# AirNir – Generative AI Platform

## Knowledge Base (Phase 1 → Phase 3.1)

### Phase 1 – Foundation

- Explored Generative AI use cases in .NET apps (chatbots, Q&A, 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, persistence in SQL Server.  
✅ Frontend: Modularized JS, streaming chat, model selector, session sidebar.  
✅ Deployment: Azure App Service + SQL Azure.

### Phase 2.2 – Prompt Templates + Clean UI

- Templates stored in DB with parameters.  
- Prompt preview updates instantly.  
- Admin panel planned for CRUD.

### Phase 2.3 – Admin Panel for Prompt Templates

✅ AdminLTE Integration with ASP.NET Core MVC.  
✅ JWT-based authentication.  
✅ CRUD for Prompt Templates with parameters & validation.  
✅ Live Preview, validation errors, and versioning support.

### Phase 2.4 – Session Cloning & Cross-Model Comparisons

- Backend: ComparisonService with pluggable providers (OpenAI, Gemini, Claude).  
- Frontend: Admin Comparison UI (side-by-side results, history).  
- Improvements: Bug fixes, deduplication, DTO cleanup.

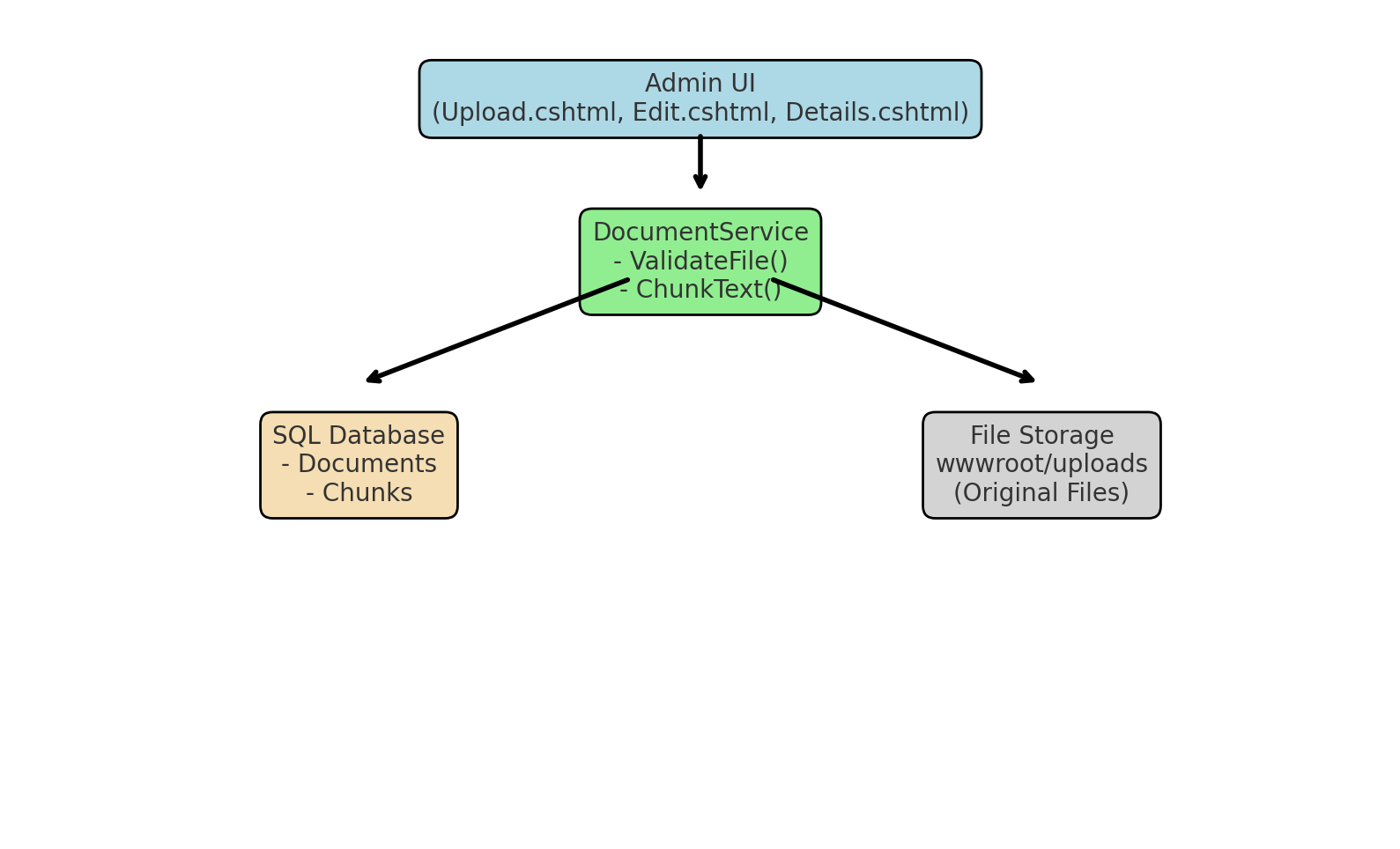
### Phase 3.1 – Document Ingestion & Chunking

- Goal: Structured ingestion of TXT, PDF, DOCX, Markdown files.  
- Entities: Document + Chunk with relationships.  
- Validation rules driven by appsettings.json (AllowedTypes, MaxFileSize).  
- Business logic in DocumentService with DTO mapping.  
- Admin UI (Upload, Edit, Delete, Details, List) using AdminLTE & Bootstrap 5.  
- Preview: PDF inline, TXT chunks, DOCX download fallback.  
- Client-side + backend validation synced with config.  
- Storage: Metadata + chunks in SQL Server, original files in wwwroot/uploads.  
- Outcome: Robust ingestion pipeline, documents prepared for embeddings.

### Architecture (Phase 3.1)

┌───────────────┐  
 │ Admin UI │ (Upload.cshtml, Edit.cshtml, Details.cshtml)  
 └──────┬────────┘  
 │  
 ▼  
 ┌──────────────────┐  
 │ DocumentService │  
 │ - ValidateFile() │  
 │ - ChunkText() │  
 └──────┬───────────┘  
 │  
 ┌───────────┴───────────┐  
 ▼ ▼  
 ┌───────────────┐ ┌───────────────┐  
 │ SQL Database │ │ File Storage │  
 │ Documents │ │ wwwroot/uploads  
 │ Chunks │ │ (Original Files)  
 └───────────────┘ └───────────────┘

### Architecture Diagram (Visual)



## Phase 3.3 – Retrieval Service

- Implemented RetrievalService with Semantic (pgvector), Keyword (SQL Server FTS), and Hybrid search.  
- Hybrid fallback: defaults to semantic if no keyword hits.  
- Query sanitization: tokenized & escaped input for FTS.  
- Admin Debug UI: side-by-side semantic & hybrid results, filters, counters, SLA monitoring (<300ms).  
- Direct DI calls used in Admin UI (no API endpoints).  
- Outcome: Retrieval pipeline production-ready and optimized.

## Updated Project Structure

/AirNir  
├── Library  
│ ├── ArNir.Core → Entities, DTOs, Config, Validations  
│ ├── ArNir.Data → DbContexts (SQL Server + Postgres), EF Migrations  
│ └── ArNir.Services → Business logic Service, Interface, Helper, Mapping (EmbeddingService, RetrievalService, RagService)  
├── Presentation  
│ ├── ArNir.Admin → AdminLTE UI Controllers, ViewModel, Views (embedding + retrieval test pages, RAG comparison)  
│ └── ArNir.Frontend → End-user search/chat interface (future, Phase 3.4+)

Architecture Reference: see docs/Phase3\_RAG\_Architecture.png