**Load Test Plan for Live Streaming Product**

**1. Introduction**

* Objective: The objective of this load testing plan is to ensure that Company B’s API, integrated into Company A’s platform, can handle the expected high levels of concurrent users during live streams without degradation in performance. This plan will focus on simulating the load generated by authenticated and non-authenticated users, as well as testing the system's scalability and resilience.
* Scope:
  + API Load Testing: Validate the API's ability to handle concurrent requests, focusing on endpoints related to commenting, posting photos, and reacting to live streams.
  + Client Application Load Testing: Ensure that the client applications (Desktop Web, Mobile Web, iOS, Android, FireTV, AndroidTV, Roku, Chromecast) perform well under the expected load.
  + System Performance: Measure the response times, throughput, and resource utilization under various load conditions.
  + Scalability Testing: Evaluate the system's ability to scale up as more users access the platform during popular live streams.
  + Stress Testing: Determine the system’s breaking point by gradually increasing the load beyond expected limits.

**2. Test Objectives**

* Ensure that the API can efficiently manage a high volume of concurrent requests, particularly on endpoints related to commenting, posting photos, and reacting to live streams.
* Verify that all client applications (Desktop Web, Mobile Web, iOS, Android, FireTV, AndroidTV, Roku, Chromecast) maintain acceptable performance levels when subjected to expected user loads.
* Determine the system's response times under different load conditions, ensuring that they remain within acceptable thresholds during peak usage.
* Analyze the system’s ability to handle a high rate of transactions per second, ensuring that it meets the required throughput levels during live stream events.
* Track the error rates during load testing to identify any potential issues with API requests or client application interactions that may arise under high traffic conditions.
* Conduct stress testing to gradually increase the load beyond normal operational limits to determine the system’s breaking point and identify the maximum load it can sustain before failure.

**3. Tools**

* [LoadNinja](https://support.smartbear.com/loadninja/docs/): For simulating API load and testing performance under various conditions.

**4.1 API Load Testing**

* Simulate Concurrent Users:
  + Simulate 5,000 authenticated users posting comments, reacting with emojis, and uploading photos.
  + Simulate 25,000 non-authenticated users viewing the live stream and interacting with the content (but not posting).
* Response Time Monitoring:
  + Measure the average response time for key API endpoints (e.g., posting comments, uploading photos).
* Throughput Testing:
  + Measure the number of transactions per second the API can handle under peak load.
* Error Rate Monitoring:
  + Monitor the error rate (e.g., 500 Internal Server Error, 404 Not Found) during the load tests to identify bottlenecks.

**4.2 Client Application Load Testing**

* Cross-Platform Load Simulation:
  + Simulate load across all supported platforms (Desktop Web, Mobile Web, iOS, Android, FireTV, AndroidTV, Roku, Chromecast).
  + Test user interactions such as loading the stream, posting comments, and reacting with emojis.
* Latency Testing:
  + Measure the time it takes for users to interact with the stream (e.g., time to load comments, time to post a reaction).
* Device-Specific Load Testing:
  + Simulate high usage on specific devices (e.g., mobile vs. desktop) to identify any platform-specific performance issues.

**4.3 Stress Testing**

* Gradual Load Increase:
  + Gradually increase the number of simulated users beyond the expected peak (e.g., 10,000 authenticated and 50,000 non-authenticated users).
  + Identify the point at which the system performance starts to degrade significantly.
* Resource Monitoring:
  + Monitor CPU, memory, and network usage on the servers during the stress test.
* Failure Point Identification:
  + Determine the maximum load the system can handle before failure (e.g., response time exceeds acceptable thresholds, error rates spike).

**4.4 Scalability Testing**

* Load Balancer Testing:
  + Test the system's ability to distribute the load across multiple servers efficiently.
* Auto-Scaling Testing:
  + Verify that the system can automatically scale resources (e.g., spinning up new server instances) in response to increased load.
* Database Performance:
  + Evaluate the performance of the database under load, particularly the ability to handle a high volume of concurrent write operations (e.g., posting comments).

**5. Test Execution**

* Environment:
  + Perform load tests in a staging environment that closely replicates the production environment.
* Execution Schedule:
  + Schedule load tests during off-peak hours to avoid disruption of the live environment.
  + Conduct regular load tests before major releases or after significant infrastructure changes.
* Load Test Duration:
  + Each load test scenario will run for a minimum of 1 hour to capture performance metrics under sustained load.

**6. Reporting**

* Metrics to Capture:
  + Response time (average, 95th percentile)
  + Throughput (requests per second)
  + Error rate (percentage of failed requests)
  + System resource utilization (CPU, memory, network)
  + Scalability metrics (load balancer performance, auto-scaling effectiveness)
* Test Reports:
  + Generate detailed reports after each test run, including graphs and analysis of key performance metrics.
  + Summarize findings, identify bottlenecks, and recommend improvements.
* Dashboard:
  + Use LoadNinja performance monitoring tools to provide real-time insights during load tests.

**7. Risks and Mitigation**

* Risk: The load test might cause unintentional downtime if run on a production-like environment.
  + Mitigation: Ensure that load tests are run in a fully isolated staging environment with similar configurations to production.
* Risk: Inaccurate load simulation due to test environment differences from production.
  + Mitigation: Regularly sync the staging environment with production configurations and data to ensure realistic load testing.

**8. Maintenance**

* Test Script Maintenance:
  + Regularly update load test scripts to reflect changes in the API or client application workflows.
* Continuous Improvement:
  + Analyze load test results to identify areas for improvement and update the load testing plan accordingly.

**9. Approval**

* Prepared by: [Your Name]
* Approved by: [Project Manager/Lead QA]