

# 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:its-community
```

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

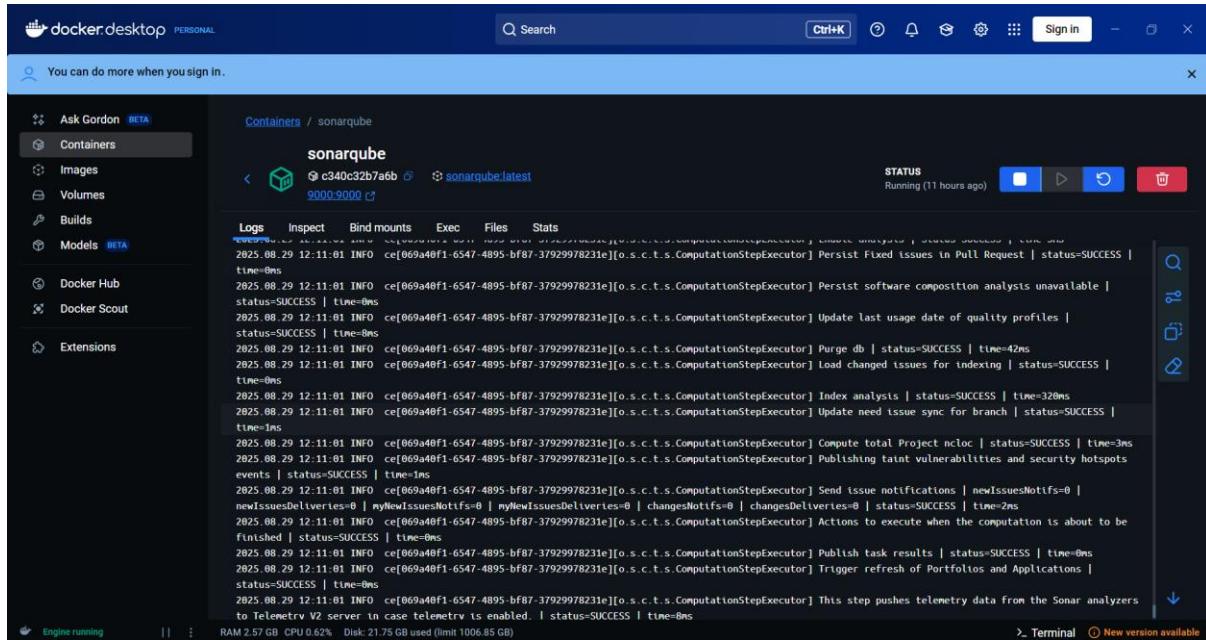


Figure 1: SonarQube Setup on Docker

### 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"
```

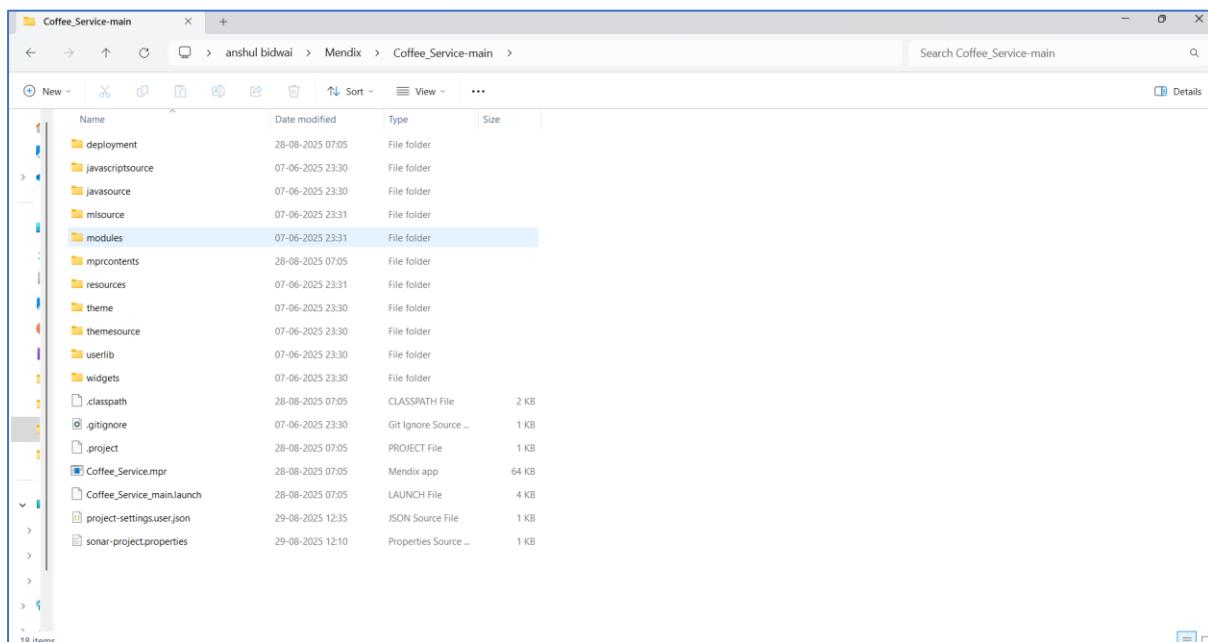
```
$env:SONAR_TOKEN = "squ_...paste-your-token..."
```

The screenshot shows the SonarQube security interface. At the top, there's a warning about using an embedded database. Below it, the navigation bar includes 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', 'Administration', and 'More'. The 'Administrator' profile is selected. The main section is titled 'Security' and contains two forms: 'Generate Tokens' and 'Enter a new password'. The 'Generate Tokens' form has fields for 'Name', 'Type' (dropdown), 'Expires in' (30 days), and a 'Generate' button. The table below lists tokens: 'mendix-sast' (User, created August 29, 2025, expires September 28, 2025). The 'Enter a new password' form has fields for 'Old Password\*' and 'New Password\*'. A 'Revoke' button is also present.

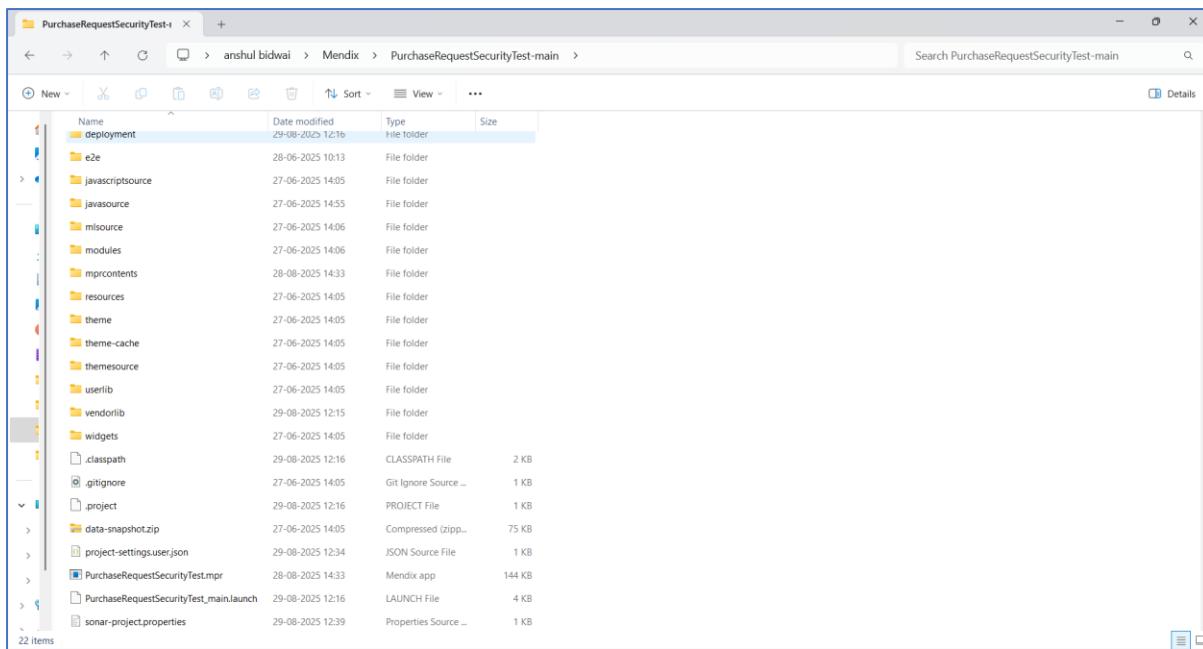
**Figure 2: Token Generation Screen**

## Step 5: Ensure Code Folders Exist

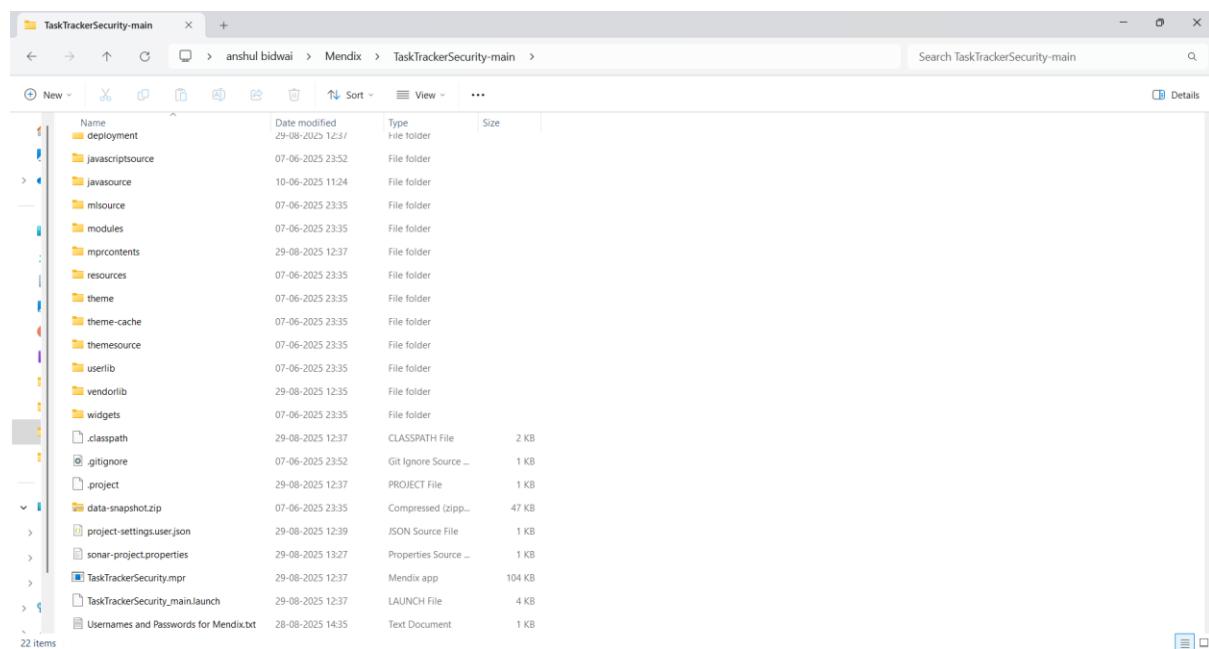
- Coffee: C:\Users\anshu\Mendix\Coffee\_Service-main



- 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

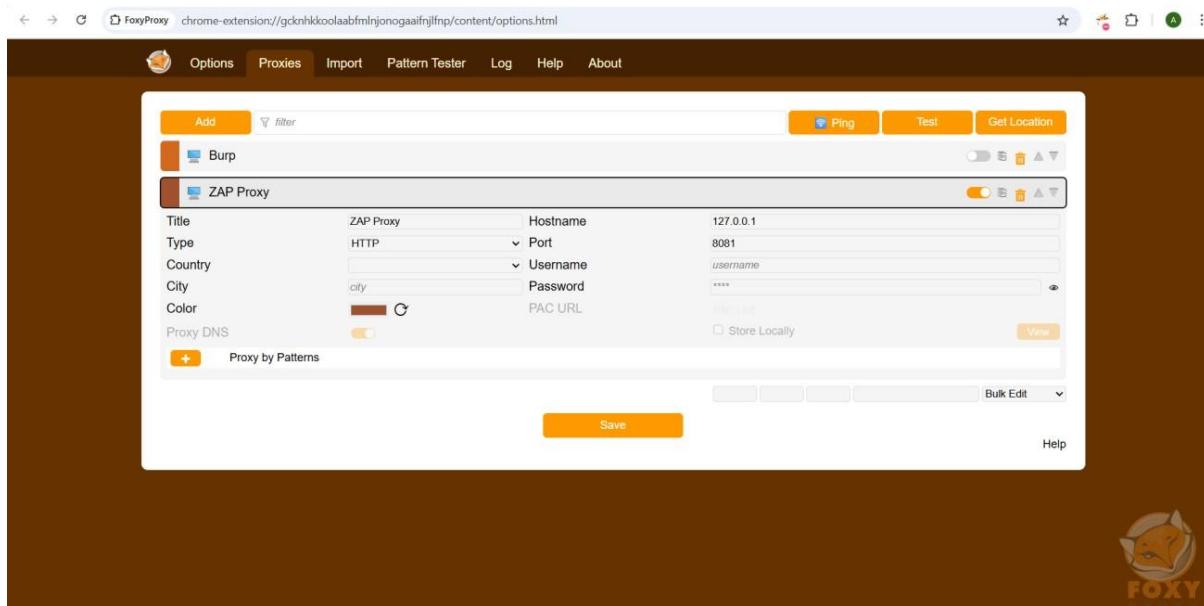


Fig. 1: Zap Proxy Setting Using FoxyProxy

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

A screenshot of the "Demo users" management interface. The top navigation bar includes tabs for "Module status", "User roles", "Administrator", "Demo users" (which is the active tab), "Anonymous users", and "Password policy".

Enable demo users  Yes  No

Demo users are only available when running locally or in a Free App.

New  Edit  Delete

User name	Entity	User roles
Customer Jack	Account	Customer
Customer Roxanne	Account	Customer
Engineer Bill	Account	Engineer
Engineer Cate	Account	Engineer

*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.

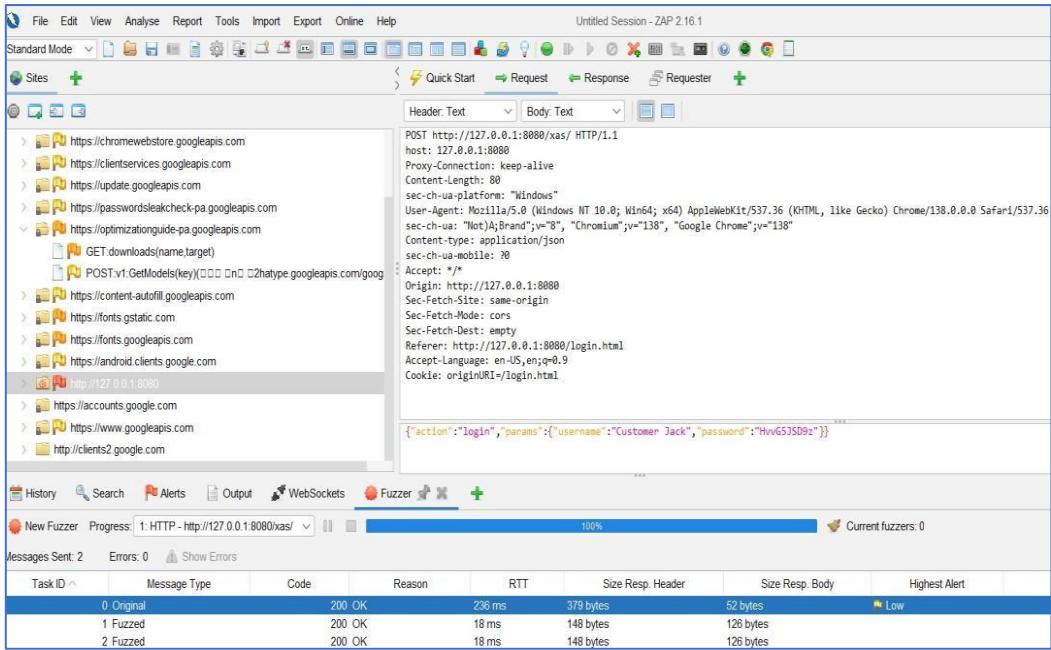
The screenshot shows the OWASP ZAP Fuzzer interface. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Import, Export, Online, and Help. The main window has tabs for Standard Mode, Sites, and Fuzzer. The Fuzzer tab is active, showing a list of URLs under the 'History' section. One URL is highlighted: `http://127.0.0.1:8080`. Below the history is a search bar containing the JSON payload: `{"action": "login", "parameters": {"username": "Customer Jack", "password": "Hv6G5J5D9z"}}`. At the bottom, there is a table titled 'Method' with columns for Method, URL, and Match. The table lists various HTTP requests and their corresponding matches.

Method	URL	Match
GET	<code>http://127.0.0.1:8080/manifest.webmanifest?638869801425941329</code>	127.0.0.1
GET	<code>http://127.0.0.1:8080/manifest.webmanifest?638869801425941329</code>	127.0.0.1
POST	<code>http://127.0.0.1:8080/xas/</code>	127.0.0.1
GET	<code>http://127.0.0.1:8080/index.html</code>	127.0.0.1
GET	<code>http://127.0.0.1:8080/index.html</code>	127.0.0.1
GET	<code>http://127.0.0.1:8080/index.html</code>	127.0.0.1
GET	<code>http://127.0.0.1:8080/mclient/system/mxui/mxui.js?638869801425941329</code>	127.0.0.1
GET	<code>http://127.0.0.1:8080/mclient/system/mxui/mxui.js?638869801425941329</code>	127.0.0.1
GET	<code>http://127.0.0.1:8080/mclient/system/mxui/mxui.js?638869801425941329</code>	127.0.0.1
GET	<code>http://127.0.0.1:8080/metamodel.json?638869801425941329</code>	127.0.0.1

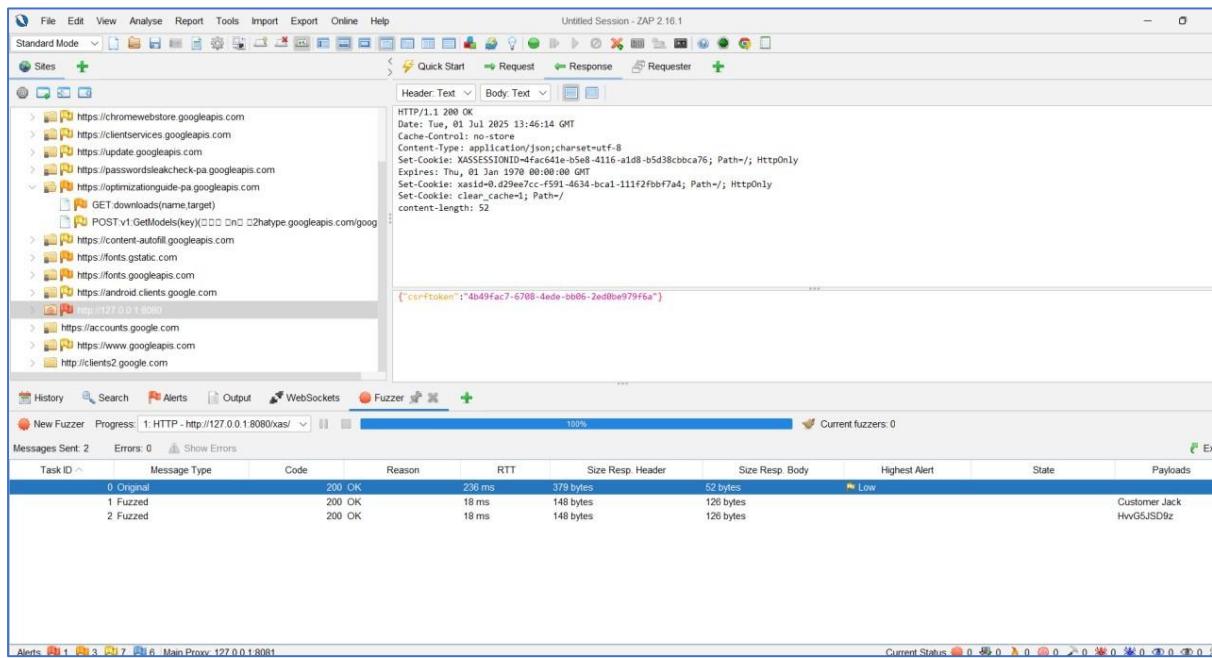
*Fig. 3: Fuzz Attack Performed*

### 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.

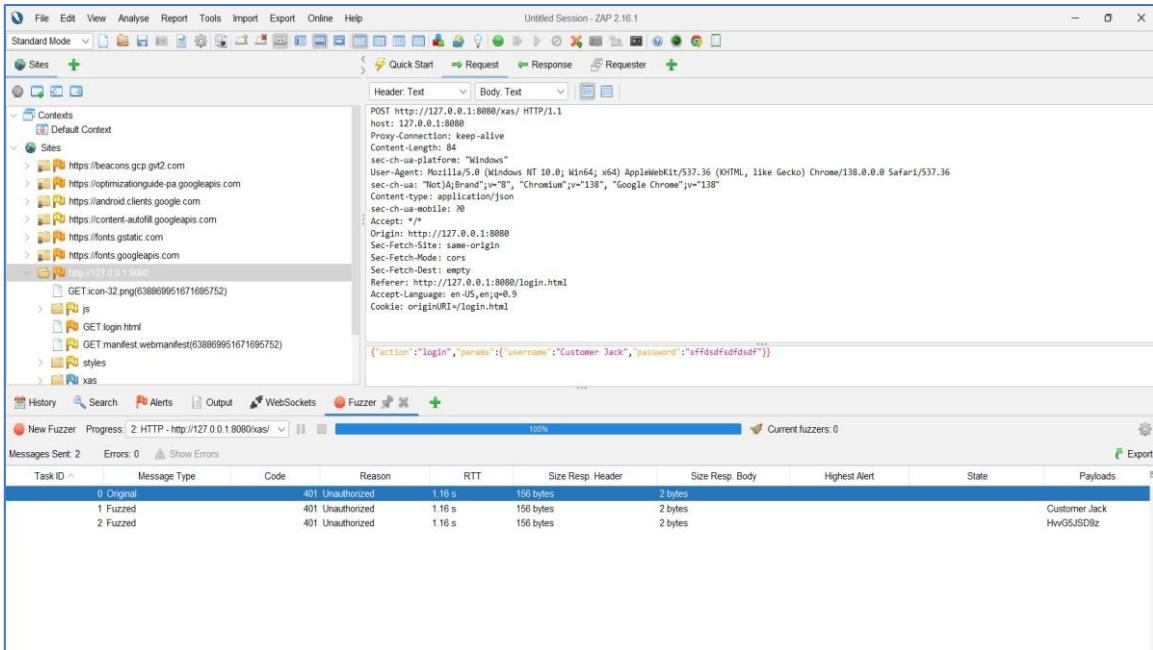


Fig. 6: Invalid Login Request

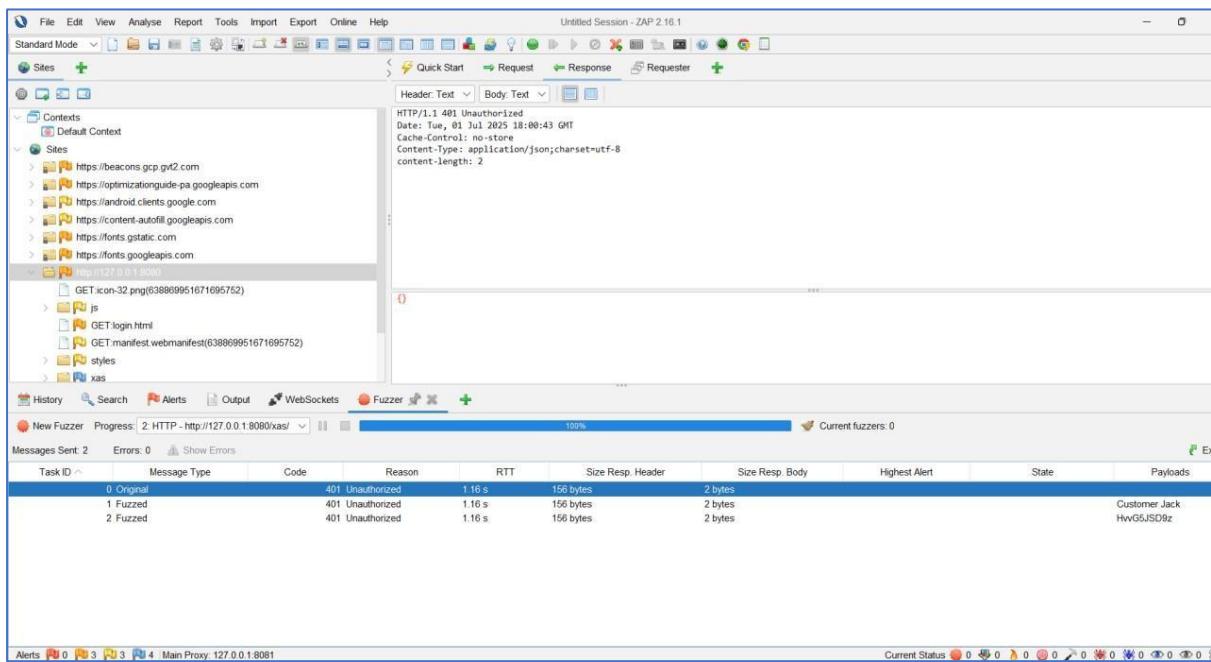


Fig. 7: 401 Unauthorized Response

## 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

The screenshot shows the ZAP interface with the 'Request' tab selected. A specific line of the request payload is highlighted, showing the extracted session cookie: XASSESSIONID=de1d8728-6a0b-4d74-9126-0bd3a5b9cea8; xsid=0.102128dc-ac3f-4786-ac39-dbaa5de0246e. Below this, the 'History' tab displays a list of requests to 'http://127.0.0.1:8080/login.html', all sharing the same session ID.

Fig. 8: Extracted Session Cookie

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

The screenshot shows the Google Chrome DevTools Application tab open. In the 'Storage' section under 'Cookies', a new cookie named 'XASSESSIONID' is listed with the value 'de1d8728-6a0b-4d74-9126-0bd3a5b9cea8'. This cookie was previously extracted from a session and is now being used in a new browser session.

*Fig. 9: Setting Session Cookie*

### 3. Attempted to access protected resources.

The screenshot shows a browser window with the URL `127.0.0.1:8080`. The page content is a login form for 'Mendix' with the tagline 'Win in a coffee-driven world.' and a heart-shaped arrangement of coffee beans with a cup of coffee on top. The developer tools sidebar is open, specifically the 'Application' tab under 'Storage'. The 'Cookies' section is expanded, showing a table of cookies. One cookie, 'XASSESSIONID', is highlighted with a value of `b794a227-2802-43b0-b02a-f5be56dc00ef`. The table includes columns for Name, Value, Do..., Path, Exp..., Size, Htt..., Sec..., Sess..., Part..., Cro..., and P... . Other visible cookies include 'DeviceType' (Value: Desktop), 'originURI' (Value: /login.html), 'Profile' (Value: Responsive), 'SessionTimeZoneOffset' (Value: -120), and 'xasid' (Value: 0.102188dc-ac3f-4786-ac39-dba...).

*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.

Processed	Method	URI	Flags
✓	GET	http://127.0.0.1:8080/	
✓	GET	http://127.0.0.1:8080/robots.txt	Seed
✓	GET	http://127.0.0.1:8080/sitemap.xml	Seed
✓	GET	http://127.0.0.1:8080/styles/web/css/main.css?638870570971915787	
✓	GET	http://127.0.0.1:8080/apple-touch-icon.png?638870570971915787	
✓	GET	http://127.0.0.1:8080/icon-32.png?638870570971915787	
✓	GET	http://127.0.0.1:8080/icon-16.png?638870570971915787	
✓	GET	http://127.0.0.1:8080/manifest.webmanifest?638870570971915787	
✓	GET	http://127.0.0.1:8080/mxclient/system/mxmlmxui.js?638870570971915787	
✓	GET	http://127.0.0.1:8080/bootstrap.com/	Out of Scope
✓	GET	https://github.com/twbs/bootstrap/blob/master/LICENSE	Out of Scope

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.

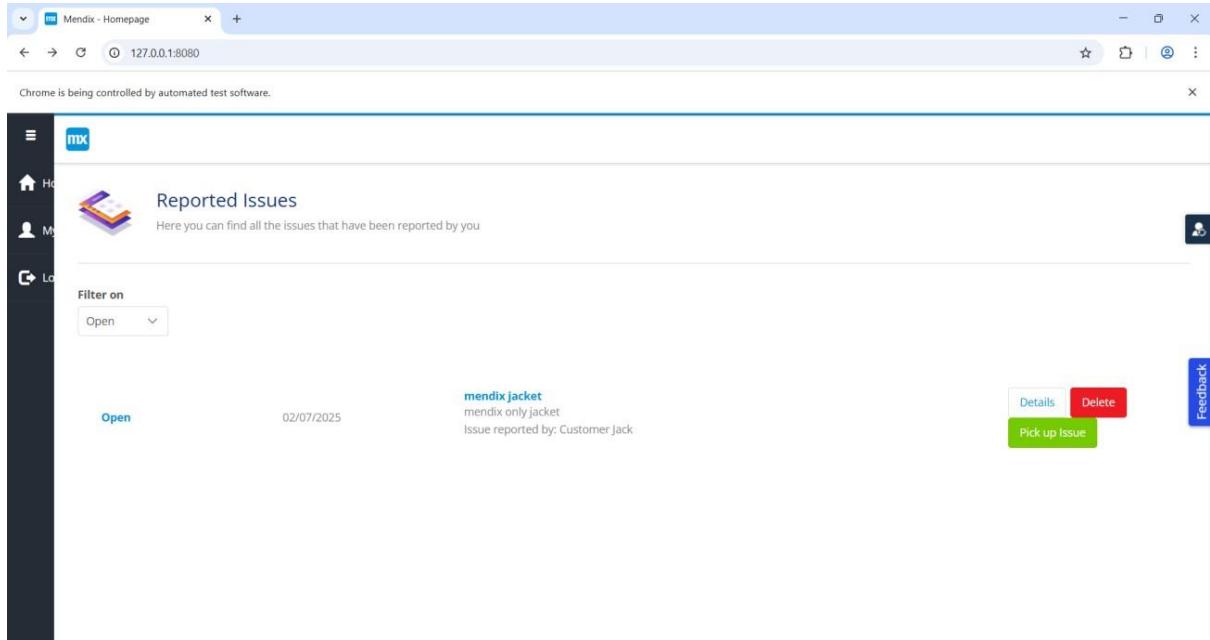
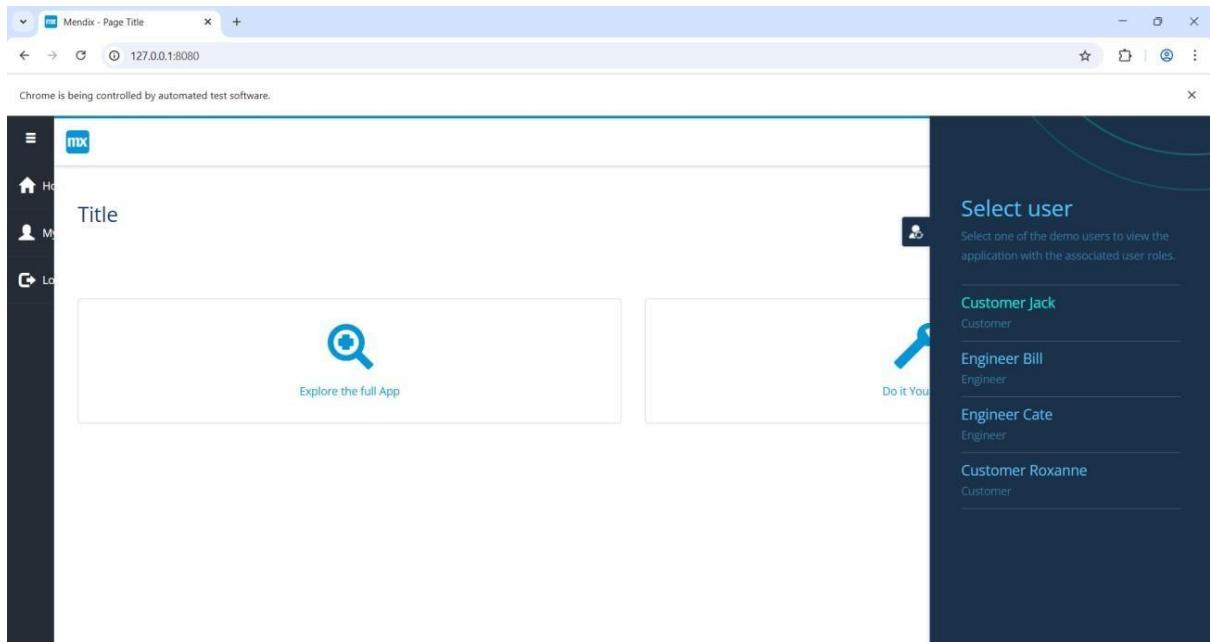
## C. Role-Based Access Control (RBAC) and IDOR

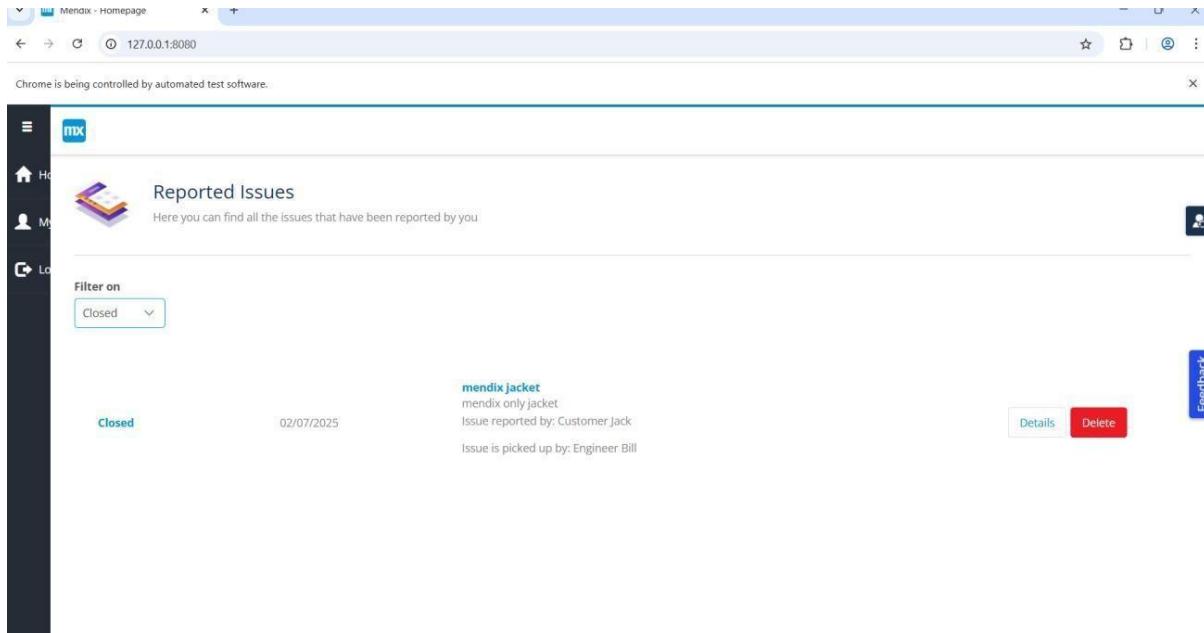
### 1. RBAC Testing

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

**Steps:**

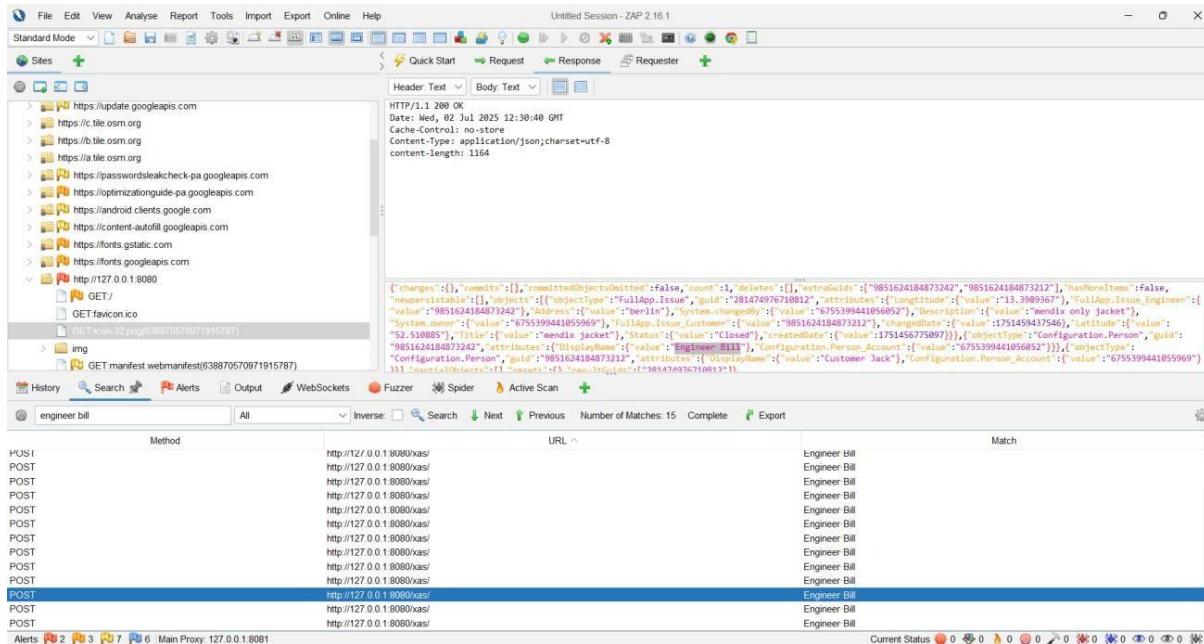
1. Logged in as each role and recorded accessible URLs.



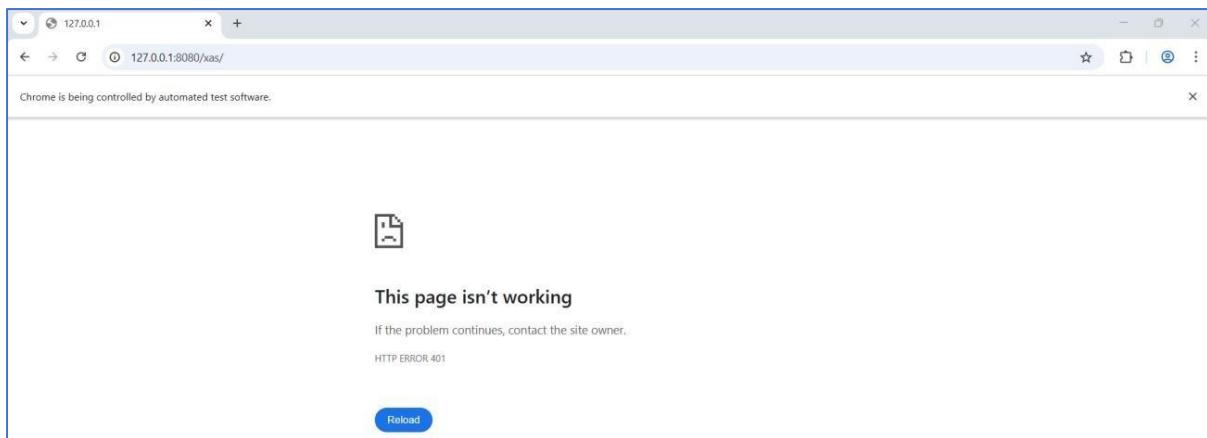


*Fig. 13: Customer Jack Dashboard, Customer Jack Task, New Task Added – Customer Jack*

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

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



**ZAP by  
Checkmarx 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
<a href="#">Hash Disclosure - BCrypt</a>	High	6
<a href="#">PII Disclosure</a>	High	1
<a href="#">Content Security Policy (CSP) Header Not Set</a>	Medium	10
<a href="#">Cross-Domain Misconfiguration</a>	Medium	2
<a href="#">Directory Browsing</a>	Medium	1
<a href="#">Missing Anti-clickjacking Header</a>	Medium	10
<a href="#">Spring Actuator Information Leak</a>	Medium	1
<a href="#">Cookie No HttpOnly Flag</a>	Low	1
<a href="#">Cookie without SameSite Attribute</a>	Low	7
<a href="#">Information Disclosure - Debug Error Messages</a>	Low	1
<a href="#">Server Leaks Version Information via "Server" HTTP Response Header Field</a>	Low	1
<a href="#">Strict-Transport-Security Header Not Set</a>	Low	22

---



---

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

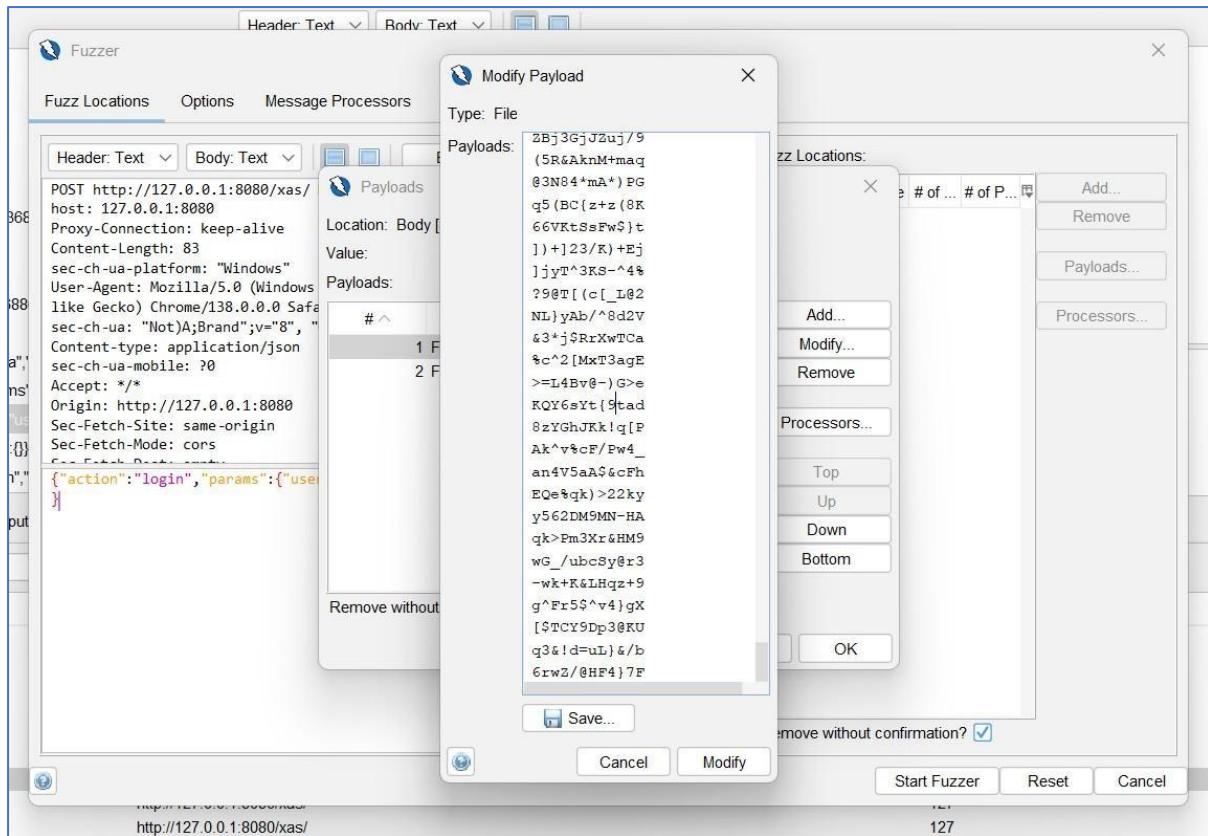
- Tracked status codes for each attempt.

#### 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.

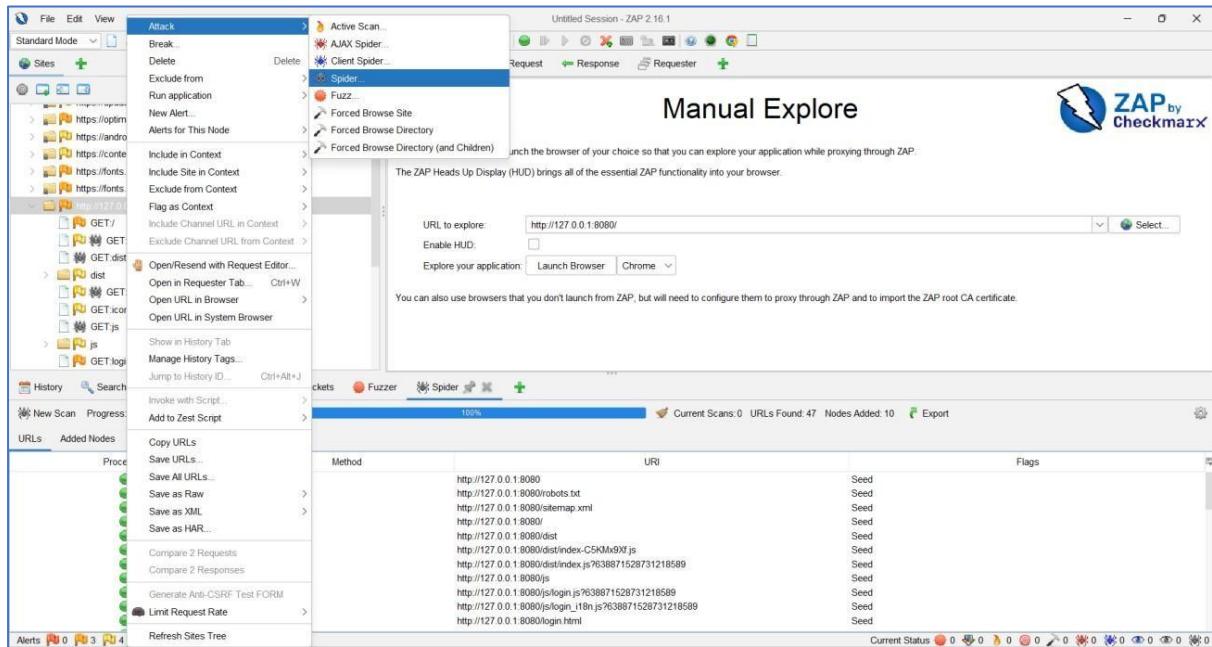


Fig. 17: Spider in Progress

## 3. Reviewed discovered URLs

The screenshot shows the ZAP 2.16.1 interface. The top navigation bar includes File, Edit, View, Analyse, Report, Tools, Import, Export, Online, Help, Standard Mode, and a toolbar with various icons. The main window has tabs for Quick Start, Request, Response, and Requester. The central area displays the "Manual Explore" screen with a green lightning bolt icon. It contains fields for "URL to explore" (http://127.0.0.1:8080), "Enable HUD" (unchecked), and "Explore your application" (Launch Browser, Chrome selected). Below this, a message states: "This screen allows you to launch the browser of your choice so that you can explore your application while proxying through ZAP. The ZAP Heads Up Display (HUD) brings all of the essential ZAP functionality into your browser." A note also says: "You can also use browsers that you don't launch from ZAP, but will need to configure them to proxy through ZAP and to import the ZAP root CA certificate." At the bottom of the central area, there are buttons for "Select", "Current Scans: 0 URLs Found: 47 Nodes Added: 10", and "Export". The bottom of the interface shows a progress bar at 100%, a status bar with "New Scan Progress: 0 http://127.0.0.1:8080", and a "Main Proxy: 127.0.0.1:8081". The bottom-most section is the "Spider" results table, which lists 47 URLs found during the scan.

Processed	Method	URI	Flags
	GET	http://127.0.0.1:8080	Seed
	GET	http://127.0.0.1:8080/robots.txt	Seed
	GET	http://127.0.0.1:8080/sitemap.xml	Seed
	GET	http://127.0.0.1:8080/	Seed
	GET	http://127.0.0.1:8080/dist	Seed
	GET	http://127.0.0.1:8080/dist/index-C5KMx9Jf.js	Seed
	GET	http://127.0.0.1:8080/dist/index.js?638871528731218589	Seed
	GET	http://127.0.0.1:8080/js	Seed
	GET	http://127.0.0.1:8080/js/login.js?638871528731218589	Seed
	GET	http://127.0.0.1:8080/js/login_i18n.js?638871528731218589	Seed
	GET	http://127.0.0.1:8080/login.html	Seed
	GET	http://127.0.0.1:8080/login.html	Seed

Fig. 18: Spider Results

#### 4. Attempted direct access to restricted pages.

This screenshot is identical to Fig. 18, showing the ZAP 2.16.1 interface in Standard Mode. The central "Manual Explore" screen and the "Spider" results table are displayed, showing the same 47 URLs found during the scan.

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
	r	r	n	
/	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

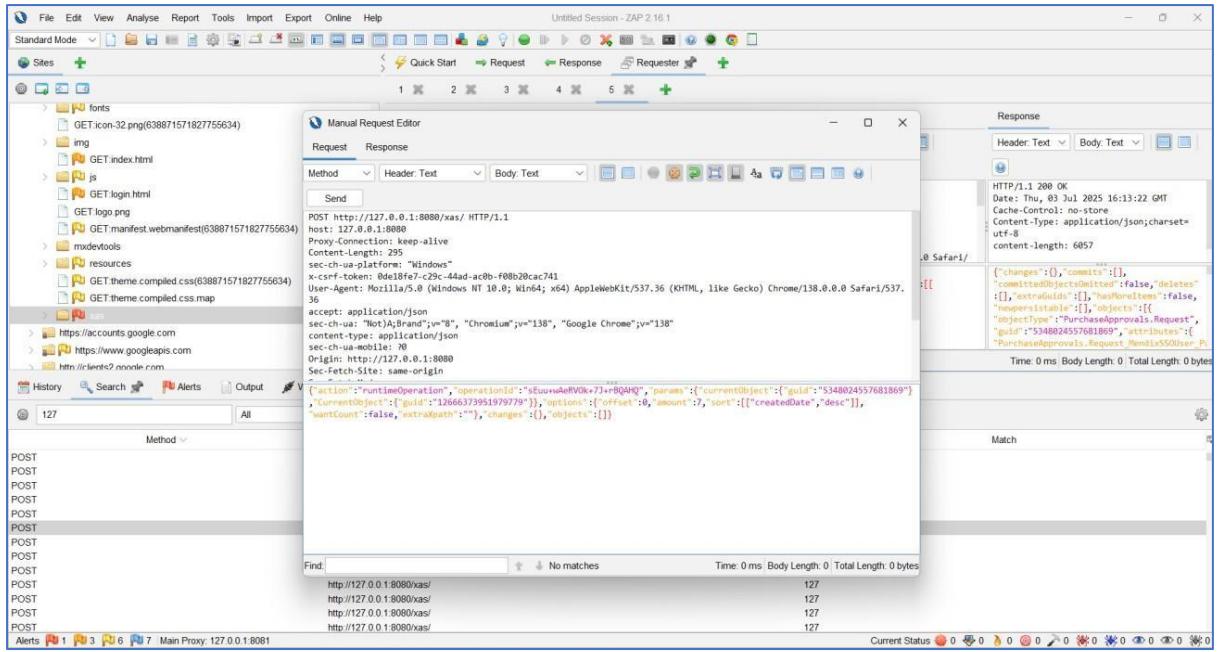
- Logged in as demo\_requester and captured a POST /xas/ request using OWASP ZAP.

The screenshot shows the OWASP ZAP interface with the following details:

- Sessions:** Shows various files and resources under a tree view, including fonts, img, js, and resources.
- Header:** Headers for the captured request:
 

```
POST http://127.0.0.1:8080/xas/ HTTP/1.1
host: 127.0.0.1:8080
Connection: keep-alive
Content-Length: 29
sec-ch-ua-platform: "Windows"
x-csrf-token: 0de15fe7-c29c-44ad-ac0b-f08b20cac741
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
accept: application/json
sec-ch-ua-device: "Brand" v="8", "Chromium" v="138", "Google Chrome" v="138"
Content-type: application/json
sec-ch-ua-mobile: ?0
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Header: accept
Referer: http://127.0.0.1:8080/index.html
Accept-language: en-US,en;q=0.9
Cookie: originURL=/login.html; XASSESSIONID=d9593c33-cc14-400d-893f-fb94ea6a9eb3; xsid=0.53477c8f-c7d6-466e-a62b-f01f473c34bd; SessionTimeZoneOffset=-120;
DeviceType=Desktop; Profile=Responsive
```
- Request Body:** Contains a JSON object representing a runtime operation:
 

```
{"action": "runTimeOperation", "operationId": "sEuuhuleIV0k+73+rQ4HQ", "params": [{"currentObject": {"guid": "12666373951979779"}, "CurrentObject": {"guid": "12666373951979779"}}], "options": [{"offset": "0", "amount": "17", "sort": [{"createdDate": "desc"}]}, {"wantCount": false, "extraPath": ""}], "changes": {}, "objects": []}
```
- Output:** Shows a list of captured requests, all of which have a status of 127.
- Bottom Status Bar:** Shows current status metrics: 0 errors, 0 warnings, 0 info, 0 alerts, 0 notifications, and 0 security issues.



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

## 2. Remove Authentication Headers

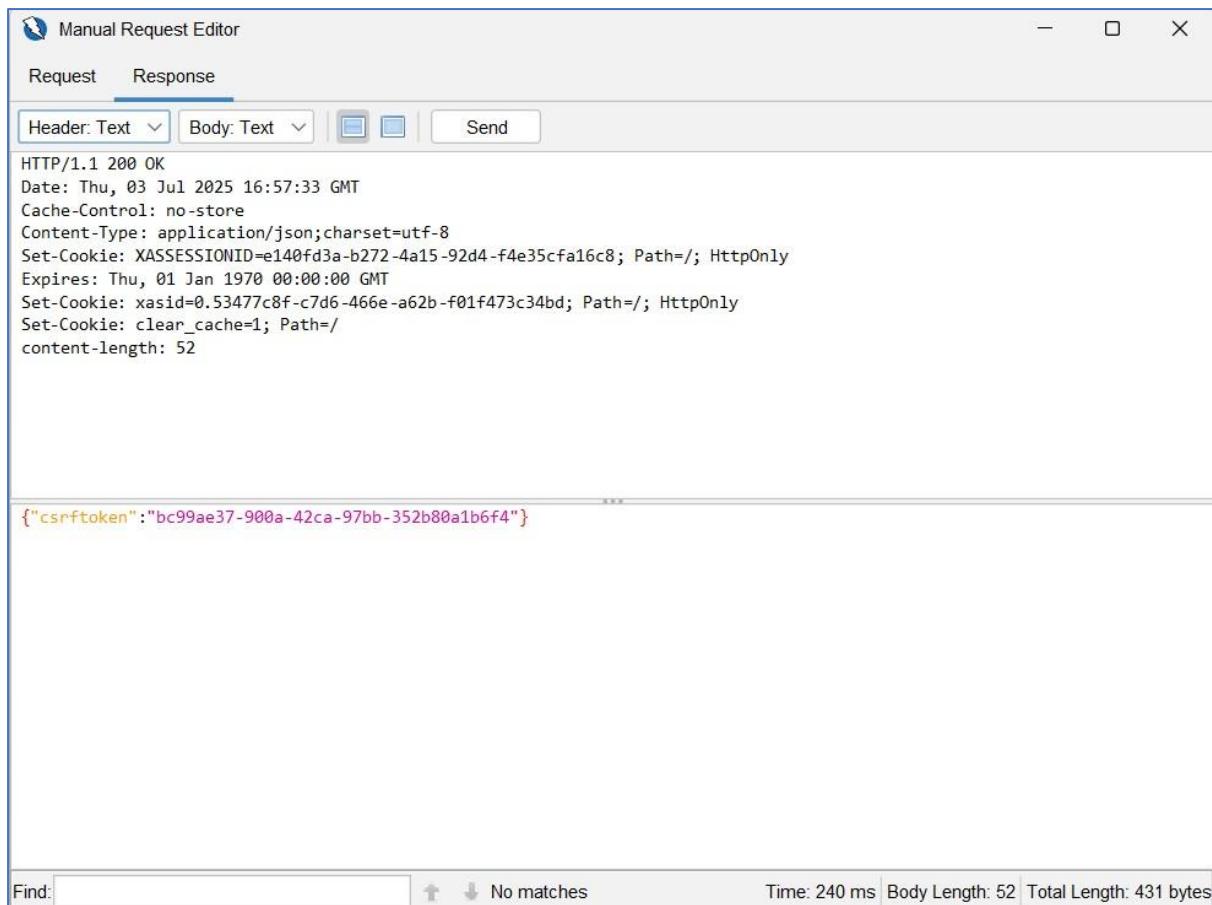
- Cleared all cookies and tokens in ZAP's Manual Request Editor.

## 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.

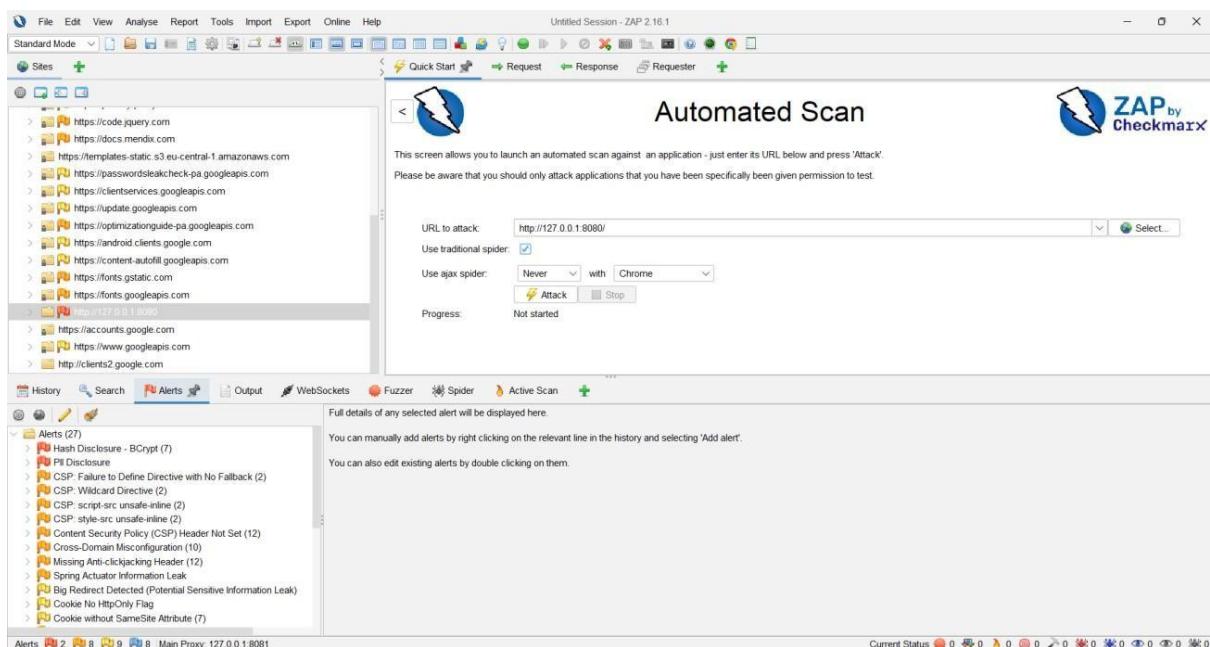
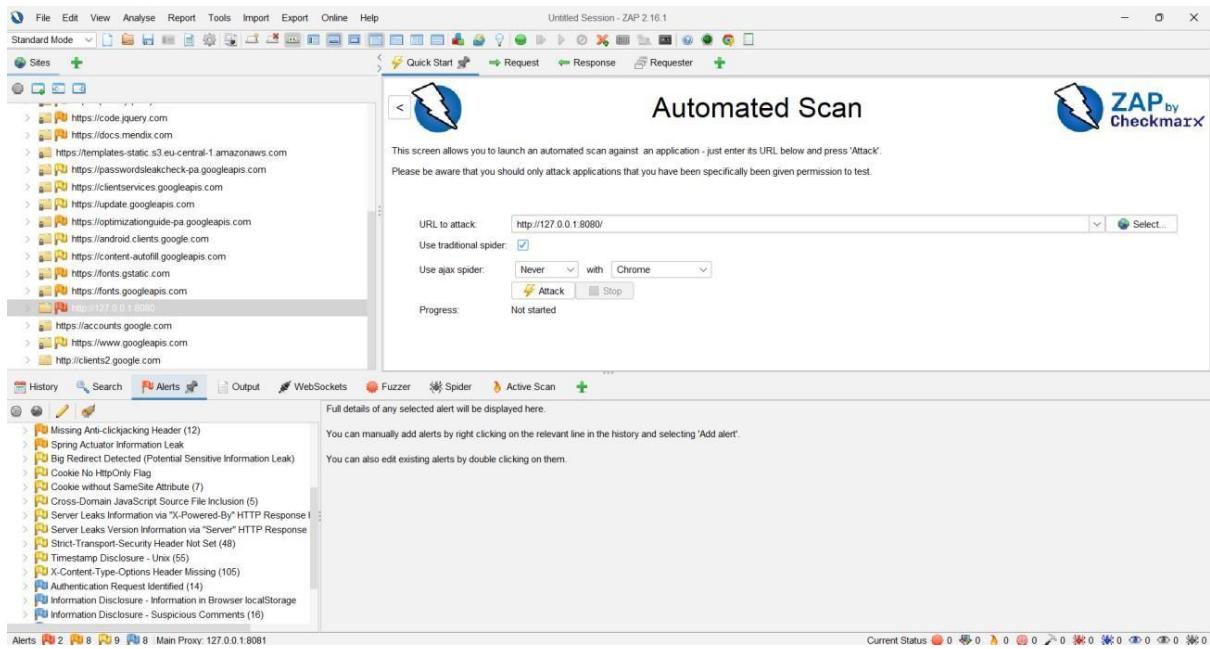


Fig. 21: ZAP Active Scan Progress Window

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

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

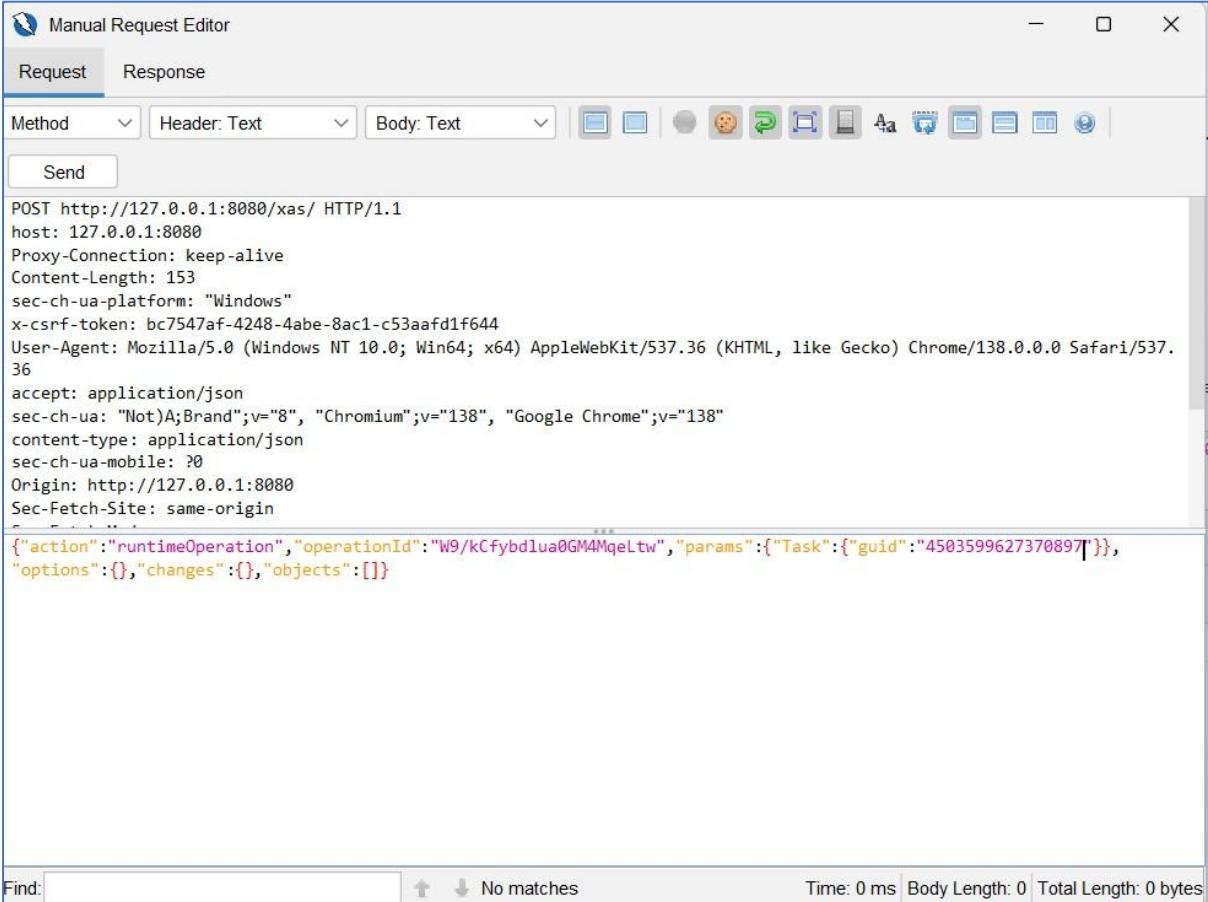
## 2.2 Insecure Direct Object Reference (IDOR)

### 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.



The screenshot shows the 'Manual Request Editor' interface. The 'Request' tab is selected. The 'Method' dropdown is set to 'POST'. The 'Header: Text' dropdown is set to 'Text'. The 'Body: Text' dropdown is set to 'Text'. Below these are various icons for file operations like Open, Save, Copy, Paste, etc. A 'Send' button is visible. The main body of the window contains an API request:

```
POST http://127.0.0.1:8080/xas/ HTTP/1.1
host: 127.0.0.1:8080
Proxy-Connection: keep-alive
Content-Length: 153
sec-ch-ua-platform: "Windows"
x-csrf-token: bc7547af-4248-4abe-8ac1-c53aaafdf1f644
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
accept: application/json
sec-ch-ua: "Not)A;Brand";v="8", "Chromium";v="138", "Google Chrome";v="138"
content-type: application/json
sec-ch-ua-mobile: ?0
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
{
  "action": "runTimeOperation",
  "operationId": "W9/kCfybdluu0GM4MqeLtw",
  "params": {
    "Task": {
      "guid": "4503599627370897"
    }
  },
  "options": {},
  "changes": {},
  "objects": []
}
```

At the bottom, there is a 'Find:' input field, a search icon, and a message 'No matches'. To the right, it says 'Time: 0 ms | Body Length: 0 | Total Length: 0 bytes'.

*Fig. 23– Modified GUID request*

5. Analyzed server response.

The screenshot shows the 'Manual Request Editor' window. At the top, there are tabs for 'Request' and 'Response'. Below the tabs are buttons for 'Header: Text' (selected), 'Body: Text', and two icons. A 'Send' button is also present. The 'Response' tab is active, displaying the following JSON content:

```

HTTP/1.1 200 OK
Date: Fri, 04 Jul 2025 13:39:03 GMT
Cache-Control: no-store
Content-Type: application/json; charset=utf-8
content-length: 373

{
  "changes": {},
  "commits": [],
  "committedObjectsOmitted": false,
  "deletes": [],
  "extraGuids": [],
  "hasMoreItems": false,
  "newpersistable": [],
  "objects": [
    {
      "objectType": "TaskTracker.CommentHelper",
      "guid": "19421773393035674",
      "attributes": {
        "Content": {
          "value": null
        },
        "hash": "d960RtiGXC+uuFc6DF7DsTsxPrG565sYIGOx8XOLc68"
      },
      "partialObjects": [],
      "resets": {},
      "resultGuids": ["19421773393035674"]
    }
  ]
}

```

At the bottom of the window, there is a 'Find:' input field, a search icon, and the text 'No matches'. To the right, it shows 'Time: 28 ms | Body Length: 373 | Total Length: 521 bytes'.

*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).

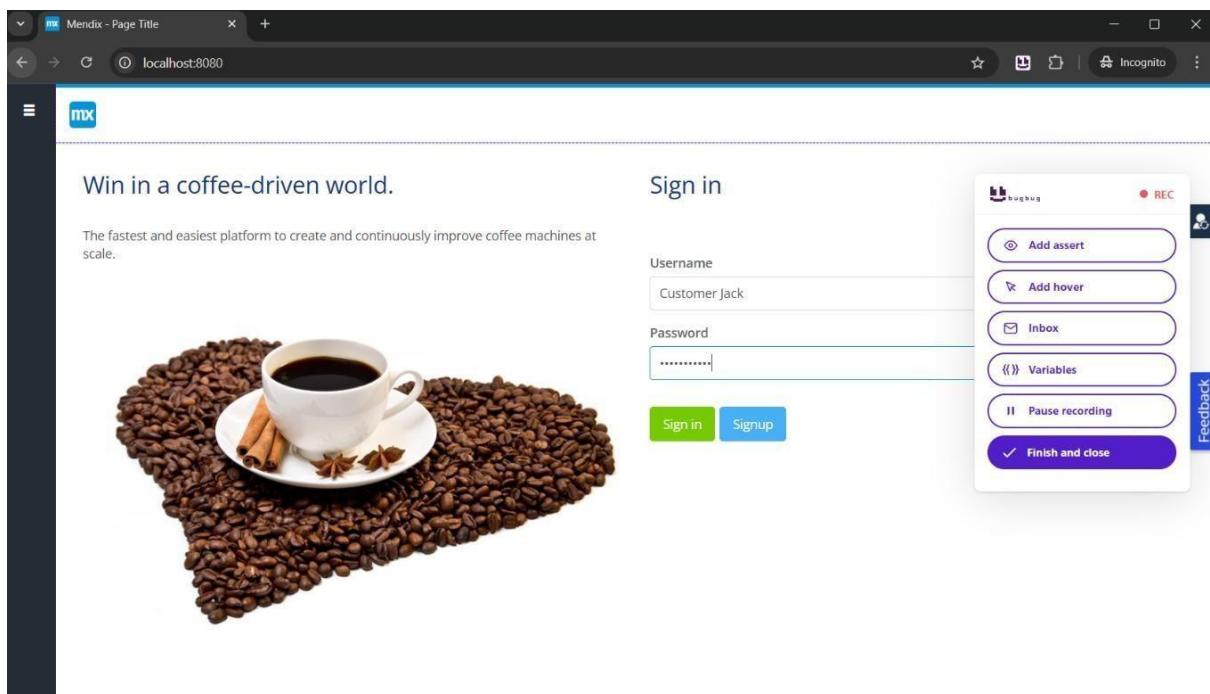
**Steps:**

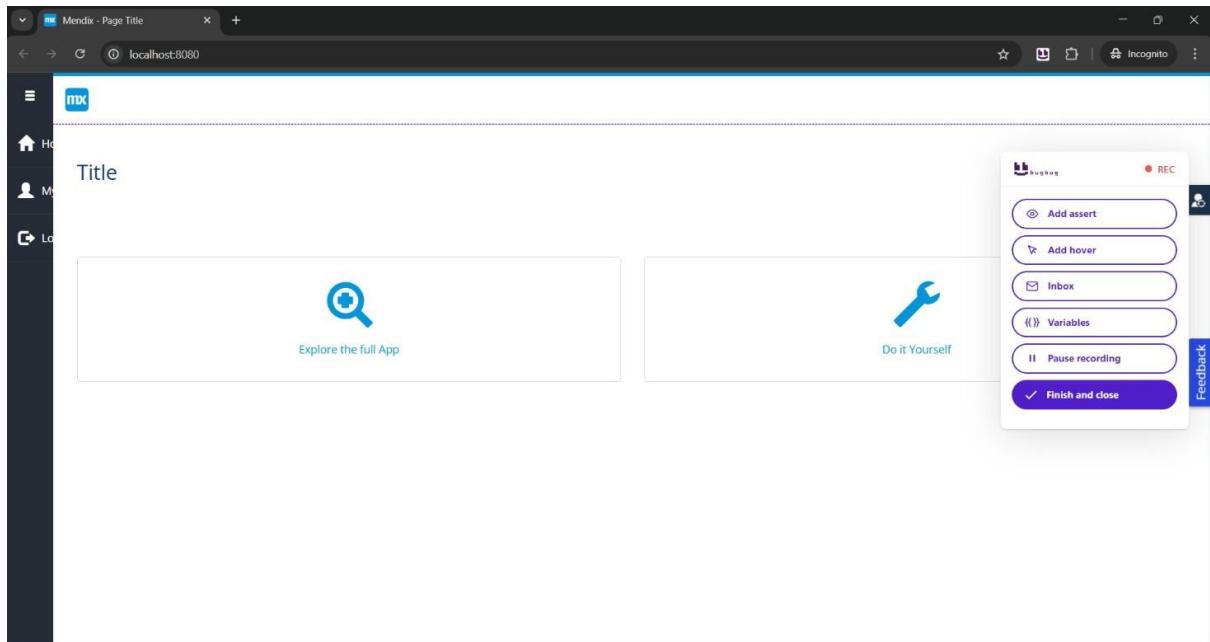
1. Navigate to login page.
2. Enter valid credentials for Customer (Jack).
3. Enter valid credentials for Engineer (Bill).
4. Click *Login*.
5. Verify dashboard loads.

**Results:**

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

**Figures:**

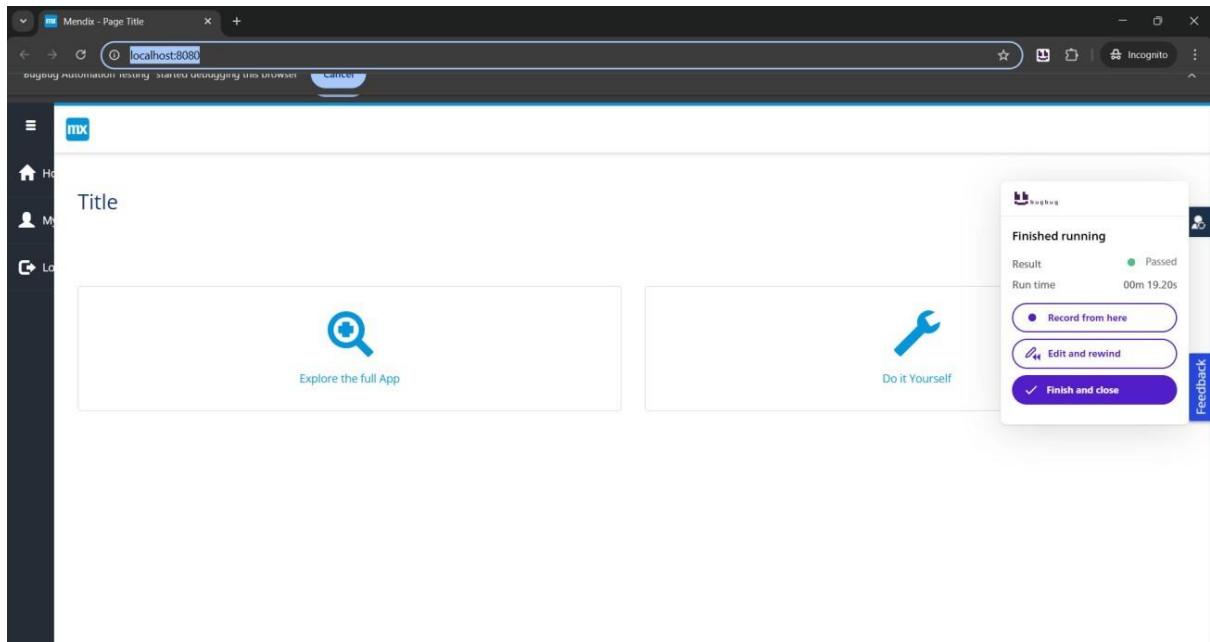




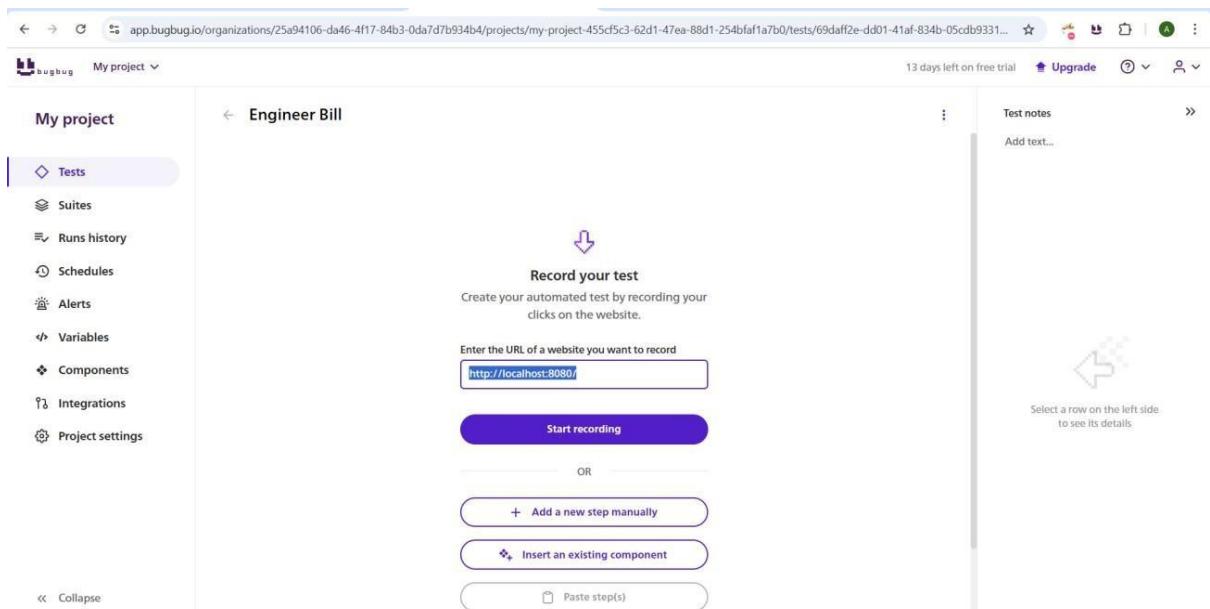
- Fig 1: Logged in as Customer Jack

A screenshot of the bugbug.io test dashboard. The left sidebar shows navigation options: 'My project' (selected), 'Tests' (highlighted in purple), 'Suites', 'Runs history', 'Schedules', 'Alerts', 'Variables', 'Components', 'Integrations', and 'Project settings'. The main panel displays a test titled 'Coffee Service UI Test' which was 'Just edited'. It contains 30 steps for a 'Login - Customer Jack' scenario. The steps include: 'Go to URL: http://localhost:8080/' (NEW), 'Click' (NEW), 'Type text: Customer Jack' (NEW), 'Click' (NEW), 'Type text: Protected content' (NEW), 'Click' (NEW), 'Click' (NEW), 'Clear input' (NEW), and 'Type text: Protected content' (NEW). There are also icons for 'Run' and 'Run in cloud'. A note on the right says 'Select a row on the left side to see its details'.

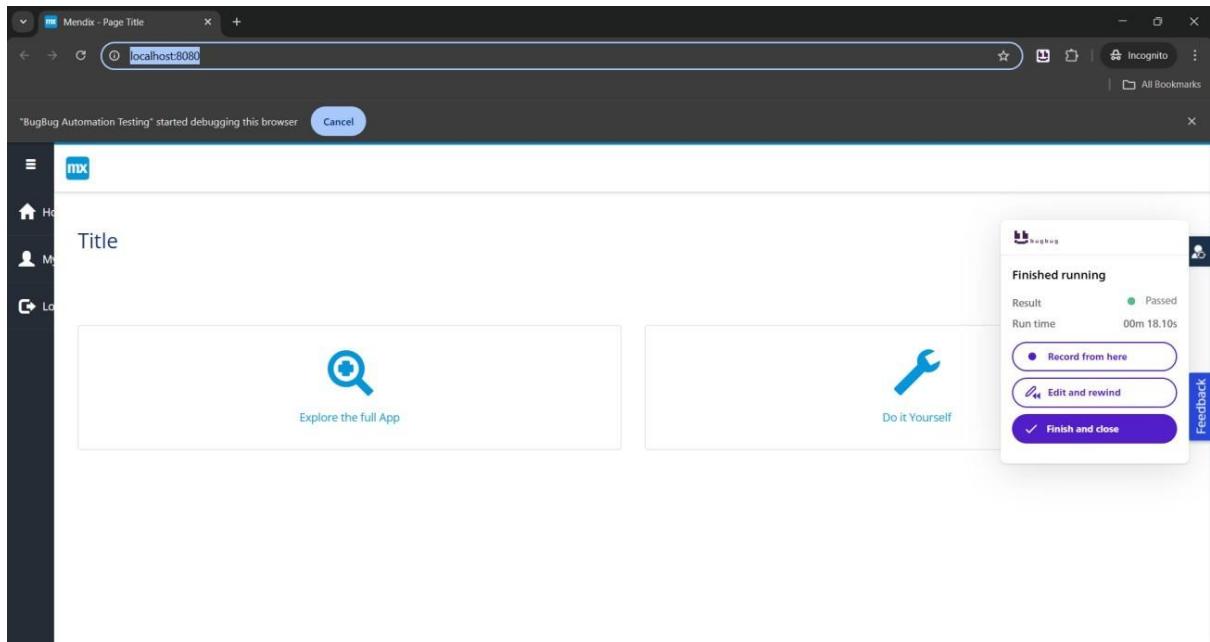
- Fig 2: Dashboard after successful login



- 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:**

The screenshot shows the bugbug.io web application interface. On the left, there's a sidebar with navigation links: Tests (selected), Suites, Runs history, Schedules, Alerts, Variables, Components, Integrations, and Project settings. The main area displays a test titled "Logout Test- Customer Roxanne". The test status is "Passed" with 24 steps. It includes a "Run" button and a "Run in cloud" button. The test details show a sequence of actions: Go to URL (http://localhost:8080/), Click, Type text (Customer Roxanne), Click, Type text (Protected content), Click, Assert element is visible, Assert element text is Logout, and Click. A note on the right says "Select a row on the left side to see its details".

- Fig 6: Logout test passed for Customer Roxanne

This screenshot shows the same bugbug.io interface as Fig 6, but for a different user, "Engineer Bill". The test title is "Logout Test- Engineer Bill". The test status is "Passed" with 26 steps. The steps listed are: Assert element is visible, Assert element text is Logout, Assert element is visible, and Click. Below the test results, there are three buttons: "Record from here", "New group", and "Paste step(s)". A note on the right says "Select a row on the left side to see its details".

- 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.

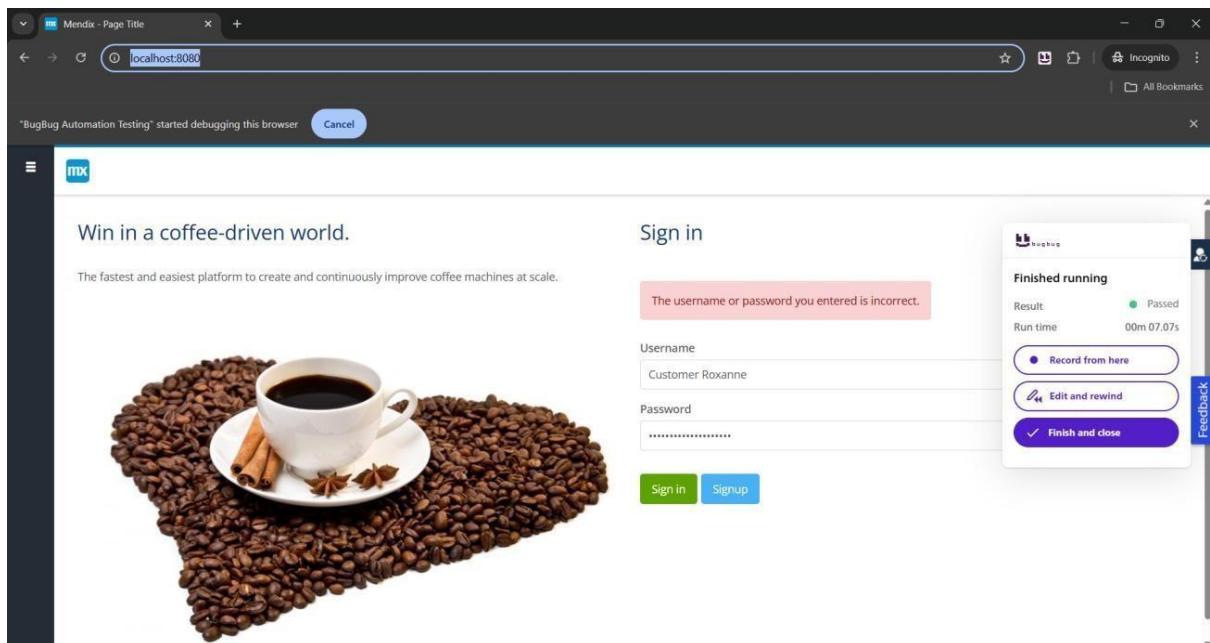
**Steps:**

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

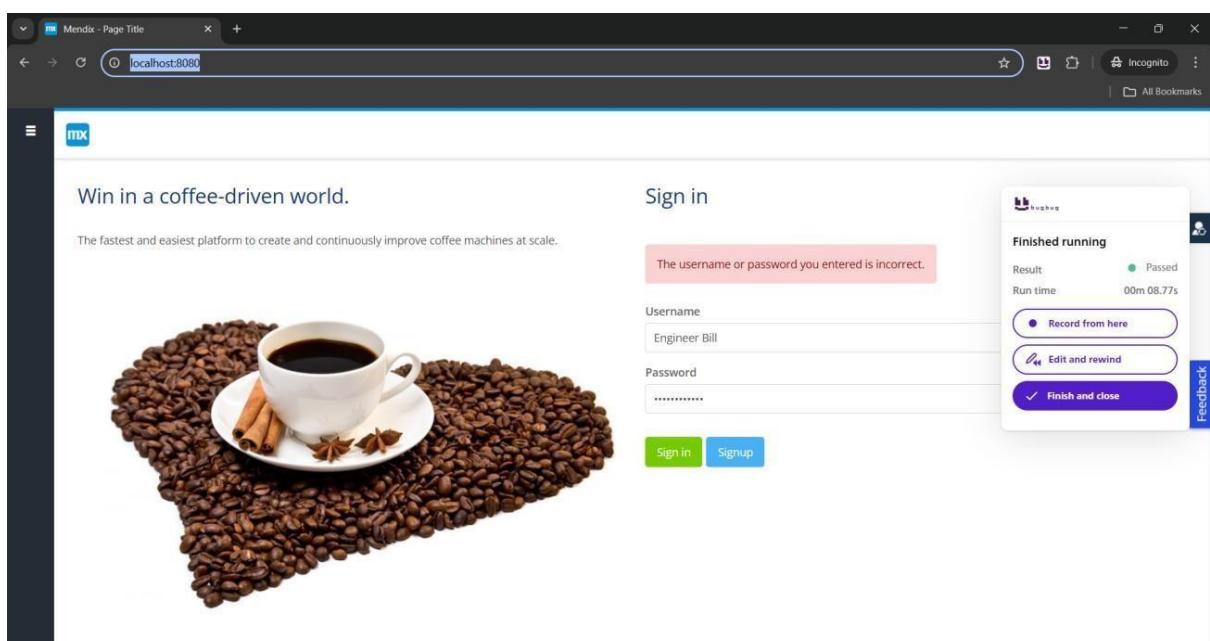
### Results:

- 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

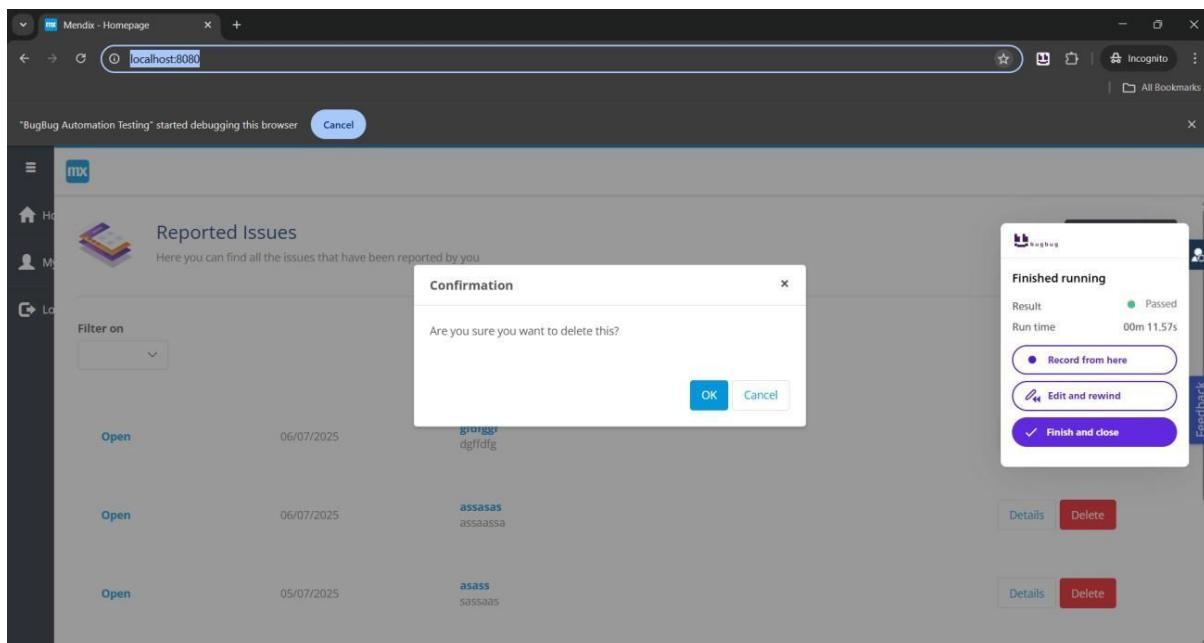
### A) Button Click Test

**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:**

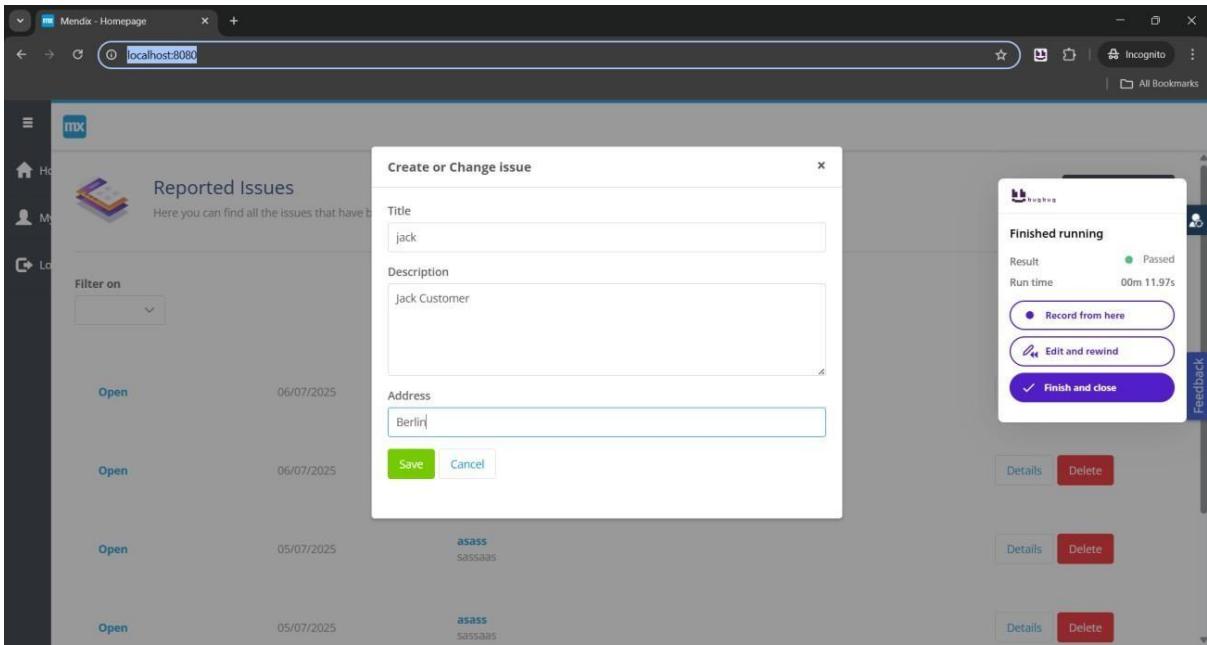
- Likelihood: Low
- Impact: Moderate

## 1.3 Form Submissions

### A) Valid Data

- Forms submitted successfully.

- Confirmation displayed.

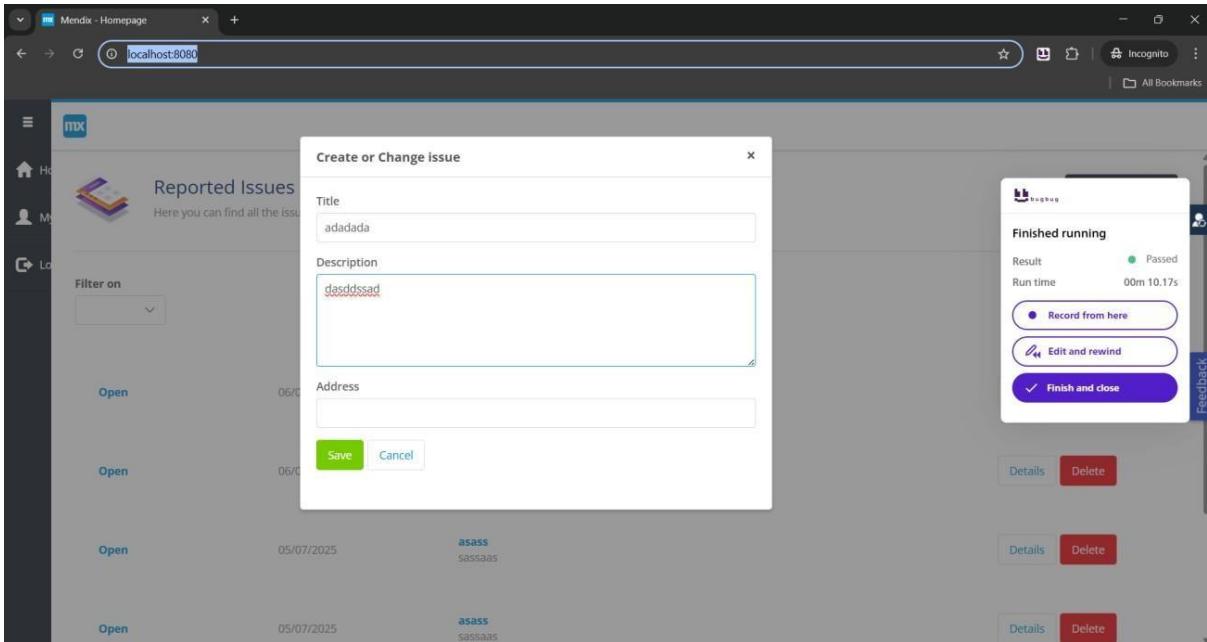


*Fig 10: Valid data test passed*

**Status:** PASSED

## B) Invalid/Missing Data

- Validation errors displayed correctly.



*Fig 11: Invalid data test passed*

**Status:** PASSED

**Risk:**

- Likelihood: Low
- Impact: Moderate

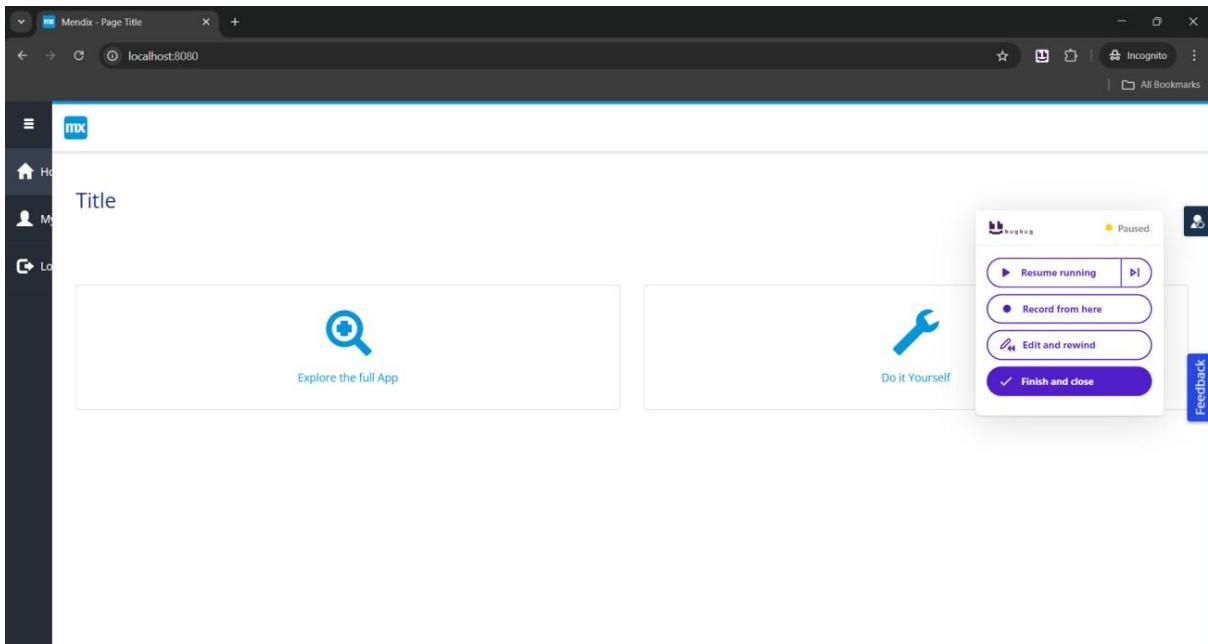
#### 1.4 Role-Based Access Control (RBAC)

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

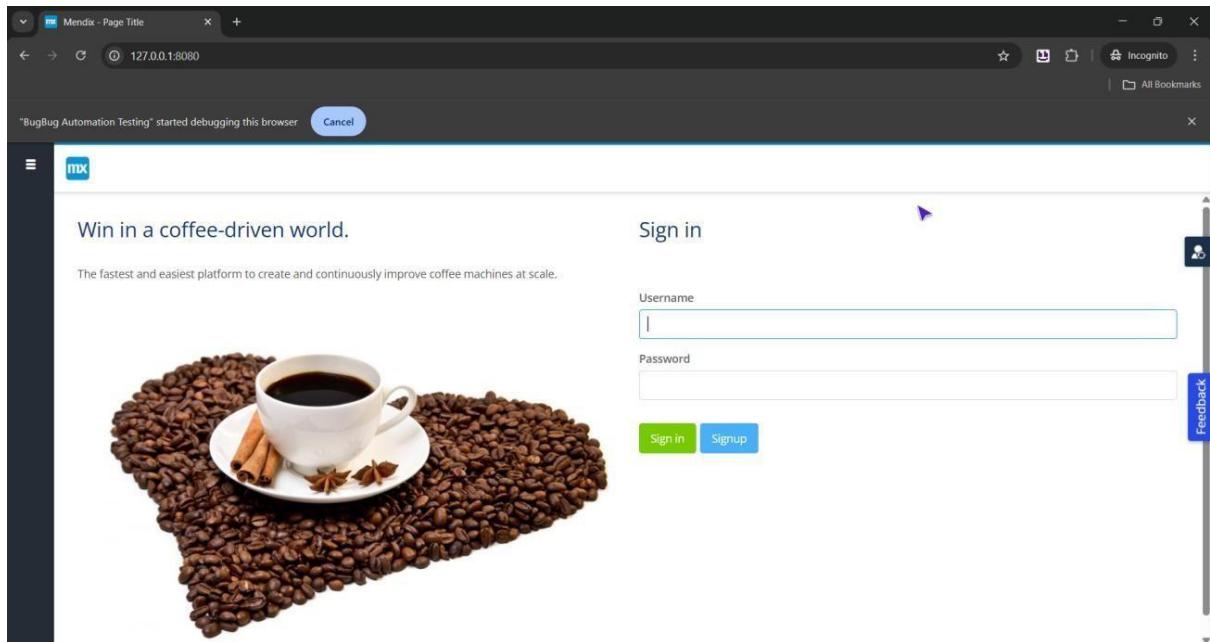
**Results:**

- 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.

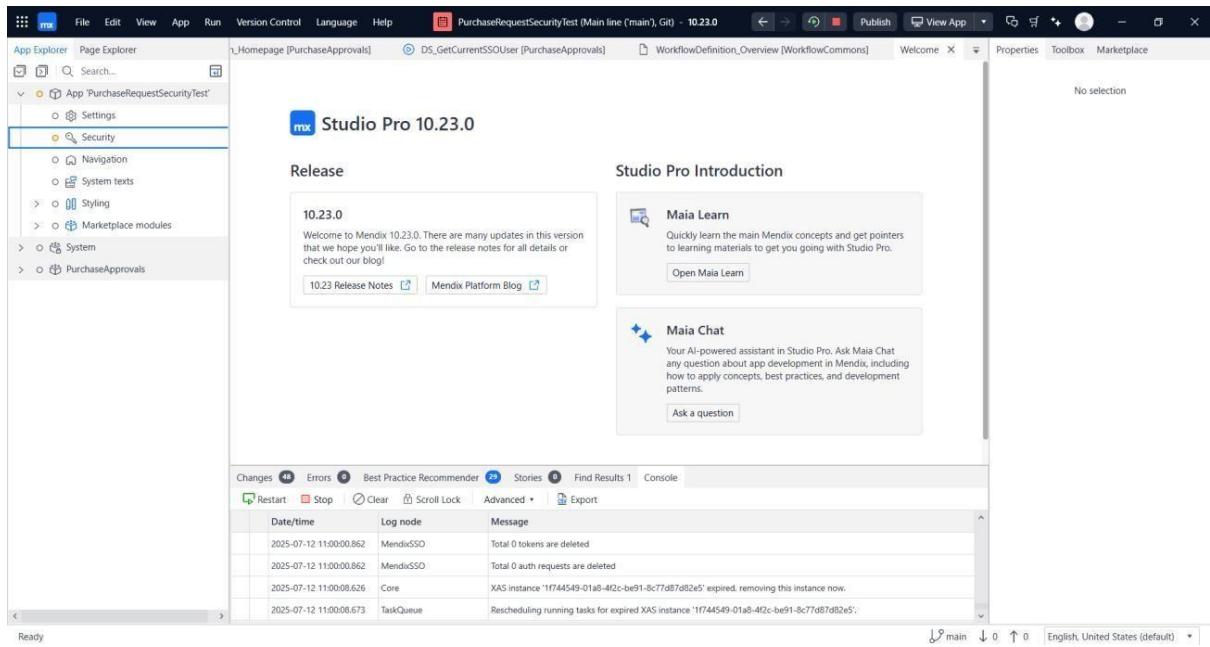
## 2. Purchase Request App

### 2.1 Authentication Tests

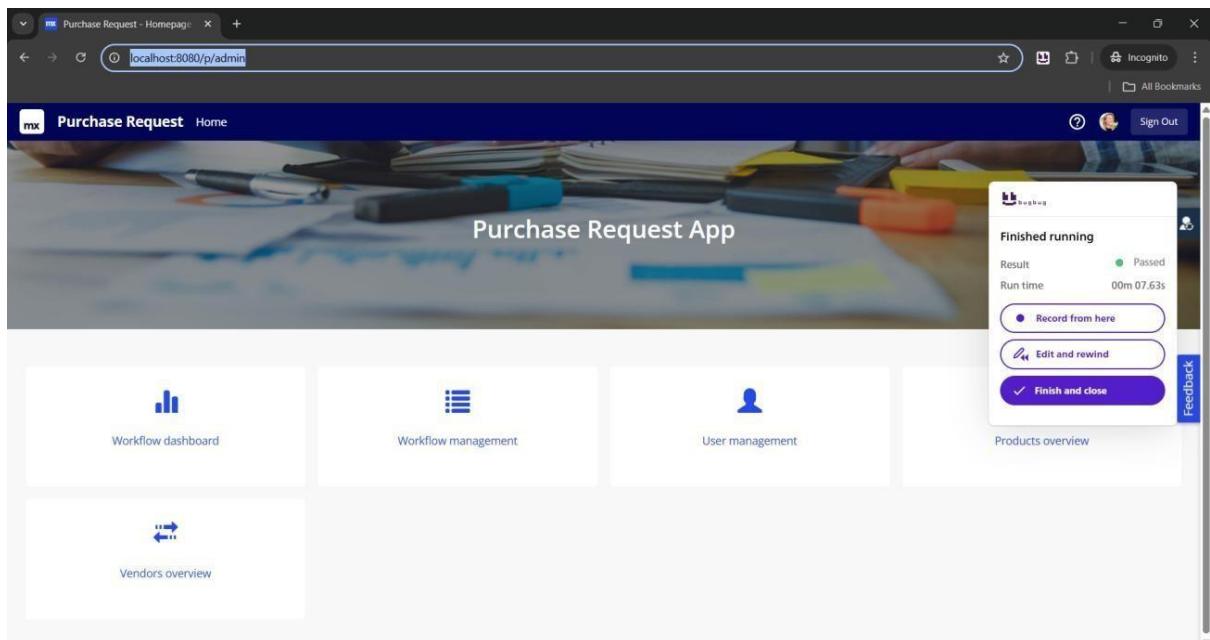
#### A) Valid Login Test

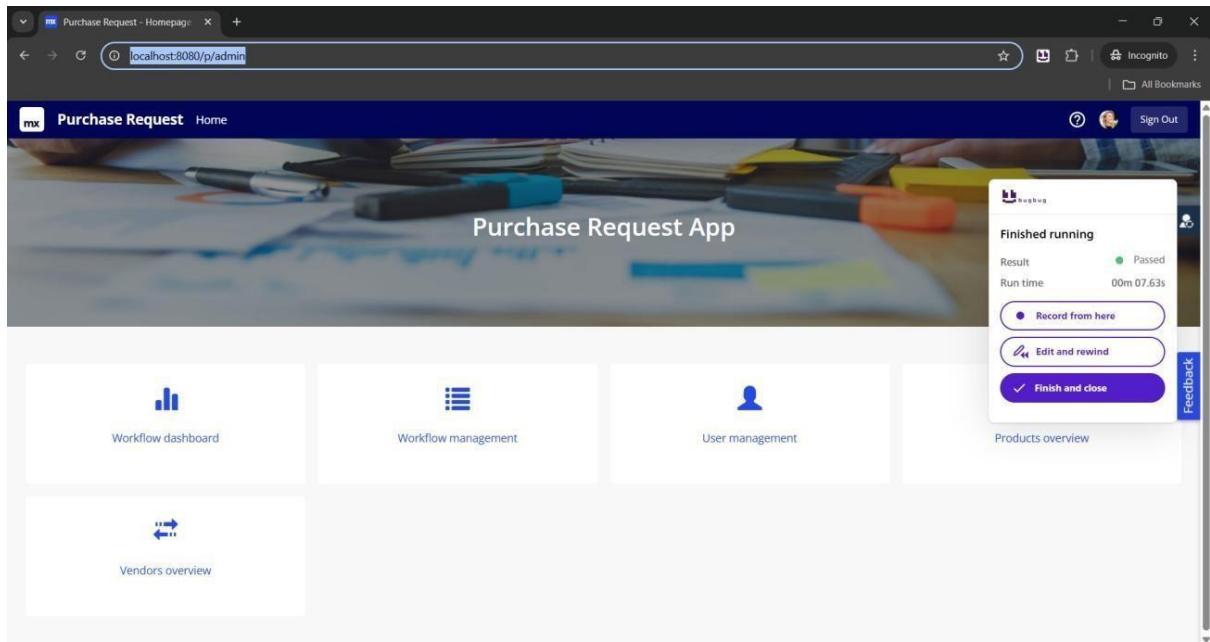
**Objective:** Verify successful login for Demo Administrator and Demo Approver.

**Figures:**



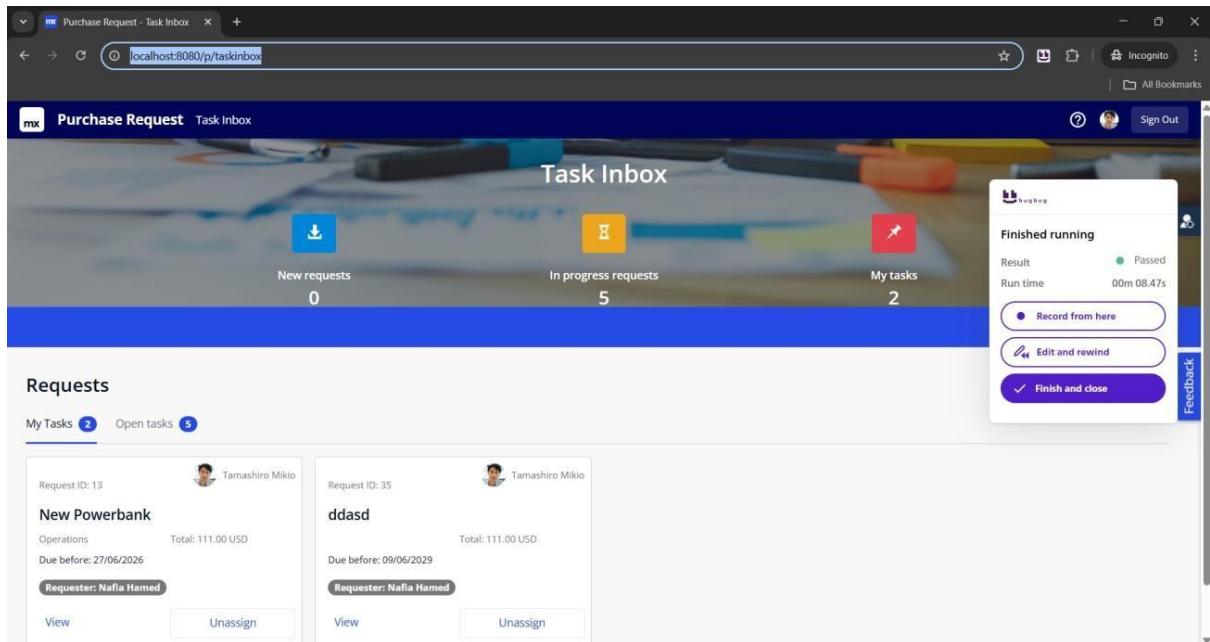
- Fig 1: App started





- Fig 2: Signed in as Demo Administrator

- Fig 3: Assertions – Administrator



- Fig 4: Login successful – Demo Approver

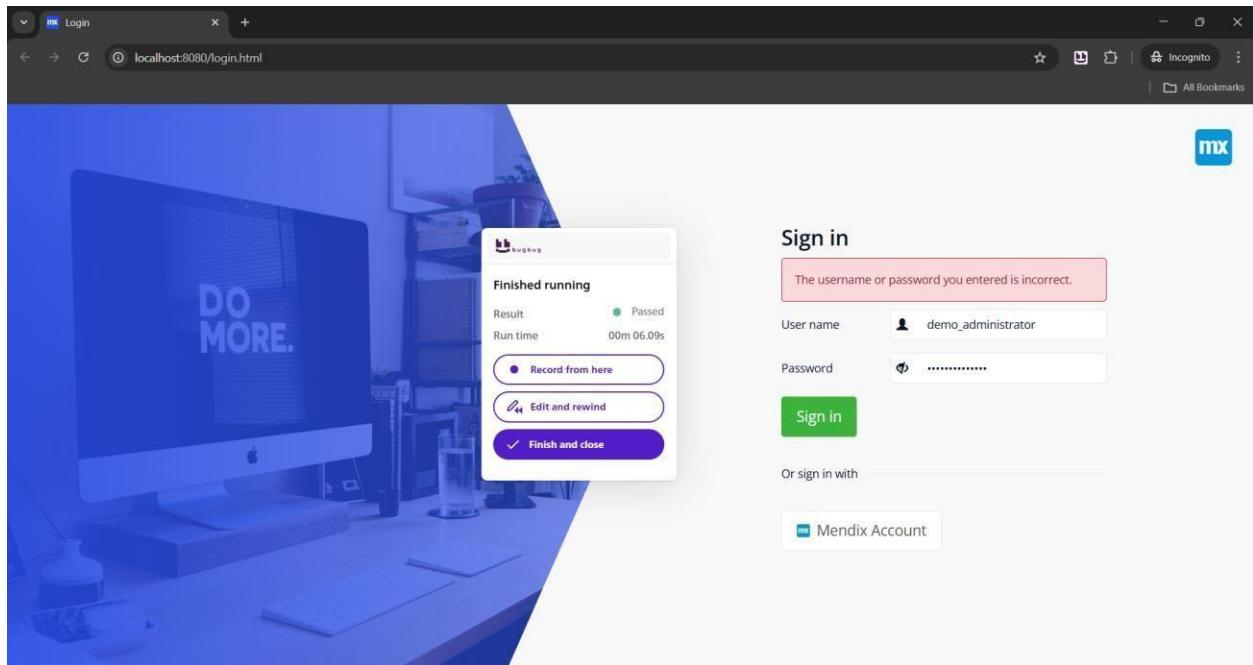
- Fig 5: Assertions – Approver

**Status:** PASSED

## B) Invalid Login Test

**Objective:** Verify error messages for invalid credentials.

**Figures:**



Invalid login tried with incorrect password on demo administrator

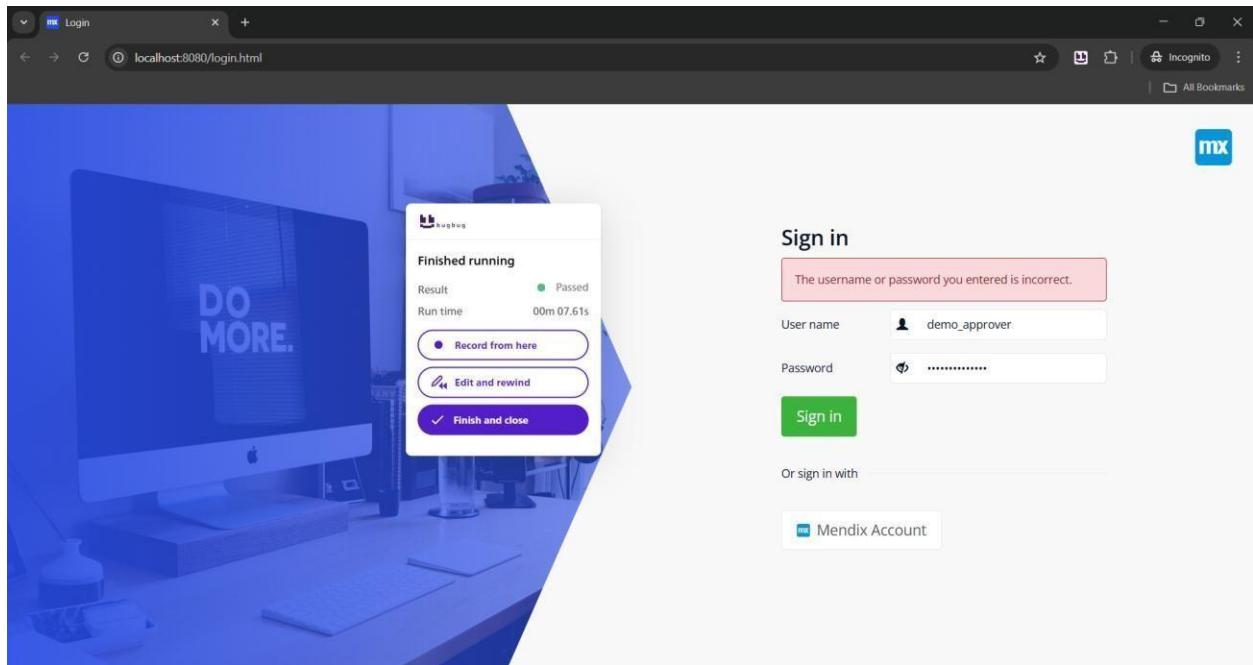
Your trial expires in 6 days. To continue using PRO features, please add a payment method.

6 days left on free trial [Upgrade](#)

Test notes  
Add text...

Select a row on the left side to see its details

Assertions Used for Invalid Tests on Demo Administrator



**Invalid login attempted on Demo Approver, Sign in Failed**

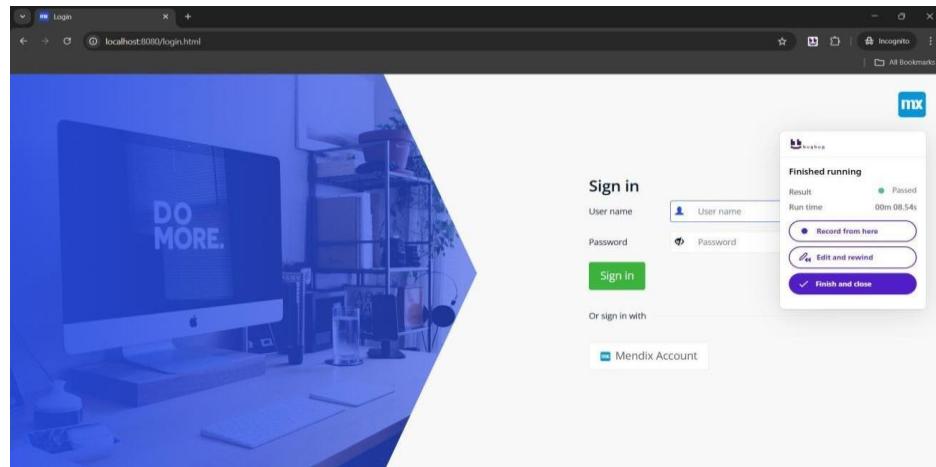
- Invalid login attempts

**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.

The screenshot shows the bugbug.io web interface. On the left, a sidebar menu includes 'My project', 'Tests' (selected), 'Suites', 'Runs history', 'Schedules', 'Alerts', 'Variables', 'Components', 'Integrations', and 'Project settings'. The main area displays a test run titled 'Logout Test Demo Administrator'. The test has passed with 117 steps, 6 steps, and a duration of 00m 07.33s. The steps include navigating to 'localhost:8080', entering 'demo\_administrator' for User name and Password, clicking 'Sign In', and asserting the element text is 'Sign Out'. A note at the top says 'Your trial expires in 6 days. To continue using PRO features, please add a payment method.' and a message '6 days left on free trial'.

### Assertions used for Demo Administrator

The screenshot shows a browser window with the URL 'localhost:8080/login.html'. The page displays a 'Sign in' form with fields for 'User name' and 'Password', and a 'Sign in' button. To the right, a 'Finished running' summary is shown: Result Passed, Run time 00m 08.74s. Buttons for 'Record from here', 'Edit and rewind', and 'Finish and close' are available. Below the form, there's a link for 'Mendix Account'.

Sign out Successfully for Demo Approver

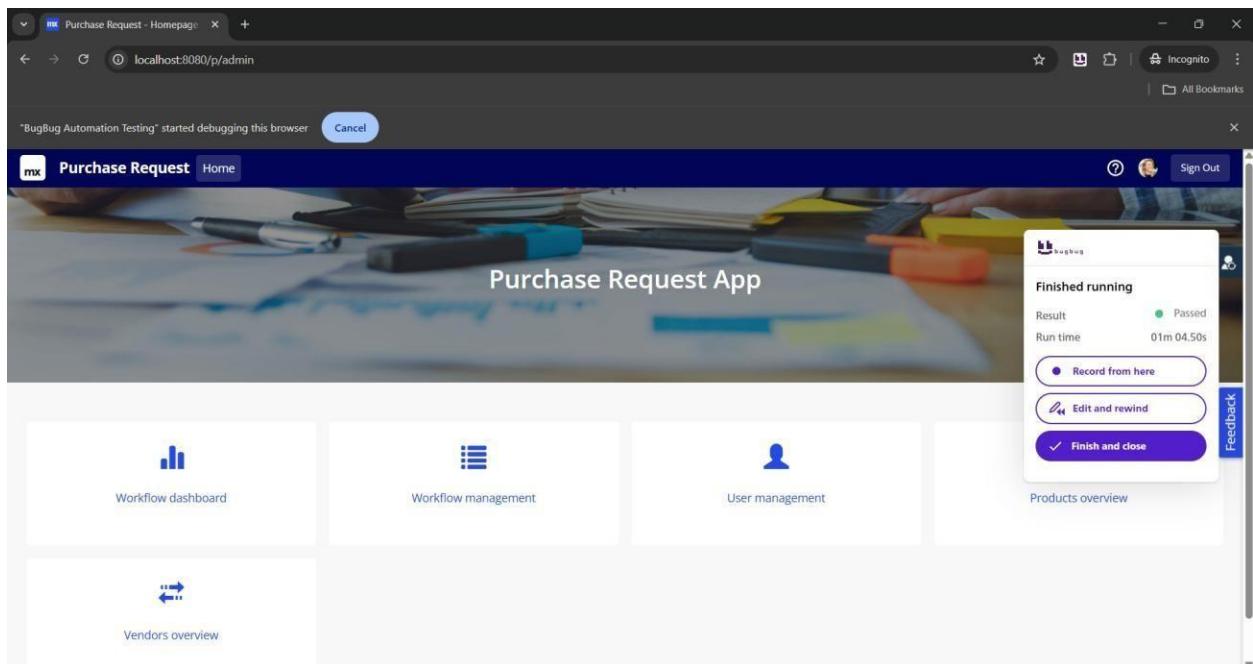
## Assertions Performed on Sign out test Status: PASSED

**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 Button Click Tests and Checking cancel and Save Button Functionality**



## Button Tests Successfully Passed for Administrator

The screenshot shows the bugbug.io web application interface. On the left, a sidebar menu includes 'My project' (selected), 'Tests' (highlighted in blue), 'Suites', 'Runs history', 'Schedules', 'Alerts', 'Variables', 'Components', 'Integrations', and 'Project settings'. Below this is a 'Collapse' button. The main area displays a test run titled 'Button Clicks Test Administrator'. The run status is 'Passed' (# 125, just now). It contains 60 steps, took 00m 53.80s, and has a 'Details' link. The steps list includes actions like 'Go to URL http://localhost:8080', 'Type text demo\_administrator', 'Click', 'Type text Protected content', 'Click', 'Click', 'Click', and 'Click'. To the right, there are sections for 'Step...', 'Cancel', 'Save', 'Screenshots' (showing a screenshot of the browser with the demo administrator page), and 'Interaction' (with an 'Action' dropdown set to 'Click', 'Element selector' dropdown, and 'Suggested' radio button selected). A banner at the top states 'Your trial expires in 6 days. To continue using PRO features, please add a payment method.' with a 'Upgrade' button.

## Assertion used for Button Clicks on Administrator account

The screenshot shows a web-based application interface for a 'Purchase Request' task. At the top, there's a header bar with the title 'Purchase Request' and a 'Task Inbox'. Below the header, a modal window is open for a task titled 'ddasd'. The modal contains fields for 'Requester' (Nafia Hamed), 'Department', and 'Shipping address'. There are 'Reject' and 'Approve' buttons at the top right of the modal. To the right of the modal, a sidebar displays 'Current Assignment' for user 'Tamashiro' (Passed, 00m 15.63s). It also includes a 'Comments' section with options like 'Record from here', 'Edit and rewind', and 'Finish and close'.

## Button Clicks Test Successful on Approver

The screenshot shows the 'bugbug' test runner interface. On the left, a sidebar lists project settings like 'Tests', 'Suites', 'Runs history', etc. The main area shows a test named 'Button Clicks Test Approver' with a status of 'Passed'. The test details show it has 15 steps and took 00m 13.16s. The test steps are listed as follows:

- Group number 76
- Go to URL <http://localhost:8080/>
- Type text demo\_approver
- Click
- Click
- Type text Protected content
- Click

On the right, there are sections for 'Test notes' and a note: 'Select a row on the left side to see its details'.

## Assertions used for Approver

The screenshot shows a web application for managing purchase requests. At the top, there are four status indicators: 'Pending approval' (7), 'In progress' (2), 'Finished' (0), and 'Rejected' (2). Below this, a section titled 'My Requests' lists three items, all marked as 'Pending Approval'. To the right, a detailed view of a request for a 'Mendix Tshirt' is shown, including creation date (12 Jul 2025), required delivery date (06 Jun 2026), and shipping address (Berlin). A bugbug test overlay is visible on the right, showing a 'Passed' result with a run time of 00m 32.89s.

## Button Clicks Test passed for requester user

The screenshot shows the bugbug test runner interface. On the left, a sidebar lists project settings like 'Tests', 'Suites', and 'Runs history'. The main area displays a test named 'Button Clicks Test Requester' which has passed. The test details show it consists of 49 steps, took 00m 22.60s, and was run on a desktop. The test steps include navigating to a URL, entering text into input fields, clicking buttons, and asserting element text. A note at the bottom says 'Select a row on the left side to see its details'.

## 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.

The screenshot shows a Mendix application interface titled "Purchase Request". In the center, a modal window titled "New product" is open. The "Name" field contains the value "Mendix Pizza Tshirt". Below it, a "Description" field has a dropdown menu open, listing several options: "addadad", "asdds", "AU1", "CN1", "DE1", "NL1", and "NL1". The "NL1" option is currently selected. At the bottom of the modal, there are "Save" and "Cancel" buttons.

## Form Submission Valid Data Administrator Passed

The screenshot shows the Bugbug.io test runner interface. On the left, a sidebar lists project components: Tests (selected), Suites, Runs history, Schedules, Alerts, Variables, Components, Integrations, and Project settings. The main area displays a test titled "Form Submission Test (Valid Data) Administrator". The test status is "Passed" (# 133, 00m 27.39s, 41 steps). The test steps are detailed as follows:

- Group number 80
- Go to URL <http://localhost:8080/>
- Type text demo\_administrator
- Click
- Click
- Type text Protected content
- Click
- Click
- Click
- Click

On the right, there is a note: "Select a row on the left side to see its details." with an arrow pointing to the left.

## Assertions Used

The screenshot shows a web browser window with two tabs. The active tab is titled "Purchase Request - New Request" and displays a summary of a purchase request. The summary includes:

- Step 1: Request information**: Title is "Mendix Tshirt", Department is "Finance". Status: Passed.
- Step 2: Add products**: Required delivery date is "09 Jan 2026". Status: Passed.
- Step 3: Summary**: Shipping address is "Berlin". Status: Pending.

Request reason is "Very Urgent". There are no attachments listed. The Product section is collapsed.

To the right of the browser, a sidebar from bugbug.io shows the test run results:

- Finished running**
- Result**: Passed
- Run time**: 00m 37.92s
- Actions**: Record from here, Edit and rewind, Finish and close.

## Form Submission Valid Data Requester Passed

The screenshot shows the bugbug.io test runner interface for a project named "My project".

**Test Details:**

- Status: Passed (# 141, just now, Desktop, Default, 00m 33.16s)
- Test notes: Add text...

**Test Steps:**

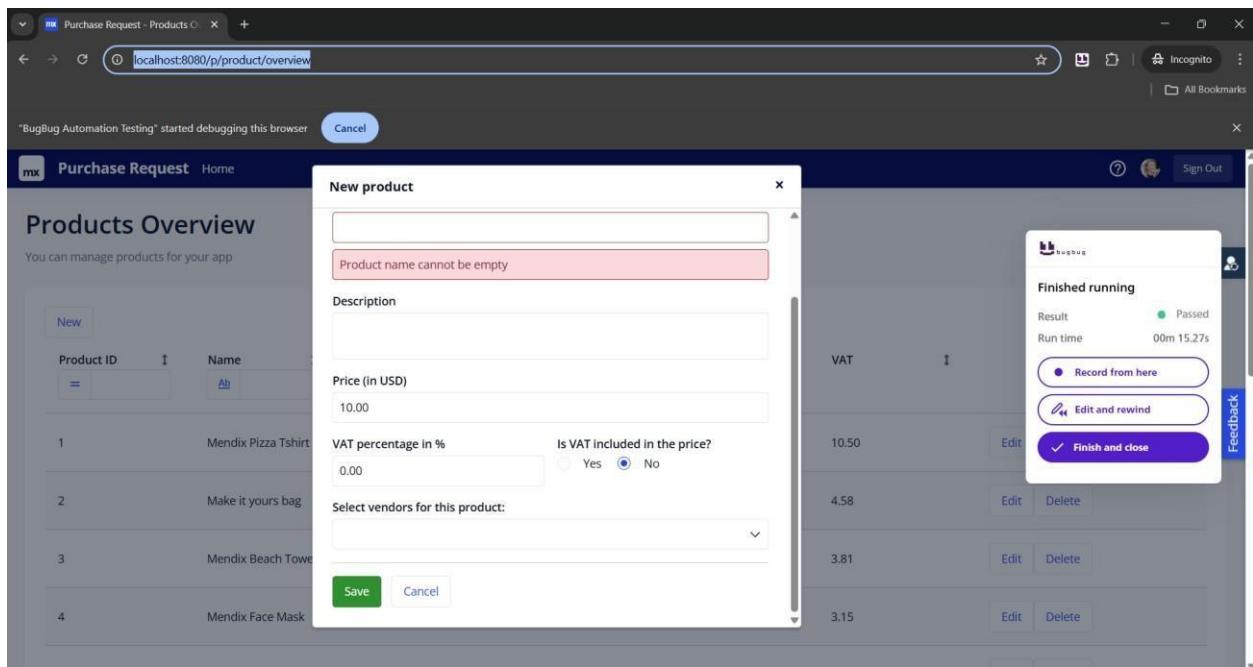
- Group number 82 (33 steps, 00m 33.16s)
  - Go to URL: http://localhost:8080/
  - Type text: demo\_requester (User name)
  - Click (Password)
  - Type text: Protected content (Password)
  - Click (Sign In)
  - Assert element text is: New Request (+ New Request)
  - Click (+ New Request)
  - Click

A sidebar on the left lists project settings like Tests, Suites, Runs history, Schedules, Alerts, Variables, Components, Integrations, and Project settings. A note says "Select a row on the left side to see its details".

*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.

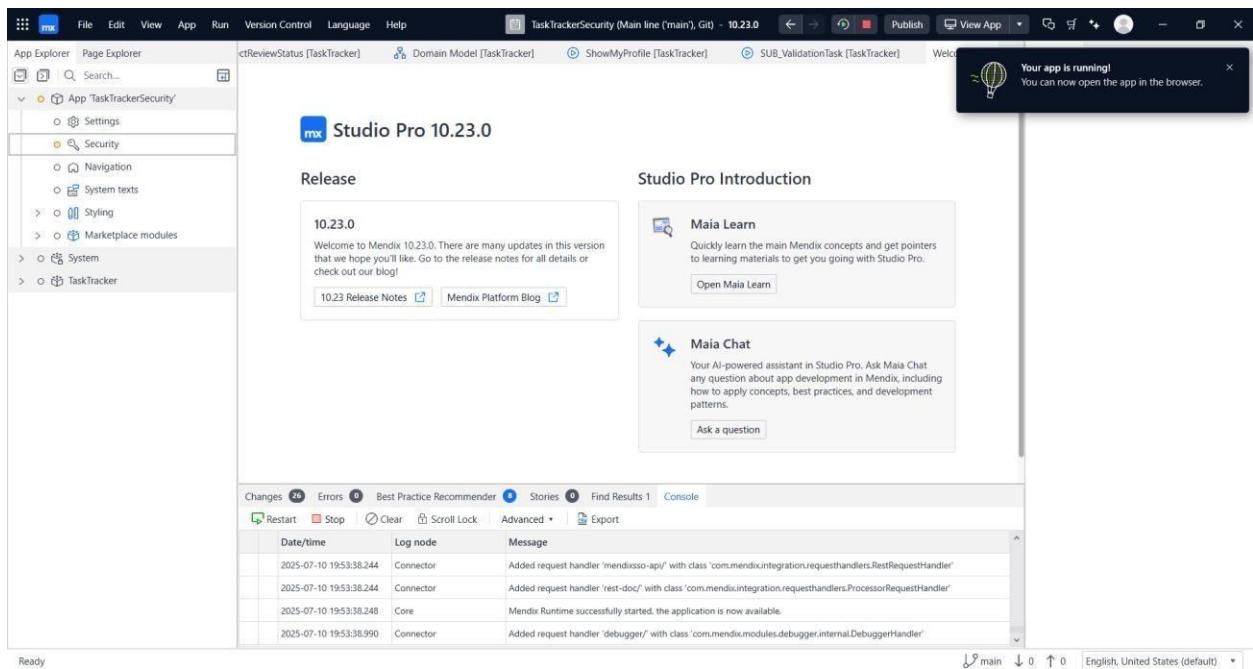
**Status:** PASSED

#### 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**

The screenshot shows the Bugbug.io web interface for a project titled "Valid Login Test Task Tracker App Manager". The left sidebar has a purple header "Tests" selected. The main area shows a "Passed" test run with 7 steps completed in 00m 07.29s. The steps include navigating to a URL, entering a user name, clicking, entering protected content, clicking again, asserting element visibility, and clicking once more. A tooltip on the right says "Select a row on the left side to see its details". The top right shows a 8 days left on free trial message and an "Upgrade" button.

## *Assertions Performed*

The screenshot shows a Mendix application titled "Task Tracker - Homepage" running on a browser. The main header includes the Mendix logo, the title "Task Tracker", and navigation links for "Home", "Team", and "My Profile". A large banner image of an aircraft engine is displayed, with the text "Mendix Aerospace" overlaid. Below the banner, a progress bar indicates "Team Progress" at 33%. On the right side, there's a modal window titled "Finished running" showing a green "Passed" status and a run time of "00m 08.75s". It has three buttons: "Record from here", "Edit and rewind", and "Finish and close". To the right of the modal, a sidebar titled "Select user" lists "Manager1" (Manager role) and "Member1" (Member role). At the bottom, a "Feedback" link is visible.

## *Tests Passed*

My project

Valid Login Test Task Tracker App Member

Passed # 72 just now Desktop Default 00m 07.06s Local Details

Test notes Add text...

Group number 40 7 steps 00m 05.43s Make component

- Type text: Protected content
- Click
- Assert element is visible
- Click

Select a row on the left side to see its details.

### Test Assertions for Member

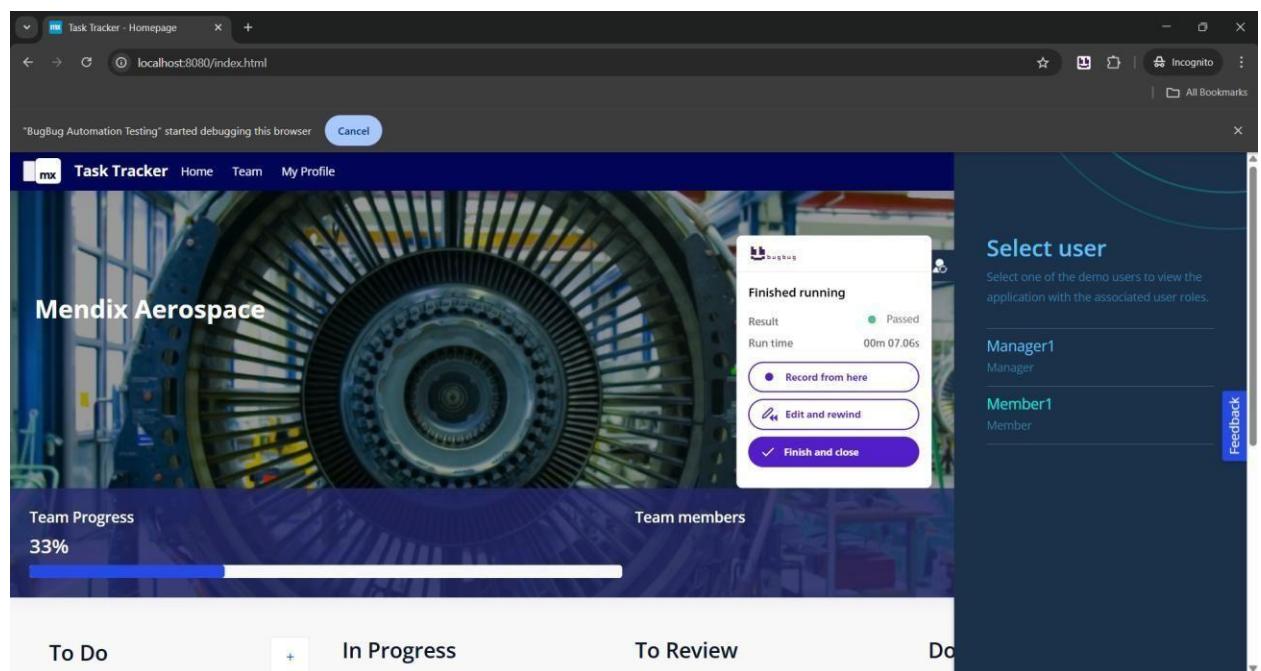


Fig 5: Test Passed for member login

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

## 3.2 UI Functional Tests

- Button clicks – PASSED.

- Form submissions:

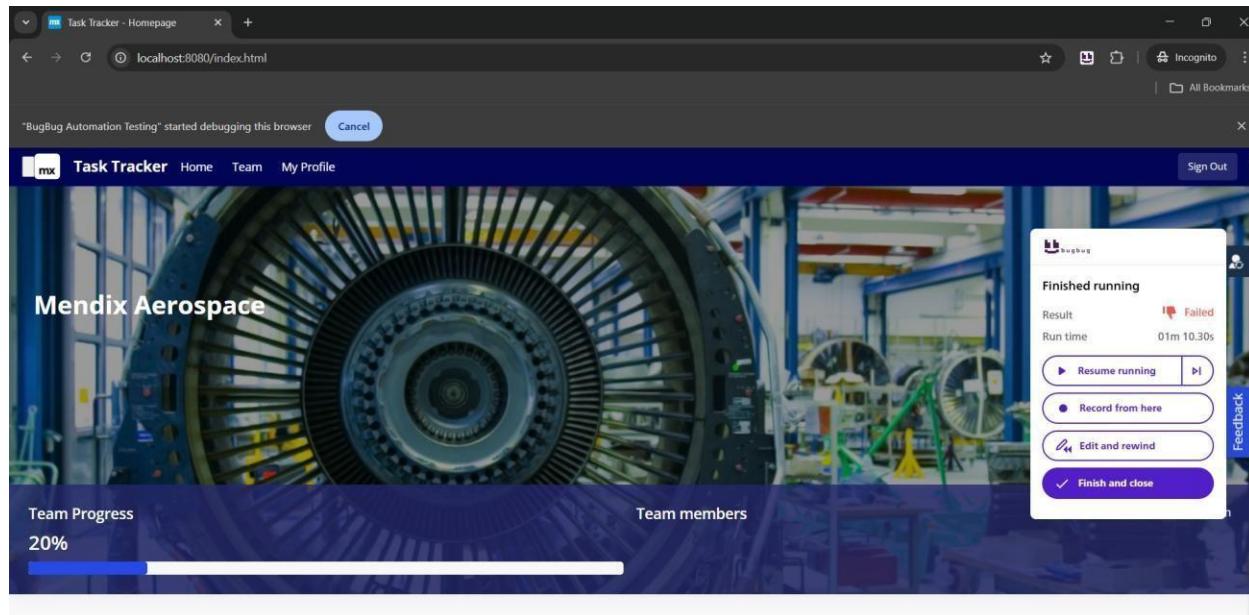
- Valid – PASSED

The screenshot shows the Bugbug.io test runner interface. On the left, there's a sidebar with options like Tests, Suites, Runs history, Schedules, Alerts, Variables, Components, Integrations, and Project settings. The main area is titled "Form Submission Valid Data" and shows a list of test steps under a "Passed" category. The steps include "Group number 55", "Assert element is visible", "Assert form value is 'good task'", "Assert element text is 'Save changes'", and "Scroll element to x: 0, y: 199". At the bottom of the test list, there are buttons for "Record from here", "New group", and "Paste step(s)". A green notification bar at the bottom says "Test has been updated successfully.".

The screenshot shows a Mendix Aerospace Task Tracker application. On the left, there's a navigation bar with "Task Tracker - Homepage" and a search bar with "localhost:8080/index.html". The main area displays a dashboard with "Mendix Aerospace" branding, "Team Progress 20%", and sections for "To Do", "In Progress", and "To Review". In the "In Progress" section, there's a task card for "EX-54 - Data can be exported to Excel" and another for "EX-57 Discounts can be set per category". To the right of the dashboard, a modal window from bugbug.io is open. It shows a summary: "Finished running", "Result: Passed", "Run time: 00m 21.72s", and a "Feedback" button. Below the summary, there are buttons for "Record from here", "Edit and rewind", and "Finish and close". The "Status" section shows icons for "To Do", "In Progress", "Review", and "Done". A "Comments" section contains the text "good task".

- Invalid – PASSED

### 3.3 RBAC Tests



To Do      +      In Progress      To Review      Done

### Role Based Access Control Failed- links can be bypassed

**Role Based Access Control-Direct URL Access/B**

Default Run Run in cloud

# 102 just now Desktop Default 01m 10.30s Local Details

Failed Go to failed step

Group number 60 at 13 of 30 01m 08.06s Make component

- Go to URL http://localhost:8080/ (Failed)
- Click (Passed)
- Type text Manager1 (Passed)
- Click (Passed)
- Type text Protected content (Passed)
- Click (Passed)
- Click (Passed)
- Click (Passed)
- View Team (Passed)

Select a row on the left side to see its details

**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.