**ENSEK Test Strategy**

**Objective:**

To ensure the functionality, security, performance, and reliability of the application by systematically validating its API, UI, integrations, and security mechanisms. Consider using Shift-Left Testing Philosophy to introduce testing earlier in development to catch issues sooner, reducing late-stage defect detection costs.

**Plan and Gather Requirements:**

What we need to achieve:

* Define Testing Objectives:
  + Clearly outline the goals and scope of testing, including the types of testing to be performed (e.g., functional, performance, security).
* Identify Test Environments:
  + Determine the specific test environments needed, considering factors like hardware, software, and network requirements.
* Gather Test Data:
  + Prepare necessary data for testing, ensuring it is representative of production data while maintaining security.

**Testing Approach:**

Employ a multi-layered testing approach, covering:

* API Testing (Postman, automated workflows, API contract validation)
* UI Testing (cross-browser validation, automated user flows, device validation)
* Integration Testing (data consistency, dependencies)
* Security Testing (unauthorized access prevention, vulnerability checks)
* Performance Testing (load tests, response time checks)

Each test type will validate critical components to ensure a seamless user experience while identifying weaknesses before production deployment.

**Environment Setup and Configuration:**

Set up Test Servers:

* Provision dedicated test servers with the required operating systems and applications.

Configure Networks:

* Establish stable network connections and ensure they minimize interference.

Set up Workstations:

* Prepare workstations for testers with the necessary configurations and tools.

Integration Tools:

* Integrate bug reporting tools, automation testing tools, and other necessary utilities.

**Types of Tests to be performed:**

Functional Testing:

* API response validation: Ensure all endpoints return the expected data.
* Authentication & authorization checks: Verify secure login, token validation, and session management.
* Data processing tests: Validate CRUD operations, data consistency, and correct error handling.

Integration Testing:

* Multi-step flows: Validate API chaining (e.g., login → purchase → order retrieval).
* Database interaction: Ensure correct data insertion, update, and retrieval across services.
* External system dependencies: Verify interactions with third-party services (e.g., payment gateways).

UI Testing:

* Cross-browser and device compatibility: Validate layouts on Chrome, Firefox, Edge, and Safari. Validate devices e.g. Laptop, TV, Phone and Tablets.
* Form validation: Check required fields, error messages, and submission handling.
* Dynamic content testing: Ensure data displayed in UI matches the backend.

Security Testing:

* Access control validation: Attempt unauthorized API access to confirm proper restrictions.
* Brute-force & rate-limiting tests: Simulate login attempts and excessive API calls.
* SQL Injection & XSS protection: Attempt malicious inputs to ensure sanitization.

Performance Testing:

* Response time benchmarks: Measure API latency under different loads.
* Load testing: Use JMeter to simulate high-traffic scenarios.
* Scaling validation: Verify system behavior under stress conditions.

Exploratory Testing:

* Manual testing sessions alongside automation can uncover hidden UI/UX flaws.

**Test Execution Plan:**

API Testing using Postman

* Automate test cases using Postman Collections to verify:
  + API contract testing
  + Successful login (200)
  + Unauthorized access (401)
  + Data validation (400/422)
  + Purchase verification (200 + correct order updates)
  + Order deletion & confirmation (200 with proper removal)

Automated UI Testing

* Use Selenium WebDriver or Playwright to:
  + Automate form interactions (login, purchase).
  + Simulate user journeys (checkout flows, dashboard navigation).
  + Detect broken elements & validation errors.

Continuous Testing in CI/CD

* Integrate testing in Azure DevOps:
  + Automated API checks after each commit.
  + Regression tests before production deployment.
  + Test coverage reports for continuous improvement.

**Test Reporting & Debugging:**

Reporting Tools

* Azure Dev Ops Bug Tracking, Run Postman and Playwright test CI/CD pipelines
* Allure Reports: Generate detailed, graphical test summaries.

Debugging Methods

* Log analysis: Identify 500 errors with trace details.
* Network monitoring: Validate API requests & responses.
* Real-time alerting: Set up failure notifications for critical flows.

**Key Considerations:**

Risk Analysis:

* Identify potential risks associated with testing and allocate testing efforts accordingly.

Automation:

* Automate repetitive tasks and regression testing to improve efficiency.

Collaboration:

* Promote communication between different teams involved in the development and testing process.

Resource Management:

* Optimize resource usage by reusing test environments and resources when possible.

Environment Separation:

* Keep test environments isolated to prevent interference and ensure consistent results.

Integration with CI/CD:

* Leverage the automation with continuous integration and continuous deployment pipelines.

Testing Phase Review:

* Periodically reviewing test coverage to ensure the most critical parts of the system are effectively being covered. For example coverage reports in Azure DevOps can help pinpoint gaps in testing efforts.