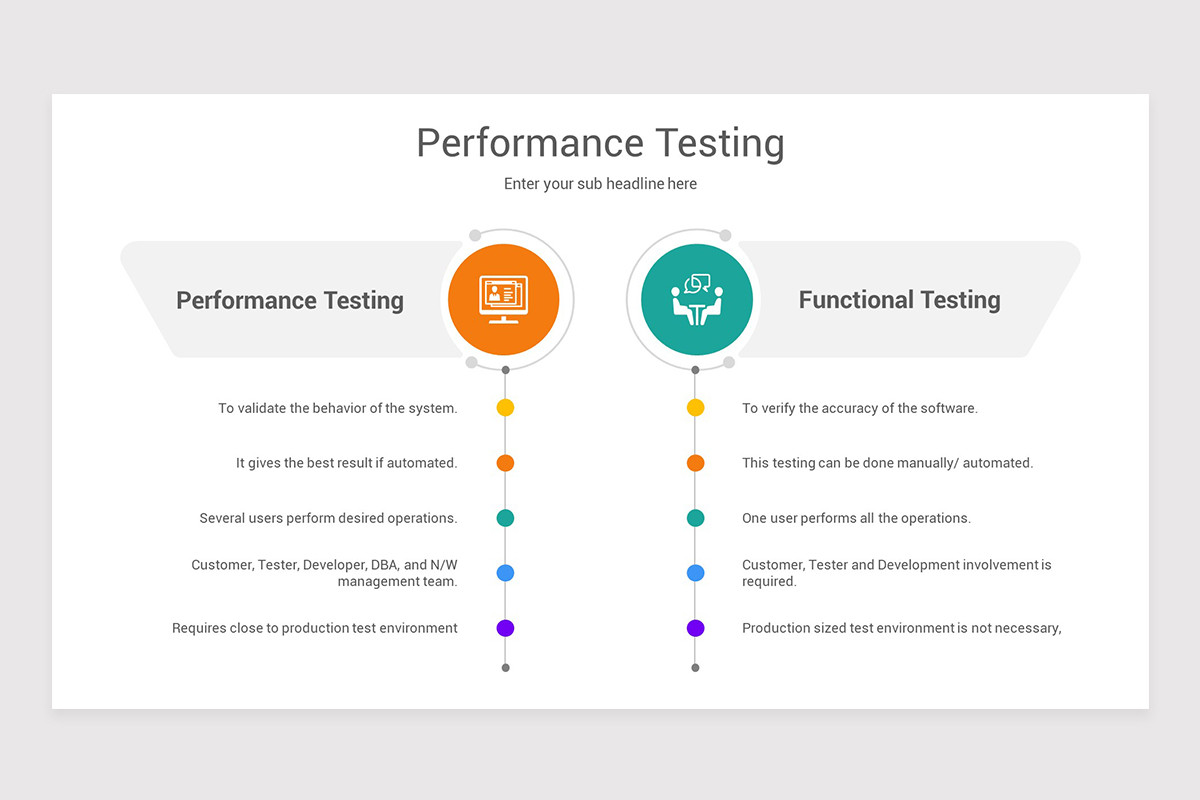
**Functional and Performace Testing**

**Project Overview**

| **Name** | : B Naga Pavan |
| --- | --- |
| **Project Name** | : Food Choices Case Study |
| **Software used** | : Tableau Public Desktop |
| **Prepared By** | : Team Leader |
| **Date** | : 27-6-2025 |
|  |  |

****

**Objective:**

The goal of performance testing is to ensure the **Food Choices application** performs reliably and efficiently under expected and peak user loads. The application should deliver fast, consistent responses while handling multiple user requests for food recommendations, diet plans, and nutritional comparisons.

**🔸 Why Performance Testing is Important:**

1. **User Experience:** Delays in generating personalized food suggestions or loading nutritional charts can frustrate users and lead to app abandonment.
2. **Data Handling:** The app often processes large food databases, filters based on dietary needs (e.g., vegan, diabetic), and visualizes nutrition data—these operations must remain smooth.
3. **Scalability:** As the user base grows, the app should scale without degradation in performance.
4. **Accuracy Under Load:** Even under high traffic, the food suggestion logic must return correct and complete data.

**🔸 Key Performance Testing Scenarios:**

| **Scenario** | **Description** |
| --- | --- |
| **Load Testing** | Test the app with expected number of concurrent users (e.g., 500 users browsing diet plans simultaneously). |
| **Stress Testing** | Push the system beyond its capacity (e.g., 2000 users at once) to observe breaking points. |
| **Spike Testing** | Simulate sudden increases in traffic (e.g., after a health campaign goes viral). |
| **Endurance Testing** | Run the system under high load for extended periods to detect memory leaks or degradation. |
| **Database Performance** | Assess the speed of food item retrieval, nutrient filtering, and recommendation engine under load. |

**🔸 Tools to Use:**

* **JMeter** – for simulating user traffic and HTTP requests.
* **LoadRunner** – enterprise-level performance testing for load/stress.
* **Gatling** – lightweight and scriptable for API-heavy backends.
* **Google Lighthouse / Chrome DevTools** – for frontend speed and performance audit.
* **New Relic / Grafana** – for monitoring server performance during tests.

**🔸 Sample Metrics to Track:**

| **Metric** | **Target** |
| --- | --- |
| Average Response Time | < 2 seconds |
| Peak Response Time | < 5 seconds |
| Throughput | 1000+ requests per minute |
| Error Rate | < 1% |
| CPU/Memory Usage | Within 70% utilization |

**🔸 Performance Bottlenecks to Watch Out For:**

* Slow response time during food plan filtering or sorting.
* High server CPU usage during meal plan generation.
* Database lag when querying large food datasets.
* Latency in loading visual dashboards (e.g., charts of vitamins/minerals).

**🔸 Outcome:**

The performance testing ensures that the Food Choices app is:

* **Scalable** for thousands of users,
* **Responsive** even with complex filters and charts,
* **Reliable** under sustained usage,
* **Robust** to handle sudden demand surges.