Loupe Project Documentation

1. Project Overview

Loupe is a Notion-style rich-text editor application that enables users to create, edit, and share pages with **AI-powered fact-checking**.
It supports collaborative document creation with advanced text formatting, drag-and-drop block reordering, and public sharing features.

The project integrates **Google Gemini AI** to provide real-time fact verification of selected text content.

---

2. System Architecture

Loupe follows a **microservices architecture** deployed via Docker containers:

- **Frontend**: React-based single-page application served with Vite
- **Backend**: FastAPI REST API server with SQLAlchemy ORM
- **Database**: PostgreSQL relational database with JSONB support for rich content
- **AI Integration**: Google Gemini API for fact-checking functionality

---

3. Containerization Setup

- **loupe-frontend**: Runs on port **5173** (React + Vite)
- **loupe-backend**: Runs on port **8000** (FastAPI)
- **db**: PostgreSQL 15, runs internally on port **5433**

---

4. Technologies Used

Backend

- FastAPI (Python async web framework)
- SQLAlchemy (ORM)
- Pydantic (data validation)
- Uvicorn (ASGI server)
- httpx (async HTTP client)
- python-dotenv (environment management)
- Alembic (database migrations)
- psycopg2-binary (PostgreSQL adapter)

Frontend

- React 19 (with TypeScript)
- Plate.js (Slate.js-based rich-text editor)
- TailwindCSS (utility-first styling)
- Vite (build tool & dev server)
- React Router (client-side routing)
- Axios (API communication)
- React DnD (drag-and-drop support)

Database

- PostgreSQL 15 with **JSONB** support

AI Integration

- Google Gemini 2.5 Flash (fact-checking API)

---

5. Database Design

Page Model

```
CREATE TABLE pages (
    id SERIAL PRIMARY KEY,
    title VARCHAR,
    content JSONB NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    share_token VARCHAR UNIQUE,
    is_public BOOLEAN DEFAULT FALSE
);
```

Field Details

- **id**: Auto-incrementing primary key
- **title**: Optional page title
- **content**: JSONB storage for Plate.js content
- **created_at**: Timestamp on creation
- **share_token**: Unique token for public sharing
- **is_public**: Boolean flag for access control

---

6. API Endpoints

Page Management

- `POST /pages/` → Create a new page

- `GET /pages/` → List all pages (paginated)
- `GET /pages/{page_id}` → Retrieve a page
- `PUT /pages/{page_id}` → Update a page
- `DELETE /pages/{page_id}` → Delete a page

Sharing

- `POST /pages/{page_id}/share` → Generate share link
- `DELETE /pages/{page_id}/share` → Revoke share access
- `GET /public/{share_token}` → Access shared page (read-only)

AI Fact-Checking

- `POST /fact-check` → Verify selected text via Gemini

---

7. Data Schemas (Pydantic Models)

- **PageBase**: Common fields (title, content)
- **PageCreate**: For creation requests
- **PageRead**: Full page data with metadata
- **ShareResponse**: Token and share URL
- **FactCheckRequest**: Text input for fact-check
- **FactCheckResponse**: Verdict with explanation

---

8. Setup & Installation

Prerequisites

- Docker & Docker Compose
- Google Gemini API Key

Steps

1. Clone the repository
2. Navigate to project root
3. Copy `backend/.env.example` → `backend/.env`

4.Add credentials:

```
GEMINI_API_KEY=your_api_key_here
DATABASE_URL=postgresql://user:password@db:5432/loupe
```

5.Run:

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

---

9. Usage Guide

Creating Pages

- Rich-text formatting (bold, italic, underline, etc.)
- Lists, headings, block quotes
- Drag-and-drop block reordering

Fact-Checking

- Select text in editor

- Click fact-check button
- AI returns verdict + explanation

Sharing

- Generate unique share URL
- Public pages are **read-only**

---

10. Migration System

Custom migration script (`migrate_db.py`) includes:

- Adding missing columns (`share_token`, `is_public`)
- Uses raw SQL with SQLAlchemy
- Logs migration status

---

11. Security Considerations

- CORS enabled for development
- Gemini API key stored in `.env`
- Public sharing managed via `share_token`
- **Authentication not yet implemented**

---

12. Performance Features

- Async FastAPI endpoints
- SQLAlchemy session management
- HTTP connection pooling
- Indexed database columns
- JSONB operations for efficient querying

---

13. Future Improvements

To enhance Loupe, the following features and improvements can be added in the future:

- **Authentication & Security** → User accounts, JWT/OAuth, and rate limiting
- **Collaboration** → Real-time editing with WebSockets and version history
- **Advanced Features** → Full-text search, templates, export options (PDF/Markdown/HTML)
- **Performance & Scalability** → Redis caching, database optimization, load balancing, CDN
- **Code & Monitoring** → Automated testing, CI/CD, and error monitoring

---

14. Conclusion

The **Loupe project** demonstrates a modern, scalable architecture for a rich-text editor with AI integration.
It provides a strong foundation for future features like **real-time collaboration, advanced search, and secure multi-user support**.