Vulnerable BookShelf-AI Comprehensive System Documentation

Executive Summary

Vulnerable BookShelf-AI is a full-stack, microservices-based platform that integrates Large Language Models (LLMs), RAG (Retrieval-Augmented Generation), document vectorization, natural-language intent detection, and traditional database operations. The system is intentionally designed with multiple controlled vulnerabilities to support experimentation, learning, penetration testing, and demonstration of modern LLM-security concerns such as prompt injection and LLM-generated SQL injection.

The platform allows users to:

- Sign up, log in, and perform natural-language queries
- Search books using a hybrid SQL + Vector database approach
- Upload PDF documents and automatically index them into a FAISS vector store
- Retrieve book summaries using RAG
- Manage books and users via an Admin Panel
- Observe realistic vulnerabilities in authentication, injection, query generation, and system architecture

This document provides a complete overview of the system, its internal architecture, implementation details, deployment instructions, and operational workflow using Docker.

Project Overview

Objectives

- Build an AI-augmented book search platform.
- Integrate LLMs for SQL generation and natural-language answering.
- Provide RAG-based book retrieval using FAISS.

- Demonstrate real-world security vulnerabilities in a controlled environment.
- Showcase a modern, extensible microservices architecture.

System Components

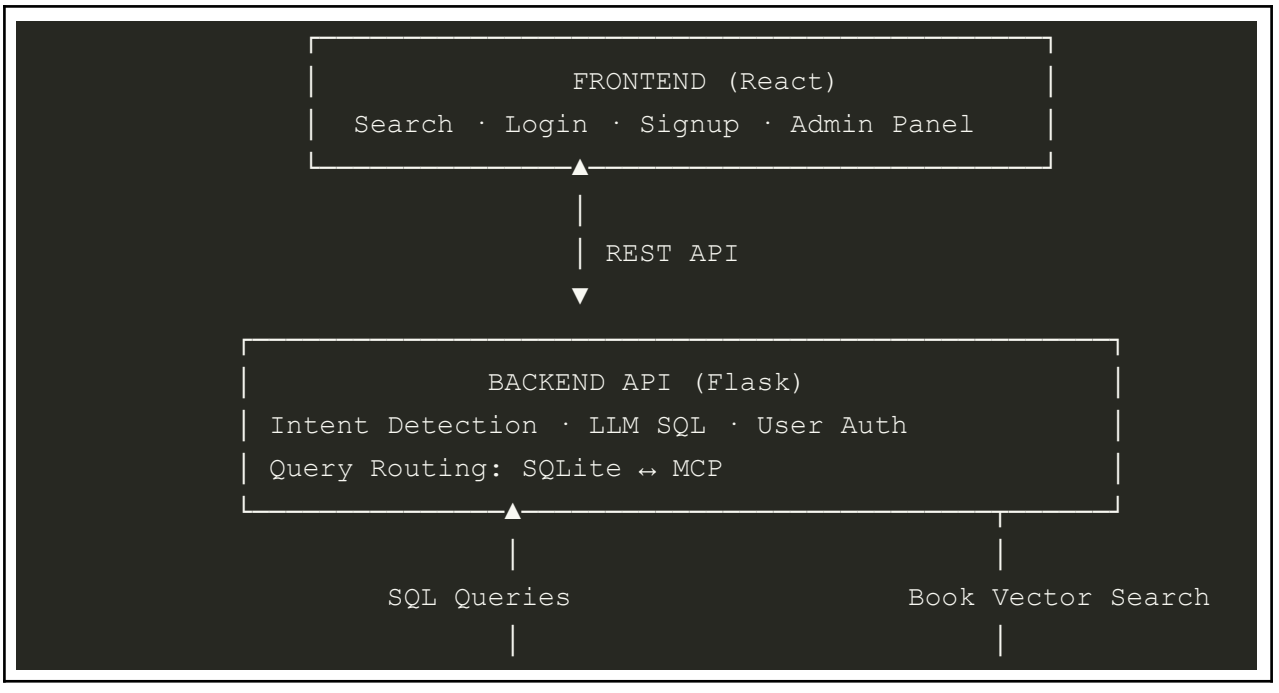
The system consists of four distinct microservices:

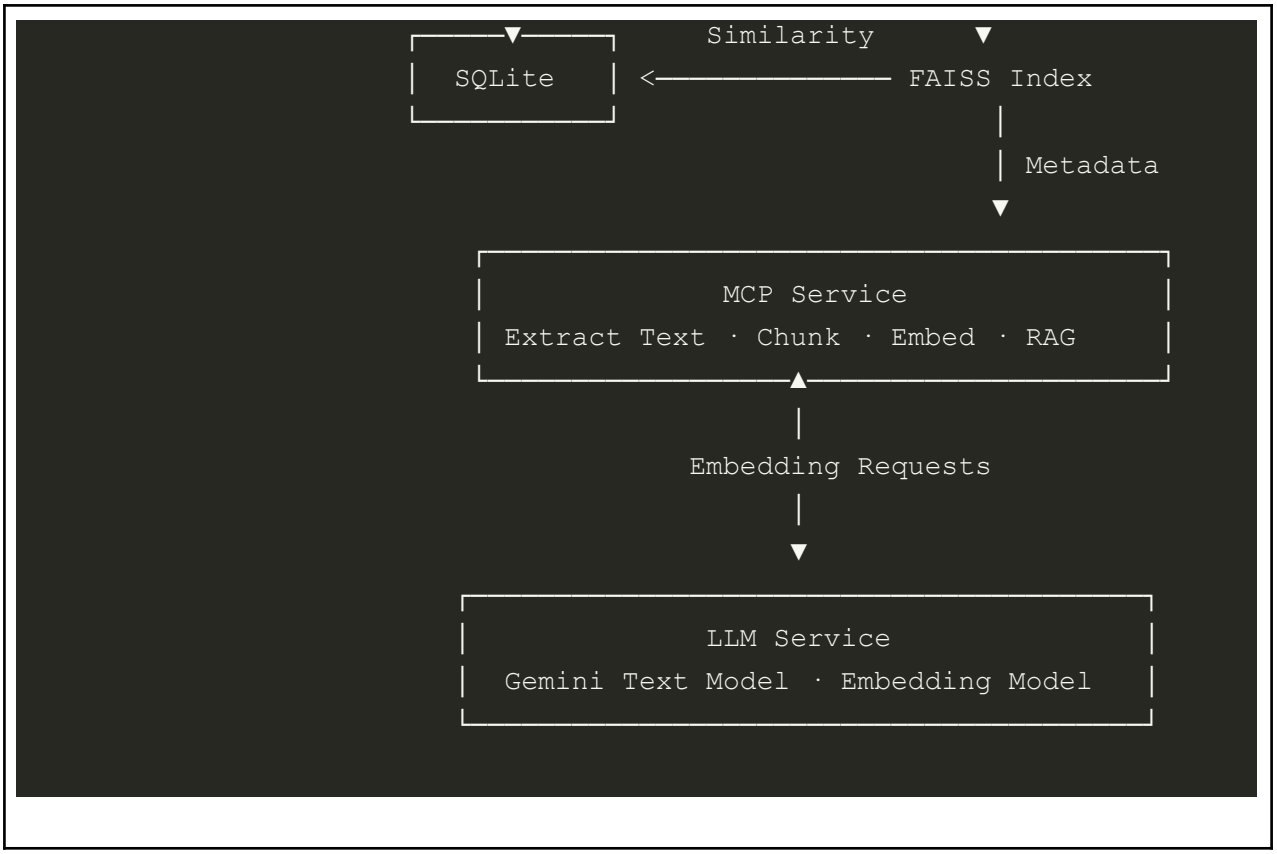
1. **Backend API Service (Flask, SQLite)**
    - Authentication, authorization, SQL routing, LLM SQL generation, intent detection.
2. **MCP Service (Model Context Processor)**
    - PDF ingestion, text extraction, chunking, embeddings, FAISS index, RAG search.
3. **LLM Service (Gemini)**
    - SQL generation, text responses, embedding generation.
4. **Frontend (React + Tailwind)**
    - User interface for search, login, signup, admin control panels.

Each service is independently containerized and orchestrated using Docker Compose.

System Architecture

High Level Design





Module Breakdown

Backend Service

Purpose

The backend acts as the **central orchestrator**, handling:

- User management
- Login and signup
- SQL query execution
- Interfacing with LLM & MCP
- Admin CRUD operations
- Intent classification
- Security-relevant vulnerable behaviors

Key Features

Intent Detection Engine

The backend classifies user queries into categories:

- chitchat
- user_query
- book_query
- list_all_books
- delete_user
- delete_book
- unknown

This ensures that natural language queries trigger appropriate backend flows.

LLM-Generated SQL (Intentionally Vulnerable)

The system prompts Gemini to output SQL queries directly.

Vulnerabilities included intentionally:

- No SQL sanitization
- No validation of table names
- Unrestricted SQL operations (when allowed)
- LLM can be tricked into generating harmful SQL

This forms the core demonstration of LLM-SQL injection.

Book Query Routing

The backend attempts RAG vector search via MCP first.
If MCP fails or returns empty results, the system falls back to SQLite queries.

Admin CRUD Operations

Includes:

- Add book
- Edit book
- Delete book
- Add user
- Edit user
- Delete user

All are intentionally insecure (no role enforcement on backend API).

MCP Service

Purpose

The MCP acts as a dedicated vector database microservice supporting:

- PDF ingestion
- Text extraction
- Chunking
- Embedding generation
- Vector indexing
- RAG search
- Book registry maintenance

PDF Ingestion Workflow

1. Upload PDF
2. Save file to MCP storage
3. Extract text (OCR-compatible extractors)
4. Chunk text with overlap
5. Generate embeddings using Gemini
6. Insert vectors into FAISS
7. Store metadata in metadata.json
8. Expose upload progress via STATUS DB

FAISS Indexing

The FAISS index stores normalized embeddings for cosine similarity search.

Book-Level Registry

Each book contains:

- Title
- Author
- Genre
- Filename
- Associated vector IDs
- Upload timestamp
- Metadata for reconstruction

RAG Search Pipeline

When MCP receives a search query:

1. Embed the user query
2. Search FAISS for nearest vectors
3. Retrieve metadata and snippets
4. Build RAG prompt
5. Send enriched prompt to Gemini
6. Return generated answer

This enables contextual book-aware answers.

LLM Service

Responsibilities

- Convert natural language to SQL
- Summarize query results
- Generate embeddings
- Answer questions using RAG context
- Ensure robustness to various API response formats

Architecture Strengths

- Batch embedding support
- Multi-format parsing
- Abstraction over Gemini SDK differences
- Clear separation of text vs SQL modes

Frontend

Modules & Views

Authentication UI

- Login
- Signup
- Session storage via localStorage

Search UI

- Single text field for all interactions
- Displays RAG answers
- Renders markdown
- Handles deletion intents

Admin Panel

- Book management
- PDF upload with live indexing status
- Vector DB and metadata listing
- User management (add/edit/delete)

Intentional Vulnerabilities

The application is **designed to be vulnerable** for education and research.

LLM-Generated SQL Injection

Gemini generates raw SQL:

- Can be manipulated via prompt injection
- Allows dropping tables, deleting records, modifying schema

Authentication & Authorization Weaknesses

- Passwords stored in plaintext
- No backend enforcement of roles
- Admin panel accessible via frontend-only gating

Input Sanitization Missing

- Search endpoint accepts unfiltered text
- LLM SQL prompt accepts malicious re-instructions
- No rate limiting or throttling

FAISS Metadata Tampering

Metadata is not validated, allowing potential poisoning experiments.

Overall, these vulnerabilities provide a safe environment for demonstrating modern AI-security risks.

Deployment with Docker

This section provides a **detailed, enterprise-level operational guide** for deploying and running the complete system via Docker.

Prerequisites

1. Install Docker Desktop

- Download:
  https://www.docker.com/products/docker-desktop/
- Ensure the Docker Engine is running.

2. Install Git

Clone the repository from the given GitHub Link:
Akshit-1201/vulnerable-book-shelf-ai

```
git clone https://github.com/Akshit-1201/vulnerable-book-shelf-ai
cd vulnerable-book-shelf-ai
```

3. Add environment files

Create the required .env files as indicated in the project structure.

- **.env**:

```
DB_PATH = /app/data/database.db
GEMINI_API_KEY = your_gemini_api_key
```

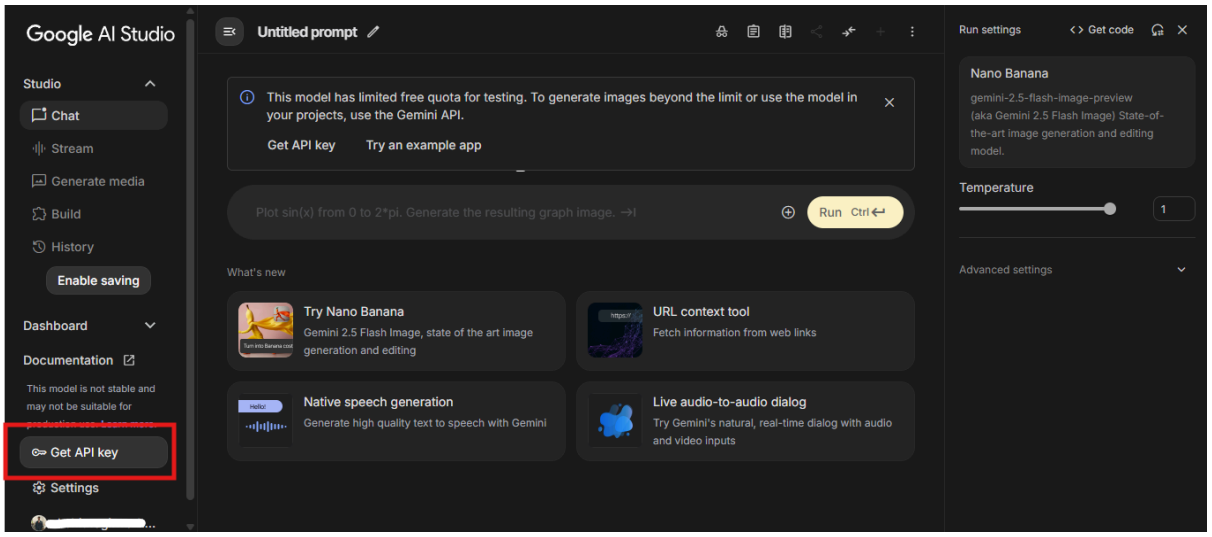
These files ensure sensitive information (like API keys and DB paths) are not hard-coded in the source.

Gemini API Key

Google Gemini APIs are managed through **Google AI Studio (makersuite)** or **Google Cloud Console**.

Steps to get Gemini API Key

- Sign in to Google Account
    - Go to [Google AI Studio](#) (makersuite).
    - Log in with your Google account (Gmail/Workspace).
- **Enable API Access**
    - If it's your first time, you may need to accept **Google Generative AI terms**.
    - You'll see options to create and manage API keys.
- **Create an API Key**
    - Click **Get API Key** or **Create API Key**.
    - Choose **"Create API Key in New Project"** (recommended) or select an existing project.
    - Copy the generated key.



Docker Compose Deployment Guide

Step 1: Navigate to the project directory

```
cd vulnerable-book-shelf-ai
```

Step 2: Build all service containers

```
docker-compose build
```

This builds:

- backend
- llm
- mcp
- frontend

Each service is isolated and built from its respective Dockerfile.

Step 3: Start all services

```
docker-compose up
```

Or run in detached mode:

```
docker-compose up -d
```

Step 4: Validate running containers

```
docker ps
```

Expected ports:

- Frontend → http://localhost:3000
- Backend → http://localhost:8000
- MCP → http://localhost:8001
- LLM → http://localhost:5000

Step 5: Access the application

Open browser:

```
http://localhost:3000
```

Step 6: System Workflow

1. Signup/Login

Users can create an account using basic credentials through the Signup page or authenticate using the Login page.

- **Default Admin Access:**
  The application is preconfigured with default administrator credentials for administrative access:

  - **Email:** admin@example.com
  - **Password:** admin123

  Logging in with these credentials grants access to the **Admin Panel**, where administrative operations such as content management and system-level controls are available.

- **Standard User Access**:
  Users who register via the **Signup** page are assigned standard user privileges. These users can access the core functionality of the application but **do not have access to the Admin Panel**.

This role-based access ensures a clear separation between administrative and regular user functionalities within the system.

2. Search

The application provides a **natural-language search interface** that allows users to interact with the system using conversational queries. The search functionality is powered by an LLM-based interpretation layer, which dynamically translates user queries into relevant database operations, vector searches, or content retrieval actions.

Users can ask natural-language questions:

- “List all books”
- “Delete user John”
- “Who wrote Atomic Habits?”
- “What does chapter three say?”

3. Admin Panel

The **Admin Panel** is accessible only to users with administrative privileges and provides centralized control over system data, users, and indexed content.

Admins can:

- **Upload PDF Documents:** Upload book PDFs for ingestion into the system. Uploaded documents are processed, chunked, and embedded before being stored in the vector database.
- **Track Indexing Progress:** Monitor the status of document ingestion, including chunk generation and embedding progress during the indexing pipeline.
- **View FAISS-Indexed Books:** Access a list of all books currently stored in the FAISS vector database, along with associated metadata.
- **Edit or Delete Users:** Manage registered users by modifying user details or removing accounts from the system.
- **Delete Books from FAISS:** Remove selected books from the vector database, including all associated embeddings and metadata.

Note: When using the free-tier Gemini API key, limitations such as strict rate limits may affect PDF ingestion. Large PDF files may not be fully processed into chunks, and the operation can terminate prematurely due to API throttling, resulting in HTTP 429 errors.

Therefore, uploading **very large PDFs with a high number of pages is not recommended** when operating under the free API tier.

Operational Considerations

Logging

- Backend logs requests and internal actions
- MCP logs PDF ingestion and indexing workflows
- LLM logs embedding/text generation errors

Restarting Services

To rebuild containers:

```
docker-compose down
docker-compose up --build
```

Resetting Vector Database

Delete:

```
data/mcp/*
```

New metadata and FAISS index will regenerate.

Conclusion

Vulnerable BookShelf-AI successfully demonstrates a complete, production-style, AI-augmented platform integrating:

- Natural-language understanding
- LLM-generated SQL
- Embedding-based document retrieval
- FAISS vector indexing
- Multi-microservice architecture
- End-to-end user and admin management

The deliberate inclusion of vulnerabilities makes this project uniquely valuable for:

- Security research and penetration testing
- Academic demonstrations of prompt and SQL injection
- Learning modern full-stack microservice patterns
- Experimenting with RAG pipelines and LLM orchestration

This system provides an excellent foundation for exploring LLM safety, secure software design, and multi-modal AI architectures.
Its clean separation of services, extensible workflow, and real-world vulnerability scenarios make it a powerful educational and research asset.