# **Functional & Performance Testing Template**

**Model Performance Test**

|  |  |
| --- | --- |
| Date | **27 June 2025** |
| Team ID | **LTVIP2025TMID31892** |
| Project Name | **Citizen AI – Intelligent Citizen Engagement Platform** |
| Maximum Marks |  |

## **Test Scenarios & Results**

**🔷 1. Objective of Performance Testing**

To ensure that **Citizen AI** can handle a large volume of user interactions and backend processes **efficiently, reliably, and securely** across different environments and usage conditions.

**🔷 2. Scope of Testing**

| **Component** | **Performance Focus** |
| --- | --- |
| Web/Mobile App Frontend | Load time, responsiveness, multi-user access |
| API Gateway | Throughput, error rate, latency |
| AI/NLP Engine | Query processing time, memory usage |
| Ticket System & DB | Read/write speed, concurrent transactions |
| Notification Engine (SMS/Email) | Delivery latency, queue handling |

**🔷 3. Types of Performance Testing Applied**

| **Type** | **Description** |
| --- | --- |
| **Load Testing** | Test system behavior under expected user loads (e.g., 10,000 concurrent users) |
| **Stress Testing** | Evaluate system behavior under peak loads and beyond capacity |
| **Spike Testing** | Introduce sudden load increases to check stability |
| **Endurance Testing** | Run system over extended periods to find memory leaks and slowdowns |
| **Scalability Testing** | Assess system performance with increasing scale of users or data |

**🔷 4. Performance KPIs (Key Metrics)**

| **KPI** | **Target Value** |
| --- | --- |
| Response Time (API) | < 500ms (average), < 1000ms (peak) |
| Max Concurrent Users | 50,000 (web/mobile), scalable |
| Chatbot Query Handling | 100 queries/sec with < 1s response time |
| Voice-to-Text Latency | < 2 seconds |
| Ticket Processing Time | < 2 seconds for write, < 1 second for read |
| DB Query Throughput | > 1000 read/writes per second |
| Error Rate | < 1% under normal load |
| System Uptime | 99.95% |

**🔷 5. Tools & Technologies Used**

| **Tool** | **Purpose** |
| --- | --- |
| **Apache JMeter** | Load testing for API and web frontend |
| **Locust** | Python-based user load simulation |
| **K6** | Lightweight load testing via JavaScript |
| **Gatling** | Scenario-based stress testing |
| **New Relic / Datadog** | Performance monitoring and metrics |
| **Postman** | API functional and performance tests |
| **Lighthouse** | Frontend performance benchmarking |

**🔷 6. Sample Test Scenarios**

| **Scenario** | **Test Type** | **Expected Outcome** |
| --- | --- | --- |
| 10,000 users log in simultaneously | Load Test | Avg. login time < 1s |
| 5000 chatbot queries in 5 minutes | Spike Test | 0% crash rate, AI response time < 1.5s |
| 24hr system uptime with 10 requests/sec | Endurance | No memory leak, CPU < 70%, latency stable |
| Voice queries during low bandwidth (2G) | Stress Test | Convert input with < 3s delay |
| Notification system with 10k SMS per minute | Load Test | 100% delivery success, < 5% delay |
| 1 lakh support tickets submitted | Scalability | DB read/write speed maintained |

**🔷 7. Sample JMeter Load Test Plan (API)**

* **Test Target:** POST /api/ticket/submit
* **Threads (Users):** 5000
* **Ramp-Up Period:** 300 seconds
* **Loop Count:** 10
* **Assertions:**
  + Response Code = 200
  + Response Time < 1s
  + Error % < 0.5%

**🔷 8. Performance Bottlenecks Identified (Sample)**

| **Issue** | **Root Cause** | **Solution** |
| --- | --- | --- |
| Latency spike in chatbot response | Model cold start delay | Use warm-start and batching optimization |
| DB slow writes under load | Locking on ticket insert | Introduce write-queue and async DB buffer |
| SMS delay on peak | API rate limits breached | Queue service (Kafka/RabbitMQ) added |

**🔷 9. Performance Optimization Techniques Applied**

* ✅ Implemented **caching** (Redis) for frequent queries
* ✅ Switched to **asynchronous APIs** for non-critical paths
* ✅ Scaled horizontally using **Kubernetes** + **Load Balancer**
* ✅ Enabled **lazy loading** and **image optimization** on frontend
* ✅ Reduced model response latency using **ONNX Runtime**