

# **Software Testing Plan**

## **1. Introduction**

Software testing is a critical activity in preventing defects, ensuring quality, and validating that the developed system meets specified requirements. This Software Testing Plan defines the testing approach, objectives, scope, and activities for the Real-Time Collaborative Online Integrated Development Environment (RCO-IDE). The purpose of this document is to ensure that the system functions correctly, securely, and efficiently before academic submission and deployment.

## **2. Objectives of Testing**

The primary objectives of software testing for this project are:

- To verify that all functional requirements specified in the SRS are correctly implemented.
- To ensure the reliability and stability of real-time collaboration features.
- To validate secure and accurate code execution.
- To identify defects early and reduce rework.
- To ensure the system meets performance and usability expectations.

## **3. Test Strategy**

The testing strategy for the RCO-IDE project follows a layered approach, covering different levels of testing:

### **3.1 Unit Testing**

Individual components such as authentication modules, editor synchronization logic, and API endpoints are tested independently to verify correctness.

### 3.2 Integration Testing

Integration testing ensures that different modules such as frontend, backend, database, WebSocket services, and execution engine work together as expected.

### 3.3 System Testing

System testing validates the complete system behavior in an environment that closely resembles real-world usage. It focuses on end-to-end workflows such as user login, session creation, collaborative editing, and code execution.

### 3.4 Acceptance Testing

Acceptance testing verifies that the system meets academic project requirements and user expectations. This testing ensures readiness for final evaluation.

## 4. Test Environment

The test environment is configured to simulate the actual operating environment of the system.

- Hardware: Standard laptop or desktop system with minimum 4GB RAM.
- Software: Modern web browsers (Chrome, Firefox), Node.js runtime, MongoDB database.
- Network: Stable internet connection with WebSocket support.
- Test Data: Sample user accounts, code snippets in Python and JavaScript, and test input files.

## 5. Test Cases

Test cases are designed to validate both normal and exceptional system behavior. Each test case includes test ID, description, input data, expected output, and pass/fail criteria.

Example Test Cases:

- Verify user registration with valid and invalid inputs.
- Validate real-time synchronization between multiple users.
- Test secure execution of Python and JavaScript code.
- Verify role-based access control for Host and Guest users.
- Check system behavior under invalid input or execution errors.

## **6. Defect Management**

Defect management ensures that identified issues are documented, tracked, and resolved systematically.

- Defect Logging: Defects are recorded with severity, priority, and reproduction steps.
- Defect Tracking: Defects are tracked until resolution using issue tracking tools.
- Defect Resolution: Developers fix defects and submit them for retesting.
- Defect Closure: Defects are closed after successful verification.

## **7. Test Deliverables**

The testing process produces the following deliverables:

- Test Plan Document
- Test Cases and Test Data
- Defect Reports
- Test Summary Report

## **8. Conclusion**

The Software Testing Plan provides a structured framework for validating the quality, functionality, and reliability of the Real-Time Collaborative Online IDE. Through systematic testing at multiple levels, the project ensures compliance with requirements and readiness for academic evaluation. Effective testing contributes significantly to delivering a robust and dependable software system.