## 1. Problem Statement & Objectives

### Problem Statement

Web applications require rigorous testing to ensure functionality, security, and performance. Manual testing is time-consuming and prone to human error. An automated testing framework is needed to streamline the testing process, improve test accuracy, and ensure efficient regression testing.

### Objectives

* Develop an automated testing framework for web applications.
* Reduce manual effort and enhance test execution speed.
* Improve test coverage and accuracy.
* Ensure compatibility across multiple browsers and platforms.
* Enable continuous integration and automated reporting.

## 2. Use Case Diagram & Descriptions

### Use Case Diagram

* Actors: Tester, Developer, CI/CD System
* Use Cases: Execute Test Cases, Generate Reports, Integrate with CI/CD, Maintain Test Scripts, Manage Test Data

### Description

* **Tester:** Runs automated test cases, analyzes reports, and modifies test scripts.
* **Developer:** Debugs failures, updates application code, and reviews test outcomes.
* **CI/CD System:** Automatically triggers test executions and collects reports.

## 3. Functional & Non-Functional Requirements

### Functional Requirements

* Test script execution for UI and API testing.
* Support for Selenium WebDriver and REST API testing tools.
* Test data management and validation mechanisms.
* Integration with JIRA and CI/CD tools (e.g., Jenkins, GitHub Actions).
* Detailed test reports and logs.

### Non-Functional Requirements

* Scalability to support large test suites.
* Cross-browser compatibility testing.
* High reliability and maintainability.
* Fast execution and minimal test flakiness.
* Secure storage of test data and credentials.

## 4. Software Architecture

* **Architecture Style:** Modular & Hybrid (Page Object Model + Data-Driven + Keyword-Driven)
* **Components:**
  + Test Execution Engine
  + Test Data Manager
  + Report Generator
  + Integration Module
  + Logging & Exception Handling

## 5. Database Design & Data Modeling

### ER Diagram

* **Entities:** Test Cases, Test Results, Test Data, Users
* **Relationships:** One-to-Many (Users to Test Cases, Test Cases to Test Results)

### Logical & Physical Schema

* **Tables:** test\_cases, test\_results, test\_data, users
* **Keys:** Primary (id), Foreign (user\_id, case\_id)

## 6. Data Flow & System Behavior

### DFD (Data Flow Diagram)

* **Context Diagram:** Shows interaction between users, test scripts, and report generation.
* **Level-1 DFD:** Details the flow of test execution, data input, and result logging.

### Sequence Diagram

* Illustrates step-by-step execution from test script invocation to report generation.

### Activity Diagram

* Shows process flow from script execution to test validation.

### State Diagram

* Represents states of a test case (Pending, Running, Passed, Failed, Skipped).

### Class Diagram

* Defines classes such as TestCase, TestSuite, TestExecutor, ReportGenerator.

## 7. UI/UX Design & Prototyping

### Wireframes & Mockups

* Dashboard displaying test results and execution logs.
* Test execution interface for manual triggering.
* Configuration UI for setting up test environments.

### UI/UX Guidelines

* Clear navigation for users.
* Dark/light mode for usability.
* Accessible design principles (contrast, readability, keyboard navigation).

## 8. System Deployment & Integration

### Technology Stack

* **Backend:** Python (Pytest, Selenium, Requests)
* **Frontend:** React (for test reporting dashboard)
* **Database:** MySQL/PostgreSQL
* **CI/CD:** Jenkins, GitHub Actions

### Deployment Diagram

* Illustrates cloud-based or on-premise distribution of test execution environment.

### Component Diagram

* High-level system dependencies, including integrations with JIRA and CI/CD tools.

## 9. Additional Deliverables

### API Documentation

* Lists available API endpoints for triggering tests and fetching results.

### Testing & Validation

* **Unit Tests:** Validate individual framework components.
* **Integration Tests:** Ensure seamless connectivity with CI/CD.
* **User Acceptance Testing:** Ensure usability for testers and developers.

### Deployment Strategy

* **Hosting:** Cloud-based test execution (AWS, Azure, or on-premise setup).
* **Pipelines:** Automated test execution post-code deployment.
* **Scaling:** Support for parallel execution to handle multiple test runs efficiently.