CoreApp – Project Technical Report

1. Project Overview

CoreApp is a Clean Architecture .NET 9 solution. It separates layers clearly: Domain → Application → Infrastructure → WebAPI → Blazor WebAssembly UI. It also contains a Core.AI module for AI features like chat, streaming responses, and function calling.

2. Domain Layer

Purpose: Holds business rules and core entities. Entities: User, Role, RefreshToken. Style: Pure C# classes with no external package dependencies. Benefit: Keeps business logic independent from infrastructure.

3. Application Layer

Purpose: Contains use cases (CQRS commands and queries). Key Tech: MediatR implements CQRS, each request (Command/Query) has a handler. FluentValidation validates requests before running. Pipeline Behaviors: ValidationBehavior, LoggingBehavior, PerformanceBehavior, UnhandledExceptionBehavior, UnitOfWorkBehavior exists but is not registered yet. Goal: Clean, testable, single-responsibility use cases.

4. Infrastructure Layer

Purpose: Implements persistence and external services. Database: SQL Server with EF Core. DbContext: CoreAppDbContext manages Users, Roles, RefreshTokens. UnitOfWork implemented for transaction control but not plugged into MediatR pipeline yet. Auth Services: AuthService for user registration/login/refresh token, TokenService creates JWT tokens, PBKDF2 password hashing with SHA256 and salt.

5. WebAPI Layer

Purpose: Exposes REST endpoints for the app. Key Tech: JWT Authentication with Microsoft.AspNetCore.Authentication.JwtBearer. Swagger (Swashbuckle) for API documentation + JWT support. CORS is open for development. System.Text.Json used for serialization. Controllers: AuthController (Register, Login, Refresh) and AiController/AgentController for AI chat.

6. Core.AI Module

Purpose: Makes AI usage provider-agnostic. Main Interfaces: IAIService, IAgentService, IAIModelProvider. Providers: OpenRouterAiService uses OpenRouter API, OllamaAiService calls local Ollama models. Features: Function calling, optional MCP integration, Semantic Kernel Agent with memory. Config in appsettings.json defines providers, models, and profiles.

7. Blazor WebAssembly UI

Purpose: Front-end for authentication and AI chat. Key Tech: AuthenticationStateProvider handles JWT, AuthMessageHandler adds token and auto-refreshes on 401, SessionStorageTokenStore keeps tokens in browser. Pages: Login, Register, Chat with streaming and agent profiles. HttpClient configured for API calls.

8. Authentication Flow

User registers/logs in via AuthController, AuthService validates and creates JWT via TokenService, tokens saved in DB, Blazor stores them and refreshes when expired.

9. AI Flow

User sends a prompt in Blazor, AiController calls IAIService which selects provider (OpenRouter/Ollama) and streams back. Agent chat uses SemanticKernelAgentService with memory and profiles.

10. Observed Gaps & Improvements

Authorization marker mismatch, UnitOfWorkBehavior not registered, secrets should be in env variables, CORS too open, no CI/CD, little testing, no abuse prevention on AI endpoints.

11. Overall

This project is a modern, modular, and scalable .NET 9 boilerplate. It uses Clean Architecture, CQRS, MediatR, FluentValidation, EF Core, JWT, Blazor WASM, and advanced AI integration with OpenRouter, Ollama, Semantic Kernel, and function calling. With some fixes it can become production-ready.

Appendix – AI Technologies Explained

This appendix explains each AI technology used in the project. It describes what it is, what it does, and how it is used in this system.

A. OpenRouter

What it is: OpenRouter is an API gateway that gives access to many Large Language Models (LLMs) with one interface. It supports chat completions, streaming tokens, and optional tool (function) calling.

What it does: It routes your request to a selected model (for example Llama or other hosted models). You pass a list of messages and configuration. It returns tokens as a stream or as a full response.

How this system uses it: The OpenRouterAiService builds a chat completion request from user messages. It reads API key and default model from configuration. If streaming is enabled, it parses chunks and sends them back to the client in real time. If tools are provided, the model can request a tool call which the system can dispatch.

B. Ollama

What it is: Ollama is a local model runner. It runs LLMs on your machine or server and exposes a simple HTTP API.

What it does: It loads models like Llama or Mistral and generates text. It can stream the output line by line.

How this system uses it: The OllamaAiService calls the /api/generate endpoint, forwards the prompt, and streams the generated tokens. It is useful when you want control, privacy, or lower cost with local inference.

C. Semantic Kernel (SK)

What it is: Semantic Kernel is a Microsoft framework for building AI apps. It helps with prompts, memories, planners, and connectors to many AI providers.

What it does: It lets you build an 'agent' that can keep a chat history, follow a system prompt, and call different models and skills.

How this system uses it: The SemanticKernelAgentService keeps per-user chat history and applies an agent profile (a predefined system prompt). It talks to a provider (OpenRouter or Ollama) through the configured connectors and streams the answer back.

D. Function Calling (Tools)

What it is: Function calling lets the model ask the host application to run a specific function (tool) with arguments. The model selects a tool when it needs external data or actions.

What it does: The app exposes tool schemas (names, parameters). The model replies with a tool call. The host runs the function and returns the result to the model.

How this system uses it: The project has an IFunctionRegistry to register functions and a FunctionSchemaGenerator to build tool schemas for the provider. The dispatcher runs the requested tool and returns the serialized result to continue the chat.

E. Provider-Agnostic Abstraction

What it is: A design where your code does not depend on a single AI provider. You can swap the backend easily.

What it does: It defines IAIService and resolves the concrete provider at runtime (OpenRouter or Ollama).

How this system uses it: The AIServiceResolver checks request options and configuration to pick the right provider and model. This makes the system flexible and testable.

F. Chat Memory

What it is: A way to remember past messages in a session. It helps the model keep context across many turns.

What it does: It stores the conversation history in memory or storage and appends it to the next prompt.

How this system uses it: The SemanticKernelAgentService uses a ChatHistory store keyed by user/session. It adds past messages and the system prompt so the model can answer with context.

G. MCP (Tool Server Protocol)

What it is: MCP in this project is a simple tool server protocol over WebSocket. It lets the app connect to external 'tool servers' and call their functions.

What it does: The server exposes a list of tools and accepts JSON-RPC style requests. The client sends a function call and receives the result.

How this system uses it: The McpClient opens a WebSocket connection to a configured server. A dispatcher proxy can forward tool calls to that server. This allows extending the system with external tools without changing core code.

H. Agent Profiles

What it is: Predefined roles with system prompts and settings. Each profile changes the behavior of the agent (for example 'Senior Developer' or 'Teacher').

What it does: It sets the system prompt and sometimes model or temperature. Users can pick a profile per chat.

How this system uses it: Profiles are stored in configuration (appsettings). The UI lets the user select a profile. The agent service applies that profile when building the prompt.

I. Streaming Responses

What it is: A way to send the answer token by token. The user sees partial text quickly.

What it does: The provider sends chunks. The backend forwards them to the browser. The UI appends tokens to the message view.

How this system uses it: Both providers support streaming. The WebAPI streams the response and the Blazor UI renders it live. It improves UX and perceived speed.

J. Configuration & Model Catalog

What it is: Central settings that hold providers, API keys, default models, and agent profiles.

What it does: It defines which provider to use by default, available models per provider, and safe defaults.

How this system uses it: AISettings and specific provider settings are bound from appsettings.json. The app reads them at startup and during requests to select a provider and model.

K. Safety, Limits, and Observability

What it is: Controls to keep the system safe and stable under load.

- Rate limiting and request size limits on AI endpoints
- Validation for prompt length and required fields
- Structured logging and tracing for debugging
- Graceful error handling for provider/network failures