PUCIT

Project 2 - (Redmine 13)

Submission to: **Dr. Mudassira Sheraz**

Submission by: **BSEF22M517** **Fiza Haider**

BSEF22M526 **Noor Fatima**

Submission date: **January 13, 2026**

Table of Contents

Introduction

Redmine is an open-source project management and issue tracking application written using the Ruby on Rails framework. This project involves deploying the Redmine application and performing various types of software testing, including functional UI testing, API testing, performance testing, security testing, and code quality analysis using industry-standard tools.

1. Deployment Environment

The Redmine application was deployed locally using Docker Desktop on a Windows 10 system. Docker was chosen to simplify the deployment process and avoid manual configuration of Ruby on Rails dependencies. The deployment uses the official Redmine Docker image with an SQLite database.

1.2. System Configuration

- **Operating System:** Windows 10
- **Deployment Tool:** Docker Desktop
- **Container Platform:** Docker with WSL2 backend
- **Application:** Redmine
- **Access Port:** 3000

1.3. Checking Prerequisites

1.3.1 Verifying Windows Subsystem for Linux (WSL2)

Before installing Docker Desktop, Windows Subsystem for Linux version 2 (WSL2) was verified to ensure compatibility. The following command was executed in the Command Prompt:

wsl --status

The output confirmed that the default WSL version was set to **Version 2**.

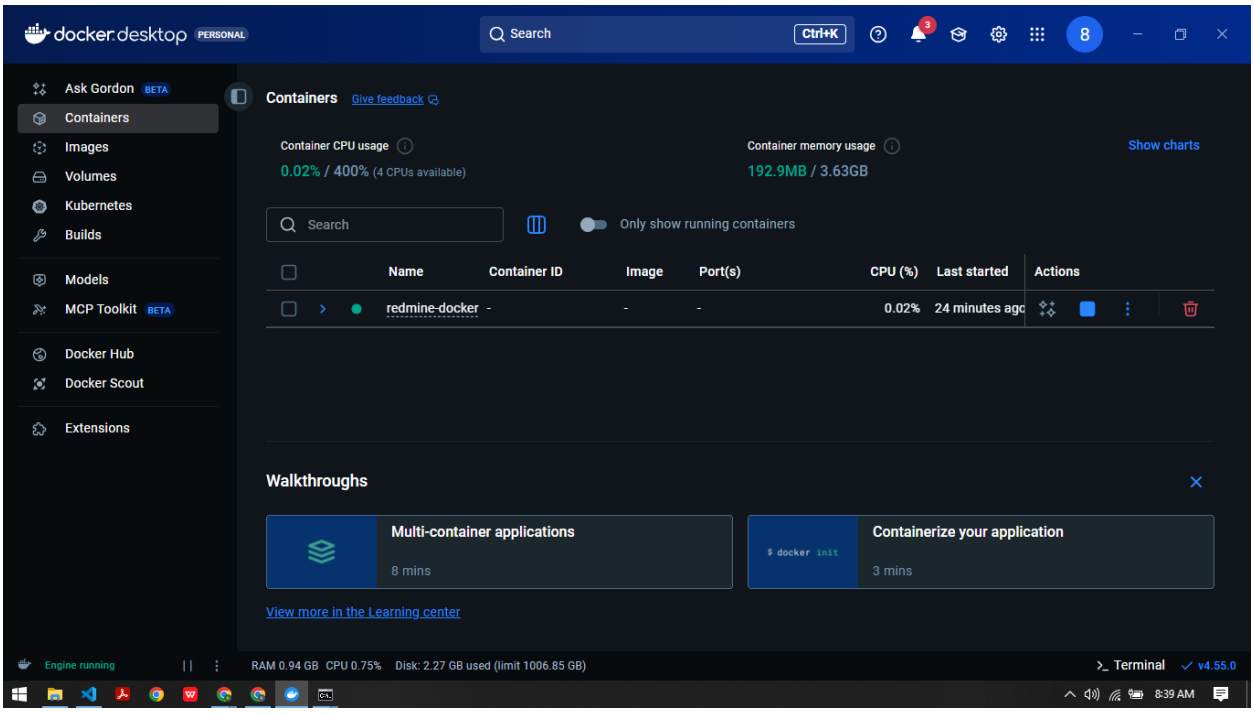
```
C:\Users\MM>wsl --status
Default Distribution: docker-desktop
Default Version: 2
```

1.4. Installing Docker Desktop

Docker Desktop was installed to enable containerized deployment of the Redmine application.

Steps Performed:

1. Docker Desktop for Windows was downloaded from the official Docker website.
2. The WSL2 backend option was enabled during installation.
3. The installation was completed, followed by a system restart.



1.5. Deploying Redmine Using Docker

Redmine was deployed using Docker Compose with the official Redmine Docker image.

1.5.1 Creating the Docker Compose File

A Docker Compose file was created at the following location:

E:\redmine-docker\docker-compose.yml

The following configuration was added in docker-compose.yml:

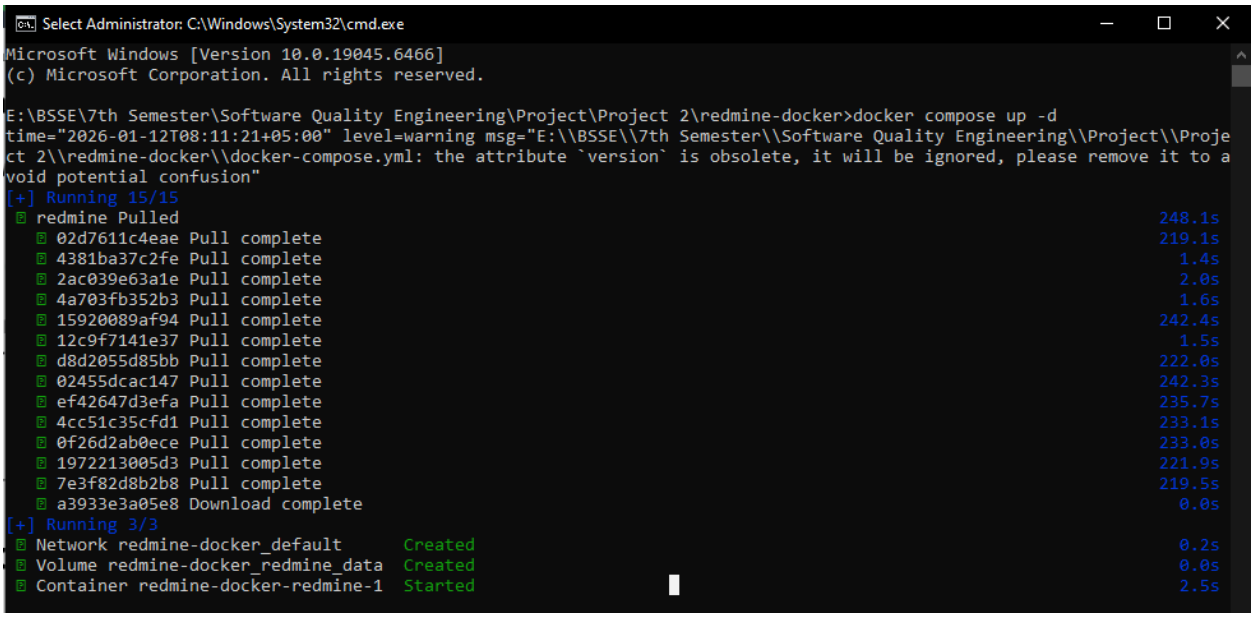
```
version: '3'
services:
  redmine:
    image: redmine:latest
    ports:
      - "3000:3000"
    restart: always
    volumes:
      - redmine_data:/usr/src/redmine/files
    environment:
      REDMINE_DB_SQLITE3: /usr/src/redmine/db/redmine.sqlite3

volumes:
  redmine_data:
```

1.5.2 Running the Application

The following command was executed from the E:\redmine-docker directory to start the Redmine application:

```
docker compose up -d
```

This command downloaded the required Redmine Docker image and started the application container in detached mode.



1.6. Accessing the Redmine Application

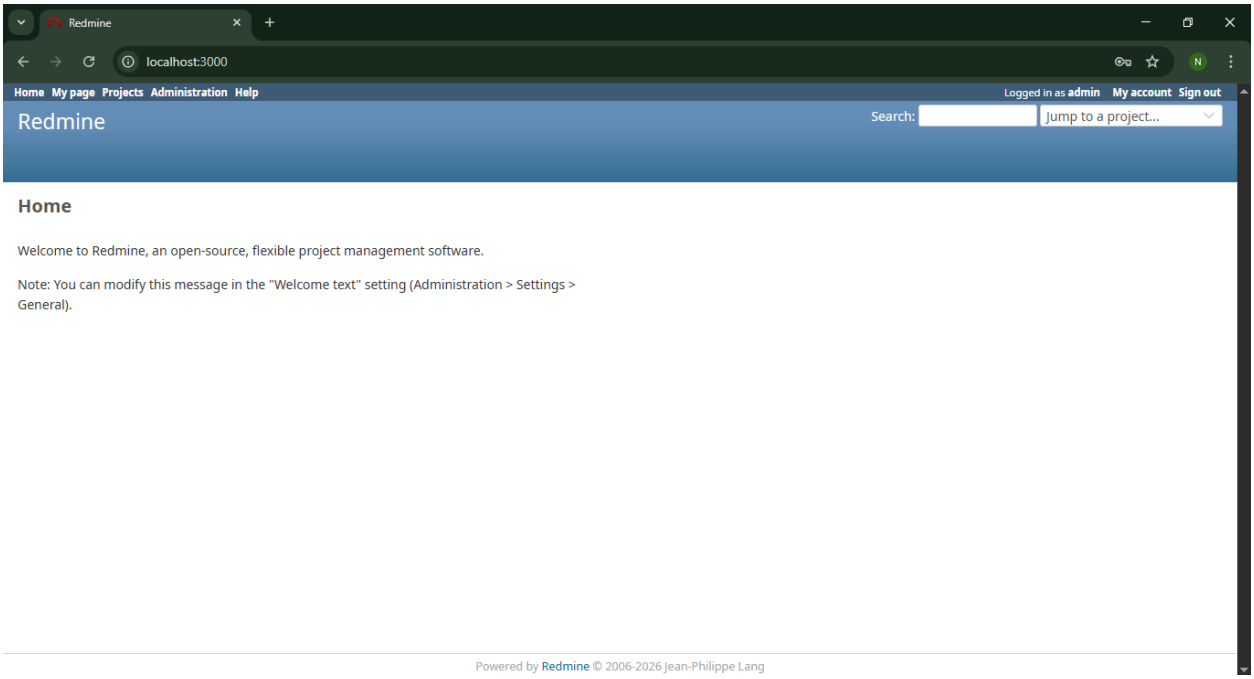
After successful deployment, the Redmine application was accessed through a web browser using the following URL:

http://localhost:3000

Default Login Credentials:
Username: admin
Password: admin

The Redmine application was successfully deployed locally using Docker Desktop. This deployment serves as the base environment for subsequent testing activities, including functional UI testing, API testing, performance testing, security testing, and code quality analysis.

The following page opened up, after putting in the credentials.



2. Performance / Load Testing

2.1 Tool Used

Apache JMeter was used to perform performance and load testing on the Redmine application. JMeter is an open-source load testing tool widely used to analyze and measure the performance of web applications under concurrent user load.

2.2 Test Objective

The objective of the performance test was to evaluate the response time, throughput, and stability of the Redmine application when accessed by multiple concurrent users. The test focused on simulating realistic user behavior by sending HTTP requests to commonly accessed application pages.

2.3 Test Environment

- **Application URL:** http://localhost:3000

- **Testing Tool:** Apache JMeter
- **Number of Users:** 50
- **Ramp-up Period:** 10 seconds
- **Loop Count:** 2
- **Test Type:** HTTP-based load testing

2.4 JMeter Test Plan Configuration

2.4.1 Thread Group Configuration

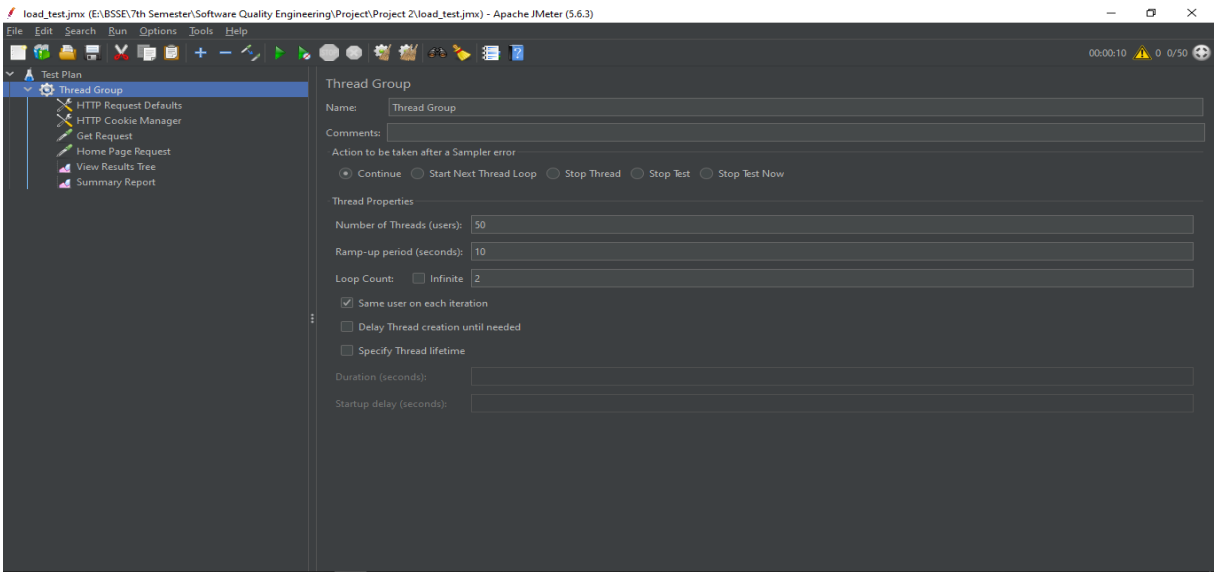
A Thread Group was added to the JMeter test plan to simulate concurrent users.

Configuration Details:
Run 1:

- Number of Threads (Users): 50
- Ramp-up Period: 10
- Loop Count: 2

Run 2:

- Number of Threads (Users): 1000
- Ramp-up Period: 10
- Loop Count: 4

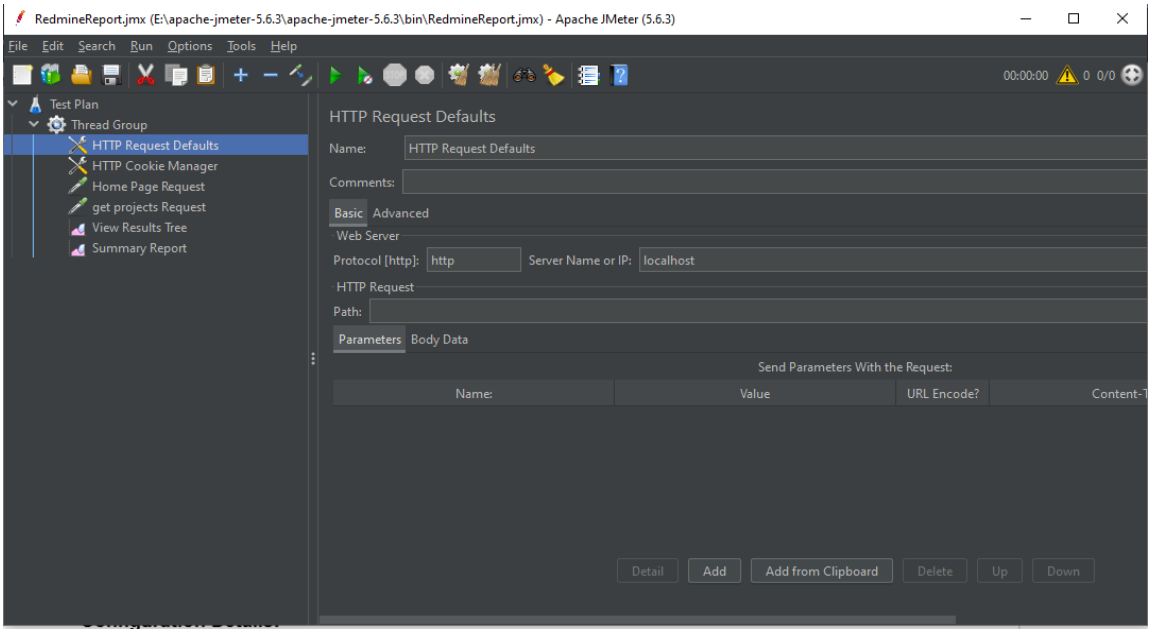


2.4.2 HTTP Request Defaults

An HTTP Request Defaults configuration element was added to avoid repeating server details for each request.

Configuration Details:

- Protocol: http
- Server Name or IP: localhost
- Port Number: 3000



2.4.3 HTTP Cookie Manager

An HTTP Cookie Manager was added to handle session management and cookies during the test execution. This can be found in the *Add Config* portion.

Configuration Details:

- Default settings were used

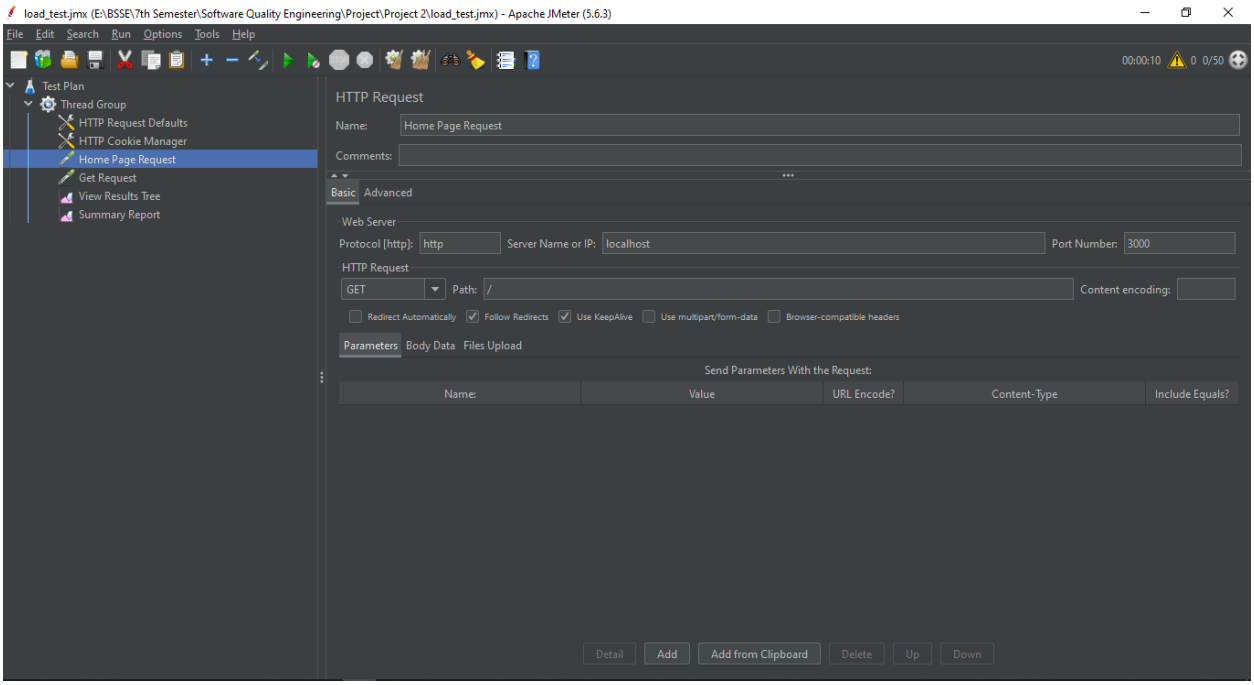
2.5 Sampler Configuration

2.5.1 Home Page Request

An HTTP Request sampler was added to simulate users accessing the Redmine home page.

Configuration Details:

- Method: GET
- Path: /

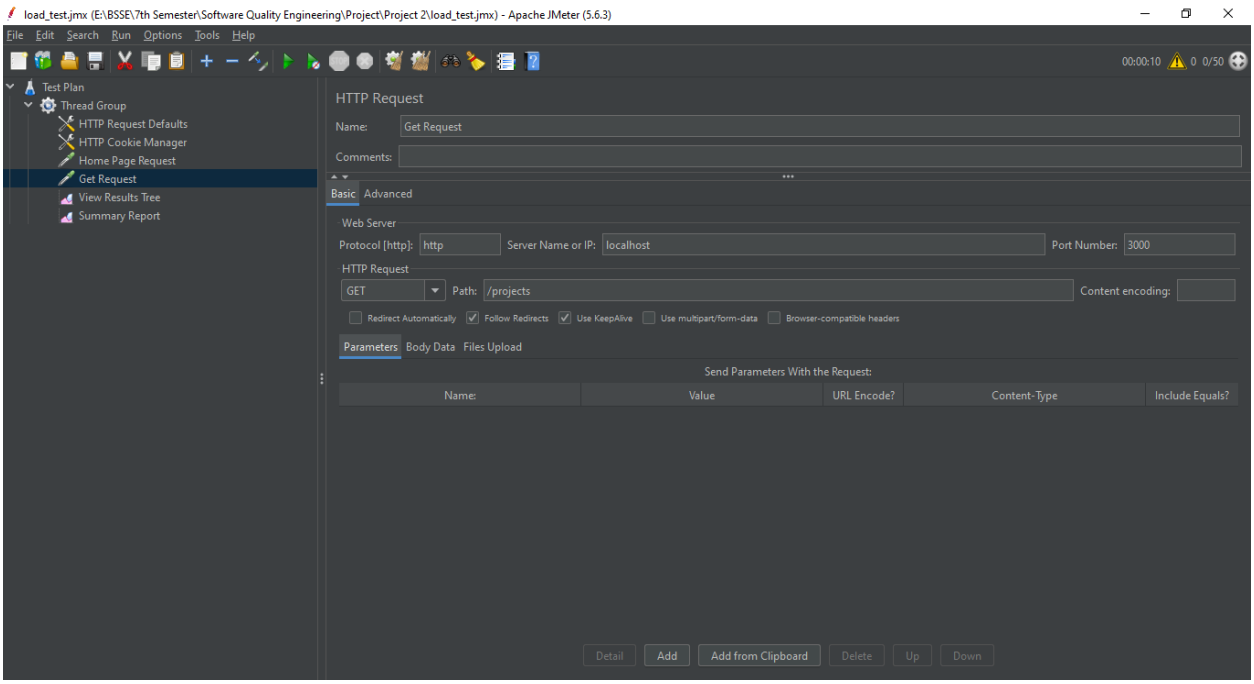


2.5.2 Projects Page Request

An HTTP Request sampler was added to simulate users accessing the projects page.

Configuration Details:

- Method: GET
- Path: /projects



2.6 Note on Login POST Request and CSRF Protection

During test design, a POST request to the /login endpoint was initially attempted to simulate user authentication. However, the request returned an HTTP **422 Unprocessable Content** response.

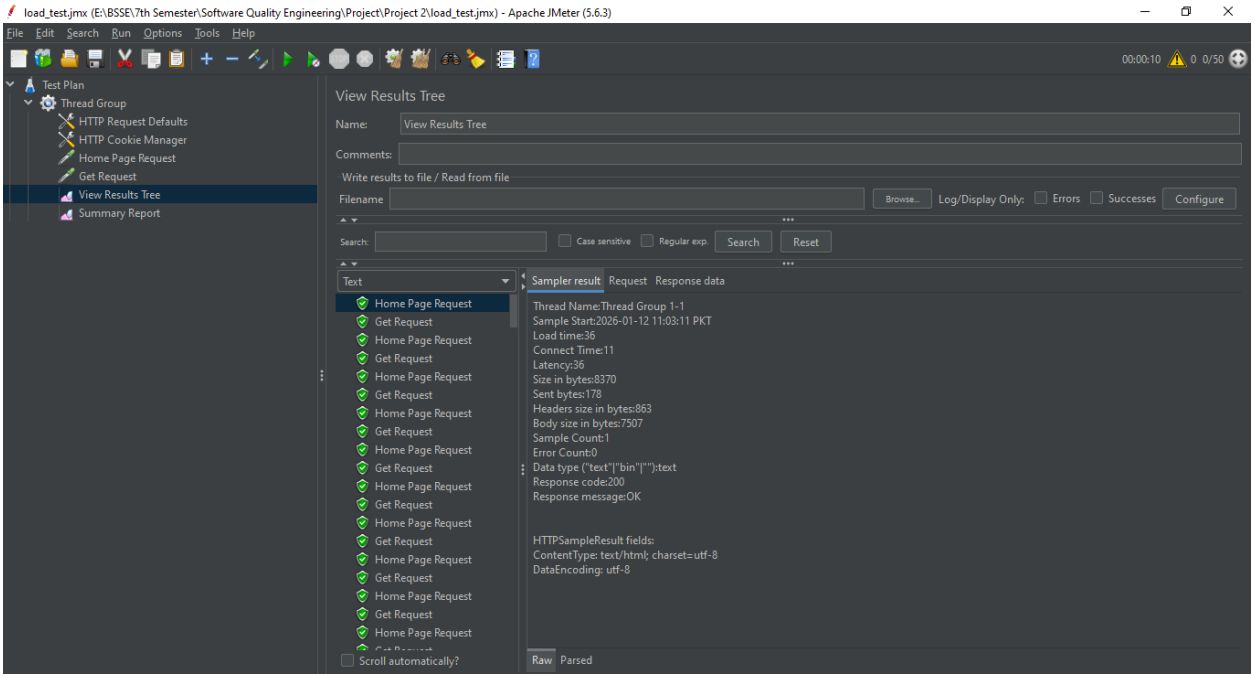
This behavior occurs because Redmine is built using the Ruby on Rails framework, which enforces **Cross-Site Request Forgery (CSRF) protection**. Rails requires an authenticity_token to be submitted with POST requests such as login. Since JMeter does not automatically extract and submit this token, the login POST request was rejected by the server.

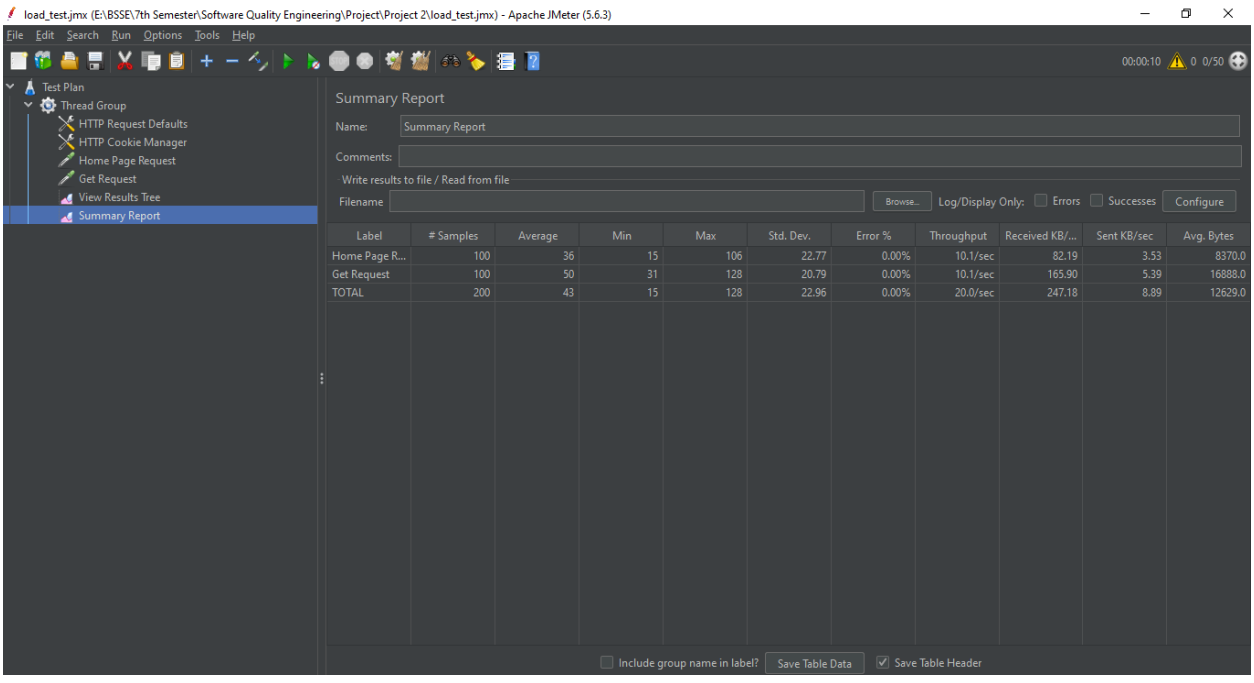
To avoid bypassing application security mechanisms and to maintain responsible testing practices, authentication endpoints were excluded from load testing. Instead, performance testing focused on page-level requests that represent typical user navigation behavior.

2.7 Listeners Used

The following listeners were added to collect and analyze test results:

- View Results Tree
- Summary Report





2.8 Test Execution

The test plan was executed using the Start option in JMeter. The application was monitored during execution to ensure stability and consistent response behavior.

2.9 Test Results and Analysis

The performance test results showed that the Redmine application handled concurrent access to the home page and projects page with stable response times and minimal error rates. The throughput values indicated consistent handling of user requests under moderate load conditions.

3. Functional/UI Testing Using Python Selenium

3.1 Tool and Setup

- **Language:** Python
- **Library:** Selenium with WebDriver Manager
- **Browser:** Google Chrome
- **Purpose:** Automate navigation, login, project creation, and administrative actions

Install required packages: *pip install selenium webdriver-manager*

3.2 Selenium Automation Script

The script performs the following actions:

1. Open Redmine homepage
2. Click **Sign in** and login with admin credentials
3. Navigate to **Projects** and create a new project
4. Explore administrative sections:
    - Users
    - Groups (create a test group)
    - Roles and permissions → Permissions report

- Workflow
- Custom fields
- Enumerations → create a new enumeration value
- Settings
- Information

5. Navigate to **My page**
6. Close the browser

Full Python Script:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import time


# CONFIG
URL = "http://localhost:3000"
USERNAME = "admin"
PASSWORD = "redminesqe"
NEW_PROJECT_NAME = "Selenium Test Project 4"
NEW_GROUP_NAME = "Test Group 1"
NEW_ENUM_NAME = "Activity 2"
```

```
# Initialize Chrome
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
driver.maximize_window()

try:
    # Open Redmine homepage
    driver.get(URL)
    time.sleep(2)

    # Click "Sign in" link
    login_link = driver.find_element(By.LINK_TEXT, "Sign in")
    login_link.click()
    time.sleep(2)

    # Enter username and password
    username_input = driver.find_element(By.ID, "username")
    username_input.send_keys(USERNAME)

    password_input = driver.find_element(By.ID, "password")
    password_input.send_keys(PASSWORD)
```

```
# Click login button
login_button = driver.find_element(By.NAME, "login")
login_button.click()
time.sleep(3)

print("Login successful!")

# Navigate to Projects page
projects_link = driver.find_element(By.LINK_TEXT, "Projects")
projects_link.click()
time.sleep(2)
print("Projects page opened!")

# Create a new project
try:
    new_project_btn = driver.find_element(By.LINK_TEXT, "New project")
    new_project_btn.click()
    time.sleep(2)

    # Enter project name
    name_input = driver.find_element(By.ID, "project_name")
    name_input.send_keys(NEW_PROJECT_NAME)
```

```
    # Click Create button
    create_btn = driver.find_element(By.NAME, "commit")
    create_btn.click()
    time.sleep(2)
    print(f"New project '{NEW_PROJECT_NAME}' created successfully!")

except Exception as e:
    print("Could not create new project:", e)

# Re-open Projects page
try:
    projects_link = driver.find_element(By.LINK_TEXT, "Projects")
    projects_link.click()
    time.sleep(2)
    print("Projects page opened successfully!")
except Exception as e:
    print("Could not open projects page:", e)

# ------------ Admin Section -----------------
driver.find_element(By.LINK_TEXT, "Administration").click()
time.sleep(2)
```

```
# Users
driver.find_element(By.LINK_TEXT, "Users").click()
time.sleep(2)
print("Users page opened!")

# Groups
driver.find_element(By.LINK_TEXT, "Groups").click()
time.sleep(2)
try:
    driver.find_element(By.ID, "group_name").send_keys(NEW_GROUP_NAME)
    driver.find_element(By.NAME, "commit").click()
    time.sleep(2)
    print(f"Group '{NEW_GROUP_NAME}' created successfully!")
except Exception as e:
    print("Could not create new group:", e)

# Roles and Permissions
driver.find_element(By.LINK_TEXT, "Roles and permissions").click()
time.sleep(2)
try:
    driver.find_element(By.LINK_TEXT, "Permissions report").click()
```

```
    time.sleep(2)
    print("Permissions report opened!")
except Exception as e:
    print("Could not open Permissions report:", e)

# Workflow
driver.find_element(By.LINK_TEXT, "Workflow").click()
time.sleep(2)
print("Workflow page opened!")

# Custom fields
driver.find_element(By.LINK_TEXT, "Custom fields").click()
time.sleep(2)
print("Custom fields page opened!")

# Enumerations
driver.find_element(By.LINK_TEXT, "Enumerations").click()
time.sleep(2)
try:
    driver.find_element(By.LINK_TEXT, "New value").click()
    time.sleep(2)
    driver.find_element(By.ID, "enumeration_name").send_keys(NEW_ENUM_NAME)
```

```
        driver.find_element(By.NAME, "commit").click()
        time.sleep(2)
        print(f"Enumeration '{NEW_ENUM_NAME}' created successfully!")
    except Exception as e:
        print("Could not create new enumeration:", e)

    # Settings
    driver.find_element(By.LINK_TEXT, "Settings").click()
    time.sleep(2)
    print("Settings page opened!")

    # Information
    driver.find_element(By.LINK_TEXT, "Information").click()
    time.sleep(2)
    print("Information page opened!")

    # Navigate to "My page" before closing
    driver.find_element(By.LINK_TEXT, "My page").click()
    time.sleep(2)
    print("My page opened successfully!")

finally:
```

```
# Close the browser
driver.quit()
print("Browser closed.")
```

3.3 Observations and Notes

1. **Login not strictly enforced**: In this Docker deployment, it is possible to perform actions like creating a project or enumeration without logging in. This is **unusual behavior** and would be considered a **functional/security issue in a real deployment**.
2. Selenium automation successfully navigates through all administrative and functional pages.
3. Status messages in the script confirm actions were completed.

3.4 Functional Issues Identified

While performing Selenium testing, the following functional issues were noted:

1. **Actions allowed without login:** Creating projects or enumerations works without authentication.
2. **Navigation delays:** Some pages like Workflow and Custom Fields load slowly due to container setup.
3. **Tracker issue:** While creating a new tracker, the drop down does not show any default status to choose from. As a result, we cannot make a tracker and assign it to a project. The following error line appears on the screen: Default status cannot be blank. As a result, we cannot effectively add *ISSUES*.

TC ID	Name	Description	Expected Results	Actual Results	Status
TC01	Login functionality	Verify that a user can log in with valid credentials	User is logged in and redirected to the dashboard	User logged in successfully	Pass
TC02	Login functionality	Verify that a user can not log in with	An error message is displayed	Error message displayed	Pass

		valid credentials	stating the invalid entry used.		
TC03	Project Creation	Verify that a new project can be created	New project is created and listed	New project created successfully	Pass
TC04	Projects Without Login	Verify that creating a project without login is blocked	User is prompted to login	Project creation is possible without login	Fail
TC05	Users Page Access	Verify that Users page can be accessed	Users page is displayed	User page opened	Pass
TC06	Groups Creation	Verify that a new group can be created	Group is created and listed	Group created successfully	Pass
TC07	Roles & Permissions	Verify access to Roles and Permissions & Permissions Report	Roles and permissions are displayed; report opens	Report opened successfully	Pass
TC08	Enumerations Creation	Verify that a new enumeration can be added	Enumeration is created and listed	Enumeration created successfully	Pass
TC09	Enumerations without Name	Verify validation when creating an empty enumeration	Error message prevents creation	Error message is displayed	Pass

TC10	Tracker Creation	Verify tracker creation functionality	Tracker is created and can be assigned to a project	Default status cannot be blank error appears; cannot create tracker	Fail

4. API Testing Using Postman

4.1 Objective

The objective of this phase was to validate the **REST APIs** of the locally deployed **Redmine** application. API testing was performed to verify authentication, request handling, response correctness, and error handling using **Postman**.

4.2 Tool Used

- **Tool:** Postman
- **Application Under Test:** Redmine (Locally Deployed)
- **Base URL:** http://localhost:8080
- **Authentication:** API Key (X-Redmine-API-Key)

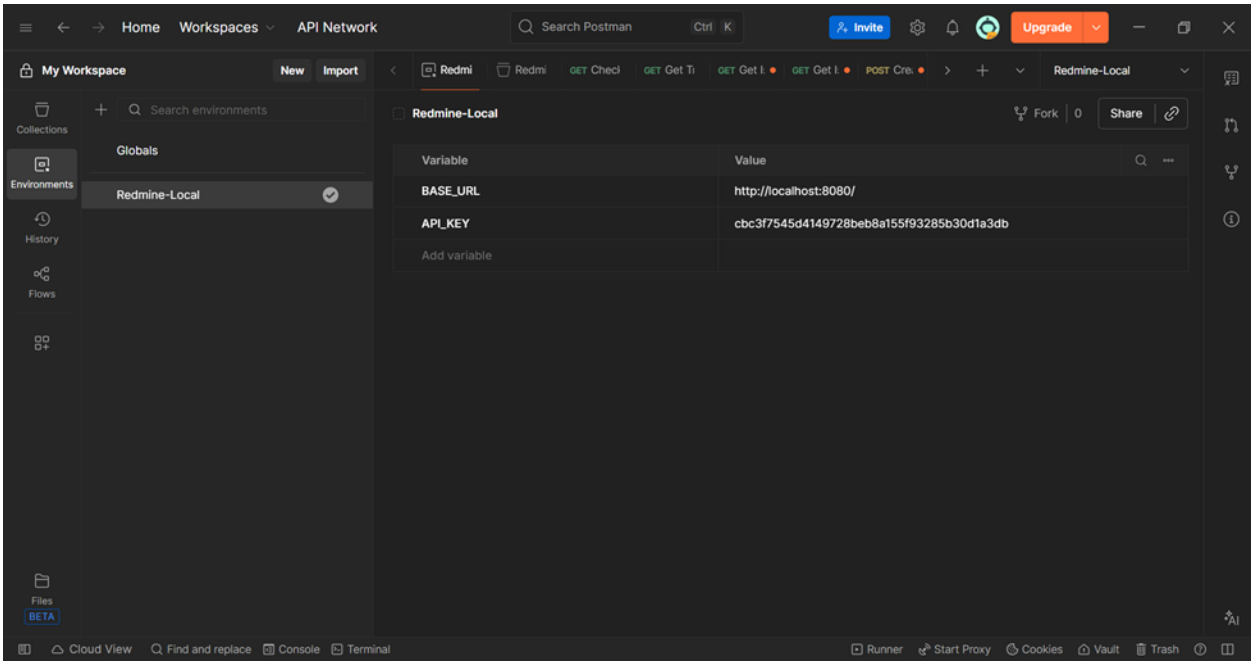
4.3 Postman Environment Configuration

To make the API requests reusable and configurable, a **Postman Environment** was created.

Environment Name

Redmine-Local

Variable Name	Value
BASE_URL	http://localhost:8080
API_KEY	(Redmine API Key)



Purpose:

Environment variables allow the same collection to be reused without hardcoding URLs or credentials.

4.4 Postman Collection Creation

A dedicated Postman collection was created to organize all API requests related to Redmine.

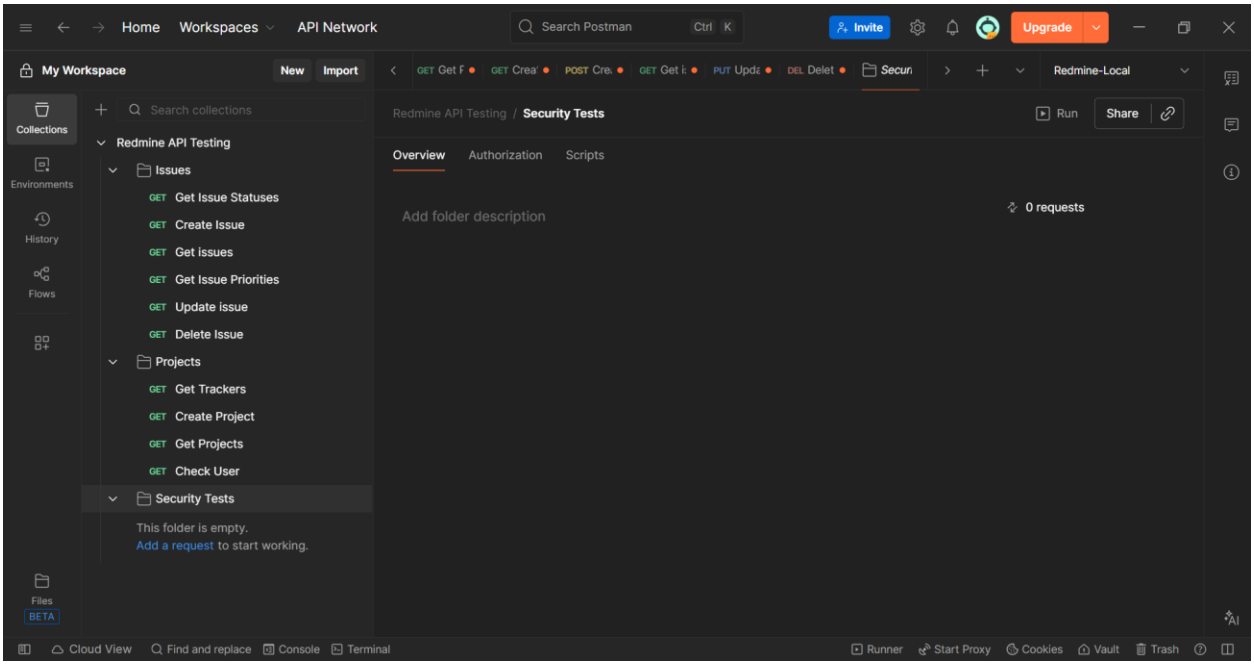
Collection Name

Redmine API Testing

Folder Structure

- Projects
- Issues
- Security Tests

Purpose:
Collections help structure API tests logically and improve maintainability.



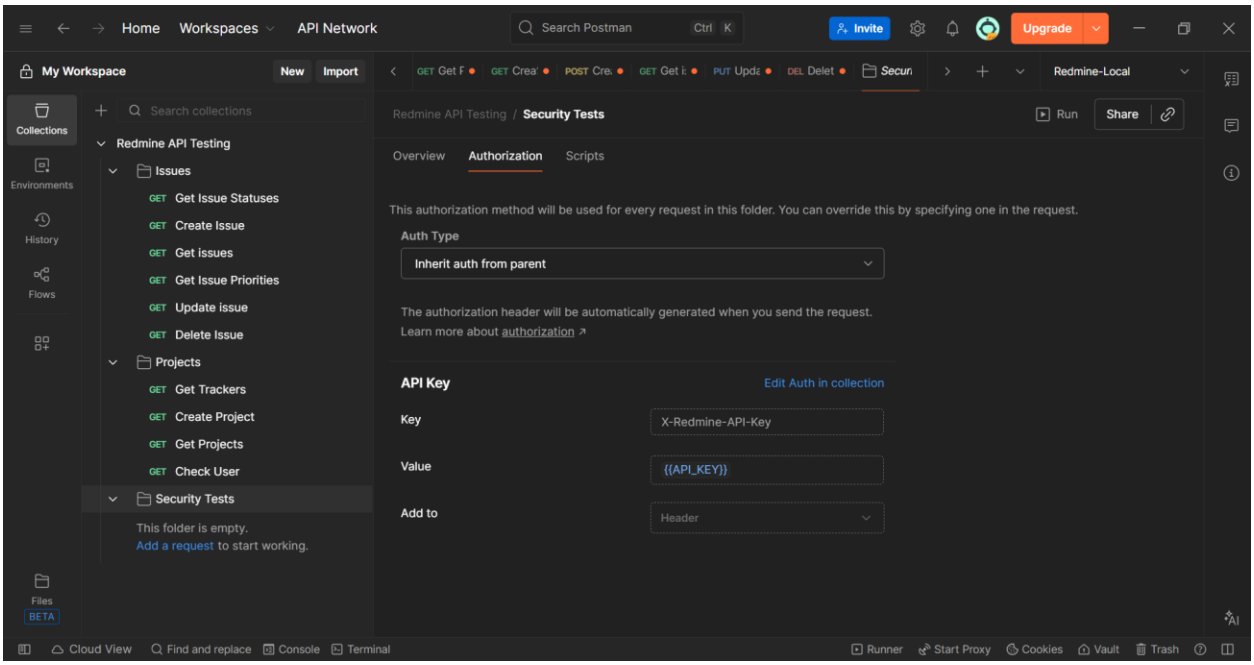
4.5 Authentication Setup (Collection Level)

Authentication was configured at the **collection level** to ensure all requests inherit the same credentials.

Authorization Configuration

- **Type:** API Key
- **Key:** X-Redmine-API-Key
- **Value:** {{API_KEY}}
- **Add to:** Header

This avoids repeating authentication headers in every request.



4.6 Global Headers Using Pre-Request Script

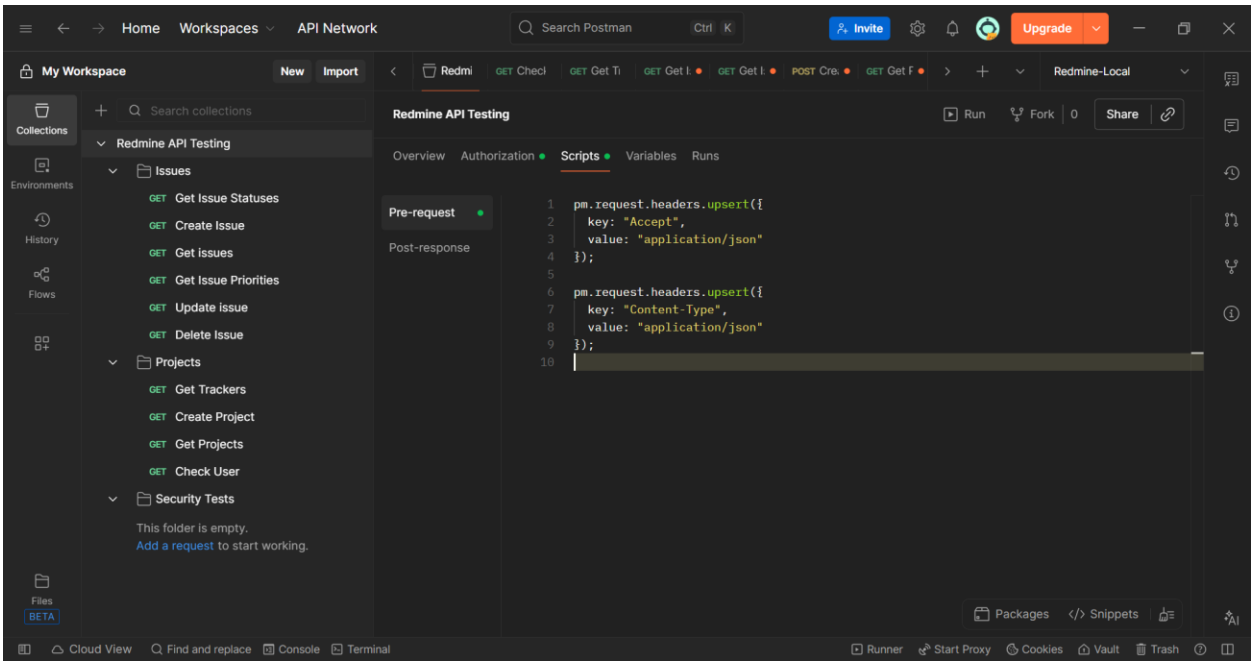
Since the latest Postman interface does not provide a global headers tab, headers were injected using a **Pre-request Script** at the collection level.

Pre-Request Script (Collection Level)

We added scripts so we don’t have to set content type & accept type in all headers

Script :

```
pm.request.headers.upsert({
  key: "Accept",
  value: "application/json"
});

pm.request.headers.upsert({
  key: "Content-Type",
  value: "application/json"
});
```



4.7 API Requests Implemented

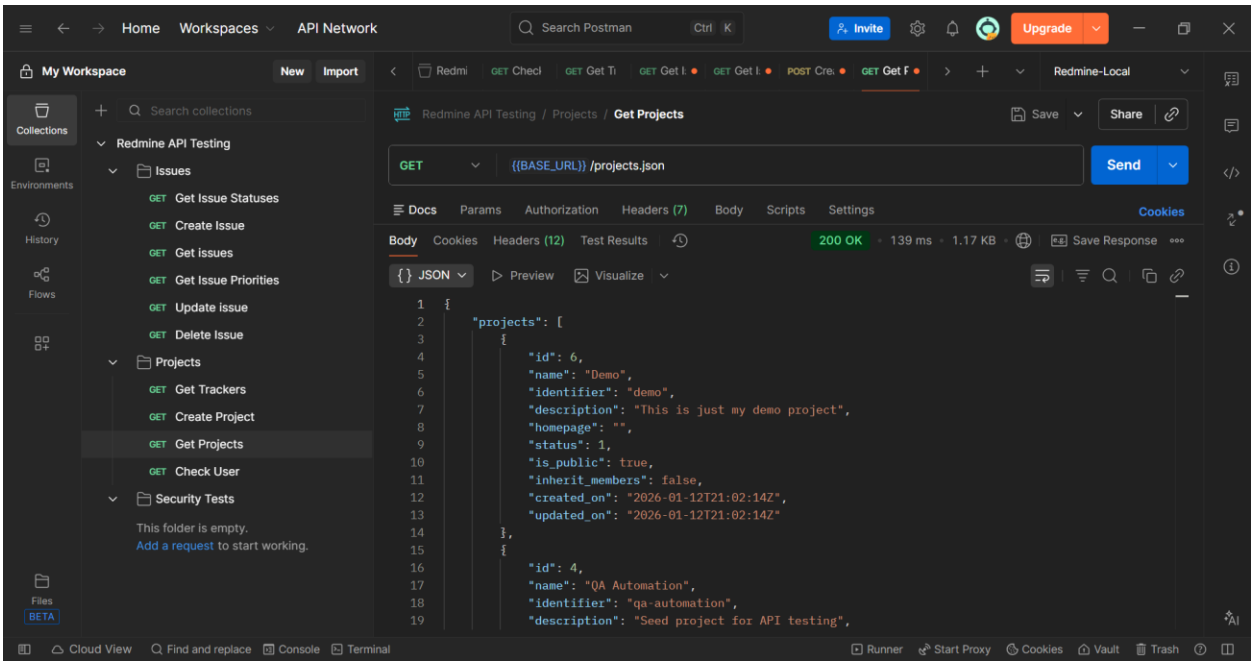
4.7.1 Get All Projects (GET)

Method: GET
Endpoint:

{{BASE_URL}}/projects.json

Expected Result:

- HTTP Status Code: 200 OK
- JSON response containing project list



4.7.2 Create Issue (POST)

Method: POST

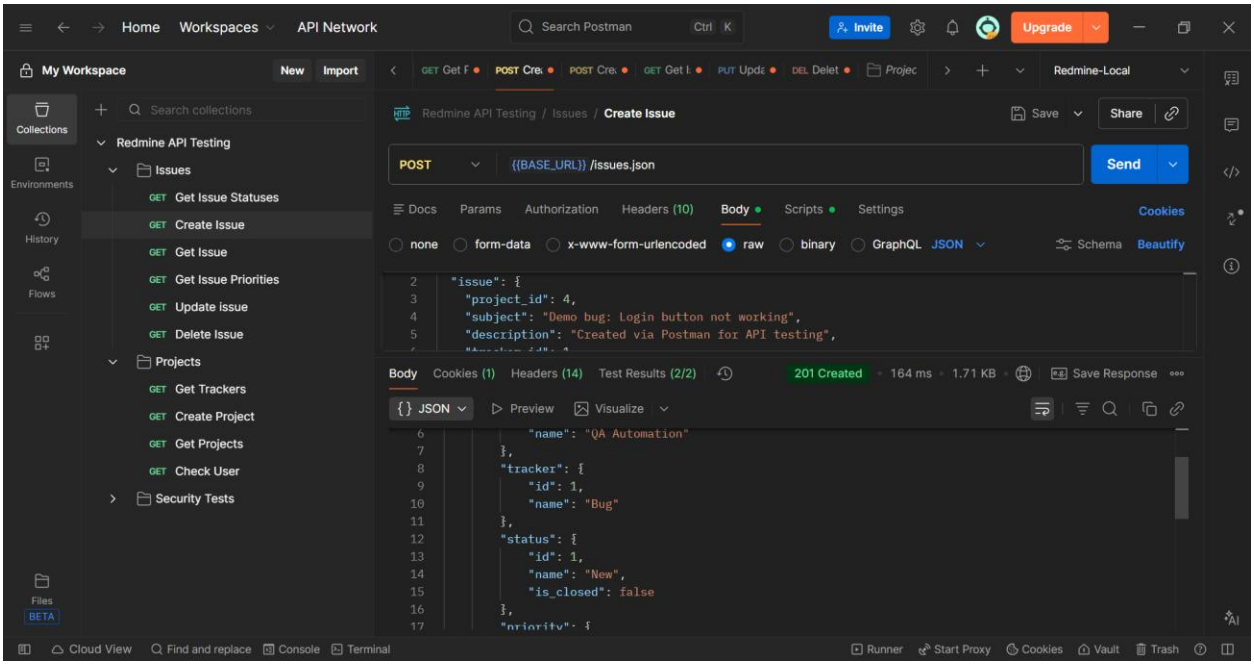
Endpoint:

{{BASE_URL}}/issues.json

Request Body (JSON)

```
{
 "issue": {
   "project_id": 4,
   "subject": "Demo bug: Login button not working",
   "description": "Created via Postman for API testing",
   "tracker_id": 1,
   "priority_id": 3,
   "status_id": 1
 }
```

}



4.7.3 Update Issue (PUT)

After successfully creating an issue, the **Update Issue** API was tested to verify that existing issues can be modified using the Redmine REST API.

Method: PUT

Endpoint

{{BASE_URL}}/issues/{{created_issue_id}}.json

(The created_issue_id variable is dynamically obtained from the Create Issue request.)

Request Body (JSON)

```
{
 "issue": {
   "status_id": 1,
   "notes": "Moved to In Progress via API",
```

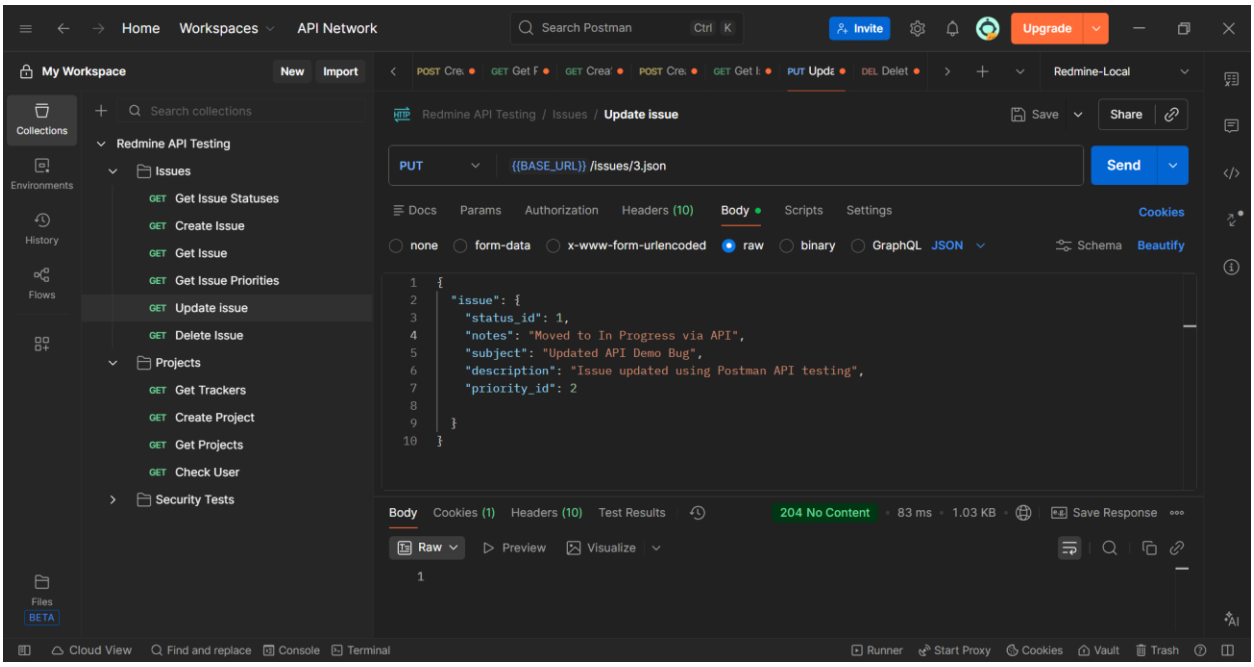
```
  "subject": "Updated API Demo Bug",
  "description": "Issue updated using Postman API testing",
  "priority_id": 2

 }
}
```

Expected Result

- **HTTP Status Code:** 204 No Content
- Issue details updated successfully
- No response body returned (as per Redmine API specification)

Purpose

This test verifies that the API supports updating existing issues and correctly applies changes to issue attributes.



4.7.4 Delete Issue (DELETE)

The **Delete Issue** API was tested to verify that issues can be removed using the REST API.

Method: DELETE

Endpoint

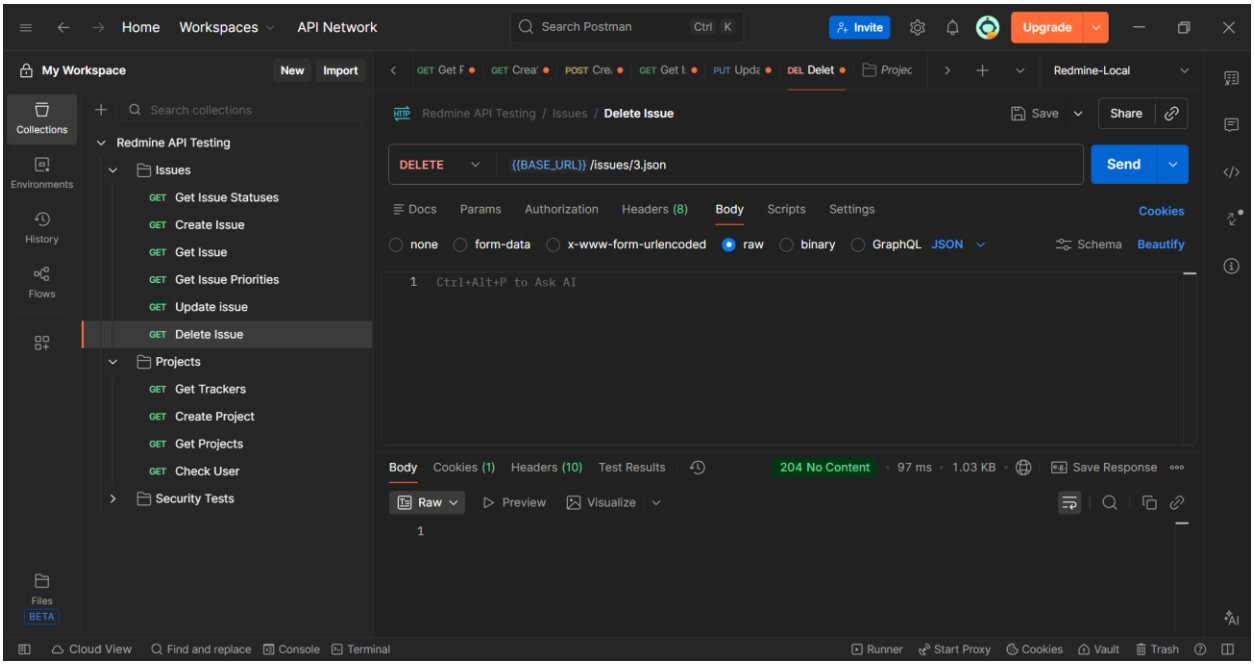
{{BASE_URL}}/issues/{{created_issue_id}}.json

Expected Result

- **HTTP Status Code:** 204 No Content
- Issue deleted successfully
- Issue no longer visible in Redmine UI

Purpose

This test validates the API's ability to delete resources and confirms proper cleanup of test data.



4.8 Pre-Request Script (Request Level)

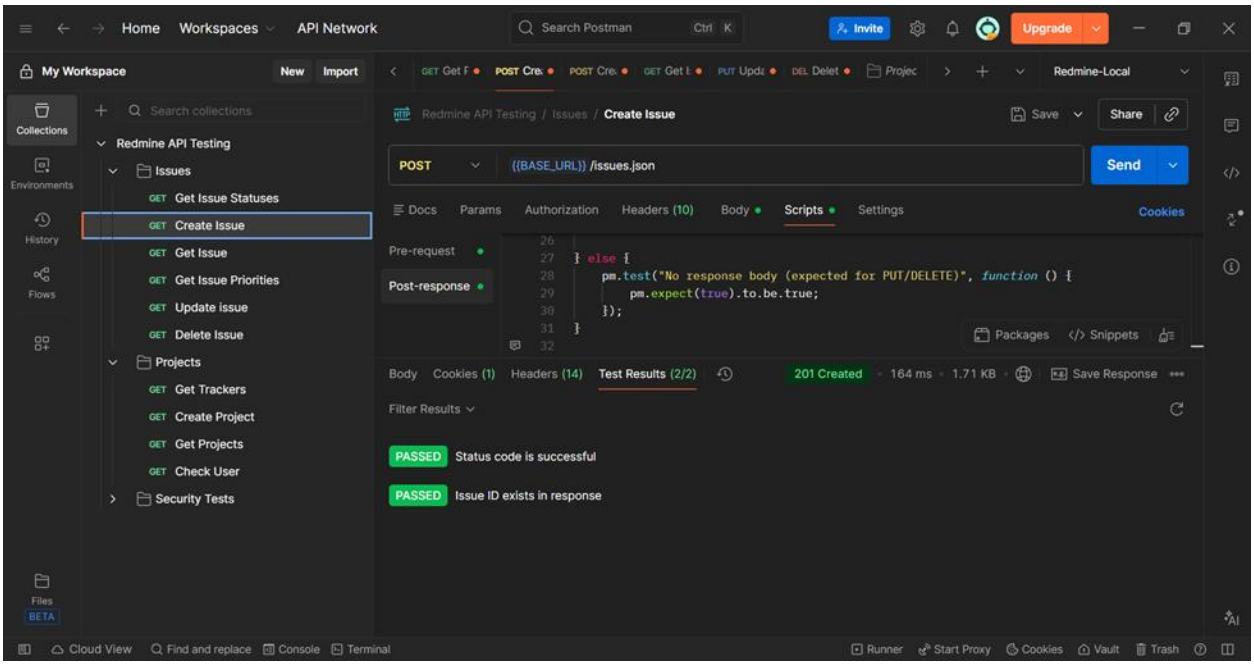
A **Pre-request Script** was added to dynamically generate unique issue data before sending the request.

Pre-Request Script

```
const randomId = Math.floor(Math.random() * 10000);

pm.variables.set("dynamic_subject", "API Demo Bug #" + randomId);


let body = {
  issue: {
    project_id: 4,
    subject: pm.variables.get("dynamic_subject"),
    description: "Generated via Postman pre-request script",
    tracker_id: 1,
    priority_id: 3,
    status_id: 1
  }
};

pm.request.body.raw = JSON.stringify(body);
```

Purpose

- Prevents duplicate data creation
- Enables reusable and dynamic API testing
- Improves automation and test reliability



4.9 Post-Request Script (Assertions / Tests)

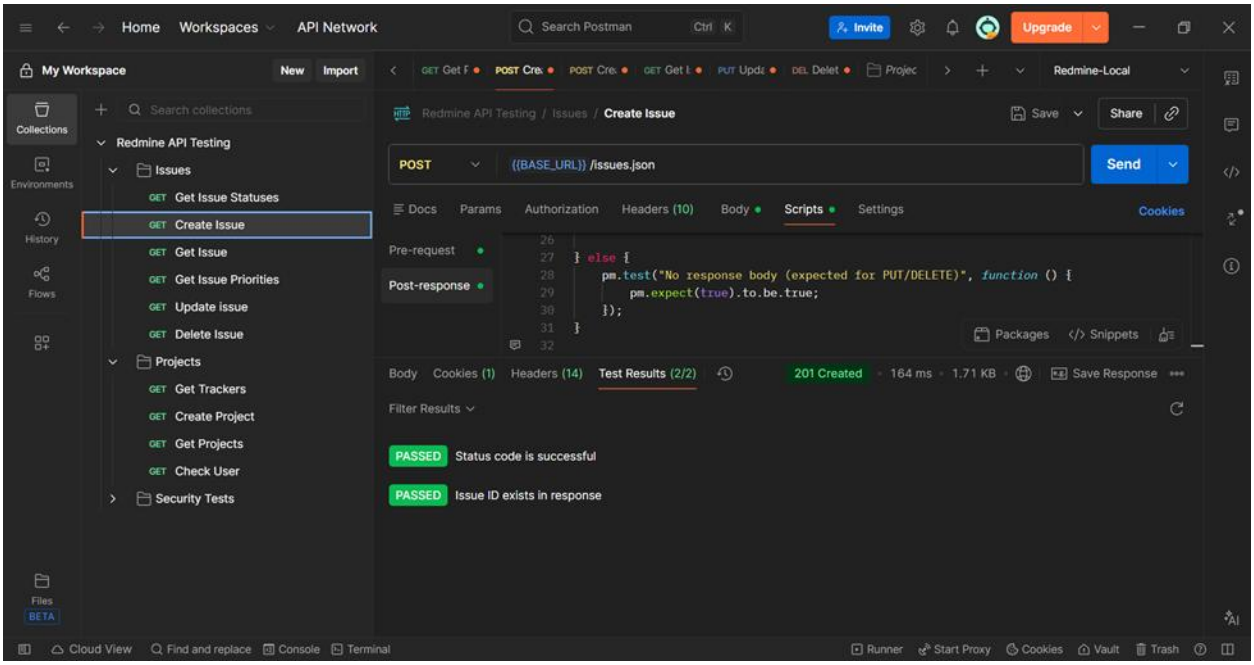
Postman **Tests scripts** were added to validate API responses and ensure correct behavior.

Tests Script (Create Issue)

```
pm.test("Status code is 201 Created", function () {
    pm.response.to.have.status(201);
});


let responseData = pm.response.json();


pm.test("Issue ID is returned", function () {
    pm.expect(responseData.issue.id).to.exist;
});
pm.environment.set("created_issue_id", responseData.issue.id);
```

Purpose

- Confirms successful request execution
- Validates response structure
- Stores issue ID for subsequent API calls



5. Security Testing Using OWASP ZAP

5.1 Objective

The objective of this phase was to perform **security testing** on the locally deployed Redmine application to identify common web vulnerabilities such as missing security headers, authentication weaknesses, and insecure configurations. OWASP ZAP was used to conduct both passive and active security scans.

5.2 Tool Used

- **Tool:** OWASP ZAP (Zed Attack Proxy)
- **Application Under Test:** Redmine (Locally Deployed)

Target URL:
http://localhost:8080

- **Testing Type:**
    - Passive Scanning
    - Active Scanning

5.3 OWASP ZAP Setup and Configuration

OWASP ZAP was configured as a local proxy to intercept and analyze HTTP traffic between the browser and the Redmine application.

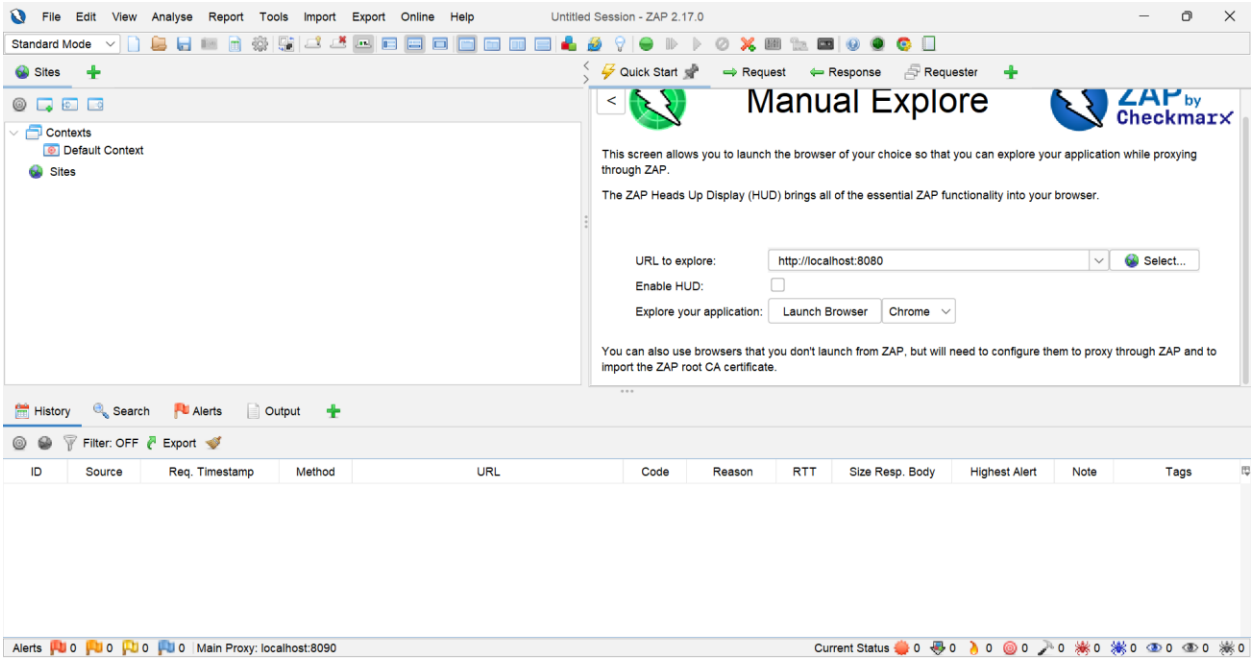
Proxy Configuration

- **Proxy Address:** localhost
- **Port:** 8090

The browser was configured to route traffic through the OWASP ZAP proxy so that all requests could be captured and analyzed.

Purpose:

Using ZAP as a proxy allows inspection of requests, responses, cookies, headers, and authentication mechanisms in real time.



5.4 Passive Scanning

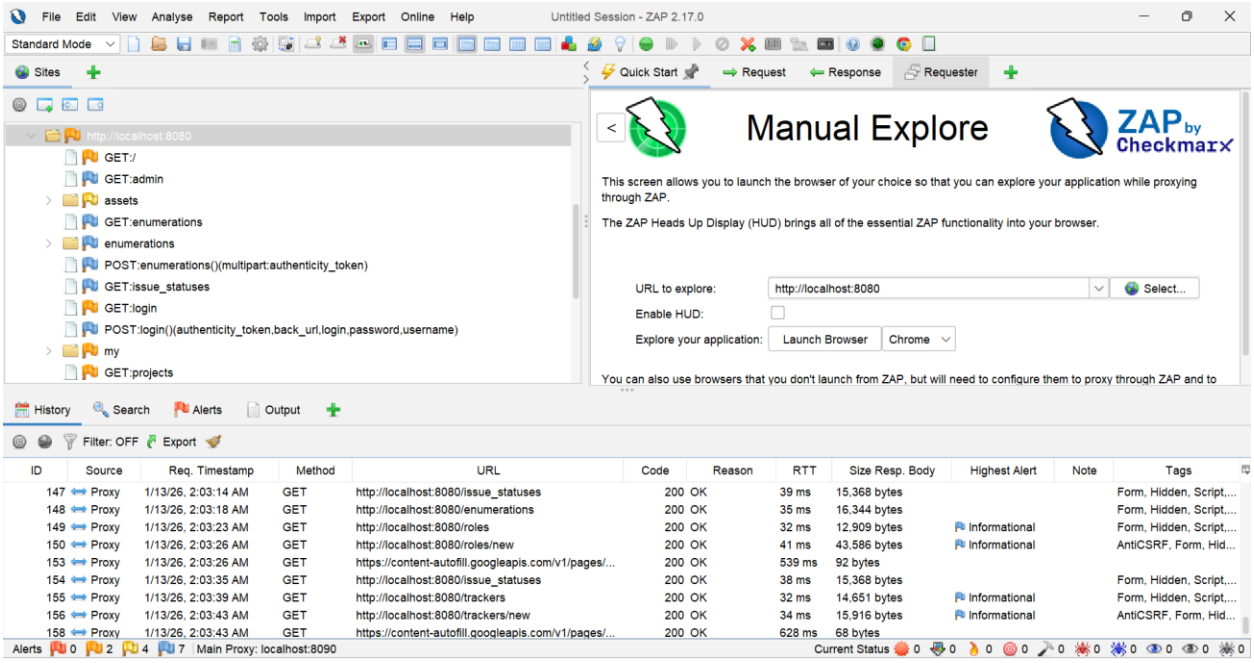
Passive scanning was performed automatically while manually interacting with the Redmine application through the browser.

Steps Performed:

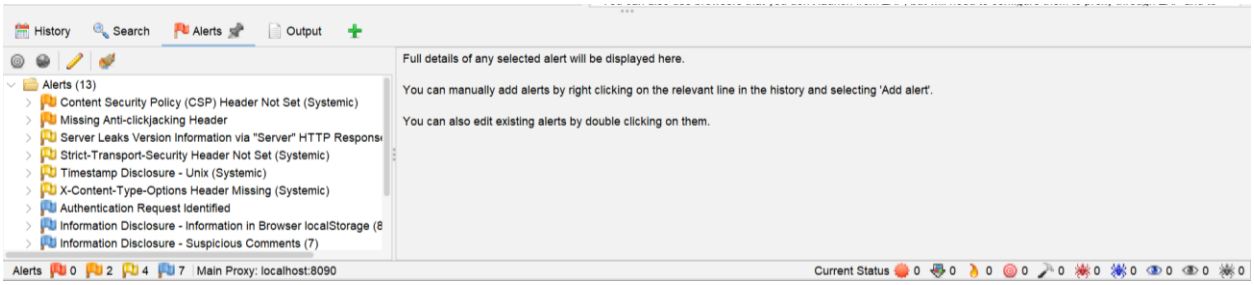
1. Logged into Redmine using valid credentials
2. Navigated through:
    - Projects
    - Issues
    - Create Issue form
    - User authentication pages
3. Observed traffic being captured in OWASP ZAP

Expected Outcome:

- Passive scan alerts generated automatically
- No impact on application behavior
- Identification of low-risk issues such as:
    - Missing security headers
    - Cookie attribute warnings



- ZAP “Alerts” tab with findings



5.5 Active Scanning

After passive scanning, an **active scan** was initiated to test for common web vulnerabilities.

Steps Performed:

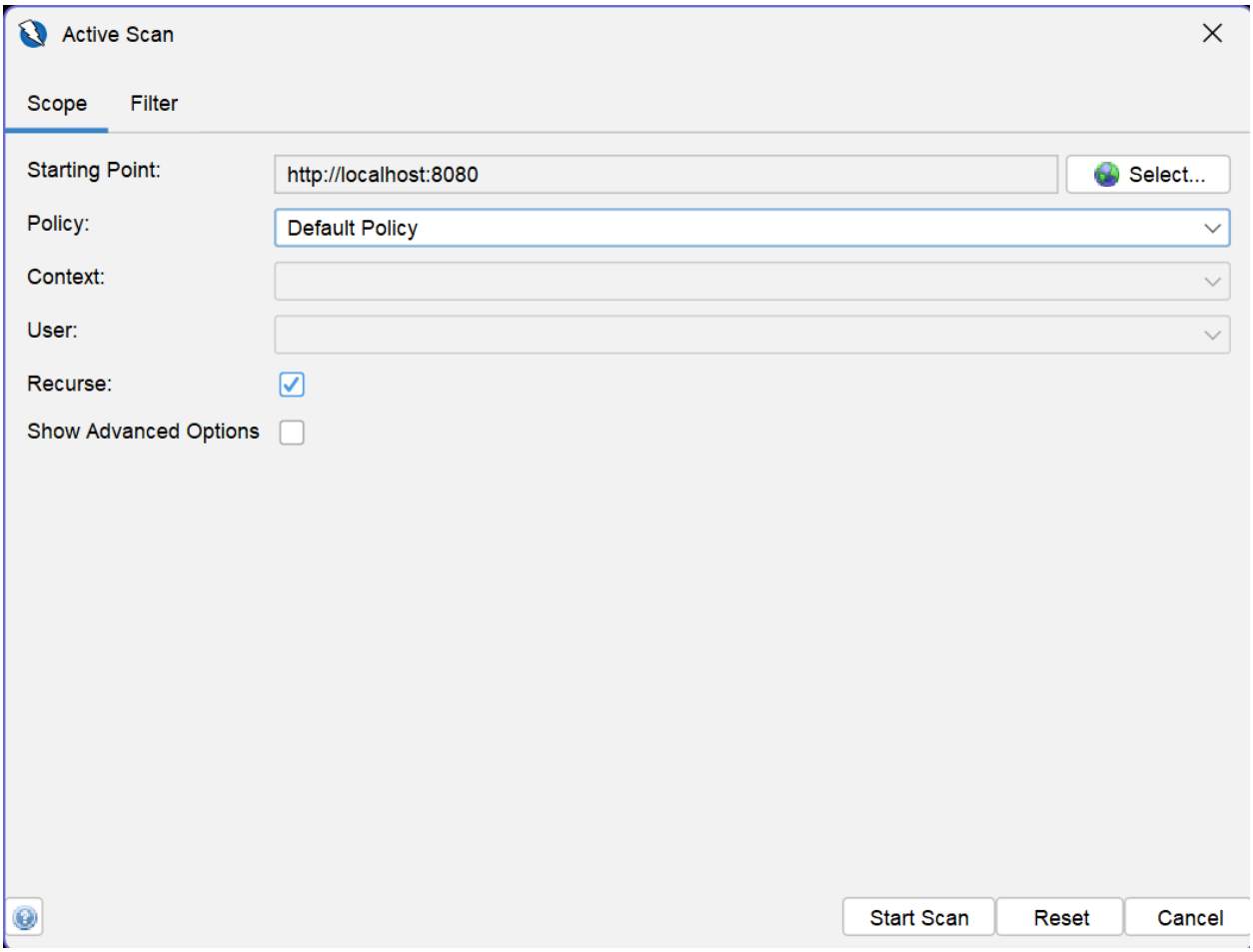
Right-clicked on the target:

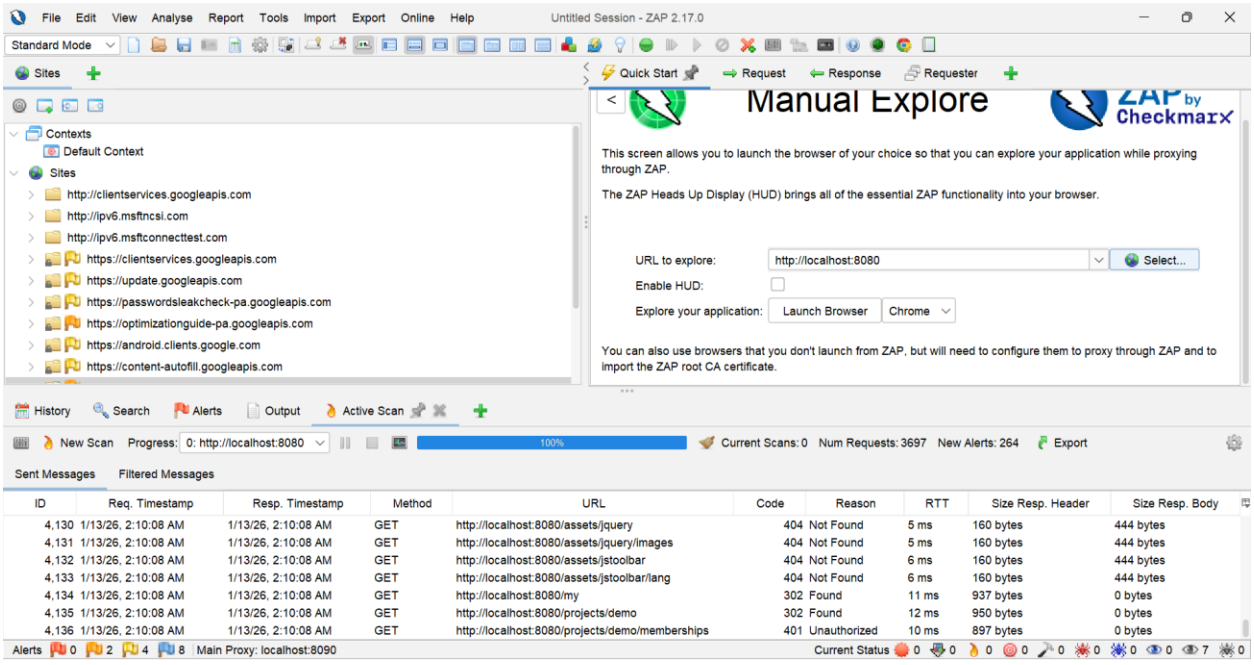
http://localhost:8080

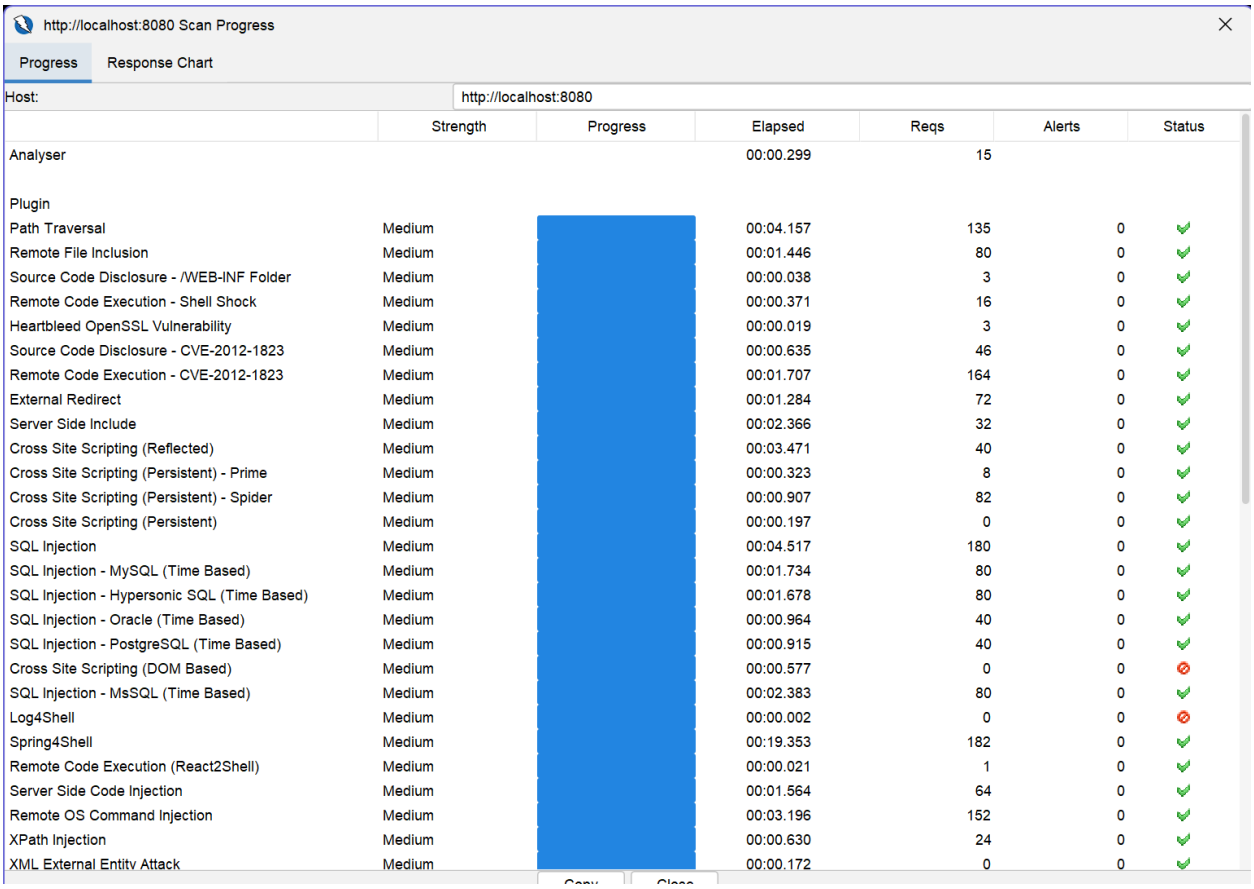
1. Selected **Attack → Active Scan**
2. Started scan with default policy

Expected Outcome:

- Automated vulnerability testing
- Detection of common OWASP Top 10 issues
- Scan completed without disrupting the application


Active Scan
Scope
Filter
Starting Point:
http://localhost:8080
Select...
Policy:
Default Policy
Context:
User:
Recurse:
Show Advanced Options
Start Scan
Reset
Cancel


File Edit View Analyse Report Tools Import Export Online Help
Untitled Session - ZAP 2.17.0
Standard Mode
Sites
Quick Start
Request
Response
Requester
Contexts
Default Context
Sites
http://clientservices.googleapis.com
http://ipv6.msftncsi.com
http://ipv6.msftconnecttest.com
https://clientservices.googleapis.com
https://update.googleapis.com
https://passwordsleakcheck-pa.googleapis.com
https://optimizationguide-pa.googleapis.com
https://android.clients.google.com
https://content-autofill.googleapis.com
Manual Explore
ZAP by Checkmarx
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.
URL to explore:
http://localhost:8080
Select...
Enable HUD:
Explore your application:
Launch Browser
Chrome
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.
History
Search
Alerts
Output
Active Scan
New Scan
Progress:
0: http://localhost:8080
100%
Current Scans: 0
Num Requests: 3697
New Alerts: 264
Export
Sent Messages
Filtered Messages
ID Req. Timestamp Resp. Timestamp Method URL Code Reason RTT Size Resp. Header Size Resp. Body
4,130 1/13/26, 2:10:08 AM 1/13/26, 2:10:08 AM GET http://localhost:8080/assets/jquery 404 Not Found 5 ms 160 bytes 444 bytes
4,131 1/13/26, 2:10:08 AM 1/13/26, 2:10:08 AM GET http://localhost:8080/assets/jquery/images 404 Not Found 5 ms 160 bytes 444 bytes
4,132 1/13/26, 2:10:08 AM 1/13/26, 2:10:08 AM GET http://localhost:8080/assets/jstoolbar 404 Not Found 6 ms 160 bytes 444 bytes
4,133 1/13/26, 2:10:08 AM 1/13/26, 2:10:08 AM GET http://localhost:8080/assets/jstoolbar/lang 404 Not Found 6 ms 160 bytes 444 bytes
4,134 1/13/26, 2:10:08 AM 1/13/26, 2:10:08 AM GET http://localhost:8080/my 302 Found 11 ms 937 bytes 0 bytes
4,135 1/13/26, 2:10:08 AM 1/13/26, 2:10:08 AM GET http://localhost:8080/projects/demo 302 Found 12 ms 950 bytes 0 bytes
4,136 1/13/26, 2:10:08 AM 1/13/26, 2:10:08 AM GET http://localhost:8080/projects/demo/memberships 401 Unauthorized 10 ms 897 bytes 0 bytes
Alerts
Main Proxy: localhost:8090
Current Status



	Strength	Progress	Elapsed	Reqs	Alerts	Status
Analyser			00:00.299	15		
Plugin						
Path Traversal	Medium		00:04.157	135	0	
Remote File Inclusion	Medium		00:01.446	80	0	
Source Code Disclosure - /WEB-INF Folder	Medium		00:00.038	3	0	
Remote Code Execution - Shell Shock	Medium		00:00.371	16	0	
Heartbleed OpenSSL Vulnerability	Medium		00:00.019	3	0	
Source Code Disclosure - CVE-2012-1823	Medium		00:00.635	46	0	
Remote Code Execution - CVE-2012-1823	Medium		00:01.707	164	0	
External Redirect	Medium		00:01.284	72	0	
Server Side Include	Medium		00:02.366	32	0	
Cross Site Scripting (Reflected)	Medium		00:03.471	40	0	
Cross Site Scripting (Persistent) - Prime	Medium		00:00.323	8	0	
Cross Site Scripting (Persistent) - Spider	Medium		00:00.907	82	0	
Cross Site Scripting (Persistent)	Medium		00:00.197	0	0	
SQL Injection	Medium		00:04.517	180	0	
SQL Injection - MySQL (Time Based)	Medium		00:01.734	80	0	
SQL Injection - Hypersonic SQL (Time Based)	Medium		00:01.678	80	0	
SQL Injection - Oracle (Time Based)	Medium		00:00.964	40	0	
SQL Injection - PostgreSQL (Time Based)	Medium		00:00.915	40	0	
Cross Site Scripting (DOM Based)	Medium		00:00.577	0	0	
SQL Injection - MsSQL (Time Based)	Medium		00:02.383	80	0	
Log4Shell	Medium		00:00.002	0	0	
Spring4Shell	Medium		00:19.353	182	0	
Remote Code Execution (React2Shell)	Medium		00:00.021	1	0	
Server Side Code Injection	Medium		00:01.564	64	0	
Remote OS Command Injection	Medium		00:03.196	152	0	
XPath Injection	Medium		00:00.630	24	0	
XML External Entity Attack	Medium		00:00.172	0	0	

5.6 Report Generation

After completing the scans, a security report was generated using OWASP ZAP.

Report Format:

- **HTML**

Steps:

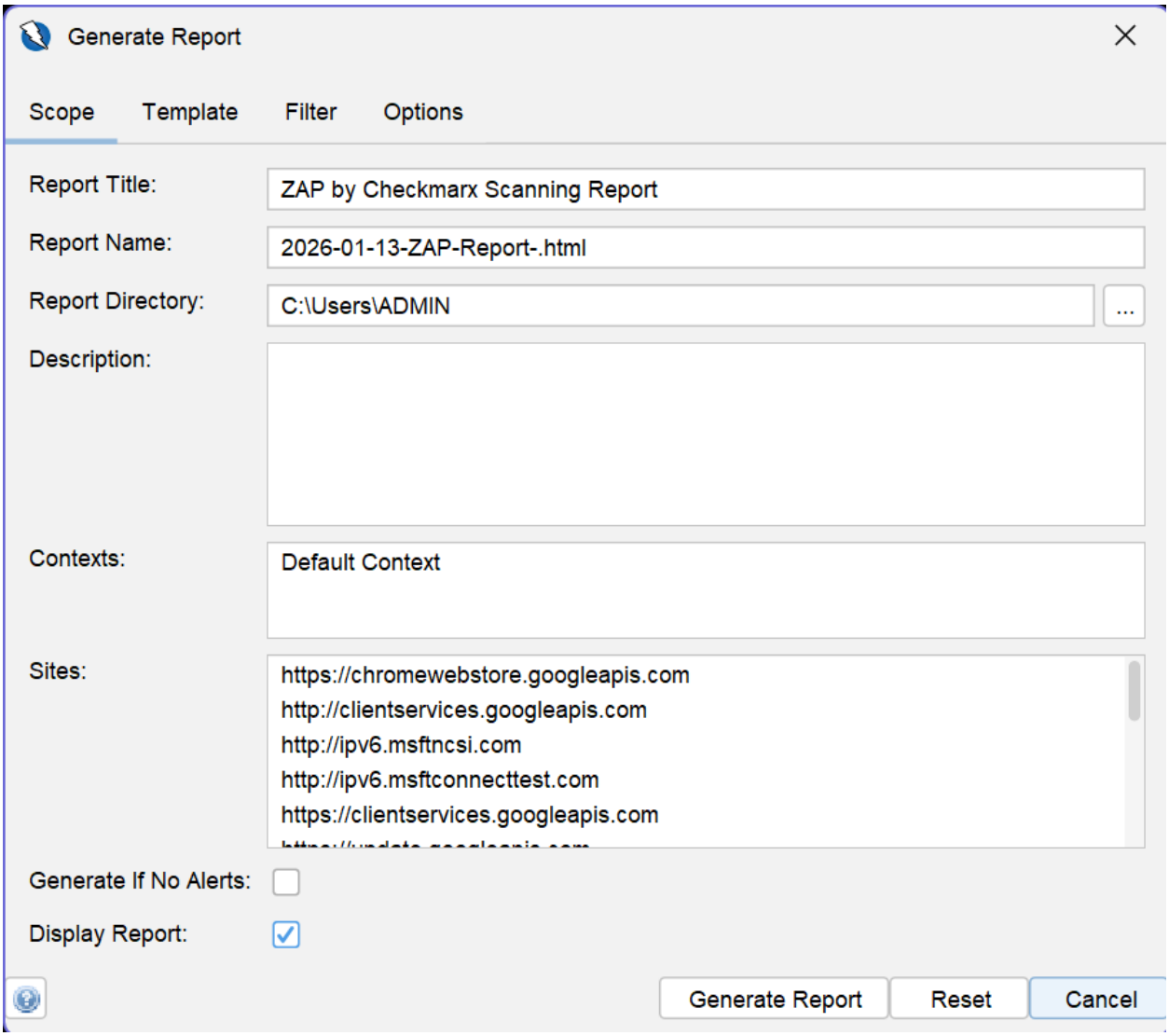
Navigate to:
Report → Generate Report

1. Select **HTML format**
2. Include all identified alerts

Save report

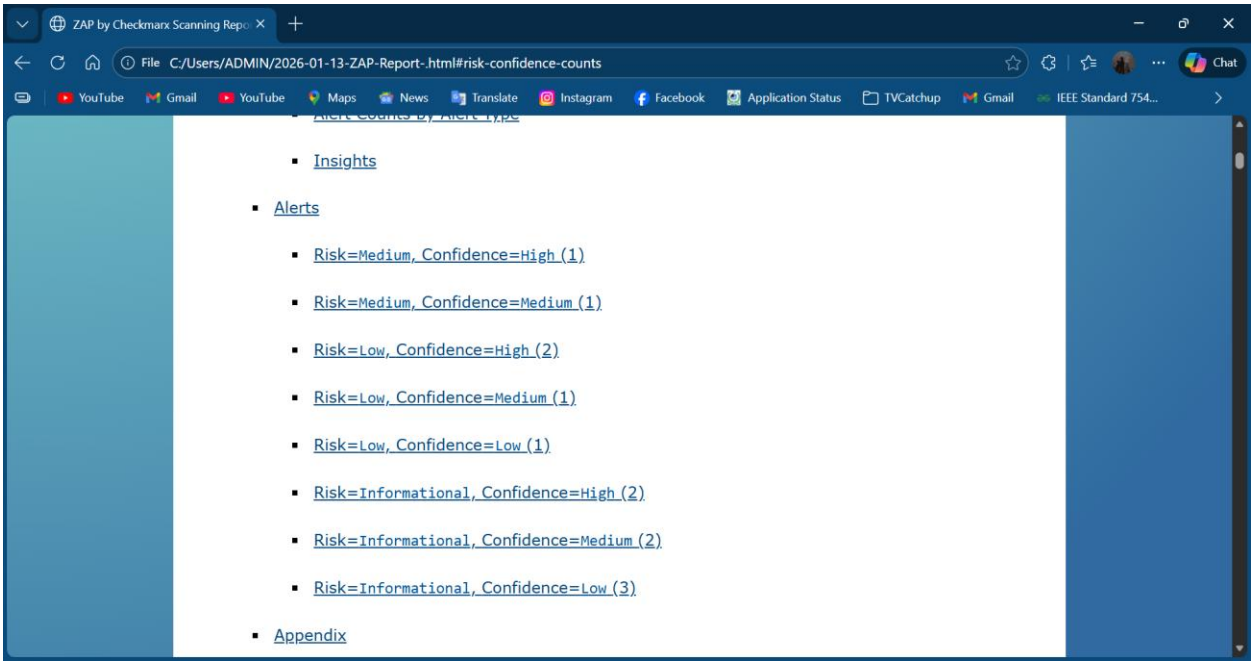
Purpose:

The report documents all identified vulnerabilities and serves as formal evidence of security testing.

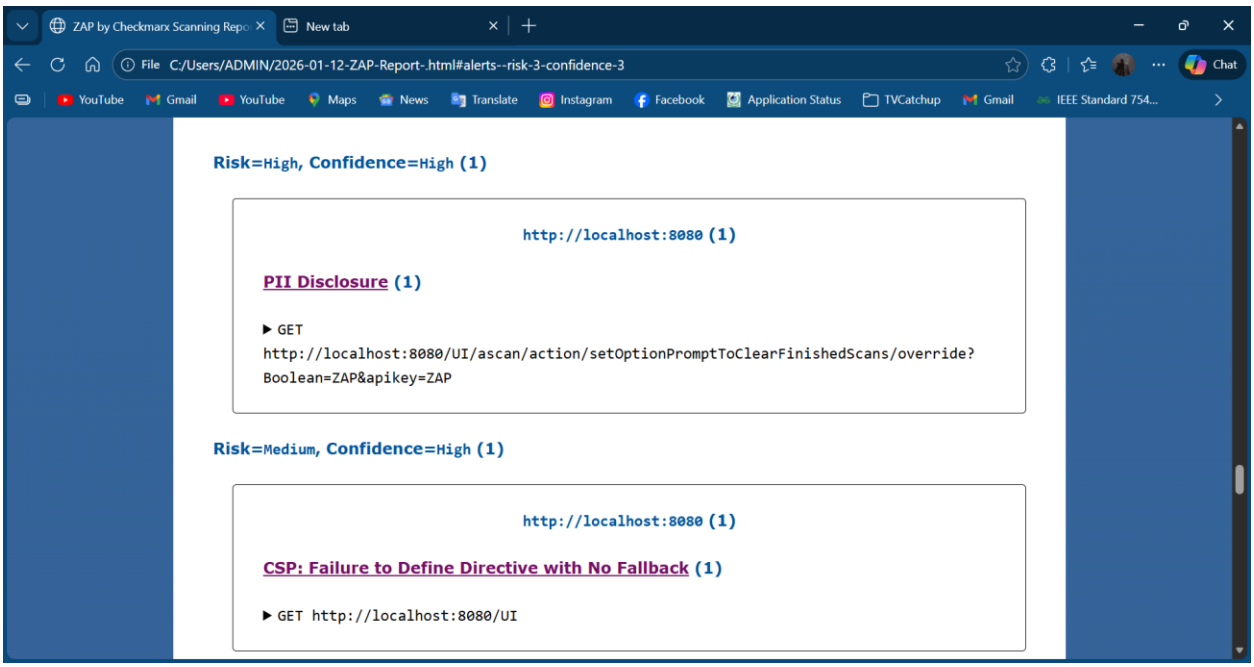


Report:

Alerts:



Some Alerts identified:



Content Security Policy (CSP) Header Not Set

Risk Level : Medium

Affected URL http://localhost:8080/

HTTP Method GET

Description

OWASP ZAP identified that the **Content Security Policy (CSP) HTTP header is not set** in responses returned by the Redmine application. CSP is an important security mechanism that helps prevent attacks such as **Cross-Site Scripting (XSS)** and **data injection attacks** by restricting the sources from which content can be loaded by the browser.

Without a properly configured CSP header, the application becomes more vulnerable to malicious scripts being executed in the user's browser, which could lead to data theft, session hijacking, or unauthorized actions.

Impact

- Increased risk of **Cross-Site Scripting (XSS)** attacks
- Potential execution of malicious JavaScript in the user's browser
- Reduced protection against client-side injection attacks

Personally Identifiable Information (PII) Disclosure

Risk Level : High

Confidence Level: High

Affected URL

http://localhost:8080/UI/ascan/action/setOptionPromptToClearFinishedScans/override

HTTP Method: GET

Description

OWASP ZAP identified a **Personally Identifiable Information (PII) Disclosure** issue during security testing of the Redmine application. The affected endpoint exposes sensitive information through a GET request, which could potentially reveal internal system details or user-related data without sufficient access controls.

PII disclosure vulnerabilities occur when applications unintentionally expose sensitive data such as usernames, internal identifiers, configuration values, or system-related information. This information can be leveraged by attackers to perform further targeted attacks against the application.

Impact

- Exposure of sensitive or internal information
- Increased risk of **information leakage**
- Potential violation of data protection and privacy principles

6. White-Box / Code Quality Testing Using SonarQube

6.1 Objective

The objective of this phase was to perform **white-box testing** and **static code analysis** on the Redmine codebase to identify code smells, maintainability issues, and potential security weaknesses. SonarQube was used to analyze the source code without executing the application.

6.2 Tool Used

- **Tool:** SonarQube
- **Scanner:** SonarScanner
- **Codebase Analyzed:** Redmine (Ruby on Rails)
- **Analysis Type:** Static (White-Box) Code Analysis

6.3 SonarQube Setup

SonarQube was installed locally and configured with the required Java environment.

Prerequisites

- Java Development Kit (JDK 25)
- SonarQube Server
- SonarScanner CLI

6.4 Starting SonarQube Server & Execution

SonarQube was started using the provided startup script.

StartSonar.bat

Once started, the SonarQube dashboard is accessible at:

http://localhost:9000

Default credentials:

Username: admin

Password: admin

After configuring the project, **SonarScanner** was executed to perform static code analysis on the Redmine source code. The scanner analyzed the entire codebase and uploaded the results to the SonarQube server for visualization and reporting.

Command Executed

```
sonar-scanner \
-Dsonar.projectKey=redmine \
-Dsonar.sources=. \
-Dsonar.host.url=http://localhost:9000 \
-Dsonar.token=<SONAR_TOKEN>
```

Execution Result:

The scan completed successfully with the following output:

```
EXECUTION SUCCESS
Analysis total time: 1:13.254 s
Total time: 1:14.466 s
```

All source files were analyzed without runtime execution of the application.

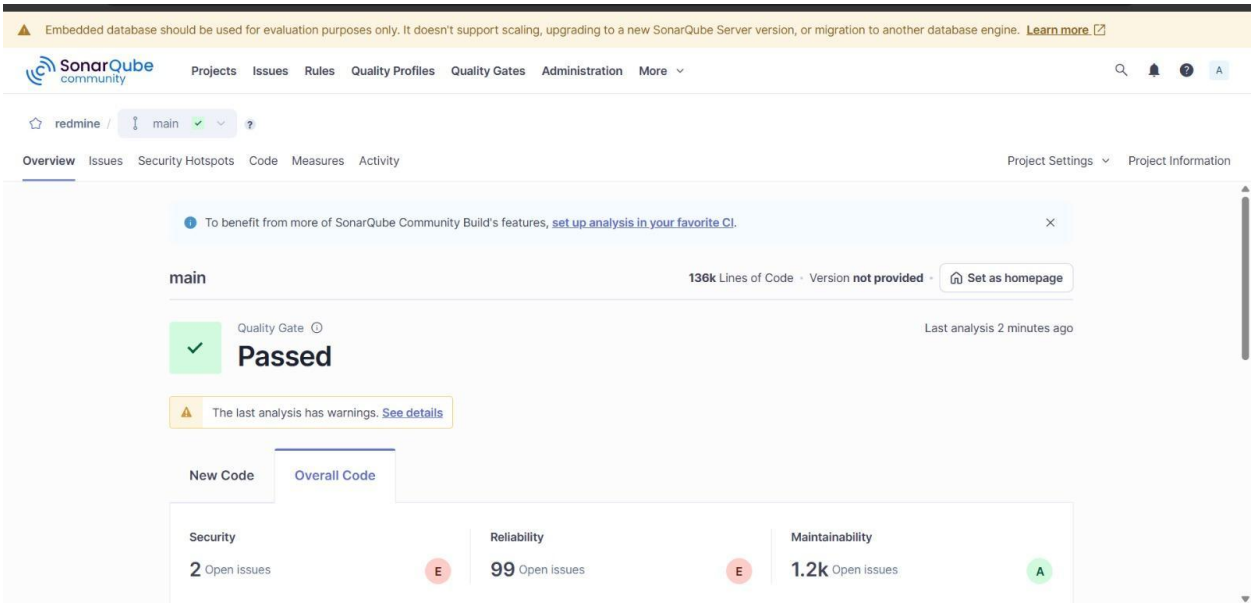
6.5 SonarQube Dashboard Overview

After successful analysis, the Redmine project appeared on the SonarQube dashboard. The **Quality Gate status was marked as "Passed"**, indicating that the codebase met the defined quality criteria.

Key observations from the dashboard:

- **Lines of Code:** ~136,000

- **Quality Gate:** Passed
- **Last Analysis:** Recently completed



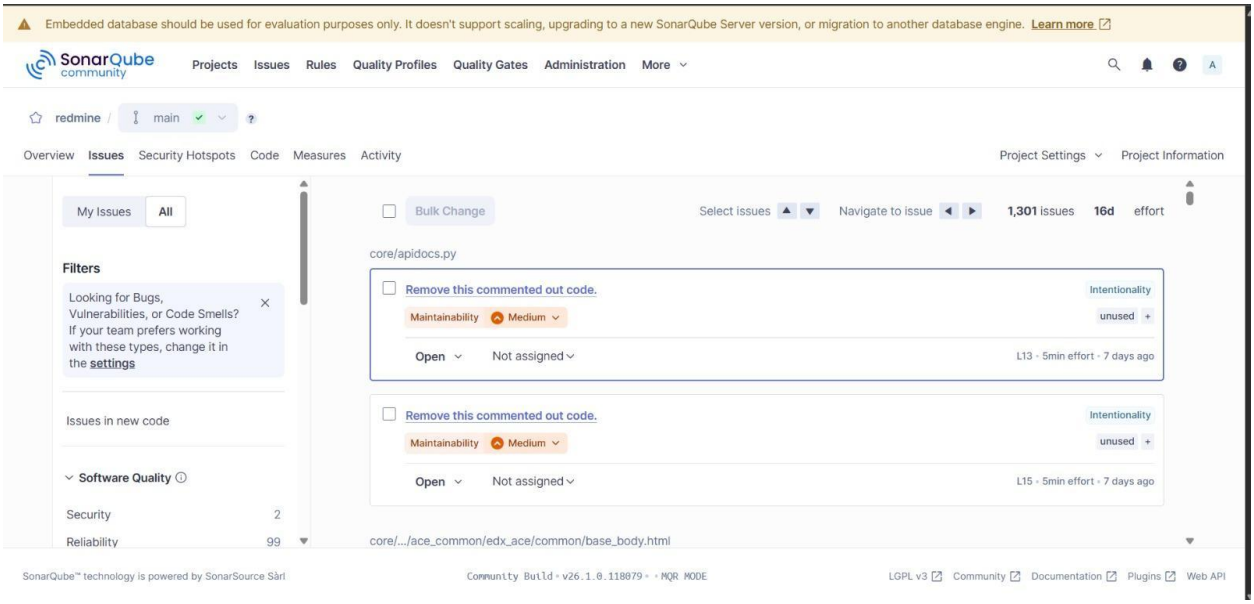
6.6 Issues Identified by SonarQube

SonarQube identified multiple issues across different quality dimensions:

6.6.1 Maintainability Issues

- Presence of commented-out code
- Unused variables and methods
- Code duplication in certain modules

These issues impact long-term maintainability and readability of the code.



6.6.2 Reliability Issues

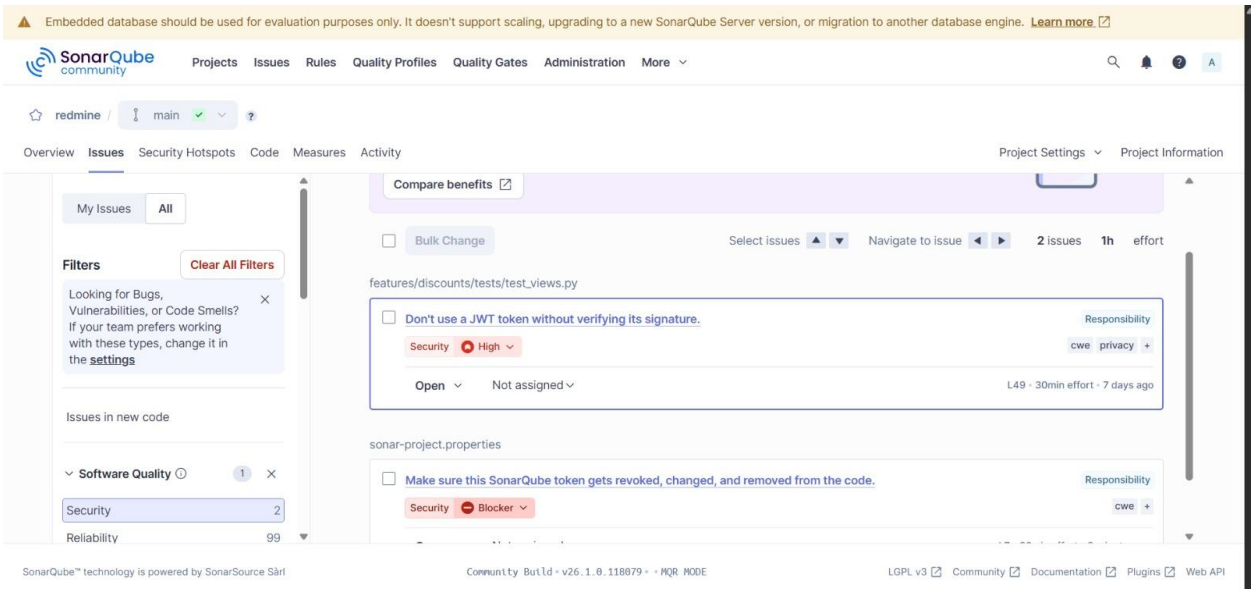
- Potential logic-related problems
- Code patterns that may lead to runtime errors

6.6.3 Security Issues

SonarQube flagged **high-severity security issues**, including:

- Use of JWT tokens without proper signature verification
- Hardcoded credentials detected in test files
- Exposure of sensitive tokens in configuration files

These issues represent potential security risks if deployed in a production environment.

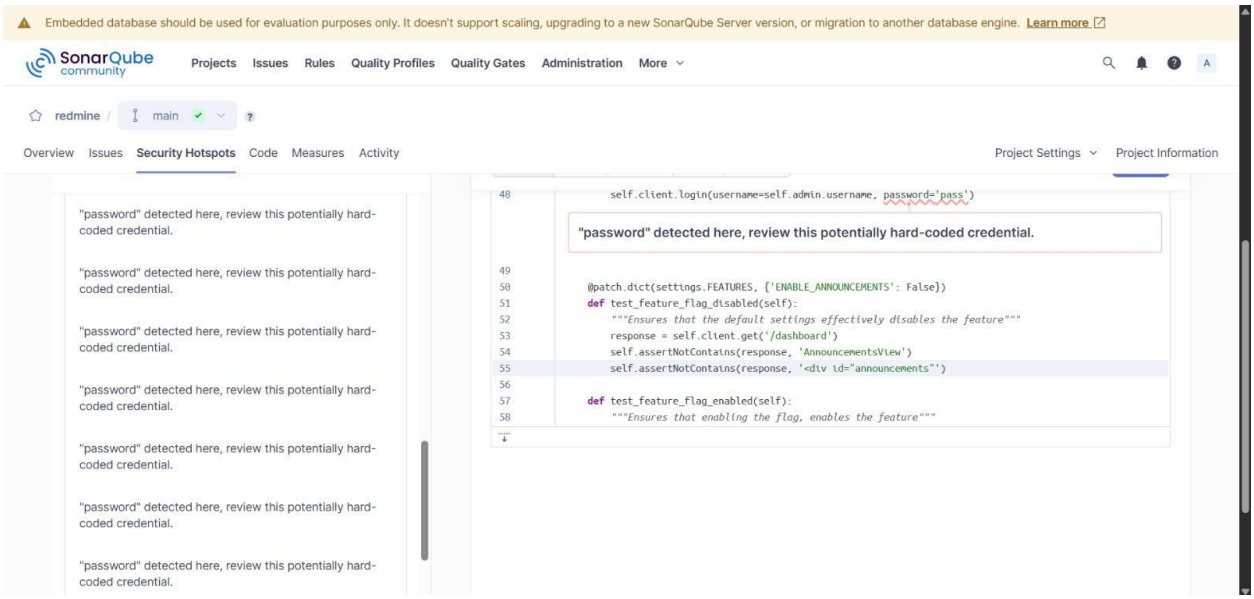


6.7 Security Hotspots

Security hotspots were also identified, highlighting code areas that require **manual review** to ensure secure implementation.

Examples include:

- Authentication logic
- Token handling mechanisms
- Credential usage in test file

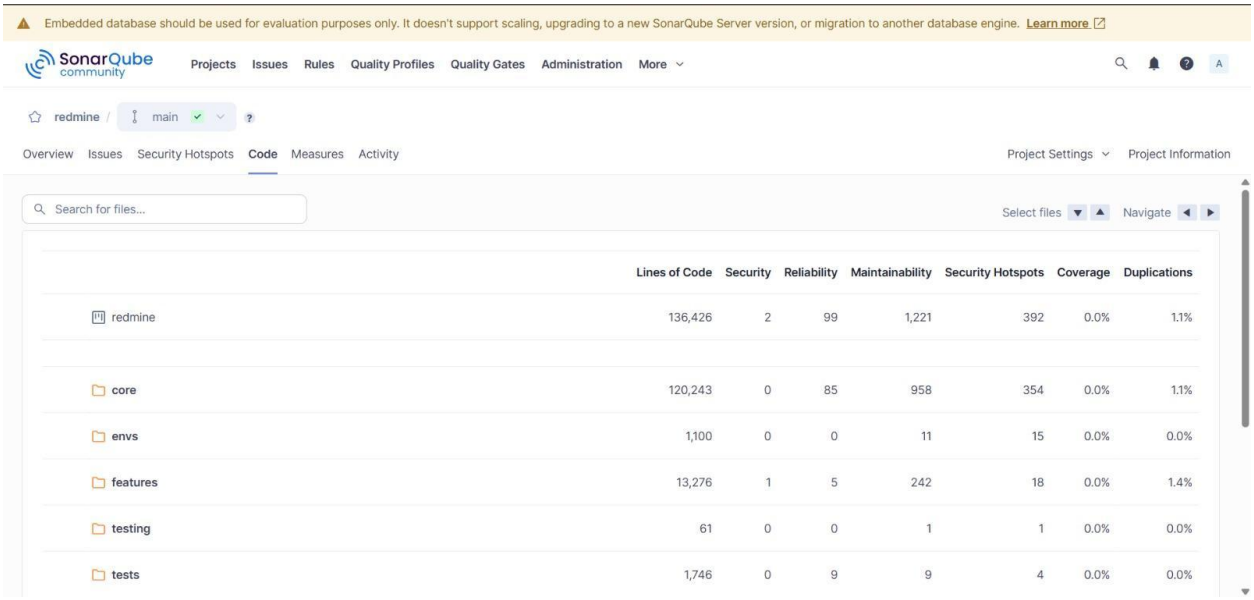


6.8 Code Structure and Metrics

The **Code** tab in SonarQube provided a structured breakdown of the Redmine project, including:

- Module-wise lines of code
- Number of issues per directory
- Duplication percentage
- Coverage statistics

This helped identify which parts of the codebase contributed most to technical debt.



6.9 Observations and Analysis

From the static analysis, the following observations were made:

- The overall code quality meets the defined quality gate
- Maintainability issues dominate the findings
- A small number of high-severity security issues require attention
- No execution or runtime testing was involved in this phase

The results demonstrate the effectiveness of static analysis in identifying internal code quality and security concerns.

Overall Conclusion:

This project implemented end-to-end software testing on a locally deployed open-source application, covering deployment, functional, API, security, performance, and static code analysis to ensure reliability, security, and maintainability.