NeuroNotes: AI-Powered Research Notes with Smart Links

Author : Misbah Juwayriyyah

Buildables Final Project SRS

Table of Contents

NeuroNotes: AI-Powered Research Notes with Smart Links

Django, Django REST Framework (DRF), PostgreSQL, Celery, RAG

1. Preface

- **Expected Readership:**
  This document is intended for project stakeholders, developers, and evaluators.
- **Rationale for Creation:**
  The system is proposed as a student project to demonstrate skills in Django, REST APIs, and applied AI (RAG). It aims to provide a collaborative, intelligent research notes system.
- **Version History:**
    - v1.0 (Initial Draft, October 2025): Created to outline functional and non-functional requirements.

2. Introduction

- **Need for the System:**
  Researchers, students, and professionals struggle with organizing large amounts of notes and papers. Conventional note apps lack **semantic linking** and **AI-assisted exploration**.
- **System Functions (brief):**
    - User account management (multi-user, roles).
    - Notes and document management with tags.
    - Automatic smart linking between related notes.
    - Timeline visualization of note evolution.
    - Optional RAG assistant to query notes with citations.
- **Strategic Objective:**
  Provide a structured yet intelligent research workspace, demonstrating Django expertise and integration of modern AI.

3. Glossary

- **RAG (Retrieval-Augmented Generation):** An AI method where retrieved documents guide language model responses.
- **Smart Link:** A semantic connection between notes detected automatically using keyword/embedding similarity.
- **DRF:** Django REST Framework, used for API development.
- **Embedding:** A numerical vector representing text meaning for similarity comparison.

4. User Requirements Definition

- **Services for the User:**
    - Register/Login with role-based access.
    - Create, edit, delete, and tag notes.
    - Upload PDF research documents.
    - Browse notes by tag, date, or timeline.
    - See smart links between related notes.
    - Collaborate with team members on shared projects.
    - Use AI assistant (optional) for querying notes.
    - Give feedback on AI answers (thumbs up/ thumbs down).
- **Nonfunctional Requirements:**
    - Security: Role-based permissions and authentication.
    - Performance: Notes retrieval <2s for average dataset.
    - Scalability: Support up to 100 concurrent users.
    - Usability: Clean, intuitive UI (could be React/HTML front).

5. System Architecture

- **High-Level Components:**
    - **Frontend:** Minimal web UI or REST API client.
    - **Backend (Django):** Handles user accounts, notes, tags, smart links.
    - **Database (PostgreSQL):** Stores notes, documents, embeddings, and links.
    - **Background Worker (Celery):** Processes smart linking asynchronously.
    - **RAG Component:** Optional embedding model + LLM integration.

6. System Requirements Specification

- **Functional Requirements:**

    - User registration, login, logout.
    - CRUD for notes, tags, documents.
    - Smart linking between notes based on similarity.
    - Timeline view for note history.
    - Sharing & collaboration features.
    - RAG-based assistant with citations.
    - Feedback collection for AI answers.
- **Nonfunctional Requirements:**
    - Cross-platform (browser-based).
    - RESTful API design.
    - Secure password handling (hashing).
    - Documentation of APIs (Swagger/Postman).

7. System Models

- **Use Case Diagram (textual form since no image here):**
  - Actors: User, Collaborator, System, AI Assistant.
  - Use Cases: Login, Manage Notes, View Timeline, Collaborate, Ask Question, Give Feedback.
- **Data Model (Core Entities):**
  - `User(id, name, email, role)`
  - `Note(id, title, content, owner_id, created_at)`
  - `Tag(id, name)`
  - `Document(id, file_path, note_id)`
  - `NoteLink(id, source_note_id, target_note_id, similarity_score)`
  - `Feedback(id, answer_id, user_id, rating)`

8. System Evolution

- **Assumptions:**
  - Initially supports only text + PDFs. Future could extend to images/audio notes.
  - Smart linking starts with embeddings, may later expand into a full knowledge graph.
  - RAG uses existing APIs but can later integrate local LLMs for offline use.

9. Appendices

- **Database:** PostgreSQL with full-text search.
- **Hardware (for demo):** 4GB RAM, dual-core CPU, optional GPU for embeddings.
- **Process Standards:** Follow Django best practices, REST API conventions, and PEP8 coding style.

10. Index

- **Functions Index:**
  - Authentication → Preface, User Requirements
  - Note Management → User Requirements, System Models
  - Smart Linking → System Models, Architecture
  - RAG Assistant → Functional Requirements, Evolution