# Student Evaluator System - Requirements Specification

## 1. Introduction

### 1.1 Purpose

This document specifies the requirements for a Student Evaluator System, a personal learning project designed to automate the process of evaluating student Java programming assignments through JUnit testing.

### 1.2 Project Description

The Student Evaluator System will allow for the upload of Java files, compile them, run predefined JUnit tests against them, and store evaluation results. The system will be implemented as a RESTful API with a CLI interface, storing structured data in MySQL and unstructured logs in DynamoDB.

### 1.3 Scope

This system will be developed as a personal learning project to run locally. It will focus on automated evaluation of Java programming assignments using JUnit tests, with data storage in both SQL and NoSQL databases.

## 2. System Requirements

### 2.1 Functional Requirements

#### 2.1.1 Assignment Management

* FR-01: The system shall allow uploading of single Java files for evaluation.
* FR-02: The system shall allow uploading of JUnit test files separately.
* FR-03: The system shall associate test files with specific assignments.
* FR-04: The system shall store submitted assignments for future reference.

#### 2.1.2 Assignment Evaluation

* FR-05: The system shall compile submitted Java files.
* FR-06: The system shall run JUnit tests against compiled submissions.
* FR-07: The system shall determine a pass/fail result based on test outcomes.
* FR-08: The system shall capture and store evaluation results.
* FR-09: The system shall log details of the evaluation process.

#### 2.1.3 Result Management

* FR-10: The system shall store evaluation scores in MySQL.
* FR-11: The system shall store unstructured evaluation logs in DynamoDB.
* FR-12: The system shall implement a HashMap to map students to their scores using student hashcodes.
* FR-13: The system shall provide API endpoints to retrieve evaluation results.

#### 2.1.4 User Interface

* FR-14: The system shall provide a RESTful API for all operations.
* FR-15: The system shall support CLI (Command Line Interface) as a consumer of the API.

#### 2.1.5 Behavior-Driven Development

* FR-16: The system shall implement automated scenarios for assignment evaluation.

#### 2.1.6 DevOps Integration

* FR-17: The system shall include CI/CD pipelines with test execution.
* FR-18: The system shall support demonstration deployment.

### 2.2 Non-Functional Requirements

#### 2.2.1 Performance

* NFR-01: The system shall compile and evaluate a standard Java assignment within 30 seconds.
* NFR-02: The API shall respond to requests within 5 seconds under normal load.

#### 2.2.2 Usability

* NFR-03: The CLI shall provide clear feedback on operations performed.
* NFR-04: The API shall return clear error messages for failed operations.

#### 2.2.3 Reliability

* NFR-05: The system shall handle compilation errors gracefully and provide meaningful feedback.
* NFR-06: The system shall maintain data integrity between MySQL and DynamoDB storage.

#### 2.2.4 Security

* NFR-07: The system shall validate all inputs to prevent code injection attacks.

#### 2.2.5 Maintainability

* NFR-08: The system shall be designed with clear separation of concerns.
* NFR-09: The system code shall include appropriate documentation.

## 3. System Architecture Overview

The Student Evaluator System will consist of the following major components:

1. Upload API: RESTful endpoints for submitting assignments and tests
2. EvaluationEngine: Core component for compiling code and running JUnit tests
3. Storage Subsystem: Management of data across MySQL and DynamoDB
4. CLI Interface: Command-line interface for interacting with the API

## 4. Data Requirements

### 4.1 Entities

#### 4.1.1 Student

* Student ID (primary key)
* Name
* Other relevant attributes

#### 4.1.2 Assignment

* Assignment ID (primary key)
* Description
* Submission date
* Associated test case information

#### 4.1.3 Evaluation

* Evaluation ID (primary key)
* Student ID (foreign key)
* Assignment ID (foreign key)
* Score/Result
* Timestamp
* Status

### 4.2 Data Storage

#### 4.2.1 MySQL

* Store structured data:
  + Student information
  + Assignment information
  + Evaluation scores and results

#### 4.2.2 DynamoDB

* Store unstructured data:
  + Compilation logs
  + Test execution logs
  + Error messages

#### 4.2.3 In-Memory Storage

* HashMap implementation for mapping student hashcodes to their scores
* Used for quick lookup of student evaluation results

## 5. Interface Requirements

### 5.1 API Endpoints

The system shall provide RESTful API endpoints for:

1. Assignment Management
   * Upload assignment
   * Upload test cases
   * List assignments
2. Evaluation
   * Trigger evaluation
   * Get evaluation status
3. Results
   * Retrieve evaluation results
   * Get detailed logs

### 5.2 CLI Interface

The CLI shall provide commands for:

1. Assignment submission
2. Test case upload
3. Evaluation triggering
4. Result retrieval

## 6. Development Roadmap (10-Day Plan)

Based on the provided outline:

1. Day 1 – Requirements
   * Assignment upload, evaluation requirements documentation
2. Day 2 – Design
   * Student, Assignment, EvaluationEngine classes design
3. Day 3 – Java
   * Upload API implementation
4. Day 4 – Evaluation
   * Test case runner implementation
5. Day 5 – MySQL
   * Implementation of score storage in MySQL
6. Day 6 – DynamoDB
   * Implementation of unstructured log storage in DynamoDB
7. Day 7 – Data Structures
   * HashMap implementation for mapping students to scores
8. Day 8 – BDD
   * Automated evaluation scenario implementation
9. Day 9 – DevOps
   * CI/CD pipeline setup with test execution
10. Day 10 – Demo
    * Final deployment and demonstration

## 7. Constraints and Assumptions

### 7.1 Constraints

* The system will run locally for development and learning purposes
* No specific frameworks are mandated for implementation

### 7.2 Assumptions

* Java Development Kit (JDK) is available for compilation
* MySQL and DynamoDB (or compatible alternatives) are available locally
* Students submit syntactically correct Java files
* JUnit test files are properly formatted and valid

## 8. Glossary

* API: Application Programming Interface
* BDD: Behavior-Driven Development
* CI/CD: Continuous Integration/Continuous Deployment
* CLI: Command-Line Interface
* JUnit: Java Unit Testing Framework
* REST: Representational State Transfer