TestAura

From Code to Confidence — AI-Powered Test Generation

Software Requirements Specification (SRS) Document

Project Name: TestAura
Version: 1.0
Author: Yash Sudhirrao Desai
Date: May 12, 2025

---

1. Introduction

1.1 Purpose

TestAura is a smart test case generator that automates unit and integration test creation for APIs and Java code using AI or static logic. The purpose of this document is to define the functional and non-functional requirements, system scope, and high-level design.

1.2 Scope

TestAura provides:

- Upload functionality for OpenAPI/Swagger specs or Java code
- AI or regex-based test case generation
- Manual test management
- Test execution via CI/CD
- Allure-based report generation
- Role-based access system
- Docker/Kubernetes deployability

This document covers the complete system specifications from a user and technical perspective.

1.3 Definitions, Acronyms, Abbreviations

Term	Definition
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment
AI	Artificial Intelligence
SRS	Software Requirements Specification

2. Overall Description

2.1 Product Perspective

TestAura is a standalone full-stack web application integrated with CI/CD tools and optionally AI APIs (OpenAI). It supports modular microservice-like architecture.

2.2 Product Features

- User authentication (JWT)
- Upload Swagger/Java files
- Smart test case generation
- Manual test case management
- CI/CD test execution pipeline
- HTML report visualization

2.3 User Classes and Characteristics

- **Tester**: Can generate, run, and view test cases.
- **Developer**: Uploads code/specs and integrates test logic.
- **Admin**: Manages users, settings, system logs.

2.4 Operating Environment

- Backend: Java 17+, Spring Boot

- Frontend: React + TailwindCSS
- Database: MySQL
- DevOps: Docker, GitHub Actions, Allure, Kubernetes

2.5 Constraints

- OpenAI API key required for AI features
- MySQL must be configured correctly
- Network dependency for external API usage

2.6 Assumptions and Dependencies

- Swagger files must be in OpenAPI 3.0+ format
- Java code should follow standard conventions
- Internet access required for AI-powered generation

3. Functional Requirements

3.1 User Authentication

- Register, login, JWT-based session
- Role-based access control

3.2 File Upload

- Accept .json (OpenAPI) or .java files
- Validate file format
- Store metadata in DB

3.3 Test Generation

- Parse input files
  Use OpenAI API or regex logic to generate tests
- Store test cases linked to uploaded file

3.4 Test Management

- List generated test cases

- Allow manual addition or editing
- Soft-delete support

3.5 Test Execution

- Trigger from UI or commit hook
- Execute tests via CI/CD
- Generate and store Allure reports

3.6 Reporting

- View HTML reports
- Summary stats (pass %, fail %, test counts)

4. Non-Functional Requirements

4.1 Performance

- Response time < 2 seconds for standard operations
- Handle 50 concurrent users

4.2 Scalability

- Horizontal scaling support via Docker/K8s

4.3 Security

- JWT auth
- HTTPS only
- Encrypted API keys

4.4 Usability

- Minimalist, accessible UI
- Light/dark theme toggle

4.5 Maintainability

- Modular services with clear separation of concerns

- Well-documented APIs

4.6 Availability

- 99.5% uptime target
- Self-healing via container orchestration

5. System Models

(Optional but recommended for inclusion in system design)

- Use case diagram
- ER diagram
- Sequence diagrams

Would you like this turned into a properly formatted .md, .docx, or .pdf file next? Or want to revise any section first?