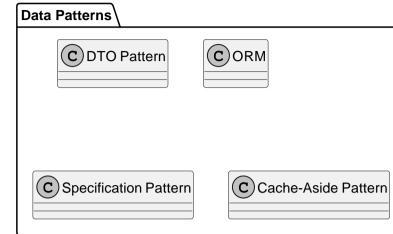
OllamaNet Documentation

Design Patterns in OllamaNet

Architectural Patterns C Unit of Work Pattern Used for resilient communication b © Microservices Architecture C CQRS Pattern C Circuit Breaker Pattern Core architectural pattern with external services and for entire OllamaNet platform RabbitMQ **Manages** Uses for commands Uses for queries Often employs © Repository Pattern © API Gateway Pattern © Event-Driven Architecture ◀ Used across all services Implemented with Ocelot Form of in the API Gateway service for data access abstraction Behavioral Patterns Creational Patterns Structural Patterns C Template Method © Builder Pattern © Dependency Injection C Strategy Pattern C Observer Pattern © Singleton © Factory Method © Proxy Pattern © Facade Pattern © Decorator Pattern C Adapter Pattern Used throughout all .NET services L Used for service discovery via RabbitMQ pub/sub via built-in DI container Implementation Examples Observer Pattern Gateway Pattern Circuit Breaker Dependency Injection Repository Pattern RabbitMQ for service discovery © Mediator Pattern C Chain of Responsibility .NET built-in DI container Entity Framework repositories - Ocelot API Gateway - Polly integration - Event-based configuration - RabbitMQ resilience - Service registration - Service-specific data access Request routing - Lifetime management Cross-cutting concerns HTTP client resilience - Query encapsulation Message subscribers InferenceService (publisher) ConversationService, L All .NET services All services with data access Primary: API Gateway AdminService,

ConversationService (subscribers)



Key design patterns used throughout the OllamaNet platform, organized by category with implementation examples.

Each pattern addresses specific architectural concerns:
- Microservices: Service boundaries and independence
- Gateway: Centralized access and cross-cutting concerns

- Repository: Data access abstraction

Circuit Breaker: Resilience and fault tolerance
 Observer: Event-driven communication

Observer: Event-driven communicationDependency Injection: Loose coupling and testability

Lines indicate relationships between patterns.

Notes indicate primary services implementing each pattern.

AdminService