Software Design Document

for

General Testing Framework for Academic Lab Programs

Prepared by

Abhiram Ashok (KTE22CS003)

Aysha Naurin (KTE22CS024)

Febin Nelson P (KTE22CS029)

Sreelakshmi K (KTE22CS063)

Department of Computer Science and Engineering

Rajiv Gandhi Institute of Technology, Kottayam

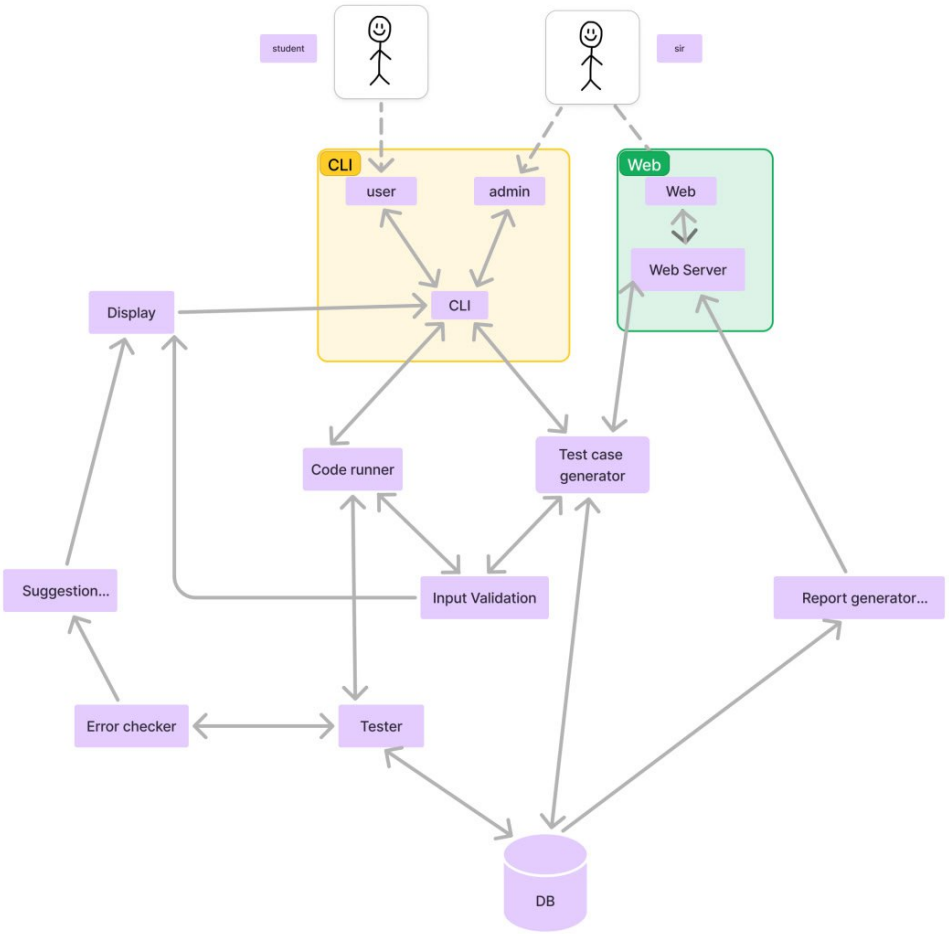
Index

1 System Architecture Design

The diagram represents the *System Architecture* of the Checkio framework. It illustrates how different components interact and the overall flow of data.

- **Users:** Two types of users interact with the system – *students* and *faculty*.
- **CLI Module:** The primary interface where users (students/admins) interact.
- **Code Execution Flow:**
  - *Test Case Generator* creates test cases.
  - *Input Validation* ensures correctness.
  - *Code Runner & Tester* execute the submitted programs.
  - *Error Checker* provides feedback and suggestions.
- **Database (DB):** Stores test cases, evaluation reports, and execution results.
- **Web Interface (Planned):** A web-based platform for faculty, integrated with the backend server.

This diagram helps in understanding the *workflow* and *system architecture*, showing how different modules communicate to automate lab program validation.



2 Dataflow Diagram

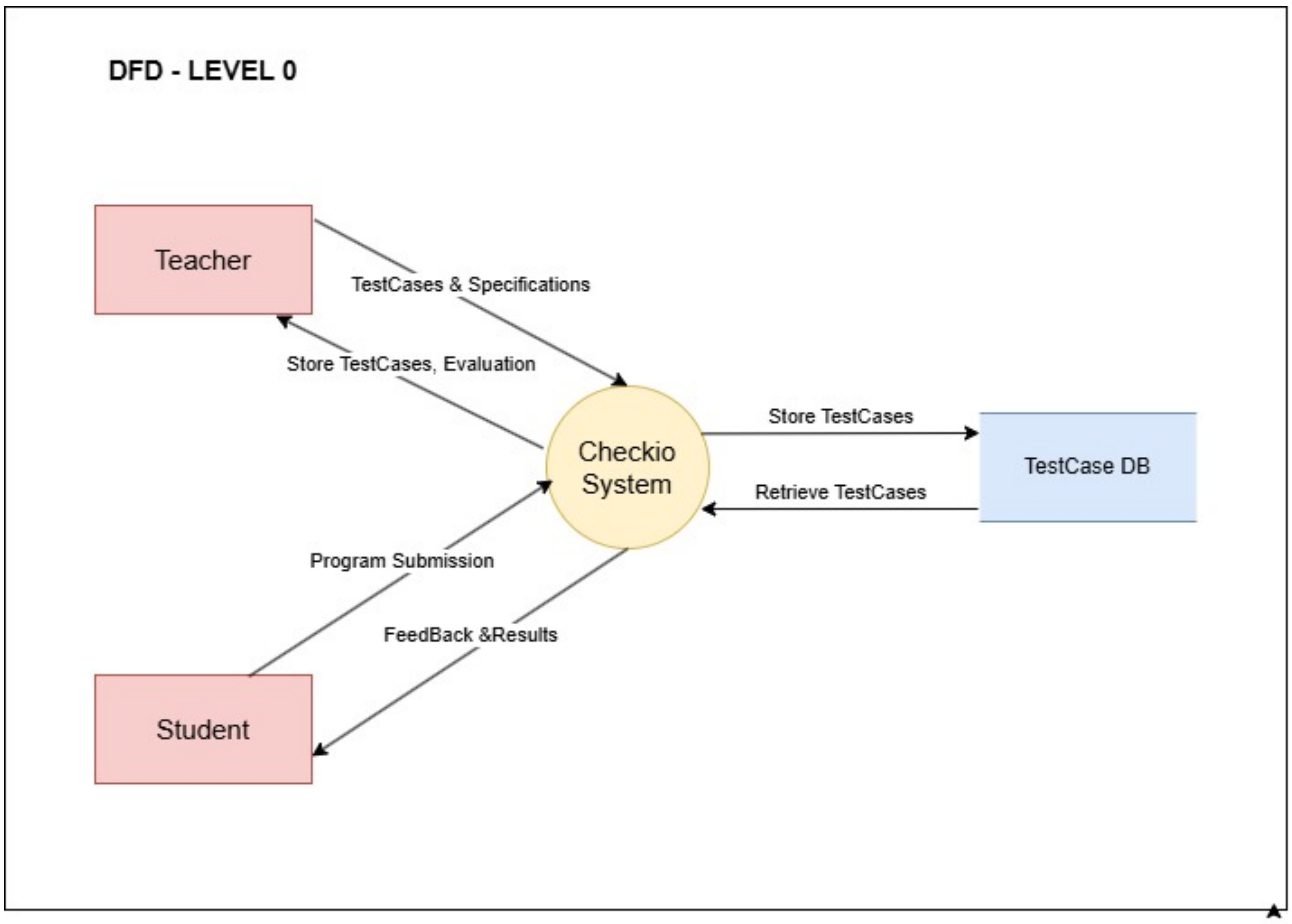
Level 0: Context Diagram

- **Teacher:** 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.

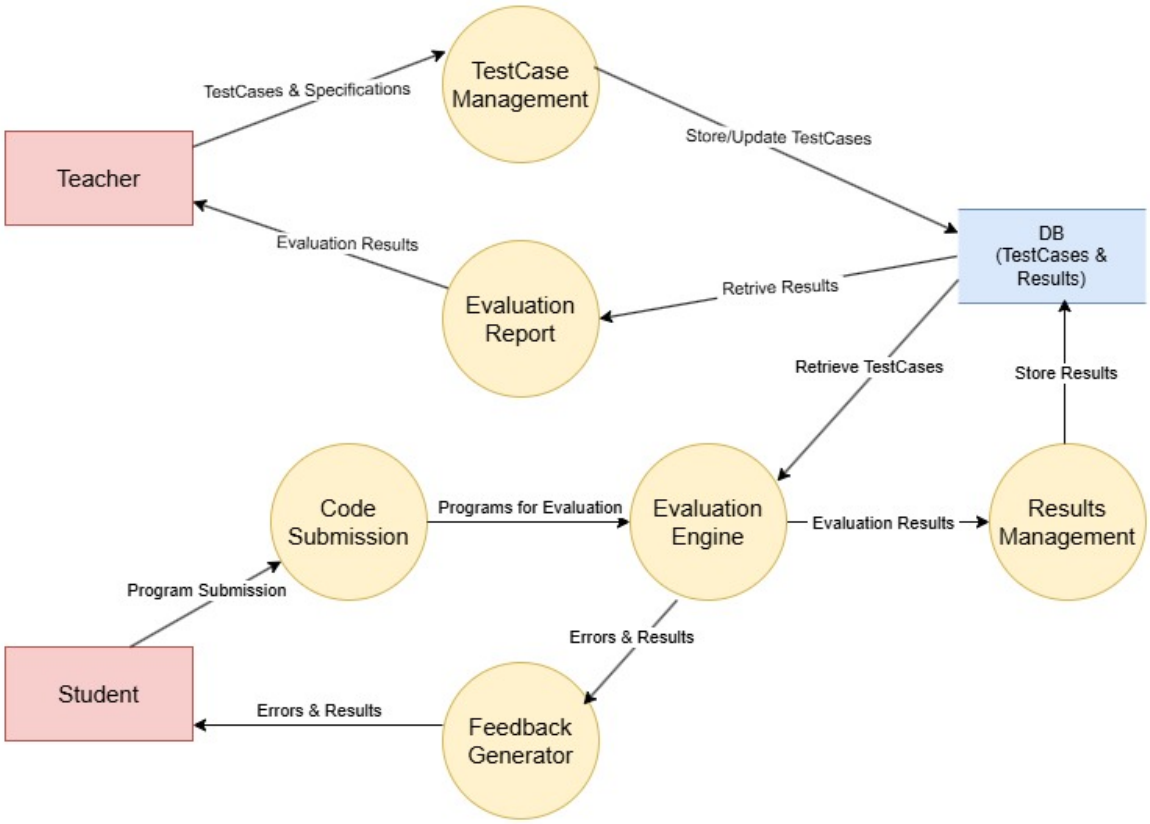
Level 1: Detailed Processes

1. **Test Case Input (by Teacher):**
   - Teachers define input conditions and expected outputs.
   - These are stored in the *Test Case Repository* for reuse.
2. **Code Submission (by Student):**
   - Students upload their program to the system.
3. **Evaluation Process:**
   - The system executes the program using the provided test cases.
   - Outputs are compared with the expected results.
4. **Error Analysis:**
   - If outputs differ, errors are identified (e.g., logic errors, syntax issues).
   - Suggestions and corrections are generated for identified problems.
5. **Result Generation:**
   - If outputs match, a success report is generated.
   - All results, including error details and suggestions, are saved in the *Evaluation Results Repository* for review.

This diagram effectively maps the flow of data, ensuring an efficient program evaluation process while maintaining a secure and robust system design.


DFD - LEVEL 0
Teacher
TestCases & Specifications
Store TestCases, Evaluation
Checkio System
Store TestCases
Retrieve TestCases
TestCase DB
Program Submission
FeedBack &Results
Student

DFD-Level 1


TestCase Management
TestCases & Specifications
Teacher
Store/Update TestCases
DB (TestCases & Results)
Evaluation Results
Evaluation Report
Retrive Results
Retrieve TestCases
Store Results
Code Submission
Programs for Evaluation
Evaluation Engine
Evaluation Results
Results Management
Program Submission
Errors & Results
Student
Errors & Results
Feedback Generator

3 Use Case Diagram

This use case diagram captures the interactions between the system and its two primary users: Faculty and Students. Each user has specific roles and access levels, ensuring smooth operation and role-based permissions.

Actors and Roles

1. **Faculty:** Responsible for defining and managing program metadata and test cases. Faculty can view and run programs, generate reports, and create configuration files.
2. **Student:** Focuses on submitting programs, running tests, and viewing limited reports.

Key Use Cases

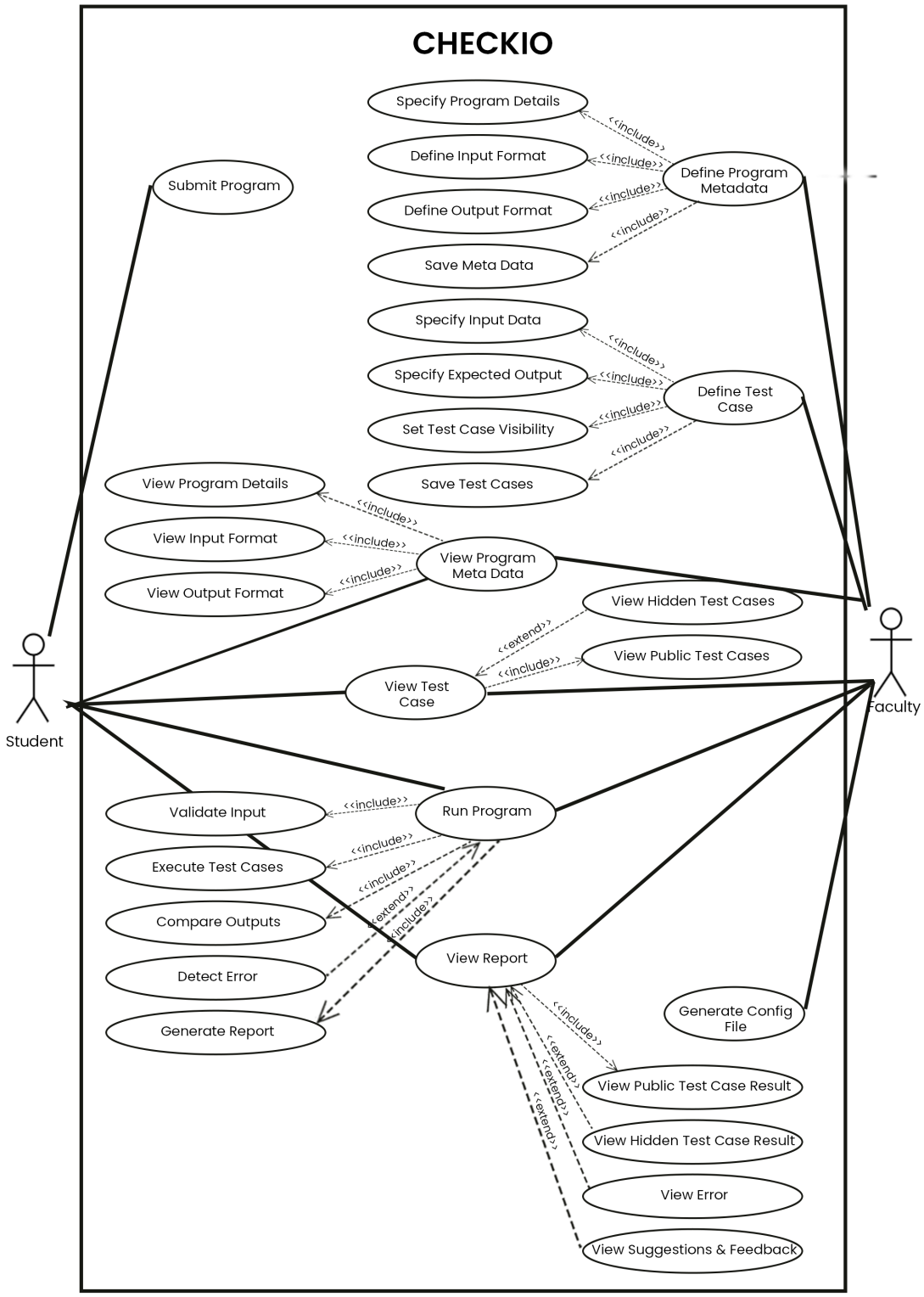
1. **Define Program Metadata (Faculty only):** Faculty sets up program details, including:
   - Defining program details (title, description).
   - Specifying input/output formats.
   - Saving metadata.
2. **Define Test Cases (Faculty only):** Faculty creates test cases for program validation, including:
   - Specifying input/output data.
   - Setting visibility (public/hidden).
   - Saving test cases.
3. **View Program Metadata:**
   - Faculty: Full access to all metadata.
   - Students: Limited view.
   - Includes: Viewing program details, input format, and output format.
4. **View Test Cases:**
   - Faculty: Can view both public and hidden test cases.
   - Students: Can only view public test cases.
   - Includes: Viewing public test cases.
   - Extends: Viewing hidden test cases (faculty only).
5. **Submit Program (Students only):** Students submit their programs for validation.
6. **Run Program (Faculty and Students):** Executes the program against test cases to validate functionality, including:
   - Validating input.
   - Executing test cases.
   - Comparing outputs.
   - Generating reports.

   Extends: Detecting errors.
7. **View Report (Faculty and Students):** Displays the test results and feedback, including:
   - Viewing public test case results.
   - Extends: Viewing hidden test case results (faculty only).
   - Viewing errors and feedback.
8. **Generate Configuration File (Faculty only):** Faculty can create a standardized configuration file containing program metadata and test cases.

Relationship Summary

- **Include:** Essential steps for completing a use case (e.g., saving metadata).
- **Extend:** Optional/conditional features like error detection or viewing hidden test cases.
- **Shared Use Cases:** Some use cases (e.g., "Run Program," "View Report") are shared but with different levels of access.

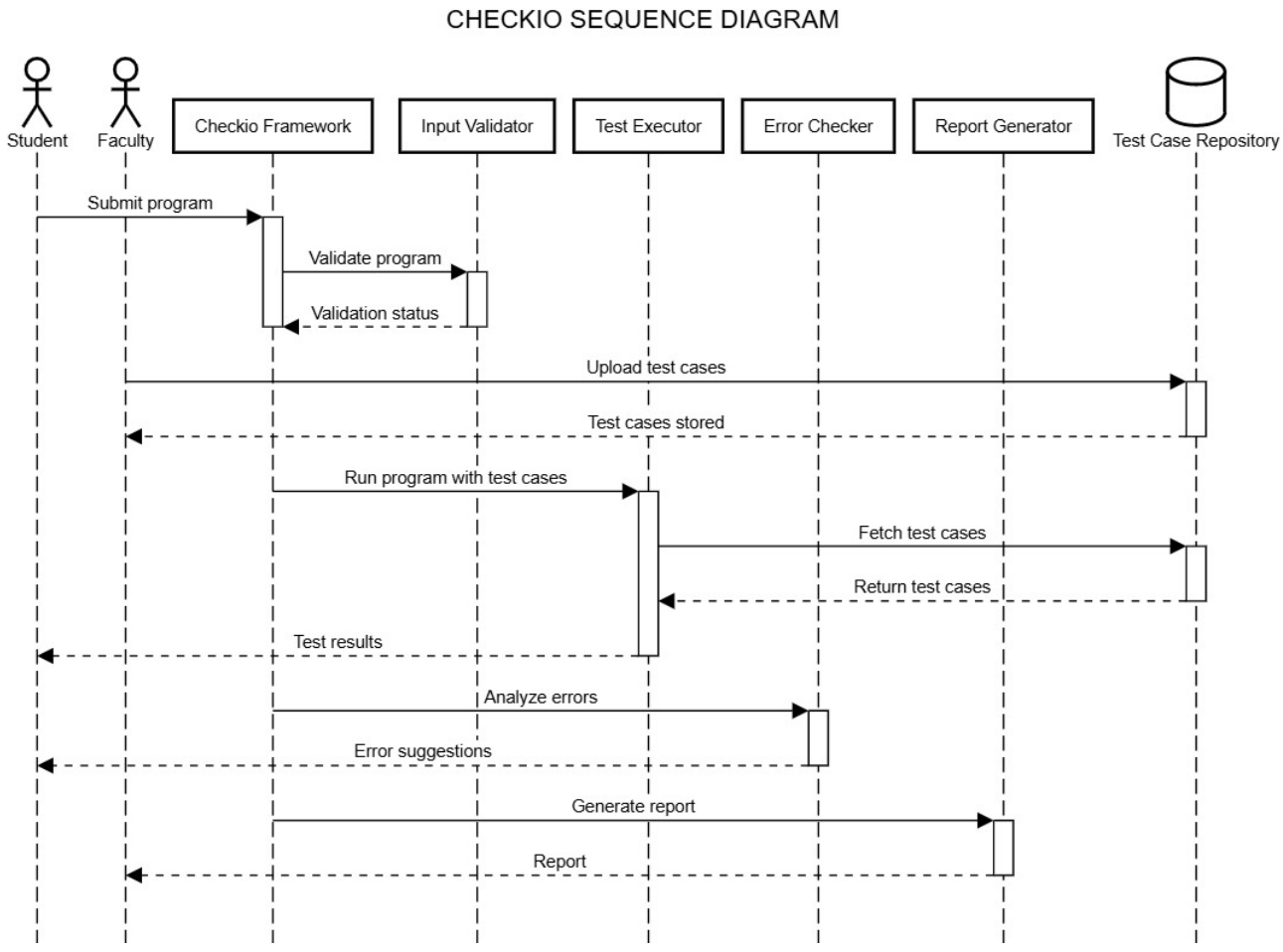
This diagram ensures a clear workflow, balancing accessibility for students and advanced control for faculty.


CHECKIO
Specify Program Details
Define Input Format
Define Output Format
Save Meta Data
Define Program Metadata
Submit Program
Specify Input Data
Specify Expected Output
Set Test Case Visibility
Save Test Cases
Define Test Case
View Program Details
View Input Format
View Output Format
View Program Meta Data
View Hidden Test Cases
View Public Test Cases
View Test Case
Student
Faculty
Validate Input
Execute Test Cases
Compare Outputs
Detect Error
Generate Report
Run Program
View Report
Generate Config File
View Public Test Case Result
View Hidden Test Case Result
View Error
View Suggestions & Feedback
<<include>>
<<extend>>

4 Sequence Diagram

This Sequence Diagram illustrates the interactions between the actors (Faculty, Student) with various components in the Checkio framework for automating the validation of academic lab programs.

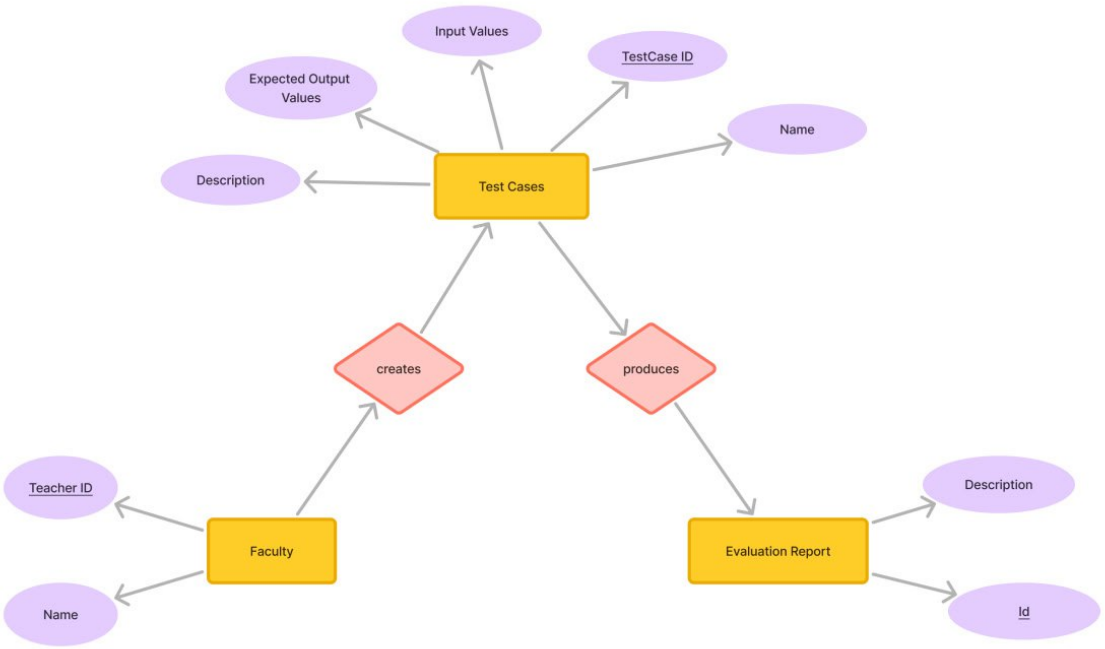
1. **Student Submission:** Students submit their program to the Checkio Framework, which validates the input using the Input Validator to ensure correctness.
2. **Faculty Contribution:** Faculties upload the necessary test cases to the Test Case Repository, ensuring that all programs are evaluated against predefined standards.
3. **Program Testing:** The framework passes the validated program to the Test Executor, which fetches test cases from the Test Case Repository and runs the program. The results are then returned to the student.
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, and delivers it to the faculty for evaluation.



5 Database Design

Entity-Relationship Diagram (ERD): The diagram represents the Entity-Relationship Model of the system, defining the key entities and their interactions.

Entities: - **Faculty:** Creates test cases. - **Test Cases:** Contains attributes like TestCase ID, Name, Input Values, Expected Output Values, and Description. - **Evaluation Report:** Generated based on test case execution, containing attributes like ID and Description.
Relationships: - Faculty *creates* Test Cases. - Test Cases *produce* Evaluation Reports.
This diagram helps define the *database structure* and *relationships* between the different components of the system.



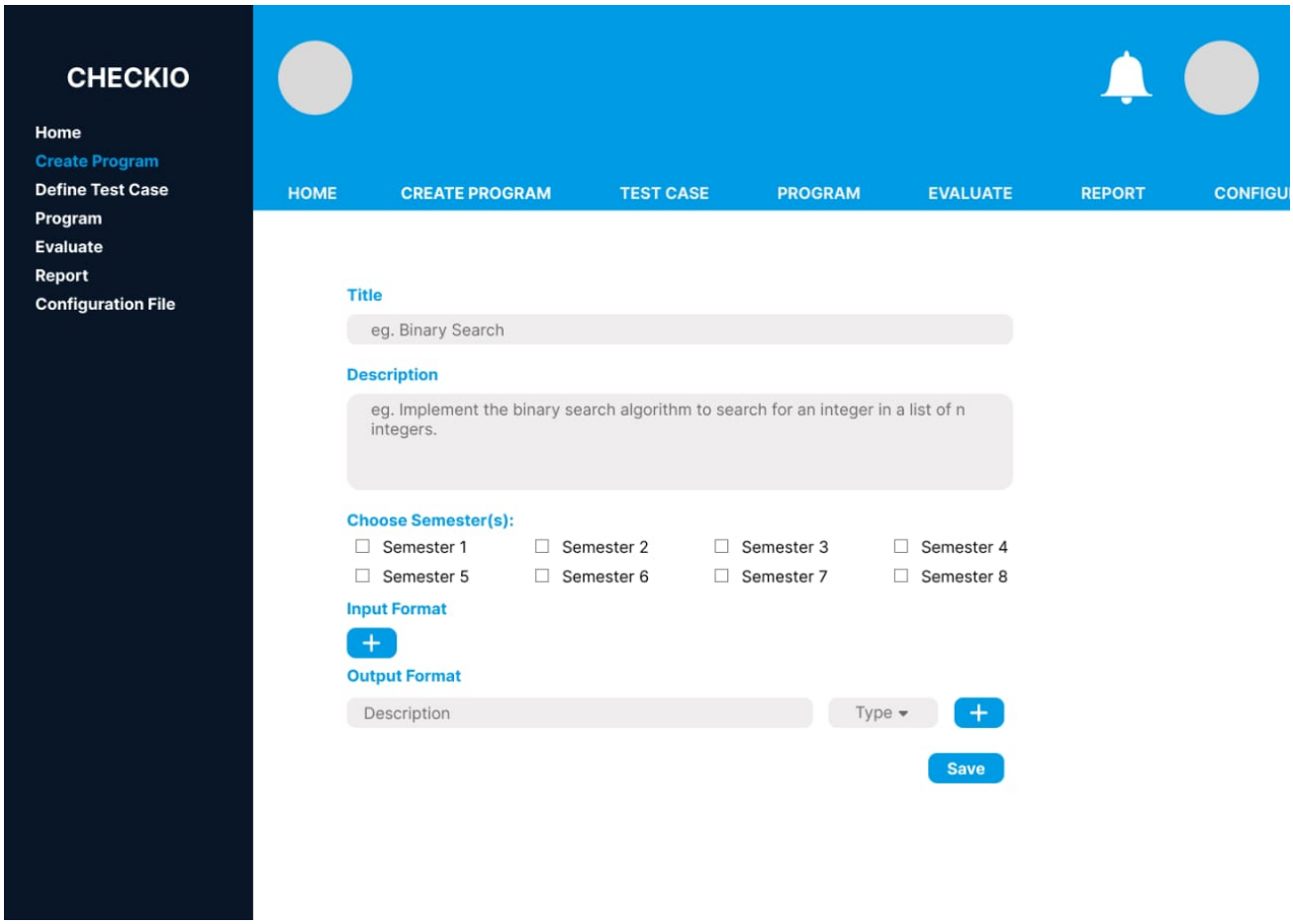
6 Algorithm Design

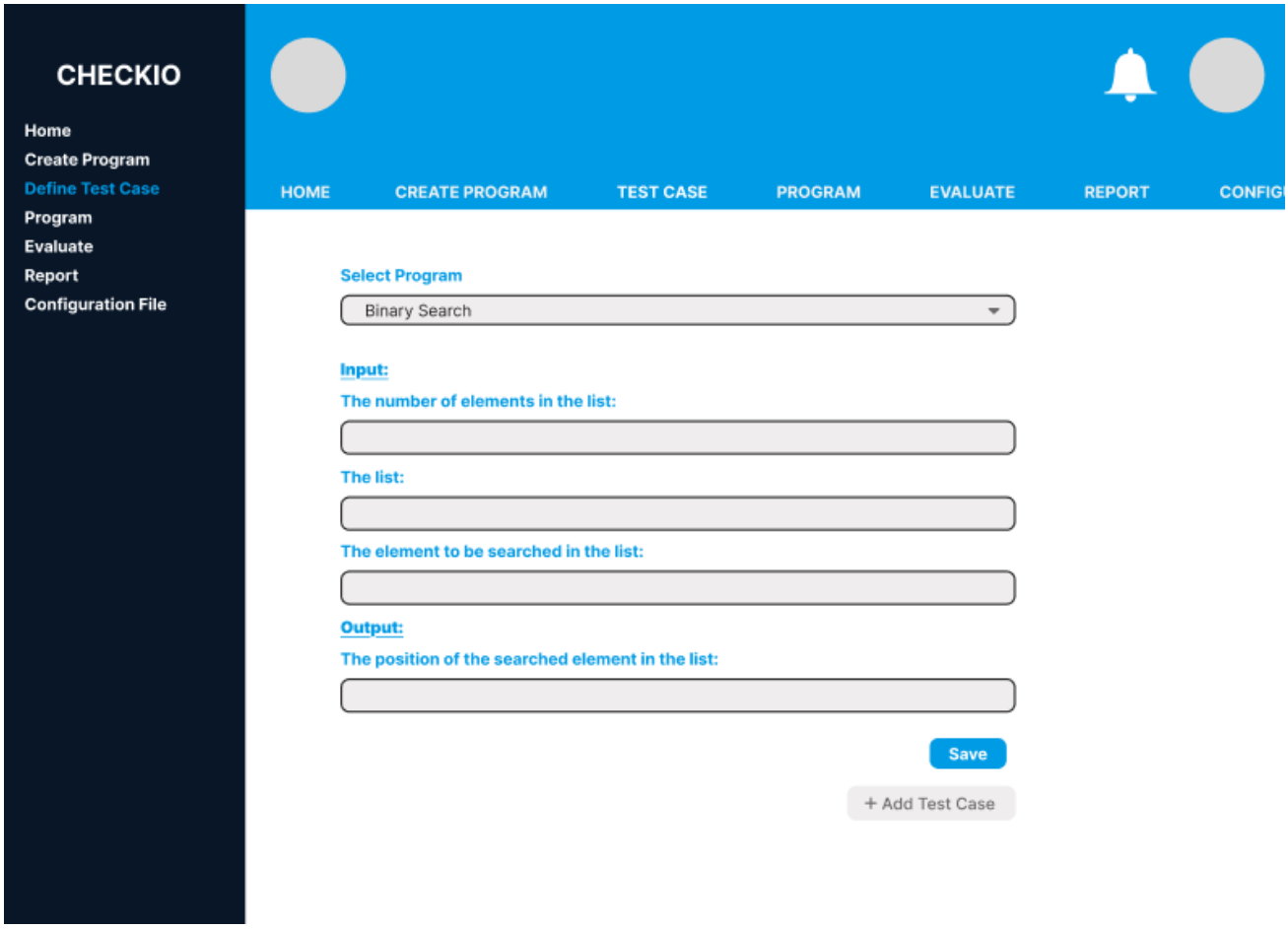
Description: This section includes the algorithm(s) used in the framework in textual format.

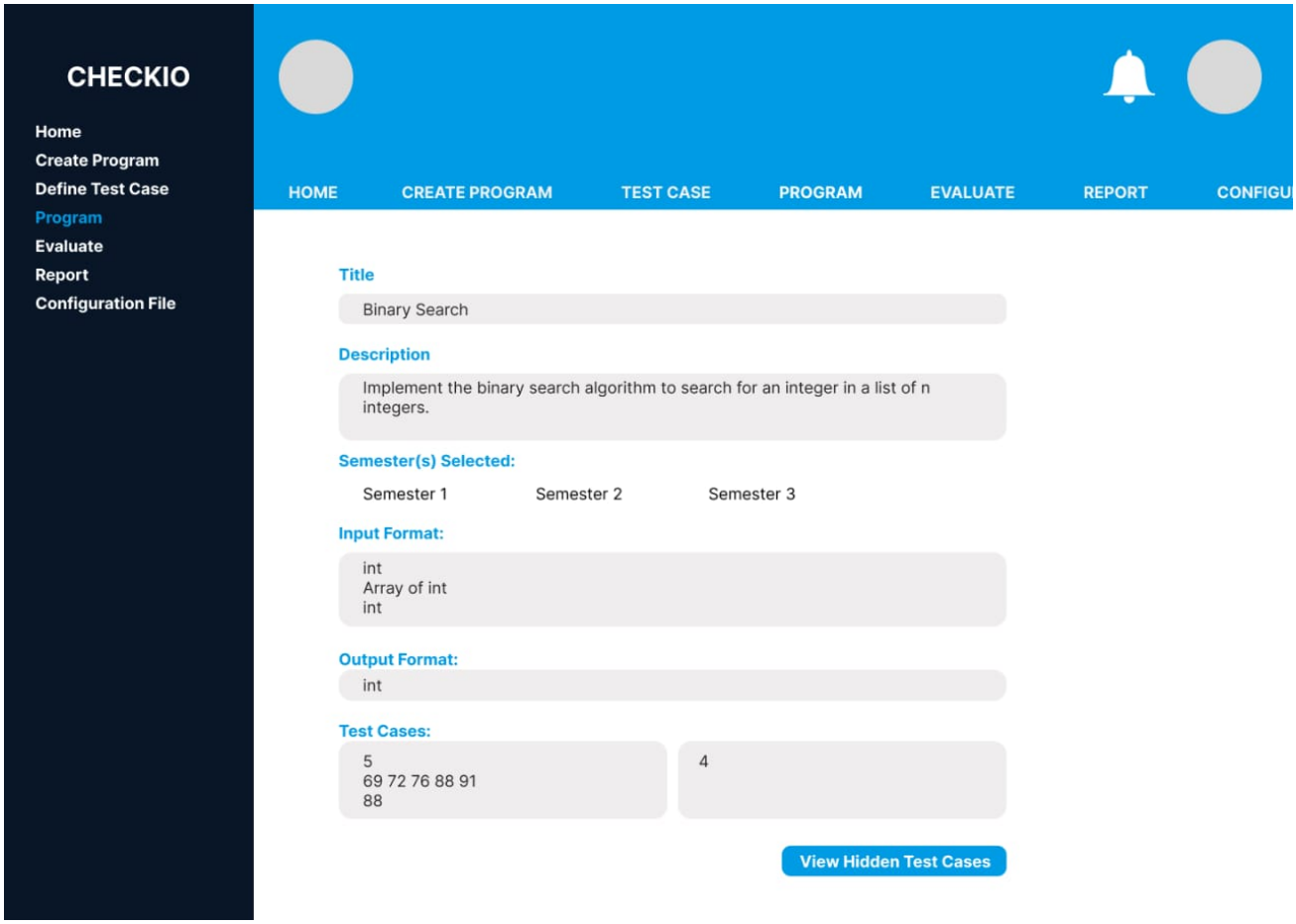
```
Algorithm: Program Evaluation System

Input:
test_cases: A list of test cases defined by the teacher, each containing:
  input_data: Input values for the program.
  expected_output: The correct output for the given input.
  student_code: The code submitted by the student as a string.

Output:
Evaluation report containing:
  Overall success or failure.
  Detailed results for each test case (pass/fail).
  Identified errors (if any) and suggestions.

Steps:
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 (e.g., type mismatch or incorrect logic).
      Generate suggestions, such as a sample corrected output or identifying discrepancies.

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.
```

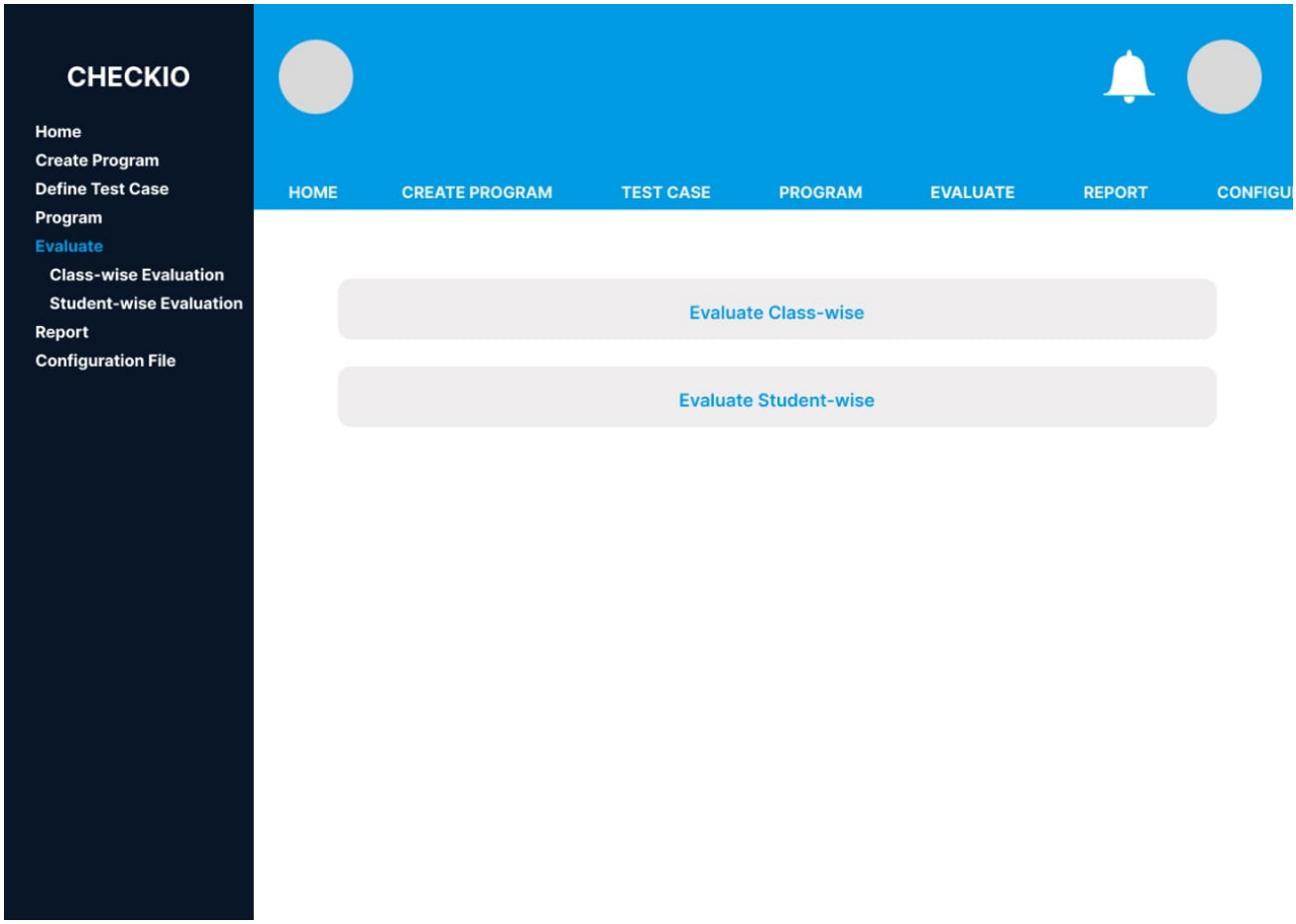
7 Interface Design

Description: The interface design section showcases the UI diagrams for the system, detailing how users interact with the application.




CHECKIO
Home
Create Program
Define Test Case
Program
Evaluate
Report
Configuration File
HOME
CREATE PROGRAM
TEST CASE
PROGRAM
EVALUATE
REPORT
CONFIG
Select Program
Binary Search
Input:
The number of elements in the list:
The list:
The element to be searched in the list:
Output:
The position of the searched element in the list:
Save
+ Add Test Case


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


CHECKIO
Home
Create Program
Define Test Case
Program
Evaluate
Class-wise Evaluation
Student-wise Evaluation
Report
Configuration File
HOME
CREATE PROGRAM
TEST CASE
PROGRAM
EVALUATE
REPORT
CONFIGU
Evaluate Class-wise
Evaluate Student-wise