DataBridge Project Overview

Project Description

DataBridge is a .NET 8.0 web application designed to serve as a robust API for managing and integrating data from various sources, focusing on product information management (PIM) and content services. The project incorporates several modern web development practices and technologies to ensure scalability, reliability, and maintainability.

Key Features

1. **API Versioning**: Implements API versioning to manage different versions of the API endpoints.

2. **Authentication and Authorization**: Uses JWT (JSON Web Token) for secure authentication and authorization.

3. **Database Integration**:

    - Utilizes Entity Framework Core with SQL Server as the primary database.
    - Includes support for PostgreSQL as an alternative database option.

4. **Health Checks**: Implements comprehensive health checks for the API, Delivra API, and Contentserv API.

5. **Background Job Processing**: Integrates Hangfire for scheduling and processing background jobs.

6. **Data Import/Export**:

    - Supports importing data from Excel and CSV files.
    - Provides functionality to export data in JSON, CSV, and Excel formats.

7. **Third-party Integrations**:

    - Integrates with Delivra API for data retrieval.
    - Integrates with Contentserv API for product information management.

8. **CORS Support**: Configures Cross-Origin Resource Sharing (CORS) for specified origins.

9. **Swagger Documentation**: Implements Swagger UI for API documentation and testing.

10. **Resilience and Retry Policies**: Uses Polly for implementing retry policies and resilience patterns.

11. **Logging**: Configures logging using console, debug, and event source loggers.

12. **Exception Handling**: Implements global exception handling for consistent error responses.

13. **File Uploads**: Configures the application to handle large file uploads.

14. **Google Sheets Integration**: Includes packages for potential integration with Google Sheets API.

15. **Email Functionality**: Incorporates SendGrid for email sending capabilities.

16. **Bulk Data Operations**: Uses EFCore.BulkExtensions for efficient bulk database operations.

17. **Caching**: Implements in-memory caching for improved performance.

18. **AutoMapper**: Utilizes AutoMapper for object-to-object mapping.

19. Data integration with Delivra, LivePerson, Google Sheets and other services.

20. API connections to various data sources.

21. Entity Framework Core for ORM.

22. PostgreSQL and SQL Server support.

23. Email functionality with SendGrid.

24. Resilience and transient-fault-handling policies with Polly.

25. Global exception handling and logging.

26. Swagger API documentation with Swashbuckle.

27. Containerization support with Docker.

Technical Stack

- **Framework**: .NET 8.0

- **API Development**: ASP.NET Core
- **Database ORM**: Entity Framework Core
- **Databases**: SQL Server, PostgreSQL
- **Authentication**: JWT (JSON Web Tokens)
- **Background Jobs**: Hangfire
- **API Documentation**: Swagger / OpenAPI
- **Dependency Injection**: Built-in .NET Core DI container
- **Logging**: .NET Core logging with multiple providers
- **Health Monitoring**: AspNetCore.HealthChecks
- **HTTP Resilience**: Polly, Microsoft.Extensions.Http.Resilience
- **Object Mapping**: AutoMapper
- **File Handling**: EPPlus (for Excel), CsvHelper
- **External Services**: SendGrid (for emails), Google Sheets API

Project Structure

The project follows a typical ASP.NET Core Web API structure with the following key components:

- **Controllers**: Handle incoming HTTP requests and return responses.
- **Services**: Contain business logic and interact with data sources.
- **Models**: Represent data structures used throughout the application.
- **Data**: Includes database context and related configurations.
- **Options**: Contain configuration options for various parts of the application.
- **Helpers**: Utility classes and methods.

Configuration and Environment

- Supports different environments (Development, Production) with environment-specific configurations.
- Uses appsettings.json for configuration management.
- Dockerized for easy deployment and scalability.

Security Considerations

- Implements JWT authentication for API security.
- Uses HTTPS redirection for secure communication.
- Configures CORS to control access from different origins.

Scalability and Performance

- Uses asynchronous programming patterns for improved performance.
- Implements caching strategies to reduce database load.
- Utilizes Hangfire for efficient background job processing.

- Employs bulk operations for handling large datasets.

Monitoring and Maintenance

- Implements health checks for monitoring system health.
- Uses Hangfire dashboard for monitoring and managing background jobs.
- Configures comprehensive logging for troubleshooting and monitoring.

Conclusion

DataBridge is a .NET 8.0 web application designed to serve as a robust API for managing and integrating data from various sources, focusing on product information management (PIM), data from MarCom platforms, and content services. The project incorporates several modern web development practices and technologies to ensure scalability, reliability, and maintainability.

Delivra API Integration Documentation

Overview

This document provides an overview and usage guide for the Delivra API integration implemented in the DelivraController. The controller facilitates various operations related to Delivra, including handling reports, segments, clickthroughs, mailing approvals, opens, and sends.

Prerequisites

- .NET 8.0 SDK
- Docker (optional, for containerization support)

Getting Started

Clone the repository and navigate to the project directory:

```
git clone https://dev.azure.com/ea-dxp/_git/DataBridge
```

```
cd DataBridge
```

Installation

Restore the .NET project dependencies:

```
dotnet restore
```

Controller Structure

The DelivraController is an API controller that handles Delivra-related operations. It is:

- Authorized
- Versioned (v1.0)
- Produces and consumes JSON

Endpoints

Base

GET api/v1.0/Delivra/{query}

- Description: Executes a direct query to the Delivra API without storing data in the database.
- Parameters: query (string) - The query to pass to the Delivra API.
- Returns: The raw result from the Delivra API.

Reports

GET api/v1.0/Delivra/Reports

- Description: Retrieves a list of reports from the database.
- Returns: A list of ReportDto objects.

POST api/v1.0/Delivra/Reports

- Description: Retrieves reports for a given date range from the Delivra API.
- Parameters: dateRange (DateRangeDto) - The start and end dates for the report query.
- Returns: A list of ReportDto objects.

PUT api/v1.0/Delivra/Reports

- Description: Updates the database with new reports for a given date range, adding only reports that don't already exist.
- Parameters: dateRange (DateRangeDto) - The start and end dates for the report query.
- Returns: A list of added ReportDto objects.

DELETE api/v1.0/Delivra/Reports

- Description: Deletes all reports from the database.

Segments

GET api/v1.0/Delivra/Segments

- Description: Retrieves a list of segments from the database.
- Returns: A list of SegmentDto objects.

POST api/v1.0/Delivra/Segments

- Description: Retrieves segments from the Delivra API.
- Returns: A list of SegmentDto objects.

PUT api/v1.0/Delivra/Segments

- Description: Updates the database with new segments, adding only segments that don't already exist.
- Returns: A list of added SegmentDto objects.

DELETE api/v1.0/Delivra/Segments

- Description: Deletes all segments from the database.

Clickthroughs

GET api/v1.0/Delivra/Clickthroughs

- Description: Retrieves a list of clickthroughs from the database.
- Returns: A list of ClickthroughDto objects.

POST api/v1.0/Delivra/Clickthroughs

- Description: Retrieves clickthroughs for a given date range from the Delivra API.
- Parameters: dateRange (DateRangeDto) - The start and end dates for the clickthrough query.
- Returns: A list of ClickthroughDto objects.

PUT api/v1.0/Delivra/Clickthroughs

- Description: Updates the database with new clickthroughs for a given date range, adding only clickthroughs that don't already exist.
- Parameters: dateRange (DateRangeDto) - The start and end dates for the clickthrough query.
- Returns: A list of added ClickthroughDto objects.

DELETE api/v1.0/Delivra/Clickthroughs

- Description: Deletes all clickthroughs from the database.

Mailing Approvals

GET api/v1.0/Delivra/MailingApprovals

- Description: Retrieves a list of mailing approvals from the database.
- Returns: A list of MailingApprovalDto objects.

POST api/v1.0/Delivra/MailingApprovals

- Description: Retrieves mailing approvals from the Delivra API.
- Returns: A list of MailingApprovalDto objects.

PUT api/v1.0/Delivra/MailingApprovals

- Description: Updates the database with new mailing approvals, adding only approvals that don't already exist.
- Returns: A list of added MailingApprovalDto objects.

DELETE api/v1.0/Delivra/MailingApprovals

- Description: Deletes all mailing approvals from the database.

Opens

GET api/v1.0/Delivra/Opens

- Description: Retrieves all report opens from the database.
- Returns: A list of OpenDto objects.

POST api/v1.0/Delivra/Opens

- Description: Retrieves report opens for a given date range from the Delivra API and stores them in the database.
- Parameters: dateRange (DateRangeDto) - The start and end dates for the opens query.
- Returns: A list of OpenDto objects.

PUT api/v1.0/Delivra/Opens

- Description: Updates the database with new report opens for a given date range, adding only opens that don't already exist.
- Parameters: dateRange (DateRangeDto) - The start and end dates for the opens query.
- Returns: A list of added OpenDto objects.

DELETE api/v1.0/Delivra/Opens

- Description: Deletes all report opens from the database.

Sends

GET api/v1.0/Delivra/Sends

- Description: Retrieves all report sends from the database.
- Returns: A list of SendDto objects.

POST api/v1.0/Delivra/Sends

- Description: Retrieves report sends for a given date range from the Delivra API and stores them in the database.
- Parameters: dateRange (DateRangeDto) - The start and end dates for the sends query.
- Returns: A list of SendDto objects.

PUT api/v1.0/Delivra/Sends

- Description: Updates the database with new report sends for a given date range, adding only sends that don't already exist.
- Parameters: dateRange (DateRangeDto) - The start and end dates for the sends query.
- Returns: A list of added SendDto objects.

DELETE api/v1.0/Delivra/Sends

- Description: Deletes all report sends from the database.

Usage Guidelines

1. Authentication: All endpoints require authorization. Ensure you have the necessary authentication token before making requests.

2. Date Ranges: When using endpoints that accept a date range, provide the startDate and endDate in the format yyyy-MM-dd.

3. Error Handling: The controller uses standard HTTP status codes. Check the response status to handle errors appropriately.

4. Data Synchronization: Use the PUT endpoints to synchronize data between the Delivra API and your local database. These endpoints will only add new data, avoiding duplication.

5. Data Cleanup: Use the DELETE endpoints with caution, as they will remove all data of the specified type from your local database.

6. Direct API Access: For custom queries not covered by the provided endpoints, use the base GET endpoint with your specific query string.

Best Practices

1. Regularly synchronize data using the PUT endpoints to keep your local database up-to-date with the Delivra API.

2. Use appropriate date ranges when querying data to manage the amount of data transferred and processed.

3. Implement proper error handling in your client application to manage potential issues with API requests.

4. Consider implementing a caching mechanism for frequently accessed data to reduce API calls and improve performance.

5. Monitor the performance of your database, especially when dealing with large datasets from the Delivra API.

Conclusion

This Delivra API integration provides a comprehensive set of endpoints to manage various aspects of your Delivra data. By following the usage guidelines and best practices, you can effectively work with Delivra data in your application.

ContentservController Documentation

Overview

The ContentservController is an API controller responsible for handling Contentserv PIM (Product Information Management) related operations. It provides endpoints for retrieving user data, product data, product hierarchies, and importing/exporting product data in various formats.

Controller Structure

The ContentservController is an API controller with the following characteristics:

- Authorized
- Versioned (v1.0)
- Base route: api/v1.0/Contentserv

Dependencies

The controller depends on several services and components:

1. AppDbContext: For database operations
2. HttpClient: For making HTTP requests
3. ILogger<ContentservController>: For logging
4. IMapper: For object mapping
5. BackendOptions and ContentservOptions: For configuration
6. ContentservTokenService: For managing Contentserv tokens

Endpoints

GET api/v1.0/Contentserv/data

- Description: Retrieves user data from the Contentserv API.
- Response:
    - Status: 200 OK
    - Body: User data in JSON format

GET api/v1.0/Contentserv/product/{id}

- Description: Retrieves product data by ID from the Contentserv API.
- Parameters:
    - id (int): The ID of the product to retrieve.
- Response:
    - Status: 200 OK
    - Body: Product data in JSON format

POST api/v1.0/Contentserv/hierarchy

- Description: Retrieves the product hierarchy tree starting from a specified ID.
- Response:
    - Status: 200 OK
    - Body: Product hierarchy data

POST api/v1.0/Contentserv/import

- Description: Imports data from an uploaded file (Excel or CSV) and saves it to the database.
- Request:
    - Body: Form data with file upload
- Response:
    - Status: 200 OK
    - Body: Import results (number of added, updated, and total processed records)

GET api/v1.0/Contentserv/products

- Description: Retrieves all products from the database.

- Response:
    - Status: 200 OK
    - Body: List of all products

GET api/v1.0/Contentserv/export

- Description: Exports products in the specified format (JSON, CSV, or XLSX).
- Parameters:
    - format (string, optional): The export format. Defaults to "json".
- Response:
    - Status: 200 OK
    - Body: File content in the specified format

GET api/v1.0/Contentserv/export-json

- Description: Exports all products as a JSON file.
- Response:
    - Status: 200 OK
    - Body: JSON file containing all products

GET api/v1.0/Contentserv/export-csv

- Description: Exports all products as a CSV file.
- Response:
    - Status: 200 OK
    - Body: CSV file containing all products

GET api/v1.0/Contentserv/export-xlsx

- Description: Exports all products as an XLSX (Excel) file.
- Response:
    - Status: 200 OK
    - Body: XLSX file containing all products

DELETE api/v1.0/Contentserv/delete-products

- Description: Deletes all products from the database.
- Response:
    - Status: 200 OK
    - Body: Message indicating the number of deleted records

Key Features

1. **Product Hierarchy**: The controller can build and retrieve product hierarchies, maintaining parent-child relationships between products.

2. **File Import**: Supports importing product data from Excel files. The import process is optimized for performance using parallel processing and thread-safe collections.

3. **File Export**: Provides multiple export formats (JSON, CSV, XLSX) for product data. The XLSX export includes basic styling and auto-fitted columns.

4. **API Integration**: Integrates with the Contentserv API, handling authentication and token management automatically.

5. **Bulk Database Operations**: Uses EF Core's bulk extensions for efficient database operations when dealing with large datasets.

Best Practices and Optimizations

1. **Parallel Processing**: The Excel import method uses Parallel.For to process rows concurrently, improving performance for large files.

2. **Thread-Safe Collections**: Uses ConcurrentBag<T> for thread-safe collection of products during parallel processing.

3. **Bulk Database Operations**: Employs BulkInsertOrUpdateAsync for efficient database updates with large datasets.

4. **Memory Efficiency**: Uses AsNoTracking and ProjectTo in some queries to reduce memory usage and improve query performance.

5. **Error Handling**: Implements comprehensive error handling and logging throughout the controller.

6. **API Authentication**: Uses a token service to handle API authentication, automatically refreshing tokens as needed.

Usage Guidelines

1. **File Import**: When importing files, ensure they are in Excel format (.xlsx or .xls). The first row should contain headers that match the property names of the Product class.

2. **Export Formats**: Choose the appropriate export format based on your needs. JSON is suitable for data interchange, CSV for simple tabular data, and XLSX for formatted spreadsheets.

3. **Product Hierarchy**: The BuildProductHierarchy endpoint can be used to retrieve and store the complete product hierarchy. This operation may take time for large hierarchies.

4. **Bulk Operations**: Be cautious when using delete operations, as they affect all products in the database.

5. **API Calls**: Direct API calls (e.g., GetData, GetProductById) fetch real-time data from Contentserv and do not interact with the local database.

Error Handling

The controller implements error handling for various scenarios:

- Invalid file formats during import
- API call failures
- Database operation errors
- Data parsing and mapping errors

Errors are logged using the injected ILogger instance. In most cases, appropriate HTTP status codes and error messages are returned to the client.

Conclusion

The ContentservController provides a comprehensive set of endpoints to manage product data in integration with the Contentserv PIM system. It offers efficient data import and export capabilities, along with direct API access to Contentserv data. By following these usage guidelines and leveraging the provided optimizations, you can effectively manage large sets of product data in your application.

AuthController Documentation

Overview

The AuthController is responsible for handling JWT (JSON Web Token) operations in the DataBridge API. It provides endpoints for generating, validating, retrieving, and managing JWT tokens.

Controller Structure

The AuthController is an API controller with the following characteristics:

- Versioned (v1.0)
- Base route: api/v1.0/Auth/Token

Dependencies

The controller depends on two services:

1. JwtTokenGenerator: Responsible for generating new JWT tokens.
2. JwtTokenProvider: Handles token storage, validation, and management.

Endpoints

POST api/v1.0/Auth/Token/Set

- Description: Generates a new JWT token and stores it.
- Request: No parameters required.
- Response:
    - Status: 200 OK
    - Body: The generated JWT token.

POST api/v1.0/Auth/Token/Validate/{token}

- Description: Validates a given JWT token and sets it if valid.
- Parameters:
    - token (string): The JWT token to validate.
- Response:
    - If valid:
        - Status: 200 OK
    - If invalid:
        - Status: 404 Not Found
        - Body: "Invalid token"

GET api/v1.0/Auth/Token/Get

- Description: Retrieves the stored JWT token.
- Request: No parameters required.
- Response:
    - Status: 200 OK

DELETE api/v1.0/Auth/Token/Clear

- Description: Clears the stored JWT token.
- Request: No parameters required.
- Response:
    - Status: 200 OK
    - Body: "Token cleared"

POST api/v1.0/Auth/Token/Invalidate

- Description: Invalidates a given JWT token by manipulating its expiration time.
- Parameters:
    - token (string): The JWT token to invalidate.
- Response:
    - If token is provided:
        - Status: 200 OK
        - Body: "Token invalidated successfully"
    - If token is not provided:
        - Status: 400 Bad Request
        - Body: "Token is required"

Usage Guidelines

1. Token Generation: Use the Set endpoint to generate and store a new JWT token.

2. Token Validation: Before using a token, validate it using the Validate endpoint. This will also set the token if it's valid.

3. Token Retrieval: Use the Get endpoint to retrieve the currently stored token.

4. Token Management:

    - Use the Clear endpoint to remove the stored token when it's no longer needed.
    - Use the Invalidate endpoint to forcefully invalidate a token before its natural expiration.

5. Error Handling:

    - Check for 404 Not Found responses when validating tokens, which indicate an invalid token.
    - Handle 400 Bad Request responses when trying to invalidate a token without providing one.

Best Practices

1. Token Security: Always transmit tokens over secure, encrypted connections (HTTPS).

2. Token Lifecycle: Implement proper token lifecycle management:

    - Generate new tokens when needed (e.g., at user login).
    - Validate tokens before using them for authentication.
    - Clear or invalidate tokens when they're no longer needed (e.g., at user logout).

3. Error Handling: Implement proper error handling in your client application to manage potential issues with token operations.

4. Token Storage: In client applications, store tokens securely (e.g., in HTTP-only cookies or secure local storage).

5. Regular Validation: Regularly validate tokens to ensure they haven't been invalidated or expired.

Conclusion

The AuthController provides a comprehensive set of endpoints to manage JWT tokens in your application. By following these usage guidelines and best practices, you can implement secure and efficient token-based authentication in your DataBridge API integration.

Project 'DataBridge' has the following package references

Top-level Package	**Requested**	**Resolved**
Asp.Versioning.Http	8.1.0	8.1.0
Asp.Versioning.Mvc.ApiExplorer	8.1.0	8.1.0
AspNetCore.HealthChecks.NpgSql	8.0.1	8.0.1
AspNetCore.HealthChecks.SqlServer	8.0.2	8.0.2
AspNetCore.HealthChecks.UI	8.0.1	8.0.1
AspNetCore.HealthChecks.UI.Client	8.0.1	8.0.1
AspNetCore.HealthChecks.UI.InMemory.Storage	8.0.1	8.0.1
AutoMapper	13.0.1	13.0.1
Google.Apis.Auth	1.68.0	1.68.0
Google.Apis.Sheets.v4	1.68.0.3421	1.68.0.3421
Microsoft.AspNetCore.Authentication.JwtBearer	8.0.6	8.0.6

Top-level Package	Requested	Resolved
Microsoft.AspNetCore.OpenApi	8.0.6	8.0.6
Microsoft.EntityFrameworkCore	8.0.6	8.0.6
Microsoft.EntityFrameworkCore.Design	8.0.6	8.0.6
Microsoft.EntityFrameworkCore.SqlServer	8.0.6	8.0.6
Microsoft.EntityFrameworkCore.Tools	8.0.6	8.0.6
MongoDB.Entities	23.1.0	23.1.0
Npgsql.EntityFrameworkCore.PostgreSQL	8.0.4	8.0.4
Polly	8.4.0	8.4.0
SendGrid	9.29.3	9.29.3
Swashbuckle.AspNetCore	6.6.2	6.6.2
Swashbuckle.AspNetCore.Annotations	6.6.2	6.6.2

Project 'UnitTests' has the following package references

Top-level Package	Requested	Resolved
coverlet.collector	6.0.0	6.0.0
Microsoft.NET.Test.Sdk	17.6.0	17.6.0
xunit	2.4.2	2.4.2
xunit.runner.visualstudio	2.4.5	2.4.5

Project 'IntegrationTests' has the following package references

Top-level Package	**Requested**	**Resolved**
coverlet.collector	6.0.0	6.0.0
Microsoft.NET.Test.Sdk	17.6.0	17.6.0
xunit	2.4.2	2.4.2
xunit.runner.visualstudio	2.4.5	2.4.5