

Figure 1: Backend API Data Flow Diagram with Cache Management

**Explanation:**

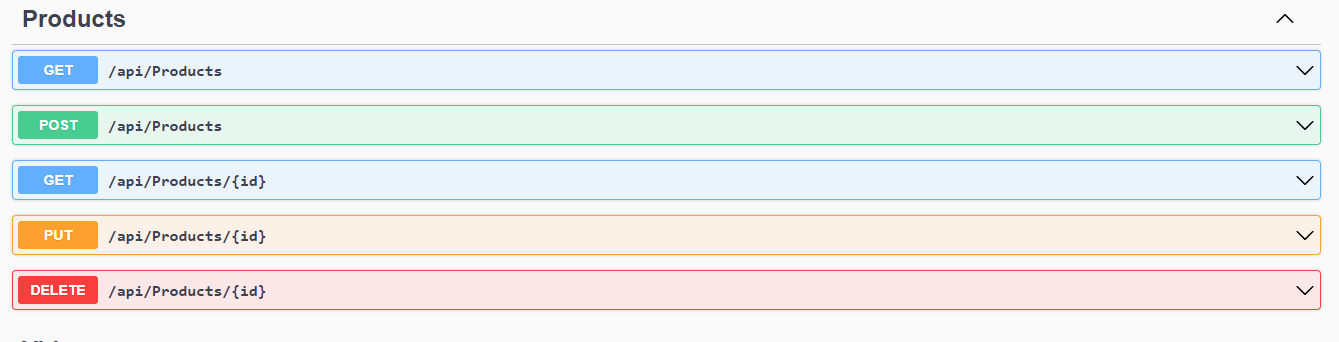
1. **Cache Management**:
   * Represents the logic for managing data in the cache, including storing and retrieving cached data. This may include operations such as setting cache keys, fetching cached data, and updating cached data.
2. **Cache Refresh Service**:
   * Represents the **CacheRefreshService** that periodically updates the cache with fresh data from the database. This service runs in the background at regular intervals to ensure that cached data remains up-to-date.
3. **Users :**
   * Represents external users interacting with the backend API. These users may include administrators, clients, or other systems consuming the API.

**Data Flow:**

1. **Client Request**:
   * Users send HTTP requests to the backend API, triggering the corresponding API endpoints.
2. **API Endpoints**:
   * The API endpoints receive and process the requests, invoking the appropriate business logic to perform the requested operations.
3. **Business Logic Layer**:
   * The business logic layer contains the application-specific logic, including controllers and services. It coordinates data processing and interacts with the cache and data access layer.
4. **Data Access Layer**:
   * The data access layer interacts with both the MySQL database and the cache. It performs CRUD operations on the database and may utilize the cache to improve performance.
5. **MySQL Database**:
   * The MySQL database stores persistent data and is accessed by the data access layer to retrieve and modify records.
6. **Cache Management**:
   * Handles caching operations, such as storing and retrieving data from the cache. It ensures that frequently accessed data is stored in the cache for faster access.
7. **Cache Refresh Service**:
   * The **CacheRefreshService** periodically updates the cache with fresh data from the database. It ensures that cached data remains up-to-date by refreshing it at regular intervals.

By incorporating cache management and the cache refresh service into the data flow, the backend API application can efficiently leverage caching to improve performance and responsiveness.

Implemented Controller:



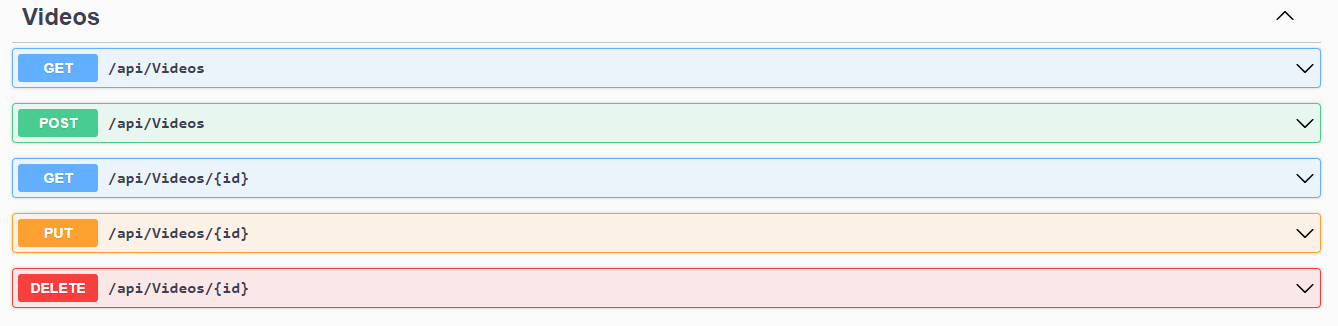


Figure 2: Implemented Controllers Products & Videos

**Products Controller:**

The Products controller manages CRUD (Create, Read, Update, Delete) operations for product entities in the backend API. Here's a breakdown of its functionality:

1. **GET Method**:
   * The **GetProducts** method returns a list of all products available in the system.
   * The **GetProduct** method retrieves a specific product by its ID.
2. **POST Method**:
   * The **PostProduct** method creates a new product based on the provided data in the request body.
   * It adds the product to the database and returns the newly created product with its assigned ID.
3. **PUT Method**:
   * The **PutProduct** method updates an existing product with the specified ID.
   * It modifies the product's attributes based on the data provided in the request body.
4. **DELETE Method**:
   * The **DeleteProduct** method removes a product from the system based on its ID.
5. **Error Handling and Validation**:
   * Proper error handling techniques are implemented to handle scenarios such as invalid requests or database errors.
   * Validation techniques ensure that incoming data meets certain criteria before processing.

**Videos Controller:**

The Videos controller handles CRUD operations related to video files associated with products. Here's an overview of its functionality:

1. **GET Method**:
   * The **GetVideos** method retrieves a list of all video files available in the system.
   * The **GetVideo** method returns a specific video file by its ID.
2. **POST Method**:
   * The **PostVideo** method allows users to upload a new video file for a product.
   * It saves the video file to the appropriate storage location and associates it with the specified product.
3. **PUT Method**:
   * The **PutVideo** method updates the metadata or content of an existing video file.
   * It modifies the video file's attributes based on the data provided in the request body.
4. **DELETE Method**:
   * The **DeleteVideo** method removes a video file from the system based on its ID.
   * It deletes the video file from storage and disassociates it from any products.
5. **Error Handling and Validation**:
   * Similar to the Products controller, the Videos controller implements error handling and validation techniques to ensure data integrity and handle exceptions gracefully.

These controllers provide a RESTful API for interacting with product and video entities, allowing clients to perform various operations such as creation, retrieval, updating, and deletion.