Generative AI .NET Project – Knowledge Base

# Generative AI .NET Project – Knowledge Base

---

## Phase 1 – Foundation

* - Explored use cases in .NET apps (chatbots, Q&A bots, summarization, content generation, code assist).
* - Tested HuggingFace models for API demo + Azure deployment.
* - Learned Prompt Engineering: zero-shot, few-shot, role prompting.
* - Findings: Few-shot & role prompting gave better results; zero-shot was verbose/unreliable.

---

## Phase 2.1 – .NET Backend & Frontend Integration

* ✅ \*\*Backend\*\*
* - .NET Core Web API with `ChatController` endpoints (send, stream, history, sessions, duplicate-session, delete).
* - Services layer with `IOpenAiService` + `IChatHistoryService`.
* - Persistence in SQL Server with `ChatSessions` + `ChatMessages`.
* ✅ \*\*Frontend\*\*
* - Modularized JS files: `chat.js`, `sessions.js`, `templates.js`, `utils.js`, `main.js`.
* - Features: streaming, typing dots, session sidebar, model selector, prompt preview.
* ✅ \*\*Deployment\*\*
* - Azure App Service + SQL Azure.
* - Fixed CORS + connection strings in App Config.

---

## Phase 2.2 – Prompt Templates + Clean UI

* - Templates stored in DB with parameters (tone, length).
* - `buildPrompt()` inserts parameters into templates.
* - `buildPromptPreview()` updates preview instantly.
* - Admin panel for template CRUD (planned in Phase 2.3).

---

## Phase 2.2 – Bug Fixes & Enhancements

* - Fixed `sessionId undefined` bug.
* - Fixed circular imports between `chat.js` and `sessions.js`.
* - Streaming shows typing dots before assistant reply.
* - Cloning sessions now copies both user + assistant messages.
* - User messages persisted before OpenAI call.

---

## Phase 2.3 – Admin Panel for Prompt Templates

* ✅ \*\*AdminLTE Integration\*\*
* - Added `/Admin` area in ASP.NET Core MVC.
* - Integrated AdminLTE v4 theme for consistent UI.
* - Sidebar navigation (Templates, Users [future], Analytics [future]).
* ✅ \*\*Authentication\*\*
* - Implemented custom authentication with backend-issued JWT.
* - Frontend (Admin) stores JWT in cookies and sends with requests.
* - Added Login/Logout flow.
* ✅ \*\*CRUD for Templates\*\*
* - Templates managed via `/Admin/Templates`.
* - Fields: `Name`, `KeyName`, `TemplateText`.
* - Data served entirely from backend API (`/api/PromptTemplate`).
* ✅ \*\*Parameterized Prompts\*\*
* - Dynamic parameter editor in Create/Edit pages.
* - Supports `text`, `number`, `boolean`, `select`, `multiselect`.
* - Parameters include: `Name`, `KeyName`, `Type`, `Options`, `DefaultValue`, `IsRequired`, `RegexPattern`.
* ✅ \*\*Live Preview\*\*
* - Preview updates instantly when editing TemplateText or parameters.
* - Inline highlighting of invalid values (red, tooltip with error).
* - Error list below preview for full context.
* - Admin can disable validation (persistent toggle, with reset button).
* ✅ \*\*Versioning\*\*
* - Template versions saved on update.
* - Version History modal lists all versions with rollback + compare.
* - Rollback restores older version to active.

---

## Project Structure

```

├──2\_OpenAIChatDemo

├──├──2\_OpenAIChatDemo

├──├──├── Controllers (AuthController.cs,ChatController.cs,PromptController.cs)

├──├──├── Data (ChatDbContext.cs)

├──├──├── TOs (ChatHistroryDto, ChatRequestDto, ChatResponseDto, ChatSessionDto, ChatMessageDto, LoginDto, PromptReviewDto, PromptPreviewResultDto, PromptTemplateCreatedDto, PromptTemplateDto, PromptTemplateParameterDto, PromptTemplateUpdateDto, PromptTemplateVersionDto)

├──├──├── odels (AdminUser. ChatSession, ChatMessage, ChatRequest, PromptTemplate, PromptTemplateParameter, PromptTemplateVersion)

├──├──├── ervices (IAdminAuthService, AdminAuthService, IOpenAiService, OpenAiService, IChatHistoryService, ChatHistoryService, IPromptTemplateService, PromptTemplateService)

├──├──├── ettings(OpenAISettings)

├──├──├── pplicationsettings (applicationsettings.Developement)

├──├── 2\_OpenAIChatFrontEnd

├──├──├── Areas/Admin/Controllers(TemplateController.cs)

├──├──├── Areas/Admin/DTOs (PromptTemplateCreateDto, PromptTemplateDto, PromptTemplateParameterDto,PromptTemplateUpdateDto)

├──├──├── Areas/Admin/Views/Shared(\_AdminLayout)

├──├──├── Areas/Admin/Views/Template(\_TemplateForm, Create, Edit, Index)

├──├──├── Areas/Admin/Views/(\_ViewStart)

├──├──├── Areas/Admin (AdminArea.cs)

├──├──├── Controllers (AccountController.cs, BaseController.cs)

├──├──├── Helper(SessionExtention.cs)

├──├──├── Models (ChatMessage, ChatRequest,)

├──├──├── Views/Account (Login)

├──├──├── Views/Home (Index,Privacy)

├──├──├── Views/Shared (\_Layout, \_ValidationScriptsPartial, Error)

├──├──├── Views (\_ViewImports, \_ViewStart)

├──├──├── applicationsettings (applicationsettings.Developement)```

---

## Phase 2.4 – Next Steps

* - Session cloning for cross-model comparisons.
* - Allow testing same input across GPT, Gemini, Claude, etc.
* - Side-by-side comparison view in Admin.

# Phase 2.4 – Session Cloning & Cross-Model Comparisons

## Backend

- Introduced ComparisonService (IComparisonService) for cross-model testing.  
- Added LLMProviders folder with pluggable providers (OpenAI, Gemini, Claude).  
- Built RunComparisonAsync to handle multiple models per input.  
- Implemented error handling + persistence (ComparisonResults, SessionComparisons).  
- Added GetHistoryAsync and GetHistoryByIdAsync for fetching comparisons.  
- Persisted error codes/messages in DB.  
- Fixed duplicate results bug (Distinct + one add per provider:model).

## Frontend (AdminLTE)

- Added /Admin/Comparison page with side-by-side grid for provider/model outputs.  
- Model & provider dropdown, input field, run button.  
- Loading spinner + auto-scroll to results.  
- Deduplication filter in JS.  
- Added /Admin/Comparison/History with DataTables + modal view.  
- Deduplication applied to modal + table.

## Improvements

- Fixed login DI bug (IAdminAuthService).  
- Clean separation of business logic into ComparisonService.  
- Reusable DTOs: ComparisonResultDto, ComparisonHistoryDto.  
- Stable for OpenAI + Gemini, Claude ready.

## Updated Project Structure (Phase 2.4)

/2\_OpenAIChatDemo  
 ├── 2\_OpenAIChatDemo (Backend)  
 │ ├── Controllers (AuthController.cs, ChatController.cs, PromptController.cs, ComparisonController.cs)  
 │ ├── Data (ChatDbContext.cs)  
 │ ├── DTOs (ChatRequestDto, ChatResponseDto, ChatSessionDto, ChatMessageDto, ComparisonRequestDto, ComparisonResultDto, ComparisonHistoryDto)  
 │ ├── Models (ChatSession, ChatMessage, ComparisonResult, SessionComparison, AdminUser, PromptTemplate)  
 │ ├── Services (IOpenAiService, OpenAiService, IChatHistoryService, ChatHistoryService, IPromptTemplateService, PromptTemplateService, IComparisonService, ComparisonService)  
 │ ├── LLMProviders (OpenAiProvider.cs, GeminiProvider.cs, ClaudeProvider.cs)  
 │ ├── Program.cs, appsettings.Development.json  
 │  
 ├── 2\_OpenAIChatFrontEnd (Frontend)  
 │ ├── Areas/Admin/Controllers (TemplateController.cs, ComparisonController.cs)  
 │ ├── Areas/Admin/Views/Template (Index.cshtml, Create.cshtml, Edit.cshtml)  
 │ ├── Areas/Admin/Views/Comparison (Index.cshtml, History.cshtml)  
 │ ├── wwwroot/admin/js (template-admin.js, comparison.js, comparison-history.js)  
 │ ├── Views/Home/Index.cshtml, Views/Shared/\_Layout.cshtml  
 │  
 ├── sql (create\_tables.sql, update\_comparison\_results.sql)

# Phase 3 – Retrieval-Augmented Generation (RAG)

## 🔹 RAG Architecture

1. Ingestion Layer  
- Document upload, parsing, and chunking (PDF, DOCX, Markdown, SQL text).  
- Clean text storage + chunking by semantic boundaries.  
  
2. Embedding & Storage Layer  
- Generate vector embeddings (OpenAI or HuggingFace).  
- Store in Postgres + pgvector or Azure Cognitive Search.  
- Maintain metadata in SQL (docId, chunkId, tags, owner, version).  
  
3. Retrieval Layer  
- Semantic + hybrid search over embeddings.  
- Ranking, deduplication, filters.  
  
4. Augmentation + Generation Layer  
- Retrieved chunks + user query → LLM.  
- Compare baseline LLM vs RAG-enhanced responses.  
- Debug mode: show retrieved chunks in Admin Panel.

## 🔄 Phase 3 Sub-Phases

Phase 3.1 – Document Ingestion & Chunking  
- Admin panel document upload.  
- Parsing (PDF/DOCX/MD → text).  
- Store text + chunks in SQL.  
  
Phase 3.2 – Embeddings & Vector Storage  
- Generate embeddings.  
- Store in pgvector or Azure Cognitive Search.  
- Link embeddings with metadata.  
  
Phase 3.3 – Retrieval Service  
- Build IRetrievalService.  
- API endpoint /api/retrieval/search.  
- Top-k semantic search results.  
- Admin Panel: show retrieval debug info.  
  
Phase 3.4 – RAG Pipeline Integration  
- Build IRagService.  
- Input: user query → retrieval → augmented prompt → LLM.  
- Compare baseline vs RAG outputs.  
  
Phase 3.5 – Admin Panel Enhancements  
- Add Documents Page (upload, list, delete, version).  
- Add RAG Comparison Page (baseline vs RAG, chunk debug view).  
  
Phase 3.6 – Deployment & Optimization  
- Deploy vector DB (Azure or pgvector).  
- Optimize retrieval (indexes, caching).  
- Add monitoring (queries/sec, storage growth).

Phase 3 RAG Architecture Diagram:

## ⚡ Expected Outcomes

- RAG-enabled chatbot in .NET + AdminLTE project.  
- Admin panel for document + RAG debugging.  
- Comparison: baseline vs RAG-enhanced responses.  
- Scalable + production-ready retrieval pipeline.

