Static Application Security Testing (SAST) using SonarQube

Tool Used: SonarQube (Dockerized)
Platforms: Windows + Docker + Mendix Applications

1. Overview

This section documents the step-by-step process of implementing **SAST using SonarQube**, analyzing three Mendix-based applications:

- **Coffee Service App**
- **Purchase Request Mendix App**
- **Task Tracker Mendix App**

The process includes:

- Setting up SonarQube in Docker
- Configuring project keys and tokens
- Running scans via Docker SonarScanner
- Analyzing vulnerabilities and security hotspots
- Generating reproducible scripts for other researchers

2. Step-by-Step Guide (Windows + Docker + SonarQube)

Step 1: Install & Verify Basics (One-time)

- Install **Docker Desktop (WSL2 backend)**
- Verify PowerShell is installed
- Check Docker is running:

docker version

Step 2: Start SonarQube in Docker (One-time)

Run or reuse a container named sonarqube on port 9000:

docker ps -a --filter "name=sonarqube"

docker run -d --name sonarqube -p 9000:9000 sonarqube:lts-community

Login to SonarQube at http://localhost:9000 using admin/admin and set a new password.

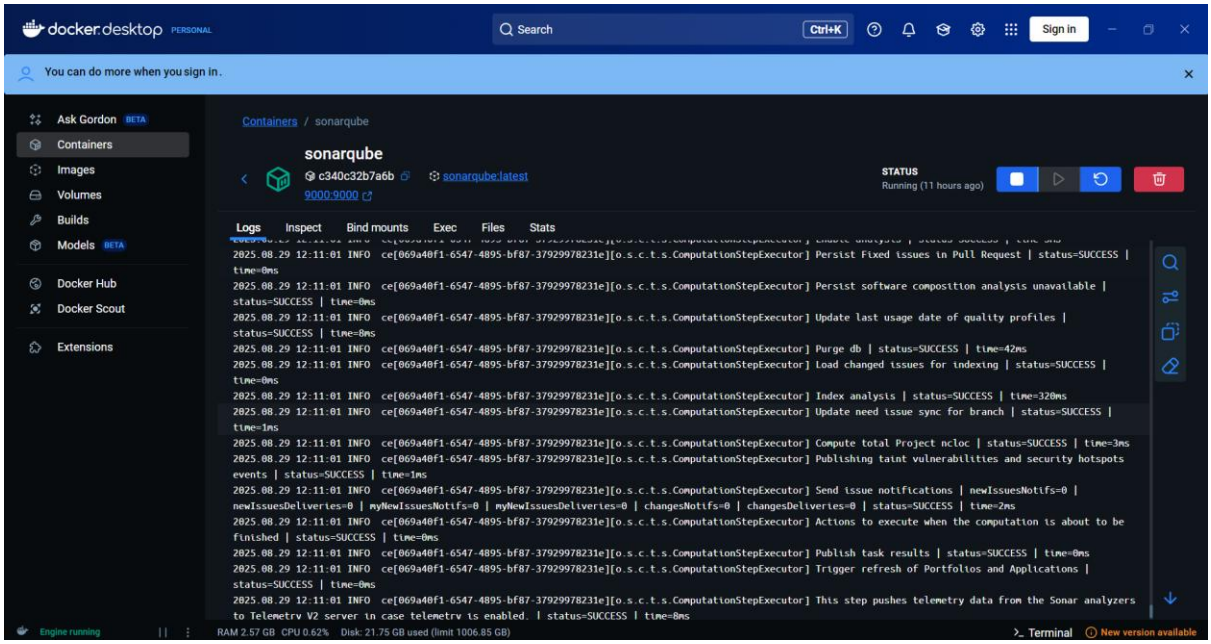


Figure 1: SonarQube Setup on Docker

Step 3: Create SonarQube Projects (One-time)

Project Key	Project Name
Coffee-Service-App	Coffee Service App
Purchase-Request-Mendix-App	Purchase Request Mendix App
Task-Tracker-Mendix-App	Task Tracker Mendix App

Set project keys to match each repository's **sonar-project.properties** file.

Step 4: Generate Project Token (One-time)

- Navigate: *Avatar → My Account → Security → Generate Tokens*
- Name: mendix-sast → Generate → Copy token (e.g., squ_...)

Set PowerShell environment variables:

$env:SONAR_HOST_URL = "http://host.docker.internal:9000"

$env:SONAR_TOKEN = "squ_...paste-your-token..."

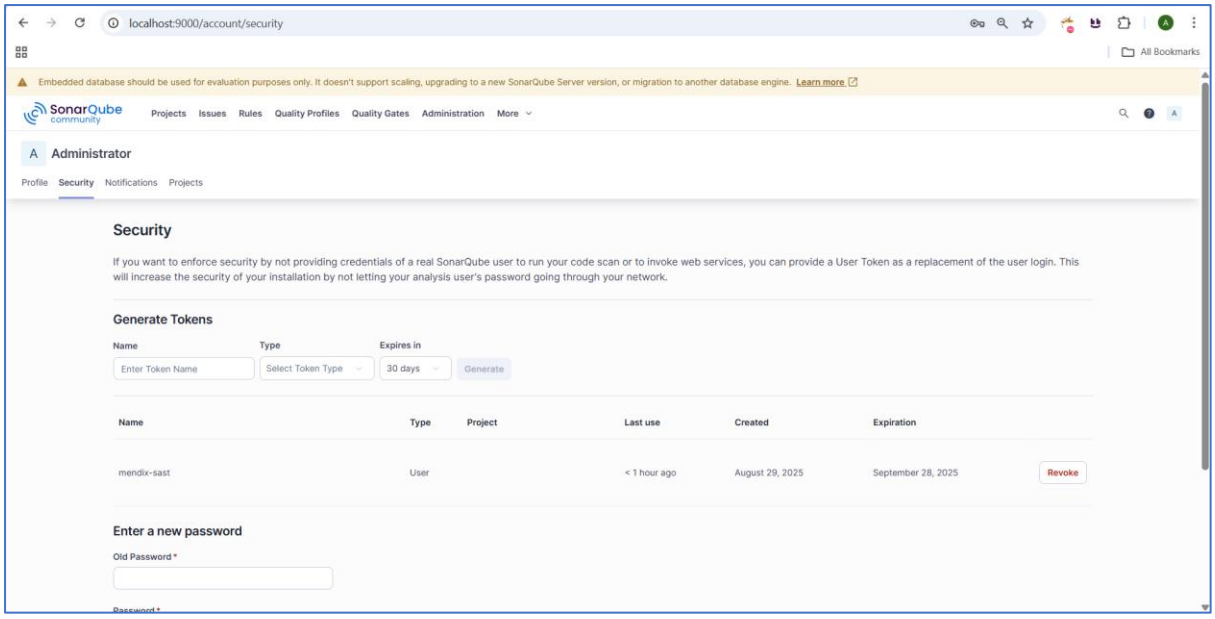
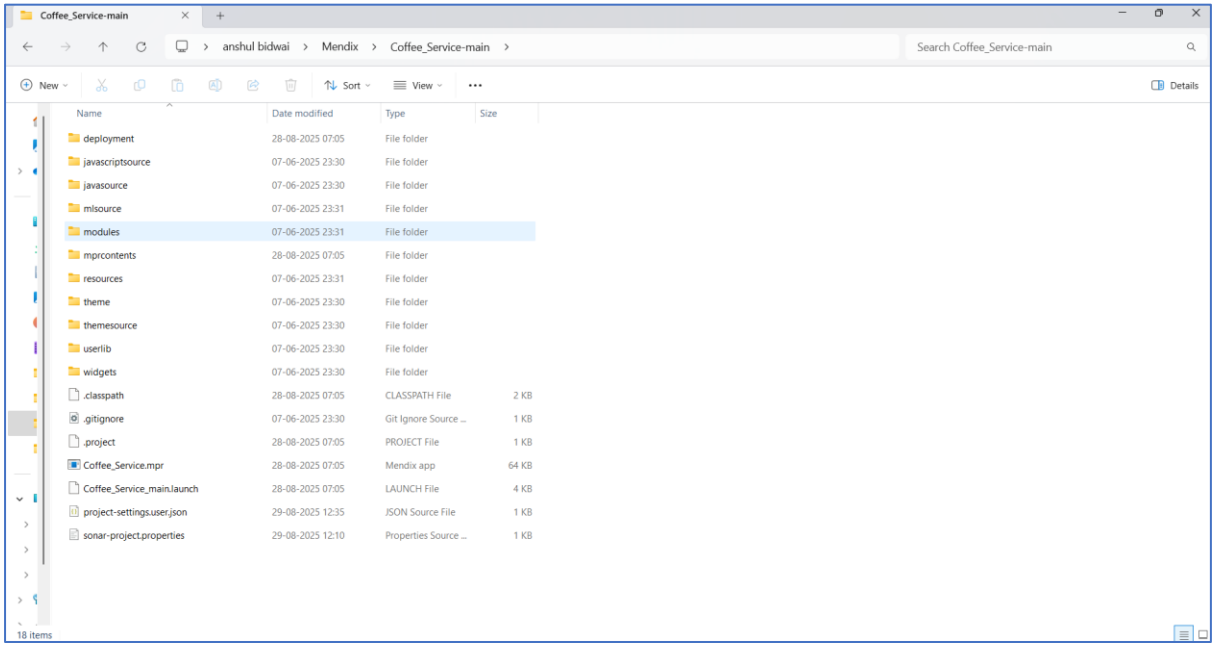


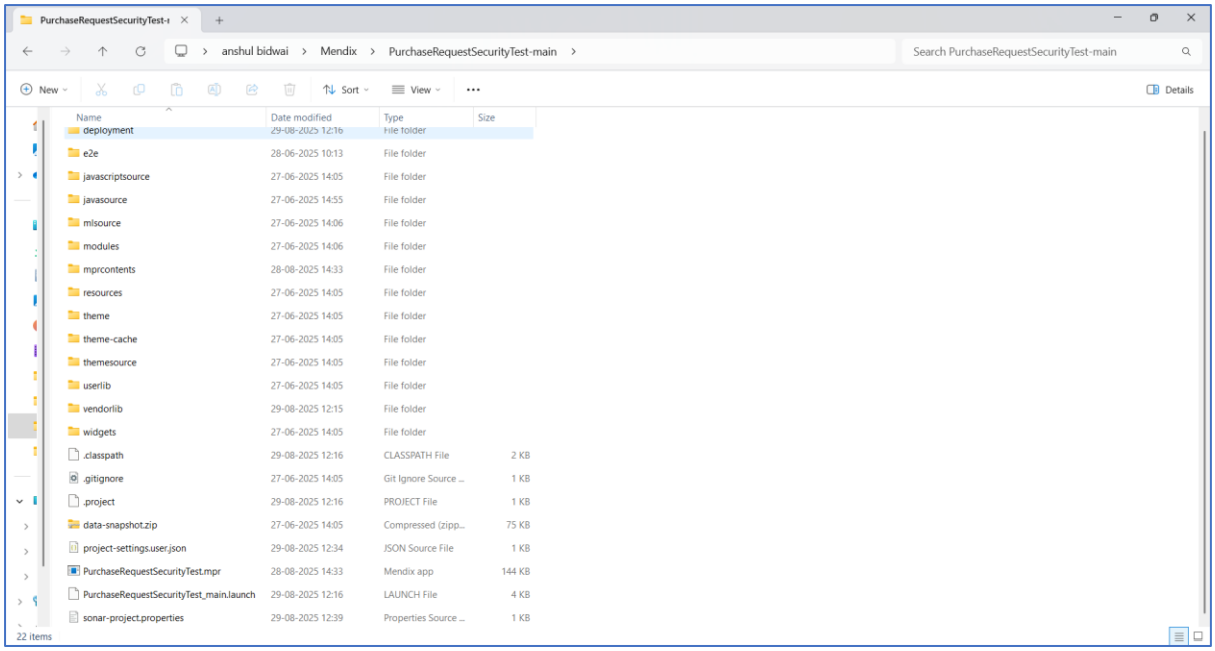
Figure 2: Token Generation Screen

Step 5: Ensure Code Folders Exist

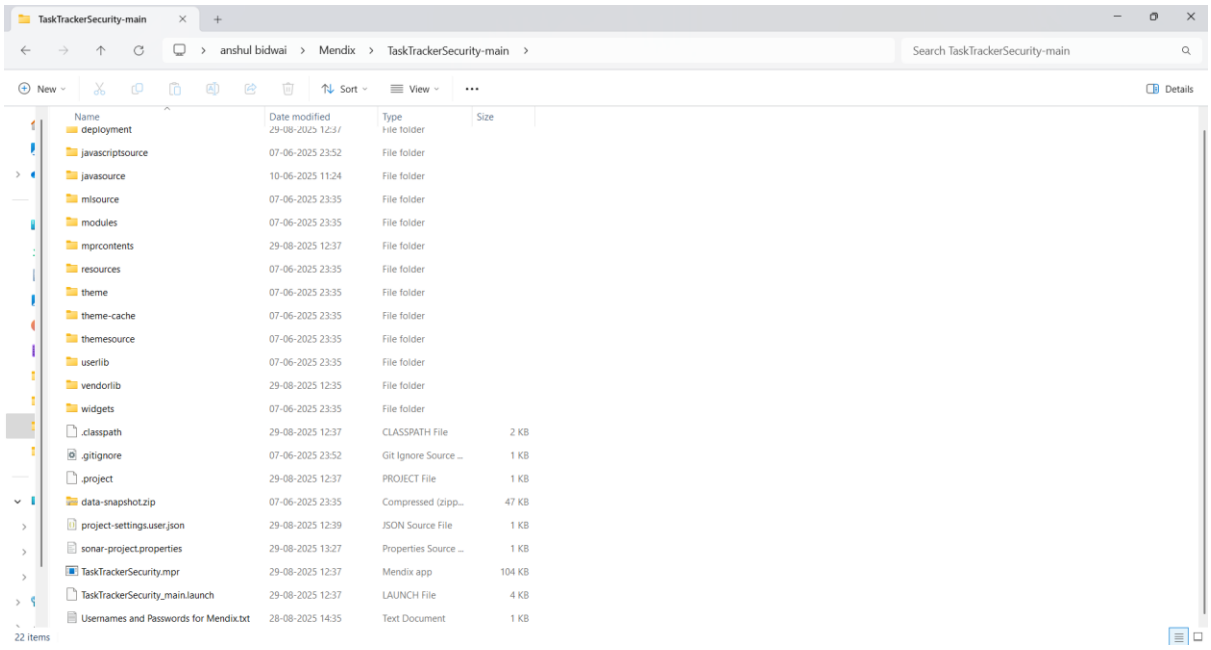
- Coffee: C:\Users\anshu\Mendix\Coffee_Service-main



- Purchase: C:\Users\anshu\Mendix\PurchaseRequestSecurityTest-main



- Task Tracker: C:\Users\anshu\Mendix\TaskTrackerSecurity-main



Step 6: Add sonar-project.properties to Each Repository

Example for **Purchase Request App**:

sonar.projectKey=Purchase-Request-Mendix-App

sonar.projectName=Purchase Request Mendix App

sonar.projectVersion=1.0

sonar.sources=.

sonar.sourceEncoding=UTF-8

sonar.java.binaries=deployment/run/bin

sonar.scm.disabled=true

sonar.exclusions=deployment/**,themesource/**,node_modules/**,**/*.min.js,**/*.map

sonar.javascript.node.maxspace=6144

This ensures stable analysis by excluding large auto-generated files.

Step 7: Run SonarScanner for Each App

Example for **Task Tracker App**:

```
docker run --rm `
  -e SONAR_HOST_URL=$env:SONAR_HOST_URL `
  -e SONAR_TOKEN=$env:SONAR_TOKEN `
  -v "C:\Users\anshu\Mendix\TaskTrackerSecurity-main:/usr/src" `
  sonarsource/sonar-scanner-cli "-Dsonar.javascript.node.maxspace=6144"
```

Step 8: Confirm Results in SonarQube

- Coffee: http://localhost:9000/dashboard?id=Coffee-Service-App
- Purchase: http://localhost:9000/dashboard?id=Purchase-Request-Mendix-App
- Task: http://localhost:9000/dashboard?id=Task-Tracker-Mendix-App

3. Key Vulnerability Example: Weak Cipher Mode

Issue

Use another cipher mode or disable padding.
This vulnerability exposes encrypted data to potential plaintext recovery attacks.

Impact

- Theft of sensitive data
- Tampering with encrypted messages
- Reduced cryptographic strength

Noncompliant Code (AES/CBC)

```
import javax.crypto.Cipher;
```

```
public static void main(String[] args) {
   Cipher.getInstance("AES/CBC/PKCS5Padding"); // Noncompliant
}
```

Compliant Code (AES/GCM)

```
import javax.crypto.Cipher;


public static void main(String[] args) {
   Cipher.getInstance("AES/GCM/NoPadding"); // Secure
}
```

4. For Other Researchers (Reproducible Setup)

- Ready-to-use **PowerShell script (run-sast.ps1)**
- Portable **sonar-project.properties templates**
- Handles:
    - Java binaries pathing
    - Git SCM disabled for Docker
    - Large JS bundle exclusion
    - Node heap tuning (6144 MB)

How to Run

```
$env:SONAR_HOST_URL = "http://host.docker.internal:9000"
$env:SONAR_TOKEN    = "squ_your_token_here"
powershell -ExecutionPolicy Bypass -File .\scripts\run-sast.ps1 `
 -CoffeeRepoPath   "C:\path\to\Coffee_Service" `
 -PurchaseRepoPath "C:\path\to\PurchaseRequest" `
 -TaskRepoPath     "C:\path\to\TaskTracker" `
 -OutRoot ".\tests\SAST" `
 -JsMaxSpaceMB 6144
```

5. Novel Contributions & Impact

1. **Mendix-specific SAST Stabilization** – Targeted exclusions and Java binaries pathing.

2. **Reproducible Pipeline** – Containerized, parameterized, and environment-independent.
3. **Failure Mode Documentation** – Token errors, SCM failures, Node OOM resolved systematically.
4. **Research-Friendly Packaging** – One-command orchestration, reproducible artifacts.

Impact: Reliable, portable, and auditable SAST pipeline for low-code/Mendix projects.

6. Metrics for Validation

- **Completion Rate:** Percentage of projects that scanned successfully.
- **False Positive Reduction:** After applying exclusions.
- **Stability Improvement:** Reduction of “bridge server unresponsive” incidents.
- **Reproducibility:** Identical issue counts across machines.

7. Artifacts Provided

- /scripts/run-sast.ps1
- sonar-project.properties templates
- tests/SAST/<app>/scanner.log

Dynamic Application Security Testing (DAST) – Coffee Service Application

A. Tools and Configuration

- **Tool:** OWASP ZAP
- **Proxy Configuration:** FoxyProxy used for interception

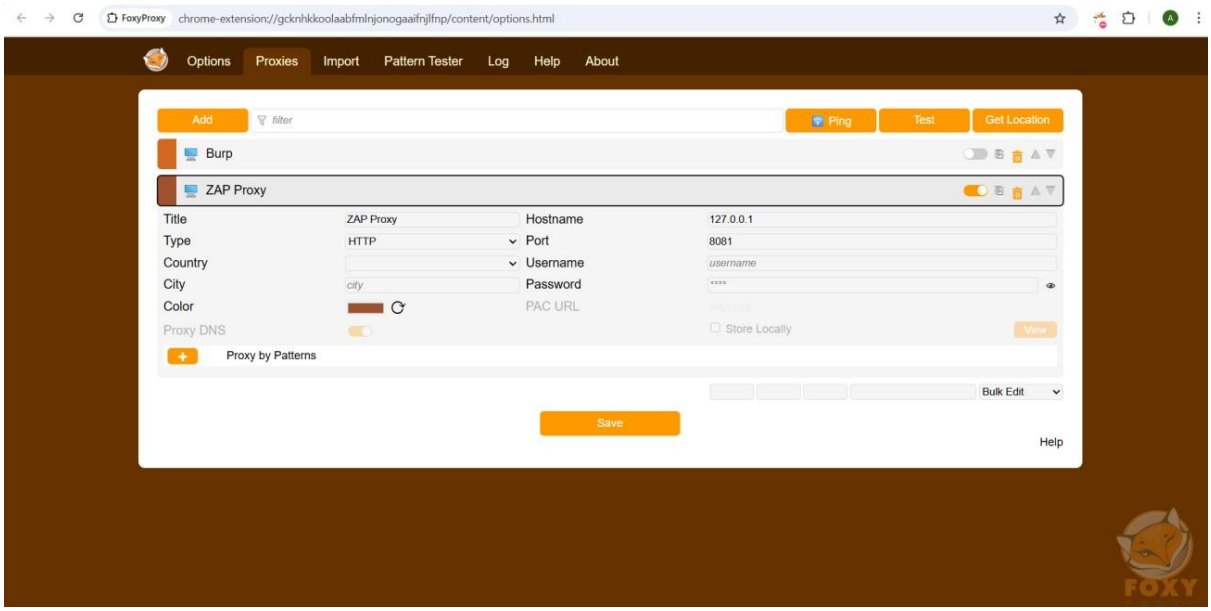


Fig. 1: Zap Proxy Setting Using FoxyProxy

- **API Testing:** Postman for manual endpoint validation

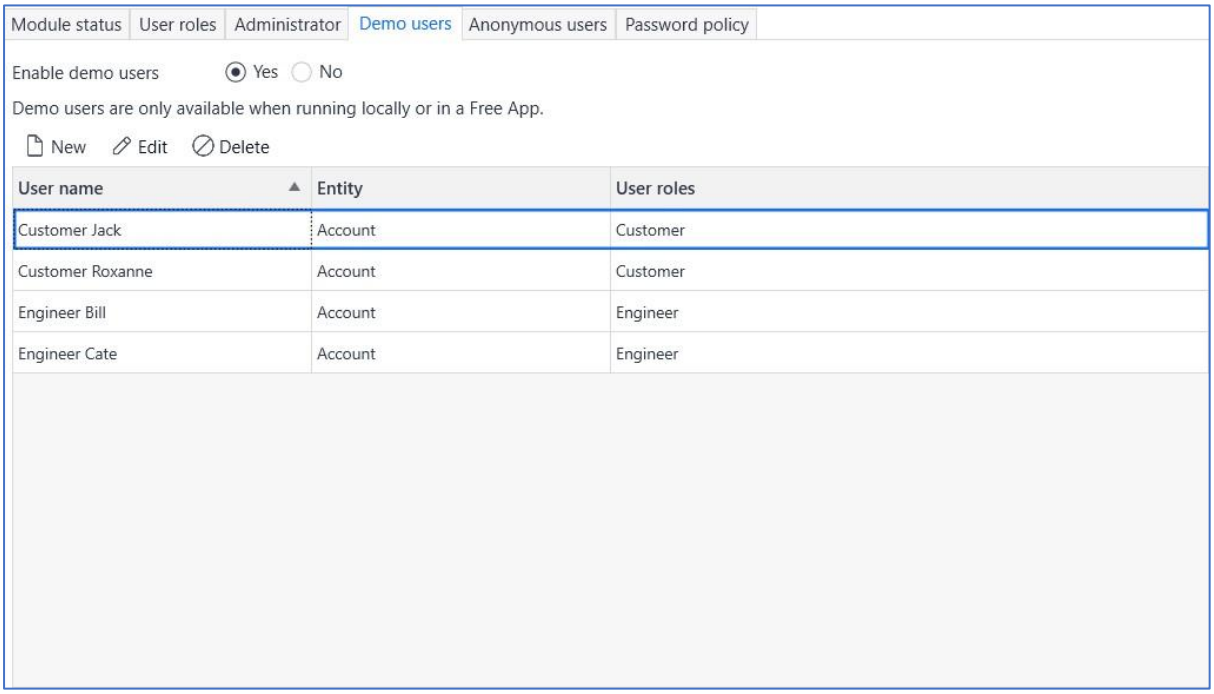


Fig. 2: Postman for API Testing

B. Authentication Security Testing

1. Login Brute Force

- **Goal:** Assess resistance against brute-force login attempts (lockout, rate-limiting, CAPTCHA).
- **Tool:** OWASP ZAP Fuzzer

Steps:

1. **Fuzzing Login Endpoint**
    - Intercepted login request, selected password parameter, applied common password wordlist.

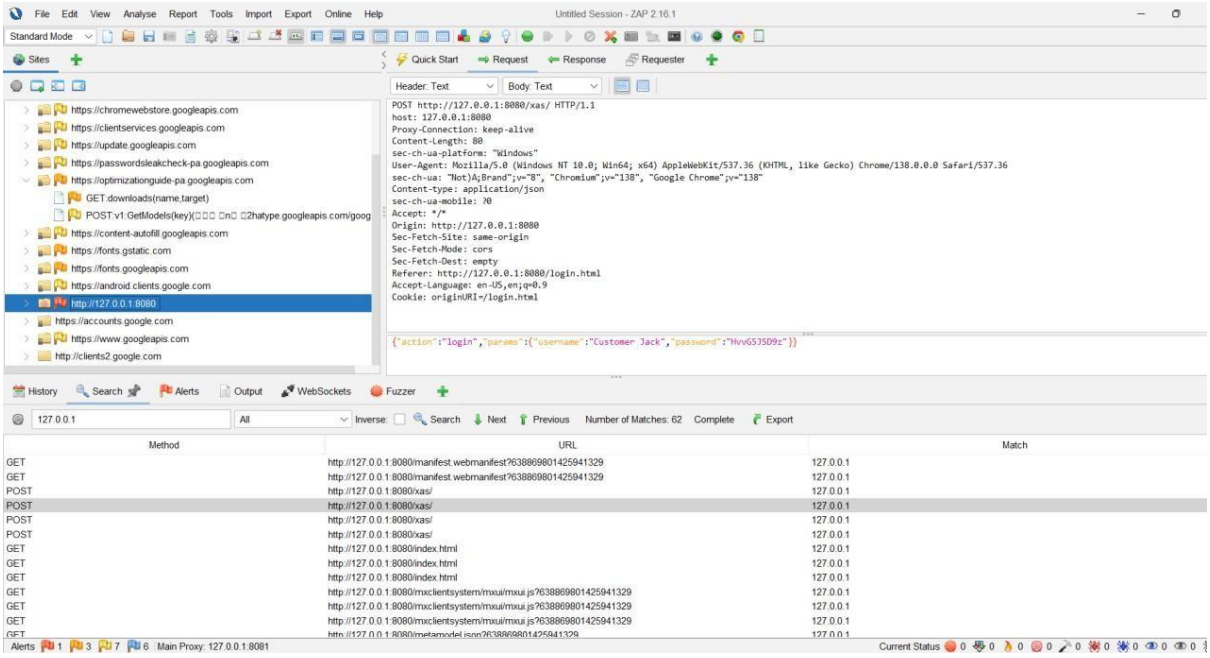


Fig. 3: Fuzz Attack Performed

2. **Testing with Valid Credentials**
    - Sent correct username/password.

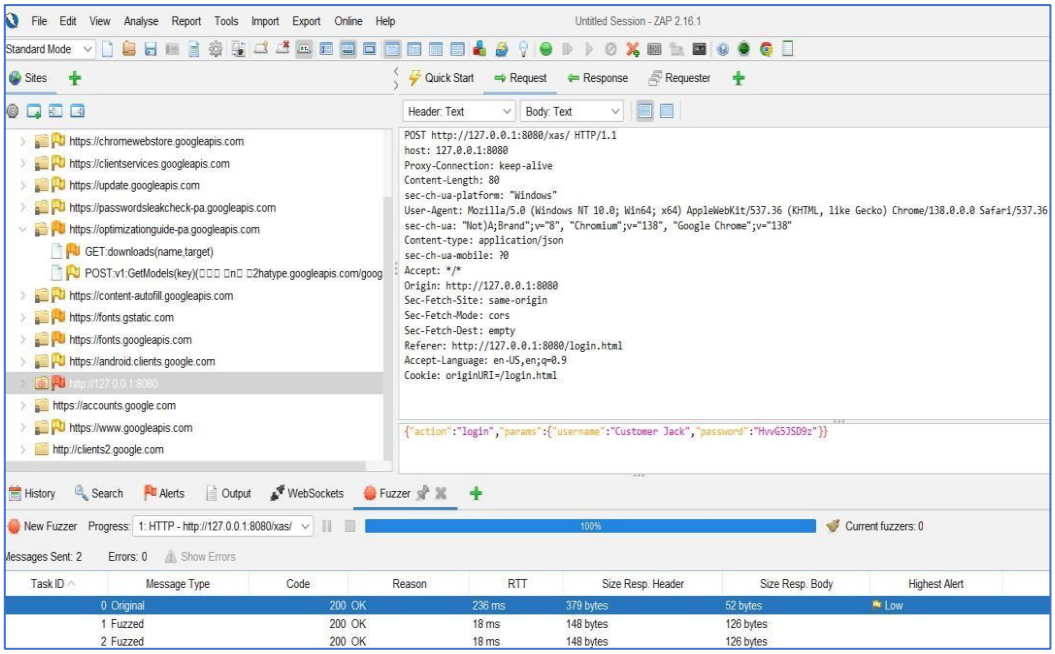


Fig. 4: Valid Login Request

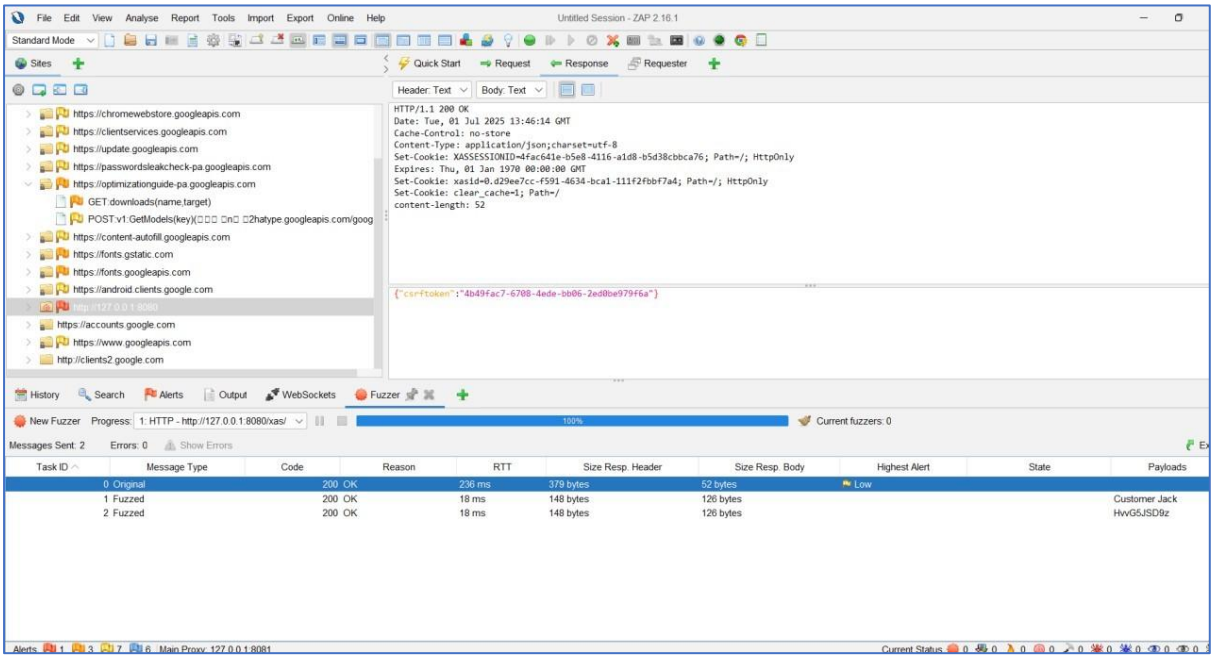


Fig. 5: 200 OK with CSRF Token

3. **Testing with Invalid Credentials**
    - Sent incorrect password.

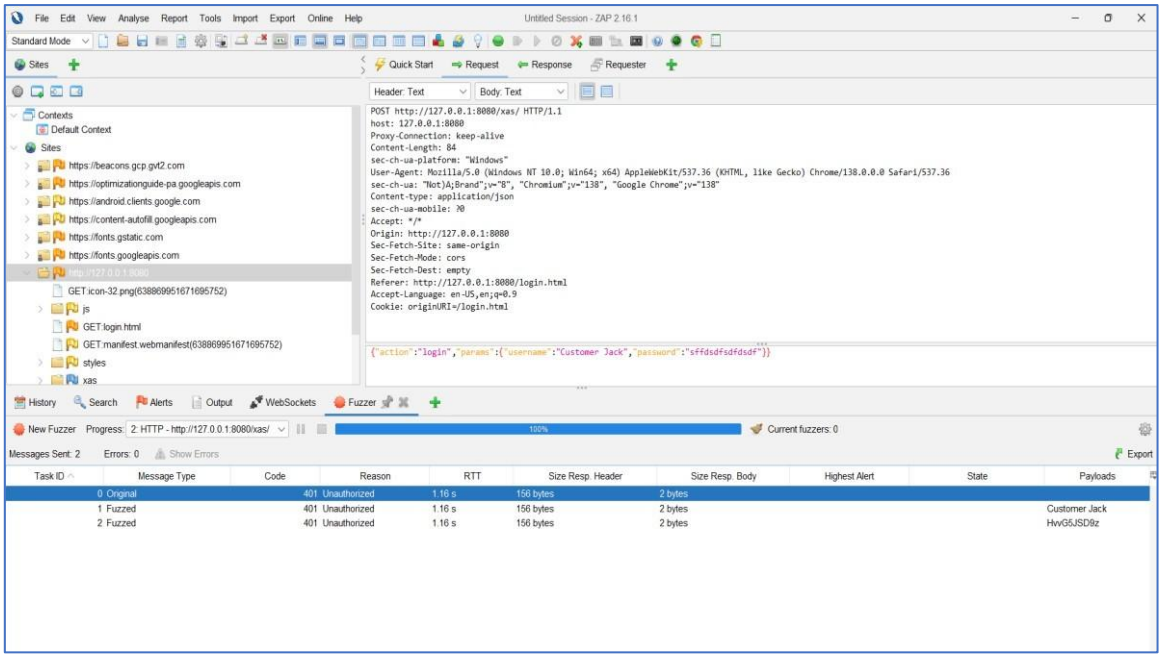


Fig. 6: Invalid Login Request

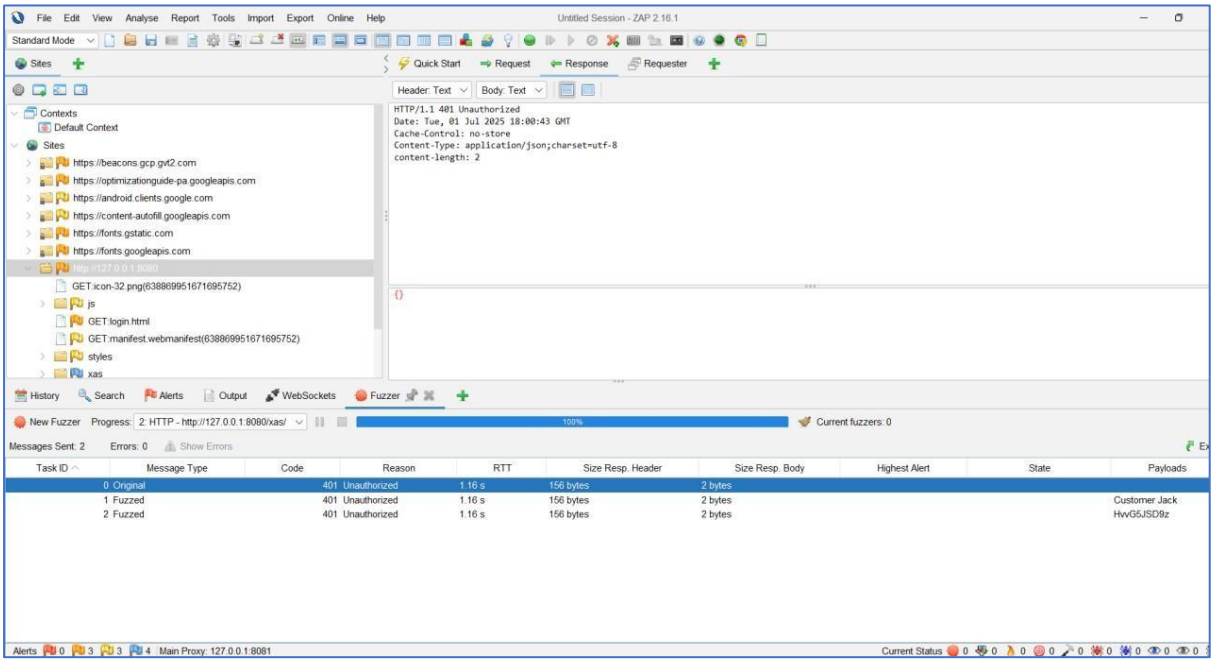


Fig. 7: 401 Unauthorized Response

Results:

- Valid credentials returned 200 OK; invalid returned 401 Unauthorized.
- No account lockout, CAPTCHA, or throttling detected.

Impact:

- Vulnerable to brute-force and credential stuffing attacks.

Recommendation:

- Implement account lockout, rate-limiting, or CAPTCHA after multiple failed attempts.

2. Session Fixation / Reuse

- **Objective:** Determine if stolen or reused session cookies allow unauthorized access.

Steps:

1. Extracted session cookie (XASSESSIONID) after valid login

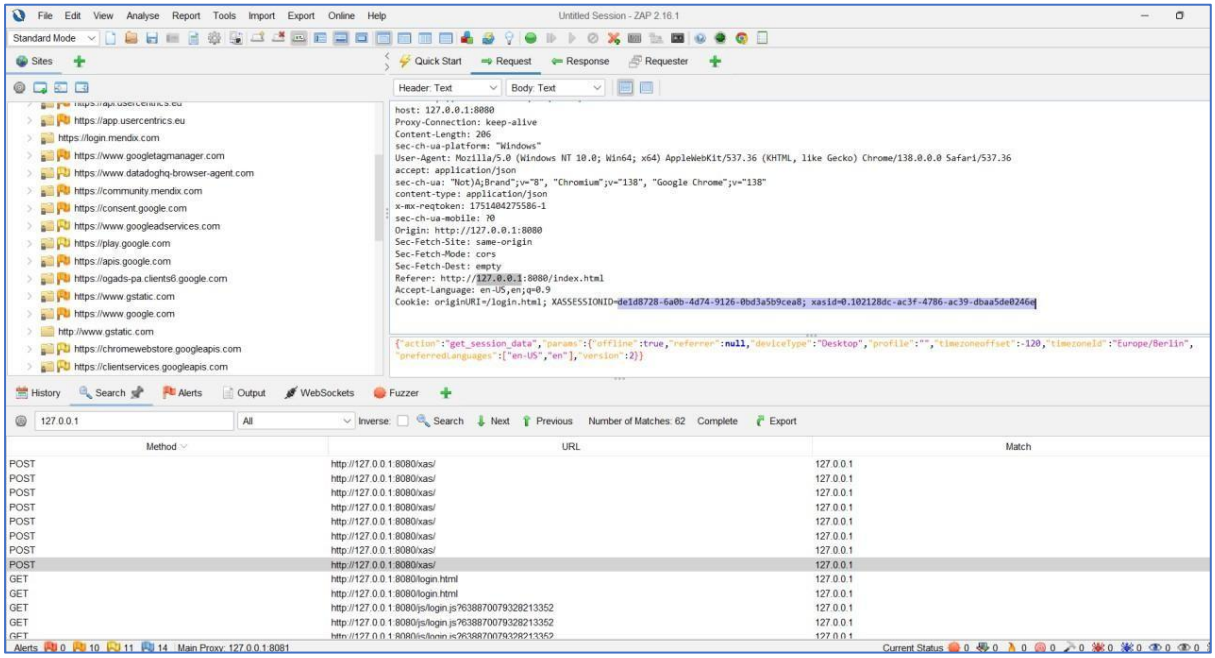


Fig. 8: Extracted Session Cookie

2. Inserted cookie in new browser session using Chrome DevTools.

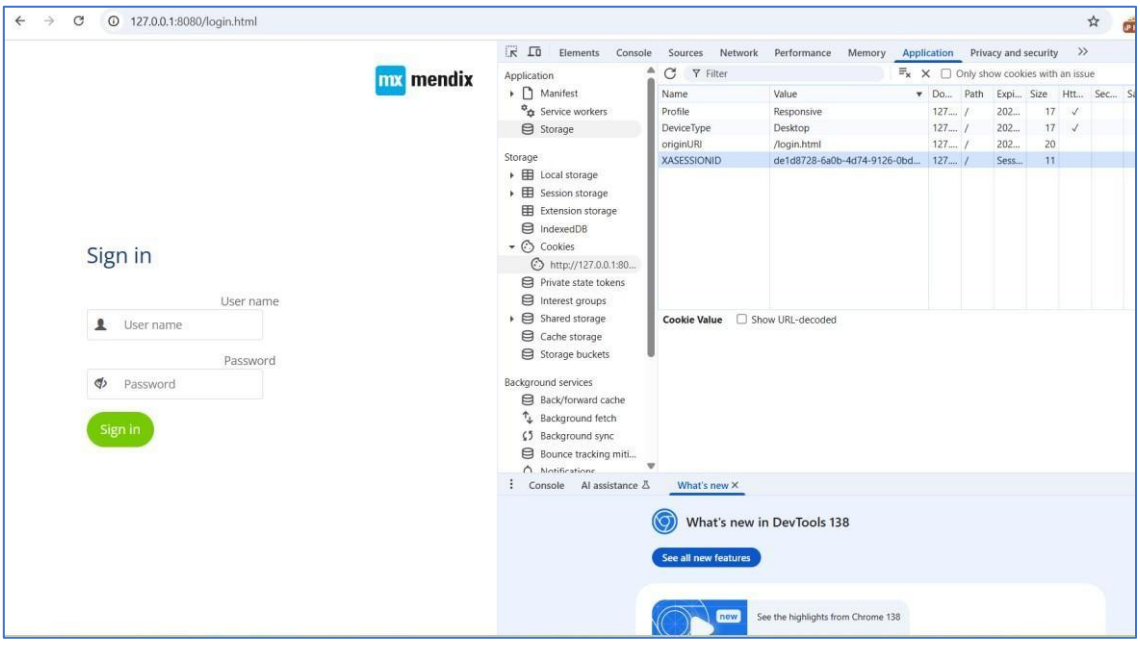


Fig. 9: Setting Session Cookie

3. Attempted to access protected resources.

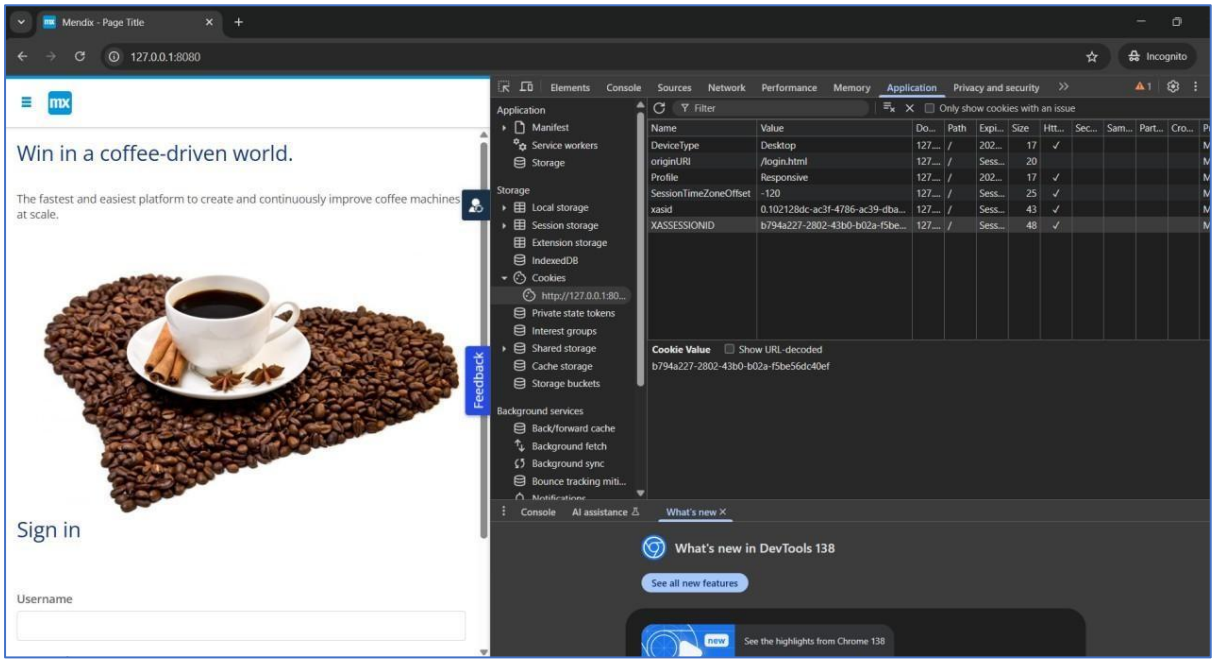


Fig. 10: Access Attempt with Fixed Session

Results:

- Reused cookies were rejected; redirected to login page.

Conclusion:

- Session fixation and reuse mitigations are effectively implemented.

3. Authentication Bypass

- **Objective:** Test if protected resources are accessible without authentication.
- **Tool:** OWASP ZAP Spider

Steps:

1. Ran unauthenticated spidering.

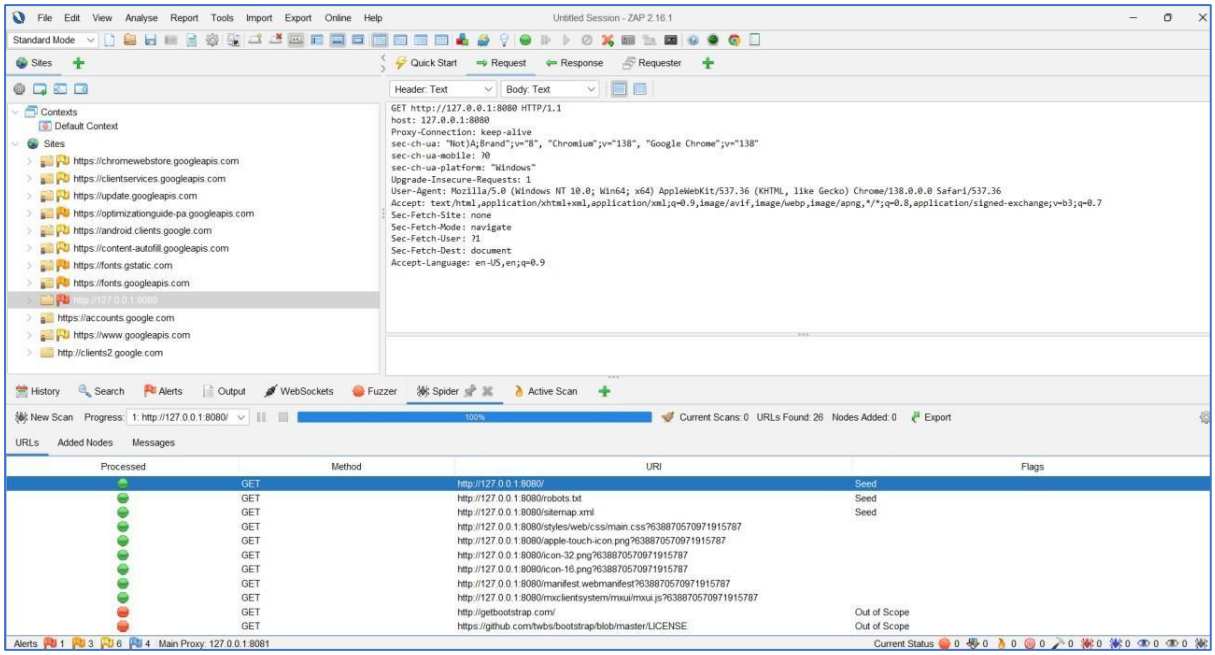


Fig. 11: ZAP Spider Results

2. Attempted direct access to protected pages.

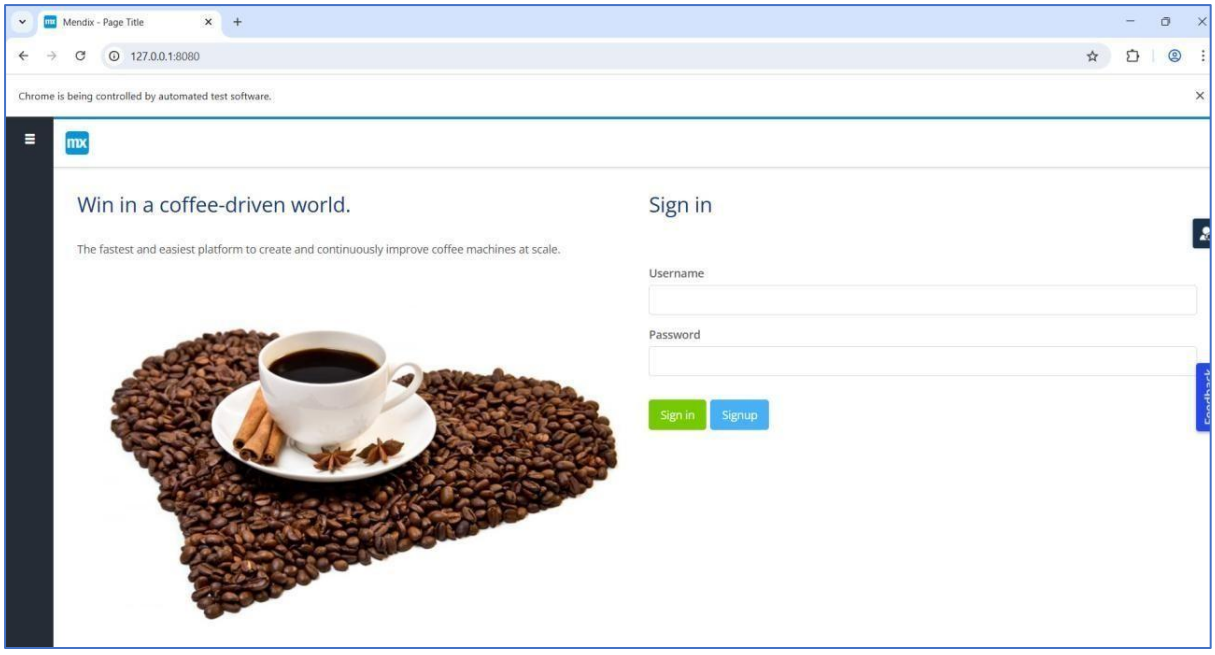


Fig. 12: HTTP Response for Protected Page

Results:

- Only static/public files (/robots.txt, /sitemap.xml) exposed.
- No protected routes accessible without authentication.

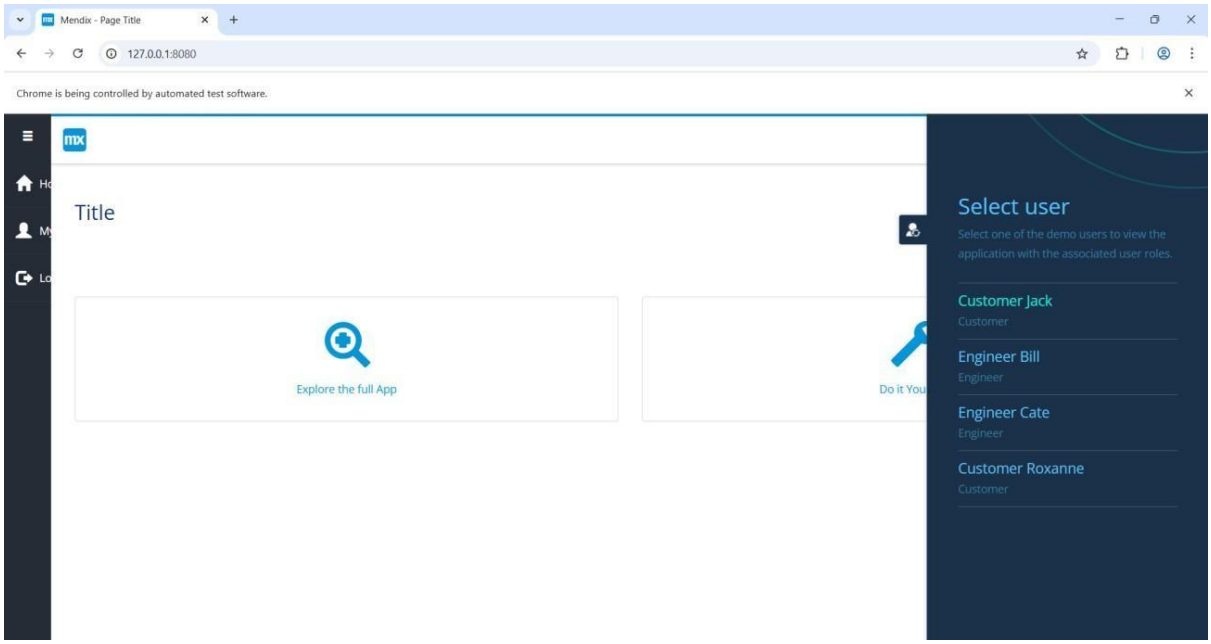
C. Role-Based Access Control (RBAC) and IDOR

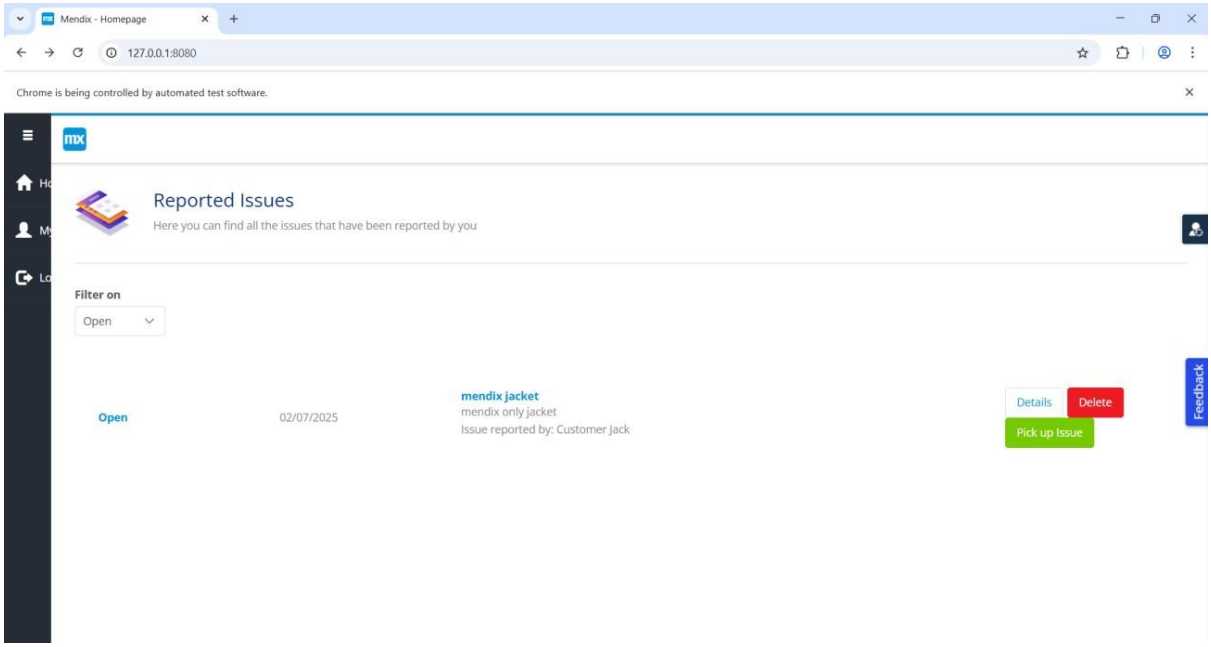
1. RBAC Testing

- **Objective:** Validate access segregation among Requester, Approver, and Admin roles.

Steps:

1. Logged in as each role and recorded accessible URLs.


Mendix - Page Title
127.0.0.1:8080
Chrome is being controlled by automated test software.
Title
Explore the full App
Select user
Select one of the demo users to view the application with the associated user roles.
Customer Jack
Customer
Engineer Bill
Engineer
Engineer Cate
Engineer
Customer Roxanne
Customer


Mendix - Homepage
127.0.0.1:8080
Chrome is being controlled by automated test software.
Reported Issues
Here you can find all the issues that have been reported by you
Filter on
Open
Open
02/07/2025
mendix jacket
mendix only jacket
Issue reported by: Customer Jack
Details
Delete
Pick up Issue
Feedback

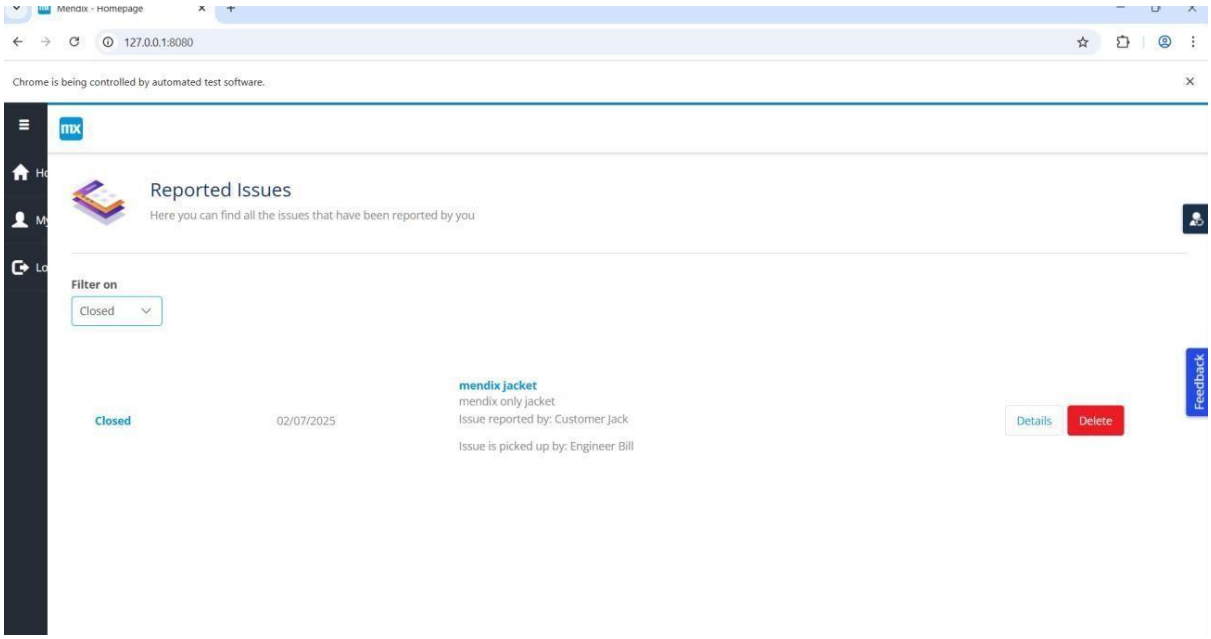


Fig. 13: Customer Jack Dashboard, Customer Jack Task, New Task Added – Customer Jack

2. Attempted cross-role access using lower-privilege accounts.

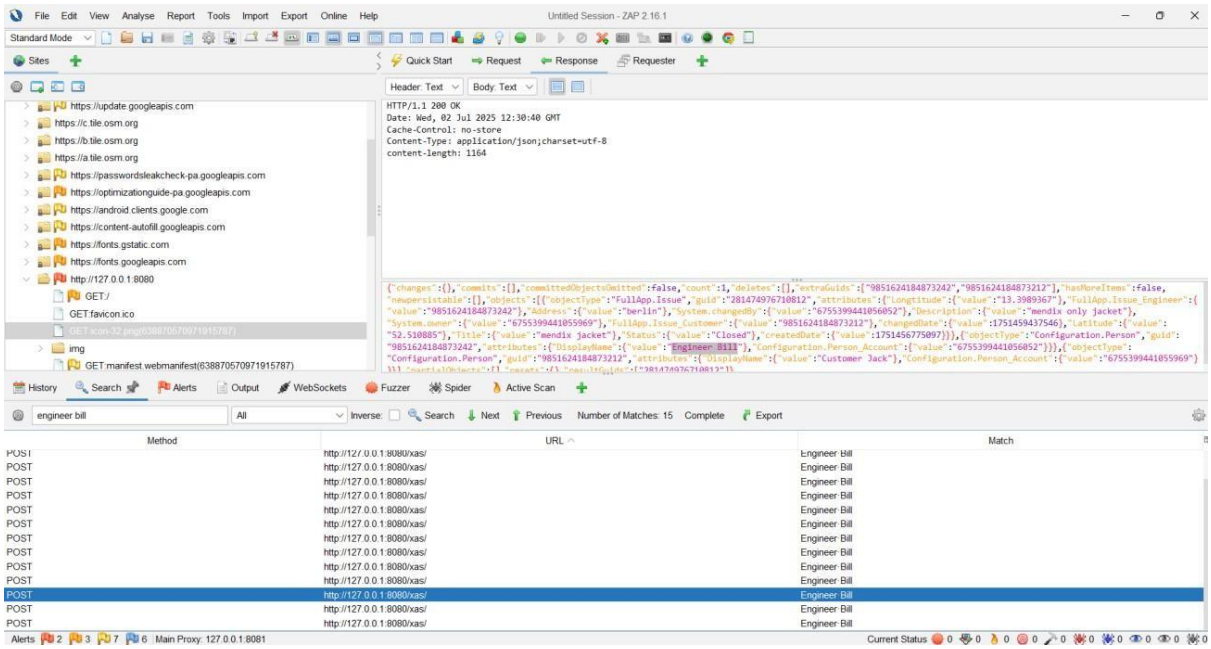


Fig.14: Attempt to Access Engineer Bill

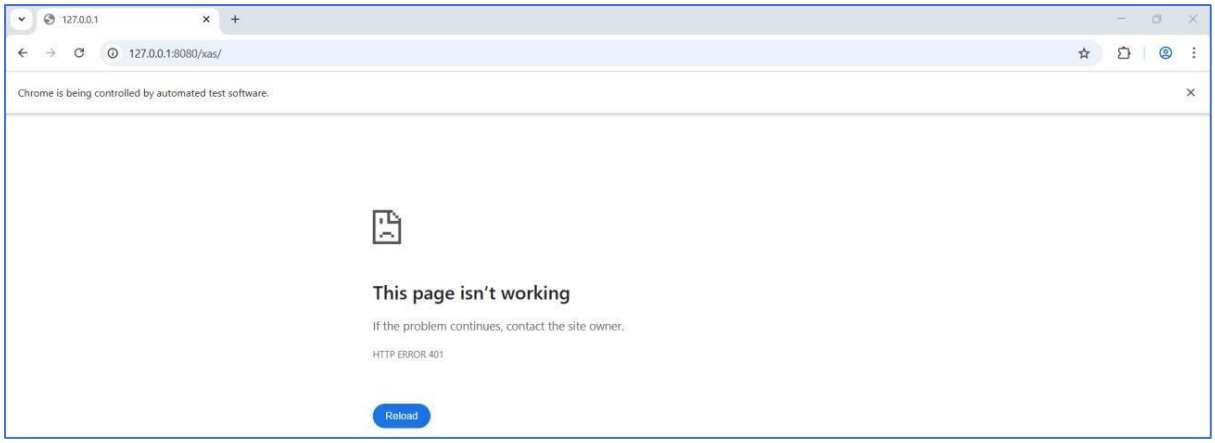


Fig. 15: Access Denied Response

Results:

- Access denied for unauthorized roles.
- No RBAC flaws detected.

2. IDOR (Insecure Direct Object Reference)

- **Objective:** Test for unauthorized data access via manipulated IDs.

Steps:

1. Identified profiledata parameter.
2. Fuzzed values (21–900) via ZAP.
3. Reviewed responses.

Results:

- All responses returned empty JSON ({}) with no sensitive data exposed.

Conclusion:

- No IDOR vulnerabilities identified.

D. API Security and Input Validation

1. Unauthenticated API Access

- **Objective:** Test if /xas/ endpoint allows unauthenticated interaction.

Key Findings:

- API returned 200 OK with CSRF token even without authentication cookies.

Risk:

- Potential unauthorized API access if CSRF token alone is sufficient.

2. Injection Testing

- **Objective:** Identify SQLi, XSS, or Command Injection vulnerabilities.

Process:

- Active scan performed on /xas/.

Results:

- No SQLi, XSS, or Command Injection detected.
- **Issue Identified:** Bcrypt password hashes exposed in API responses.

E. CSRF Testing

- **Objective:** Verify anti-CSRF protection across roles and transactions.

Process:

1. Passive scan during navigation. *(Fig. 1–4: Role-based Navigation and Action Pages)*
2. Active scan for missing CSRF tokens. *(Fig. 5: Active Scan Results)*

Results:

- No CSRF vulnerabilities detected.

F. Other Security Issues & Information Disclosure

High Risk

1. **Hash Disclosure (BCrypt)** – Hashes exposed in POST /xas/.
2. **PII Disclosure** – Credit card number leaked via POST /xas/.

Medium Risk

- Missing CSP header.
- Directory browsing enabled.
- Missing anti-clickjacking header.
- Spring Actuator /xas/actuator/health exposed.

Low Risk

- Missing HttpOnly & SameSite flags.
- HSTS not enabled.
- Missing X-Content-Type-Options.
- Timestamp leaks.

- Debug error messages exposed (/metamodel.json).

G. Summary Table

Severity	*Issue*	*Endpoint(s)*	*Recommendation*
High	Hash Disclosure (BCrypt)	POST /xas/	Remove hashes from API responses
High	PII Disclosure	POST /xas/	Mask/remove sensitive data
Medium	CSP Header Not Set	/, /xas/	Add Content-Security-Policy header
Medium	Directory Browsing Enabled	/widgets/...	Disable directory listing
Medium	Missing Anti-clickjacking	All responses	Add X-Frame-Options/CSP headers
Medium	Spring Actuator Info Leak	/xas/actuator/health	Restrict or disable in production
Low	Cookie Flags Missing	All cookies	Set HttpOnly, SameSite
Low	HSTS Not Set	All endpoints	Enable HSTS
Low	X-Content-Type-Options Missing	Multiple endpoints	Add nosniff header
Low	Debug/Info Disclosure	/metamodel.json	Suppress debug outputs

H. Key Recommendations

- Remove sensitive data (hashes, PII) from API responses.
- Add missing headers: CSP, HSTS, X-Frame-Options.
- Harden cookies with HttpOnly and SameSite.
- Restrict debug and actuator endpoints in production.

Active Scan for Coffee Service App

Sites: https://chromewebstore.googleapis.com https://clientservices.googleapis.com https://update.googleapis.com https://c.tile.osm.org https://b.tile.osm.org https://a.tile.osm.org https://optimizationguide-pa.googleapis.com https://android.clients.google.com https://fonts.gstatic.com https://fonts.googleapis.com https://www.googleapis.com https://accounts.google.com http://clients2.google.com https://content-autofill.googleapis.com https://passwordsleakcheck-pa.googleapis.com http://127.0.0.1:8080

Generated on Thu, 3 Jul 2025 23:12:12

ZAP Version: 2.16.1

ZAP by Checkmarx

Summary of Alerts

Risk Level	Number of Alerts
High	2
Medium	5
Low	7

Alerts

Name	Risk Level	Number of Instances
Hash Disclosure - BCrypt	High	6
PII Disclosure	High	1
Content Security Policy (CSP) Header Not Set	Medium	10
Cross-Domain Misconfiguration	Medium	2
Directory Browsing	Medium	1
Missing Anti-clickjacking Header	Medium	10
Spring Actuator Information Leak	Medium	1
Cookie No HttpOnly Flag	Low	1
Cookie without SameSite Attribute	Low	7
Information Disclosure - Debug Error Messages	Low	1
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	1
Strict-Transport-Security Header Not Set	Low	22

Timestamp Disclosure - Unix	Low	8
X-Content-Type-Options Header Missing	Low	29
Authentication Request Identified	Informational	4
Information Disclosure - Information in Browser sessionStorage	Informational	1
Information Disclosure - Suspicious Comments	Informational	3
Modern Web Application	Informational	1
Re-examine Cache-control Directives	Informational	12
Retrieved from Cache	Informational	1
Session Management Response Identified	Informational	1
User Agent Fuzzer	Informational	24

Dynamic Application Security Testing (DAST) – Purchase Request Application

A. Tools and Setup

- **Tool:** OWASP ZAP (Zed Attack Proxy)
- **Proxy:** Configured for login and API traffic interception
- **Test Users:** demo_requester, demo_approver, demo_admin

- **Environment:** Pre-production instance
- **Objective:** Identify authentication, session, and access control vulnerabilities before production deployment

B. Authentication Security

1. Brute-Force Login Test

Objective: Evaluate resistance against automated password guessing attacks.

Test Environment:

- User: demo_requester
- Endpoint: /login (application login page/API)
- Tool: OWASP ZAP Fuzzer

Steps:

1. **Capture Baseline Login Request**
    - Intercepted HTTP POST login request using ZAP.
    - Identified username/password fields for fuzzing.
2. **Configure and Launch Fuzzer**
    - Loaded password wordlist and initiated attack.
3. **Monitor Responses**
    - Tracked status codes for each attempt.
4. **Test Correct vs. Incorrect Passwords**
    - Incorrect password returned error.
    - Correct password succeeded with CSRF token issued.
5. **Evaluate Lockout Mechanisms**
    - Performed 20–50+ attempts.

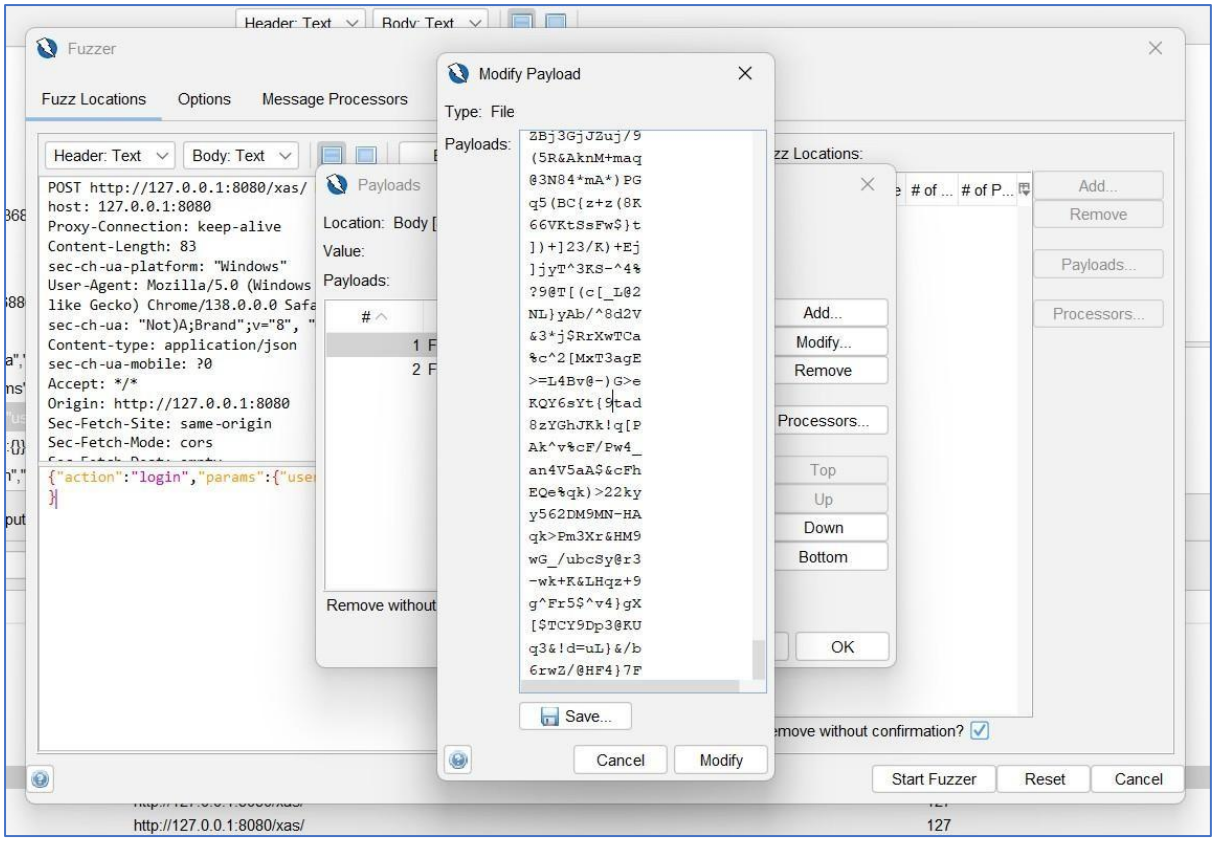


Fig. 16: No Lockout Observed

Results:

- Only valid credentials allowed login.
- No account lockout, CAPTCHA, or rate-limiting implemented.

Impact: High brute-force risk.

Recommendations:

- Implement account lockout or temporary bans after repeated failures.
- Add CAPTCHA after multiple failed attempts.
- Introduce rate-limiting (e.g., exponential backoff).

2. Session Reuse / Fixation

Objective: Test if stolen session cookies can be reused across users.

Steps:

1. Logged in as demo_requester and extracted session cookie.
2. Logged in as demo_approver in a separate browser, replaced cookie.
3. Attempted access with manipulated session.

Results:

- Session tokens cannot be reused between users.
- No session fixation vulnerability found.

Recommendation: Maintain current session management controls and review periodically.

3. Login Bypass

Objective: Determine if unauthenticated users can access protected resources.

Steps:

1. Cleared cookies; initiated anonymous access.
2. Ran OWASP ZAP Spider to enumerate endpoints.

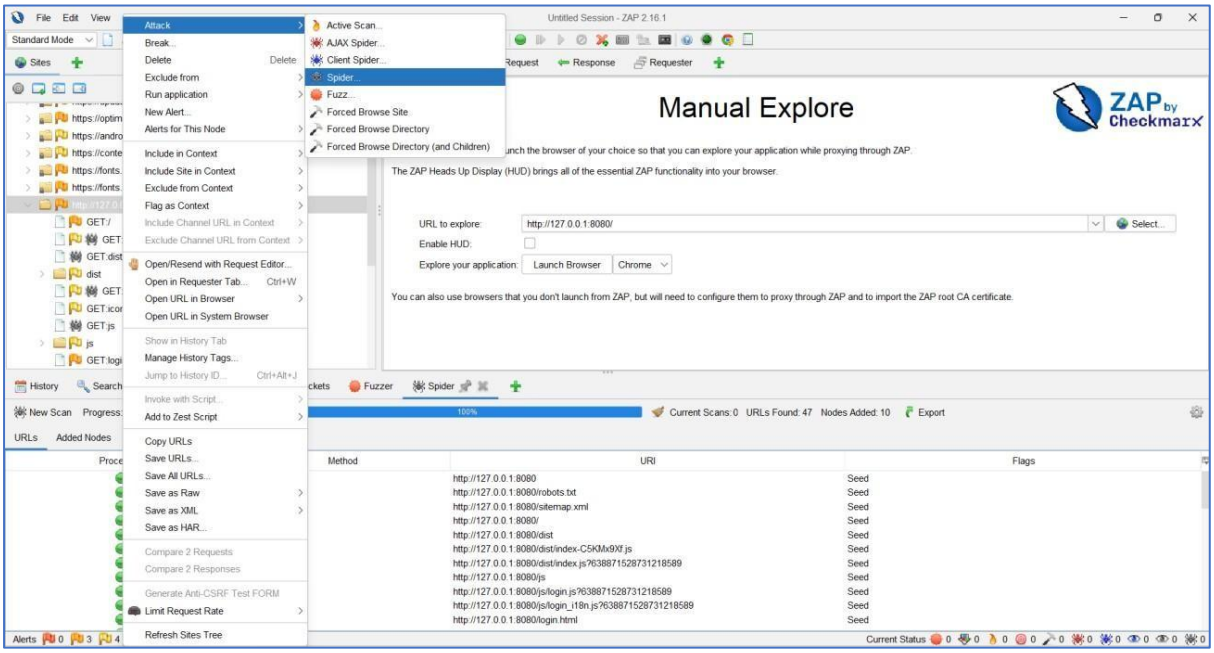


Fig. 17: Spider in Progress

3. Reviewed discovered URLs

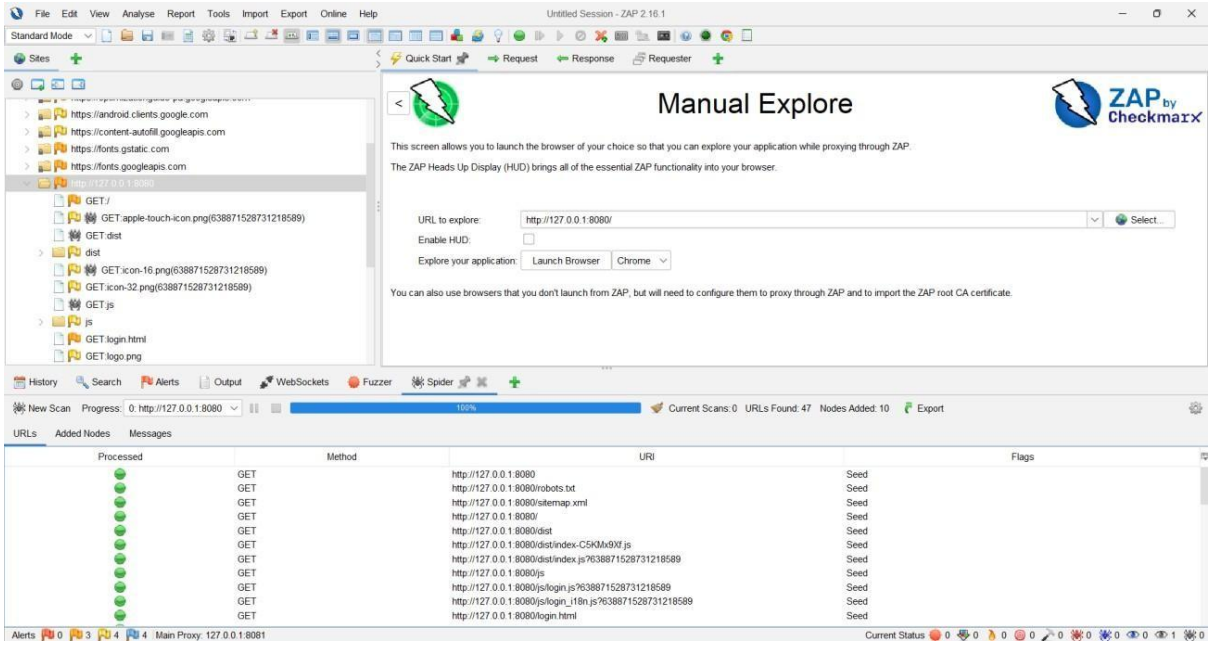


Fig. 18: Spider Results

4. Attempted direct access to restricted pages.

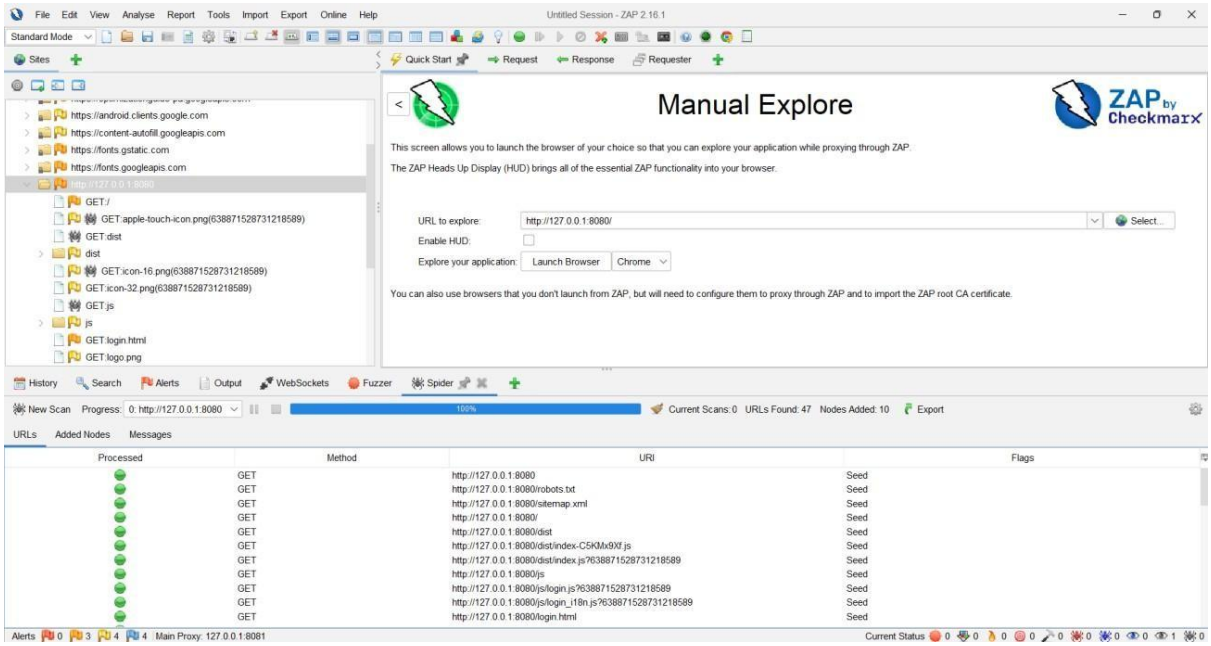


Fig. 19: Access Denied

Results:

- Only public resources accessible: landing page, login page, static files.
- All protected pages redirected to login or returned 401/403.

Conclusion: No login bypass vulnerabilities detected.

C. Access Control Testing (RBAC & IDOR)

1. Role-Based Access Control (RBAC)

Objective: Verify access segregation across Requester, Approver, and Admin roles.

Steps:

1. Logged in with each role individually.
2. Ran OWASP ZAP Spider for each session.
3. Compared accessible URLs per role.
4. Tested unauthorized access attempts.

Results Table – RBAC Accessibility

URL / Endpoint	*Requester*	*Approver*	*Admin*	*Description*
/	Yes	Yes	Yes	Home/Dashboard
/login.html	Yes	Yes	Yes	Login Page
/xas/	Yes	Yes	Yes	API Endpoint
/PurchaseApprovals.Admin_Homepage.js	No	No	Yes	Admin Homepage
/PurchaseApprovals.Product_Overview.js	Yes	Yes	Yes	Product Overview
/PurchaseApprovals.TaskInbox.js	No	Yes	No	Approver Inbox
/PurchaseApprovals.Task_Approver.js	No	Yes	No	Approver Task Page

Findings:

- Requesters and approvers restricted to their respective areas.
- Admin has full access.
- Unauthorized access attempts returned 403 or error page.

Recommendation: Maintain current RBAC configuration; review when adding new features.

2. Insecure Direct Object Reference (IDOR)

Objective: Identify unauthorized resource access via manipulated IDs.

Steps:

1. Located API requests with object IDs.
2. Tampered ID in request payload.
3. Observed server response.

Results:

- Unauthorized requests returned 401.
- No data leakage detected.

Conclusion: Application is not vulnerable to IDOR in tested endpoints.

D. Key Findings

- **High Risk:** No brute-force mitigation on login.
- **Medium Risk:** None observed in RBAC or IDOR.
- **Low Risk:** General best practices (e.g., periodic session review).

E. Recommendations

- Enforce brute-force protection: account lockout, CAPTCHA, and rate-limiting.
- Continue periodic review of session and RBAC controls.
- Retest after major feature changes or production deployment.

Dynamic Application Security Testing (DAST) – Comprehensive Security Report

1. Security Analysis – IDOR (Insecure Direct Object Reference)

Positive Findings

- The application enforces proper access controls; users cannot access or modify resources they do not own by manipulating object identifiers.

Negative Findings

- No IDOR vulnerabilities were detected at the tested endpoints.

Conclusion & Recommendation

- The tested endpoints correctly enforced authorization controls, returning 401 Unauthorized when unauthorized access was attempted.

- **Recommendation:** Maintain strong access controls across all endpoints and **periodically retest** for IDOR vulnerabilities, especially after adding new features or APIs.

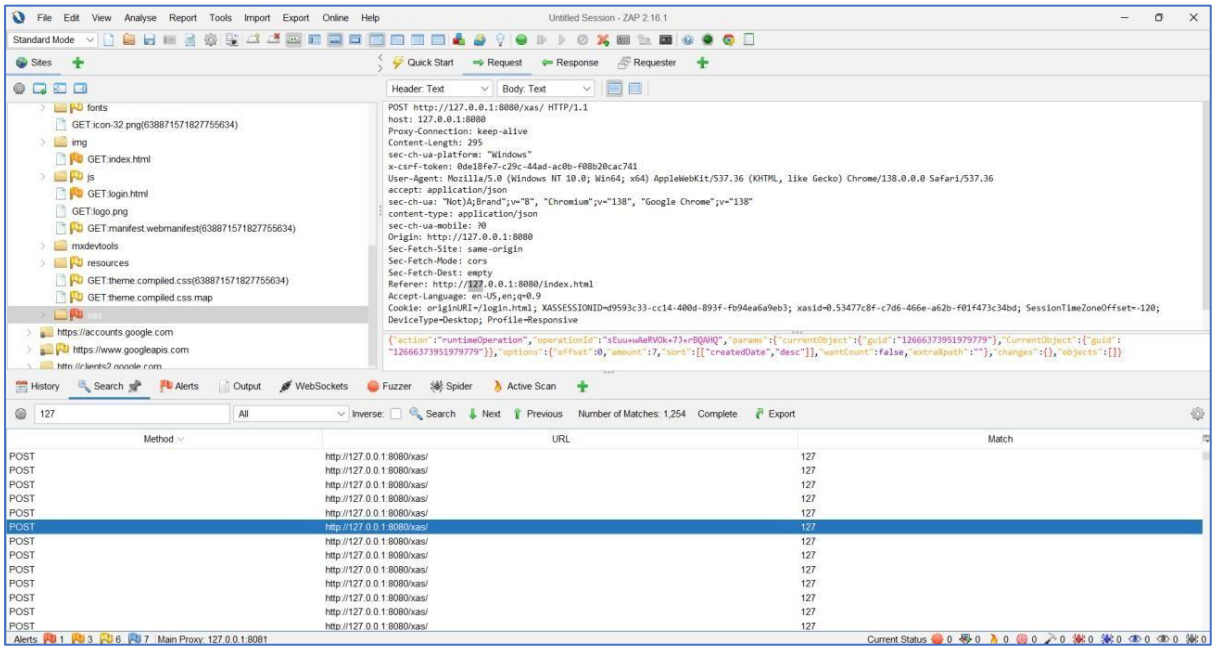
2. API Security & Input Validation

2.1 Unauthenticated API Access (OWASP A01: Broken Access Control)

Objective:
To determine whether sensitive API endpoints (e.g., /xas/) can be accessed without proper authentication.

Methodology:

1. **Intercept Valid Request**
    - Logged in as demo_requester and captured a POST /xas/ request using OWASP ZAP.



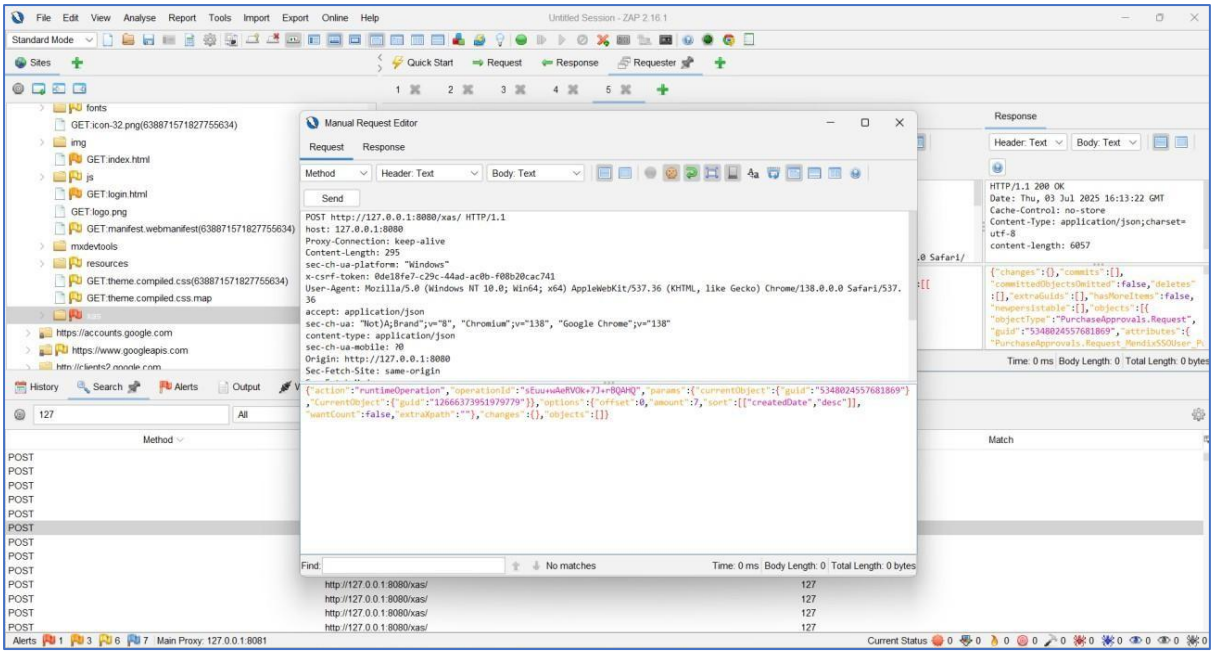


Fig. 20: Logged-in session and Changed the GUID in Request

2. **Remove Authentication Headers**
    - Cleared all cookies and tokens in ZAP's Manual Request Editor.
3. **Send Unauthenticated Request**
    - Sent modified request to the server without authentication details.
4. **Analyze Response**
    - Observed 200 OK response with a valid CSRF token.

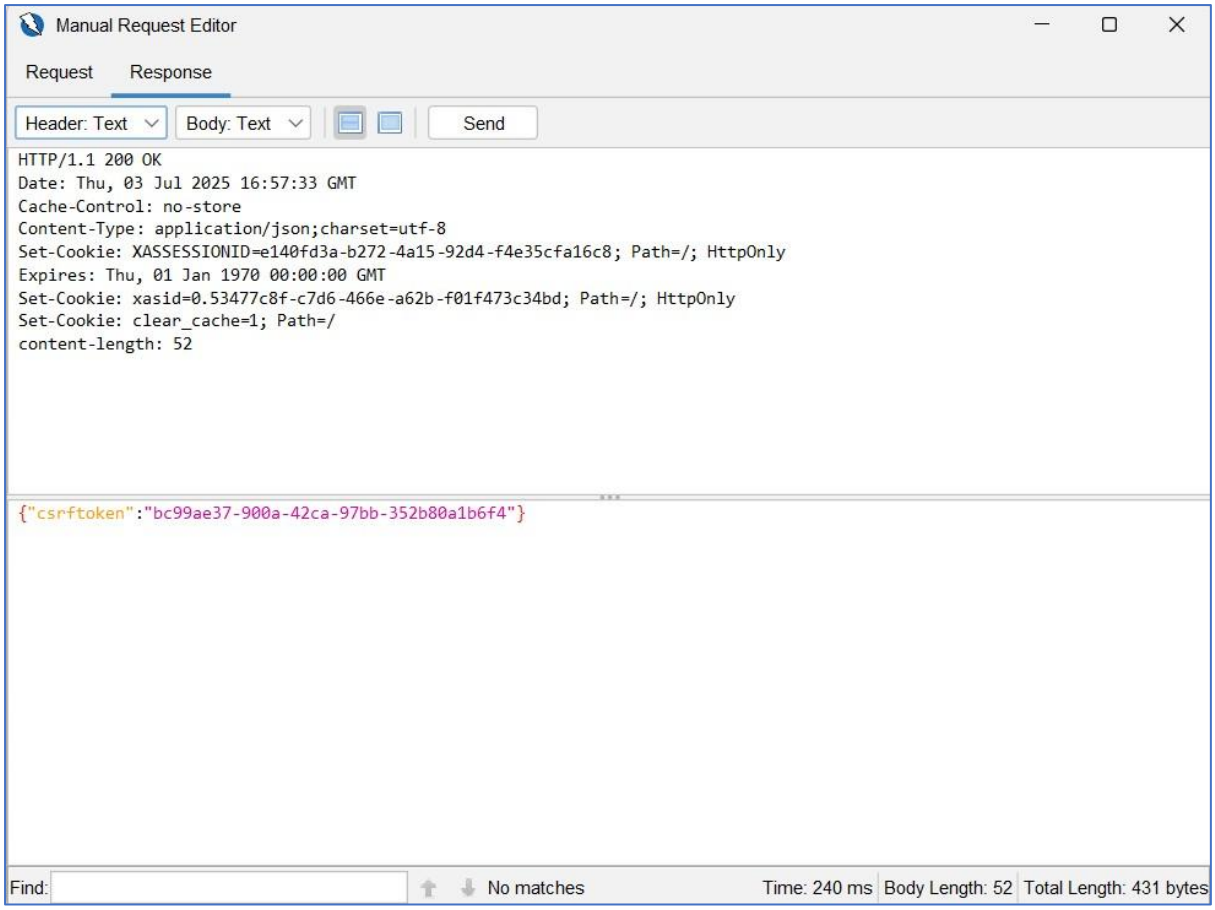


Fig. 21: Exposed CSRF token

Results:

- /xas/ endpoint was accessible without authentication.
- Expected behavior: 401 Unauthorized or 403 Forbidden.
- Actual behavior: **200 OK with CSRF token exposed**.

Severity: Critical (OWASP A01)

Impact:

- Allows unauthenticated attackers to retrieve sensitive tokens.

Recommendation:

- Enforce authentication and authorization checks for all sensitive API endpoints.
- Ensure unauthenticated requests receive 401/403 and do not expose CSRF tokens.

Summary Table:

Endpoint	Method	Response Code	Access Granted?	Severity	Figure

/xas/	POST	200	Yes	Critical	Fig. 3–4

2.2 Injection Vulnerabilities (OWASP A03: Injection)

Objective:

To identify potential SQL Injection, XSS, or Command Injection vulnerabilities via OWASP ZAP Active Scan.

Methodology:

1. **Target Identification**
    - Proxied authenticated traffic to locate key endpoints.
2. **Active Scan Execution**
    - Ran ZAP Active Scan with payloads for SQLi, XSS, Command Injection.

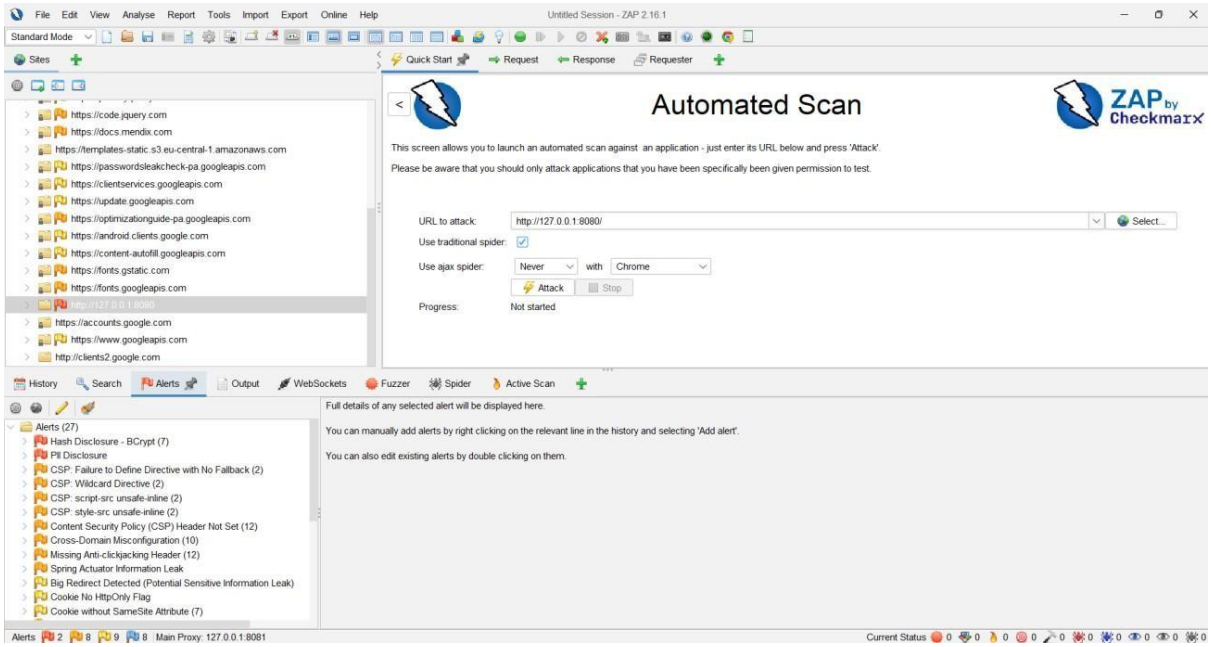


Fig. 21: ZAP Active Scan Progress Window

Shows the progress and configuration of the ZAP Active Scan, confirming coverage of all selected endpoints

3. **Review Findings**
    - Examined alerts for evidence of injection vulnerabilities.

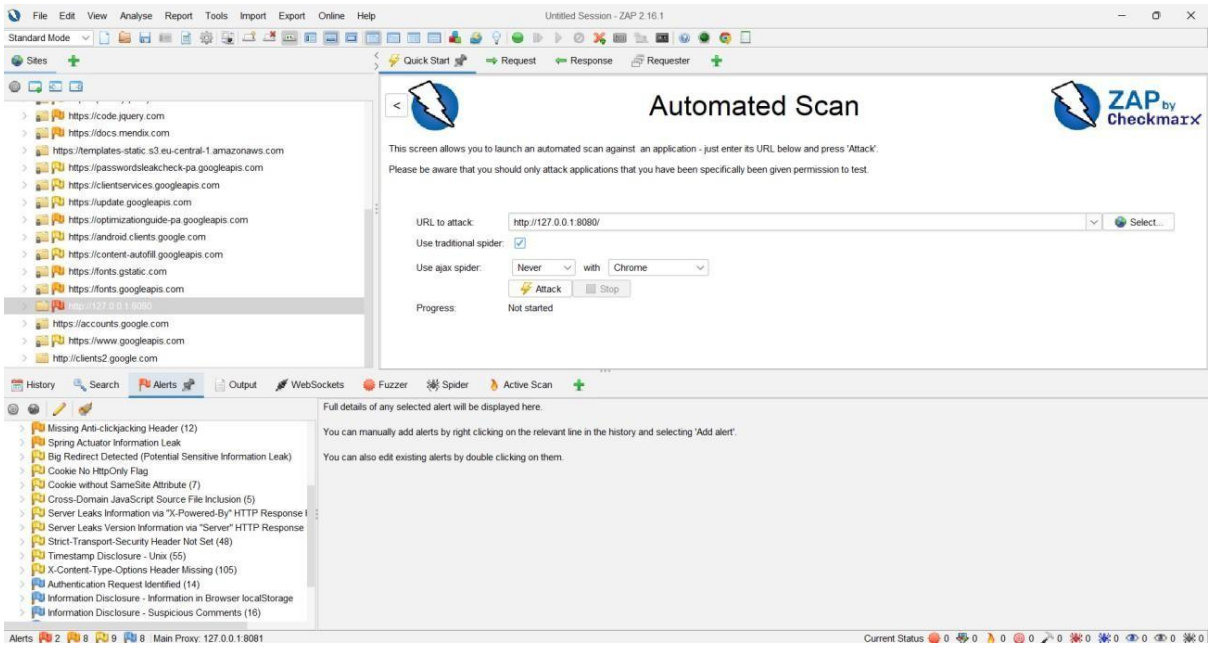


Fig. 22: No injection detected

Results:

- No SQLi, XSS, or command injection vulnerabilities detected.
- Input fields properly sanitized or blocked.

Recommendation:

- Maintain strict input validation.
- Retest after code changes or new features.

3. Cross-Site Request Forgery (CSRF) Testing (OWASP A05:2021)

Objective:
To verify presence of CSRF protections (anti-CSRF tokens) for all POST and state-changing requests.

Methodology:

1. **Setup & Passive Scan**
    - Logged in, proxied traffic via ZAP, and passively scanned for CSRF patterns.
2. **Manual Review**
    - Inspected POST requests for CSRF tokens.
3. **Analysis of POST Requests**
    - Intercepted /xas/ POST request lacking anti-CSRF token.

Results:

- No anti-CSRF tokens found for sensitive endpoints.

Severity: Critical (OWASP A05)

Impact:

- Attackers could forge authenticated user actions.

Recommendation:

- Enforce session-specific CSRF tokens for all POST/PUT/DELETE requests.
- Validate tokens server-side for every request.

CSRF Analysis Table:

Method	URL	CSRF Token Present?	Comment
POST	http://127.0.0.1:8080/xas/	No	Vulnerable to CSRF

4. Other Security Issues & Information Disclosure

High Risk

- **Hash Disclosure – BCrypt**: Password hashes exposed at POST /xas/ (7 instances).
- **Impact:** Offline cracking risk.
- **Recommendation:** Never return password hashes in API responses.

Medium Risk

1. Missing Content Security Policy (CSP) headers (/, /index.html, /login.html).
2. Missing anti-clickjacking headers (X-Frame-Options, CSP frame-ancestors).
3. Spring Actuator health endpoint exposed (/xas/actuator/health).

Low Risk

- Missing HttpOnly & SameSite cookie flags.
- Missing HSTS headers.
- Missing X-Content-Type-Options: nosniff.
- Timestamp leakage in JS files.

Summary Table:

Severity	Vulnerability	Affected Endpoint(s)	Remediation
High	Hash Disclosure (BCrypt)	POST /xas/	Remove hashes from responses

Medium	Missing CSP Header	/, /index.html, /login.html	Add Content-Security-Policy header
Medium	Anti-clickjacking Header	Main pages	Add X-Frame-Options or CSP
Medium	Spring Actuator Leak	/xas/actuator/health	Restrict/disable actuator endpoints
Low	Cookie Security Flags	Multiple cookies	Set HttpOnly & SameSite flags
Low	HSTS Header Missing	All endpoints	Add HSTS
Low	X-Content-Type-Options	Most responses	Add nosniff header
Low	Timestamp Disclosure	/dist/*.js	Remove internal timestamps

5. Conclusion & Recommendations

The assessment revealed:

- **Critical vulnerabilities:** Unauthenticated API access, missing CSRF tokens, and password hash exposure.
- **Medium risks:** Missing security headers, actuator leaks.
- **Low risks:** Cookie misconfigurations and information disclosures.

Immediate Actions:

- Enforce authentication and CSRF protections.
- Remove sensitive data from API responses.
- Apply missing security headers.
- Restrict internal endpoints.

Penetration Testing & Vulnerability Assessment Report – Task Tracker

2.1 Role-Based Access Control (RBAC) Testing

Objective

Assess whether the application enforces proper RBAC for Member and Manager roles, ensuring only authorized resources and features are accessible (OWASP A01: Broken Access Control).

Methodology

1. Logged in as **Member1**; explored UI and captured accessible features.
2. Ran **OWASP ZAP Spider** for Member1 to enumerate URLs.
3. Compiled list of accessible endpoints.
4. Repeated the process for **Manager1**.
5. Compared accessible URLs between both roles.

Results

- Both Member1 and Manager1 had **identical access** to all major pages, including:
    - TaskOverview, TaskEdit, TeamOverview, MyAccountViewEdit
    - /xas/ API endpoint
    - All static assets and widgets

Access Comparison Table:

Page/Endpoint	Member1 Access	Manager1 Access	Comment
TaskOverview	Yes	Yes	Accessible to both
TaskEdit	Yes	Yes	Accessible to both
TeamOverview	Yes	Yes	Should be restricted?
MyAccountViewEdit	Yes	Yes	Accessible to both
/xas/ API	Yes	Yes	Accessible to both

Security Analysis

- **Finding:** RBAC not enforced. Both roles share identical privileges.
- **Risk:** If manager-only features (e.g., TeamOverview) are intended to be restricted, this is a **critical RBAC flaw**.
- **Impact:** Regular users may access or modify managerial data, risking data confidentiality and integrity.

Conclusion & Recommendations

- Enforce role-based restrictions on sensitive features.

- Implement **backend authorization checks**, not only frontend logic.
- Retest after remediation to confirm proper RBAC enforcement.

2.2 Insecure Direct Object Reference (IDOR)

Objective

Evaluate whether users can access or manipulate other users' data by altering object identifiers (GUIDs) in API requests (OWASP A01: Broken Access Control).

Methodology

1. Logged in as **Member1** and captured an API request containing GUID.
   Fig. – Member1 GUID: 4503599627370897
2. Retrieved Manager1's GUID: 4503599627370898.
3. Modified Member1's request to include Manager1's GUID.
4. Sent tampered request.

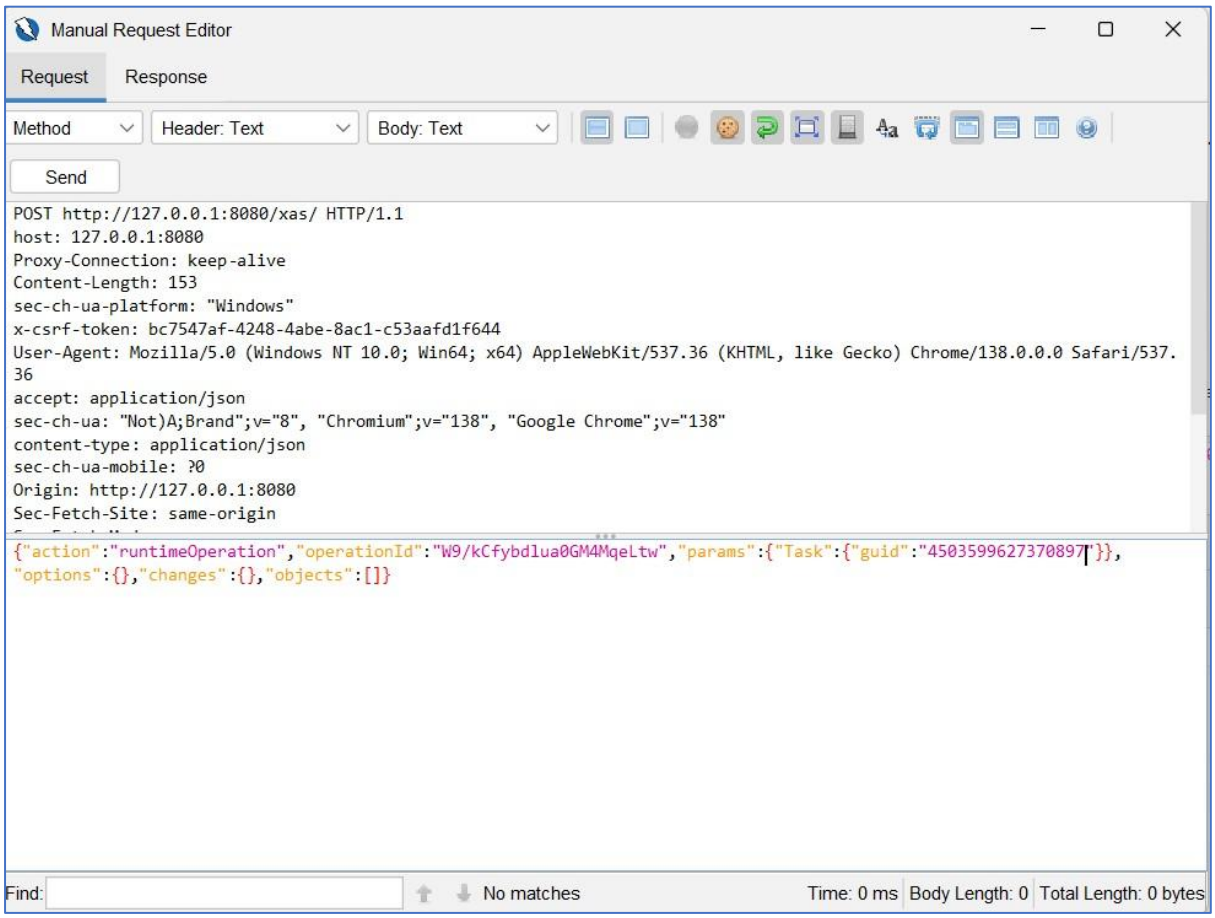


Fig. 23– Modified GUID request

5. Analyzed server response.

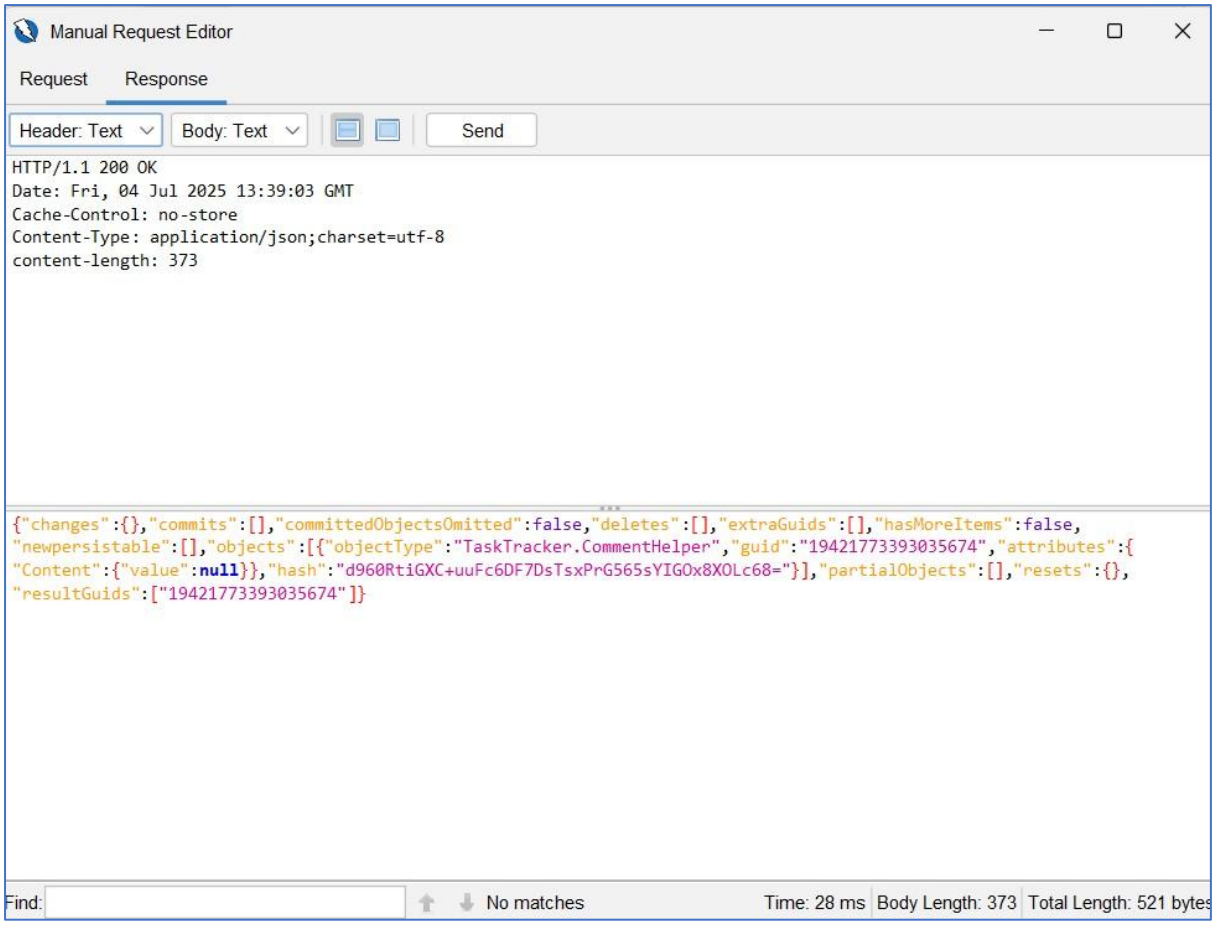


Fig. 24– 200 OK response

Results

- Server responded with **200 OK** for unauthorized resource access.
- No validation of object ownership occurred.

Security Analysis

- **Vulnerability:** IDOR confirmed.
- **Impact:** Unauthorized data access or manipulation.
- **Risk:** Critical (OWASP A01).

Conclusion & Recommendations

- Implement **server-side ownership validation** for all object IDs.
- Do not rely on user-supplied IDs for authorization.
- Conduct **regular IDOR testing** after code changes.

3. API Security & Input Validation

3.1 Unauthenticated API Access

Objective: Determine if /xas/ API allows unauthenticated access.

Methodology:

- Captured authenticated request using ZAP.
- Removed all cookies and headers.
- Resent the request.
- Observed response.

Results:

- Response: 401 Unauthorized.
- No data or logic exposed.

Analysis:

- Endpoint correctly enforces authentication.

Conclusion & Recommendations:

- Continue enforcing strict authentication checks.
- Retest after new endpoints or features are deployed.

3.2 Injection Vulnerabilities

Objective: Detect SQL Injection, XSS, and Command Injection vulnerabilities.

Methodology:

- Ran **OWASP ZAP Active Scan** on user-input endpoints.
- Reviewed alerts.

Results:

- No high or medium severity injection flaws detected.

Conclusion & Recommendations:

- Maintain **input sanitization and validation**.
- Retest periodically.

4. Cross-Site Request Forgery (CSRF)

Objective

Assess presence of anti-CSRF tokens for critical state-changing requests.

Methodology

- Passive scan during normal user activity (Member & Manager roles).
- Checked POST/PUT/DELETE requests for CSRF tokens.

Results

- **No anti-CSRF tokens present.**
- Critical endpoints unprotected.

Security Analysis

- **Vulnerability:** CSRF risk confirmed.
- **Risk:** High – enables unauthorized actions via forged requests.

Conclusion & Recommendations

- Implement **unique CSRF tokens** for all state-changing requests.
- Validate server-side.

5. Other Vulnerabilities (Active Scan)

High-Risk Findings

1. **Hash Disclosure – BCrypt**
    - Location: /xas/
    - Risk: Offline password cracking
    - Remediation: Remove hashes from responses
2. **PII Disclosure**
    - Location: /xas/
    - Risk: Identity theft, compliance breach
    - Remediation: Mask/remove sensitive data

Medium-Risk Findings

- Missing CSP headers (/, /index.html, /login.html)
- Cross-Domain Misconfiguration (CORS *)
- Missing Anti-Clickjacking Headers
- Spring Actuator Information Leak (/xas/actuator/health)

Low-Risk Findings

- Missing HttpOnly/SameSite flags

- Server version disclosure
- Missing HSTS headers
- Timestamp leakage in JS
- Missing X-Content-Type-Options: nosniff

Summary Table

Severity	Issue	Endpoint(s)	Recommendation
High	Hash Disclosure	POST /xas/	Remove hashes from all responses
High	PII Disclosure	POST /xas/	Mask/remove sensitive data
Medium	Missing CSP Header	All pages	Add strict CSP headers
Medium	CORS Misconfiguration	External resources	Limit allowed origins
Medium	Missing Clickjacking Header	All responses	Add X-Frame-Options or CSP
Medium	Spring Actuator Leak	/xas/actuator/health	Restrict/disable actuator
Low	Cookie Flags Missing	All cookies	Add HttpOnly & SameSite
Low	HSTS Not Set	All endpoints	Enforce HSTS
Low	Timestamp Disclosure	Static assets, APIs	Remove unnecessary timestamps
Low	X-Content-Type-Options	All	Add nosniff header

---

6. Conclusion & Urgent Recommendations

- Critical vulnerabilities include **RBAC failure, IDOR, missing CSRF tokens, and sensitive data exposure**.
- Medium and low risks involve **misconfigurations and missing headers**.

Immediate Actions

1. Remove password hashes & PII from API responses.
2. Implement proper **RBAC & server-side ownership validation**.
3. Add CSRF tokens to all state-changing requests.
4. Enforce security headers: **CSP, HSTS, X-Frame-Options, X-Content-Type-Options**.

5. Restrict actuator endpoints and clean up sensitive information leaks.

BugBug Analysis Report

1. Coffee Service App

1.1 Authentication Tests

A) Valid Login Test

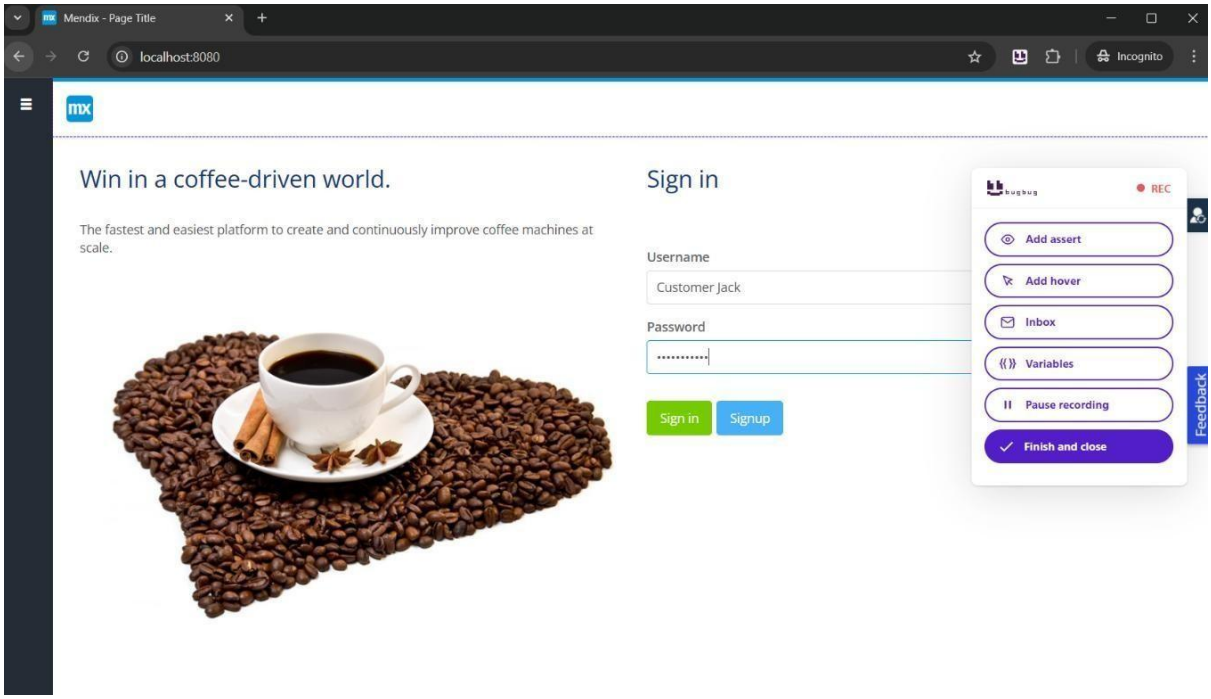
Objective: Verify successful login for Customer (Jack) and Engineer (Bill).
Steps:

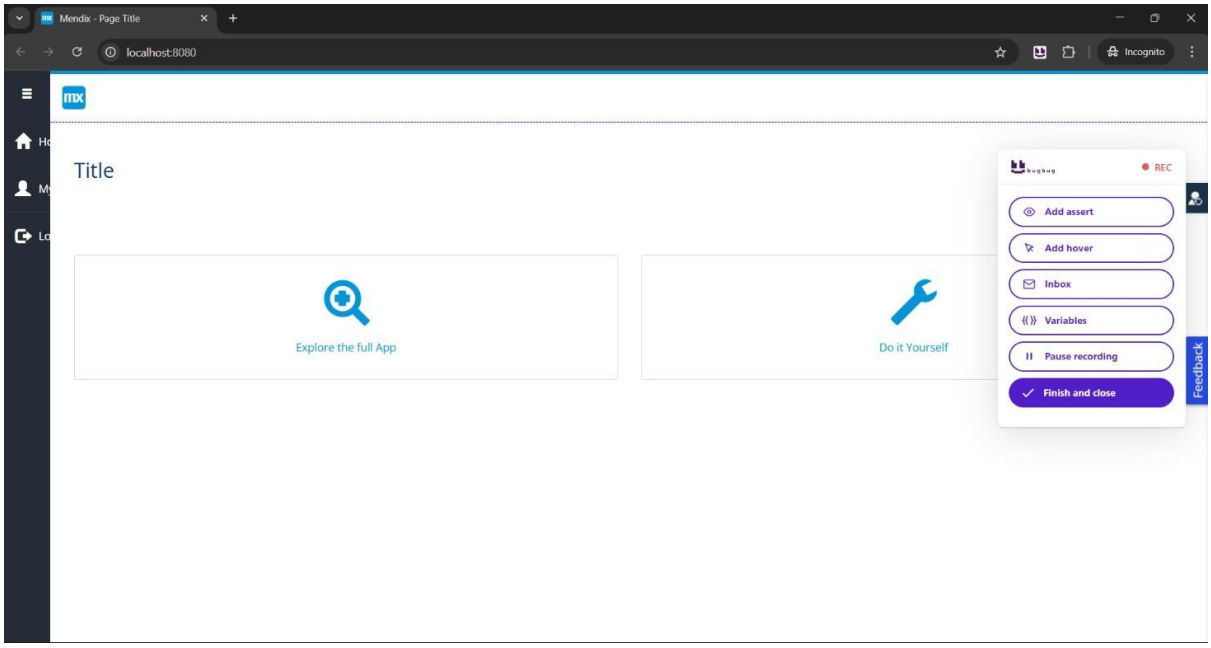
1. Navigate to login page.
2. Enter valid credentials for Customer (Jack).
3. Enter valid credentials for Engineer (Bill).
4. Click *Login*.
5. Verify dashboard loads.

Results:

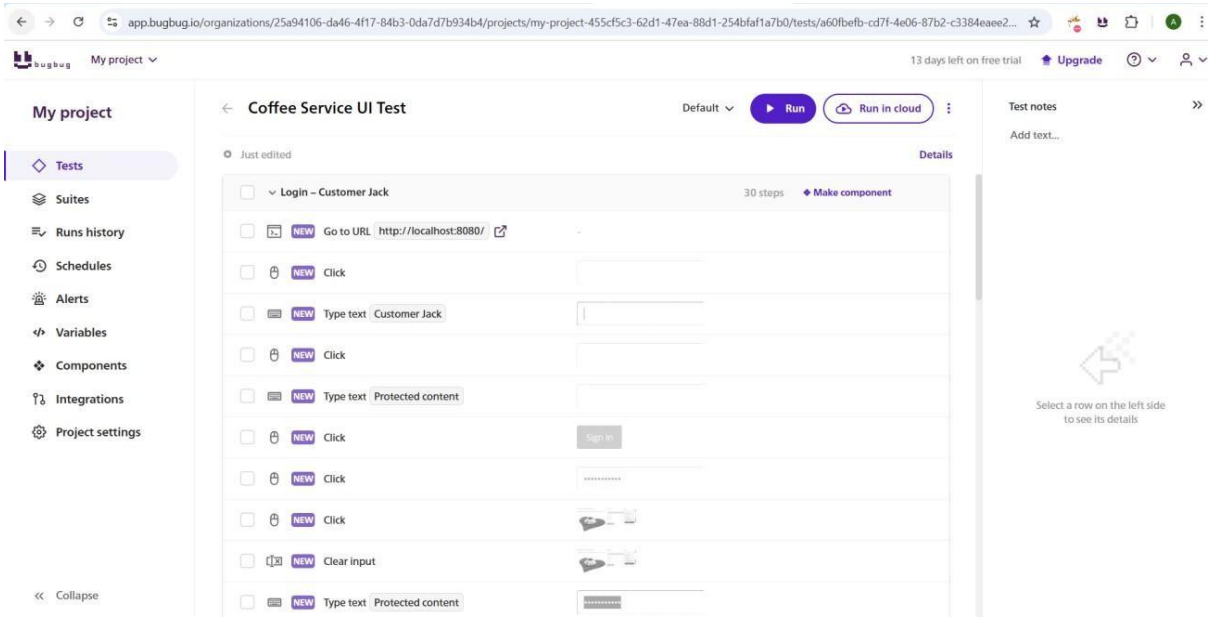
- Customer: Login successful, dashboard displayed.
- Engineer: Login successful, dashboard displayed.

Figures:

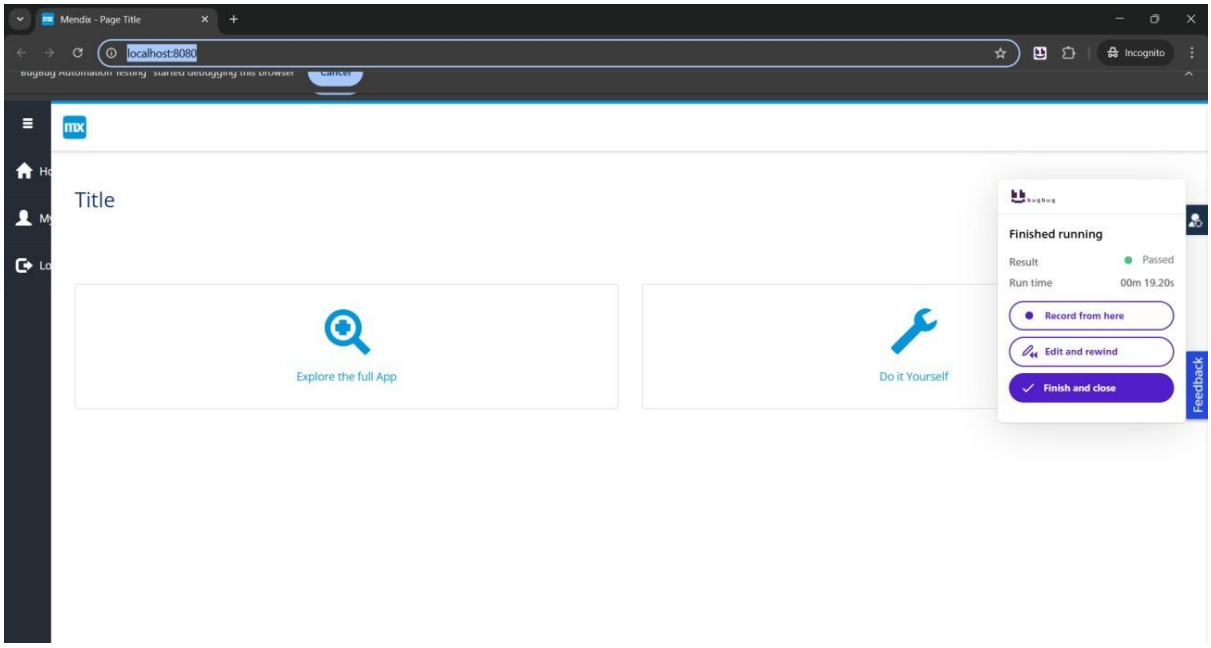




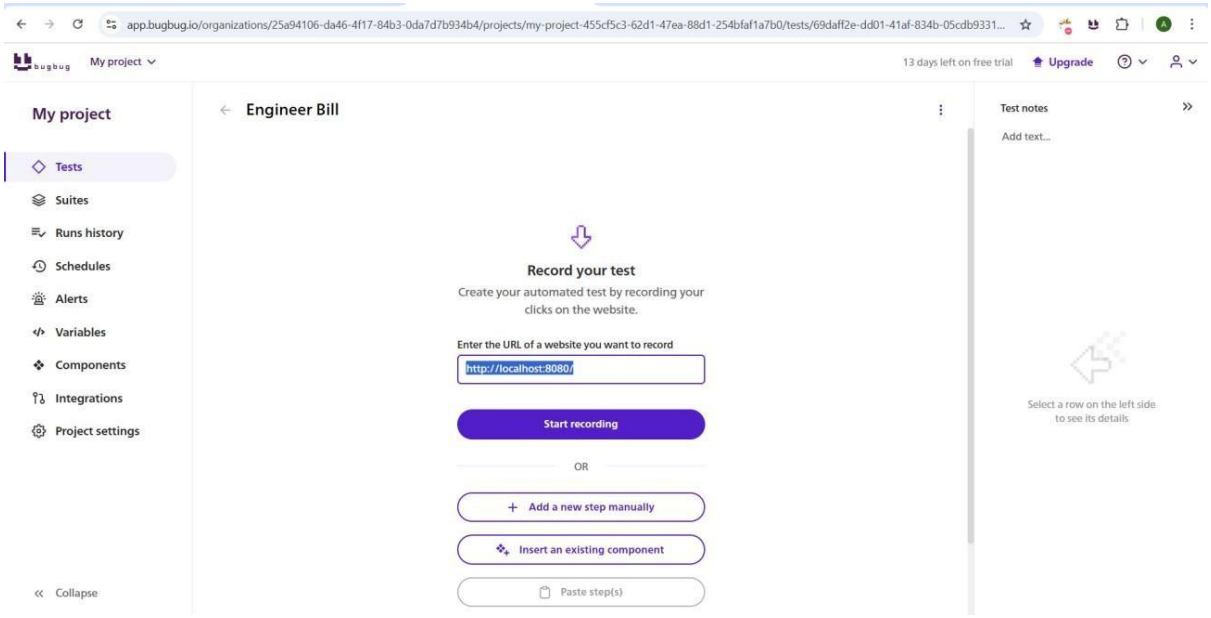
- Fig 1: Logged in as Customer Jack



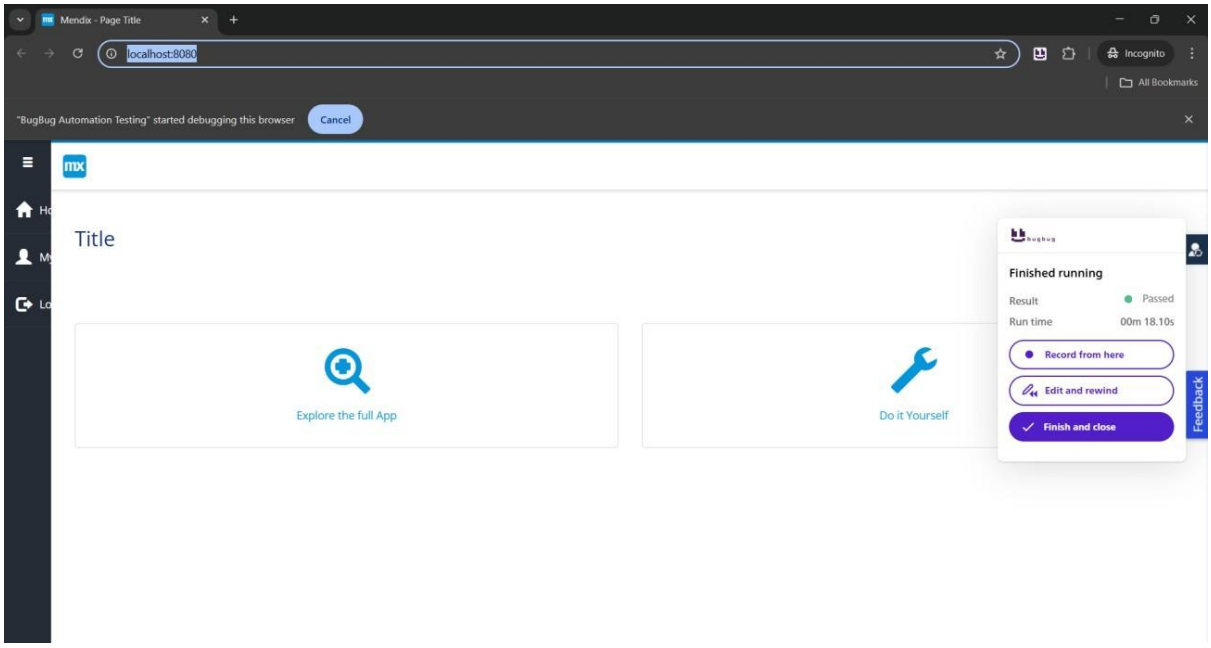
- Fig 2: Dashboard after successful login



- Fig 3: Login test passed for Customer Jack



- Fig 4: Starting test for Engineer Bill



- Fig 5: Login passed for Engineer Bill

Status: PASSED
Risk:

- Likelihood: Low
- Impact: High

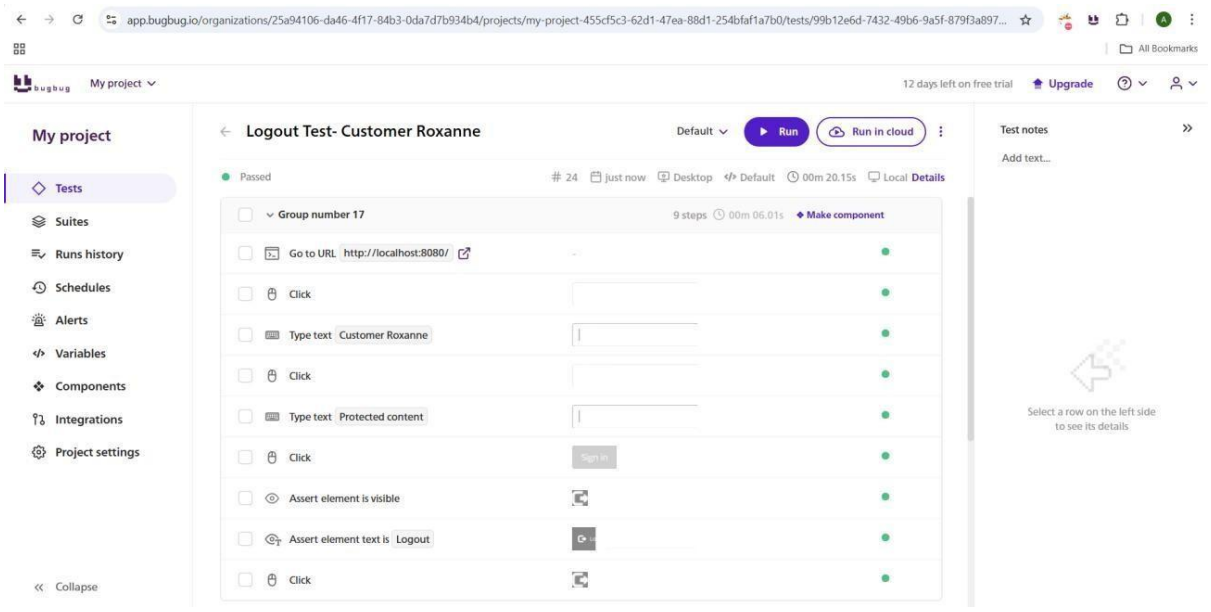
Recommendation: Retest after authentication module updates.

B) Logout Test

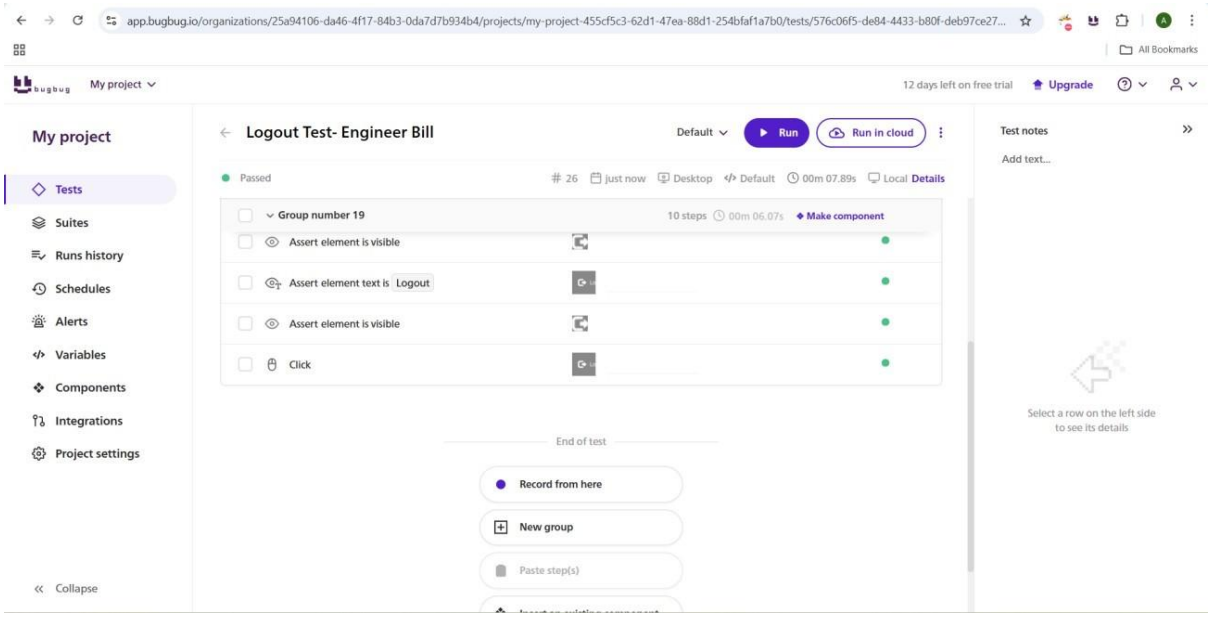
Objective: Verify successful logout for both roles.
Steps:

1. Log in as Customer (Roxanne) and Engineer (Bill).
2. Click *Logout*.
3. Verify redirection to login screen.

Figures:



- Fig 6: Logout test passed for Customer Roxanne



- Fig 7: Logout test passed for Engineer Bill

Status: PASSED
Risk:

- Likelihood: Low
- Impact: Moderate

C) Invalid Login Test

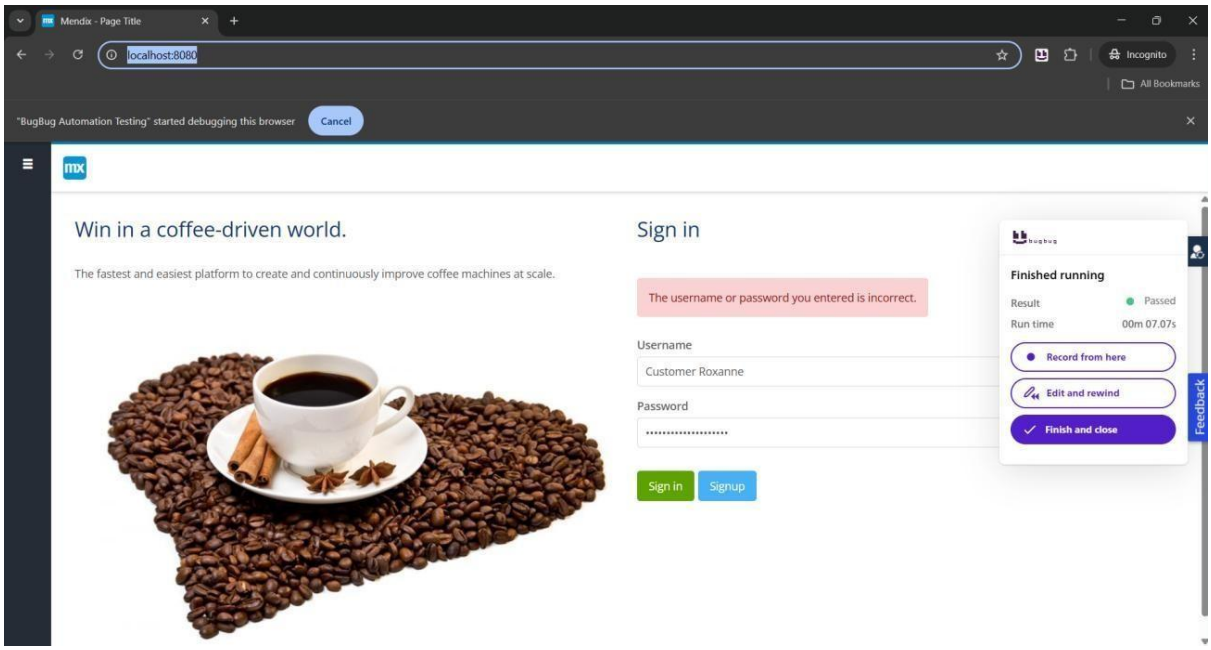
Objective: Verify consistent error messages for invalid credentials.
Steps:

1. Attempt login with incorrect credentials for both roles.
2. Observe error message.

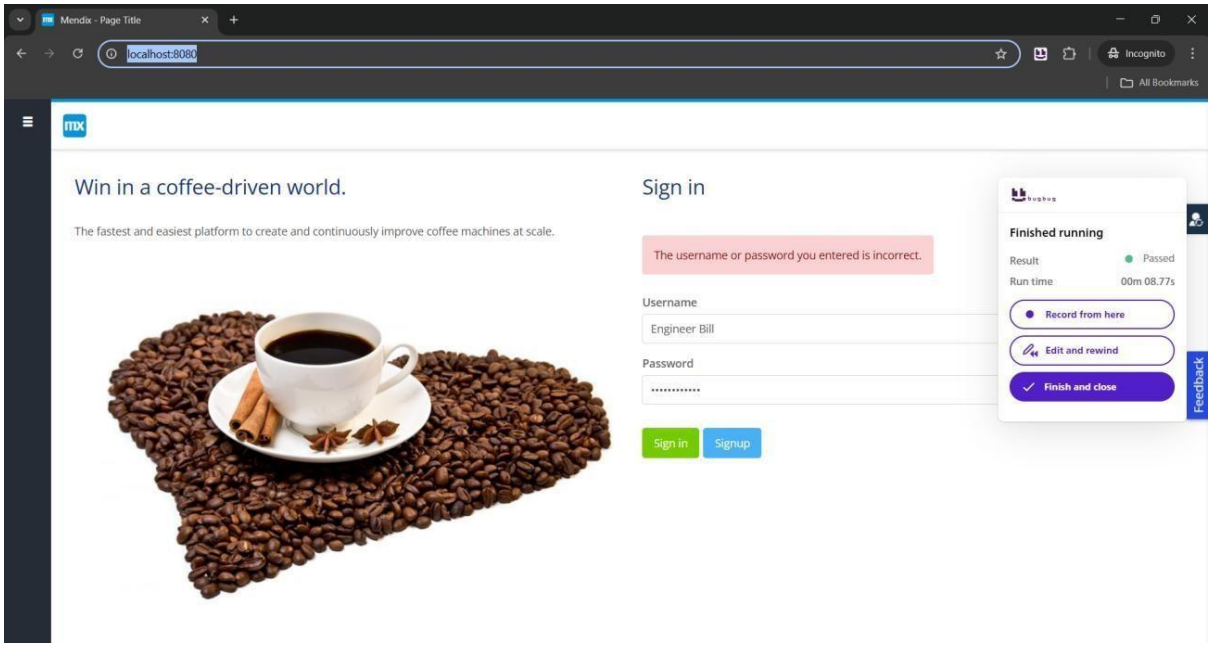
Results:

- Customer: Displays “Invalid credentials”.
- Engineer: Displays “Invalid credentials”.

Figures:



- Fig 8: Invalid login for Customer Roxanne



- Fig 9: Invalid login for Engineer Bill

Status: PASSED
Risk:

- Likelihood: Low
- Impact: Moderate

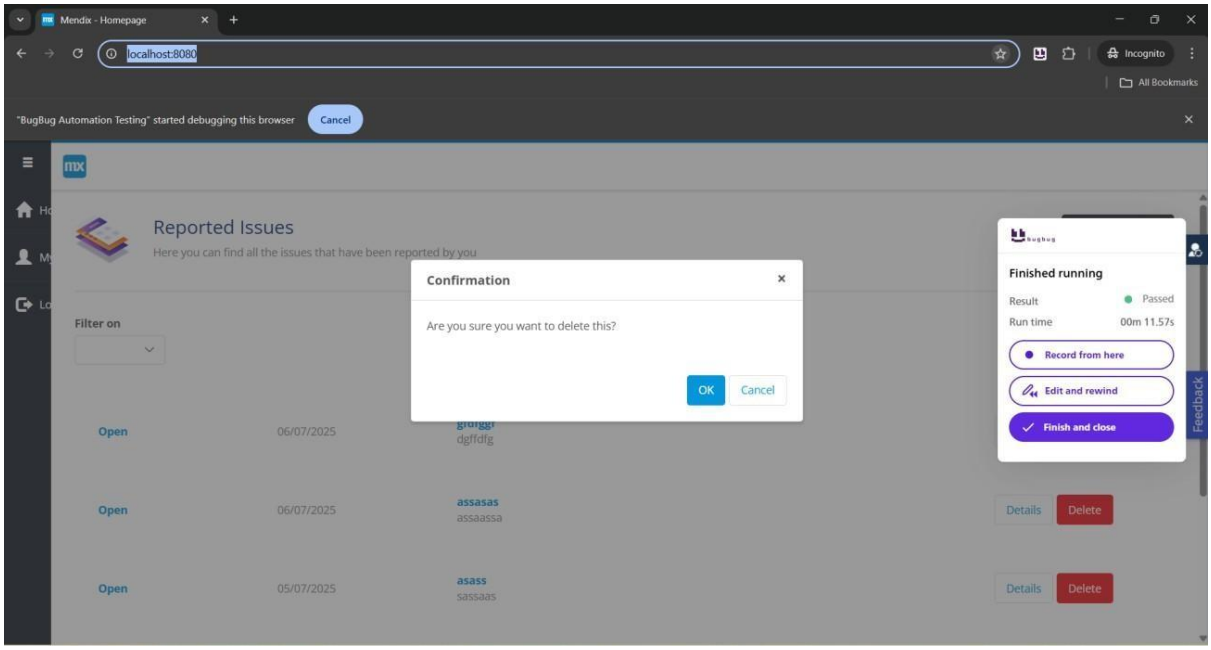
1.2 UI Functional Tests

A) Button Click Test

Objective: Verify Save, Cancel, and Next buttons work correctly.
Steps:

1. Click each critical button.
2. Assert expected behavior.

Figures:



- Fig 10: Button click test passed

Status: PASSED
Risk:

- Likelihood: Low
- Impact: Moderate

1.3 Form Submissions

A) Valid Data

- Forms submitted successfully.

- Confirmation displayed.

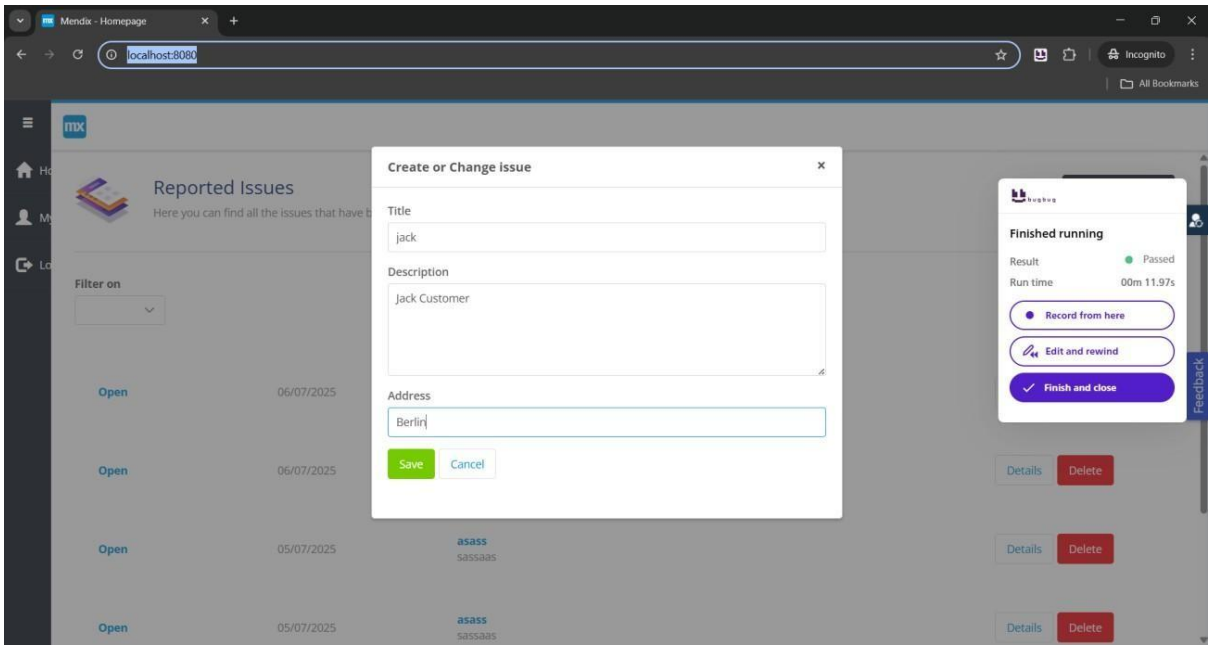


Fig 10: Valid data test passed

Status: PASSED

B) Invalid/Missing Data

- Validation errors displayed correctly.

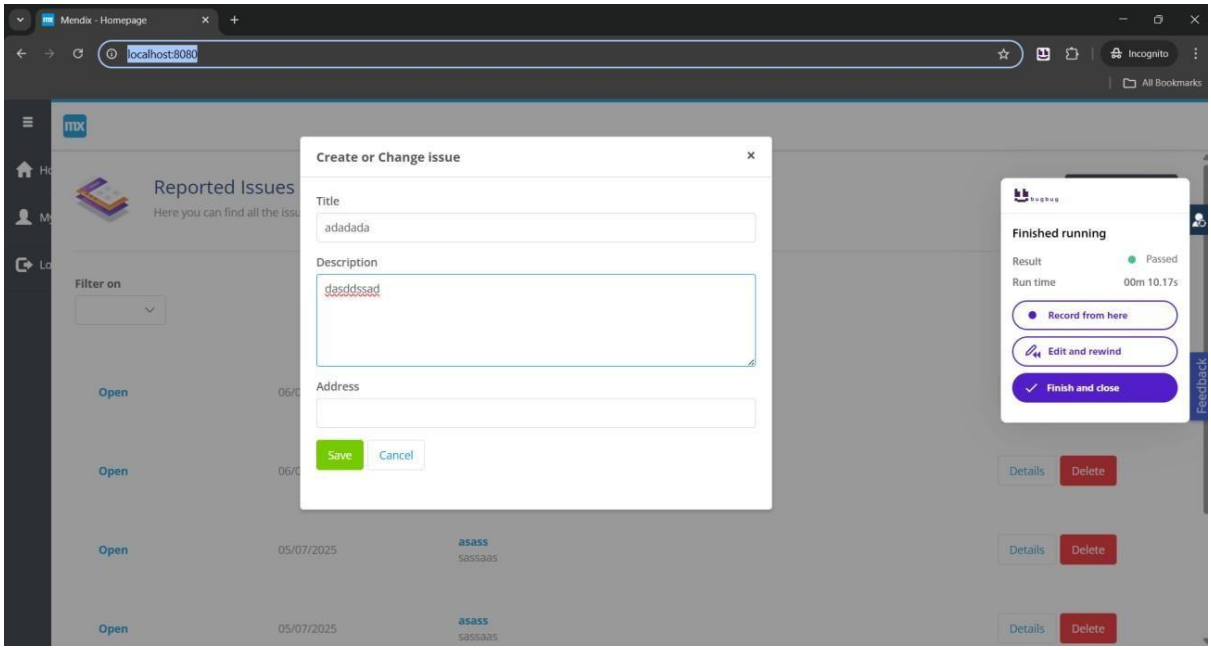


Fig 11: Invalid data test passed

Status: PASSED

Risk:

- Likelihood: Low
- Impact: Moderate

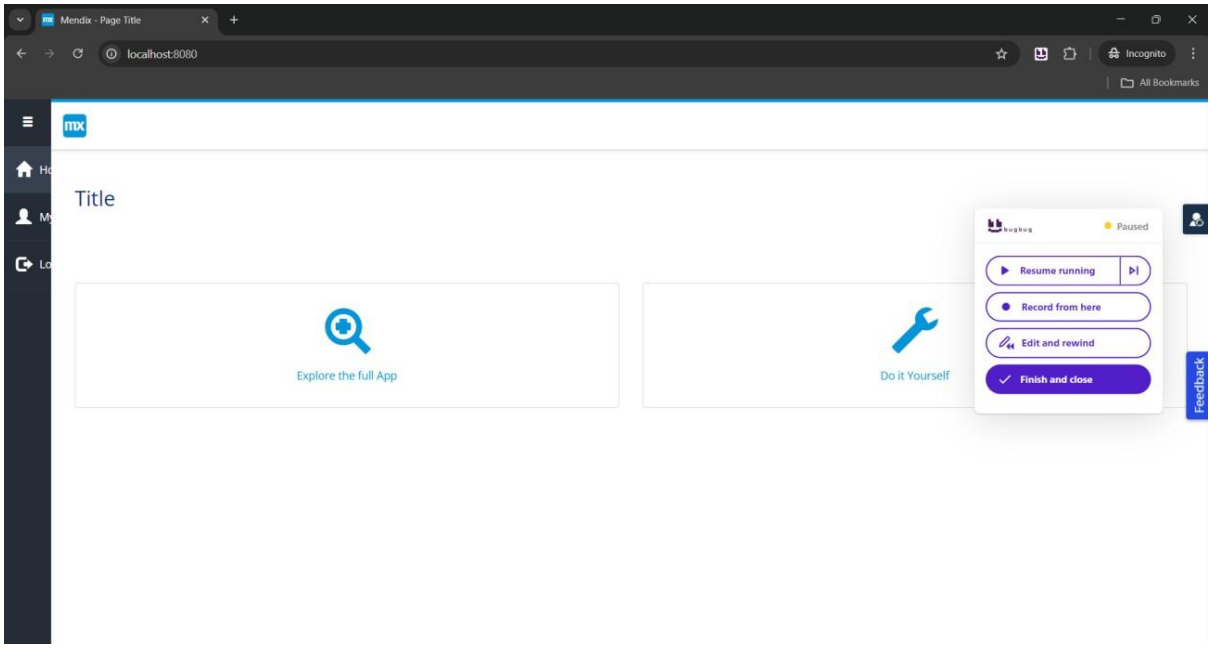
1.4 Role-Based Access Control (RBAC)

Objective: Verify that Customer, Engineer, Admin, and Manager can only access their authorized features.

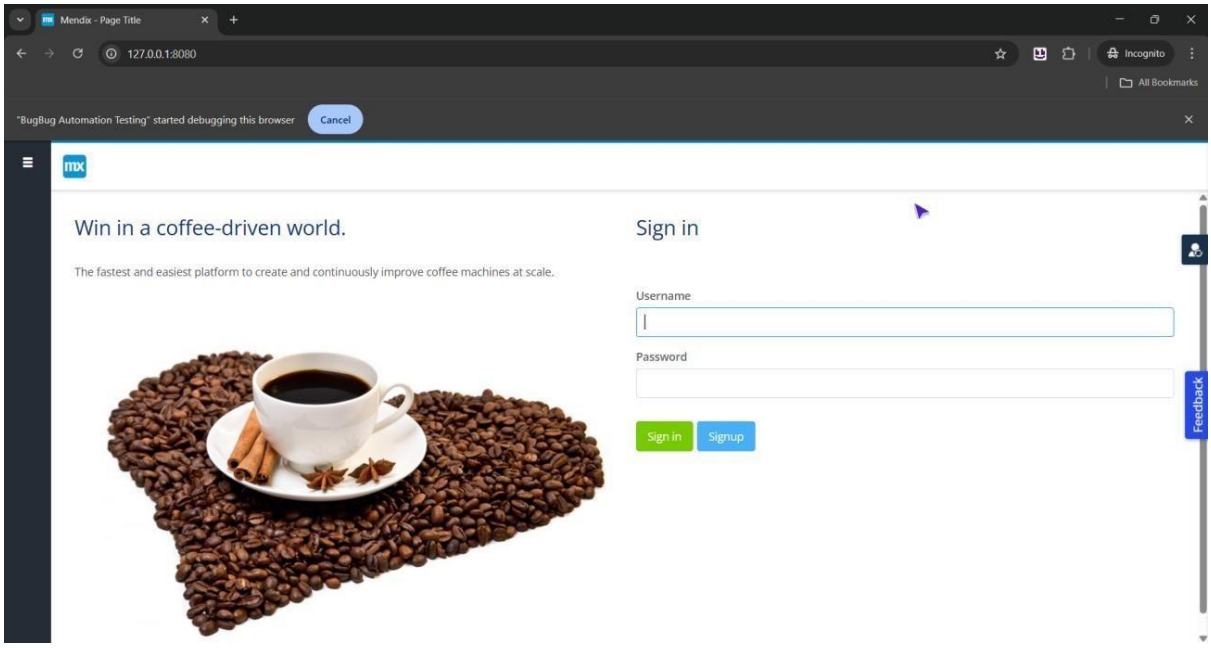
Results:

- Unauthorized access attempts were redirected.

Figures:



- Logged in as Customer Jack



- Attempted Engineer URL http://127.0.0.1:8080/

Status: PASSED
Risk:

- Likelihood: Low
- Impact: Critical

1.5 General Recommendations

- Integrate BugBug tests into CI/CD.
- Expand test coverage regularly.
- Store screenshots/logs for audits.
- Perform manual exploratory testing periodically.

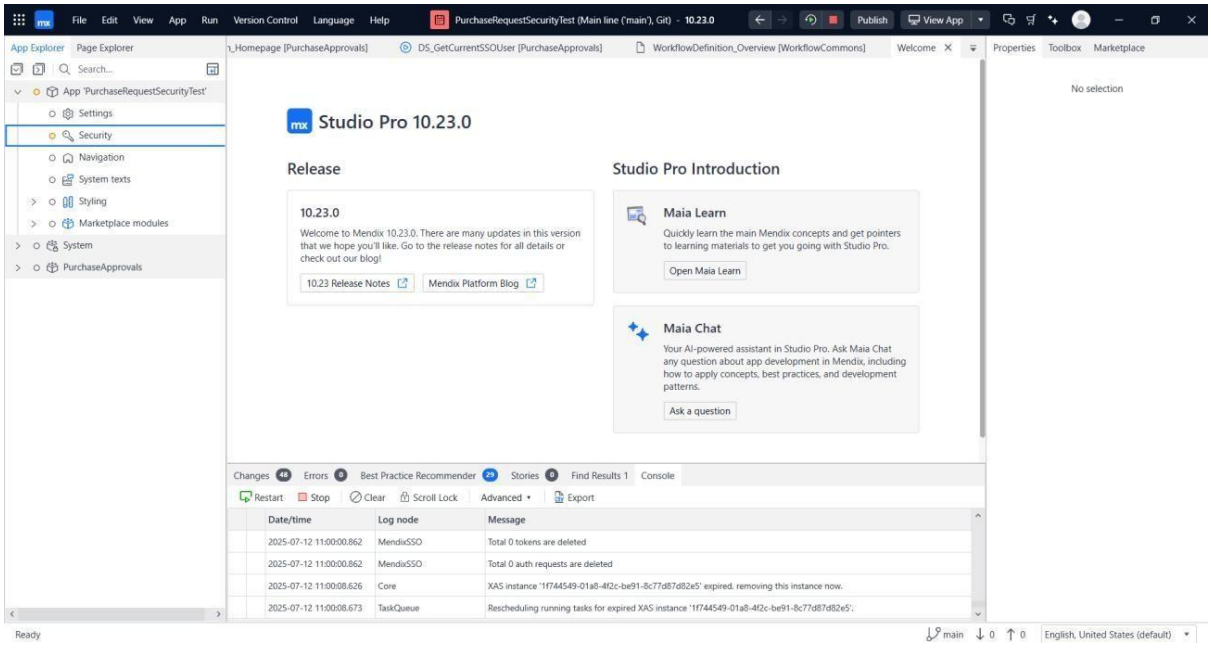
2. Purchase Request App

2.1 Authentication Tests

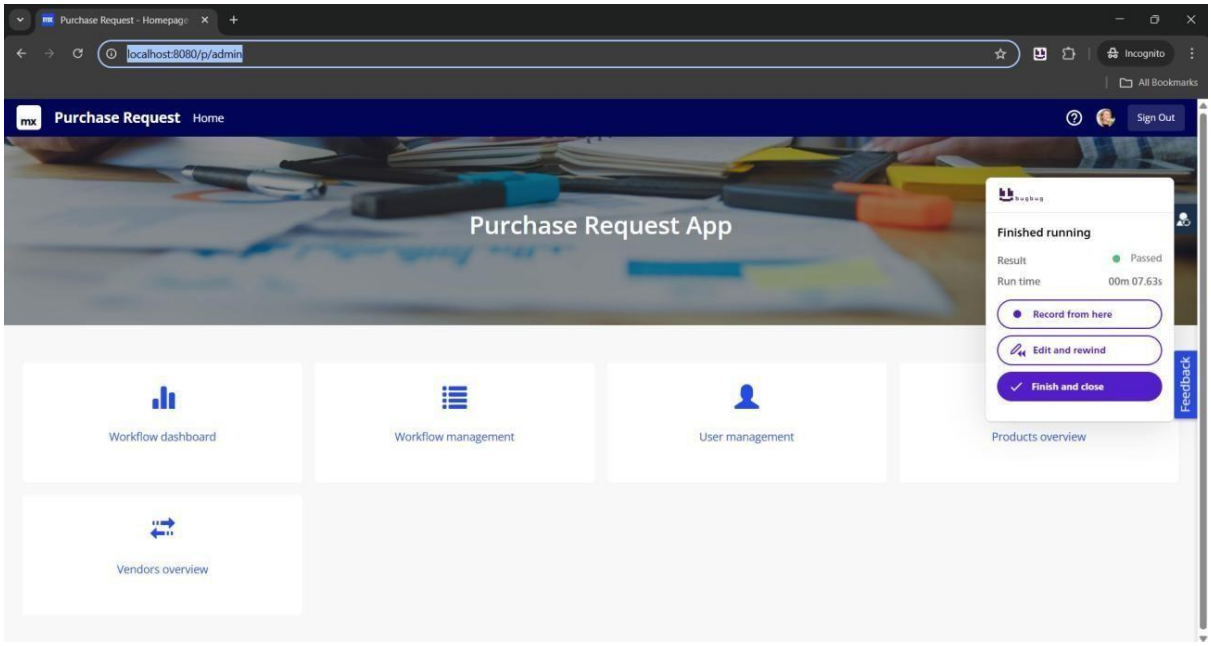
A) Valid Login Test

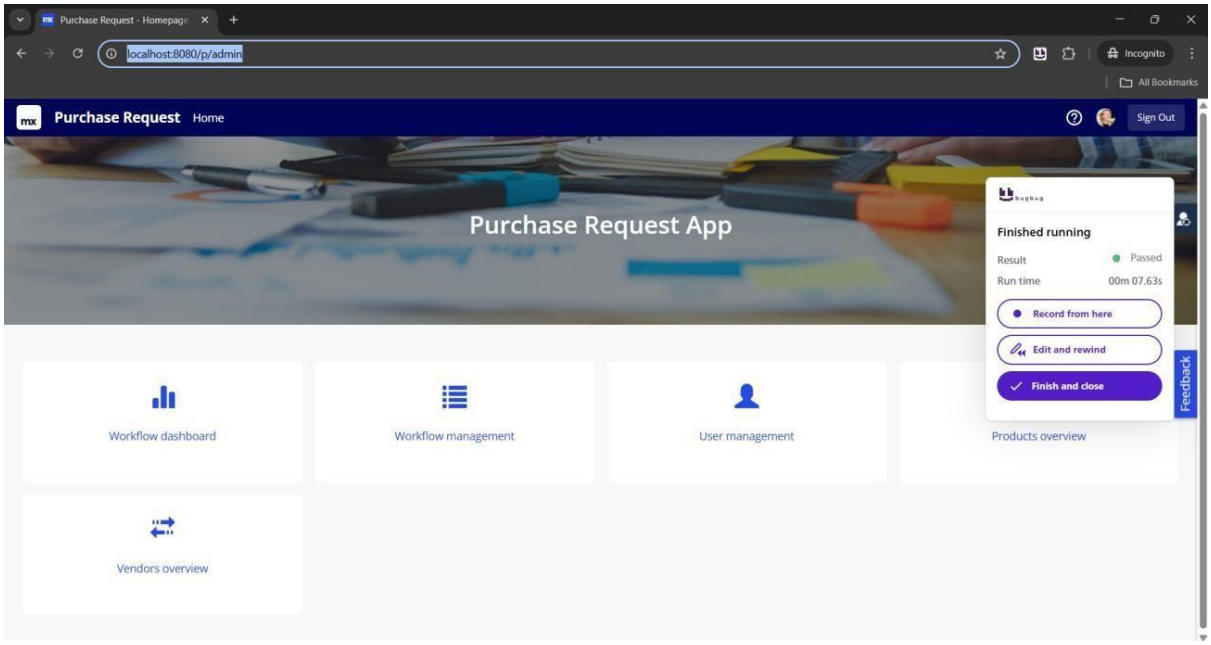
Objective: Verify successful login for Demo Administrator and Demo Approver.

Figures:

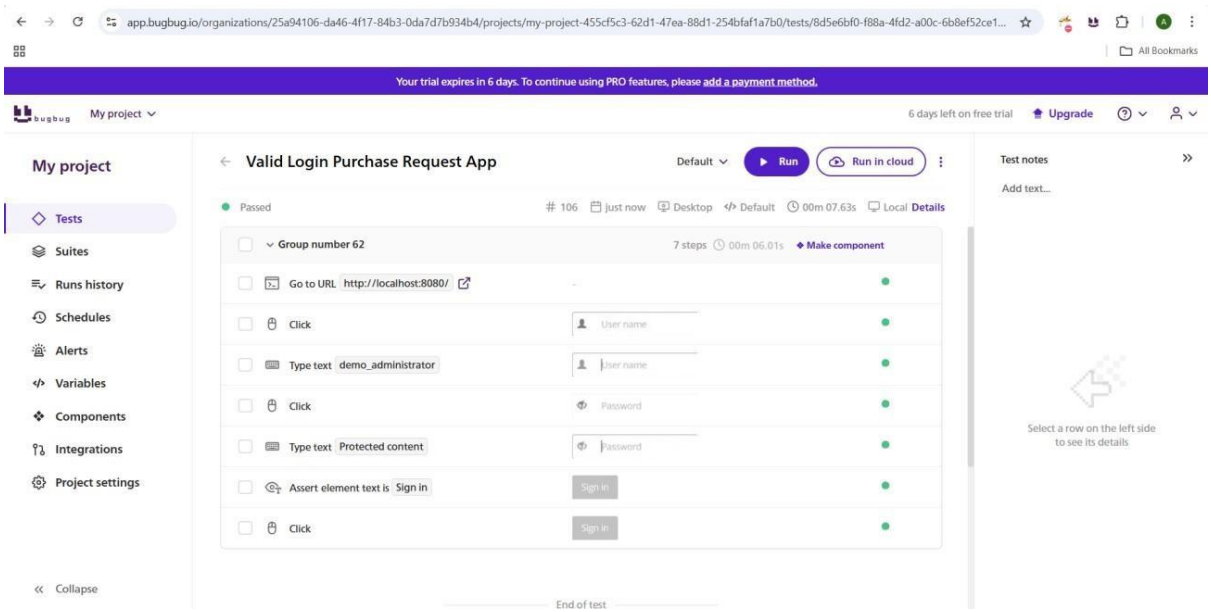


- Fig 1: App started

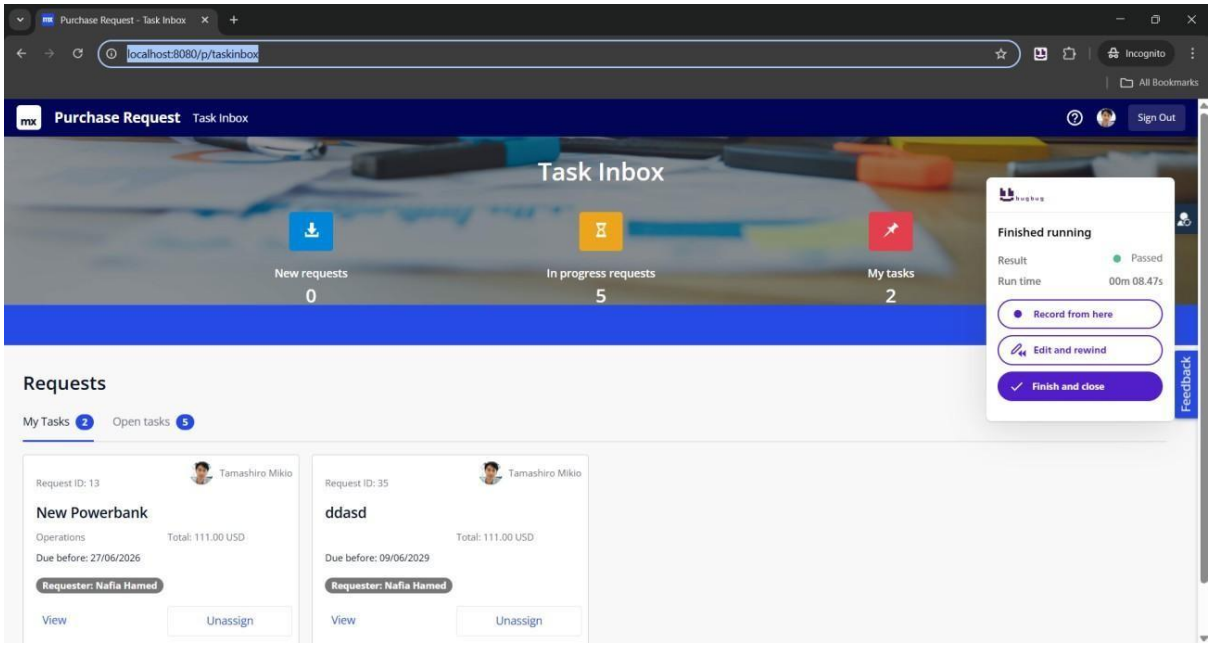




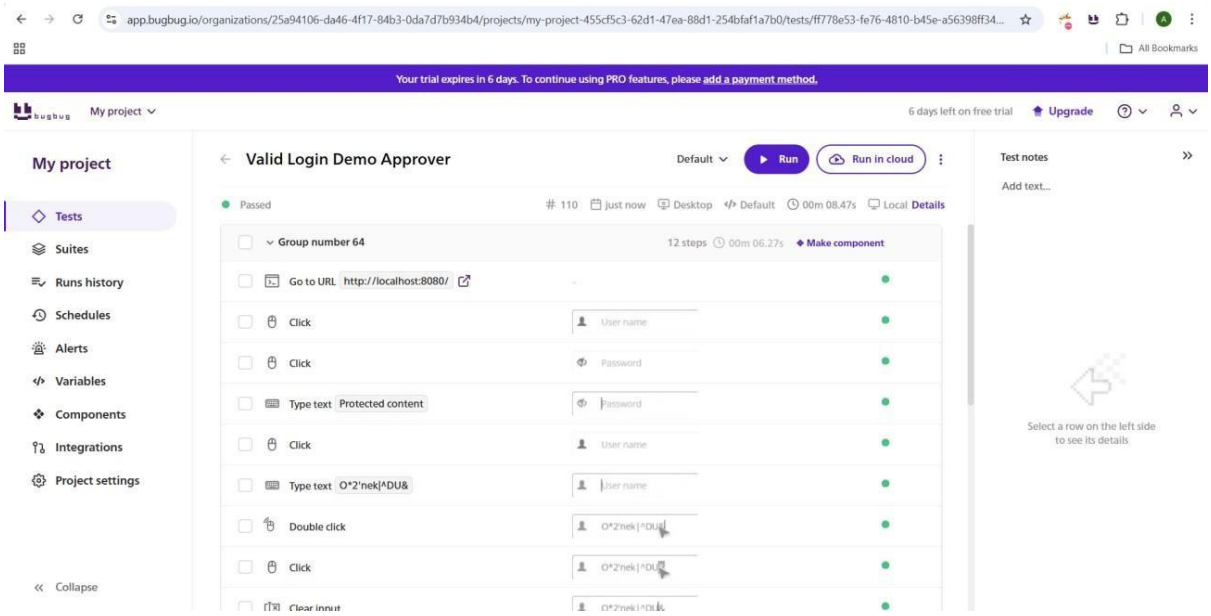
- Fig 2: Signed in as Demo Administrator



- Fig 3: Assertions – Administrator



- Fig 4: Login successful – Demo Approver



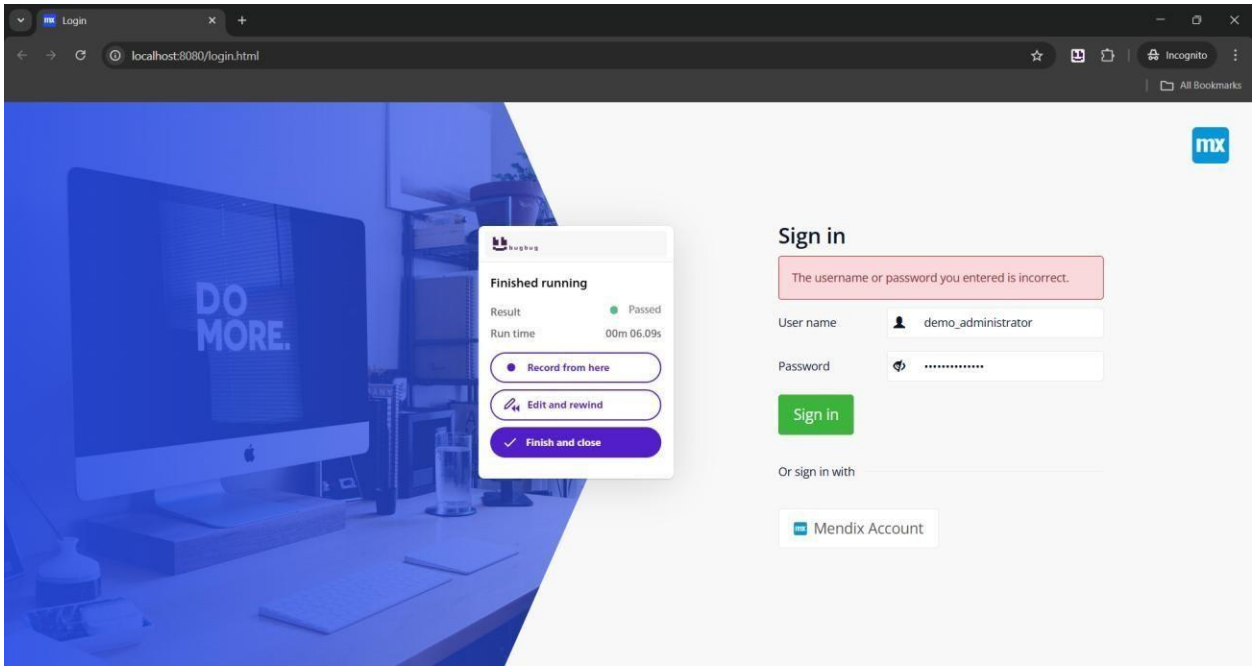
- Fig 5: Assertions – Approver

Status: PASSED

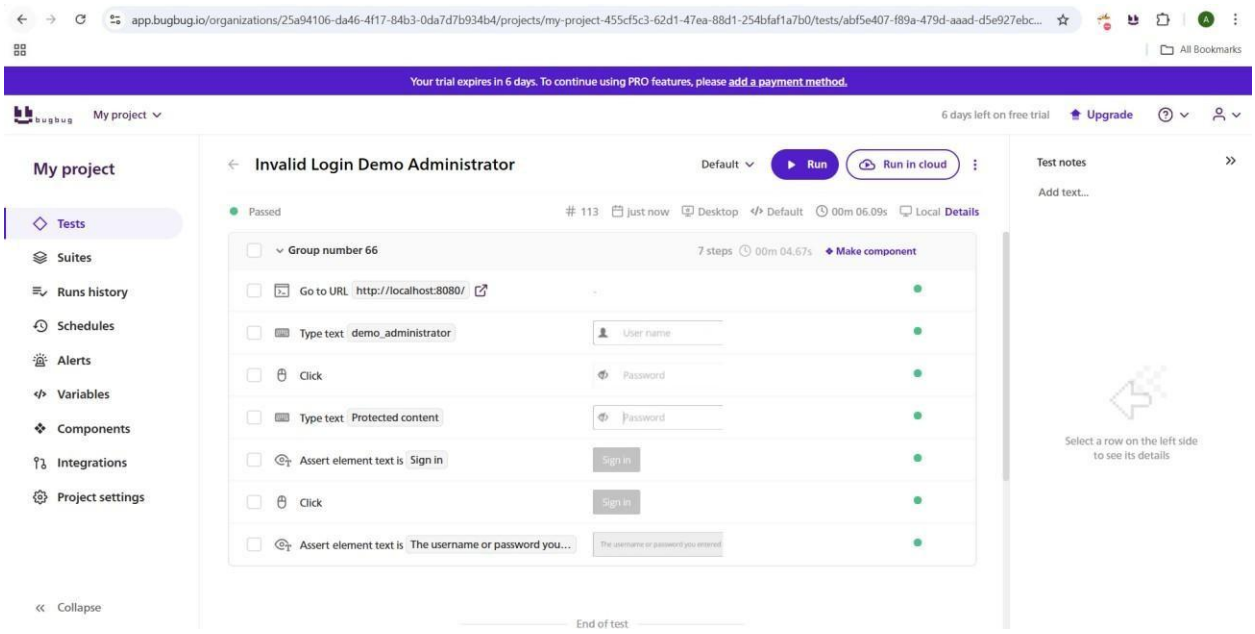
B) Invalid Login Test

Objective: Verify error messages for invalid credentials.

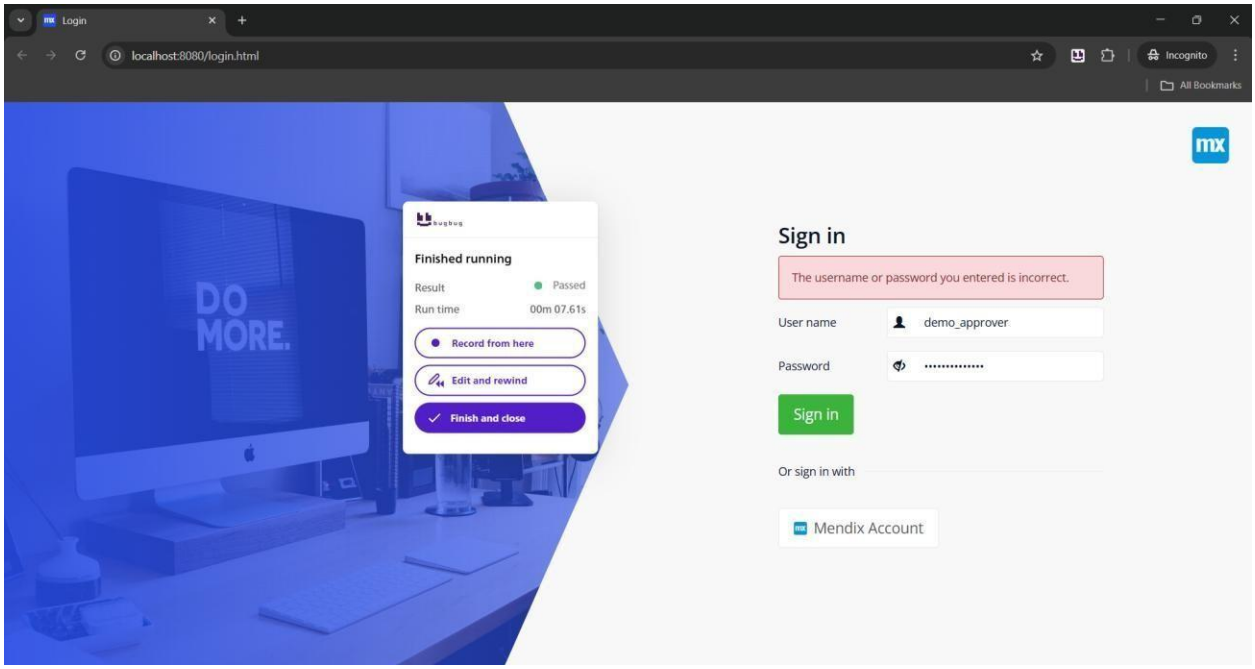
Figures:



Invalid login tried with incorrect password on demo administrator



Assertions Used for Invalid Tests on Demo Administrator



Invalid login attempted on Demo Approver, Sign in Failed

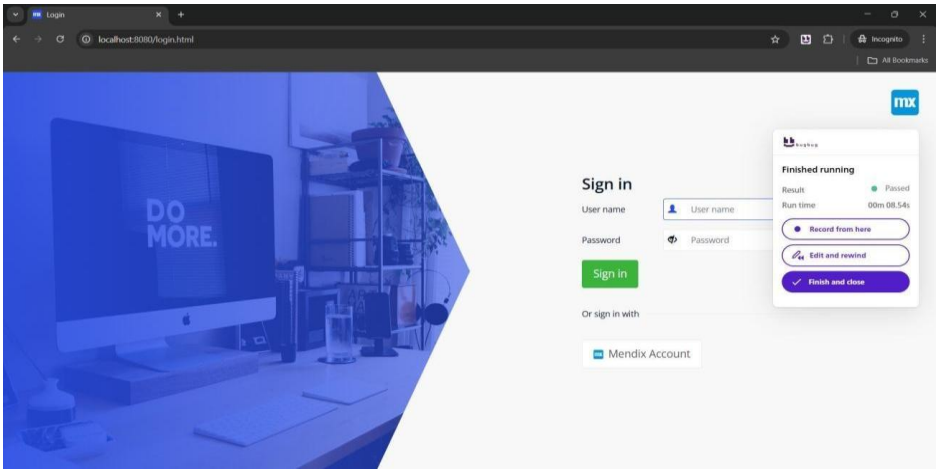
- Invalid login attempts

Status: PASSED

C) Logout Test

Figures:

B)



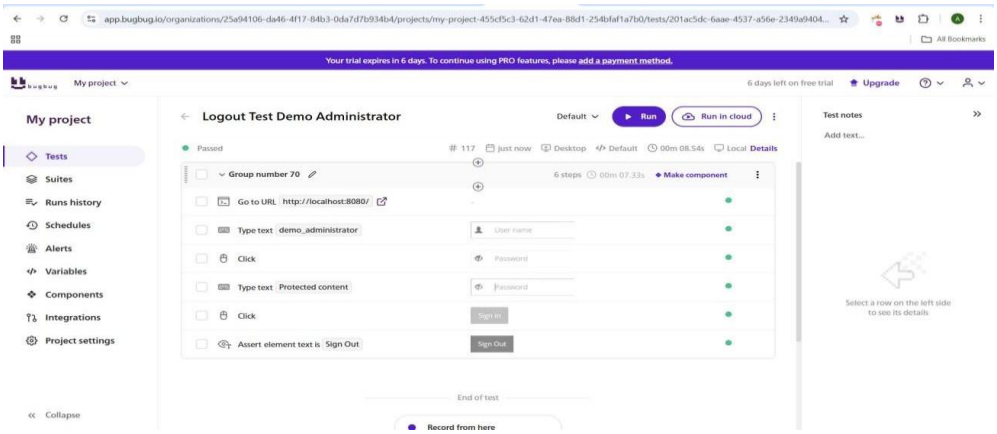
Sign out was successful for demo administrator

Test Objective: Validate successful logout and ensure session termination with no residual access.

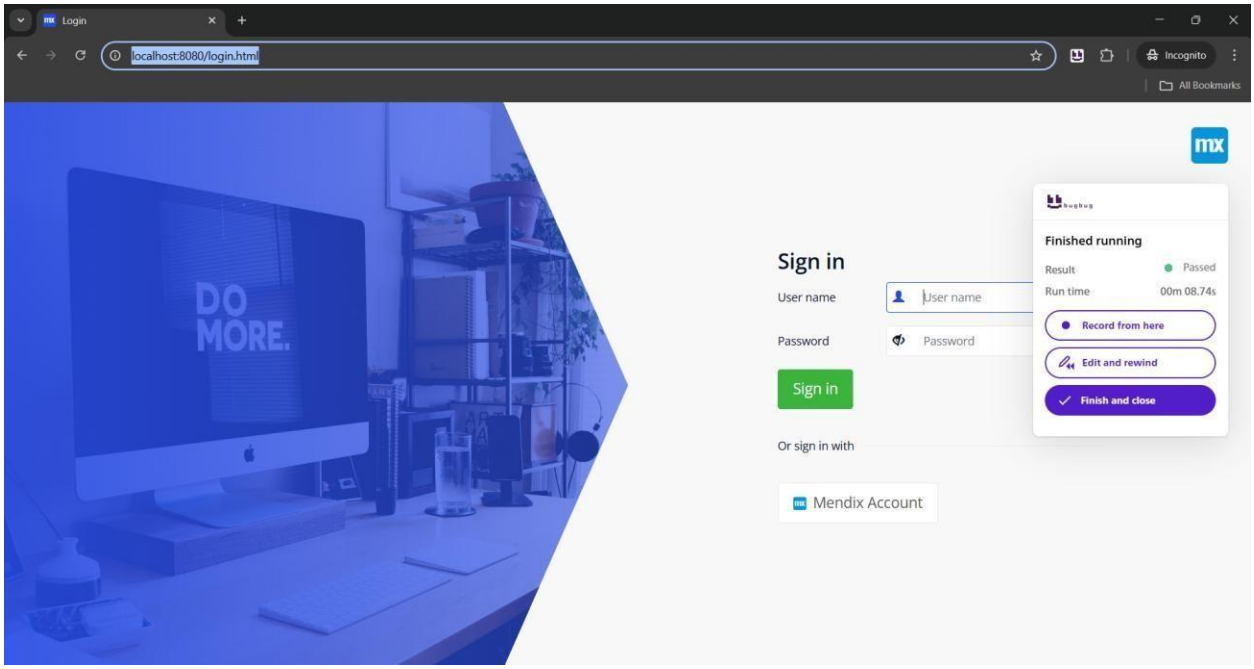
Steps:

- **Logout after valid login as Demo Administrator and Demo Approver.**

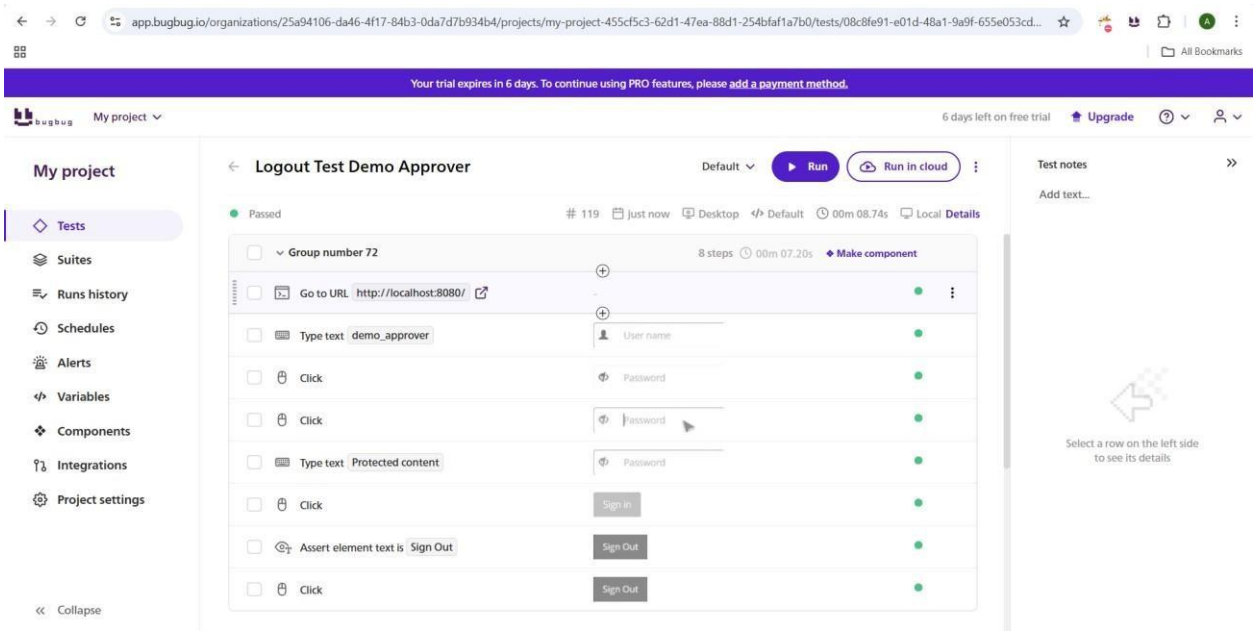
- **Verify redirection to the login screen.**



Assertions used for Demo Administrator



Sign out Successfully for Demo Approver



Assertions Performed on Sign out test Status: PASSED

Status: PASSED

2.2 UI Functional Tests

- **Button Clicks:** Passed for Administrator, Approver, Requester.
- **Test Objective: Verify all critical buttons trigger correct behaviors.**
  Steps:
- **Test Cancel, Save, and other major buttons for all user roles (Administrator, Approver, Requester).**

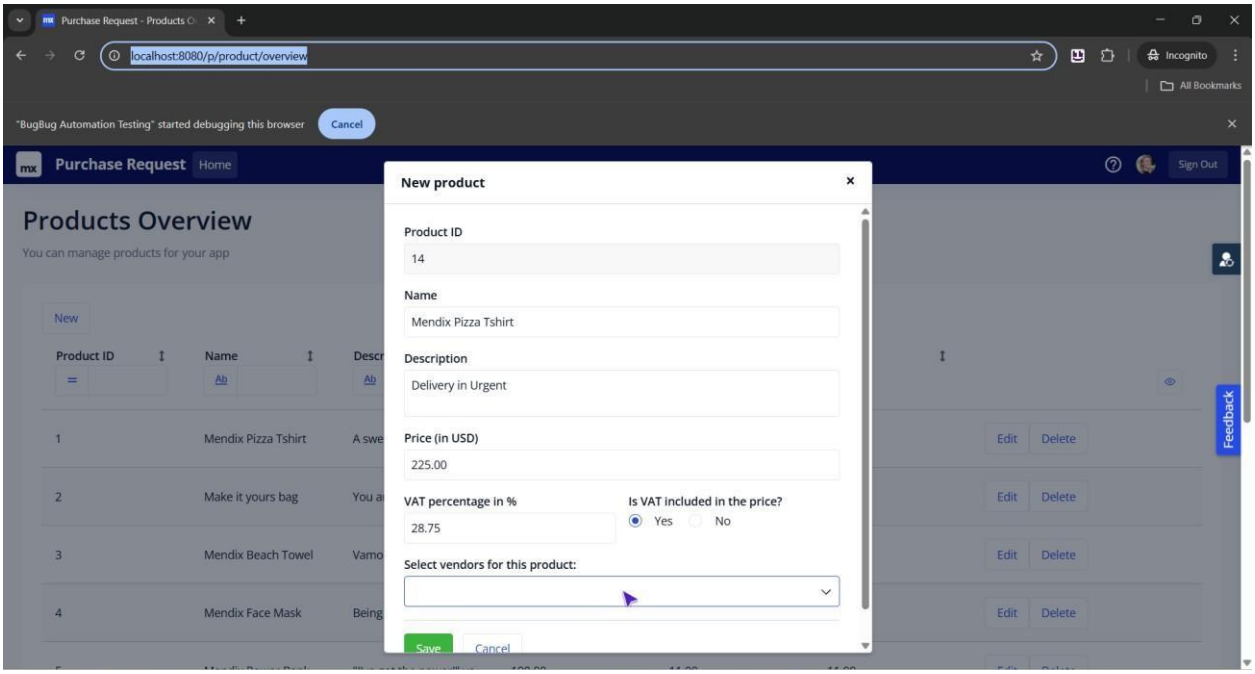
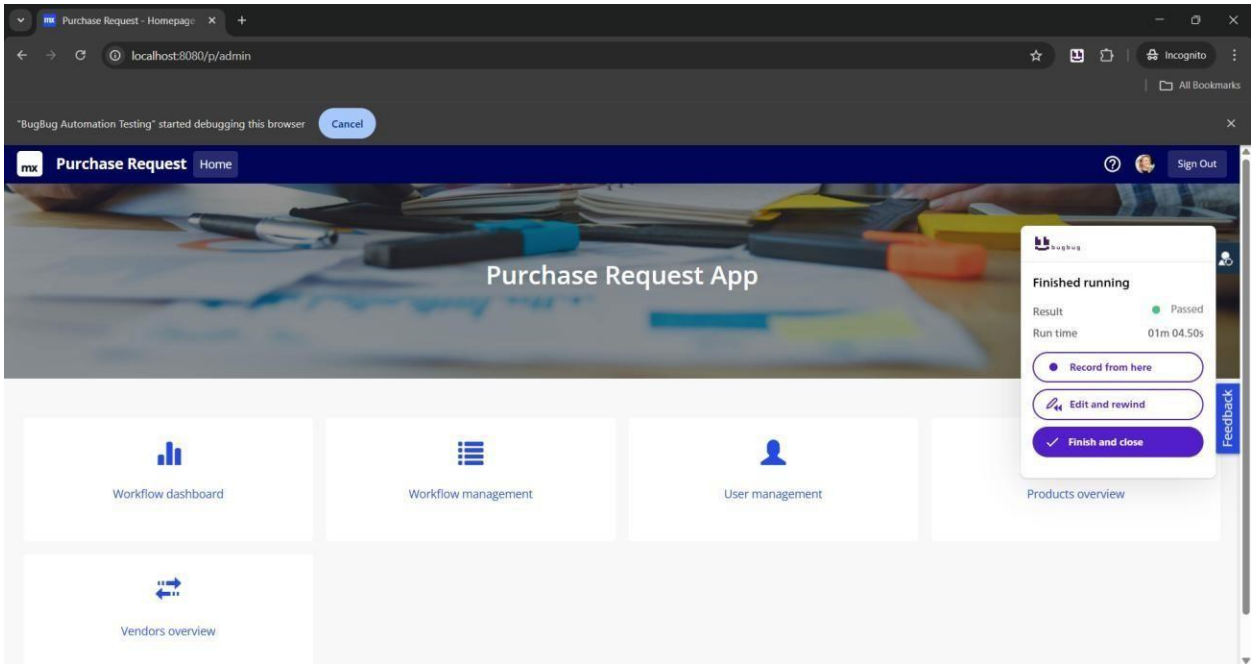
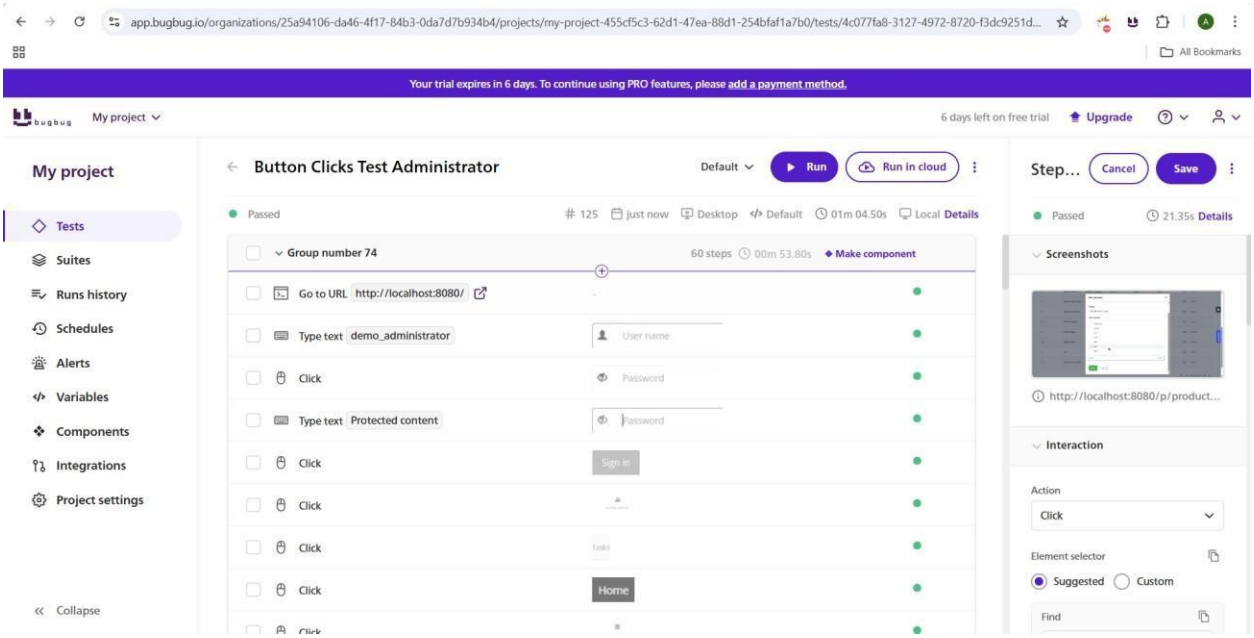


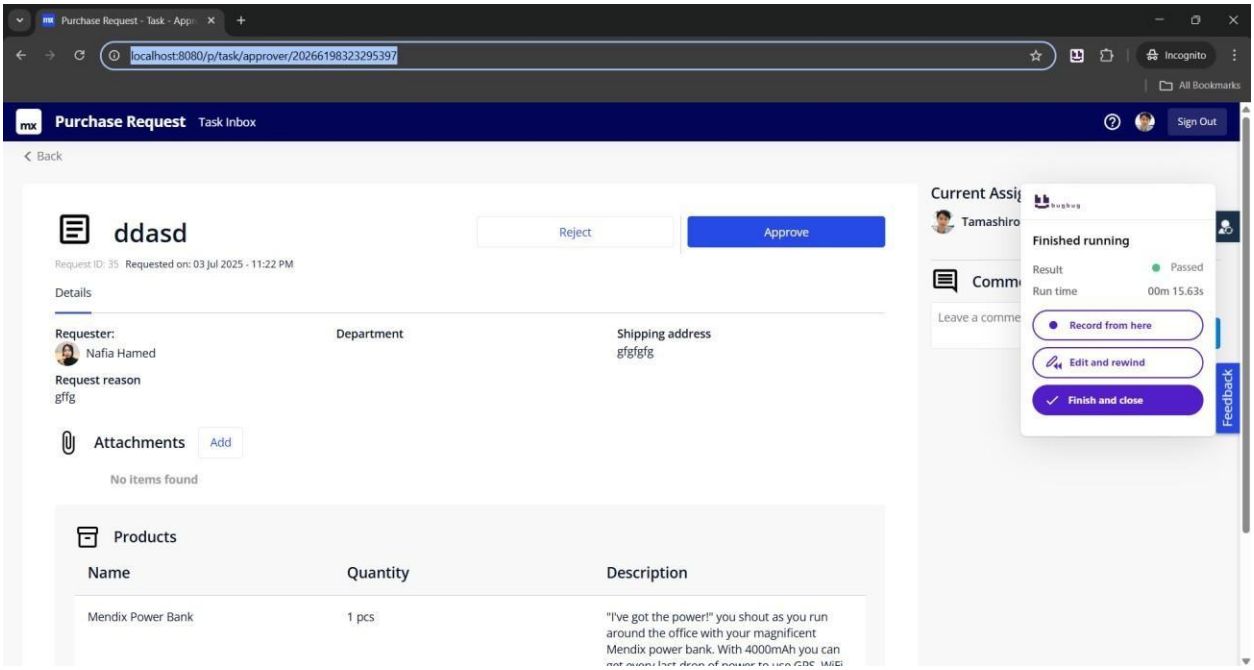
Figure 13: Running Buton Click Tests and Checking cancel and Save Button Functionality



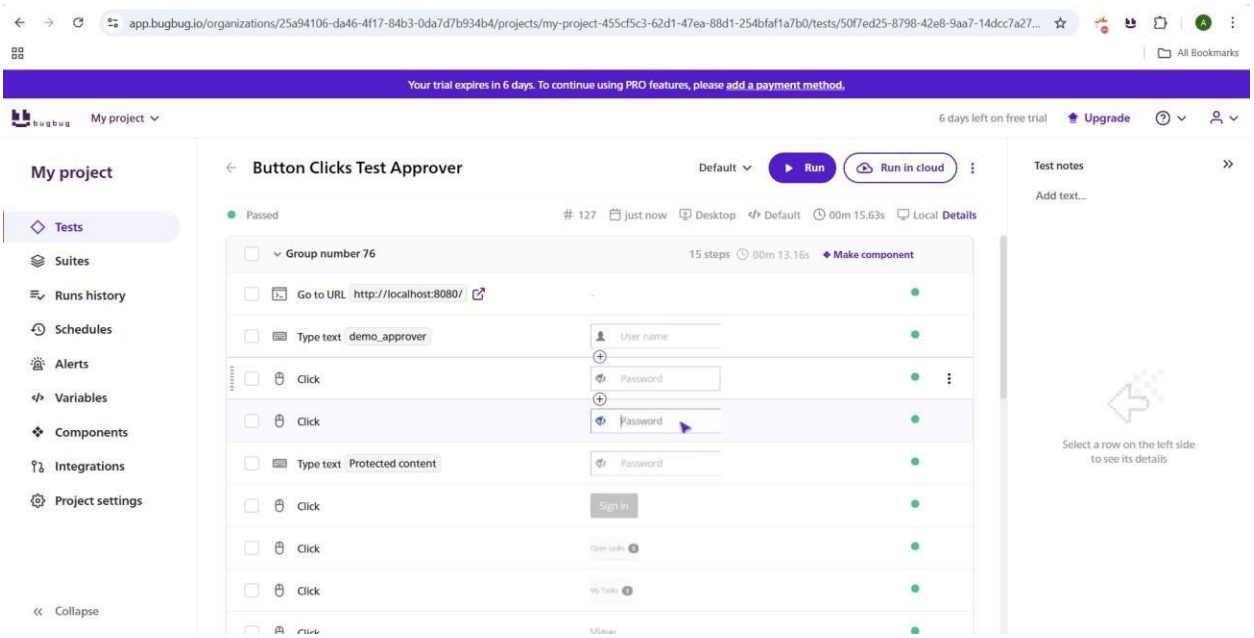
Button Tests Successfully Passed for Administrator



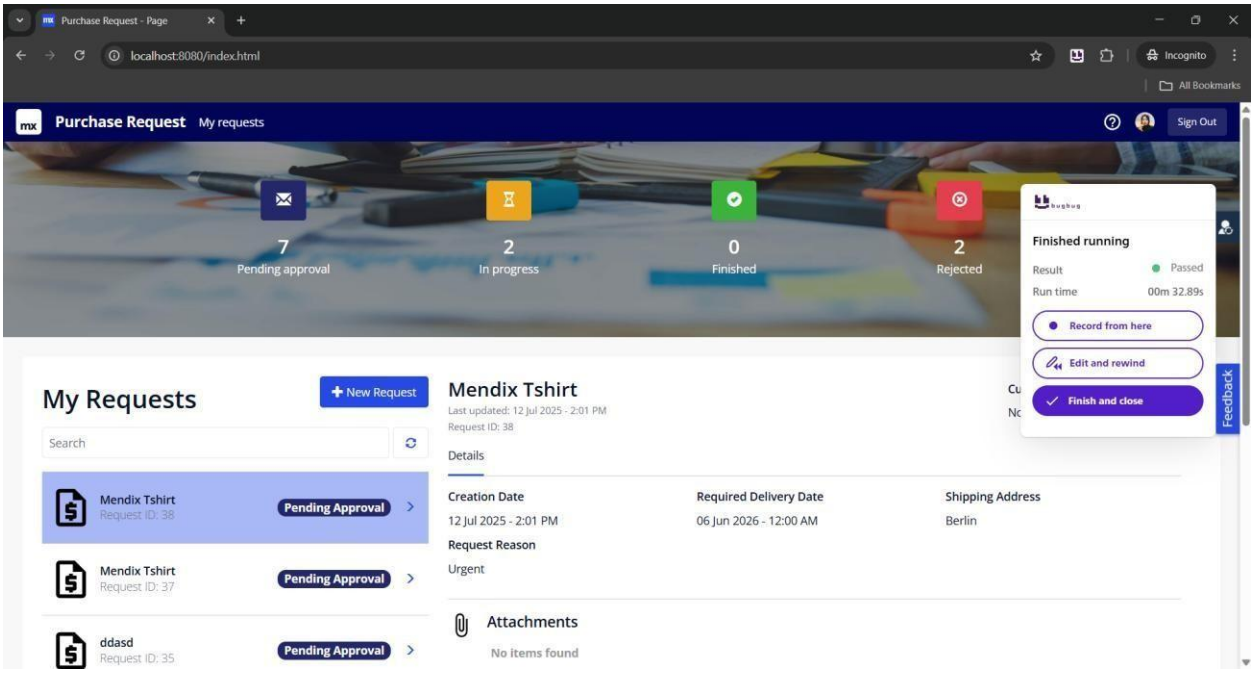
Assertion used for Button Clicks on Administrator account



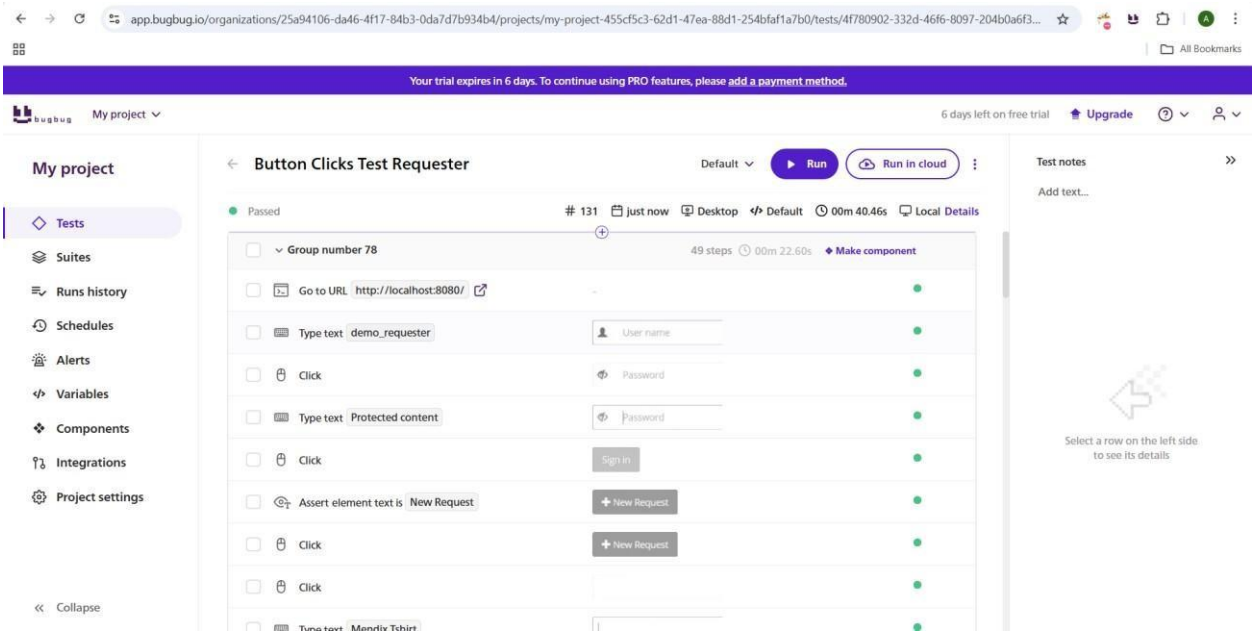
Button Clicks Test Successful on Approver



Assertions used for Approver



Button Clicks Test passed for requester user



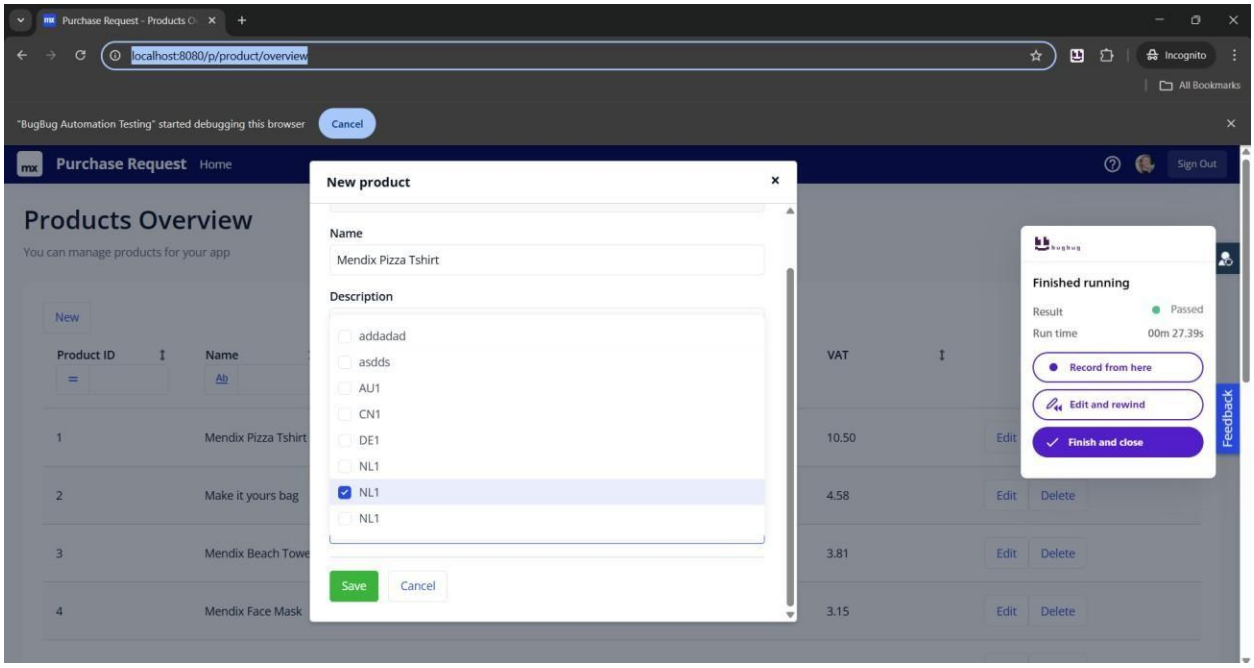
Assertions used for Requester user

- **Form Submission (Valid):** Passed
- **Form Submission (Invalid):** Passed

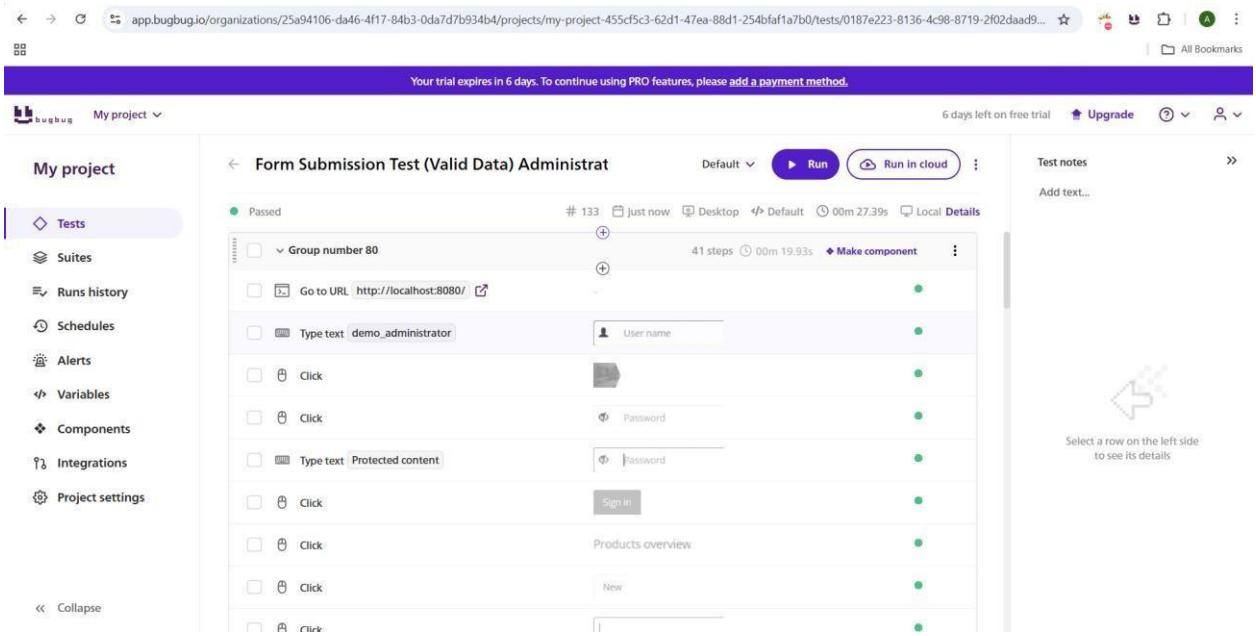
Risk: High if validation fails.

2.3 RBAC Tests

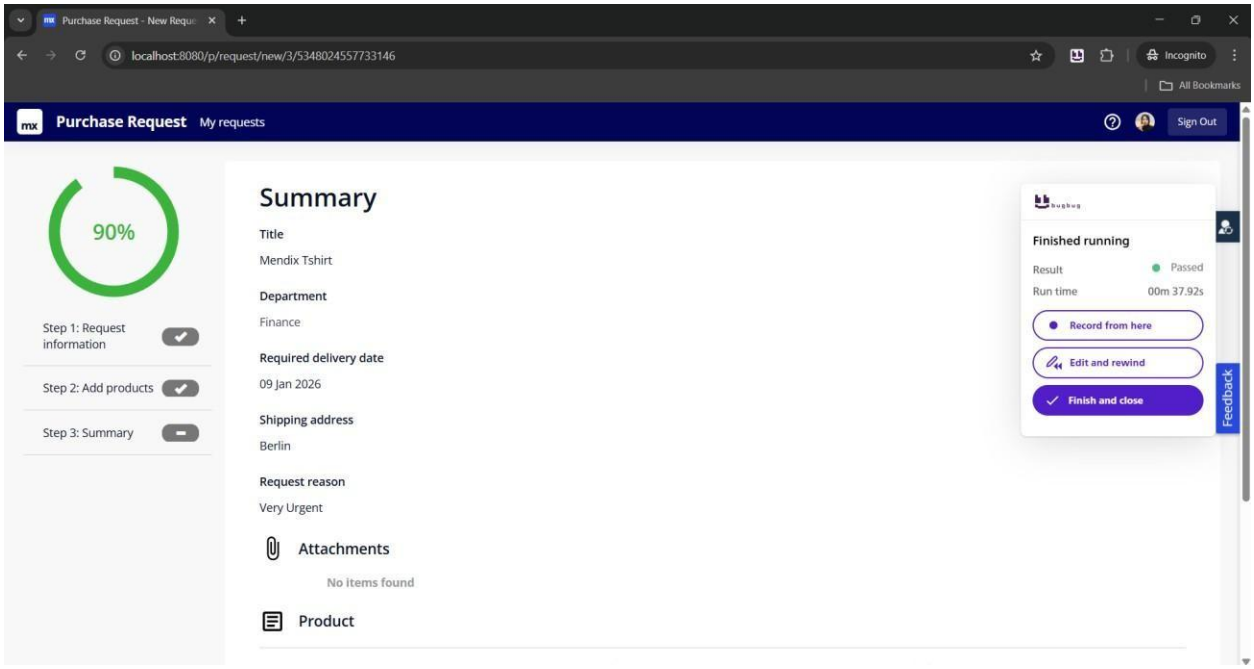
- Requester could not access Admin features.



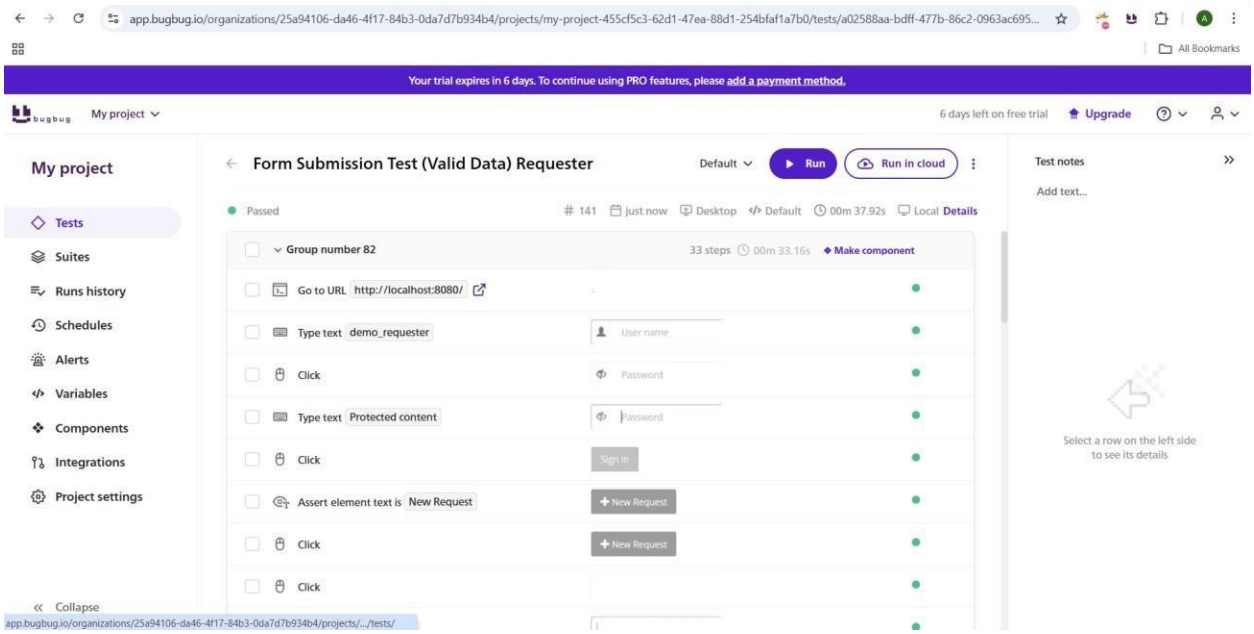
Form Submission Valid Data Administrator Passed



Assertions Used



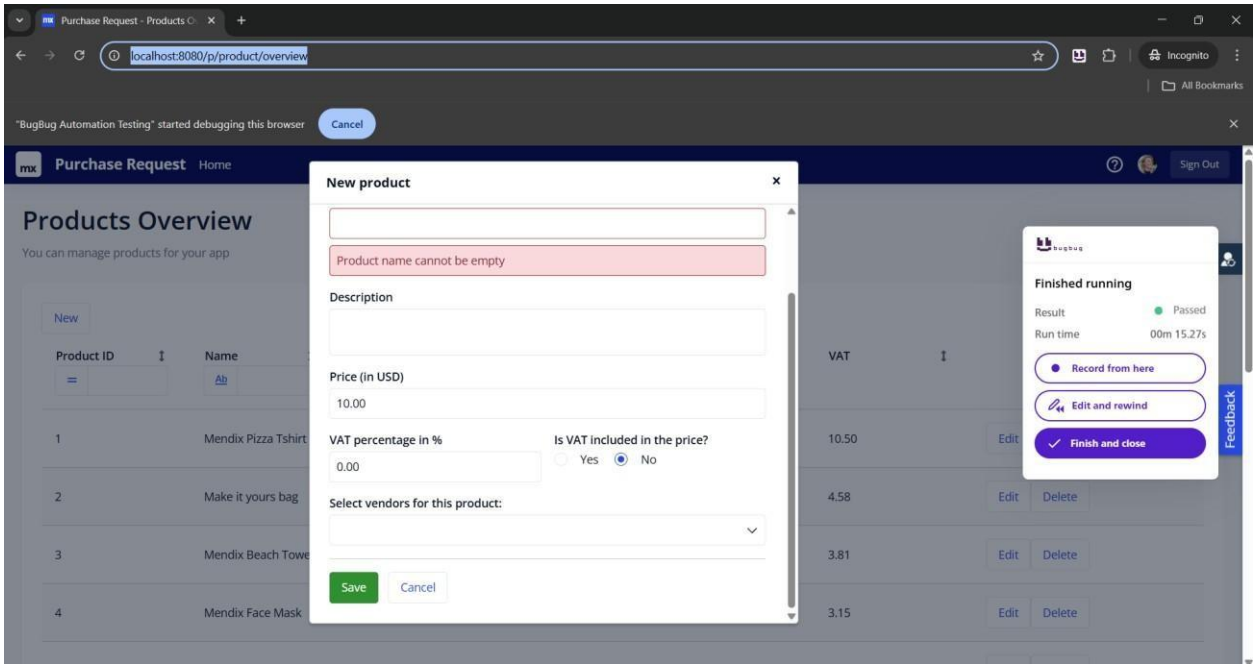
Form Submission Valid Data Requester Passed



Assertions used

Status: PASSED

C) Form Submission Test (Invalid/Missing Data)



Shows Error on Admin dashboard for missing data

Test Objective: Ensure appropriate validation and error handling for invalid or missing form data.

Steps:

- **Attempt submissions with invalid or missing data.**

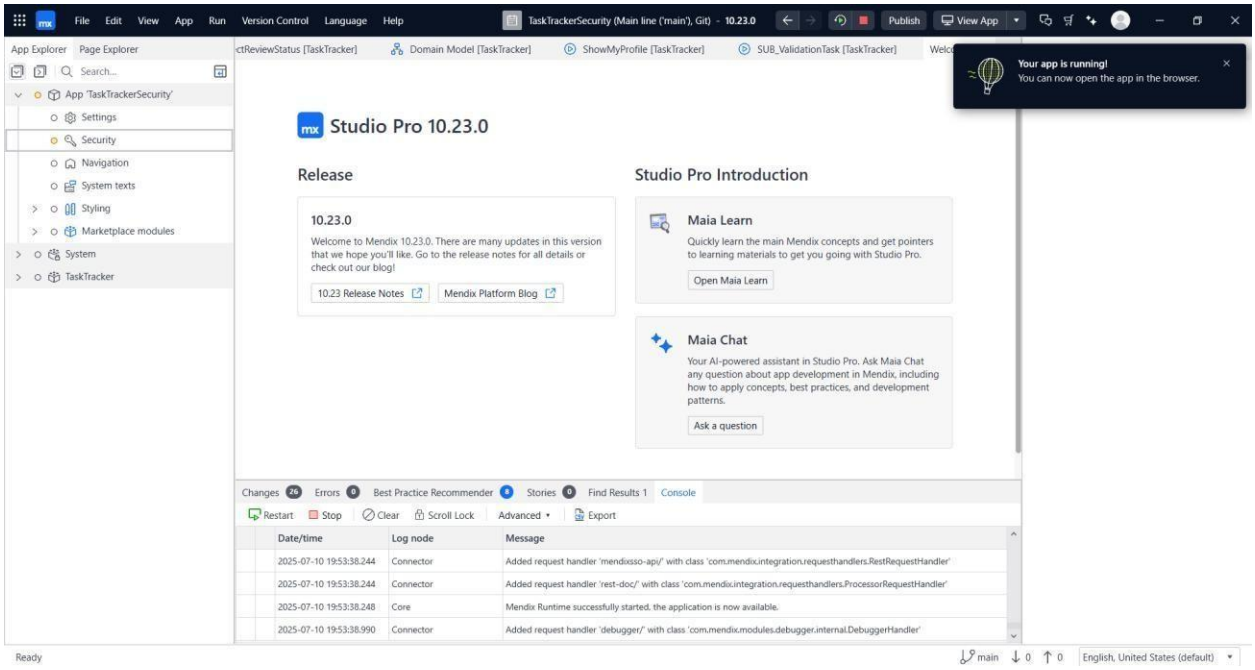
Status: PASSED

2.4 Recommendations

- Implement automated RBAC regression tests.
- Enhance data validation controls.
- Include BugBug automation in CI/CD.

3. Task Tracker Security App

3.1 Authentication Tests



Successful App Run From Mendix Platform

1. Authentication Tests

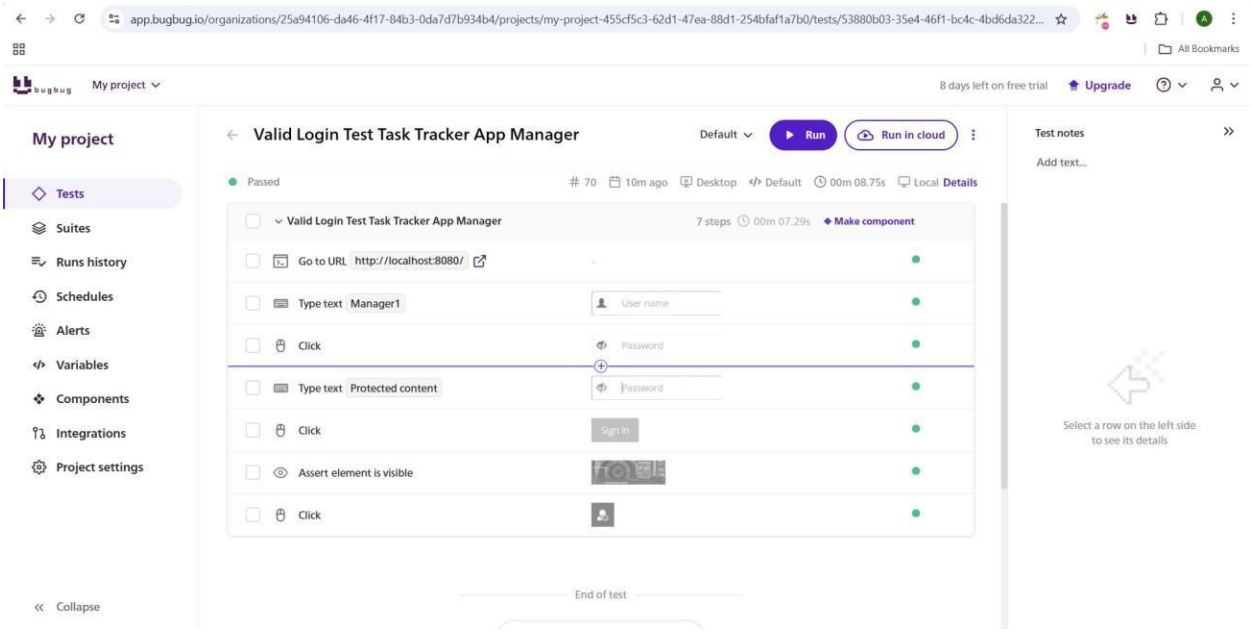
a) Valid Login

Test Objective:

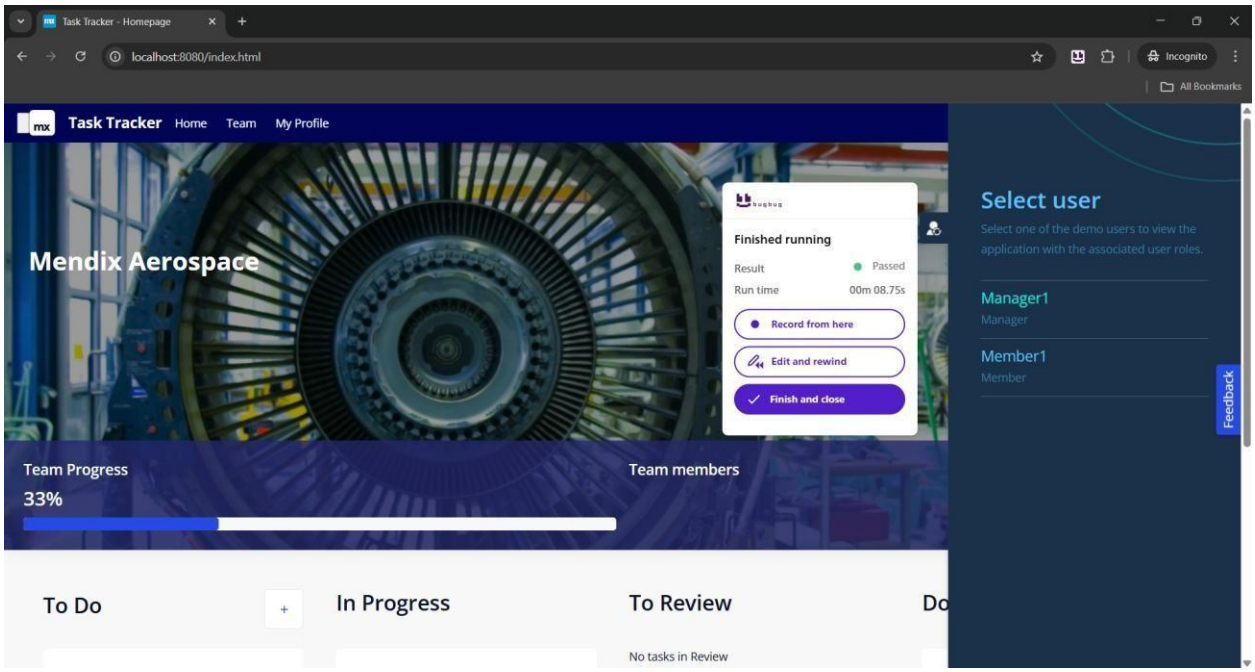
Ensure that users can log in successfully using valid credentials and access appropriate role-specific dashboards.

Steps:

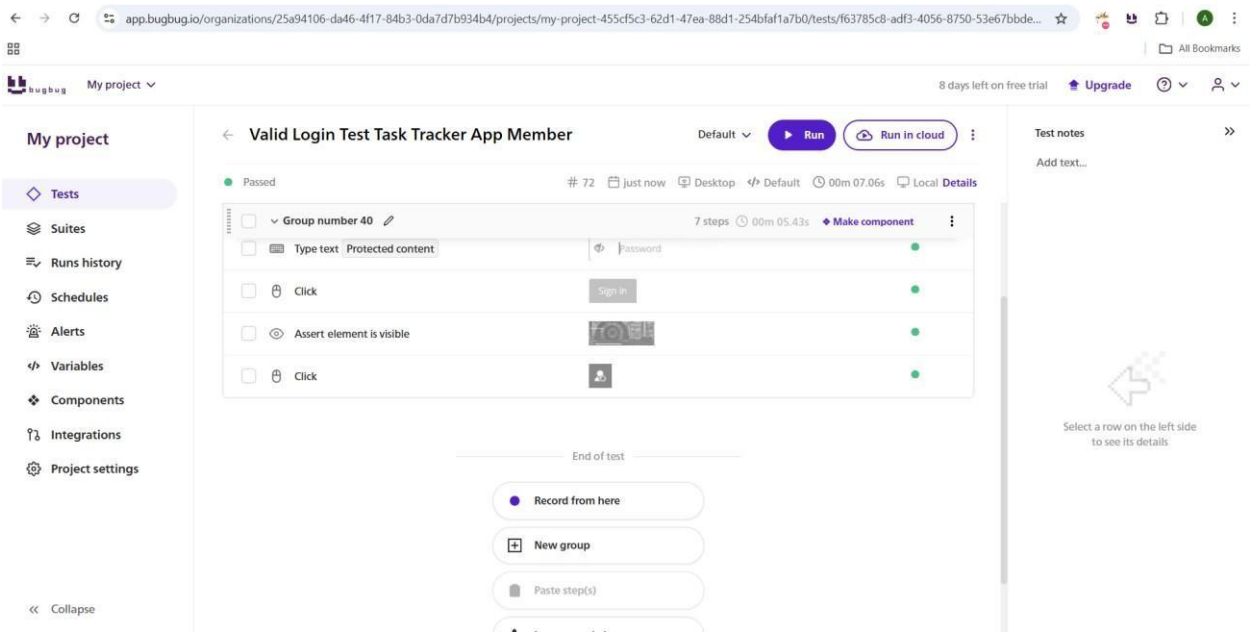
- **Log in as Manager1 (Manager role)**
- **Log in as Member1 (Member role)**
- **Confirm correct dashboard and access privileges**



Assertions Performed



Tests Passed



Test Assertions for Member

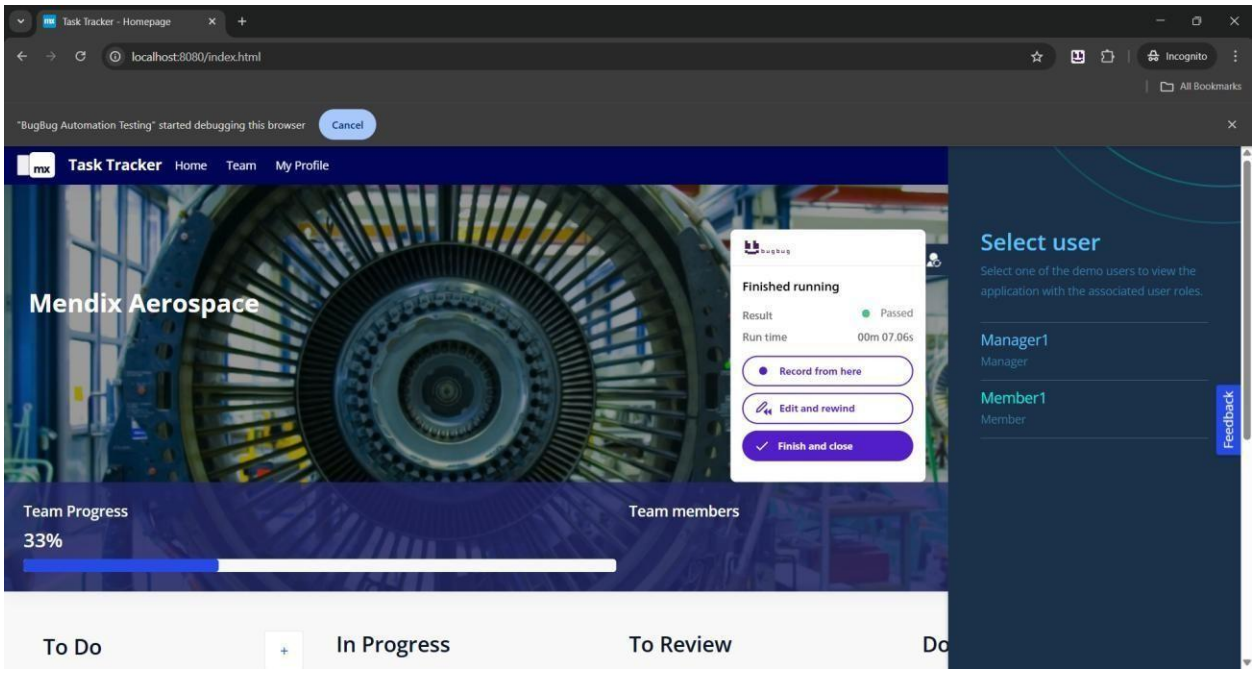


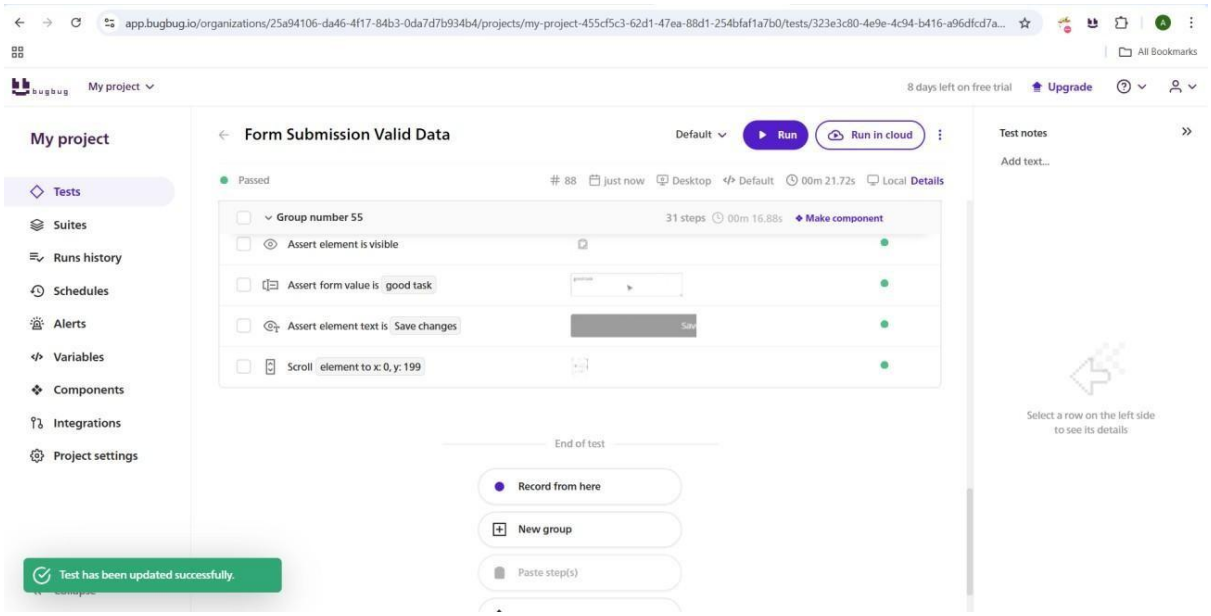
Fig 5: Test Passed for member login

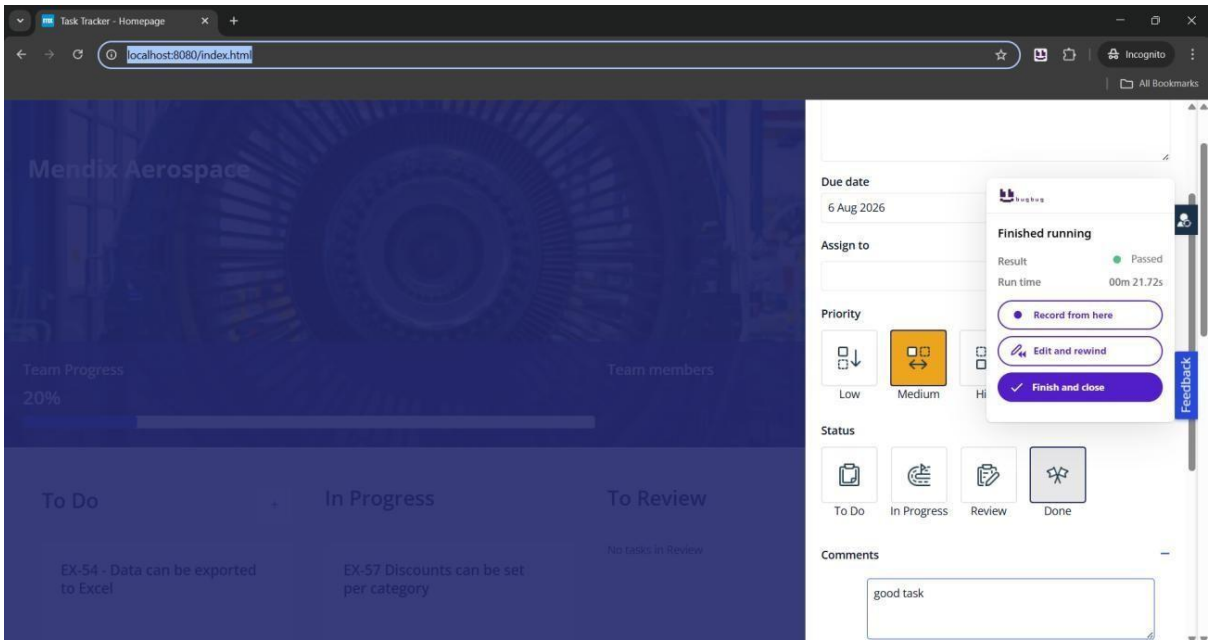
- **Valid Login:** Manager1 & Member1 – PASSED.
- **Invalid Login:** Error messages consistent – PASSED.
- **Logout:** Successful – PASSED.

3.2 UI Functional Tests

- Button clicks – PASSED.

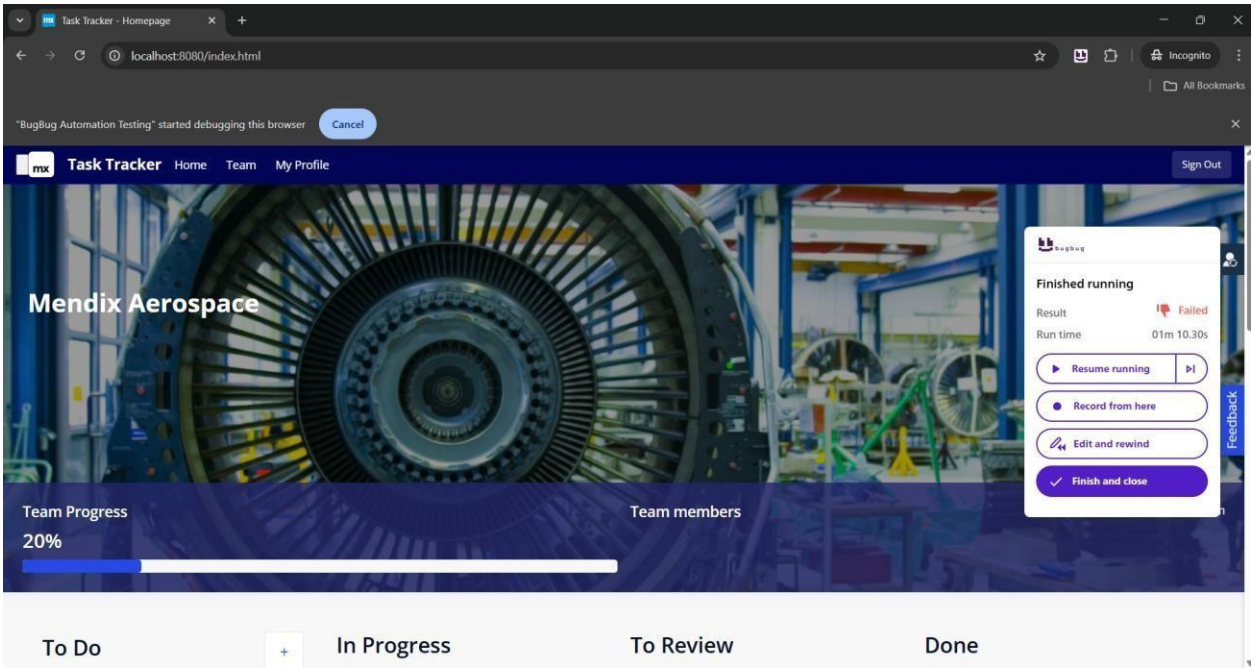
- Form submissions:
    - Valid – PASSED



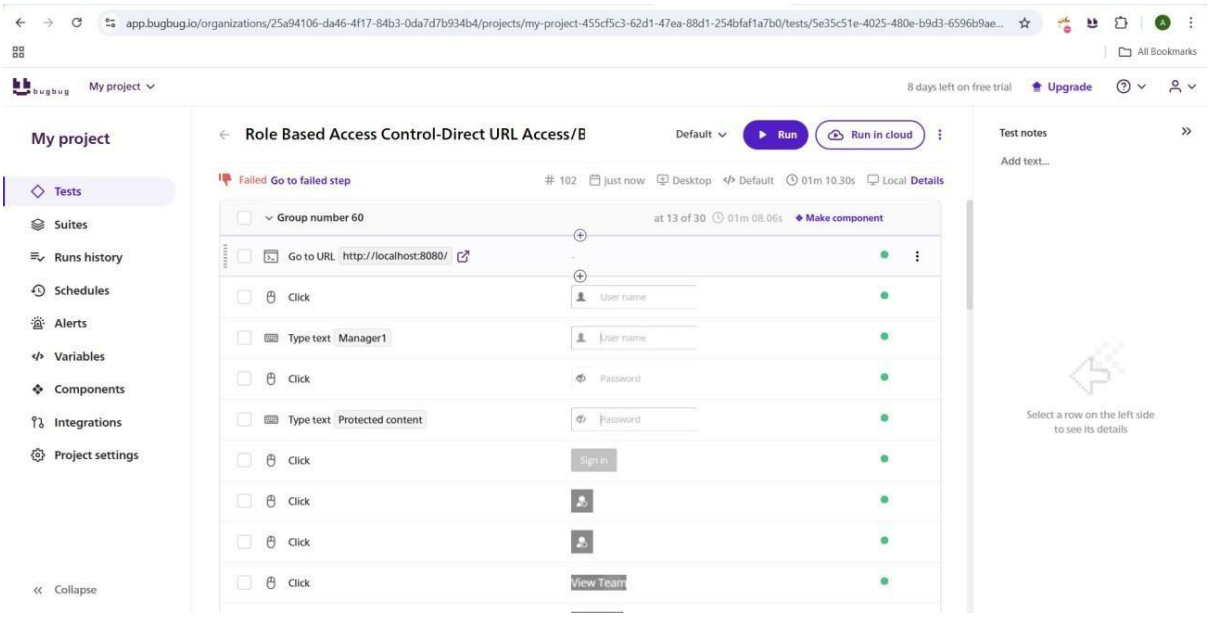


    - Invalid – PASSED

3.3 RBAC Tests



Role Based Access Control Failed- links can be bypassed



Result: FAILED – Member1 bypassed restrictions and accessed Manager-only pages.

Risk:

- Likelihood: High
- Impact: Critical

Recommendation:

- Immediately fix RBAC enforcement.
- Perform code audits & penetration testing.

- Add automated alerts for access violations.
- Integrate BugBug testing in CI/CD.
- Update scenarios regularly.
- Conduct manual penetration testing for high-risk modules.

Overall Recommendations (All Apps)

- Automate testing pipelines.
- Regularly update tests with feature changes.
- Save detailed logs/screenshots for audit.
- Strengthen RBAC with proactive monitoring.