Explore Service

Overview

ExploreService is a microservice component of the OllamaNet system that provides exploration capabilities for AI models. It allows users to browse, search, and get detailed information about available AI models and their associated tags. The service is designed with performance in mind, implementing sophisticated caching strategies and resilience patterns to ensure efficient model discovery.

Core Functionality

Model Discovery

- Retrieve paginated lists of available AI models
- Get detailed information about specific models
- Filter models by various criteria
- Search models by name, description, or tags
- Sort models by different attributes

Tag Browsing

- Browse all available tags
- Find models associated with specific tags
- Get tag details and metadata
- Discover related tags

Caching & Performance

- Efficient caching of frequently accessed model data
- Resilient cache operations with fallback strategies
- Performance-optimized queries for model discovery
- Resource management for efficient operation

Architecture

Layered Architecture

ExploreService follows a clean, layered architecture pattern:

- **API Layer**: ExploreController handling HTTP requests and responses
- **Service Layer**: IExploreService implementation for business logic
- **Data Access Layer**: Repository pattern for data operations
- **Caching Layer**: Sophisticated Redis-backed caching with resilience patterns

Key Components

ExploreController

- Handles all HTTP requests for model exploration
- Implements RESTful API endpoints
- Manages authentication and authorization
- Handles error responses and status codes

ExploreService

- Implements business logic for model discovery
- Coordinates between repositories and cache
- Handles model filtering and sorting
- Implements search functionality

CacheManager

- Provides high-level caching abstraction
- Implements cache-aside pattern with database fallback
- Handles cache failures with circuit breaker pattern
- Manages TTL for different data types

RedisCacheService

- Low-level Redis operations with error handling
- Implements retry logic with exponential backoff
- Handles connection failures and timeouts
- Serialization and deserialization of cached data

Repositories

- IAIModelRepository for model data access
- ITagRepository for tag data access
- Query specifications for efficient data retrieval
- Coordination via IUnitOfWork

API Design

RESTful Endpoints

Model Exploration

- **GET /api/Explore/Models** - Get paginated list of models
  - Query parameters: page, pageSize, sortBy, sortDirection
  - Returns: Paginated list of model summaries
- **GET /api/Explore/Models/{id}** - Get model details by ID
  - Returns: Detailed model information
- **GET /api/Explore/Models/Search** - Search models by criteria
  - Query parameters: query, tags, page, pageSize
  - Returns: Paginated list of matching models

Tag Exploration

- **GET /api/Explore/Tags** - Get all available tags
  - Returns: List of all tags
- **GET /api/Explore/Tags/{id}** - Get tag details by ID
  - Returns: Detailed tag information
- **GET /api/Explore/Tags/{id}/Models** - Get models for a specific tag
  - Query parameters: page, pageSize
  - Returns: Paginated list of models with the specified tag

Response Models

ModelSummaryResponse

```
{
  "id": "guid-string",
  "name": "llama2-7b",
  "description": "A 7 billion parameter language model",
  "tags": [
    {
      "id": "guid-string",
      "name": "Language Model"
    },
    {
      "id": "guid-string",
      "name": "7B Parameters"
    }
  ],
  "isInstalled": true,
  "size": "4.2 GB"
}
```

ModelDetailResponse

```
{
  "id": "guid-string",
  "name": "llama2-7b",
  "description": "A 7 billion parameter language model",
  "tags": [
    {
      "id": "guid-string",
      "name": "Language Model"
    },
    {
      "id": "guid-string",
      "name": "7B Parameters"
    }
  ],
```

```
  "isInstalled": true,
  "size": "4.2 GB",
  "parameters": "7 billion",
  "family": "Llama",
  "version": "2.0",
  "license": "Meta License",
  "quantization": "Q4_K_M",
  "contextLength": 4096,
  "dateAdded": "2023-06-15T10:30:45Z",
  "lastUsed": "2023-06-20T14:22:33Z"
}
```

TagResponse

```
{
  "id": "guid-string",
  "name": "Language Model",
  "description": "General purpose language models",
  "modelCount": 12
}
```

PaginatedResponse

```
{
  "items": [...],
  "page": 1,
  "pageSize": 10,
  "totalItems": 42,
  "totalPages": 5
}
```

Caching Strategy

Two-Tier Caching Architecture

The service implements a sophisticated two-tier caching approach:

1. **High-Level CacheManager**:
    - Provides GetOrSetAsync method with database fallback
    - Handles cache failures gracefully
    - Implements circuit breaker pattern
    - Manages domain-specific TTL values
2. **Low-Level RedisCacheService**:
    - Direct Redis operations with error handling

- Retry logic with exponential backoff
- Timeout handling with configurable thresholds
- Serialization error handling

Cache Keys and TTL Values

- **AllModels**: List of all models (15 min TTL)
- **ModelDetail:{id}**: Detailed model information (30 min TTL)
- **ModelsByTag:{tagId}**: Models for a specific tag (20 min TTL)
- **AllTags**: List of all tags (60 min TTL)
- **TagDetail:{id}**: Detailed tag information (60 min TTL)

Cache Implementation

```
public async Task<T> GetOrSetAsync<T>(string key, Func<Task<T>> factory, TimeSpan
expiration)
{
    try
    {
        // Try to get from cache first
        var cachedValue = await _redisCacheService.GetAsync<T>(key);
        if (cachedValue != null)
        {
            _logger.LogDebug("Cache hit for key: {Key}", key);
            return cachedValue;
        }

        _logger.LogDebug("Cache miss for key: {Key}", key);

        // If not in cache, get from data source
        var result = await factory();

        // Store in cache
        await _redisCacheService.SetAsync(key, result, expiration);

        return result;
    }
    catch (CacheException ex)
    {
        // Log the cache failure but continue with data source
        _logger.LogWarning(ex, "Cache operation failed for key: {Key}", key);
        return await factory();
    }
}
```

Circuit Breaker Pattern

```
public async Task<T> GetAsync<T>(string key)
{
```

```
    if (_circuitBreaker.IsOpen)
    {
        _logger.LogWarning("Circuit is open, skipping cache operation for key:
{Key}", key);
        throw new CircuitOpenException($"Cache circuit is open for key: {key}");
    }

    try
    {
        var result = await _redisCache.GetStringAsync(key);

        if (result == null)
        {
            return default;
        }

        _circuitBreaker.Success();
        return JsonSerializer.Deserialize<T>(result);
    }
    catch (Exception ex)
    {
        _circuitBreaker.Failure();
        throw new CacheException($"Failed to get value from cache for key: {key}",
ex);
    }
}
```

Exception Handling

Exception Hierarchy

- **ExploreServiceException**: Base exception for all service exceptions
- **ModelNotFoundException**: Thrown when a requested model is not found
- **TagNotFoundException**: Thrown when a requested tag is not found
- **DataRetrievalException**: Thrown when data access fails
- **CacheException**: Base exception for cache-related issues
  - **ConnectionFailureException**: Redis connection issues
  - **TimeoutException**: Redis operation timeout
  - **SerializationException**: JSON serialization/deserialization issues
  - **CircuitOpenException**: Circuit breaker is open

Error Responses

```
{
  "status": 404,
  "message": "Model not found",
  "details": "The model with ID '12345' does not exist",
  "timestamp": "2023-06-15T10:30:45Z",
  "path": "/api/Explore/Models/12345"
}
```

Configuration Management

RedisCacheSettings

```
"RedisCacheSettings": {
  "ConnectionString": "content-ghoul-
42217.upstash.io:42217,password=xxx,ssl=True,abortConnect=False",
  "DefaultTTLMinutes": 15,
  "AllModelsTTLMinutes": 15,
  "ModelDetailTTLMinutes": 30,
  "ModelsByTagTTLMinutes": 20,
  "AllTagsTTLMinutes": 60,
  "TagDetailTTLMinutes": 60,
  "MaxRetryAttempts": 3,
  "RetryDelayMilliseconds": 100,
  "RetryDelayMultiplier": 2,
  "OperationTimeoutMilliseconds": 1000,
  "CircuitBreakerThreshold": 5,
  "CircuitBreakerResetTimeSeconds": 30
}
```

ExploreServiceSettings

```
"ExploreServiceSettings": {
  "DefaultPageSize": 10,
  "MaxPageSize": 100,
  "DefaultSortField": "Name",
  "DefaultSortDirection": "Ascending",
  "EnableSearchHighlighting": true,
  "MaxSearchResults": 50
}
```

Integration Points

Database Layer

- Entity Framework Core for data access
- Repository pattern for data operations
- Shared Ollama_DB_layer with other services

Redis Cache

- Distributed caching for performance optimization
- Resilient operations with fallback strategies
- Domain-specific TTL configurations

Authentication System

- JWT token validation for secure access
- Role-based authorization for endpoints
- User identity for personalized results (future)

Performance Optimization

Efficient Data Retrieval

- Optimized database queries for model listing
- Pagination to limit result set size
- Projection queries to retrieve only needed fields
- Eager loading of related entities where appropriate

Strategic Caching

- Frequently accessed data cached with appropriate TTL
- Cache invalidation on data changes (via AdminService)
- Two-tier caching architecture with fallback
- Circuit breaker to prevent cascade failures

Resource Management

- Connection pooling for database access
- Proper disposal of resources
- Asynchronous operations for non-blocking I/O
- Timeout handling for external dependencies

Known Issues

- Advanced filtering capabilities limited
- No usage statistics for models
- Cache invalidation strategy needs improvement
- Search functionality is basic and could be enhanced

Future Enhancements

- Enhanced search with full-text capabilities
- User-specific model recommendations
- Usage statistics and popularity metrics
- Model comparison features
- Advanced filtering and faceted search
- Integration with vector search for semantic similarity