## **Project Day 4 – Evaluation**

**Date:** 13-09-2025

**Name**: **Abdul Samad**

**Title: E-Learning Platform with Auto Evaluation.**

### **1. Objective**

Implement and test the automatic evaluation component of the platform.  
 Today’s focus was on building a Test Case Runner that can execute student submissions against predefined test cases and generate an Evaluation Result.

**2. Key Deliverables**

* Design and implement a **TestCaseRunner** module.
* Integrate the runner with the existing EvaluationEngine.
* Store evaluation results (score, remarks, logs) in the database.
* Provide basic logging and error handling.

**3. Technology Stack (Specific to Evaluation)**

| **Layer** | **Technology** | **Reason** |
| --- | --- | --- |
| Language | **Java 17** | Strong type system, good process control for running external commands. |
| Execution Env. | **Java ProcessBuilder** / Runtime.exec() | To compile and run submitted code safely in a sandboxed manner. |
| Testing | **JUnit 5** (for writing instructor test cases) | Widely used Java testing framework, easy integration with runner. |
| Database | MySQL / PostgreSQL | Store evaluation scores and feedback. |
| Security | Custom sandbox (limited file permissions) | Prevent malicious code execution. |

**4. Class Design Updates**

**TestCaseRunner.java**

public class TestCaseRunner {

public EvaluationResult runTests(Assignment assignment, List<TestCase> tests) {

int passed = 0;

for (TestCase t : tests) {

boolean result = executeSingleTest(assignment.getFilePath(), t);

if (result) passed++;

}

int score = (int) ((passed / (double) tests.size()) \* 100);

return new EvaluationResult(assignment.getAssignmentId(), score,

passed + " of " + tests.size() + " tests passed");

}

private boolean executeSingleTest(String filePath, TestCase test) {

// Compile and run the student's Java file

// Compare output to expected output

// Return true if matches

return true; // placeholder

}

}

**TestCase.java**

public class TestCase {

private final String input;

private final String expectedOutput;

// constructor, getters

}

**Integration with EvaluationEngine**

public class EvaluationEngine {

private final TestCaseRunner runner = new TestCaseRunner();

public EvaluationResult evaluate(Assignment assignment) {

List<TestCase> predefinedTests = TestCaseRepository.loadForAssignment(assignment);

return runner.runTests(assignment, predefinedTests);

}

}

**5. Workflow**

1. **Instructor** uploads a set of predefined test cases to the database or a JSON file.
2. **Student** uploads an assignment (Day 3 upload API).
3. **EvaluationEngine** triggers TestCaseRunner:  
   * Compiles the student’s code.
   * Executes with each test input.
   * Captures stdout/stderr and compares with expected output.
4. **Result** is stored in the EvaluationResult table and returned to the frontend.

**6. Testing Activities**

* Created three sample programming tasks (e.g., “Sum of Two Numbers”) and their test cases.
* Submitted both **correct** and **faulty** solutions to validate scoring:  
  + Correct solution scored **100 %**.
  + Faulty solution scored **60 %** with detailed “test X failed” remarks.
* Verified that infinite loops/timeouts are safely terminated with a process timeout.

**7. Security & Safety Measures**

* Limited execution time using ProcessBuilder’s wait with timeout.
* Disallowed file system access and external network calls by running code in a restricted directory.
* Sanitized all student input to avoid command injection.

**8. Challenges Encountered**

* Handling compilation errors gracefully required capturing compiler stderr and returning it in the remarks.
* Ensuring portability across different OS environments (Windows vs Linux).

**9. Next Steps**

* Build a simple **results dashboard** for students to view evaluation scores and feedback.
* Add support for multiple programming languages (Python/JavaScript) using containerized runners.
* Implement plagiarism detection as an optional feature.