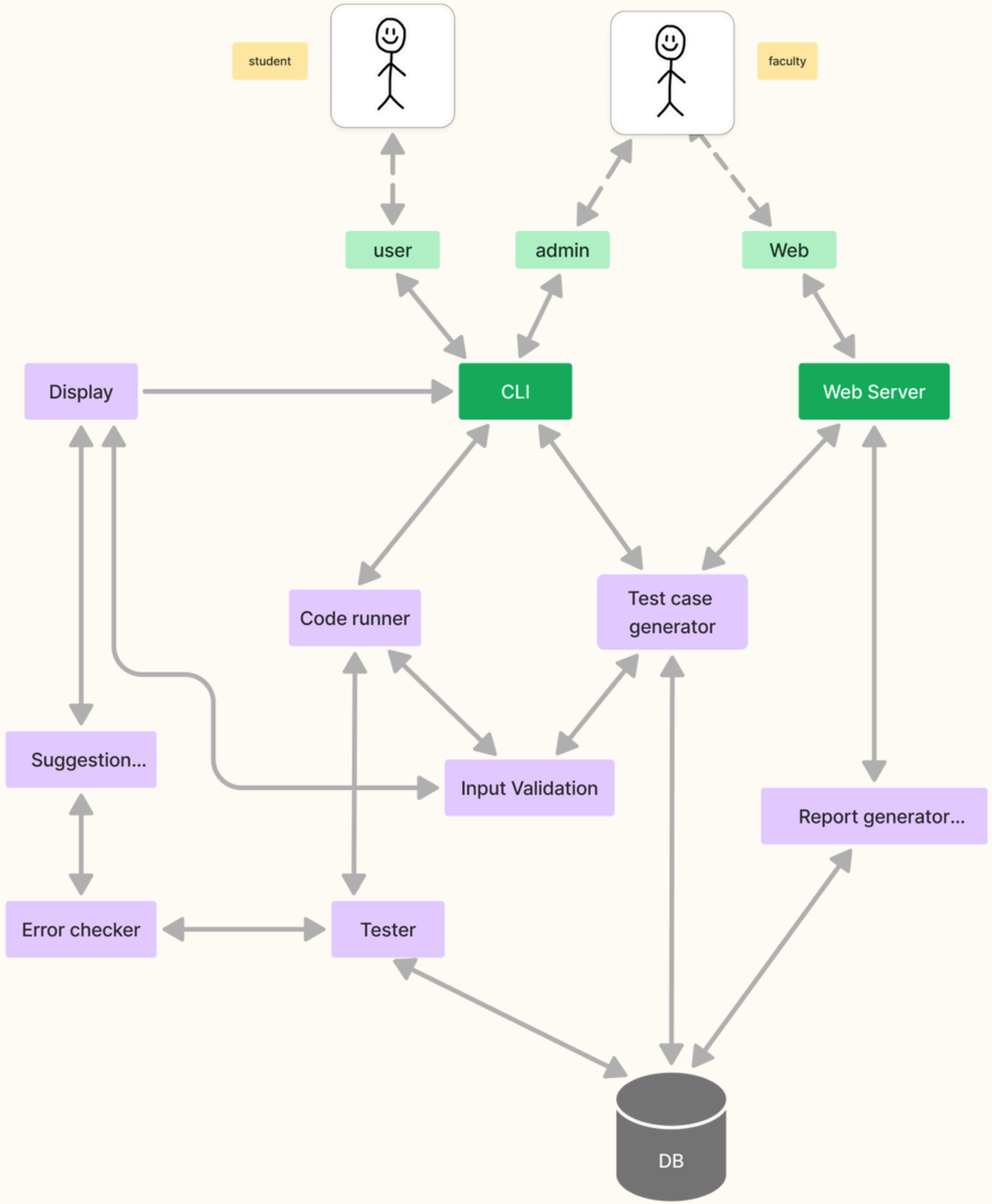
- Define Program Metadata
- Define Test Cases
- View Program Metadata
- View Test Cases
- Run Program
- View Report
- Generate Configuration File

## Student

- View Program Metadata
- View Test Cases
- Submit Program
- Run Program
- View Report

# SYSTEM ARCHITECTURE

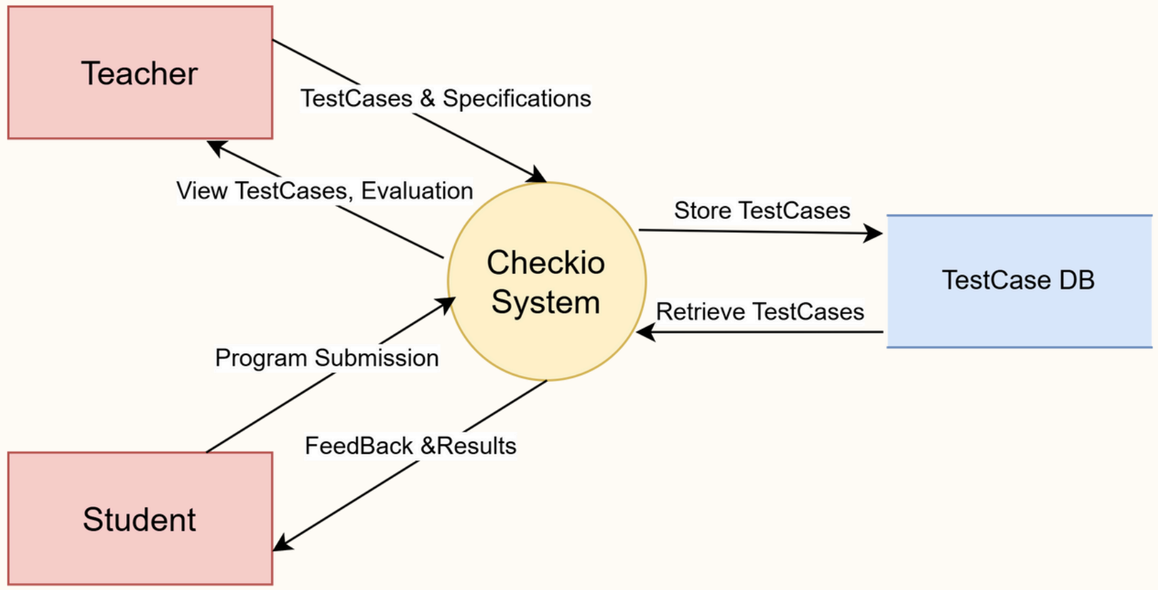
- **Users**
  - Students: Run test cases on their code.
  - Faculty: Create test cases and view reports.
- **Interface**
  - CLI Interface – Main way for users to interact.
  - Web Interface – Provides a browser-based interface.
- **Core Modules**
  - Test Case Generator: Faculty create test cases.
  - Code Runner: Executes student code against test cases.
  - Input Validation: Checks for correct input format.
  - Tester: Compares actual vs expected output.
  - Error Handling & Suggestions
  - Error Checker: Detects mistakes in student code.
  - Suggestion Module: Provides hints to fix errors.
  - Report Generator: Generates and sends reports to the corresponding faculty
- **Database**
  - Stores test cases, results, and reports.



# DATA FLOW DIAGRAM

## Level 0: Context Diagram:

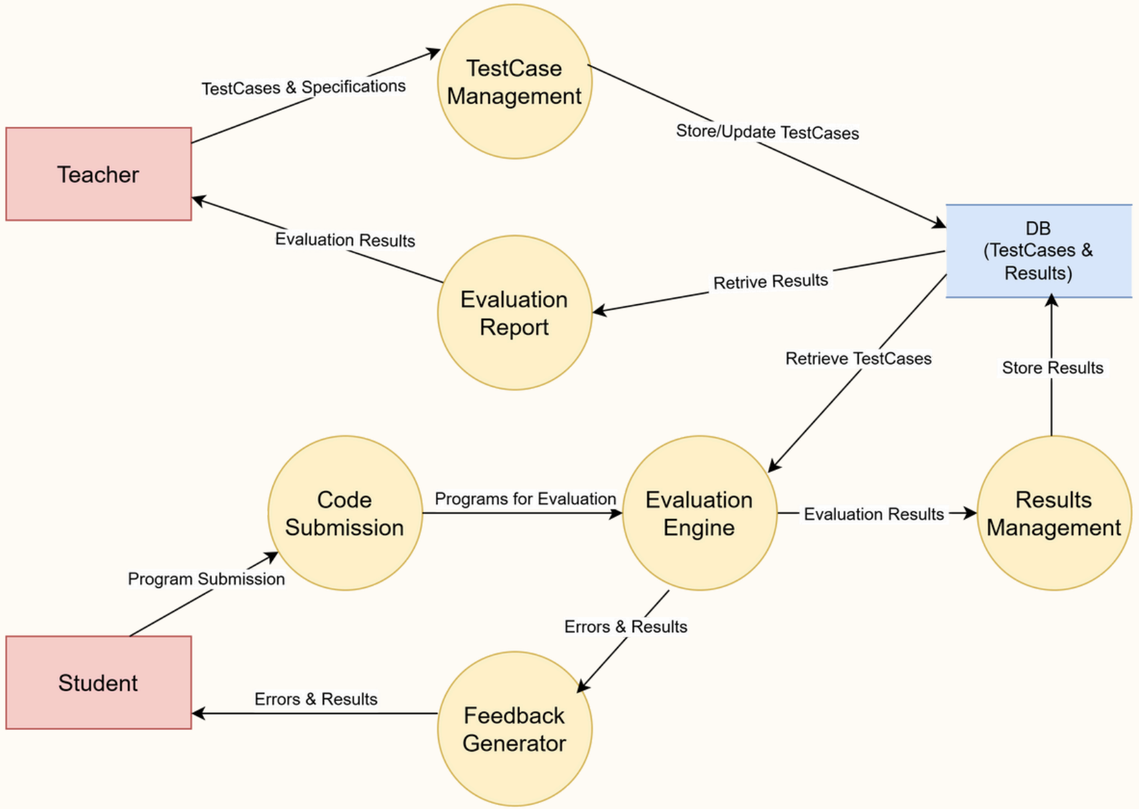
- Faculty: Provides test cases, expected outputs, and input conventions to the system.
- Student: Submits their program for evaluation.
- Checkio System: Evaluates the program against the test cases and outputs success reports or error feedback.



# DATA FLOW DIAGRAM

## Level 1: Detailed Processes

- Test Case Input: Teachers define inputs and expected outputs; saved in the repository.
- Code Submission: Students upload their program to the system.
- Evaluation: System executes the program using predefined test cases; outputs are verified.
- Error Analysis: Errors are identified, and suggestions provided.
- Result Generation: Success or error reports are saved in the repository.



# SEQUENCE DIAGRAM

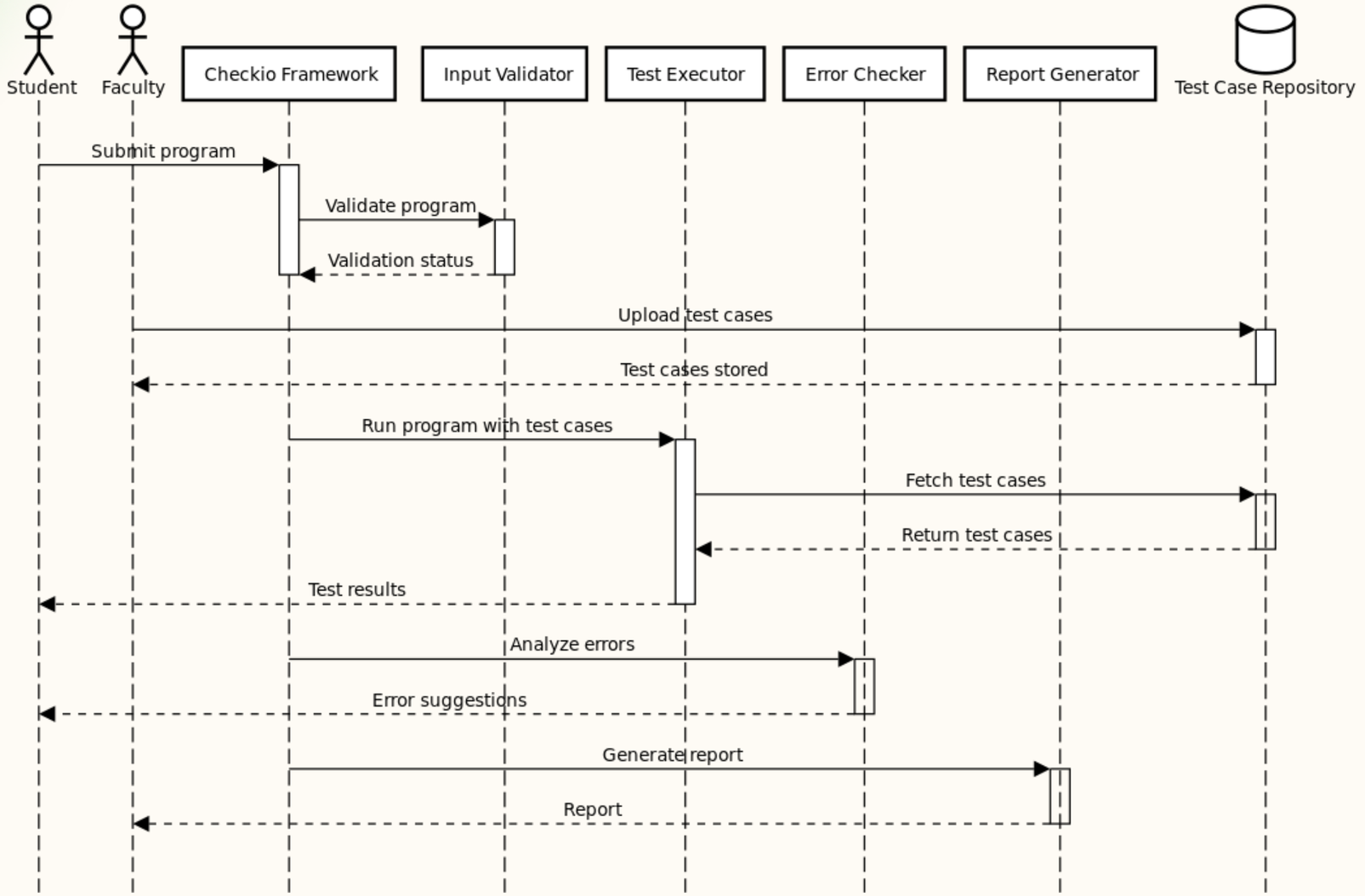
**1. Student Submission:** Students submit their program to the Checkio Framework.

**2. Faculty Contribution:** Faculties upload the necessary test cases to the Test Case Repository.

**3. Program Testing:** The framework passes the validated program to the Test Executor, which fetches test cases from the Test Case Repository.

**4. Error Analysis:** The framework collaborates with the Error Checker to analyze any issues in the program and provide constructive suggestions for corrections.

**5. Report Generation:** Finally, the Report Generator compiles a detailed report of test results, including passed and failed test cases.



# ALGORITHM DESIGN

## Input:

Test Cases: A list of cases each containing :

- *Input data: Input values for the program. These include files, texts or any other types of inputs to the program*
- *Expected output: The correct output for the given input.*
- *Student code: The source code submitted by the student as a file.*

## Output:

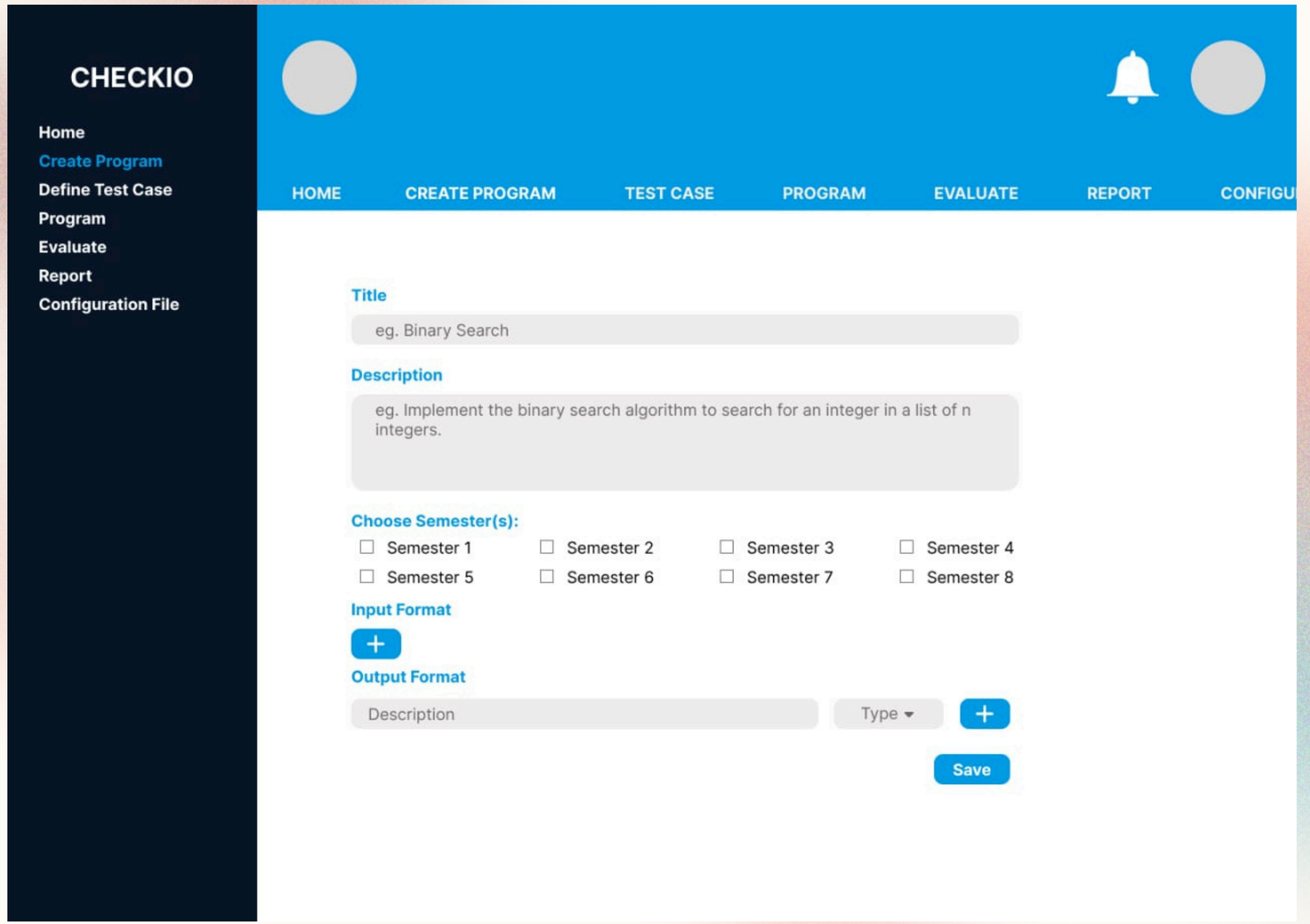
Evaluation report containing :

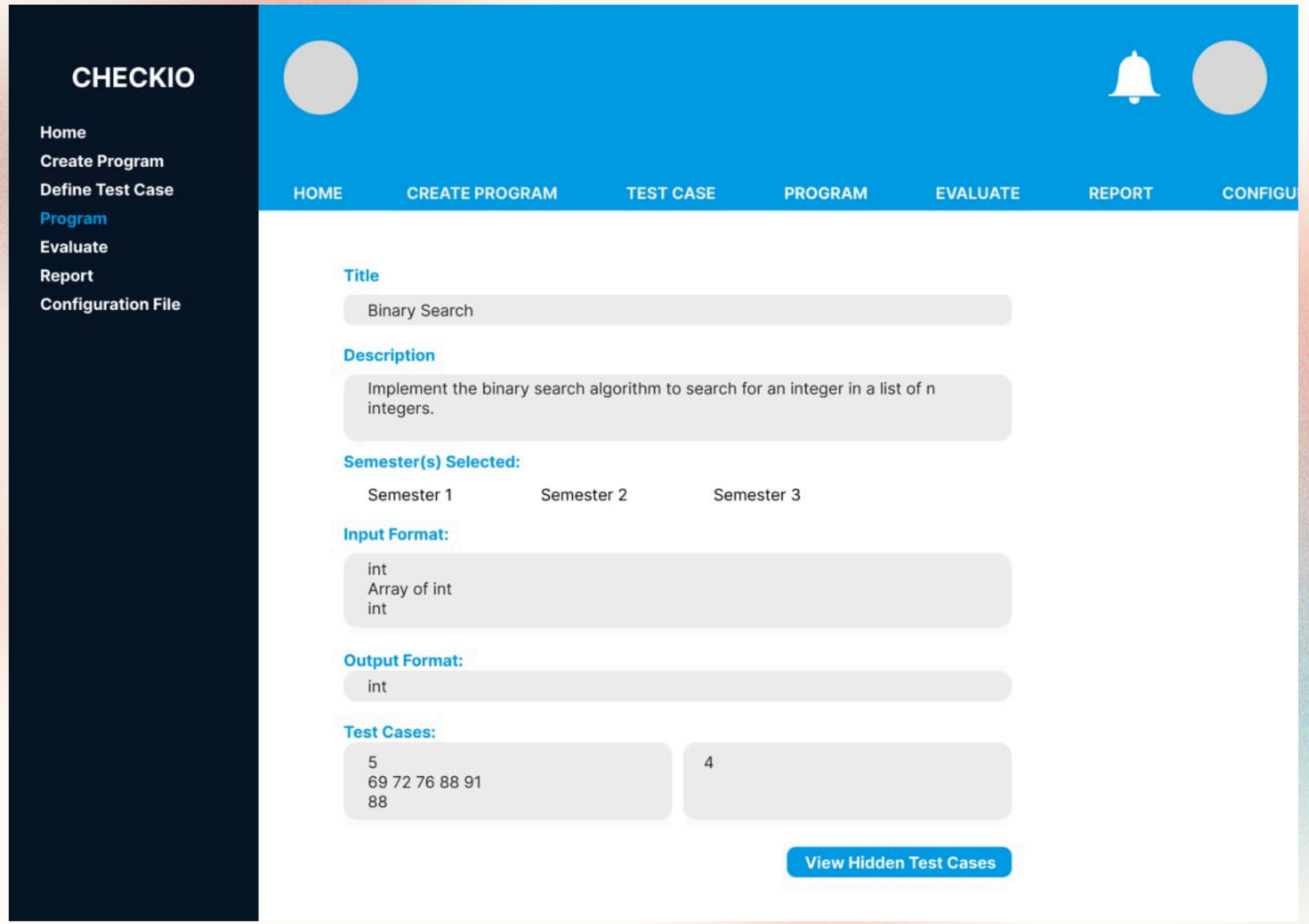
- *Overall success or failure.*
- *Detailed results for each test case (pass/fail) along with other details like the time taken to run the code, and other useful metrics.*
- *Identified errors (if any) and suggestions. These include potential error code, fixes, and other suggestions to improve the code*

# ALGORITHM DESIGN

**Step 1:** Start.
**Step 2:** Parse the test_cases provided by the teacher.
**Step 3:** For each test case in the test_cases, do the following:
**Step 3.1:** Pass the input_data to the student's program.
**Step 3.2:** Execute the student's program and capture the output (program_output).
**Step 3.3:** Compare the program_output with the expected_output:
If they match, mark the test case as "pass."
Otherwise:
Mark the test case as "fail."
Analyze the error by comparing outputs.
Generate suggestions.
**Step 4:** Summarize the results:
Calculate the overall pass/fail rate.
Add detailed feedback for each failed test case.
**Step 5:** Prepare the evaluation_report containing:
Overall success or failure.
Results for all test cases (pass/fail).
Suggestions for improvement.
**Step 6:** Output the evaluation_report as structured data or formatted text.
**Step 7:** End.

# UI DESIGNS


CHECKIO
Home
Create Program
Define Test Case
Program
Evaluate
Report
Configuration File
HOME
CREATE PROGRAM
TEST CASE
PROGRAM
EVALUATE
REPORT
CONFIGU
Title
eg. Binary Search
Description
eg. Implement the binary search algorithm to search for an integer in a list of n integers.
Choose Semester(s):
Semester 1
Semester 2
Semester 3
Semester 4
Semester 5
Semester 6
Semester 7
Semester 8
Input Format
Output Format
Description
Type
Save


CHECKIO
Home
Create Program
Define Test Case
Program
Evaluate
Report
Configuration File
HOME
CREATE PROGRAM
TEST CASE
PROGRAM
EVALUATE
REPORT
CONFIGU
Title
Binary Search
Description
Implement the binary search algorithm to search for an integer in a list of n integers.
Semester(s) Selected:
Semester 1
Semester 2
Semester 3
Input Format:
int
Array of int
int
Output Format:
int
Test Cases:
5
69 72 76 88 91
88
4
View Hidden Test Cases

# TOOLS USED

## Python

Core programming language for building logic and backend services.

## Flask

Lightweight web framework used for creating the web server.

## React

Frontend library for developing a dynamic and responsive UI.

## Figma

Design tool for system architecture, UI/UX prototyping, and wireframing

# TIMELINE

| | |
|---|---|
| **Week 1-2** | • Planning & System Design |
| **Week 3** | • Core Module Development |
| **Week 4-5** | • CLI Development And Web Server |
| **Week 6** | • Frontend Development |
| **Week 7** | • Testing And Finalization |

# Thank You

Any Queries?