**Project Report: MEM Automation Using Selenium Framework**

### **Title Page**

**Project Title:** MEM Website Automation Using Selenium Framework  
 **Submitted By:** Krutik Dipakbhai Aghera (210170107017)  
 **Institution:** Vishwakarma Government Engineering College, Ahmedabad  
 **University:** Gujarat Technological University  
 **Submission Date:** April 2025

### **Certificate**

This is to certify that **Krutik Dipakbhai Aghera** has successfully completed the project entitled **"MEM Website Automation Using Selenium Framework"** under the supervision of **Prof. Nakul Dave** in partial fulfilment for the degree of Bachelor of Engineering in Computer Engineering at Vishwakarma Government Engineering College, affiliated with Gujarat Technological University.

### **Declaration**

I hereby declare that the project work titled **"MEM Website Automation Using Selenium Framework"** is the result of my own work carried out under the guidance of Prof. Nakul Dave and Ishita Bhatt (External Guide, Meditab). The report has not been submitted elsewhere for the award of any other degree.

**Name:** Krutik Dipakbhai Aghera  
 **Sign:** \_\_\_\_\_\_\_\_\_\_\_\_

### **Acknowledgement**

I would like to sincerely thank **Mr. Rajesh Shah**, Automation Department Manager at Meditab, and **Mr. Bhavik Tikudiya**, Sr. Lead Automation Engineer at Meditab, for their mentorship. I am grateful to **Prof. Karan P. Bhatt** for academic guidance and to all my colleagues and the development team for their continued support.

### **Abstract**

This project focuses on building a robust automation testing framework for the Meditab Employee Management (MEM) portal using **Selenium, Java, TestNG, Maven**, and **Extent Reports**. The objective is to automate key task workflows such as task creation, editing, and verification using dynamic scripts that are reusable and maintainable. The project is designed with scalability in mind, providing a clear separation between test logic, page elements, and scenario workflows.

Through our custom-built framework, we aim to:

* Reduce repetitive manual testing efforts
* Increase test coverage with dynamic scripts
* Maintain flexibility using Map<String, String> driven logic
* Handle all types of UI components including dropdowns and date pickers dynamically

The result is a modular, maintainable, and powerful test suite capable of executing a variety of functional and regression tests with minimal changes to code.

### **Table of Contents**

* **CERTIFICATE** .............................................................................................. II
* **INTERNSHIP CERTIFICATE** ................................................................... III
* **DECLARATION** ........................................................................................... IV
* **ACKNOWLEDGMENT** ................................................................................. V
* **ABSTRACT** ..................................................................................................... VI
* **LIST OF FIGURES** ..................................................................................... VII
* **LIST OF TABLES** ...................................................................................... VIII

### **CHAPTER 1 – OVERVIEW OF THE COMPANY**

1.1 History  
 1.2 Different Products  
 1.3 Capacity of Company

### **CHAPTER 2 – INTRODUCTION**

2.1 Project Overview  
 2.2 Purpose of Project  
 2.3 Scope of Project  
 2.4 Objective  
 2.5 Technology and Tools

### **CHAPTER 3 – PROJECT STRUCTURE AND MANAGEMENT**

3.1 Project Planning  
 3.2 Project Scheduling and Representation (Gantt Chart)

### **CHAPTER 4 – PROJECT AND SYSTEM REQUIREMENT STUDY**

4.1 User Characteristics  
 4.2 Hardware and Software Requirements  
 4.3 Assumptions and Dependencies

### **CHAPTER 5 – SYSTEM/RESOURCE ANALYSIS**

5.1 Study of Current System  
 5.2 Problems and Weakness  
 5.3 Requirements of New System  
 5.4 Feasibility Study  
 5.5 Functions of System  
 5.6 Activity Diagram  
 5.7 Use Case Diagram  
 5.8 Sequence Diagram

### **CHAPTER 6 – AXES DESIGN**

6.1 Login Page and Interface Design

### **CHAPTER 7 – IMPLEMENTATION PLANNING**

7.1 Implementation Environment  
 7.2 Program/Modules Specification  
 7.3 Coding Standards

### **CHAPTER 8 – TESTING**

8.1 Testing Plan  
 8.2 Testing Strategy

### **CHAPTER 9 – CONCLUSION**

9.1 Problems Encountered and Solutions  
 9.2 Summary of Project Work

### **CHAPTER 10 – LIMITATIONS AND FUTURE ENHANCEMENT**

10.1 Limitations  
 10.2 Future Enhancement

### **CHAPTER 11 – BIBLIOGRAPHY AND REFERENCES**

### **Appendices**

Appendix 1: Plagiarism Report  
 Appendix 2: Weekly Log Book  
 Appendix 3: Attendance Sheet  
 Appendix 4: Feedback Form

### **Chapter 1: Introduction**

**1.1 Background** Manual testing of dynamic enterprise applications like the Meditab Employee Management (MEM) portal is time-consuming, error-prone, and inefficient in modern agile workflows. As the complexity of web applications increases, the need for reliable, maintainable, and reusable test automation frameworks becomes imperative.

**1.2 Purpose of the Project** This project was undertaken to automate repetitive user actions in the MEM portal—especially task creation, editing, and management functionalities—using Selenium and Java. The goal is to build a framework that allows:

* Reusability through modular functions
* Flexibility through dynamic locators
* Clean architecture for easier maintenance

**1.3 Objectives**

* Automate core MEM workflows: login, task assignment, task editing
* Design a flexible locator strategy using Map<String, String>
* Implement a reporting system using Extent Reports
* Maintain a DRY (Don’t Repeat Yourself) codebase by refactoring common logic

**1.4 Scope of Work** This project covers:

* Automation of task management and login module
* Page object model-based framework design
* Support for dynamic components (dropdowns, date pickers)
* Implementation of data-driven testing
* Structured HTML reporting and debugging support

**[Insert Screenshot Placeholder: MEM Home Page Screenshot]**

**1.5 Benefits**

* Reduces manual testing efforts and errors
* Enables rapid test cycles before deployments
* Offers scalability for future modules

**[Insert Screenshot Placeholder: Sample Test Execution]**

**1.6 Limitations**

* Automation scripts require maintenance on UI changes
* Complex dynamic components may need custom handling

### **Chapter 2: Introduction**

**2.1 Project Overview** The MEM Automation project is designed to simplify and accelerate testing of the internal Meditab Employee Management (MEM) portal. With features like dynamic task creation, editing, and user-based task management, this automation framework is essential for reducing repetitive manual testing. Built with Java and Selenium WebDriver, this automation framework integrates TestNG for testing workflows and ExtentReports for professional reporting.

**2.2 Purpose of Project** The core purpose is to build a test automation framework that allows flexible test execution, reusable scripts, and maintainable logic. The framework improves reliability and scalability in testing processes by reducing human error and manual effort in test cycles.

**2.3 Scope of Project**

* Automating key workflows on the MEM web portal
* Supporting dynamic elements such as dropdowns, input fields, and date pickers
* Providing reporting with screenshots using ExtentReports
* Structuring the project to allow scalability and maintainability

**2.4 Objective**

* Automate task assignment, modification, and validation modules
* Handle UI components dynamically with generalized logic
* Maintain execution logs and screenshot-based reporting
* Ensure cross-browser execution compatibility

**2.5 Technology and Tools**

* **Java:** Backend programming language used for writing automation logic
* **Selenium WebDriver:** Browser automation tool
* **TestNG:** Test execution and assertions framework
* **Maven:** Dependency and build management
* **Extent Reports:** Customized reporting tool for viewing test execution results
* **IDE:** Eclipse or IntelliJ IDEA for development

**[Insert Screenshot Placeholder: Eclipse IDE with Project Structure]**

### **Chapter 3: Project Structure and Management**

**3.1 Project Planning**

The development approach includes multiple stages:

* **Feasibility Analysis:** Assessing automation readiness
* **Requirement Gathering:** Functional and non-functional testing needs
* **Manual & Automation Scope Split**
* **Tool Selection:** Selenium, Maven, Jenkins, TestNG
* **Test Strategy:** Unit, Interface, Security, Cross-platform, Performance
* **Test Planning:** Estimation, Scheduling, Risk Handling, Communication Flow
* **Knowledge Transfer:** Application walkthrough, manuals, test case creation
* **Test Environment Setup:** Software stack, global test setup
* **Scripting:** Identifying reusable components, writing modular scripts
* **Execution:** Logging, Reporting, Batch execution
* **Maintenance:** Update test scripts, improve accuracy

**3.1.2 Cost Estimation – COCOMO Model** Used for estimating development effort. Calculated via Function Points (FP) and Lines of Code (LOC).

**Table 3.1: COCOMO Calculation Table** [Insert Table Placeholder: COCOMO Cost Estimation Table here]

**Figure 3.1: Questions dependent on the project** [Insert Diagram Placeholder]

**Figure 3.2: Function Points Calculation** [Insert Diagram Placeholder]

**Figure 3.3: LOC/FP Calculation** [Insert Diagram Placeholder]

**3.2 Project Scheduling (Gantt Chart)**

* Detailed bar chart with phase durations
* Tracks progress, dependencies, resource allocation

**Figure 3.4: Gantt Chart** [Insert Gantt Chart Placeholder here]

### **Chapter 4: Project and System Requirement Study**

**4.1 User Characteristics** The following types of users interact with the MEM system:

* **Hardware Developers:** To validate application compatibility with target hardware.
* **Production Team:** For final round of verification and approval.
* **Sales Team:** To demonstrate core features to prospective clients.
* **Clients:** End-users after tailored customization based on business needs.

**4.2 Hardware and Software Requirements**

**Software Requirements:**

* Eclipse IDE (or compatible Java IDE)
* Java JDK (version 1.8 or above)
* Selenium WebDriver
* TestNG framework
* Maven Repository for dependency management

**Hardware Requirements:**

* Minimum 8 GB RAM
* CPU with minimum 3.5 GHz processing power
* Stable internet connectivity for online test scenarios

**4.3 Assumptions and Dependencies**

**Assumptions:**

* The testing environment will not run any conflicting background applications.
* Autolock or system sleep settings are disabled.

**Dependencies:**

* Reusable methods and utilities (clearFieldWithActions, clickUsingJS, etc.)
* Locator map with dynamic XPath/CSS locators
* External property/config files for managing environment-specific data
* Test data in Excel/CSV or JSON format for data-driven testing
* Logging/reporting utilities (ExtentReports)

### **Chapter 5: System/Resource Analysis**

**5.1 Study of Current System/Resource** The current MEM system relies heavily on manual validation for each functional update. There is no unified utility to automate tasks or ensure dynamic execution of repeatable scripts. Additionally, the application under development often undergoes UI or logic changes that render manual regression highly inefficient.

**5.2 Current System Problems and Weakness**

* Inability to handle real-time failures like crashes or power interruptions.
* High risk of inconsistency when scenarios are updated in the application.
* Browser compatibility and driver session issues in manual tests.

**5.3 Requirements of New System**

**Functional Requirements:**

* Robust failure handling and crash recovery
* Script execution in batch mode with minimal/no human input
* Filtering and batching of test scripts by feature/module
* Dynamic interaction with dropdowns, calendars, modals, and AJAX-based elements

**Logging and Reporting Enhancements:**

* Detailed logs with exception tracing
* Screenshot capture on failure and timestamps
* Exportable HTML reports with status highlighting

**Non-Functional Requirements:**

* **Reliability:** Continuous availability during execution
* **Performance:** Quick script execution and reporting
* **Scalability:** Adaptable for more modules and concurrent executions
* **Usability:** Modular and reusable components for rapid development

**5.4 Feasibility Study**

* **Technical Feasibility:** Built using open-source, industry-standard tools
* **Operational Feasibility:** Can be run by QA testers with basic Selenium knowledge
* **Economic Feasibility:** Low-cost implementation using Java and Maven

**5.5 New System Features**

* Execute automation scripts reliably even during mid-execution issues
* Allow command-line based or Jenkins-triggered execution
* Provide runtime logs and auto-generated test summaries
* GUI-friendly result presentation for management review

**5.6 Functions of the System**

* **Admin Functions:** Add/update users, assign tasks, monitor status
* **Employee Functions:** View assigned tasks, update status, assign subtasks

**Figure 5.1: Activity Diagram for Automation Testing** [Insert Activity Diagram Placeholder here]

**Figure 5.2: Use Case Diagram for MEM Portal** [Insert Use Case Diagram Placeholder here]

**Figure 5.3: Framework Architecture for Automation** [Insert Framework Diagram Placeholder here]

**Figure 5.4: Sequence Diagram for Task Flow** [Insert Sequence Diagram Placeholder here]

### **Chapter 6: AXES Design**

**6.1 Login Page and Interface Design** The login page of the MEM automation portal is the gateway for authenticated access. It includes validation mechanisms for correct credentials, handles failed login attempts, and provides options for password recovery. This section showcases screenshots and descriptions of different login functionalities tested via automation.

**Key UI Components Automated:**

* Username and Password fields (input validation)
* Login button (click action and success/failure condition)
* "Forgot Password" link (display and reset flow)
* Dashboard/Home page rendering post login
* Error messages and alerts (toast or modal-based)

**6.1.1 Login Attempt Validation** Automated scripts input valid and invalid credentials to validate user login logic, session handling, and redirection to the homepage or error screen.

**Figure 6.1: Login Attempt Validation** [Insert Screenshot Placeholder]

**6.1.2 Forgot Password Flow** Scripts simulate multiple wrong login attempts to check if the "Forgot Password" link becomes visible and whether the flow properly resets credentials.

**Figure 6.2: Forgot Password Flow** [Insert Screenshot Placeholder]

**6.1.3 Invalid Username Test** If the user enters an incorrect or non-existent username and clicks on "Forgot Password," the system must block the reset request.

**Figure 6.3: Incorrect Credential Test** [Insert Screenshot Placeholder]

**6.1.4 Reset Password Confirmation** Valid username entries should allow a reset with a confirmation popup shown upon clicking reset.

**Figure 6.4: Update Password UI** [Insert Screenshot Placeholder]

**6.1.5 Post Login Homepage Rendering** Upon valid login, users should be redirected to their personalized dashboard. The automation framework verifies this transition and UI rendering.

**Figure 6.5: Home Page View Post Login** [Insert Screenshot Placeholder]

### **Chapter 7: Implementation Planning**

**7.1 Implementation Environment** Automation testing requires a controlled and consistent test environment. Based on the complexity and volume of MEM test scenarios, the following setups were utilized:

* **High-Performance Machine:** Minimum 8 GB RAM, multi-core CPU to execute parallel test scripts
* **Dedicated Test Environment:** For running test scripts without interruptions from other processes
* **Utility GUI:** For batch execution, script import/export, and real-time report monitoring
* **Browser Support:** ChromeDriver, EdgeDriver, and compatibility with Firefox to support cross-browser testing

Test data and configuration files are stored in a central, accessible directory and injected during runtime via property loaders. This helps simulate real-world test conditions like multi-user access and dynamic form interactions.

**7.2 Program/Module Specifications** The automation scripts have been organized into modular test cases that align with MEM portal modules:

* Login Module
* Task Assignment Module
* Task Editing Module
* Search and Filter Module
* User Profile Update Module

Each module is scripted with reusable methods and dynamic locators sourced from a centralized locator map, ensuring maintainability and clarity.

**7.3 Coding Standards** The following practices are strictly observed in the codebase:

* **CamelCase Naming Convention:** For class, method, and variable names
* **Single Responsibility Principle:** Each method handles one distinct function
* **Modular Methods:** Shared logic like clearing fields, scrolling, or dropdown selection is abstracted
* **Code Comments:** Brief inline comments are added for better understanding
* **Error Handling:** Try-catch blocks with fallback mechanisms where applicable

These standards ensure consistency, readability, and reduce onboarding time for new contributors to the automation framework.

### **Chapter 8: Testing**

**8.1 Testing Plan** Automation testing is carried out based on test strategy, scope, execution timeline, and deliverables. The framework is tailored to validate core business logic, system behavior, and functional flow across multiple scenarios.

**Test Components:**

* **In-Scope Modules:** Login, Task Management, User Management, Profile Update
* **Out-of-Scope:** External API integrations, third-party services
* **Testing Deliverables:**
  + Automated test scripts
  + Execution logs
  + ExtentReports with screenshots
  + Bug reports and failure trace logs

**Test Schedule & Execution:**

* Scripting phase aligned with sprint tasks
* Daily regression suite execution
* Manual fallback for unautomated corner cases

**Figure 8.1: Automation Testing Flowchart** [Insert Flowchart Placeholder here]

**8.2 Testing Strategy** The strategy focuses on identifying test candidates that are repetitive, data-driven, and sensitive to user roles or workflows. Manual testing complements automation where exploratory coverage is needed.

**Testing Types Covered:**

* **Regression Testing:** Run on each release cycle
* **Sanity Testing:** Quick checks for new builds
* **Data-Driven Testing:** Different data combinations using external inputs
* **Cross-Browser Testing:** Chrome, Edge, Firefox
* **Negative Testing:** Verifying system response to invalid inputs and conditions

**Strategic Considerations:**

* Test scenarios prone to human error were automated
* Functionalities used across multiple roles were prioritized
* Scenarios hard to replicate manually (e.g., session expiry, load) were included
* Frequently reused flows were modularized

### **Chapter 9: Conclusion**

**9.1 Problems Encountered and Possible Solutions**

**Redundancy in Script Logic:** Initial versions of scripts included duplicate logic for interacting with fields and dropdowns across different modules. This redundancy was resolved by creating reusable methods for these actions (e.g., clearFieldWithActions, searchAndSelectRecord).

**Execution Challenges:** There was no utility in place to run multiple scripts as a batch or track status. We proposed a custom script execution utility with a centralized dashboard, enhancing test visibility and control for QA teams.

**Performance Bottlenecks in UI Testing:** Frequent UI changes during development caused tests to break. To mitigate this, dynamic locator strategies and property-driven configurations were introduced.

**Database Query Complexity:** Long or nested queries slowed down test validation. Developers were advised to restructure queries with optimized joins and indexing.

**9.2 Summary of Project Work** This project demonstrated the power and flexibility of a Selenium-based automation framework tailored for an enterprise-grade web application. We successfully:

* Developed a modular, extensible framework
* Automated end-to-end scenarios for core MEM functionalities
* Integrated ExtentReports for enhanced visibility
* Created reusable methods for improved maintainability

Automation ensured:

* Faster test execution
* Lower manual testing effort
* Increased accuracy
* Faster feedback cycles during development

The framework lays a solid foundation for continuous improvement and scale-up to include more test cases and features.

### **Chapter 10: Limitations and Future Enhancement**

**10.1 Limitations** Despite its advantages, the developed automation framework has a few constraints:

* **UI Dependency:** Any minor change in the application UI requires frequent updates to locators and scripts.
* **Initial Setup Time:** Framework setup and element mapping require upfront investment before test creation.
* **Automation Scope:** Not all features (like CAPTCHA, OTPs, or complex third-party integrations) are feasible for full automation.
* **Script Intelligence:** The automation scripts operate based on predefined logic and cannot replicate nuanced decision-making like human testers.

**10.2 Future Enhancements** To improve the framework’s robustness and usability, the following enhancements are proposed:

* **Script Generation Utility:** Develop a utility where QA can input flow data and generate automation scripts automatically.
* **Jenkins Integration:** Enable Continuous Integration/Continuous Testing by connecting the framework with Jenkins or other CI/CD tools.
* **Visual Regression Tools:** Add tools like Applitools for verifying UI alignment, element shifts, or visual bugs.
* **Cross-Platform Execution:** Integrate cloud test platforms like BrowserStack or Selenium Grid for large-scale execution.
* **Dashboard Analytics:** Build a live dashboard to track script execution, failures, trends, and overall system stability metrics.

These enhancements will allow the automation suite to scale, adapt, and evolve alongside the growing demands of the MEM platform.

### **Chapter 11: Bibliography and References**

The following references were used throughout the development and documentation of the MEM Automation Framework project:

1. **Guru99 Selenium WebDriver Tutorial**<https://www.guru99.com/selenium-tutorial.html>
2. **Guru99 TestNG Framework Guide**<https://www.guru99.com/all-about-testng-and-selenium.html>
3. **Maven Repository**<https://mvnrepository.com/>
4. **TutorialsPoint Performance Testing Guide**<https://www.tutorialspoint.com/software_testing_dictionary/performance_testing.htm>
5. **W3Schools Software Testing Resource**<https://www.w3schools.in/software-testing/stress-and-performance-testing>
6. **Project Figures and Diagrams** Internal documentation and test planning materials provided during internship at Meditab Software Pvt. Ltd.
7. **Official Java Documentation**<https://docs.oracle.com/javase/8/docs/>
8. **ExtentReports Official Documentation**<https://extentreports.com/docs/versions/4/java/>