Test Plan Components

1. **Introduction/Overview**
    - **Purpose**: Purpose of the test plan, validating the functionality, stability, and security of the Contact List App's API and UI. This ensures that the application operates as expected and adheres to specified requirements.
    - **Scope**: Focusing on core functionalities like login, contact management (add, edit, delete, view contacts), and API endpoints.
    - **Stakeholders**: Product Owner, QA Team, Developers
2. **Test Objectives**
    - **Primary Goals**:
        - Confirm that each core feature functions as expected.
        - Verify API endpoint correctness for CRUD operations.
        - Ensure the app is free from basic vulnerabilities (e.g., Cross-Site Scripting XSS).
        - Document test results for future reference or automation.
3. **Test Strategy**
    - **Testing Types**:
        - **API Testing**: Ensure all API endpoints respond correctly and adhere to requirements. Refer to sheets
        - **Functional UI Testing**: Verify that user-facing features (like adding/deleting contacts) work smoothly. Refer to sheets
        - **Non-functional Testing**: Validate usability aspects, such as responsiveness and performance. Screenshots on bottom for Jmeter
        - **Security Testing**: Test for basic vulnerabilities like Cross-Site Scripting and SQL injection. Refer to sheets

4. **Test Levels**:

API Testing

Purpose:
API testing ensures that the backend services (like adding, updating, deleting, or retrieving contacts) work as expected. We verify that data flows correctly through the API and that each function meets its requirements.

Test Levels for API Testing:

1. **CRUD Operations Verification:**
    - **Create:** Check that new contacts can be added via the API.
    - **Read:** Verify that contacts can be retrieved correctly.
    - **Update:** Ensure that contact details can be modified.
    - **Delete:** Confirm that contacts can be deleted.
2. **Input Validation:**

    - Test how the API handles valid and invalid inputs (e.g., blank fields, incorrect data types).
3. **Status Codes and Response Validation:**
    - Check if each API endpoint returns the correct status codes (200 for success, 404 for not found, etc.) and response formats as defined in the API documentation.
4. **Error Handling:**
    - Verify that errors are correctly captured and reported, such as “Contact not found” if an invalid ID is used.

UI Testing

Purpose:
UI testing validates that user-facing features, such as adding or deleting contacts, work smoothly and provide a positive user experience. This involves checking the application’s visuals and interactions on the screen.

Test Levels for UI Testing:

1. **Functional Verification:**
    - **Adding Contacts:** Ensure that users can add new contacts by filling out forms and submitting them.
    - **Deleting Contacts:** Confirm that contacts can be deleted from the list with proper confirmation.
    - **Viewing and Editing Contacts:** Verify that users can view and update contact details accurately.
2. **UI Element Functionality:**
    - Confirm that UI elements like buttons, links, and forms behave as expected (e.g., Add Contact button opens the form, Delete button removes a contact).
3. **Error Messages and Alerts:**
    - Check that users see helpful error messages if something goes wrong (e.g., incomplete contact forms).

Non-Functional Testing

Purpose:
Non-functional testing focuses on the app’s performance, usability, and reliability. It ensures that the app can handle load and remains responsive.

Test Levels for Non-Functional Testing:

1. **Performance Testing:**
    - **Load Testing:** Use JMeter to simulate multiple users accessing the application simultaneously, observing performance under expected usage.
    - **Stress Testing:** Gradually increase the load beyond typical usage limits to see how the app reacts under extreme conditions.
    - **Endurance Testing:** Test the app's behavior over an extended period, ensuring it doesn't degrade or slow down over time.
2. **Usability Testing:**
    - **Responsiveness:** Verify that the app is responsive on different devices and screen sizes.
    - **User Experience:** Assess if navigation and interactions are user-friendly and intuitive.

Security Testing

Purpose:
Security testing is designed to uncover vulnerabilities within the application, ensuring it's safe from common attacks such as Cross-Site Scripting (XSS) and SQL Injection.

Test Levels for Security Testing:

1. **Cross-Site Scripting (XSS):**
    - Test input fields (e.g., comments) to ensure any injected scripts are sanitized and don't execute, protecting users from potential malicious scripts.
2. **SQL Injection:**
    - Attempt to enter SQL commands in fields (like login) to check if the app securely handles and blocks these inputs, preventing unauthorized database access.
3. **Authentication and Authorization:**
    - Verify that unauthorized users cannot access restricted features and that the app correctly enforces permissions.
4. **Data Protection:**
    - Ensure sensitive data (e.g., contact details) is securely handled, especially during storage and transmission.

5. **Test Environments**:

Test Environments for Web Application

1. **Browsers**
    - **Primary Browsers** (where the application should perform smoothly):
        - Google Chrome (latest version)

        - Mozilla Firefox (latest version)
        - Safari (for macOS and iOS devices)
        - Microsoft Edge (latest version)
    - **Older Browser Versions** (if the app needs backward compatibility):
        - Chrome versions two versions older than the latest
        - Firefox versions two versions older than the latest
        - Internet Explorer 11 (if specifically requested or required)
2. **Operating Systems**
    - **Desktop OS**:
        - Windows 10, Windows 11
        - macOS (latest version and one previous version)
    - **Mobile OS**:
        - iOS (latest two versions)
        - Android (latest two versions)
3. **Devices**
    - **Desktop Devices**:
        - Windows and macOS-based desktop and laptop configurations with varying resolutions (e.g., 1920x1080, 1366x768).
    - **Mobile Devices**:
        - iPhone (latest model and one previous model)
        - Android phones (popular models such as Samsung Galaxy S series, Google Pixel)
    - **Tablet Devices**:
        - iPad (latest model and one previous model)
        - Samsung Galaxy Tab (latest model and one previous model)
4. **Network Conditions**
    - Testing under different network conditions to simulate real-world user environments:
        - **Wi-Fi (High-Speed)**: To simulate users with stable internet.
        - **3G/4G**: To replicate conditions for users with slower, less stable mobile data.
        - **Offline Mode**: Testing how the app handles network interruptions or disconnects.

Environment Setup for Testing Types

- **Functional UI Testing**:
    - This will be conducted across primary browsers and devices as described above, ensuring essential functions work as expected.
- **API Testing**:
    - Tests can be run directly on a testing server or staging environment, independent of browsers or devices. The API test environment should mirror the production environment closely in terms of data, configurations, and access controls.
- **Non-Functional Testing (Performance, Usability)**:

    - **Load Testing & Stress Testing**: Conducted on a Jmeter
    - **Usability Testing**: Conducted multiple thread groups for consistency, responsiveness, and ease of use.
- **Security Testing**:
    - Use a secure test environment identical to production (especially regarding data handling) to identify vulnerabilities without affecting live data. Ensure firewall and security measures are properly configured to avoid unintended exposure.

Test Deliverables

1. **Documents**
    - **Test Case Spreadsheet**: A comprehensive spreadsheet that includes all test cases for API, functional UI, non-functional, and security testing, covering both the required actions and expected outcomes.
        - **Details in the Spreadsheet**: Each test case will have an ID, title, description, preconditions, test steps, expected and actual results, and pass/fail status.
    - **Defect Logs in Jira**: All identified defects will be documented in Jira, detailing the defect description, steps to reproduce, expected vs. observed behavior, and severity level.
2. **Test Data**
    - **Sample Contact Information**: username- sameerahworsley6@gmail.com password- Cookiefat3$ For testing the contact management features, various data entries will be created. These may include:
        - Names, email addresses, phone numbers, and other relevant details that reflect typical user inputs.
        - Edge case data (e.g., special characters, extremely long names, invalid email formats) to test validation and handling.
    - **Login Credentials**:
        - Test user accounts will be created with fake email addresses and predefined passwords to validate login functionality and access levels.
        - Different user roles, if applicable, to ensure permission-based functionality.
3. **Defect Log**
    - **Jira Tracking**: All defects encountered during testing will be logged in Jira for easy tracking and prioritization.
    - **Information Included in Each Defect Log**:
        - **Defect Description**: A summary of the defect observed.
        - **Steps to Reproduce**: Clear, step-by-step instructions for reproducing the defect.
        - **Expected Behavior**: What should ideally happen if the application is functioning correctly.
        - **Observed Behavior**: What actually happened during the test.

- **Severity Level**: Defining the impact of the defect (e.g., Low, Medium, High), which will assist in prioritizing fixes.

API Endpoints

These are the key API endpoints you'll likely interact with, focusing on the operations available for handling contacts.

Endpoint	HTTP Method	Function	Description
`/contacts`	GET	List Contacts	Retrieves a list of all contacts in the database.
`/contacts/{id}`	GET	Get Contact Details	Retrieves details for a specific contact using an ID.
`/contacts`	POST	Add New Contact	Creates a new contact with the provided details.
`/contacts/{id}`	PUT	Update Contact	Updates the details of an existing contact.
`/contacts/{id}`	DELETE	Delete Contact	Deletes a specific contact by ID.

`/users/login`	POST	User Login	Authenticates a user and provides an access token.
`/users/logout`	POST	User Logout	Ends the user session and invalidates the access token.

These endpoints cover the **CRUD operations** for contacts as well as the login and logout functions for user sessions.

Critical UI Functions

These are the essential user-facing actions you'll test to ensure a smooth experience within the Contact List App.

1. **User Login**:
    - Allows users to log into the app to access and manage contacts.
    - Includes form validation for login fields (e.g., email and password).
2. **Add Contact**:
    - Users can add new contacts with details like name, email, and phone.
    - Includes form validation to ensure only valid information is added.
3. **Edit Contact**:
    - Allows users to update contact details for an existing contact.
    - Validates form fields to prevent errors or invalid entries.
4. **Delete Contact**:
    - Users can delete a contact permanently from their contact list.
    - Confirms deletion to prevent accidental data loss.
5. **Search Contact**:
    - Users can search for specific contacts by name or other criteria.
    - Ensures results are displayed quickly and accurately.
6. **User Logout**:
    - Allows users to log out securely, ensuring no session data remains active after logout.

These functions are key to the Contact List App and represent the primary operations a user will need.

Security Scenarios

To ensure the app is secure, here are the primary security tests to perform, focusing on input validation, session handling, and prevention of unauthorized access.

1. **Input Validation for Contact Fields**:
    - Test the input fields in the contact forms (name, email, phone) to ensure they do not accept invalid, overly long, or potentially malicious data.
    - Validate data formats (e.g., email format, numeric-only phone fields).
2. **Cross-Site Scripting (XSS) Checks**:
    - Test input fields (such as Add Contact or Log in) to ensure they are sanitized and do not allow JavaScript or HTML injection.
    - Example: Enter a script tag in the “Notes” field (e.g., `<script>alert('XSS')</script>`) and confirm it does not execute.
3. **SQL Injection Checks**:
    - Test for SQL injection vulnerabilities by entering SQL syntax in text fields (e.g., “` OR 1=1; --”) and verify the app does not execute unintended commands or crash.
4. **Unauthorized Access Prevention**:
    - Test to ensure that users who are not logged in cannot access protected endpoints or pages.
    - Example: Try to access the contacts page or send an API request to `/contacts` without logging in.

These security tests will help confirm that the Contact List App is secure against some of the most common vulnerabilities.

1. **Test Schedule**
    - **High-Level Timeline**:
        - **Day 1-2**: Familiarize with the app, API documentation, and prepare the test plan.
        - **Day 3-5**: Write detailed test cases for API, UI, security, and non-functional testing.
        - **Day 6-7**: Execute manual tests.
        - **Day 8**: Log defects and prepare a summary report.

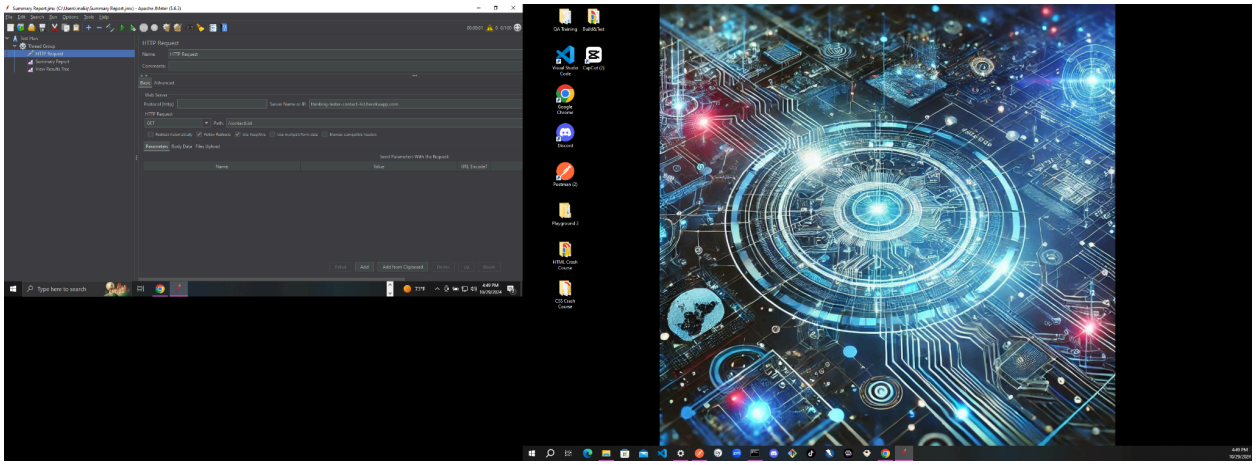
Dependencies

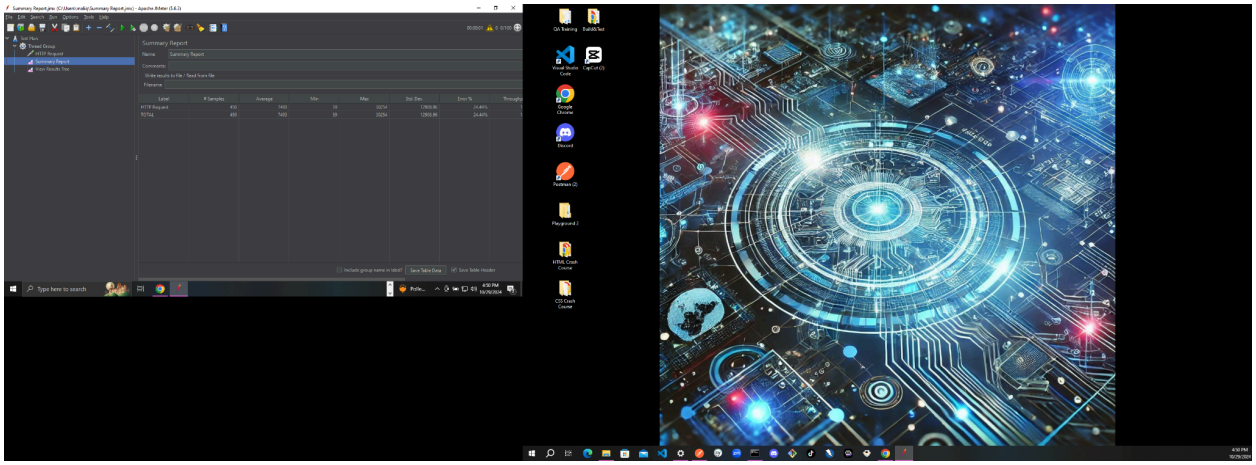
1. **API Documentation Access**:
    - **Requirement**: Access to the API documentation detailing each endpoint's parameters, expected responses, authentication requirements, and error codes.

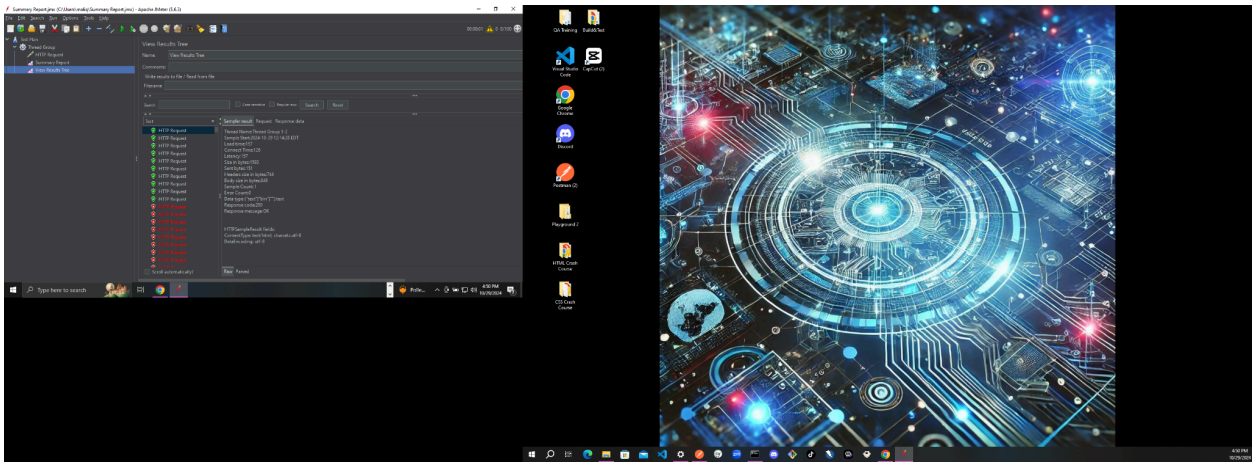
    - **Purpose**: To ensure API tests are developed correctly according to endpoint specifications.
    - **Dependency Resolution**: Verify that the documentation is accurate and up-to-date. Confirm access permissions to view or download the documentation.
2. **Test Data for Contacts**:
    - **Requirement**: Sample contact data, such as names, emails, and phone numbers, to conduct functional and performance testing.
    - **Purpose**: To ensure that tests reflect realistic usage scenarios and allow for accurate validation of the app's behavior.
    - **Dependency Resolution**: Develop a dataset or script to generate test data. Confirm that test data can be cleared or reset as needed.
3. **Access to Bug Tracking Tool (Jira)**:
    - **Requirement**: Access to a Jira project to log and track defects identified during testing.
    - **Purpose**: To document and communicate defects, track their resolution, and provide visibility for all team members.
    - **Dependency Resolution**: Confirm that the QA team has the necessary permissions to log, update, and manage defects in Jira.
4. **Testing Tools**:
    - **Requirement**: Access to tools like JMeter for API and performance testing,and website for UI testing.
    - **Purpose**: To conduct comprehensive functional, performance, and security tests..
5. **Cross-Browser Testing Requirements**:
    - **Requirement**: Access to the most popular browsers (e.g., Chrome, Firefox, Safari, Edge) for UI testing.
    - **Purpose**: To verify consistent behavior across different browsers and platforms.
    - **Dependency Resolution**: Confirm that testing can be performed on required browsers, and that a device lab or cloud service (like BrowserStack or Sauce Labs) is available if needed.

Jmeter Results

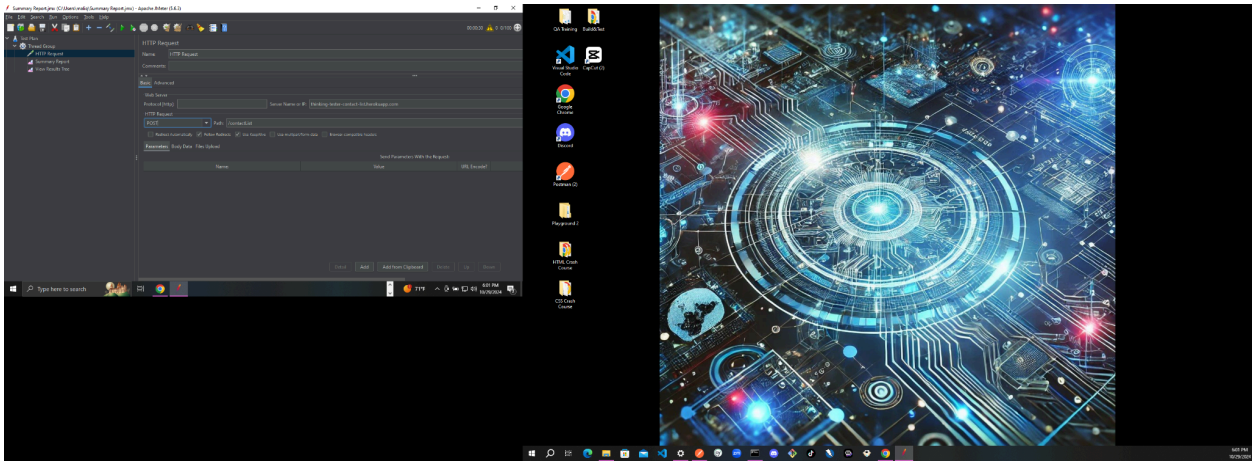
GET

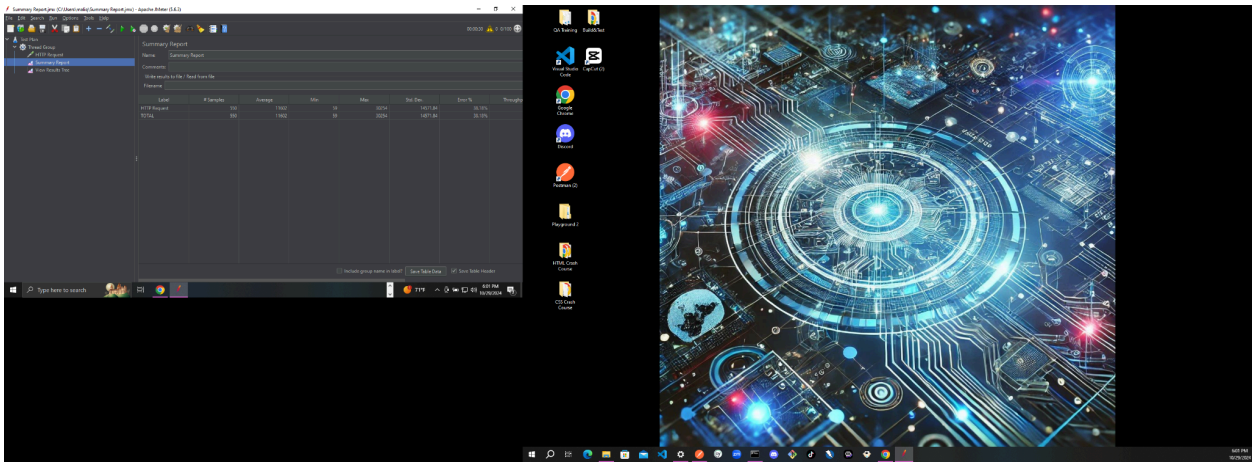


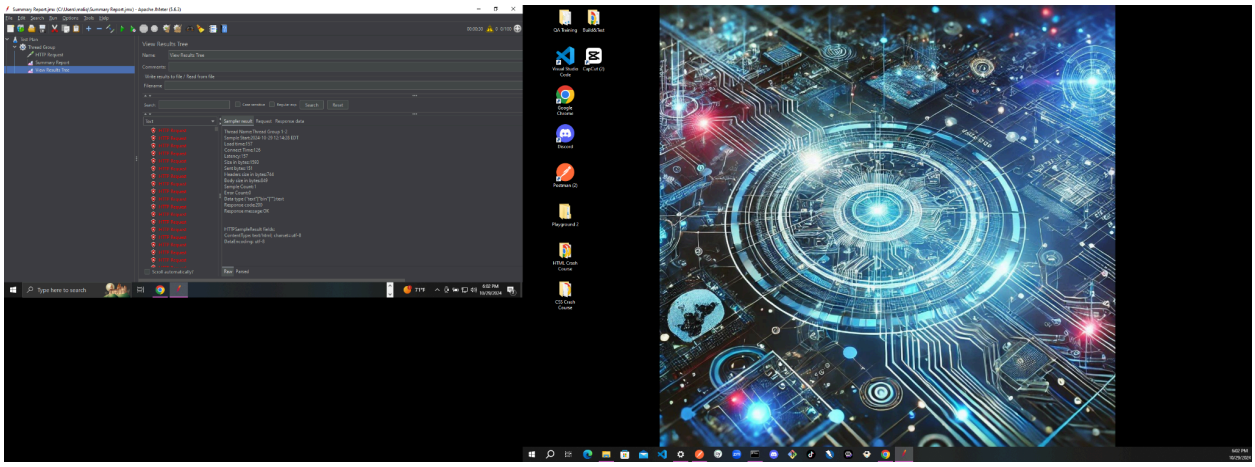




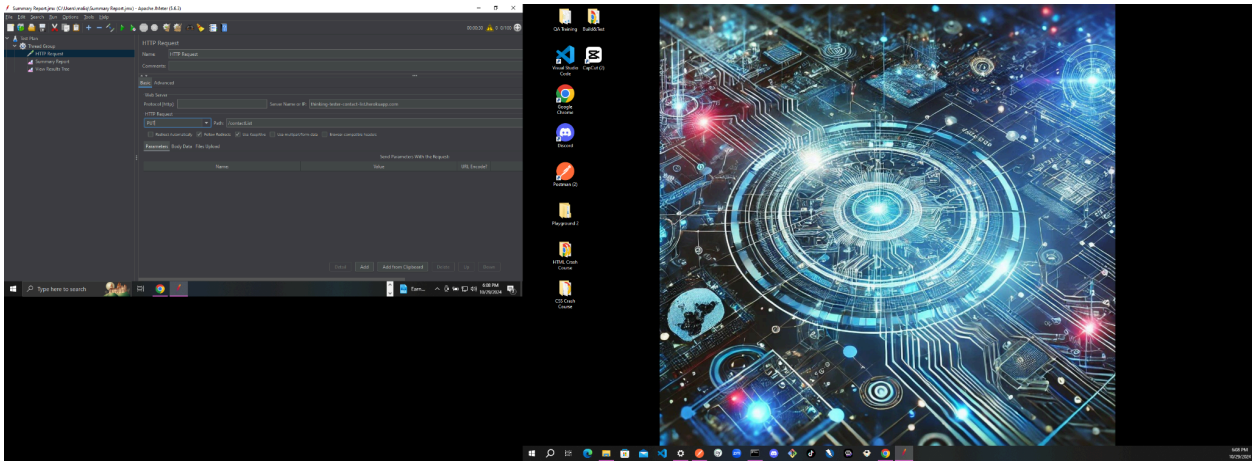
POST

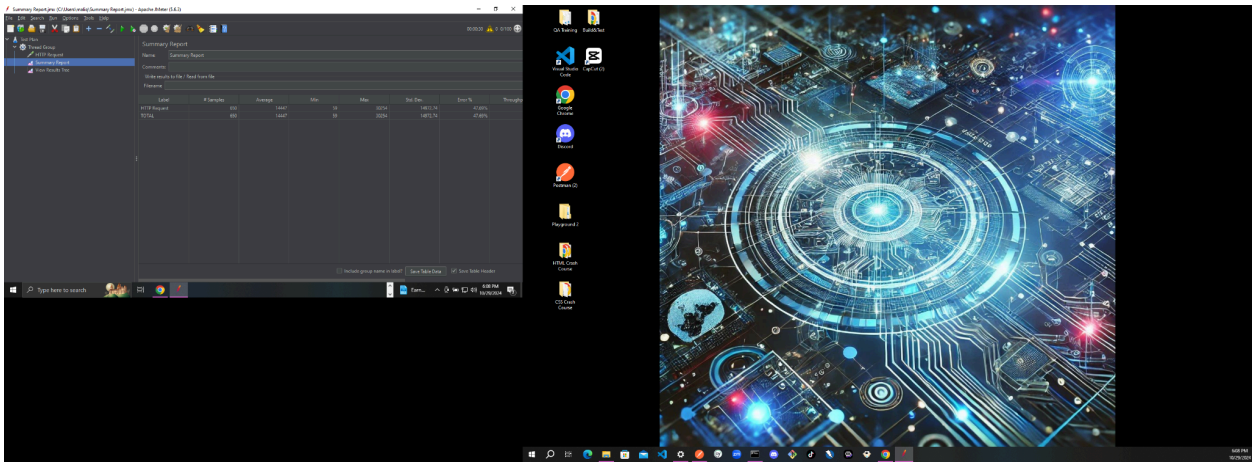


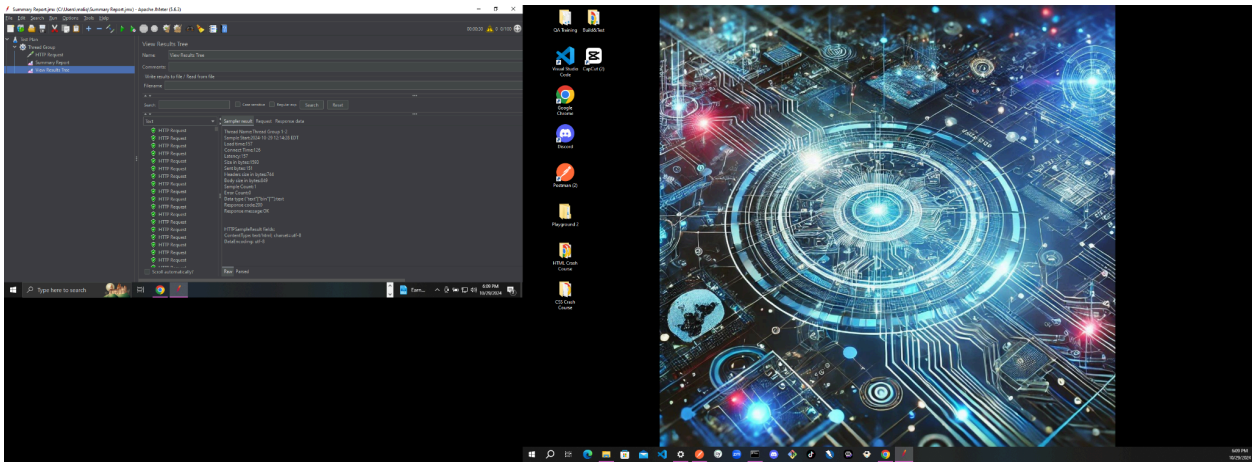




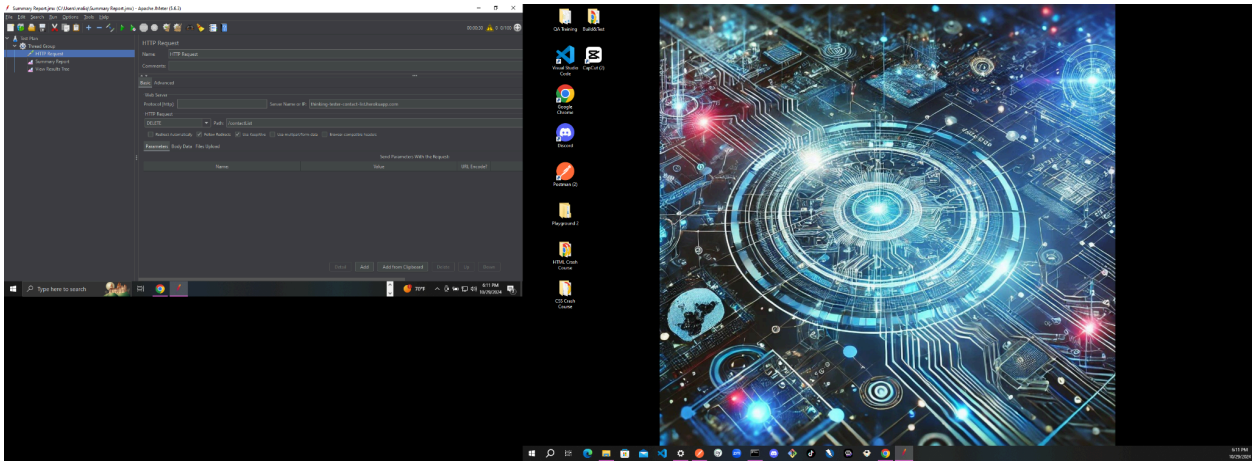
PUT

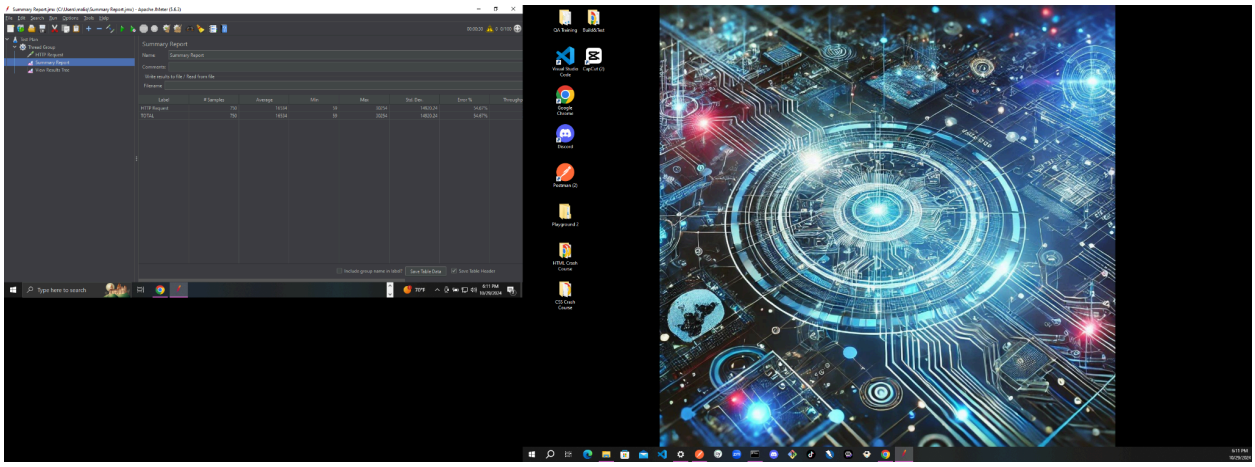


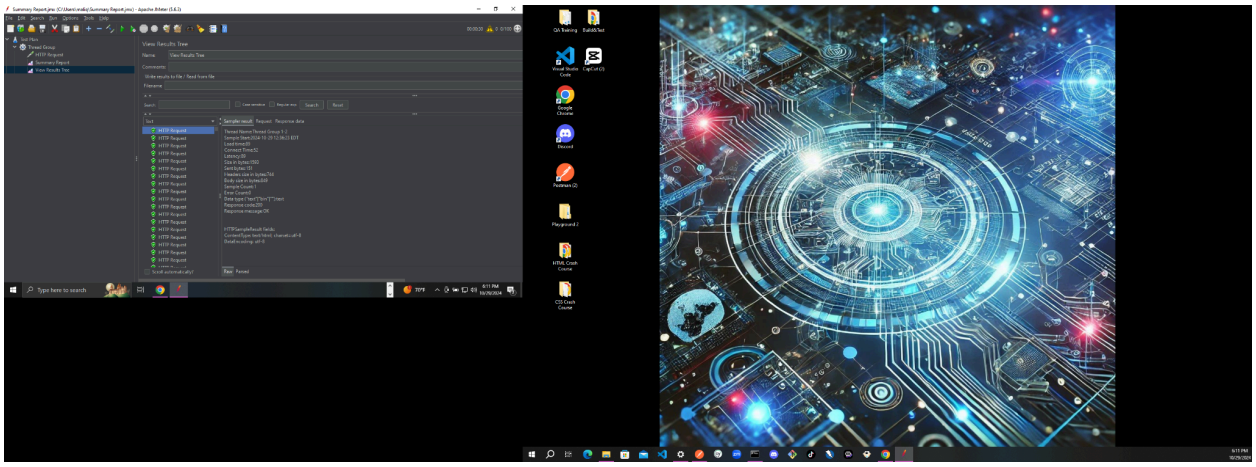




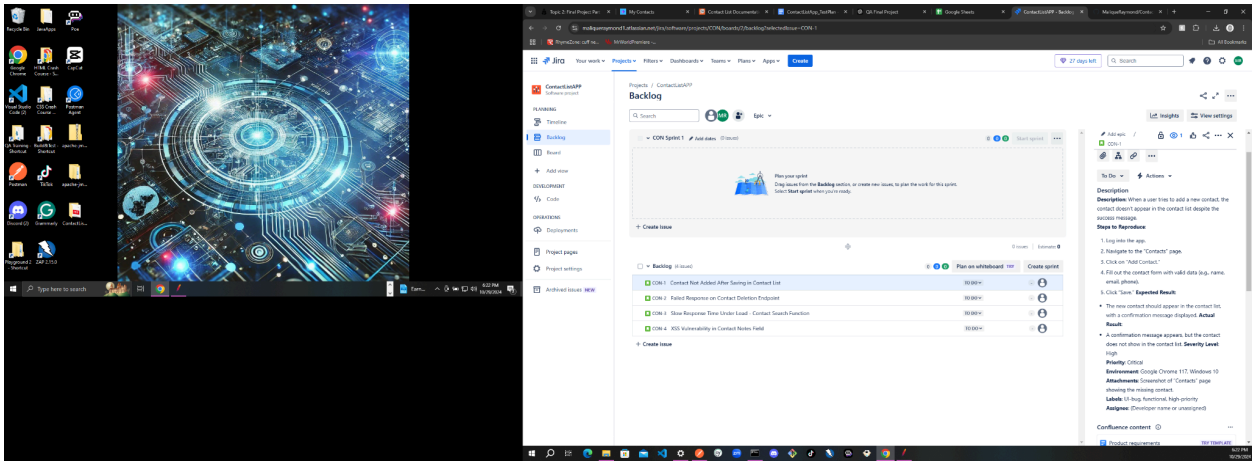
DELETE

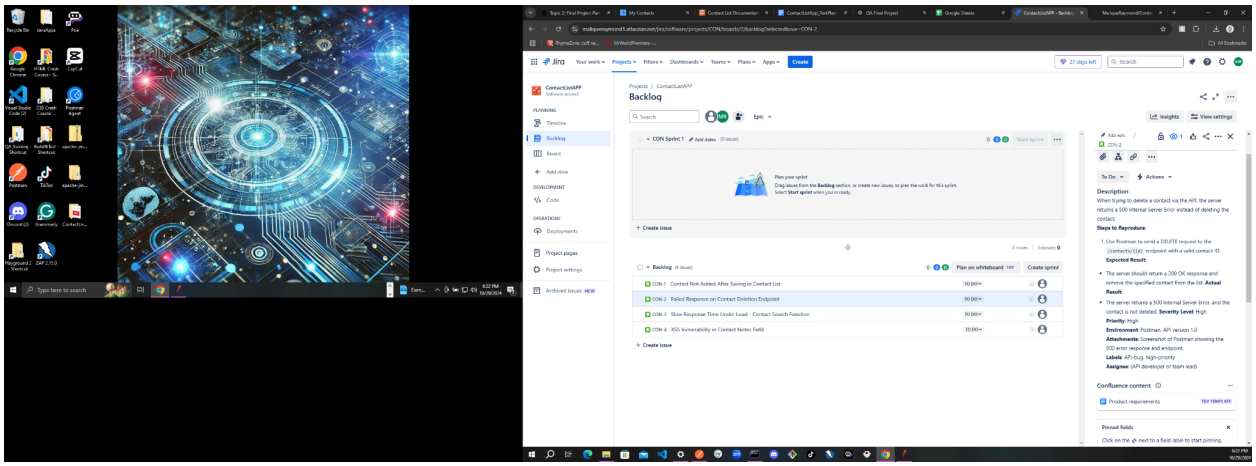


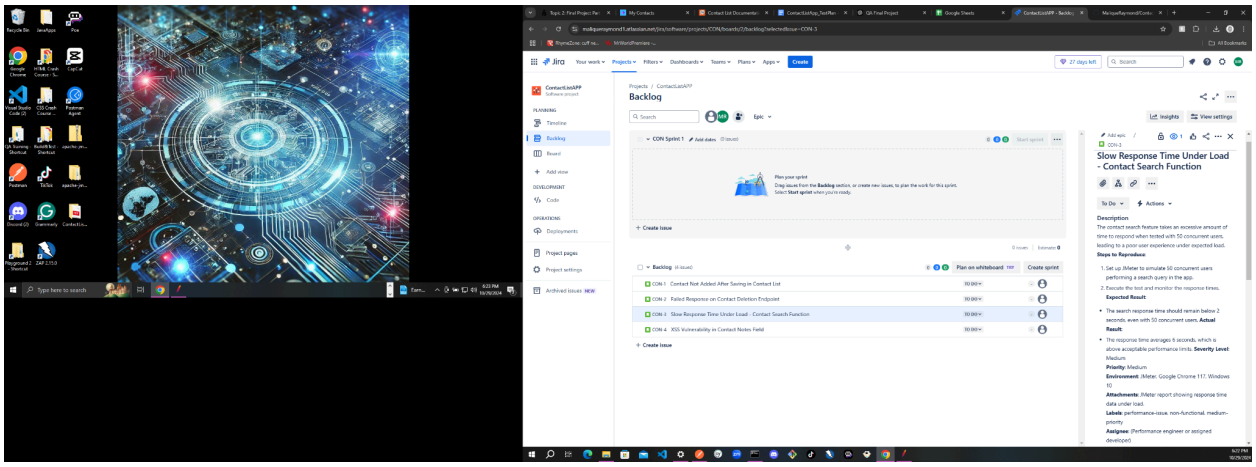


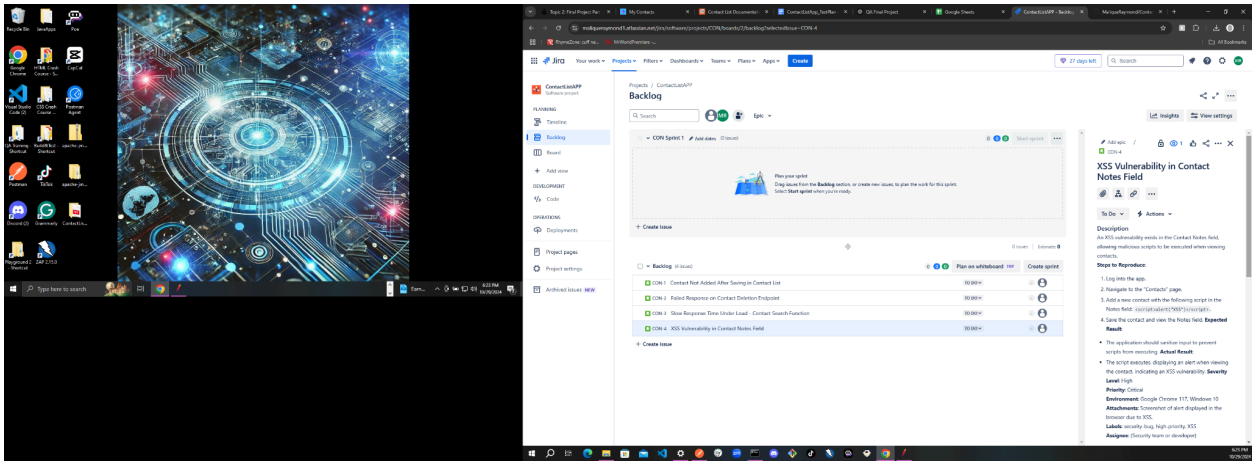


JIRA Screenshots






Backlog
CON Sprint 1
Plan your sprint
Drag issues from the Backlog section, or create new issues, to plan the work for this sprint.
Select Start sprint when you're ready.
Create issue
Backlog (4 issues)
CON-1 Contact Not Added After Saving in Contact List
CON-2 Failed Response on Contact Deletion Endpoint
CON-3 Slow Response Time Under Load - Contact Search Function
CON-4 XSS Vulnerability in Contact Notes Field
Slow Response Time Under Load - Contact Search Function
Description
The contact search feature takes an excessive amount of time to respond when tested with 50 concurrent users, leading to a poor user experience under expected load.
Steps to Reproduce:
1. Set up JMeter to simulate 50 concurrent users performing a search query in the app.
2. Execute the test and monitor the response times.
Expected Result:
• The search response time should remain below 2 seconds even with 50 concurrent users. Actual Result:
• The response time averages 6 seconds, which is above acceptable performance limits. Severity Level: Medium
Priority: Medium
Environment: JMeter, Google Chrome 117, Windows 10
Attachments: JMeter report showing response time data under load.
Labels: performance-issue, non-functional, medium-priority
Assignee: (Performance engineer or assigned developer)


Backlog
CON Sprint 1
Plan your sprint
Drag issues from the Backlog section, or create new issues, to plan the work for this sprint.
Select Start sprint when you're ready.
Create issue
Backlog (4 issues)
CON-1 Contact Not Added After Saving in Contact List
CON-2 Failed Response on Contact Deletion Endpoint
CON-3 Slow Response Time Under Load - Contact Search Function
CON-4 XSS Vulnerability in Contact Notes Field
XSS Vulnerability in Contact Notes Field
Description
An XSS vulnerability exists in the Contact Notes field, allowing malicious scripts to be executed when viewing contacts.
Steps to Reproduce:
1. Log into the app.
2. Navigate to the "Contacts" page.
3. Add a new contact with the following script in the Notes field: <script>alert("XSS")</script>.
4. Save the contact and view the Notes field. Expected Result:
• The application should sanitize input to prevent scripts from executing. Actual Result:
• The script executes, displaying an alert when viewing the contact, indicating an XSS vulnerability. Severity Level: High
Priority: Critical
Environment: Google Chrome 117, Windows 10
Attachments: Screenshot of alert displayed in the browser due to XSS.
Labels: security bug, high-priority, XSS
Assignee: (Security team or developer)