**1. Non-Functional Requirements**

**Performance Requirements**

Web APIs must be fast, reliable, and capable of handling varying amounts of traffic without breaking down. Here are some key performance requirements:

* **Response Time**: The average response time for any API request should not exceed 2 seconds, with a target of under 1 second for critical API calls.
* **Throughput**: The API should be able to handle at least 1000 requests per second (RPS) in normal operation, scaling to 5000+ RPS during peak loads.
* **Scalability**: The API should be designed to scale horizontally to handle increased traffic, either by adding more servers or using cloud-based scaling solutions like AWS Auto Scaling.
* **Load Testing**: The API should be able to handle the maximum anticipated number of users (e.g., 10,000 concurrent users) without degrading the service.
* **Uptime/Availability**: The API should maintain 99.99% uptime (max 4.38 hours downtime per year).

**Security Requirements**

Security is crucial for protecting sensitive data, ensuring the privacy of users, and preventing malicious attacks. Below are some critical security guidelines for a Restful API, based on the **OWASP API Security Top 10**:

* **Authentication & Authorization**:
  + Use **OAuth 2.0** or **JWT (JSON Web Tokens)** for secure, stateless user authentication.
  + Ensure the API supports role-based access control (RBAC), allowing specific endpoints to be accessed only by users with the appropriate permissions.
* **Data Encryption**:
  + Encrypt sensitive data both in transit (using **TLS/SSL**) and at rest (using **AES encryption** or similar strong encryption standards).
* **Input Validation**:
  + Validate and sanitize all incoming user inputs to prevent **SQL Injection** and **Cross-Site Scripting (XSS)** attacks.
* **Rate Limiting & Throttling**:
  + Implement rate limiting to prevent **Denial of Service (DoS)** attacks, restricting the number of requests a client can make in a given time window (e.g., 100 requests per minute).
* **Error Handling**:
  + Do not expose sensitive information in error messages (e.g., stack traces or database details). Instead, return generic error messages such as "Invalid request."
* **Logging and Monitoring**:
  + Implement centralized logging for all API requests and responses, especially for failed authentication attempts or suspicious activities.
  + Set up real-time monitoring and alerting for unusual traffic patterns or potential security breaches.
* **API Gateway**:
  + Use an API Gateway to handle routing, load balancing, security policies, and rate limiting. This acts as a middle layer between clients and your API.

For further guidance, refer to the **OWASP API Security Top 10** checklist, which provides comprehensive security best practices.

**2. Software Test Plan**

**Test Strategies**

Testing is critical for ensuring the functionality, performance, and security of your API. Here’s a breakdown of test strategies that you will use:

**Unit Testing**

* **Objective**: Test individual components or functions of the API, such as controller methods, utility functions, etc.
* **Tools**: Use frameworks like **JUnit** (Java), **pytest** (Python), or **Mocha/Chai** (JavaScript).
* **Test Focus**:
  + Validate the correctness of business logic.
  + Ensure individual API endpoints handle edge cases and return correct responses.

**Integration Testing**

* **Objective**: Test interactions between components, such as between the API and the database.
* **Tools**: Use **Postman**, **RestAssured** (Java), or **Supertest** (Node.js).
* **Test Focus**:
  + Ensure that the API integrates correctly with other services, databases, or external APIs.
  + Test data flow from the API to the database and ensure consistent updates or retrievals.

**Functional Testing**

* **Objective**: Test the overall functionality of the API endpoints to ensure they behave as expected.
* **Tools**: **Postman**, **SoapUI**, or **Jest**.
* **Test Focus**:
  + Validate that the API endpoints return the correct data, status codes (e.g., 200 OK, 400 Bad Request), and perform expected actions (e.g., POST creates, GET retrieves).

**Performance Testing**

* **Objective**: Ensure that the API performs well under load and stress.
* **Tools**: Use **JMeter**, **Gatling**, or **LoadRunner** for simulating traffic and measuring response times, throughput, and scalability.
* **Test Focus**:
  + Measure how the API handles increasing traffic.
  + Test API performance under normal and peak load conditions to ensure it meets the performance requirements.

**Security Testing**

* **Objective**: Test for vulnerabilities and ensure the API is secure from common attacks.
* **Tools**: Use **OWASP ZAP** (Zed Attack Proxy), **Burp Suite**, or **Postman** for security scanning.
* **Test Focus**:
  + Test for common vulnerabilities like **SQL injection**, **XSS**, **Cross-Site Request Forgery (CSRF)**, and improper authentication/authorization.
  + Check for adequate protection against DoS attacks through rate limiting and IP blocking.

**Usability Testing**

* **Objective**: Ensure the API is easy to use and integrates well with other tools and clients.
* **Tools**: **Swagger/OpenAPI** for documenting the API and **Postman** for testing.
* **Test Focus**:
  + Ensure that the API documentation is clear, consistent, and easy to follow.
  + Verify that users can easily understand how to interact with the API.

**Regression Testing**

* **Objective**: Ensure that new changes or features don’t break existing functionality.
* **Tools**: **JUnit**, **Selenium**, or **RestAssured**.
* **Test Focus**:
  + Test all previously functional API endpoints to ensure they still work after new code changes.
  + Run automated tests after each deployment to ensure nothing is broken.

**Tools for Test Automation**

Here are some tools you can use for test automation of your Restful APIs:

1. **Postman**: For functional, security, and regression testing of your API.
2. **RestAssured**: A Java-based library for automated testing of REST APIs.
3. **JMeter**: For load testing and performance testing of your API.
4. **OWASP ZAP**: A security testing tool to find vulnerabilities in your API.
5. **JUnit**: For unit and integration testing of the backend logic and APIs.
6. **Cucumber**: For Behavior-Driven Development (BDD) testing, useful for testing API functionality with real user scenarios.
7. **Swagger/OpenAPI**: For documenting and testing your APIs.

**Test Coverage and Reporting**

* **Coverage Tools**: Use **JaCoCo** (for Java) or **coverage.py** (for Python) to track test coverage and ensure that all important code paths are covered.
* **Reporting Tools**: Use **Allure** or **ExtentReports** to generate detailed reports for automated tests, including success rates, failures, and execution logs.