**📘 NotesAI Project Enhancement Plan (Detailed)**

This document outlines the step-by-step enhancement roadmap for the NotesAI application, divided into 3 strategic phases. Each phase includes the goal, detailed features, procedures, and recommended technologies for implementation.

**✅ Phase 1: Core Features & UX Enhancements (MVP+ Level)**

**🎯 Goal**

Refine existing functionality and improve user experience for a more stable MVP product.

**🔧 Features, Procedure & Technologies**

**1. Edit Notes Functionality**

* **Procedure:**
  + Add an "Edit" button for each note in the UI.
  + Create /edit/<note\_id> route in Flask.
  + When note is edited, update content in SQLite.
  + Recompute new embeddings and update document in ChromaDB.
* **Technologies:** Flask, SQLite, ChromaDB, HTML Forms

**2. Tagging / Categorization**

* **Procedure:**
  + Add a text input for tags in the note creation form.
  + Save tags in SQLite with each note.
  + Allow filtering by tag in the frontend.
* **Technologies:** SQLite (new column), HTML, Flask routing logic

**3. Note Preview in Answers**

* **Procedure:**
  + During response generation, retrieve note titles or snippets from Chroma metadata.
  + Display them below the AI's answer.
* **Technologies:** Jinja2 templating, Flask, ChromaDB get\_metadata()

**4. Responsive UI and Styling**

* **Procedure:**
  + Integrate Bootstrap or TailwindCSS.
  + Improve layout, fonts, and spacing.
* **Technologies:** Bootstrap/Tailwind, HTML/CSS

**5. Clear ChromaDB Button (In Progress)**

* **Procedure:**
  + Implement Flask endpoint to call ChromaDB delete\_collection() or delete\_documents().
  + Add confirmation dialog on frontend.
* **Technologies:** Flask, ChromaDB, JavaScript (optional)

**🗓 Timeline:**

* Week 1–2: Edit Notes + Tagging
* Week 3: UI/Preview integration
* Week 4: Testing, bug fixing

**🚀 Phase 2: Advanced AI Features & Input Flexibility**

**🎯 Goal**

Enhance AI capabilities, increase flexibility in note inputs and allow better interactions.

**🤖 Features, Procedure & Technologies**

**1. Multi-turn Conversation Support**

* **Procedure:**
  + Use Flask session or Redis to store chat history.
  + Pass prior question and answer as context to the prompt.
* **Technologies:** Flask session, GPT prompt engineering

**2. Semantic Note Search**

* **Procedure:**
  + Add a search bar to query ChromaDB using similarity search.
  + Display matching notes without GPT.
* **Technologies:** ChromaDB .query(), Flask, HTML

**3. Summarize Notes**

* **Procedure:**
  + Add a "Summarize" button on each note card.
  + Pass note content to GPT with a summary prompt.
* **Technologies:** GPT API, Flask route, JavaScript (AJAX optional)

**4. Upload Notes via File**

* **Procedure:**
  + Accept .txt, .pdf, .docx files in a form.
  + Use PyMuPDF, python-docx, or pdfplumber to extract text.
  + Add extracted text as new notes.
* **Technologies:** Flask request.files, file parsers, ChromaDB

**5. Rate or Mark Important Notes**

* **Procedure:**
  + Add "Star/Favorite" toggle in UI.
  + Store a boolean or rating in SQLite.
  + Option to filter favorites.
* **Technologies:** SQLite, Flask, CSS (star toggle)

**🗓 Timeline:**

* Week 5–6: Multi-turn + Semantic Search
* Week 7: Upload Notes
* Week 8: Summarize + Rating

**☁️ Phase 3: User Management, Deployment, & APIs**

**🎯 Goal**

Make the app cloud-ready, support multiple users, and enable REST API access.

**🌐 Features, Procedure & Technologies**

**1. User Authentication**

* **Procedure:**
  + Implement login/signup forms.
  + Use hashed passwords (bcrypt).
  + Associate notes and questions with user ID.
* **Technologies:** Flask-Login, SQLite, bcrypt

**2. Admin Dashboard**

* **Procedure:**
  + View total users, note stats, popular tags.
  + Ability to delete notes/users from backend.
* **Technologies:** Flask Admin Panel, Jinja2

**3. REST API Layer**

* **Procedure:**
  + Convert major functions (add\_note, ask, delete, etc.) into Flask API routes.
  + Return JSON instead of rendering templates.
* **Technologies:** Flask RESTful, Postman (testing)

**4. Dockerize App**

* **Procedure:**
  + Create a Dockerfile and docker-compose.yml.
  + Include SQLite persistence and volume mounting for Chroma.
* **Technologies:** Docker, Docker Compose

**5. Deploy to Cloud**

* **Procedure:**
  + Choose Render, GCP, or AWS EC2.
  + Configure environment variables and volume persistence.
* **Technologies:** Render/GCP/AWS, Gunicorn, Nginx (optional)

**6. (Optional) Analytics Dashboard**

* **Procedure:**
  + Use Plotly/Dash or embed charts in admin dashboard.
  + Track search terms, answer counts, tags used.
* **Technologies:** Plotly, Chart.js, Flask

**🗓 Timeline:**

* Week 9–10: Auth + Admin
* Week 11: API routes
* Week 12: Docker + Deploy

**🧩 Summary Table**

| **Phase** | **Focus** |  |  |  |  | **Weeks** |  |  |  |  | **Key Deliverables** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Core Features + UI |  |  |  |  | 1–4 |  |  |  |  | Tags, Edit, Styling, Clear Chroma |
| 2 | Smart AI + Flexibility |  |  |  |  | 5–8 |  |  |  |  | Summarize, Upload, Multi-turn |
| 3 | User-ready & Deployment |  |  |  |  | 9–12 |  |  |  |  | Auth, API, Docker, Cloud hosting |