Project Test Report

for

"আহার"

A web-based food delivery application

Prepared by

Fardin Alam Alif MUH2025001M

Md. Asif Mahmud MUH2025004M

Prity Rani Dash BFH2025021F

Hasanur Rahman Shishir MUH2025022M

Khos Mahmuda Akter BKH2025036F

Institute of Information Technology

Noakhali Science & Technology University

Supervised By,

Iftekharul Alam Efat

Assistant Professor

Institute of Information Technology

Noakhali Science and Technology University **29.2.24**

---

1. INTRODUCTION

1.1 PURPOSE

"আহার" is a web-based food delivery application designed to address the challenges faced by individuals living away from home for work or education. To provide hygienic, high-quality, high-quality, affordable, homemade food, this project endeavors to offer a convenient solution to the daily food needs of users.

The purpose of the "আহার" project is to develop and implement a user-friendly web-based platform that facilitates the ordering and delivery of homemade food. This platform aims to provide a convenient solution for individuals who depend on others for their daily meals, offering a wide range of food options delivered directly to their doorstep.

1.2 BACKGROUND

In today's fast-paced world, many individuals, including university students and working professionals, live away from home and face challenges in accessing hygienic and affordable homemade food. The "আহার" project is initiated to address this need by providing a reliable platform for ordering and delivering homemade meals.

1.3 SCOPE

The scope of the "আহার" project encompasses the development, implementation, and launch of a comprehensive web-based platform that facilitates user registration, menu browsing, order placement, secure payment integration, order tracking, and feedback submission. Additionally, the project includes features for vendor registration, menu management, order handling, and delivery coordination.

1.4 PROJECT IDENTIFICATION

Document	**Created or Available**	**Received or Reviewed**
Requirements Specification	Yes	Yes
Functional Specification	Yes	Yes
Use-Case Reports	Yes	Yes
User’s Manuals	No	Yes

2. PROJECT OVERVIEW

Project Name: "আহার"

2.1 Project Description

"আহার" is a web-based food delivery application catering to individuals living away from home, offering homemade food of high quality at a low cost. It aims to provide a platform where users can select from various food items for their meals and have them delivered to their doorstep. The application facilitates user registration, menu browsing, order placement, secure payment integration, order tracking, and feedback submission. Additionally, it provides a platform for vendors to register their businesses, manage menus, and handle orders efficiently. Delivery drivers act as intermediaries between customers and vendors, ensuring timely delivery of orders. Admins oversee all activities within the application, managing customers, vendors, delivery drivers, menus, payments, and generating sales reports.

This overview outlines the key objectives and features of the "আহার" project, emphasizing its role in bridging the gap between food providers and consumers through a user-friendly and efficient web-based platform.

3. TEST STRATEGY

The testing strategy for the "আহার" project encompasses a systematic approach to verify and validate the application's functionality, performance, security, and usability.

3.1 Test Requirements

Identifying test requirements and objectives is a crucial step in ensuring the effectiveness and completeness of the testing process for the "আহার" project. This involves understanding the functional and non-functional aspects of the application, as well as the expectations and goals of stakeholders. Below are the test requirements and objectives identified for the project:

3.1.1 Functional Requirements:

- Verify that all functional requirements outlined in the Software Requirements Specification (SRS) are implemented correctly.
- Ensure that the application's features and functionalities meet the needs of users, vendors, delivery drivers, and administrators.

3.1.2 Non- functional Requirements:

- Validate the performance, security, usability, and scalability aspects of the application.
- Ensure compliance with regulatory requirements and industry standards related to food delivery applications.

3.2 Test Objectives:

3.2.1 Functional Testing:

- Validate each functional requirement to ensure that the application behaves as expected.
- Verify that users can perform key actions such as registration, ordering, tracking, and providing feedback without encountering errors.
- Confirm that all user roles (customers, vendors, delivery drivers, administrators) have access to the appropriate features and permissions.

3.2.2 Usability Testing:

- Evaluate the user interface (UI) and user experience (UX) of the application to ensure ease of use and navigation.
- Verify that the application is intuitive and user-friendly, with clear instructions and prompts for users.

3.2.3 Performance Testing:

- Assess the responsiveness and speed of the application under normal and peak load conditions.
- Determine the application's capacity to handle concurrent user interactions, orders, and transactions without slowdowns or crashes.

3.2.4 Security Testing:

- Identify and address potential vulnerabilities such as data breaches, injection attacks, and unauthorized access.
- Ensure that sensitive user data (e.g., personal information, payment details) is encrypted and protected from unauthorized access.

3.2.5 Compatibility Testing:

- Verify that the application functions correctly across different web browsers (Chrome, Firefox, Safari, Edge) and operating systems (Windows, macOS, Linux).

3.2.6 Scalability Testing:

- Evaluate the application's ability to accommodate an increasing number of users, vendors, and orders over time without degradation in performance.

3.2.7 Regression Testing:

- Ensure that new updates or modifications to the application do not introduce defects or regressions in existing functionalities.

3.2.8 User Acceptance Testing (UAT):

- Involve stakeholders (users, vendors, administrators) in testing to validate that the application meets their expectations and requirements.

3.3 Test Cases

To design test cases based on functional requirements and user scenarios for the "আহার" project, we'll consider various functionalities outlined in the Software Requirements Specification (SRS) and user scenarios described in the project documentation. Below are examples of test cases for some key functionalities:

❖ **User Registration (FR-1)**

3.3.1 Successful User Registration

- **Preconditions:** User navigates to the registration page.
- **Test Steps:**
  1. Enter valid user details (name, email, contact number, password, delivery address).
  2. Click on the "Register" button.
- **Expected Result:** User should be successfully registered and redirected to the login page.

3.3.2 Invalid User Registration

- **Preconditions:** User navigates to the registration page.
- **Test Steps:**
  1. Enter invalid or incomplete user details.
  2. Click on the "Register" button.
- **Expected Result:** User should receive appropriate error messages for invalid input fields.

❖ **Manage User Profile (FR-2)**

3.3.3 Update User Profile

- **Preconditions:** User is logged in and navigates to the profile settings page.
- **Test Steps:**
  1. Modify existing profile information (e.g., change email/contact number, delivery address).
  2. Click on the "Save Changes" button.
- **Expected Result:** User profile should be updated successfully with the new information.

❖ **Search Food Items (FR-4)**

3.3.4 Search for Available Food Item

- **Preconditions:** User is logged in and navigates to the search bar.
- **Test Steps:**
  1. Enter a valid food item name in the search bar.
  2. Press Enter or click on the search icon.
- **Expected Result:** System should display a list of vendors offering the searched food item.

❖ **Placing Customer Order (FR-5)**

3.3.5 Place Customer Order

- **Preconditions:** User is logged in and navigates to the menu.
- **Test Steps:**
  1. Select desired food items from the menu.
  2. Proceed to checkout and provide delivery address.
  3. Choose the payment method and confirm the order.
- **Expected Result:** User should receive an order confirmation with details and estimated delivery time.

❖ **Tracking the Order (FR-6)**

3.3.6 Track Order Status

- **Preconditions:** User is logged in and has an active order.
- **Test Steps:**
  1. Navigate to the order tracking page.
  2. Enter order ID or view active orders.
- **Expected Result:** System should display the current status of the user's order (e.g., confirmed, out for delivery).

3.4 Security Testing

3.4.1 Risks/Issues

1. Data Breach: Unauthorized access leading to a data breach.

2. Encryption Vulnerability: Potential weaknesses in encryption/decryption processes.

3. Authentication Flaws: Security vulnerabilities in user authentication.

4. Insufficient Authorization: Risks of users gaining unauthorized access.

3.4.2 Mitigation

3.4.2.1 For Data Breach

- Regularly perform comprehensive security risk assessments, including penetration testing and vulnerability assessments.
- Implement monitoring and logging mechanisms to detect and respond to suspicious activities promptly.

3.4.2.2 For Encryption Vulnerability

- Utilize strong encryption algorithms for sensitive data during transmission (e.g., TLS/SSL for communication channels) and storage (e.g., AES for data at rest).
- Regularly update encryption protocols and algorithms to mitigate emerging vulnerabilities.

3.4.2.3 For Authentication Flaws

- Implement multi-factor authentication (MFA) for user logins, requiring users to provide multiple forms of identification.
- Regularly educate users on the importance of strong passwords and secure authentication practices.

3.4.2.4 For Insufficient Authorization

- Establish and enforce role-based access controls (RBAC) to ensure that users have access only to functionalities and data relevant to their roles.
- Conduct regular access reviews to verify that user permissions align with their roles.

3.4.3 Items to be Tested

Item to Test	**Test Description**
Secure Data Transmission	Validate the encryption/decryption processes during the transfer of user data, ensuring that sensitive information is securely transmitted over the network.
User Authentication	Assess the robustness of user authentication processes by testing various scenarios, such as valid and invalid credentials, multi-factor authentication, and session management.

Payment Security	Test the security of payment transactions, ensuring the encryption of payment data, and validating the integration with secure payment gateways.
Input Validation	Validate input fields to ensure they properly handle and sanitize user inputs, preventing common vulnerabilities like SQL injection and Cross-Site Scripting (XSS).
API Security	Assess the security of APIs, including authentication and authorization mechanisms, to ensure that they are protected against unauthorized access and abuse.

3.4.4 Items Not to be Tested: Test Approach

Items not to be tested	Explanation
Low-level Unit Functionalities (Covered in Unit Testing)	Low-level unit functionalities, tested during unit testing, are not explicitly tested in security testing. Unit testing ensures the correctness of individual units of code.
Non-Security Features (BMI Calculator)	Functionalities that are not (BMI Calculator) directly related to security, such as non-sensitive user interface elements, may not be explicitly tested in the security testing phase.

3.4.5 Penetration Testing

Simulate cyber-attacks to identify vulnerabilities in the system. This includes testing for potential exploits and weaknesses in the application's security defenses.

- **Security Scanning:** Use tools like OWASP ZAP to conduct automated scans for common security issues, vulnerabilities, and misconfigurations in the application.
- **Code Analysis:** Utilize static code analysis tools to identify and analyze security flaws and vulnerabilities within the application's source code.

3.4.5.1 Test Pass/ Fail Criteria

➔ **Pass Criteria**
- No critical security vulnerabilities are identified during testing.
- All identified security issues are resolved satisfactorily.

➔ **Fail Criteria**

- Identification of severe security flaws that pose a significant risk to the confidentiality, integrity, or availability of the system.
- Unresolved security issues that could lead to unauthorized access, data breaches, or other security breaches.

3.4.5.2 Test Entry/ Exit Criteria

➔ **Entry Criteria:**

1. **Completion of Previous Testing Phases:**
    - Unit testing and integration testing have been successfully completed.
    - Identified issues from previous testing phases have been resolved.
2. **Availability of Integrated System for Security Testing:**
    - The application is in a stable state, and all components are integrated and ready for security testing.

➔ **Exit Criteria:**

1. **Successful Completion of Security Tests:**
    - All security tests pass without critical errors.
    - Identified security issues are resolved.
2. **Security Test Summary Report:**
    - Generate a comprehensive Security Test Summary Report that includes details on the testing approach, executed tests, identified vulnerabilities, and their resolutions.
3. **Approval for Production Release:**
    - Approval from the security testing team and relevant stakeholders for the production release of the application.

3.4.6 Test Deliverables

- **Security Test Cases and Scripts:** Detailed test cases and scripts covering all aspects of security testing, including secure data transfer, authentication, authorization, payment security, input validation, and more.
- **Security Test Summary Report:** A comprehensive report summarizing the security testing activities, including the testing approach, executed tests, identified vulnerabilities, and their resolutions. This report serves as a reference for stakeholders and future assessments.

3.4.7 Test Plan

- **Secure Data Transmission:** Validate encryption/decryption processes during the transfer of user data.
- **User Authentication:** Assess the robustness of user authentication processes.
- Payment Security: Test the security of payment transactions and payment gateway integration.
- **Input Validation:** Validate input fields to prevent common vulnerabilities like SQL injection and XSS.
- **API Security:** Assess the security of APIs, including authentication and authorization mechanisms

.

3.4.8 Procedures for Testing Sign-Off and Product Release

1. **Generate Security Test Summary Report:**
    - Create a detailed report summarizing the security testing activities and outcomes.
2. **Approval for Production Release:**
    - Obtain approval from key stakeholders, including the security testing team, for the production release of the application.

3.4.9 Testing Tools

1. **OWASP ZAP (Zed Attack Proxy):**
    - Used for automated security scanning to identify common vulnerabilities.
2. **Burp Suite:**
    - A tool for penetration testing, examining security of web applications, and identifying potential vulnerabilities.
3. **Static Code Analysis Tools:**
    - Utilized to analyze the source code for security flaws. Specific tools may include SonarQube, Checkmarx, or similar.

3.5 Load Testing

3.5.1 Risks/ Mitigation

3.5.1.1 Increased Server Load During Peak Usage

- **Mitigation:**
  - Conduct thorough capacity planning to estimate server requirements during peak usage.
  - Implement auto-scaling mechanisms to dynamically adjust resources based on demand.

3.5.1.2 Performance Bottlenecks

- **Mitigation:**
  - Identify potential bottlenecks through performance profiling and conduct targeted optimizations.
  - Utilize load balancing to distribute traffic evenly across multiple servers.

3.5.1.3 Database Overload

- **Mitigation:**
  - Optimize database queries and indexing to handle increased loads efficiently.
  - Implement database caching mechanisms to reduce the load on the database server.

3.5.1.4 Inadequate Concurrent User Testing

- **Mitigation:**
  - Perform realistic load testing with a focus on concurrent user scenarios.
  - Consider user behavior patterns and simulate realistic usage patterns.

3.5.1.5 Third-Party Integration Issues

- **Mitigation:**
  - Coordinate with third-party service providers to ensure their systems can handle increased load.
  - Implement fallback mechanisms in case of third-party service failures.

3.5.2 Items to be tested

Items to test	**Test Description**
User Load Capacity	Evaluate the system's ability to handle a specified number of concurrent users during peak usage.

Response Time Under	Measure the response time of critical functionalities as the user load increases to ensure acceptable performance.
Throughput and Transactions per Second	Assess the system's throughput and the number of transactions it can handle per second to understand its processing capabilities.
Database Performance	Evaluate database performance by measuring response times for queries and assessing the impact of concurrent users on database operations.
Scalability Testing	Assess the system's scalability by increasing the load gradually and determining how well it scales with additional resources.

3.5.3 Items not be Tested

Items not to Tested	Explanation
Individual Component Performance (Covered in Unit Testing)	Performance of individual components is typically tested during unit testing. Load testing focuses on the overall system performance under varying loads.
Non-Functional Features Unrelated to Load	Features such as UI responsiveness and design aesthetics are not the primary focus of load testing. These aspects are covered in other testing phases.
Functional Testing Scenarios	Functional scenarios, covered in functional testing, are not explicitly tested in load testing. Load testing assesses how well the system performs under various loads but doesn't validate individual functionalities in detail.

3.5.4 Test Approach

1. **Define Test Objectives:** Clearly articulate the load testing objectives, including assessing system capacity, scalability, and identifying potential performance bottlenecks.
2. **Identify Key Scenarios:** Identify critical user scenarios that represent typical usage patterns, ensuring coverage of high-impact functionalities such as user logins, order placements, and payment processing.
3. **Create Realistic User Profile**: Develop realistic user profiles that mirror the expected user distribution and behavior during peak usage, considering factors like geographical location

and user roles.

4. **Set Performance Criteria:** Establish clear performance criteria such as acceptable response times, throughput, and resource utilization thresholds to serve as benchmarks for success.
5. **Design Test Cases:** Develop load test cases that simulate various levels of user load, gradually increasing the load to observe the system's behavior under stress.
6. **Simulate Peak Usage:** Simulate scenarios that represent peak usage, ensuring the system's ability to handle maximum expected load without significant degradation in performance.
7. **Conduct Stress Testing:** Apply stress to the system beyond its anticipated capacity to identify breaking points and understand the system's behavior under extreme conditions.
8. **Monitor Resource Utilizations:** Implement monitoring tools to continuously monitor CPU, memory, disk, and network utilization during load testing to identify resource bottlenecks.
9. **Evaluate Scalability:** Assess the system's scalability by gradually increasing the load and observing how well it scales with additional resources.
10. **Collaborate with Development Team:** Work closely with the development team to address identified performance issues, implement optimizations, and iterate on load testing scenarios.
11. **Execute Test Scenarios:** Execute load test scenarios using load testing tools, simulating realistic user behavior and load conditions.
12. **Collect and Analyze Data**: Collect performance data, including response times, throughput, and resource utilization, and analyze the results against predefined performance criteria.
13. **Iterative Testing:** Conduct iterative testing, making adjustments to test scenarios and configurations based on the analysis of previous test results.
14. **Generate Performance Reports:** Generate comprehensive performance reports, including insights into system behavior, identified bottlenecks, and recommendations for improvements.

3.5.5 Test Pass/ Fail Criteria

➔ **Pass Criteria:**
- Response times for critical functionalities meet predefined performance criteria under the expected load.
- Throughput remains within acceptable limits.
- Resource utilization (CPU, memory, etc.) is within defined thresholds.
- No critical system failures or degradation in performance is observed under stress conditions.
- The system scales effectively with increased load, demonstrating scalability.

➔ **Fail Criteria:**
- Response times exceed acceptable thresholds, leading to degraded user experience.
- Throughput falls below acceptable levels.
- Resource utilization reaches critical levels, causing system instability.
- Critical system failures occur under stress conditions.
- The system exhibits poor scalability, struggling to handle increased load effectively.

3.5.6 Test Entry/ Exit Criteria

➔ **Entry Criteria:**

- Completion of functional and integration testing phases.
- Availability of a stable and fully integrated application for load testing.
- Defined load testing objectives and scenarios.
- Established performance criteria and benchmarks.

➔ **Exit Criteria:**

- Successful completion of load testing scenarios.
- Performance metrics meeting predefined criteria.
- Implementation of necessary optimizations based on load testing results.
- Approval from stakeholders, including the development and testing teams, for the application's performance under load.
- Generation and review of comprehensive load testing reports.

3.5.7 Test Plan

3.5.7.1 Test Objectives

- Assess system capacity, scalability, and performance under various user loads.
- Identify potential performance bottlenecks and areas for optimization.
- Verify that the application meets predefined performance criteria and benchmarks.

3.5.7.2 Test Scenarios

- User login under normal load.
- Order placement and processing under normal and peak loads.
- Simulated peak usage scenarios to evaluate system behavior.
- Stress testing to determine breaking points and system behavior under extreme conditions.

3.5.7.3 Test Environment

- Use a dedicated testing environment that mirrors the production environment.
- Ensure the availability of monitoring tools to capture real-time metrics during load testing.

3.5.7.4 Testing Tools

1. **Apache JMeter:**
    - For designing and executing load tests, simulating multiple users and analyzing performance metrics.

2. **LoadRunner:**
    - For performance testing, including load, stress, and scalability testing.

3.5.7.5 Test Scenarios Execution

- Gradually increase user load to assess system behavior under different conditions.
- Monitor resource utilization (CPU, memory, disk, network) during load testing.
- Analyze performance metrics, including response times, throughput, and errors.
- Conduct stress testing beyond expected capacity to identify breaking points.

3.5.7.8 Performance Metrics

- Response time for critical functionalities.
- Throughput and transactions per second.
- CPU, memory, disk, and network utilization.

3.5.7.9 Test Criteria

- Response times meet predefined benchmarks.
- Throughput remains within acceptable limits.
- Resource utilization stays within defined thresholds.
- No critical system failures or degradation in performance under stress conditions.
- The system scales effectively with increased load, demonstrating scalability.

3.5.8 Procedures for Testing Sign-Off and Product Release

3.5.8.1 Sign-Off Criteria

- Successful completion of load testing scenarios.
- Performance metrics meeting predefined criteria.
- Implementation of necessary optimizations based on load testing results.
- Approval from stakeholders, including the development and testing teams, for the application's performance under load.
- Generation and review of comprehensive load testing reports.

3.5.8.2 Release Procedures

- Generate a final load testing report summarizing test objectives, scenarios, results, and recommendations.
- Conduct a review meeting with stakeholders to discuss the load testing outcomes.
- Obtain approval for production release based on load testing results and stakeholder feedback.

3.5.9 Tools Used

1. **Apache JMeter:** Download Apache JMeter
2. **LoadRunner:** LoadRunner - Micro Focus

3.5.10 Test Deliverables

- **Test Plan:** Comprehensive document outlining the overall testing strategy, objectives, scope, resources, schedule, and testing approach.
- **Test Cases:** Detailed test cases specifying the steps to be executed, expected results, and criteria for pass/fail for each test scenario.
- **Final Test Report:** A comprehensive report summarizing the testing activities, including test results, identified issues, and recommendations.

4. Environments Needs

1. **Hardware:**

- **Server:** Minimum dual-core processor, 8 GB RAM, and 100 GB storage for hosting the application and database.
- **Client Devices:** Desktops or laptops with a minimum of 4 GB RAM, capable of running modern web browsers.

2. **Software:**

- **Operating System:** Windows 10/11 Home on client devices.
- **Databases:** MySQL as the database management system for storing and retrieving data.
- **Web Server:** Localhost web server to host and serve the PHP-based application.
- **Programming Languages:** PHP as the primary programming language for developing the application.
- **Encryption/ Decryption:** OpenSSL for secure encryption and decryption processes.

4.1 Boundary value testing

4.1.1 Test risks / issues

- Incorrect handling of boundary values may lead to system errors.
- Inadequate validation of boundary conditions may result in unexpected behavior.
- Variability in interpretation of boundary values across different modules.

4.1.2 Mitigation Strategies

- Develop a comprehensive set of boundary value test cases.
- Conduct peer reviews to ensure consistent understanding of boundary values.
- Use automated testing tools for systematic boundary value testing.

4.1.3 Items to Be Tested

ID	Functional Requirement	Description
FR-1	User Registration	Users can sign up by providing basic information.
FR-3	Change Password	Users can change their account password.
FR-4	Search Food Items	Users can search for food based on preferences.
FR-5	Place Customer Order	Users can customize and place orders.
FR-7	Make Payment	Users can securely pay through various methods.
FR-10	Manage Delivery Address	Users can add, edit, or remove delivery addresses.
FR-11	Manage Menus	Admins can update and manage food item lists.
FR-12	Pick Up Order	Users can opt to pick up orders from the restaurant.
FR-13	Manage Discounts	Admins can create and update discounts.
FR-14	Monthly Report Generation	System generates business reports monthly.
FR-15	Calculate BMI	System calculates and displays BMI for customers.

4.1.4 Items Not to Be Tested

- Boundary values related to features outside the scope of the specified functional requirements.

4.1.5 Test Approach

- Identify critical boundary values for each relevant input field.
- Develop test cases to evaluate system behavior at and around boundary values.
- Implement automated testing where applicable for efficient boundary value validation.

4.1.6 Test Pass / Fail Criteria

- **Pass Criteria:** System behaves as expected at and around specified boundary values.
- **Fail Criteria:** System exhibits errors or unexpected behavior during boundary value tests.

4.1.7 Test Entry /Exit Criteria

➔ **Entry Criteria**
- Availability of the latest system build with implemented boundary value checks.
- Completion of unit testing to ensure basic functionality.

➔ **Exit Criteria:**
- Successful execution of all boundary value test cases.
- Resolution of any identified issues related to boundary conditions.

4.1.8 Test Deliverables

- Boundary Value Test Cases and Scripts.
- Boundary Value Test Summary Report.

4.1.9 Test Plan (Boundary Value Testing)

4.1.9.1 Test Approach and Deliverables

➔ **Approach:**
- Identify critical boundary values for each relevant input field.
- Develop test cases to evaluate system behavior at and around boundary values.
- Implement automated testing where applicable for efficient boundary value validation.

➔ **Deliverables:**

- Boundary Value Test Cases and Scripts.
- Boundary Value Test Summary Report.

4.1.9.2 Test Areas and Features

1. **User Registration (FR-1):**
   - Allows users to create an account by providing essential details.
2. **Change Password (FR-3):**
   - Enables users to update their account password for security.
3. **Search Food Items (FR-4):**
   - Provides a search feature for users to find food based on preferences.
4. **Place Customer Order (FR-5):**
   - Allows users to customize and place food orders.
5. **Make Payment (FR-7):**
   - Enables users to complete transactions using various payment methods.
6. **Contact Customer Support (FR-9):**
   - Provides users with the means to reach out for assistance or inquiries.
7. **Manage Delivery Address (FR-10):**
   - Allows users to add, edit, or remove delivery addresses.
8. **Manage Menus (FR-11):**
   - Permits administrators to update and manage the list of available food items.
9. **Pick-Up Order (FR-12):**
   - Allows users to choose the option of picking up their orders from the restaurant.
10. **Manage Discount (FR-13):**
    - Gives administrators the ability to create and manage discounts for customers.
11. **Monthly Report Generation (FR-14):**
    - Automatically generates business reports on a monthly basis.
12. **Calculate BMI (FR-15):**
    - Calculates and displays the Body Mass Index (BMI) for customers based on profile information.

4.1.9.3 Tools Used

- Use testing tools such as **JUnit** for automated boundary value testing.
- Use testing tools such as **PhPUnit** for automated boundary value testing.

4.2. User acceptance testing

4.2.1 Test risks/issues

- **Misalignment with User Expectations:** The risk of the system not meeting user expectations.
- **Incomplete User Training:** Potential issues if users are not adequately trained on the system.
- **Limited User Involvement:** Insufficient user involvement leading to overlooked requirements.

4.2.2 Mitigation Strategies

- Conduct user workshops to gather and clarify requirements.
- Implement user training sessions for system understanding.
- Establish a communication channel for continuous user feedback.

4.2.3 Items to Be Tested

ID	Functional Requirement	Description
FR-1	User Registration	Users can sign up by providing basic information.
FR-3	Change Password	Users can change their account password.
FR-4	Search Food Items	Users can search for food based on preferences.
FR-5	Place Customer Order	Users can customize and place orders.
FR-7	Make Payment	Users can securely pay through various methods.
FR-10	Manage Delivery Address	Users can add, edit, or remove delivery addresses.

FR-11	Manage Menus	Admins can update and manage food item lists.
FR-12	Pick Up Order	Users can opt to pick up orders from the restaurant.
FR-13	Manage Discounts	Admins can create and update discounts.
FR-14	Monthly Report Generation	System generates business reports monthly.
FR-15	Calculate BMI	System calculates and displays BMI for customers.

4.2.4 Items Not to Be Tested

- Performance-related aspects (Outside scope for UAT).

4.2.5 Test Approach

- Scenario-based Testing: Develop and execute test scenarios aligned with user workflows.
- User Workshops: Conduct workshops to ensure user understanding of functionalities.
- Continuous Feedback: Establish a feedback loop for users to provide ongoing input.

4.2.6 Test pass/fail criteria

- **Pass Criteria:** Users can perform all tested functionalities without critical issues.
- **Fail Criteria:** Significant user-related issues that impact system usability.

4.2.7 Test entry / exit criteria

➔ **Entry Criteria:**
- Completion of system development and unit testing.
- Availability of the UAT environment.

➔ **Exit Criteria:**
- Users successfully perform all UAT test cases.
- Identified issues are resolved or have an agreed-upon resolution plan.

4.2.8 Test Deliverables

- UAT Test Cases and Scripts.
- UAT Test Summary Report.

4.2.9 Test Plan (UAT)

4.2.9.1 Test Areas and Features

1. **User Registration (FR-1):**
   - Allows users to create an account by providing necessary information.
   - Facilitates the initial step for users to access the system and personalized services.
2. **Manage User Profile (FR-2):**
   - Enables users to view and edit their profile information.
   - Provides a platform for users to maintain and customize their account details.
3. **Changing the Current Password (FR-3):**
   - Allows users to update their account password for security.
   - Enhances account protection and user control over access credentials.
4. **Search Food Items According to User Preferences (FR-4):**
   - Provides a search function for users to find food items based on preferences.
   - Enhances user experience by facilitating quick and personalized food discovery.
5. **Placing the Customer Order (FR-5):**
   - Allows users to select and customize their food orders.
   - Streamlines the process for users to place orders according to their preferences.
6. **Tracking the Order (FR-6):**
   - Enables users to monitor the real-time status and location of their orders.
   - Enhances transparency and provides users with information on their order's progress.
7. **Make Payment Through Payment Method (FR-7):**
   - Allows users to securely complete transactions using various payment methods.
   - Ensures a convenient and secure payment process for customers.
8. **Customer Feedback Submission Based on Previous Order History (FR-8):**
   - Allows users to submit feedback and reviews based on their previous orders.
   - Encourages continuous improvement based on customer experiences.
9. **Contact Customer Support (FR-9):**
   - Provides users with a direct means to contact customer support for assistance.
   - Ensures a responsive and accessible support channel for user queries.
10. **Managing the Customer's Delivery Address (FR-10):**
    - Allows users to add, edit, or remove delivery addresses.
    - Offers flexibility and convenience in managing delivery preferences.
11. **Managing the Menus (FR-11):**

- Enables administrators to update and manage the list of available food items.
- Ensures the menu is up-to-date and reflective of the restaurant's offerings.

12. **Pick Up Order (FR-12):**
- Allows users to choose the option of picking up their orders from the restaurant.
- Provides an alternative and convenient order fulfillment method.

13. **Manage Discount (FR-13):**
- Enables administrators to create and manage discounts for customers.
- Supports promotional activities and enhances customer engagement**.**

14. **Monthly Report Generation (FR-14):**
- Automatically generates business reports on a monthly basis.
- Facilitates data-driven decision-making and business analysis.

15. **Calculating BMI for Customer (FR-15):**
- Provides a feature to calculate and display the Body Mass Index (BMI) for customers.
- Enhances user engagement by offering health-related information.

16. **Approval of Patients Access Request (Administrators) (FR-9):**
- Allow accountants to review and approve access requests from patients.
- Confirm the system's response to varying lengths of access request details.

17. **Add New Doctor Profiles (Administrators) (FR-10):**
- Enable administrators to add new doctor profiles to the system.
- Verify the system's behavior with varying lengths of administrator actions.

4.2.10 Test deliverables

The following are the test deliverables:

- Test Plan
- Test Case
- Final Test Report

4.9.3 Environmental needs

➔ **Hardware:**
- Server: Minimum dual-core processor, 8 GB RAM, 100 GB storage
- Client Devices: Desktops or laptops with a minimum of 4 GB RAM and modern web browsers

➔ **Software:**
- Operating System: Windows 10/11 Home
- Database: Mysql

- Web Server: Localhost
- Programming Language: PHP
- Encryption/Decryption: OpenSSL

4.3. Path Testing

4.3.1 Test Risks/ Issues

- **Complexity in Path Identification:** Difficulty in identifying and testing all possible paths through the food delivery system.
- **Incomplete Path Coverage:** The risk of missing critical paths during testing.
- **Time-Consuming:** Due to the exhaustive nature of testing all paths, it might be time-consuming.
- **Maintenance:** Changes in the codebase might affect existing paths.

4.3.2 Mitigation Strategies

- Utilize Control Flow Diagrams to visualize code paths.
- Prioritize critical and frequently traversed paths.
- Use automated testing tools to generate and execute paths efficiently.
- Continuous code reviews to ensure code changes are accounted for in path testing.

4.3.3 Items To Be Tested

Test ID	Items to be tested	Description
ID-1	User Registration Path	Verify successful user registration and entry into the system.
ID-2	Change Password Path	Test the path for changing the current password.
ID-3	Search Food Items Path	Verify the path for searching food items according to user preferences.
ID-4	Manage Menus Item	Test the path for managing menus (add item, remove item, view item, edit item).
ID-5	Calculate BMI Path	Confirm the path for calculating BMI for customers.

Items Not to be Tested

- Integration and interactions outside the scope of the application's internal paths.

4.3.4 Test Pass/ Fail Criteria

- **Pass Criteria:** Successful traversal and validation of critical paths without errors.
- **Fail Criteria:** Identified critical issues and deviations in tested paths.

4.3.5 Test entry / exit criteria

➢ **Entry Criteria:**
- Availability of the latest codebase.
- Completion of code development for the respective paths.

➢ **Exit Criteria:**
- Successful traversal and validation of all identified critical paths.
- Resolution of identified path-related issues.

Test deliverables

- Path Testing Cases and Scripts.
- Path Testing Summary Report.

Test environmental

Environment: Access to a comprehensive path testing environment.

Staffing: Testers with expertise in path testing and code analysis.

Training: Training on the use of path testing tools and methodologies.

5. Test Case Design

Test case design involves creating detailed scenarios outlining input, actions, and expected outcomes to systematically validate software functionalities. It ensures thorough coverage of test scenarios, aiding in effective testing execution. The design includes clear steps, input data, and expected results, contributing to a comprehensive testing strategy. This process enhances the quality and reliability of software by identifying and rectifying potential issues.

5.1 Functional testing of user interface and features

Functional testing for the "আহার" project involves validating specific features and functionalities to ensure they meet the intended requirements. Using JUnit and Selenium, functional testing is automated, allowing for efficient and repetitive testing of user registration, password changes, food item searches, menu management, and BMI calculation. JUnit provides a robust framework for organizing and executing tests, while Selenium automates browser interactions, simulating user behavior. Test scripts are designed to mimic user actions and verify expected outcomes, ensuring the application's functional integrity. Here are functional requirement which will be tested.

SL. No	**Functional Requirements**
FR1	User Registration to enter into the system
FR2	Changing the current password
FR3	Search food items according to user preferences
FR4	Managing the Menus (add item, remove item, view item, edit item)
FR5	Calculating BMI

5.1.1 Test Case 1: User Registration to Enter into the System

Test Case 1.1	Successful User Registration
Test Steps	- Open the application registration page. - Enter valid user details (name, email, password).

	- Click on the "Register" button.
Expected Result	User should be successfully registered and redirected to the login page

Test Data:

SL No	Input		Expected Output
	Name	**Email**	
TC1	Jhon	jhon@gmail.com	Registration Successful
TC2	-	jhon@gmail.com	Please fill username
TC3	Jhon	-	Please fill email
TC4	Jhon	jhon@	Invalid email format

5.1.2 Test Case 1.2: Registration with Existing Email

Test Case 1.2	Registration with Existing Email
Test Steps	- Open the application registration page. - Enter an email that is already registered. - Click on the "Register" button.
Expected Result	System should display an error message indicating that the email is already in use.

Test Data:

SL No	Input		Expected Output
	Name	**Email**	

TC5	Jhon	jhon@gmail.com	Existing email already registered in the system.
TC6	Jhon	jhon1@gmail.com	Registration Successful

5.1.3 Test Case 2: Changing the Current Password

Test Case 2	**Changing the Current Password**
Test Steps	- Log in to the application. - Navigate to the profile settings. - Enter the current password and a new, valid password. - Click on the "Change Password" button.
Expected Result	Passwords should be successfully changed, and a confirmation notification should be displayed.

Test Data:

SL No	**Input**			**Expected Output**
	Current Password	New Password	Confirm New Password	
TC7	123456	abc123	abc123	Password should be successfully changed
TC8	-	abc123	abc123	Please fill the current password
TC9	123456	-	abc123	Please fill the new password.
TC10	123456	abc123	-	Please confirm the new password.
TC11	123456	abc123	xyz123	Password confirmation does not match the new password.

5.1.4 Test Case 3.1: Search Food Items According to User Preferences

Test Case 3.1	Successful Food Item Search
Test Steps	- Open the application. - Enter a valid keyword in the search bar. - Click on the "Search" button
Expected Result	Relevant search results matching the keyword should be displayed.

Test Data:

SL No	Input	Expected Output	Remark
	Keyword		
TC12	Pizza	Relevant search results matching the keyword displayed.	Valid keyword for a popular food item.
TC13	Dragon Fruit	No matching items found.	Keyword does not exist in the application.
TC14	Chicken Curry	Relevant search results matching the keyword displayed.	Keywords with special characters and spaces.
TC15	2021 Special	Relevant search results matching the keyword displayed.	Keyword with numeric values.
TC16	-	Invalid keywords	Empty Search Result

5.1.5 Test Case 4.1: Managing the Menus (Add Item, Remove Item, View Item, Edit Item)

Test Case 4.1	Managing the Menus (Add Item, Remove Item, View Item, Edit Item)
Test Steps	- Log in to the application as an admin. - Navigate to the menu management section. - Add a new menu item with valid details. - Click on the "Add" button.
Expected Result	The new menu item should be successfully added to the system

5.1.6 Test Data 4.2: Managing the Menus (Add Item)

SL No	Input	Expected Output	Remark
	Keyword		
TC17	Pizza	The new menu item should be successfully added	Valid keyword for a popular food item.

SL No	Input	Expected Output	Remark
	Keyword		
TC18	"Burger" with Existing Item ID	Error message indicating duplicate item ID	Keyword does not exist in the application.
TC19	“Burger" with Price -5	Error message indicating invalid price	Keywords with special characters and spaces.
TC20	-	Error message indicating required fields not filled	Keyword with numeric values.

5.1.7 Test Data 4.3: Managing the Menus (Remove Item)

SL No	Input	Expected Output	Remark
	Menu Item ID to Remove		
TC21	123	The menu item with ID 123 should be removed.	Valid ID for an existing menu item to remove.
TC22	-	Error message indicating no ID provided.	Empty ID for removal is not allowed.
TC23	1234	Error message indicating item not found.	Trying to remove a non-existing item.

5.1.8 Test Data 4.4: Managing the Menus (Edit Item)

SL No	Input	Expected Output	Remark
	Menu Item ID to Edit		
TC24	123	The menu item with ID 123 should be updated.	Valid ID for an existing menu item to update.
TC25	-	Error message indicating no ID provided.	Empty ID for update is not allowed.
TC26	1234	Error message indicating item not found.	Trying to update a non-existing item.

5.1.9 Test Case 5: Calculating BMI

Test Case 5	Calculating BMI
Test Steps	- Log in to the application. - Navigate to the BMI calculation section. - Enter valid height and weight values. - Click on the "Calculate BMI" button.
Expected Result	The BMI should be calculated accurately and displayed on the screen

Test Data:

SL No	Input		Expected Output	Remark
	Height (c.m)	Weight (kg.)		
TC27	170	70	BMI calculated accurately and displayed on screen	Valid height and weight values for BMI calculation.
TC28	-	70	Error message indicating empty height.	Empty height is not allowed.
TC29	170	-	Error message indicating empty weight.	Empty weight is not allowed.
TC30	-170	70	Error message indicating negative height.	Negative height is not allowed.
TC31	170	-70	Error message indicating negative weight.	Negative weight is not allowed.
TC32	0	170	Error message indicating invalid height.	Zero height is not allowed.
TC33	170	0	Error message indicating invalid weight.	Zero weight is not allowed.
TC34	One Seventy	Seventy	Error message indicating non-numeric input.	Non-numeric input for height and weight.

5.2 Regression testing

Regression testing for the "আহার" project involves systematically verifying that recent code changes haven't adversely affected existing functionalities. Through automated testing tools like JUnit and Selenium, regression testing ensures the stability of previously validated features after each code modification. This process aids in identifying and addressing unintended side effects, maintaining the overall reliability of the web-based food delivery application. Test scripts are designed to re-execute critical test cases, confirming that the recent changes do not introduce new defects or compromise established functionalities.

5.3 Performance testing

Performance testing for the "আহার" project involves assessing the system's responsiveness, scalability, and stability under varying workloads. Utilizing tools such as Apache JMeter, performance testing simulates different user scenarios, measuring response times, throughput, and resource utilization. This testing ensures that the application can handle expected user loads without degradation in performance. Key metrics like response times for user actions, server resource utilization, and transaction throughput are monitored to identify potential bottlenecks or performance issues. The goal is to optimize system performance and guarantee a seamless experience for users during peak usage periods.

5.4 Security testing

Security testing for the "আহার" project focuses on identifying vulnerabilities and ensuring robust protection against potential threats. This involves evaluating the application's resistance to unauthorized access, data breaches, and other security risks. Testing tools like OWASP ZAP may be employed to simulate common security attacks and assess the application's resilience. Security testing covers aspects such as authentication mechanisms, data encryption, secure communication protocols, and protection against common web vulnerabilities like SQL injection and cross-site scripting (XSS). The goal is to fortify the application against security threats and safeguard sensitive user information, ensuring a secure and trustworthy user experience.

6. Testing Tools Used

6.1. Functional Testing Tools

In the "আহার" project, we rely on Selenium for testing the web interface across various browsers. JUnit is our go-to tool for validating Java-specific functionalities, especially those related to backend processes. To create and execute detailed test suites, we leverage TestNG, which enables parameterization and parallel testing. These functional testing tools collectively ensure the accuracy and reliability of the food delivery application's features, enhancing its overall performance.

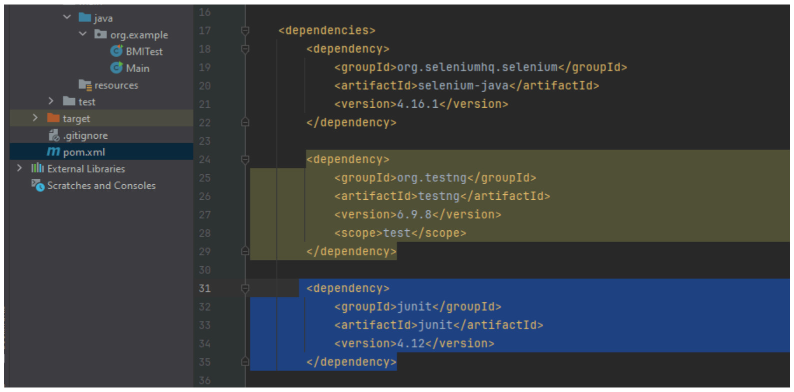
6.1.1 Selenium

In the "আহার" project,Selenium can be employed to automate the testing of the web-based food delivery application's user interface and features. Test scripts can simulate user interactions, such as registration, menu browsing, and order placement, across different browsers like Chrome, Firefox, and Safari.Selenium is primarily used for functional and acceptance testing at the user interface level rather than traditional unit testing. Unit testing typically involves testing individual units of code in isolation, which is common. However, if you are interested in incorporating Selenium-like browser automation into your unit testing process, you might consider a combination of Selenium with a unit testing framework. Here's a simplified description of how you might approach it.

6.1.1.1 Setup Unit Testing Framework

- In the "আহার" project, we select JUnit as unit testing.

- Then, Set up our project with JUnit and necessary dependencies.



```
    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.16.1</version>
        </dependency>

        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.9.8</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>

```

Figure 1 - Adding JUnit maven dependencies.

6.1.1.2 Isolate Units for Testing

Identify specific units or components of your code that wish to test. They are-

SL. No	Functional Requirements
FR1	User Registration to enter into the system
FR2	Changing the current password
FR3	Search food items according to user preferences
FR4	Managing the Menus (add item, remove item, view item, edit item)
FR5	Calculating BMI

6.1.1.3 Use Selenium for Component Testing

Here we Integrate Selenium WebDriver into our unit tests to interact with the user interface components or simulate user actions. As my "আহার" project is a PHP based project. So, we use selenium to perform JUnit tests.

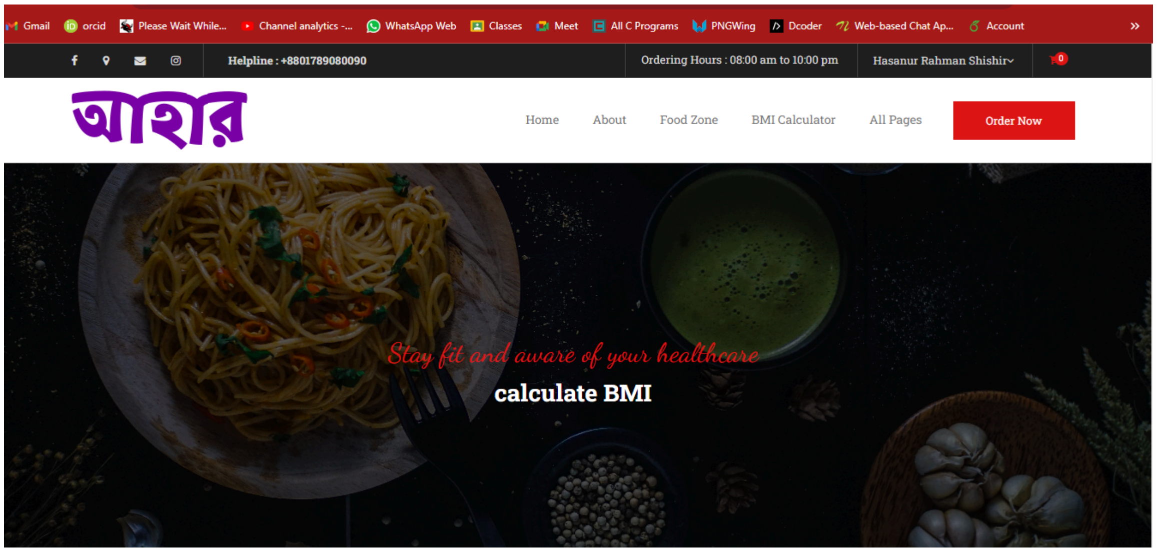
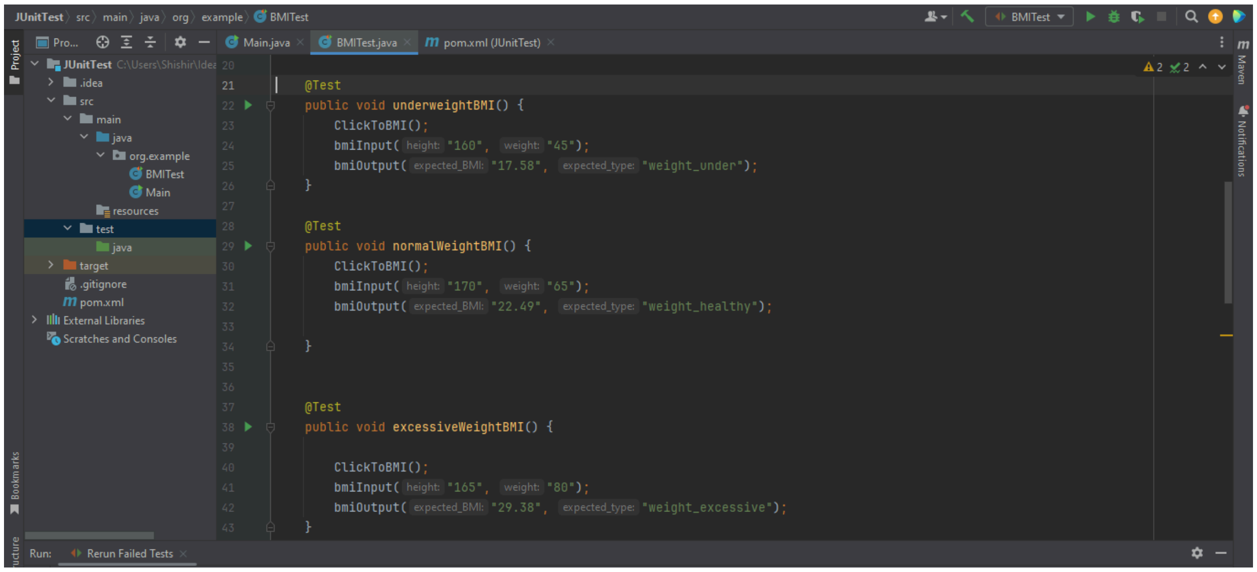


Figure 2 - Performing JUnit testing with Selenium.

6.1.1.4 Write Test cases

Using assertions and test annotation provided by the JUnit framework to verify the expected outcomes of our unit's behavior.

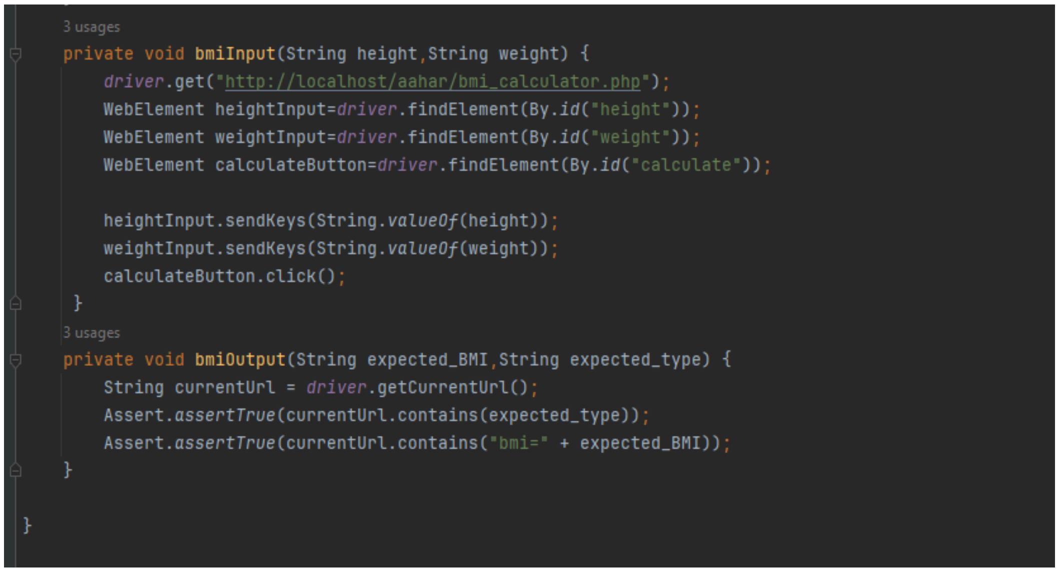


```java

    @Test
    public void underweightBMI() {
        ClickToBMI();
        bmiInput( height: "160",  weight: "45");
        bmiOutput( expected_BMI: "17.58",  expected_type: "weight_under");
    }

    @Test
    public void normalWeightBMI() {
        ClickToBMI();
        bmiInput( height: "170",  weight: "65");
        bmiOutput( expected_BMI: "22.49",  expected_type: "weight_healthy");

    }


    @Test
    public void excessiveWeightBMI() {

        ClickToBMI();
        bmiInput( height: "165",  weight: "80");
        bmiOutput( expected_BMI: "29.38",  expected_type: "weight_excessive");
    }
```

Figure 3 - Writing test cases.

6.1.1.5 Execute Unit Tests

After defining test cases we execute the JUnit test. JUnit test will perform unit testing by using selenium. And show the number of passed and failed test cases. Here, assertions are use for performing the test.



```
    3 usages
    private void bmiInput(String height,String weight) {
        driver.get("http://localhost/aahar/bmi_calculator.php");
        WebElement heightInput=driver.findElement(By.id("height"));
        WebElement weightInput=driver.findElement(By.id("weight"));
        WebElement calculateButton=driver.findElement(By.id("calculate"));

        heightInput.sendKeys(String.valueOf(height));
        weightInput.sendKeys(String.valueOf(weight));
        calculateButton.click();
     }
    3 usages
    private void bmiOutput(String expected_BMI,String expected_type) {
        String currentUrl = driver.getCurrentUrl();
        Assert.assertTrue(currentUrl.contains(expected_type));
        Assert.assertTrue(currentUrl.contains("bmi=" + expected_BMI));
    }

}
```

Figure 4 - Assertions of Junit testing

6.1.16 Repeat for Different Units

Then we repeat this process for my other units.

These testing tools collectively contribute to a robust testing strategy for "আহার," covering both frontend and backend aspects of the web application. They facilitate automation, efficient test execution, and comprehensive reporting, ensuring the reliability and functionality of the food delivery platform.

6.2 Security Testing Tools

To ensuring the utmost security of the "আহার" project, we rely on the robust security testing tool, OWASP ZAP (Open Web Application Security Project Zed Attack Proxy). That takes center stage for its proficiency in active scanning and pinpointing vulnerabilities, offering invaluable insights for strengthening our web-based food delivery application. Complemented by tools such as Burp Suite, Nessus, and Acunetix, our testing process becomes comprehensive and meticulous. Together, these tools play a pivotal role in systematically identifying, analyzing, and recommending remediation for potential security risks, ensuring the robust protection of user data and fortifying user trust in our platform.

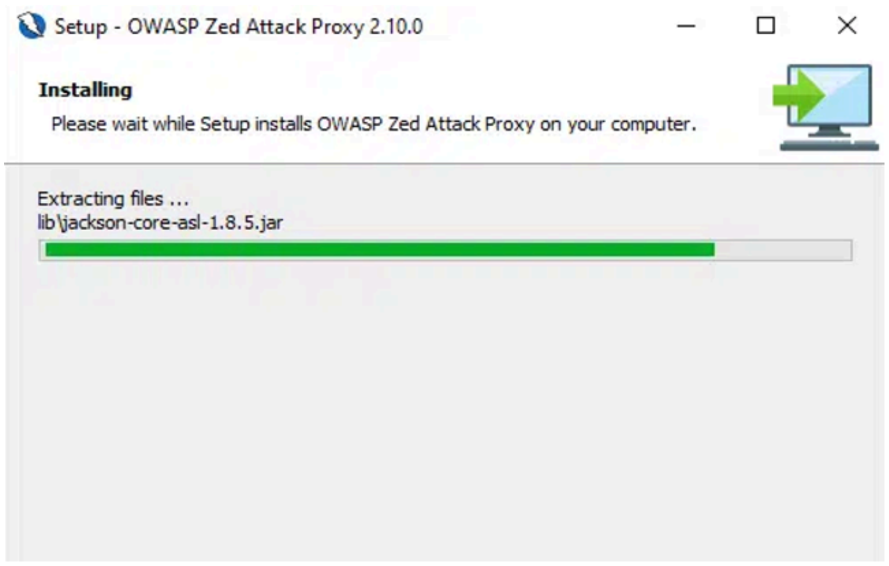
6.2.1 OWASP ZAP

In the "আহার" project, maintaining a secure environment for our web-based food delivery application is of utmost importance. To ensure the robustness of our security measures, we employ OWASP ZAP (Open

Web Application Security Project Zed Attack Proxy), a renowned tool designed for identifying vulnerabilities and potential threats in web applications.The steps of security testing-

6.2.1.1 Setup OWASP ZAP

We start by integrating OWASP ZAP into our testing environment. The setup involves configuring the tool to align with the specifics of the "আহার" application.



Setup - OWASP Zed Attack Proxy 2.10.0

Installing

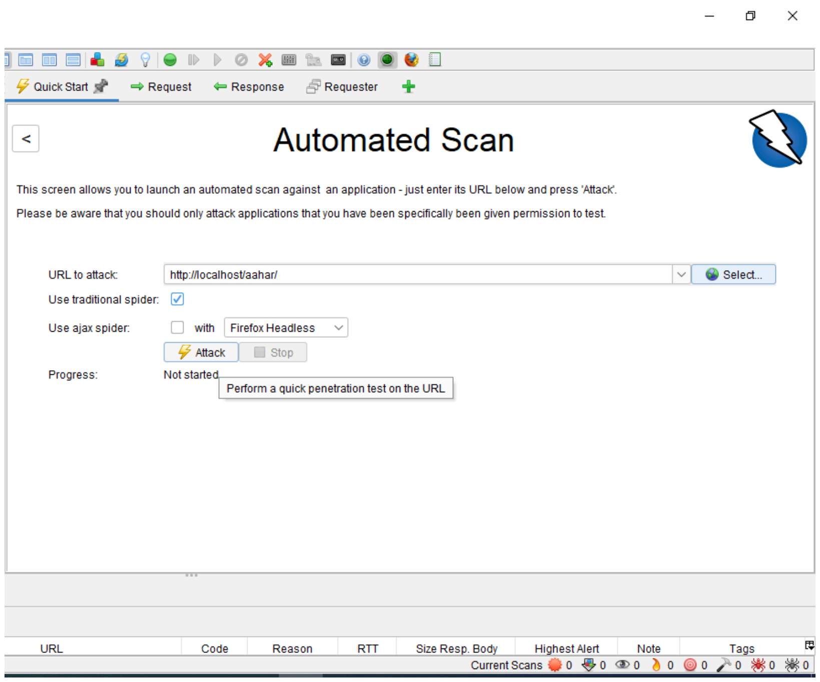
Please wait while Setup installs OWASP Zed Attack Proxy on your computer.

Extracting files ...
lib\jackson-core-asl-1.8.5.jar

Figure 4 - Setup OWASP ZAP

6.2.1.2 Configure Target

Once integrated, we specify the target URL of the "আহার" application within OWASP ZAP. This allows the tool to focus its security scan on our web application. Here, our site's hosting is localhost. The targeted site is - *http://localhost/aahar/*



This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://localhost/aahar/

Use traditional spider: ✓

Use ajax spider: ☐ with Firefox Headless

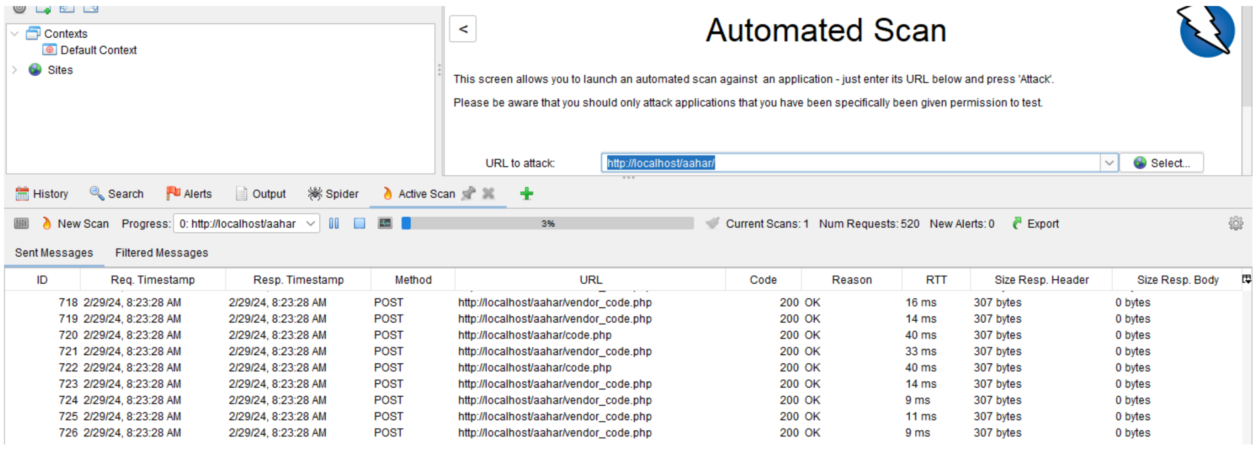
Progress: Not started

Perform a quick penetration test on the URL

Figure 5 - Configure target url

6.2.1.3 Perform Active Scan

With the target configured, we initiate an active scan using OWASP ZAP. This entails simulating potential attacks on our application to uncover vulnerabilities in its code of our project.



This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://localhost/aahar/

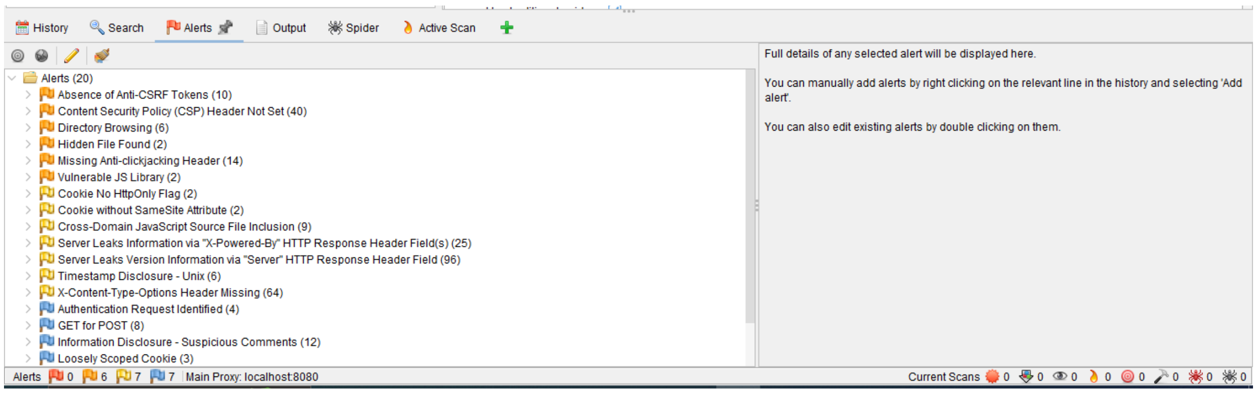
Progress: 0: http://localhost/aahar 3% Current Scans: 1 Num Requests: 520 New Alerts: 0

ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
718	2/29/24, 8:23:28 AM	2/29/24, 8:23:28 AM	POST	http://localhost/aahar/vendor_code.php	200	OK	16 ms	307 bytes	0 bytes
719	2/29/24, 8:23:28 AM	2/29/24, 8:23:28 AM	POST	http://localhost/aahar/vendor_code.php	200	OK	14 ms	307 bytes	0 bytes
720	2/29/24, 8:23:28 AM	2/29/24, 8:23:28 AM	POST	http://localhost/aahar/code.php	200	OK	40 ms	307 bytes	0 bytes
721	2/29/24, 8:23:28 AM	2/29/24, 8:23:28 AM	POST	http://localhost/aahar/vendor_code.php	200	OK	33 ms	307 bytes	0 bytes
722	2/29/24, 8:23:28 AM	2/29/24, 8:23:28 AM	POST	http://localhost/aahar/code.php	200	OK	40 ms	307 bytes	0 bytes
723	2/29/24, 8:23:28 AM	2/29/24, 8:23:28 AM	POST	http://localhost/aahar/vendor_code.php	200	OK	14 ms	307 bytes	0 bytes
724	2/29/24, 8:23:28 AM	2/29/24, 8:23:28 AM	POST	http://localhost/aahar/vendor_code.php	200	OK	9 ms	307 bytes	0 bytes
725	2/29/24, 8:23:28 AM	2/29/24, 8:23:28 AM	POST	http://localhost/aahar/vendor_code.php	200	OK	11 ms	307 bytes	0 bytes
726	2/29/24, 8:23:28 AM	2/29/24, 8:23:28 AM	POST	http://localhost/aahar/vendor_code.php	200	OK	9 ms	307 bytes	0 bytes

Figure 6 - Performing active scan

6.2.1.4 Review Scan Results

OWASP ZAP generates comprehensive scan results that we meticulously review. The results highlight potential security issues such as SQL injection, cross-site scripting (XSS), and other vulnerabilities. After scanning we got 20 security alerts.



Alerts (20)
- Absence of Anti-CSRF Tokens (10)
- Content Security Policy (CSP) Header Not Set (40)
- Directory Browsing (6)
- Hidden File Found (2)
- Missing Anti-clickjacking Header (14)
- Vulnerable JS Library (2)
- Cookie No HttpOnly Flag (2)
- Cookie without SameSite Attribute (2)
- Cross-Domain JavaScript Source File Inclusion (9)
- Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) (25)
- Server Leaks Version Information via "Server" HTTP Response Header Field (96)
- Timestamp Disclosure - Unix (6)
- X-Content-Type-Options Header Missing (64)
- Authentication Request Identified (4)
- GET for POST (8)
- Information Disclosure - Suspicious Comments (12)
- Loosely Scoped Cookie (3)

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.

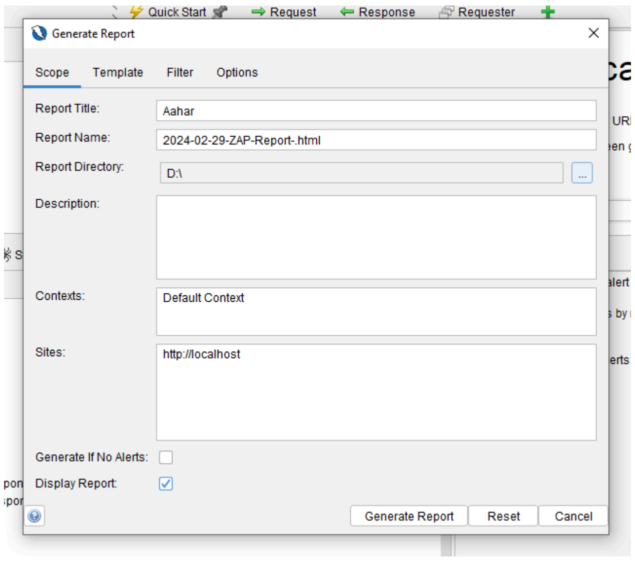
You can also edit existing alerts by double clicking on them.

Alerts 0 6 7 7 Main Proxy: localhost:8080 Current Scans 0 0 0 0 0 0 0 0

Figure 7 - Scanned vulnerabilities

6.2.1.5 Generate Reports

For documentation and analysis, OWASP ZAP offers detailed reports summarizing the identified vulnerabilities and their severity levels. These reports provide actionable insights to prioritize and address security concerns effectively. Click to explore the detailed report.



Report Title: Aahar

Report Name: 2024-02-29-ZAP-Report-.html

Report Directory: D:\

Description:

Contexts: Default Context

Sites: http://localhost

Generate If No Alerts: ☐

Display Report: ✓

Figure 8 - Report generation

By seamlessly integrating OWASP ZAP into our security testing process, we systematically evaluate the resilience of the "আহার" application against real-world cyber threats. The tool's capabilities in simulating attacks contribute significantly to fortifying our security posture, ensuring the confidentiality and integrity of user data on our food delivery platform.

6.3 Performance Testing Tools

Ensuring optimal performance of the "আহার" project's web-based food delivery application is paramount. To achieve this, we leverage JMeter, a robust performance testing tool known for its efficiency in simulating various scenarios and measuring the application's responsiveness under different loads. Complemented by tools such as Apache Benchmark, Gatling, and LoadRunner, our performance testing process becomes comprehensive and meticulous. Together, these tools play a pivotal role in systematically assessing, analyzing, and optimizing the application's performance, guaranteeing a seamless user experience.

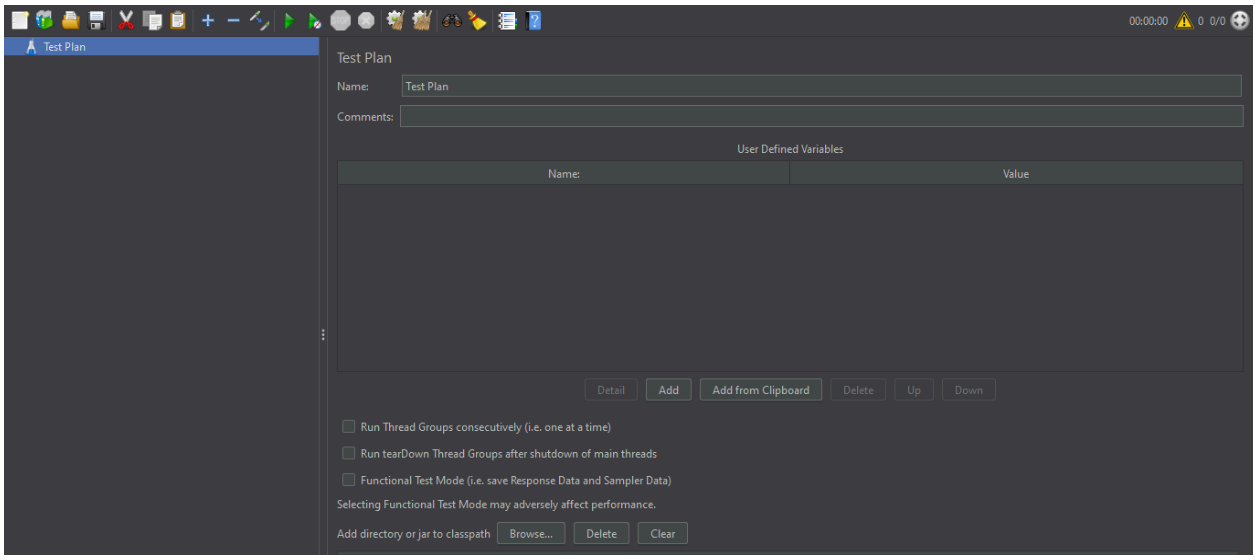
6.3.1 JMeter

In the "আহার" project, maintaining high-performance standards for our web-based food delivery application is crucial. To ensure the efficiency and reliability of our application under varying conditions, we employ JMeter, a widely-used performance testing tool.

The steps of performance testing:

6.3.1.1 Setup JMeter

We initiate the performance testing process by setting up JMeter in our testing environment. This involves configuring the tool to align with the specific requirements of the "আহার" application.


Test Plan
Name: Test Plan
Comments:
User Defined Variables

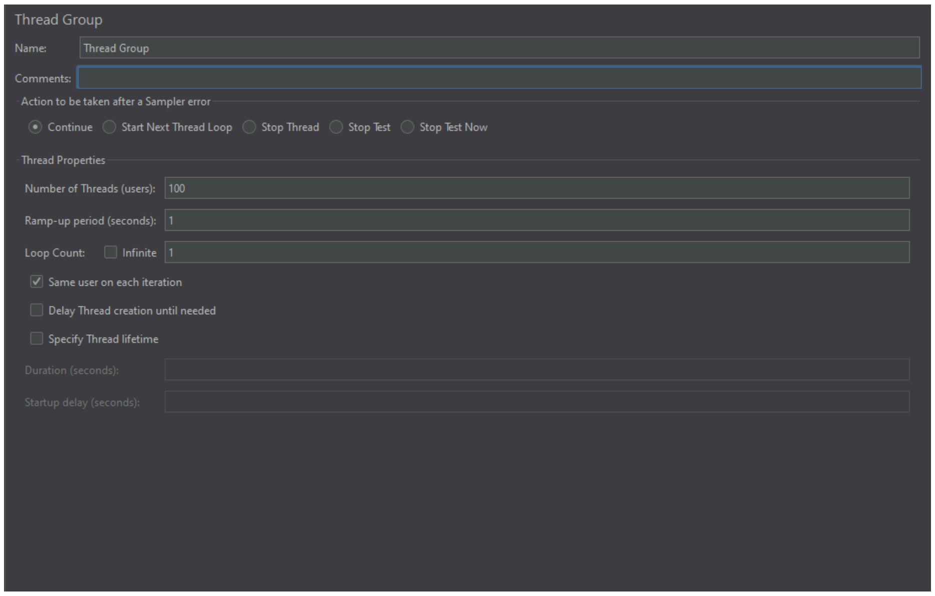
Name:	Value

☐ Run Thread Groups consecutively (i.e. one at a time)
☐ Run tearDown Thread Groups after shutdown of main threads
☐ Functional Test Mode (i.e. save Response Data and Sampler Data)
Selecting Functional Test Mode may adversely affect performance.
Add directory or jar to classpath

Figure 9 - setup JMeter

6.3.1.2 Configure Test Plan

Once integrated, we create a comprehensive test plan within JMeter. This involves specifying various parameters such as the number of virtual users, test duration, and scenarios to be simulated. Our test plan is designed to mimic real-world usage patterns on the "আহার" platform.



Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 100

Ramp-up period (seconds): 1

Loop Count: Infinite 1

[✓ Same user on each iteration]

Delay Thread creation until needed

Specify Thread lifetime

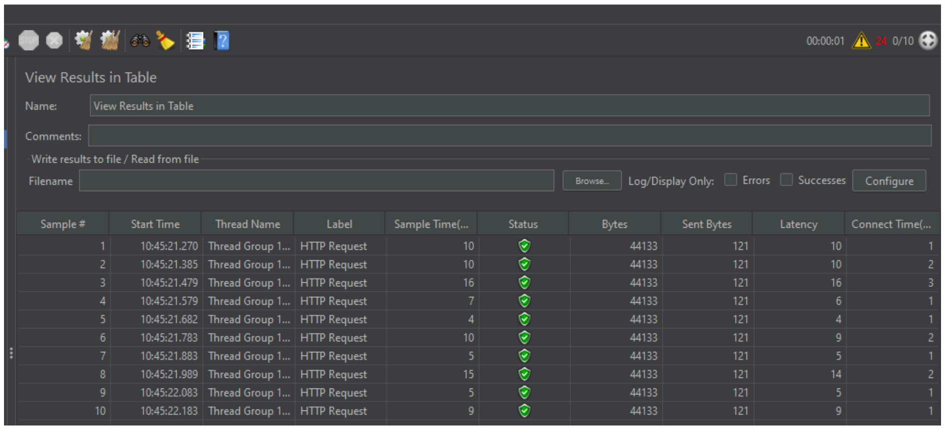
Duration (seconds):

Startup delay (seconds):

Figure 10 - Configure test plan

6.3.1.3 Execute Performance Tests

With the test plan configured, we execute performance tests using JMeter. These tests simulate different levels of user activity, enabling us to measure the application's response time, throughput, and resource utilization under varying loads.



00:00:01 24 0/10

View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename

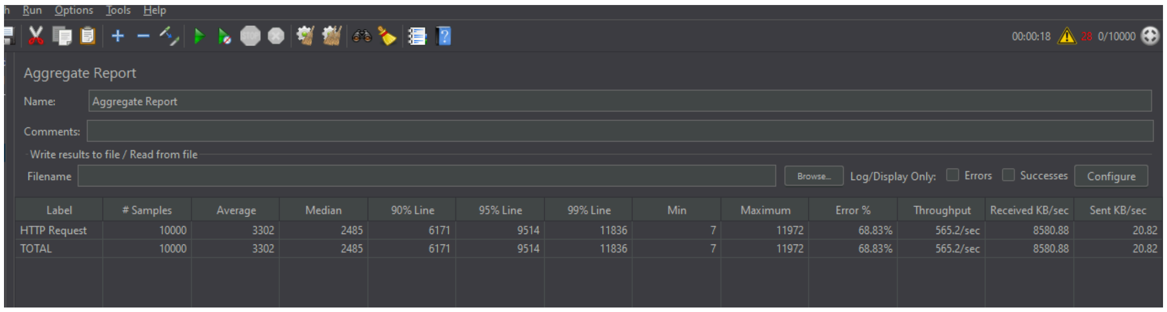
Log/Display Only: Errors Successes

Sample #	Start Time	Thread Name	Label	Sample Time(...	Status	Bytes	Sent Bytes	Latency	Connect Time(...
1	10:45:21.270	Thread Group 1...	HTTP Request	10	✓	44133	121	10	1
2	10:45:21.385	Thread Group 1...	HTTP Request	10	✓	44133	121	10	2
3	10:45:21.479	Thread Group 1...	HTTP Request	16	✓	44133	121	16	3
4	10:45:21.579	Thread Group 1...	HTTP Request	7	✓	44133	121	6	1
5	10:45:21.682	Thread Group 1...	HTTP Request	4	✓	44133	121	4	1
6	10:45:21.783	Thread Group 1...	HTTP Request	10	✓	44133	121	9	2
7	10:45:21.883	Thread Group 1...	HTTP Request	5	✓	44133	121	5	1
8	10:45:21.989	Thread Group 1...	HTTP Request	15	✓	44133	121	14	2
9	10:45:22.083	Thread Group 1...	HTTP Request	5	✓	44133	121	5	1
10	10:45:22.183	Thread Group 1...	HTTP Request	9	✓	44133	121	9	1

Figure 11 - Execute performance test

6.3.1.4 Analyze Test Results

JMeter generates comprehensive test results, including metrics such as response time, latency, and throughput. We meticulously analyze these results to identify performance bottlenecks, potential issues, and areas for improvement.



Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
HTTP Request	10000	3302	2485	6171	9514	11836	7	11972	68.83%	565.2/sec	8580.88	20.82
TOTAL	10000	3302	2485	6171	9514	11836	7	11972	68.83%	565.2/sec	8580.88	20.82

Figure 12 - Analyze test result

6.3.1.5 Generate Performance Reports

For documentation and further analysis, JMeter provides detailed performance reports summarizing key metrics and performance indicators. These reports offer actionable insights to optimize the application's performance effectively.

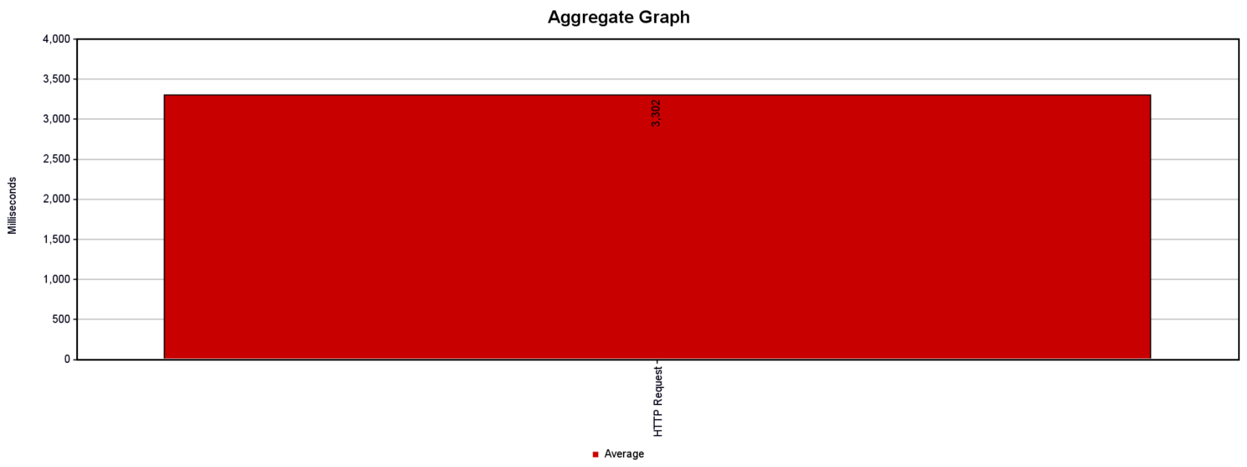


Figure 12 - Performance reports

By seamlessly integrating JMeter into our performance testing process, we systematically evaluate and enhance the "আহার" application's responsiveness and scalability. The tool's capabilities in simulating realistic user loads contribute significantly to delivering a high-performance and reliable food delivery platform for our users.

6.4 Regression Testing Tools

Maintaining the stability and reliability of the "আহার" project's web-based food delivery application is a continuous effort. To ensure that new updates or modifications do not adversely impact existing functionalities, we employ regression testing. For this purpose, we utilize Selenium, a powerful regression testing tool known for its ability to automate web browsers. Complemented by tools such as TestNG, JUnit, and Cypress, our regression testing process becomes comprehensive and meticulous. Together, these tools play a pivotal role in systematically verifying that recent changes have not introduced unintended side effects, ensuring a consistently smooth user experience.

6.4.1 Selenium

In the "আহার" project, maintaining the integrity of our web-based food delivery application is essential. To achieve this, we employ Selenium, a widely-used regression testing tool known for its automation capabilities and cross-browser compatibility.

The steps of regression testing:

6.4.1.1 Setup Selenium

We initiate the regression testing process by setting up Selenium in our testing environment. This involves configuring the tool to interact with the different components and functionalities of the "আহার" application.

6.4.1.2 Create Test Scripts

Once integrated, we create robust test scripts using Selenium. These scripts cover critical functionalities and user workflows within the "আহার" application. TestNG is used to organize and manage these scripts effectively.

6.4.1.3 Execute Regression Tests

With the test scripts in place, we execute regression tests using Selenium. These tests automatically navigate through the application, interacting with various elements to ensure that existing features work as intended after recent updates or changes.

6.4.1.4 Review Test Results

Selenium provides detailed test results, indicating any failures or discrepancies in the application's behavior. We meticulously review these results to identify and address any regression issues that may have arisen due to recent changes.

6.4.1.5 Generate Regression Reports

For documentation and analysis, Selenium generates comprehensive regression reports summarizing the outcomes of the tests. These reports offer insights into the impact of recent changes on existing functionalities and help prioritize necessary corrections.

7. Test Execution and Results

7.1 Test Execution

Test execution for the "আহার" project involves systematically running the designed test cases to validate the functionality, performance, and security of the web-based food delivery application. Test cases are executed across different testing environments and configurations to ensure comprehensive coverage and identify potential issues under various scenarios. Test results are meticulously recorded and analyzed to track the application's behavior, uncover defects, and verify compliance with specified requirements.

To ensure the quality and functionality of the "আহার" project, various test cases were executed using a combination of manual and automated testing approaches:

7.1.1 Functional Testing:

Functional testing is executed using Selenium WebDriver integrated with JUnit for automated testing of the user interface and features. Test scripts are designed to mimic user actions and verify expected outcomes across different browsers, ensuring the application's functional integrity**.**

- **Manual Testing:** We manually interacted with the application, performing actions like user registration, food item searches, menu management, and BMI calculation. This helped identify usability issues and ensure a smooth user experience. Automated Testing: Selenium and JUnit were used to automate repetitive tasks and test specific functionalities. This provided efficient test execution and reduced manual effort.

- **Automated Testing:** Selenium and JUnit were used to automate repetitive tasks and test specific functionalities. This provided efficient test execution and reduced manual effort.

Test Case 1.1: User Registration to Enter into the System

Test Case ID	Status
TC1	PASS
TC2	PASS

TC3	PASS
TC4	PASS

Test Case 1.2: Registration with Existing Email

Test Case ID	Status
TC5	PASS
TC6	PASS

Test Case 2: Changing the Current Password

Test Case ID	Status
TC7	PASS
TC8	PASS
TC9	PASS
TC10	PASS
TC11	PASS

Test Case 3.1: Search Food Items According to User Preferences

Test Case ID	Status
TC12	PASS
TC13	PASS
TC14	FAIL
TC15	PASS
TC16	FAIL

Test Data 4.1: Managing the Menus (Add Item)

Test Case ID	Status
TC17	PASS
TC18	PASS
TC19	PASS
TC20	FAIL

Test Data 4.2: Managing the Menus (Remove Item)

Test Case ID	Status
TC21	PASS
TC22	PASS
TC23	FAIL

Test Data 4.3: Managing the Menus (Edit Item)

Test Case ID	Status
TC24	PASS
TC25	PASS
TC26	PASS

Test Case 5: Calculating BMI

Test Case ID	Status
TC27	PASS
TC28	PASS
TC29	PASS
TC30	FAIL
TC31	FAIL
TC32	FAIL
TC33	FAIL
TC34	PASS

7.1.2 Regression Testing

Objective: Verify that recent code changes haven't adversely affected existing functionalities.
Tools Used: JUnit, Selenium WebDriver.
Process: Previously validated features are re-tested using automated testing tools like JUnit and Selenium WebDriver to ensure stability and identify any regressions caused by recent code modifications.

7.1.3 Performance Testing

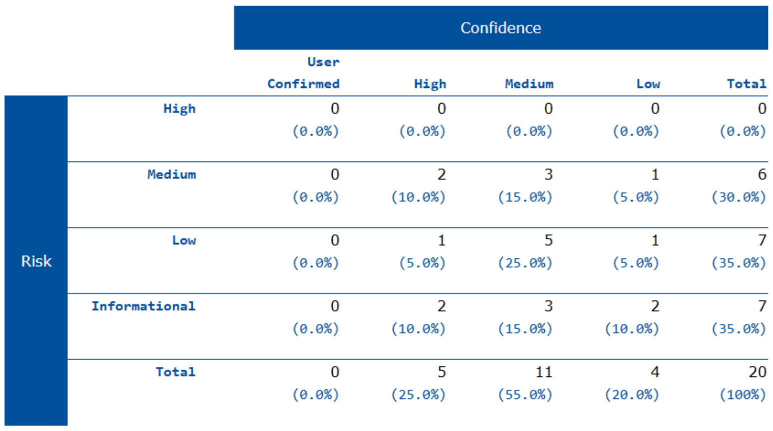
Objective: Assess the system's responsiveness, scalability, and stability under varying workloads.
Tools Used: Apache JMeter.
Process: Realistic user scenarios and workload profiles are simulated using Apache JMeter to measure

key performance metrics such as response times, throughput, and resource utilization.

Number of user (1 Sec)	Passed	Failed
100	100	0
1000	986	14
10000	9514	486

7.1.4 Security Testing

Objective: Identify vulnerabilities and ensure robust protection against potential threats.
Tools Used: OWASP ZAP.
Process: Security vulnerabilities are identified through vulnerability assessments and penetration testing using tools like OWASP ZAP. Compliance with security standards and best practices is verified to safeguard sensitive user information.



		Confidence				
		User Confirmed	High	Medium	Low	Total
Risk	High	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	Medium	0 (0.0%)	2 (10.0%)	3 (15.0%)	1 (5.0%)	6 (30.0%)
	Low	0 (0.0%)	1 (5.0%)	5 (25.0%)	1 (5.0%)	7 (35.0%)
	Informational	0 (0.0%)	2 (10.0%)	3 (15.0%)	2 (10.0%)	7 (35.0%)
	Total	0 (0.0%)	5 (25.0%)	11 (55.0%)	4 (20.0%)	20 (100%)

Figure 13 - Risk analysis from generated report

8. Challenges Faced

During the testing process, several challenges were encountered:

1. **Test Environment Setup:** Setting up the test environment with realistic data and configurations was challenging due to the complexity of the application architecture.
2. **Script Maintenance:** Maintaining and updating test scripts for functional testing required continuous effort, especially when there were frequent changes in the application.

3. **Performance Tuning:** Optimizing test scenarios and load profiles for performance testing to simulate real-world conditions accurately was challenging, particularly in determining the appropriate load levels.
4. **Security Testing Coverage:** Ensuring comprehensive coverage of security testing across all aspects of the application, including both automated scans and manual testing, required careful planning and execution.
5. **Regression Test Suite Maintenance:** Managing and maintaining an extensive regression test suite to validate existing functionalities after each software release posed challenges in identifying relevant test cases, prioritizing them, and optimizing test execution time. Streamlining regression testing processes and automating repetitive tasks were essential for efficient test suite maintenance.
6. **Test Data Management:** Acquiring and managing realistic test data that accurately represents various user scenarios and edge cases proved to be a significant challenge. Ensuring data privacy and compliance while maintaining the diversity and volume of test data added complexity to the testing process.
7. **Scalability Testing:** Assessing the application's ability to handle increasing user loads and data volumes posed challenges in simulating realistic scenarios and predicting performance bottlenecks. Conducting scalable performance testing to identify and address scalability issues required sophisticated testing infrastructure and methodologies.
8. **Regression Test Suite Maintenance:** Managing and maintaining an extensive regression test suite to validate existing functionalities after each software release posed challenges in identifying relevant test cases, prioritizing them, and optimizing test execution time. Streamlining regression testing processes and automating repetitive tasks were essential for efficient test suite maintenance.
9. **Resource Constraints:** Working within resource constraints, including budgetary limitations, time constraints, and staffing limitations, presented challenges in allocating resources effectively and prioritizing testing activities. Balancing competing priorities and optimizing resource utilization were essential for achieving testing objectives within project constraints.

9. Recommendations

Based on our testing experience, the following recommendations are provided:

Automation Framework Implementation: Invest in developing a robust automation framework for functional testing to improve test script maintenance and scalability.

Continuous Integration/Continuous Deployment (CI/CD): Integrate testing into the CI/CD pipeline to enable automated testing with every code change, ensuring early detection of defects.

Enhance Requirement Clarity: Collaborate closely with stakeholders to clarify and refine requirements before test case design. Clear and unambiguous requirements reduce the risk of misunderstandings and ensure comprehensive test coverage.

Invest in Test Automation: Identify repetitive test cases suitable for automation and invest in robust test automation frameworks. Automation can increase test coverage, improve efficiency, and accelerate regression testing while reducing manual effort.

Establish Data Management Procedures: Develop systematic procedures for managing test data, including data generation, anonymization, and maintenance. Implement tools and techniques to streamline data management processes and ensure data integrity.

Maintain Stable Testing Environments: Establish consistent and stable testing environments that closely resemble production settings. Implement version control and configuration management practices to minimize environment discrepancies and ensure reliable test results.

Prioritize Defect Resolution: Define clear criteria for prioritizing defects based on severity, impact, and project objectives. Allocate resources effectively to address critical defects promptly and minimize their impact on project timelines.

Continuous Training and Skill Development: Provide ongoing training and skill development opportunities for testing teams to keep pace with evolving technologies and methodologies. Foster a culture of learning and knowledge sharing to enhance team capabilities.

Improve Communication and Collaboration: Foster open communication and collaboration among cross-functional teams involved in the testing process. Establish regular meetings, status updates, and feedback mechanisms to ensure alignment and transparency.

Implement Metrics-Driven Testing: Define key performance indicators (KPIs) and metrics to measure testing progress, effectiveness, and efficiency. Use metrics to identify areas for improvement, track trends over time, and make data-driven decisions.

10. Conclusion

In conclusion, effective test case design is crucial for ensuring the quality and reliability of software applications. By following systematic approaches, leveraging automation, and addressing key challenges, testing teams can enhance test coverage, efficiency, and effectiveness. Collaboration, communication, and continuous improvement are essential principles that underpin successful test case design efforts. By implementing the recommendations outlined above, organizations can streamline their testing processes, mitigate risks, and deliver high-quality software solutions that meet user expectations and business requirements.