**Low‑Level Technical Specification: Testing**

Defines testing strategy, environments, tools, frameworks, data management, CI integration, reporting, and governance.

## **1. Overview**

Ensure reliability, correctness, performance, and security of all services (frontend, backend, AI, database) through automated and manual testing.

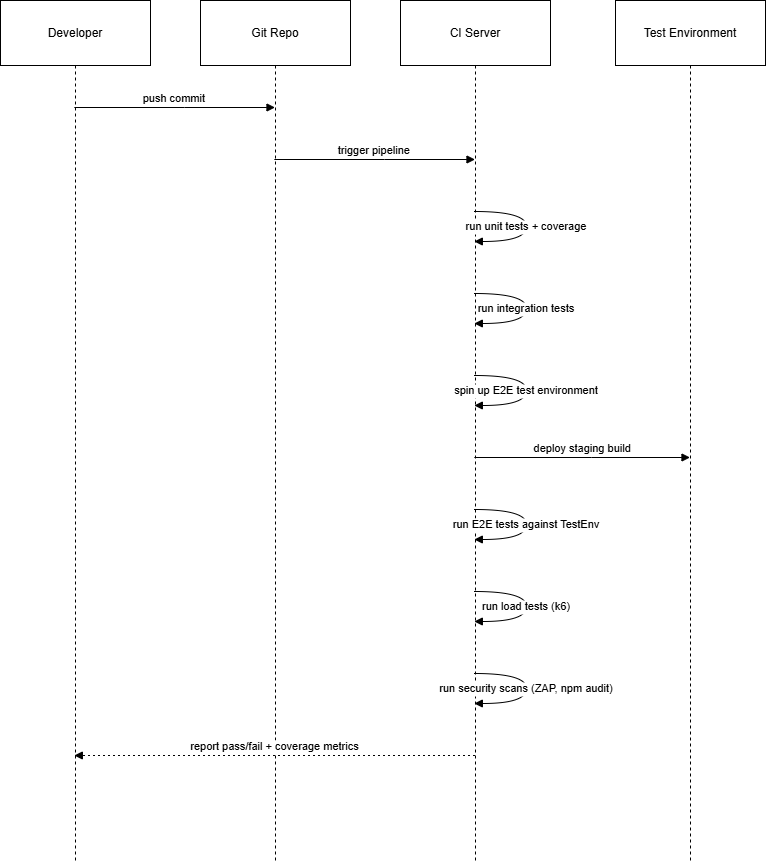
## **2. Test Types & Frameworks**

1. **Unit Testing**
   * **Scope**: individual functions, components, utility modules
   * **Frameworks**: Jest (frontend & backend), pytest (AI service, Python)
   * **Mocking**: jest-mock, pytest-mock, nock for HTTP interactions
   * **Coverage**: ≥ 80% threshold per service; enforce in CI
2. **Integration Testing**
   * **Scope**: service interactions (e.g., API ↔ Database, API ↔ AI service, WebSocket flows)
   * **Frameworks**: Supertest (Node.js API), pytest + httpx for AI, socket.io-client for real-time
   * **Test DB**: pg-mem (in-memory Postgres) or local PostgreSQL container
   * **Test Cache**: redis-mock or real Redis container in CI
3. **End‑to‑End (E2E) Testing**
   * **Scope**: full user flows across frontend & backend
   * **Frameworks**: Cypress (web), Detox or Appium (mobile)
   * **Scenarios**: login, session creation, real-time voting, async voting, AI suggestions display, error flows
4. **Performance & Load Testing**
   * **Scope**: API latency, WebSocket concurrency, AI service throughput
   * **Tool**: k6 (load scripts)
   * **Targets**:  
     + API: 1,000 RPS with <200 ms 95th percentile latency
     + WebSocket: 500 concurrent connections broadcasting votes within 100 ms
     + AI: 50 concurrent inference calls with <300 ms 95th percentile latency
5. **Security Testing**
   * **Scope**: OWASP Top 10, JWT handling, XSS/CSRF, dependency vulnerabilities
   * **Tools**: OWASP ZAP (automated scan), npm audit/snyk, manual pen test scripts, dependency-check plugins
6. **Accessibility Testing**
   * **Scope**: WCAG 2.1 AA for web UI
   * **Tools**: axe-core (Cypress plugin), manual keyboard + screen reader checks

## **3. Test Environments & Data Management**

* **Isolated Environments**:  
  + **Unit/Integration**: local dev, CI containers
  + **E2E**: dedicated test namespace in staging cluster
* **Test Data**:  
  + Seed scripts for Postgres (db/fixtures/\*.sql)
  + Redis preload via scripts (session keys)
  + Sample AI stories & expected suggestions in JSON fixtures
* **Data Reset**:  
  + Fresh database migrations + seed before each test suite run
  + Redis FLUSHALL between integration tests

## **4. CI Integration & Workflow**



* **Steps**:  
  + **Lint & Static Analysis** (ESLint, Flake8, bandit)
  + **Unit Tests**
  + **Integration Tests**
  + **Build & Deploy to TestEnv**
  + **E2E Tests**
  + **Performance Tests**
  + **Security Scans**
* **Gates**:  
  + Block merge on failure of any tests or coverage < 80%

## **5. Reporting & Metrics**

* **Test Reports**:  
  + JUnit XML for all suites (consumed by CI)
  + HTML coverage reports (Jest, pytest) stored as artifacts
* **Dashboards**:  
  + Expose k6 performance metrics to Grafana
  + OWASP ZAP results via Jenkins plugin or HTML report
* **Alerts**:  
  + CI failure notifications via Slack/email
  + Performance thresholds breached trigger alerts

## **6. Test Governance & Maintenance**

* **Test Ownership**:  
  + Dev teams own unit/integration tests
  + QA engineers own E2E and security tests
* **Review Process**:  
  + Peer review for new tests
  + Quarterly audit of test coverage and flaky tests
* **Flaky Test Handling**:  
  + Mark as unstable and triage within 1 sprint
  + Quarantine consistently flaky tests

*End of Testing Specification*