**22F-3623, 22F-3715**

**BISAM AHMAD, IMAN FATIMA**

**Task no 3**

Our Final Year Project (FYP) is **"reqres.in"**, We’ll tailor the **Quality and Testing Requirements** document accordingly. The project is presumably an API service like **reqres.in**, which provides a set of RESTful APIs for testing purposes, specifically designed to simulate responses for educational and test automation purposes. Below is the customized document for your project.

## ****FYP project website name : reqres.in****

### ****Non-Functional Requirements for reqres.in Web APIs****

#### 1. **Performance Requirements**

Performance testing is crucial for APIs to ensure that they can handle the expected number of requests without any performance degradation. Since **reqres.in** is a simulated API service used for testing and educational purposes, we need to define performance benchmarks based on expected usage.

##### ****Performance Benchmarks:****

* **Response Time:**
* **< 200ms** for GET requests to return mock data, like user information.
* **< 500ms** for POST/PUT requests to simulate creating or updating data (e.g., user creation).
* **< 1 second** for DELETE requests to simulate resource deletion.
* **Throughput:**
* The API should be able to handle up to **5000 requests per second** (RPS) for GET requests (simulated data retrieval) during peak times.
* **Scalability:**
* The API should be scalable to handle a growing number of requests. This involves ensuring that it can run multiple instances behind a load balancer to manage **increased traffic**.
* **Concurrency:**
* The system should support **1000 concurrent requests** per second, simulating multiple users accessing or interacting with the API at once without slowing down.
* **Load Testing:**
* Performance tools like **Apache JMeter** or **Locust** should be used to validate these benchmarks by simulating the expected load, which could be a large number of simultaneous API calls.

#### 2. **Security Requirements for reqres.in Web APIs**

Since reqres.in is primarily used for testing purposes and is not a production-level system, security is still important, especially for teaching and ensuring developers are aware of vulnerabilities they may encounter in a real API.

**Key Security Requirements:**

* **Authentication and Authorization:**
* Since **reqres.in** is designed to simulate API responses, authentication mechanisms are typically simulated (e.g., by returning a fake token). However, for real-world applications, using **OAuth 2.0** with **JWT** tokens would be ideal for authentication and authorization of users.
* **Role-based access control (RBAC)** can be simulated to control access to different resources (e.g., admins managing user data, regular users accessing public data).
* **Input Validation:**
* Ensure that all incoming requests are validated, and proper checks are in place to prevent **SQL injection**, **XSS attacks**, and other input-related vulnerabilities.
* Use **JSON Schema Validation** to ensure that incoming request bodies confirm to the expected structure (for POST/PUT requests).
* **Data Encryption:**
* Enforce **SSL/TLS** encryption to ensure that any communication between clients and the API is encrypted.
* If reqres.in were to handle real sensitive data, it would be important to ensure **data encryption at rest** for storing sensitive information.
* **Rate Limiting:**
* Implement basic **rate limiting** to prevent abuse or **DoS (Denial of Service)** attacks by limiting the number of requests that can be made per user/IP within a certain time frame (e.g., 100 requests per minute).
* **Logging and Monitoring:**
* Implement **basic logging** of API calls, errors, and potential security breaches. These logs can be analyzed for suspicious activity or for improving the service.
* **OWASP API Security Top 10**:
* Implement a set of security checks based on the OWASP API Security Top 10 checklist to mitigate common API vulnerabilities.
* Example: **Broken Object Level Authorization** (ensuring users can only access their own data), **Broken Authentication**, **Excessive Data Exposure**.

#### Resources:

* OWASP API Security Top 10
* OWASP API Security Checklist

## ****Software Test Plan for reqres.in Web APIs****

The **test plan** for **reqres.in** focuses on various test strategies to ensure that all API endpoints function as expected, are secure, and can handle simulated traffic effectively.

### ****1. Test Strategy Overview****

The test plan will cover the following types of testing:

* **Unit Testing**:
* This will verify that individual components (e.g., endpoint handlers) are working as expected.
* **Tools**: **Jest**, **Mocha** (Node.js), **JUnit** (Java).
* **Integration Testing**:
* Test that the API endpoints integrate correctly with the back-end systems (e.g., database or mock services).
* **Tools**: **Postman**, **Supertest** (Node.js), **RestAssured** (Java).
* **Functional Testing**:
* Ensure the correct functionality of each endpoint by sending valid requests and checking if the responses are accurate.
* For example, testing GET requests for user data, POST requests for creating users, etc.
* **Tools**: **Postman**, **RestAssured**, **Jest + Frisby.js**.
* **Security Testing**:
* Simulate attacks like **SQL Injection**, **Cross-Site Scripting (XSS)**, and **Authentication Bypass** to ensure the API is secure.
* **Tools**: **OWASP ZAP**, **Burp Suite**.
* **Load Testing**:
* Measure the API’s response time and stability under varying load conditions (e.g., simulate 1000 concurrent requests).
* **Tools**: **Apache JMeter**, **Locust**, **Gatling**.
* **Regression Testing**:
* After implementing new features or changes, run tests to ensure that existing functionality is not broken.
* **Tools**: **Selenium** (for full-stack testing), **JUnit**, **Mocha**.

### ****2. Test Levels****

* **Unit Tests**:
* These tests will verify that individual components (functions, methods) of the API work as expected.
* Example: Testing if the response data format is correct for the GET /users endpoint.
* **API Endpoint Tests**:
* Test each individual API endpoint to ensure it responds as expected with correct data and status codes.
* Example: A POST /users request should return the newly created user data with HTTP status 201 (Created).
* **Security Tests**:
* Focus on testing for vulnerabilities such as broken authentication, SQL injection, data exposure, etc.
* Example: Test the POST /login endpoint for **broken authentication** by sending invalid credentials.
* **Load and Stress Tests**:
* Verify that the API can handle heavy traffic (e.g., thousands of concurrent requests) without significant performance degradation.
* Example: Test if the GET /users endpoint can handle **5000 requests per second**.

### ****3. Test Automation Tools****

* **Postman**:
* **Use Case**: Automate API testing and create collections of tests that can be run periodically or as part of a CI pipeline.
* **Capabilities**: Automated API request execution, validating responses, running assertions.
* **RestAssured** (Java):
* **Use Case**: API testing in Java-based environments.
* **Capabilities**: Validates JSON responses, tests authentication flows, performs status code checks.
* **Jest + Frisby.js**:
* **Use Case**: Testing APIs in Node.js environments, particularly with JSON responses.
* **Capabilities**: JSON data validation, status code validation, and mocking.
* **Apache JMeter**:
* **Use Case**: Perform **load testing** and **stress testing** to simulate real-world traffic conditions.
* **Capabilities**: Traffic simulation, performance bench-marking, analyzing server load.
* **OWASP ZAP / Burp Suite**:
* **Use Case**: Perform security scans to find vulnerabilities in the API.
* **Capabilities**: Vulnerability detection for issues like **Cross-Site Scripting**, **SQL Injection**, etc.

### ****4. Example Test Cases****

#### ****Test Case 1****: ****GET /users****

* **Objective**: Verify that the API returns a list of users.
* **Input**: Valid GET /users request.
* **Expected Result**: HTTP 200 OK, JSON response with an array of user data.

#### ****Test Case 2****: ****POST /users****

* **Objective**: Verify that the POST /users endpoint creates a new user.
* **Input**: Valid POST request with user data (e.g., name, job).
* **Expected Result**: HTTP 201 Created, response body contains the created user’s data.

#### ****Test Case 3****: ****POST /login****

* **Objective**: Verify that the login API returns a token with valid credentials.
* **Input**: Valid POST /login request with email and password.
* **Expected Result**: HTTP 200 OK, a valid JWT token in the response.

#### ****Test Case 4****: ****DELETE /users/{id}****

* **Objective**: Verify that the DELETE /users/{id} endpoint deletes a user.
* **Input**: Valid DELETE request with a specific user ID.
* **Expected Result**: HTTP 204 No Content, the user is deleted from the system.

**Summary:**

This document outlines the **quality and testing requirements** for the **reqres.in** project. By defining **performance benchmarks**, **security requirements**, and **creating a detailed test plan** with automated test strategies, you can ensure the **robustness** and **security** of the API. Proper testing will guarantee that **reqres.in** is reliable for educational purposes, simulating a wide variety of API interactions and traffic scenarios.