OllamaNet - Project Brief

Project Overview

A comprehensive .NET-based microservices architecture that enables seamless integration with a wide suit of open Source Models, an open-source large language model framework. The platform provides a complete solution for AI model management, conversation capabilities, user authentication, administration, and model exploration through a collection of specialized microservices working together in a distributed system. The platform is designed to deliver a robust, scalable, and maintainable solution for organizations looking to leverage large language models in their applications.

Core Components

Gateway Service

API gateway using Ocelot for unified access to all microservices with:

- Centralized routing to appropriate backend services
- JWT authentication and claims forwarding
- Role-based access control
- Modular configuration with service-specific files
- Dynamic configuration reloading
- Variable substitution for service URLs

Conversation Service

Manages all aspects of user conversations with AI models including:

- Conversation organization with folders
- Message history persistence and retrieval
- Folder Organization Support for organizing conversations in user folders
- Note creation and management
- Feedback collection on AI responses
- RAG (Retrieval-Augmented Generation) with vector database integration
- Document processing and text extraction
- Sophisticated caching strategy for performance optimization

Auth Service

Handles user authentication, authorization, and profile management with:

- JWT-based authentication with comprehensive validation
- Refresh token functionality for persistent sessions
- Password management (reset, change, forgot password flows)
- Role-based authorization with admin capabilities
- User profile management
- Secure cookie handling for refresh tokens

Admin Service

Provides comprehensive administrative capabilities for platform management including:

- User management (create, read, update, delete users; manage roles)
- AI model management (add, update, delete models; manage metadata)
- Tag management (create, update, delete tags for categorization)
- Inference operations (install, uninstall, manage AI models)
- Progress streaming for long-running operations
- Domain-driven design with clear separation of concerns
- Robust validation and error handling

Explore Service

Enables discovery and browsing of available AI models with:

- Paginated lists of available AI models
- Detailed information about specific models
- Tag browsing and filtering
- Cached model metadata for performance
- Resilient caching with circuit breaker patterns
- Efficient data retrieval strategies

Shared Infrastructure

- **DB layer**: Common data access layer with repositories and entity definitions
- **Redis Cache**: Distributed caching with domain-specific TTL values
- **SQL Server**: Centralized database for all services
- **Service Defaults**: Common configuration and middleware components

Project Goals

1. Create a modular, maintainable platform for interacting with Ollama AI models
2. Provide seamless conversation capabilities with real-time streaming responses
3. Enable secure user authentication and resource management
4. Offer comprehensive administration tools for platform management
5. Support efficient discovery and exploration of available AI models
6. Implement robust caching and performance optimization across all services
7. Ensure high availability and fault tolerance through resilient design patterns
8. Support document-based context enhancement through RAG capabilities
9. Deliver a platform suitable for both development and production environments
10. Create a foundation for future AI-powered applications and services

Key Requirements

Functional Requirements

1. **User Authentication**: Registration, login, password management, and persistent sessions
2. **Conversation Management**: Create, retrieve, update, delete, and organize conversations

3. **Real-time Chat**: Process user messages and stream AI model responses
4. **Model Management**: Add, update, remove, and categorize AI models
5. **Model Discovery**: Browse, search, and filter available models
6. **Document Processing**: Upload, process, and use documents for context enhancement
7. **Administrative Controls**: Comprehensive platform management capabilities
8. **Folder Organization**: Organize conversations in hierarchical folders
9. **Note Management**: Create and manage notes associated with conversations
10. **Feedback Collection**: Gather and process user feedback on AI responses

Success Criteria

1. All microservices deployed and functioning in both development and production environments
2. Real-time conversation capabilities with streaming responses functioning correctly
3. Authentication system providing secure access with role-based permissions
4. Administrative interface allowing complete platform management
5. Model exploration system providing efficient discovery of available models
6. Caching system demonstrating significant performance improvements
7. System capable of handling the target user load with acceptable performance
8. Comprehensive documentation available for all system components
9. RAG capabilities enhancing conversation context with document-based information
10. System meeting all defined performance and availability targets

Stakeholders

1. **Development Team**: Responsible for implementation and maintenance
2. **Product Owner**: Defines requirements and priorities
3. **End Users**: Consumers of the conversation and model exploration capabilities
4. **Administrators**: Users of the administrative features
5. **Operations Team**: Responsible for deployment and maintenance
6. **Security Team**: Ensures compliance with security standards

Technical Stack

- **.NET 9.0**: Modern .NET platform for building all services
- **ASP.NET Core**: Web API framework for RESTful endpoints and streaming responses
- **Entity Framework Core**: ORM for database operations through the shared Ollama_DB_layer
- **SQL Server**: Primary relational database for data persistence (db19911.public.databaseasp.net)
- **Redis**: Distributed caching for performance optimization using Upstash (content-ghoul-42217.upstash.io)
- **Ocelot**: API Gateway implementation for request routing and load balancing
- **OllamaSharp**: Client library for interacting with Ollama AI models
- **Semantic Kernel**: Microsoft's framework for AI chat completion capabilities
- **FluentValidation**: Request validation framework for input validation
- **Swagger/OpenAPI**: API documentation and interactive testing
- **Pinecone**: Vector database for RAG implementation
- **JWT Authentication**: Token-based security with comprehensive validation
- **Serilog**: Structured logging framework
- **RabbitMQ**: Message broker for service discovery (ConversationService)

Integration Points

- **Ollama API**: Integration with the Ollama inference engine via ngrok endpoint
- **Redis Cache**: Upstash-hosted Redis for distributed caching
- **SQL Server**: Hosted database for persistent storage
- **Frontend Application**: Web UI consuming the microservices APIs via Gateway
- **Pinecone**: Vector database for semantic search capabilities
- **RabbitMQ**: Message broker for service discovery and configuration updates
- **File Storage**: For document storage and processing

Constraints

- Must maintain backward compatibility with existing routes and APIs
- Configuration changes should be tracked and reversible
- Must support both development and production environments
- Redis caching must gracefully degrade when unavailable
- Authorization must be enforced consistently across all services
- Must handle both streaming and non-streaming response patterns
- Limited integration testing across services
- Dependency on external Ollama API via ngrok during development

Risks

1. **Performance Bottlenecks**: High traffic could impact response times, mitigated by caching and optimization
2. **Cache Consistency**: Data changes might not immediately reflect in cached results, addressed by strategic invalidation
3. **External Dependencies**: Reliance on Ollama API availability, mitigated by timeout handling and fallbacks
4. **Security Vulnerabilities**: Potential authentication weaknesses, addressed by comprehensive validation
5. **Scaling Challenges**: Potential issues with high concurrent user loads, addressed by horizontal scaling design
6. **Data Consistency**: Shared database approach could lead to schema conflicts, mitigated by clear ownership boundaries
7. **Integration Complexity**: Multiple services increase integration testing complexity, addressed by comprehensive API testing
8. **Technical Debt**: Rapid development could introduce technical debt, addressed by regular refactoring
9. **Documentation Gaps**: Incomplete documentation could hinder maintenance, addressed by documentation requirements
10. **Deployment Complexity**: Microservices increase deployment complexity, addressed by containerization and orchestration