### **Case Study: Performance Testing for GitHub API**

#### Background:

A software development company relies heavily on the GitHub API to integrate their custom tools and workflows with GitHub repositories. As the company's projects and development teams grow, there is a need to ensure that the GitHub API integration can handle the increasing number of requests, respond promptly, and scale effectively.

#### Objectives:

* Load Testing: Simulate a realistic number of API requests to evaluate the API's performance under normal usage.
* Rate Limit Analysis: Understand how the GitHub API rate limiting affects the application and identify potential issues.
* Response Time Analysis: Measure and analyze response times for critical API endpoints to identify performance bottlenecks.
* Scalability Assessment: Evaluate how the API scales with increasing request loads.

#### Steps:

1. Identify Scenarios:

* Create a Repository For an Authenticated user
* Get a Repository.
* Update a Repository.
* Create a File
* Update a File
* Delete a File.
* Delete a Repository.
* Get List of all Repositories.

2. Create JMeter Test Plan:

* Define Thread Groups for different API scenarios.
* Configure HTTP Request Samplers for each GitHub API endpoint.
* Utilize HTTP Header Managers to include required authentication tokens.
* Implement listeners (e.g., View Results Tree) for result analysis.

3. Load Testing:

* Simulate a realistic number of concurrent users making API requests.
* Monitor response times and error rates.
* Identify the maximum load the API can handle without degrading performance.

4. Rate Limit Analysis:(Timer)

* Exceed the API rate limits intentionally to observe how the application handles rate-limited scenarios.
* Evaluate the impact on user experience during rate-limiting conditions.
* Adjust the test scenarios to comply with GitHub API rate limits.

5. Response Time Analysis:

* Focus on critical API endpoints such as those dealing with code commits, issues, and collaborators.
* Use listeners and timers to analyze response times.
* Identify any slow-performing API endpoints.

6. Scalability Assessment:

* Gradually increase the number of concurrent API requests to assess how the API scales.
* Monitor server resources to identify scalability bottlenecks.
* Evaluate how the application performs as the request load grows.

#### Tools Used:

* JMeter: For creating and executing performance tests on GitHub API.
* GitHub API Tokens: For authentication and authorization.
* Monitoring Tools: (e.gGrafana) for real-time monitoring of server resources during tests.