**Usage:**

Below is the usage documentation for each API endpoint provided by the TestAPIController:

**1. Fetching Entities**

**Endpoint:**

GET /api/TestAPI

**Description:**

This endpoint retrieves a list of entities based on specified query parameters.

**Query Parameters:**

* Search: Search string to filter entities based on various fields.
* Gender: Gender of the entity to filter.
* StartDate: Start date for filtering entities by the Date\_T field.
* EndDate: End date for filtering entities by the Date\_T field.
* Countries: List of countries to filter entities by the Country field.
* Page Number: Page number.
* PageSize: Number of entities to include per page.
* SortBy: Field to sort entities by.
* SortOrder: Sorting order (Ascending or Descending).

**Example Request:**

GET /api/TestAPI?Search=bob&Gender=Male&StartDate=2022-01-01&EndDate=2023-12-31&Countries=USA,Canada&PageNumber=1&PageSize=10&SortBy=FirstName&SortOrder=Ascending

**2. Fetching a Single Entity**

**Endpoint:**

GET /api/TestAPI/{id}

**Description:**

This endpoint retrieves a single entity by its ID.

**Path Parameters:**

* id: ID of the entity to retrieve.

**Example Request:**

GET /api/TestAPI/123

**3. Creating an Entity**

**Endpoint:**

POST /api/TestAPI

**Description:**

This endpoint creates a new entity.

**Request Body:**

* JSON object representing the entity to be created.

**Example Request:**

POST /api/TestAPI

Content-Type: application/json

{

"name": {

"FirstName": "John",

"MiddleName": "Doe",

"Surname": "Smith"

},

"gender": "Male",

"address": {

"AddressLine": "123 Main St",

"City": "Anytown",

"Country": "USA"

},

"date": {

"DateType": "Birth",

"Date\_T": "1985-06-15T00:00:00"

}

}

**4. Updating an Entity**

**Endpoint:**

PUT /api/TestAPI/{id}

**Description:**

This endpoint updates an existing entity.

**Path Parameters:**

* id: ID of the entity to update.

**Request Body:**

* JSON object representing the updated entity data.

**Example Request:**

PUT /api/TestAPI/123

Content-Type: application/json

{

"address": {

"addressLine": "string",

"city": "string",

"country": "string"

},

"date": {

"dateType": "string",

"date\_T": "2024-02-25T23:22:29.086Z"

},

"name": {

"firstName": "string",

"middleName": "string",

"surname": "string"

},

"gender": "string",

"id": 0

}

**5. Deleting an Entity**

**Endpoint:**

DELETE /api/TestAPI/{id}

**Description:**

This endpoint deletes an existing entity.

**Path Parameters:**

* id: ID of the entity to delete.

**Example Request:**

DELETE /api/TestAPI/123

**2. Directory Structure:**

root folder

│ Program.cs

│

└───Controllers

│ │ TestAPIController.cs

│

└───Models

│ │ Models.cs

│

└───Repositories

│ IEntityRepository.cs

│ MockEntityRepository.cs

All concerns are separated. Namely the API definitions, the data layer, and the models.

* Controllers: Contains controller classes responsible for handling HTTP requests and responses.
* Models: Defines data models used in the application.
* Repositories: Contains interfaces and classes for interacting with data repositories.
* Program.cs: Serves as the entry point of the application and configures services and middleware.

**3. Files Overview:**

**TestAPIController.cs:**

* Has all the endpoint definitions.
* Has retry and backoff mechanism.
* Utilizes the IEntityRepository interface to interact with the data repository. Makes use of the dependency injection feature.

**Models.cs:**

* Defines the structure of data models used in the application.
* Includes classes for Entity, Address, Date, Name, and EntityQueryParameters.
* Represents the entities stored in the database and query parameters for filtering, sorting, and pagination.

**IEntityRepository.cs:**

* Defines the contract for interacting with data repositories.
* Includes methods for retrieving entities based on query parameters and by ID.

**MockEntityRepository.cs:**

* Provides an implementation of the IEntityRepository interface using mocked data.
* Generates mock entities using the Bogus library.
* Allows for filtering, sorting, and pagination of entities based on query parameters.
* Has ability to simulate operation failures.

**Program.cs:**

* Serves as the entry point of the application.
* Sets up dependency injection for the IEntityRepository interface. Whenever an object of type IEntityRepository is requested an instance of MockEntityRepository is provided (the instance is only created once- it is shared)
* Sets up Swagger for API for convenient testing.

**4. External Dependencies**

Bogus (used to created dummy data)

**Note: the code has been explained through inline comments**