KYC360
Comply and Outperform

INTERNSHIP ASSESSMENT

SUBMITTED BY:
SANIDHYA JAIN

February 2024

INTRODUCTION

1.1 PURPOSE

This internship assessment document provides an overview of my performance in developing a REST API with endpoints serving Entity data, as well as mocking a database interaction layer for the KYC360 project. While I embarked on this task with minimal prior experience in .NET Core Web API, I approached it with enthusiasm and a willingness to learn. The document outlines the steps taken, challenges faced, solutions implemented, and reflections on the experience gained throughout the process.

1.2 OBJECTIVES

The primary objectives of this assessment were to:

1. Develop a functional REST API using C# and .NET Core Web API technology.
2. Mock a database interaction layer and generate mock data for the API endpoints.
3. Demonstrate problem-solving skills, adaptability, and the ability to learn new technologies.

Through this assessment, I aimed to showcase my ability to grasp new concepts, tackle challenges, and deliver a viable solution within the given constraints. The document will provide insights into my approach, decision-making process, and reflections on areas of strength and improvement.

SCHEMA MODEL

The schema model for the Entity data is designed to represent information about individuals, including their **addresses, dates, names, gender, and deceased status.**

Namespace: The **myrestapi** namespace encapsulates the classes and interface defined within it.

IEntity Interface

The IEntity interface defines the properties common to all entities. These properties include:

- **Addresses:** A list of addresses associated with the entity.
- **Dates:** A list of dates associated with the entity.
- **Deceased:** A boolean flag indicating whether the entity is deceased.
- **Gender:** The gender of the entity.
- **Id:** Unique identifier for the entity.
- **Names:** A list of names associated with the entity.

Entity Class: The Entity class implements the IEntity interface. This class serves as the concrete implementation of the entity data structure.

Address Class: The Address class represents a physical address and includes properties for the address line, city, and country.

Date Class: The Date class represents a date associated with the entity. It includes properties for the date type and the actual date value.

Name Class: The Name class represents a person's name and includes properties for the first name, middle name, and surname.

TASKS

CRUD ENDPOINTS

CRUD operations (Create, Read, Update, Delete) are fundamental to any API as they provide the basic functionalities to manage resources. Here's an explanation of each CRUD operation and its significance:

Creating an Entity

HTTP Method: POST
Endpoint: /api/entities
Description: This endpoint allows clients to create a new entity by providing the necessary data in the request body.
Implementation:
Generates a unique identifier for the new entity.
Adds the entity to the mock database.
Returns a response with the newly created entity and a status code of 201 (Created).

Reading Entities (Listing All Entities)

HTTP Method: GET
Endpoint: /api/entities
Description: This endpoint retrieves a list of entities from the mock database, optionally filtered based on query parameters.
Implementation:
Retrieves all entities from the mock database.
Applies optional filters such as search, deceased status, gender, start/end dates, and countries.
Returns a response with the list of filtered entities and a status code of 200 (OK).

Retrieving a Single Entity
HTTP Method: GET
Endpoint: /api/entities/{id}
Description: This endpoint retrieves a single entity by its unique identifier (ID).
Implementation:
Searches for the entity with the specified ID in the mock database.
Returns a response with the found entity if it exists, along with a status code of 200 (OK). If not found, returns a 404 (Not Found) status code.

Updating an Entity
HTTP Method: PUT
Endpoint: /api/entities/{id}
Description: This endpoint allows clients to update an existing entity by providing the updated data in the request body.
Implementation:
Searches for the entity with the specified ID in the mock database.
If found, updates the entity with the provided data.
Returns a response with the updated entity and a status code of 200 (OK).
If the entity does not exist, returns a 404 (Not Found) status code.

Deleting an Entity
HTTP Method: DELETE
Endpoint: /api/entities/{id}
Description: This endpoint allows clients to delete an existing entity by its unique identifier (ID).
Implementation:
Searches for the entity with the specified ID in the mock database.
If found, removes the entity from the mock database.
Returns a response with no content and a status code of 204 (No Content). If the entity does not exist, returns a 404 (Not Found) status code.

These CRUD endpoints provide essential functionalities for managing entities within the API, enabling clients to create, read, update, and delete entity resources as needed.

RETRIEVING A LIST OF ENTITIES ENDPOINT

The endpoint **/api/entities** allows users to retrieve a list of entities based on various criteria such as search terms and filters, if required.

Implementation Details:

- **Fetching and Returning Entities:**
  - The endpoint retrieves entities from the **mockDatabase** list.
  - It applies filtering based on the provided parameters such as name, address, deceased status, gender, start and end dates, and countries.
  - Entities that match the filtering criteria are returned as a collection.
- **Response Format:**
  - The response format is in JSON format.
  - It contains an array of entity objects, each representing an entity retrieved from the database.
  - Each entity object includes properties such as addresses, dates, deceased status, gender, ID, and names.

RETRIEVING A SINGLE ENTITY ENDPOINT

The endpoint /api/entities/{id} allows users to retrieve a single entity by its unique identifier (ID).

Implementation Details:

- **Fetching and Returning a Single Entity:**

The endpoint receives the entity ID as a parameter in the URL.
It searches the mockDatabase list for an entity with the provided ID.
If the entity is found, it is returned in the response.
If the entity is not found, a 404 Not Found response is returned.

- **Error Handling for Invalid IDs:**

If the provided ID does not match any entity in the database, the endpoint returns a 404 Not Found response.
This ensures that clients receive meaningful feedback when attempting to retrieve non-existent entities.

SEARCHING ENTITIES ENDPOINT

The endpoint **/api/entities** allows users to search across entities based on specific search criteria. Supported search fields include Address Country, Address Line, First Name, Middle Name, and Surname.

Implementation Details:

- **Handling the Search Query Parameter:**
  - The endpoint receives a **search** query parameter in the request URL.
  - The value of the **search** parameter represents the text to be searched across the specified fields.
- **Searching Across Specified Fields:**
  - The endpoint searches for entities that match the search criteria across the specified fields:
    - Address Country
    - Address Line
    - First Name
    - Middle Name
    - Surname
  - It filters entities based on whether any of the specified fields contain the search text.
  - Case-insensitive search is performed for better flexibility.
- **Response Format:**
  - The response format is in JSON.
  - It contains an array of entity objects, each representing an entity that matches the search criteria.
  - If no entities match the search criteria, an empty array is returned.

ADVANCED FILTERING ENDPOINT

The endpoint **/api/entities** provides advanced filtering capabilities for retrieving entities based on various criteria. Supported filtering parameters include Gender, Start Date and End Date, and Countries.

Implementation Details:

- **Handling Optional Query Parameters:**
  - The endpoint receives optional query parameters in the request URL, including:
    - **gender**: Gender of the entity.
    - **startDate**: Start date for filtering based on Date.DateValue field.
    - **endDate**: End date for filtering based on Date.DateValue field.
    - **countries**: List of countries for filtering based on Address.Country field.
  - Parameters are optional, allowing users to apply specific filters as needed.
- **Filtering Based on Specified Criteria:**
  - The endpoint applies filters based on the provided query parameters.
  - Gender filtering: Entities are filtered based on the specified gender, if provided.
  - Date filtering: Entities are filtered based on the specified start and end dates, if provided. All start and end dates are inclusive.
  - Country filtering: Entities are filtered based on the specified countries, if provided.
- **Response Format:**
  - The response format remains consistent with the previous endpoints and is in JSON.
  - It contains an array of entity objects, each representing an entity that matches the filtering criteria.
  - If no entities match the filtering criteria, an empty array is returned.

Example Request:-
GET /api/entities/?gender=Male&startDate=2000-01-01&endDate=2000-12-31

This request filters entities to include only those with gender "Male", dates falling between January 1, 2000, and December 31, 2000

API DOCUMENTATION

- **Endpoint URLs:**
  - **/api/entities**: Retrieves a list of entities with search and filter capabilities.
  - **/api/entities/{id}**: Retrieves a single entity by ID.
  - **/api/entities?search={searchTerm}&gender={gender}&startDate={startDate}&endDate={endDate}&countries={country1,country2}**: Endpoint for searching and filtering entities.
- **Request Parameters:**
  - **search**: Text to search across specified fields.
  - **gender**: Gender of the entity.
  - **startDate**: Start date for filtering entities.
  - **endDate**: End date for filtering entities.
  - **countries**: List of countries for filtering entities.
- **Response Format:**
  - JSON format containing an array of entity objects.
  - Each entity object includes properties such as ID, addresses, dates, deceased status, gender, and names.

CONCLUSION

In conclusion, the project to develop a REST API with CRUD functionality and advanced search/filtering capabilities has been successfully accomplished. This endeavor has not only met the specified requirements but also provided valuable insights and learning experiences.

Throughout the project, several key accomplishments have been achieved:

1. **Functional API**: The API now offers essential CRUD operations, allowing users to create, read, update, and delete entities efficiently. Additionally, advanced features such as searching and filtering have been implemented, enhancing the API's usability and flexibility.
2. **Robust Testing**: Rigorous testing, including unit tests for individual endpoints and integration tests to ensure proper interactions between endpoints, has been conducted. This ensures the reliability and correctness of the API, contributing to its overall quality.
3. **Comprehensive Documentation**: Detailed documentation for each endpoint, including endpoint URLs, request parameters, and response formats, has been provided. This documentation serves as a valuable resource for API users, facilitating ease of use and understanding.
4. **Reflection and Learnings**: Throughout the project, valuable insights have been gained, and lessons have been learned.

In conclusion, the completion of this project signifies the successful development of a functional and well-tested REST API. The experience gained from this project has furthered knowledge and skills in software development, paving the way for continued growth and improvement in this field.