**Project Setup (10 steps)**

1. Install Node.js & npm
2. Install Python 3.8+
3. Create a new Next.js app (npx create-next-app)
4. Create a Python virtual environment (python -m venv venv)
5. Activate your Python virtual environment
6. Initialize a Git repo for your project
7. Create .gitignore for node\_modules, venv, etc.
8. Create basic folder structure: /frontend, /backend
9. Initialize npm in /frontend
10. Initialize Python project in /backend (add requirements.txt)

**Backend Environment Setup (10 steps)**

1. Install FastAPI (pip install fastapi uvicorn)
2. Install LangChain (pip install langchain)
3. Install ChromaDB (pip install chromadb)
4. Install Gemini API SDK or client (if available)
5. Install additional dependencies (e.g., python-dotenv)
6. Setup .env file for API keys (Gemini, Chroma config)
7. Create main FastAPI app file (main.py)
8. Add CORS middleware for frontend communication
9. Setup basic root endpoint / to test server
10. Run uvicorn main:app --reload and confirm server runs

**Vector Store & Embeddings (15 steps)**

1. Choose embedding model (Gemini embeddings or OpenAI temporarily)
2. Implement code to generate embeddings from text input
3. Load sample documents (PDFs, text) for testing
4. Preprocess documents (clean text, remove noise)
5. Split documents into chunks (e.g., 500-1000 chars)
6. Generate embeddings for each chunk
7. Setup ChromaDB instance locally or cloud
8. Create functions to insert document embeddings into ChromaDB
9. Create functions to query similar documents from ChromaDB
10. Test inserting and querying embeddings manually
11. Wrap embedding + vector store calls in reusable classes
12. Integrate LangChain’s vectorstore wrapper for ChromaDB
13. Test LangChain + ChromaDB embedding insertion and retrieval
14. Handle edge cases (empty docs, API errors)
15. Log embedding + retrieval performance metrics

**RAG Chain Implementation (15 steps)**

1. Setup LangChain chains for retrieval + generation
2. Implement prompt templates for Gemini chat API
3. Setup retrieval chain combining Chroma retriever and Gemini LLM
4. Build function to accept user query, run retrieval, then generation
5. Add logic to limit number of retrieved chunks (e.g., top 5-10)
6. Ensure context chunks are concatenated properly in prompt
7. Handle token length limits and chunk truncation
8. Test RAG function with static queries and print results
9. Add error handling for failed API calls
10. Create unit tests for retrieval + generation pipeline
11. Optimize prompt for concise, relevant answers
12. Integrate temperature and other generation parameters
13. Setup caching of frequent queries (optional)
14. Log user queries and model responses (for improvement)
15. Ensure security of API keys in backend

**FastAPI Endpoints (10 steps)**

1. Design REST API endpoints (/chat, /upload-doc, /status)
2. Implement /upload-doc to accept PDF/text uploads
3. Parse uploaded documents and add to ChromaDB embeddings
4. Implement /chat endpoint that accepts user messages
5. Call RAG chain inside /chat endpoint to get responses
6. Return chat response JSON to frontend
7. Add authentication or API key validation (optional)
8. Add health check /status endpoint
9. Test all endpoints with Postman or curl
10. Add logging for all API requests and errors

**Frontend Setup (15 steps)**

1. Setup Next.js pages and components folder
2. Create basic chat UI (input box, send button, message list)
3. Implement API call to FastAPI /chat endpoint on submit
4. Display user messages and assistant replies in chat window
5. Style chat UI for usability and responsiveness
6. Add file upload component to send PDFs to /upload-doc
7. Show upload status and errors to user
8. Manage chat state with React hooks or Context API
9. Add loading indicator while waiting for API response
10. Handle errors gracefully (API failure, network issues)
11. Implement scroll-to-bottom on new messages
12. Save chat history locally (optional)
13. Add environment variables for API URLs
14. Test frontend independently with mock API
15. Prepare for deployment (build, optimize)

**Integration & Testing (10 steps)**

1. Run backend and frontend locally together
2. Test full flow: upload document → ask question → get answer
3. Fix any CORS or connection issues
4. Add unit tests for backend endpoints
5. Add integration tests for frontend-backend communication
6. Conduct user testing and gather feedback
7. Monitor logs and fix bugs or latency issues
8. Optimize vector store update speed and search latency
9. Validate response quality and tweak prompts or chunk size
10. Add rate limiting or usage tracking (optional)

**Deployment (10 steps)**

1. Choose hosting for backend (AWS, DigitalOcean, Heroku, etc.)
2. Choose hosting for frontend (Vercel, Netlify, etc.)
3. Setup CI/CD pipelines for both frontend and backend
4. Secure environment variables in deployment platform
5. Deploy backend API and test publicly
6. Deploy frontend and connect to backend URL
7. Test production environment fully end-to-end
8. Setup monitoring and alerting for API uptime
9. Document usage and APIs for users/developers
10. Plan for future updates and data refreshes

**Maintenance & Scaling (10 steps)**

1. Monitor user queries and chatbot performance regularly
2. Update company docs and reindex ChromaDB periodically
3. Improve prompt engineering based on user feedback
4. Scale backend resources as user base grows
5. Explore adding features: user accounts, analytics, multilingual support