Performance Testing

1. Load Testing
   - Measures system performance under expected user load.
   - Example: Testing when 100 donors and 50 volunteers use the app simultaneously.
   - Expected Result: Response time < 3 seconds per request.
2. Stress Testing
   - Determines the system's breaking point by gradually increasing the load.
   - Ensures the system remains stable even when the number of concurrent users exceeds capacity.
   - Expected Result: Graceful failure without data loss.
3. Scalability Testing
   - Assesses how the system handles increasing workloads (e.g., adding more donors or NGOs).
   - Ensures the app can scale horizontally using cloud resources.
4. Endurance (Soak) Testing
   - Runs the application over a long duration to detect memory leaks or performance degradation.
   - Example: Continuous data input and tracking over 48 hours.
5. Spike Testing
   - Observes system behavior when the load suddenly increases (e.g., festival season donation spikes).
   - Expected Result: No system crash or major performance drop.

3. Performance Metrics

Parameter	Expected Standard
Response Time	≤ 3 seconds
Throughput	≥ 100 transactions/ minute
CPU Utilization	≤ 75% under load
Memory Usage	≤ 70% during peak
System Uptime	≥ 99%

4. Tools Used

- Apache JMeter – for load and stress testing
- Postman / Newman – for API performance validation
- Google Lighthouse – for frontend performance metrics
- Firebase Performance Monitor (if using Android app) – for mobile analytics

5. Test Environment

- Frontend: HTML / ReactJS / Android
- Backend: Java / Node.js / Django
- Database: MySQL / Firebase
- Network: 4G Internet or Wi-Fi (5 Mbps or higher)
- Hardware: 8 GB RAM, Intel i5 Processor

6. Test Results Summary

Scenario	Users Simulated	Average Response (sec)	Status
Normal Load	50 concurrent users	1.8	Passed
Peak Load	150 concurrent users	2.9	Passed
Stress (Crash Test)	300 concurrent users	4.8	Minor Lag
Endurance (24 hrs)	75 active users	2.3	Passed