**Project Development Phase**

**Model Performance Test**

|  |  |
| --- | --- |
| **Date** | **21 February 2025** |
| **Team ID** | **SWTID1743254297** |
| **Project Name** | **TUNETRAIL- YOUR PATHWAY TO PERFECT PLAYLIST** |
| **Maximum Marks** | **4** |

**6. Performance Testing**

**6.1 Objectives, Types, Metrics, Tools, Scenarios, Optimization & Reporting**

Performance testing ensures that the application is fast, reliable, scalable, and can handle various real-world conditions across frontend and backend systems. The key objectives and elements of performance testing are detailed below:

**Objectives of Performance Testing**

* Ensure Fast and Reliable Streaming  
  Test stream start times (≤ 2s), buffering frequency (≤ 1 per 10 min), and playback stutter.
* Handle Concurrent Users and Actions  
  Simulate scenarios like:
  + 1000 users logging in simultaneously
  + 500 users searching
  + 800 users streaming
  + 200 creating playlists
  + 1000 using recommendations
* Backend/API Performance
  + API response time ≤ 200 ms
  + Use tools like JMeter, k6, and Postman to measure and optimize endpoints.
* Database Performance
  + Query time ≤ 300 ms under heavy load.
  + Optimize with indexing, caching (Redis), and query tuning.
* Frontend and Mobile Responsiveness
  + Page load time ≤ 1 second
  + Test across devices and simulate 3G/4G/5G using Chrome DevTools, Lighthouse, etc.
* Resource Usage and Stability
  + CPU/Memory ≤ 80% under peak load
  + Uptime ≥ 99.9%
  + Monitor using Prometheus, Grafana, AWS CloudWatch.
* Scalability and Fault Tolerance
  + Test system scaling with users and content growth.
  + Use Kubernetes, Docker, PM2 for load balancing and scaling.

**Types of Performance Testing**

| Type | Description | Example |
| --- | --- | --- |
| Load Testing | Normal load capacity | 500 users streaming simultaneously |
| Stress Testing | Beyond normal limits | 10,000 users or mass uploads |
| Soak Testing | Long-duration performance | 100 users streaming for 12 hours |
| Spike Testing | Sudden increase/decrease in load | Viral content spike |
| Scalability Testing | System's ability to grow with increased resources | Adding DB nodes and servers |
| Volume Testing | Large data processing | Millions of tracks/playlists |
| Latency Testing | Delay between request and response | Stream start delay |
| Concurrent Testing | Diverse simultaneous user actions | Streaming, searching, uploading together |

**Key Performance Metrics**

| Metric | Ideal Target |
| --- | --- |
| API Response Time | ≤ 200 ms |
| Stream Start Time | ≤ 2 seconds |
| Page Load Time | ≤ 1 second |
| Uptime | ≥ 99.9% |
| Concurrent Users Supported | ≥ 1000 |
| Buffering Frequency | ≤ 1 per 10 min |
| DB Query Time | ≤ 300 ms |
| CPU/Memory Usage | ≤ 80% |
| Error Rate | < 1% |

**Performance Testing Tools**

| Area | Tools |
| --- | --- |
| Backend Load Testing | JMeter, k6, Artillery |
| API Performance | Postman, Insomnia, k6 |
| Frontend Performance | Google Lighthouse, Chrome DevTools, WebPageTest |
| Monitoring/Logging | Prometheus, Grafana, New Relic, ELK Stack |
| Cloud Performance | AWS CloudWatch, Azure Monitor |
| CI/CD Integration | Jenkins, GitHub Actions with testing plugins |

**Test Scenarios**

* API Load Scenarios:  
  1000 users login, 500 search concurrently, 800 stream simultaneously.
* Streaming Scenarios:  
  Measure stream start delay, playback for 10+ minutes, actions like pause/resume/seek.
* Upload/Admin Scenarios:  
  Artists uploading 10+ tracks, loading analytics dashboard with heavy data.
* Playlist/User Behavior:  
  200 users saving playlists, 1000 accessing recommended songs.
* Mobile/Device Testing:  
  Varying screen resolutions and network throttling for mobile performance.

**Test Reporting and Analysis**

* Monitor trends in response times (min, max, avg)
* Analyze error types (timeouts, 500s)
* Track CPU/memory/network graphs
* Simulate user behaviors and assess outcomes
* Pinpoint bottlenecks in APIs, DB queries
* Provide actionable recommendations for improvements