PROJECT DEFINITION DOCUMENT (PDD)

Project Name: SnippetVault (Internal: Project Jacked) **Version:** 2.0 (Final Draft) **Date:** February 9, 2026 **Document Owner:** Engineering Lead

---

1. EXECUTIVE SUMMARY

SnippetVault is a cloud-native, high-performance knowledge management system designed to streamline the storage and retrieval of code snippets for software engineers.

Unlike traditional note-taking applications, SnippetVault is engineered with an "Infrastructure-First" mindset. It prioritizes data persistence, high availability, and automated delivery pipelines. The project serves a dual purpose: providing a utility for developer productivity and acting as a proof-of-concept for enterprise-grade architecture using open-source technologies.

---

2. PROJECT SCOPE

2.1 In-Scope

- **User Authentication:** Secure login via third-party providers (GitHub).
- **Core Core:** Full Create, Read, Update, Delete (CRUD) capabilities for code snippets.
- **Search Engine:** Real-time filtering of data based on language and tags.
- **Performance Layer:** Implementation of Redis for caching frequently accessed data.
- **Hosting:** Deployment of the full stack on a Virtual Private Server (VPS).
- **Automation:** A fully automated CI/CD pipeline for testing and deployment.

2.2 Out-of-Scope

- **Native Mobile Apps:** (iOS/Android) – Web interface only.
- **Social Features:** (Sharing, Comments, Likes) – Single-player focus.
- **Paid Subscriptions:** The system is free-to-use/self-hosted.

---

3. SYSTEM ARCHITECTURE

The solution utilizes a **Headless Architecture**, decoupling the user interface from the logic and data layers to ensure scalability.

3.1 The Frontend (Client Layer)

- **Technology:** Next.js (React Framework).
- **Role:** Handles all user interactions, rendering, and state management. It communicates with the backend solely via RESTful API calls.
- **Key Feature:** Client-side routing for a "Single Page Application" feel.

3.2 The Backend (Logic Layer)

- **Technology:** Laravel (PHP Framework).
- **Role:** Acts as the API Gateway. It validates incoming requests, enforces security policies, and orchestrates data movement between the database and cache.
- **Key Feature:** Stateless authentication using API Tokens (Sanctum).

3.3 The Data Layer

- **Primary Database:** MySQL.
    - **Purpose:** Permanent storage of user profiles, snippet content, and metadata.
- **High-Speed Cache:** Redis.
    - **Purpose:** In-memory storage for user sessions, API rate limiting counters, and queued background jobs.

---

4. INFRASTRUCTURE & HOSTING STRATEGY

The project requires a production environment that mimics enterprise constraints (Linux, Firewalls, Containerization) rather than managed "Serverless" solutions.

4.1 Cloud Provider

- **Platform:** Oracle Cloud Infrastructure (OCI).
- **Tier:** "Always Free" Compute Instance.
- **Justification:** Provides raw access to a Virtual Machine (VM) with substantial RAM (24GB), allowing for the concurrent execution of multiple Docker containers (App, DB, Cache) without performance degradation.

4.2 Server Configuration

- **Operating System:** Ubuntu Linux 22.04 LTS (ARM64 Architecture).
- **Containerization:** Docker & Docker Compose.

    - **Strategy:** The entire application stack is containerized. This ensures the "Dev" environment on the local machine is identical to the "Prod" environment on the cloud.

4.3 Networking & Security

- **Web Server:** Nginx (running as a Reverse Proxy container).
- **SSL/TLS:** Automated certificate management via Let's Encrypt to ensure HTTPS encryption (Port 443).
- **Firewall:** configured to block all incoming traffic except for HTTP (80), HTTPS (443), and SSH (22).
- **Database Isolation:** The MySQL and Redis databases are inaccessible from the public internet; they communicate only within the internal Docker network.

---

5. FUNCTIONAL REQUIREMENTS

5.1 Authentication Module

- **FR-01:** System must allow users to authenticate using OAuth 2.0 (GitHub).
- **FR-02:** System must maintain user sessions across page reloads using Redis-backed session storage.
- **FR-03:** System must auto-create a user profile upon first login using data provided by the OAuth provider (Username, Avatar).

5.2 Snippet Operations

- **FR-04:** Users can create snippets with: Title, Code Body, Programming Language (Syntax), and Tags.
- **FR-05:** Users can edit or delete only snippets they own.
- **FR-06:** The system must validate input to prevent SQL Injection and XSS attacks.

5.3 Performance & Search

- **FR-07:** Search bar must filter results by Title or Tag.
- **FR-08:** Search results for "popular tags" must be cached in Redis for 10 minutes to reduce database load.

---

6. DEVOPS & AUTOMATION (CI/CD)

6.1 Continuous Integration (The "Test" Phase)

- **Trigger:** Any code push to the main branch on GitHub.
- **Action:** GitHub Actions will spin up a temporary environment to:
    1. Run backend unit tests (PHPUnit).
    2. Check code styling standards (Linting).
    3. Verify frontend build compilation.
- **Gatekeeper:** If any step fails, the deployment is blocked.

6.2 Continuous Deployment (The "Release" Phase)

- **Trigger:** Successful completion of the CI phase.
- **Action:** GitHub Actions will securely SSH into the Oracle Cloud server.
- **Execution:**
    1. Pull the latest Docker images.
    2. Gracefully restart the containers.
    3. Run database migrations automatically.

---

7. PROJECT DELIVERABLES

Upon completion, the project will consist of:

1. **Source Code:** A GitHub repository containing the Frontend, Backend, and Infrastructure configuration (`docker-compose.yml`).
2. **Live Application:** A publicly accessible URL (e.g., `https://snippetvault.app`) secured with SSL.
3. **Documentation:** A technical `README` detailing local setup and deployment commands.

End of Document