KYC360
Comply and Outperform

About us

Company Description:

KYC360 is an award-winning technology company based in **London, Jersey and India** that offers compliance solutions for businesses. Their solutions include configurable, scalable, auditable and risk-based on-boarding, screening and pKYC solutions.

Role Description:

KYC360 is building a team in India and is on the lookout for a passionate developer to join our dynamic team and contribute to the development of exciting products!

As an entry-level developer, you'll play a key role in enhancing and expanding our diverse product suite.

What You'll Do:

Collaborate on the creation of new features and enhancements for KYC360's innovative product suite Stay current with the latest technologies and ensure our software remains cutting-edge Assist in triaging and improving core features of our existing software to meet the needs of our rapidly growing user base Work alongside a supportive team dedicated to building an exceptional product!

What We're Looking For:

You are an eager developer with a strong desire to learn and thrive in a startup environment Possess solid knowledge of Object-Oriented Programming (OOP) Comfortable working in a fast-paced setting and able to adapt to evolving requirements Familiarity or experience with SQL/RDBMS is a significant advantage Familiarity or experience with code versioning and/or CI/CD tools is highly desirable Exposure or experience with ASP.NET MVC is a plus.

KYC360
Comply and Outperform

C# Developer technical test

Table of Contents

KYC360
Comply and Outperform

Introduction

In this exercise, you will be building a *REST* API with endpoints serving Entity data from a mocked database.

Constraints

You are expected to write the API in C# using a .NET Core Web API project. You are not expected to have any previous .NET Core Web API experience and unfamiliarty with this project type will not be held against you when reviewing your submission. However, it is expected that you will be able to produce a functional API.

Database

At KYC360 we use SQL Server as our primary data storage technology. Setting up an instance of SQL Server with the necessary table schema and data required for the functionality in this test is outside of the scope of submission.

We *do* expect you to mock a database interaction layer in any way you see fit and generate data (which can be randomized) for the purposes of this test. The implementation of this database layer is left up to you.

Schema model

We have provided a C# interface and classes representing a single Entity below:

```
public interface IEntity
{   public List<Address>? Addresses { get; set; }

    public List<Date> Dates { get; set; }

    public bool Deceased { get; set; }

    public string? Gender { get; set; }

    public string Id { get; set; }

    public List<Name> Names { get; set; }
}

public class Entity : IEntity
{
    ...
}

public class Address
{  public string? AddressLine { get; set; }

    public string? City { get; set; }

    public string? Country { get; set; }
}

public class Date
{  public string? DateType { get; set; }

    public DateTime? Date { get; set; }
}

public class Name
{   public string? FirstName { get; set; }

    public string? MiddleName { get; set; }

    public string? Surname { get; set; }
}
```

KYC360
Comply and Outperform

Test

This test represents a common request when building an API.

- You need to provide a set of CRUD endpoints for Creating, Reading, Updating, and Deleting Entities.
- You need to provide an endpoint for retrieving a list of Entities, with searching and filtering capabilities.

Listing entities

Please provide an endpoint to fetch a list of entities.

Single entity

Users would like to be able to retrieve a single entity from the API. Please provide an endpoint to fetch an entity by id.

Searching entities

Users would now like to be able to search across the entities using the API. Your endpoint for fetching a list of entities will need to support searching for entities through the following fields:

- Address Country
- Address Line
- FirstName
- MiddleName
- Surname

If a user was to call your endpoint and provide a ?search=bob%20smith query parameter, your endpoint will return any entities where the text bob smith exists in any of the fields listed above.

Advanced filtering

The users would now like the ability to filter entities. Your endpoint for fetching a list of entities will need to support filtering using the following optional query parameters:

Parameter	Description
gender	Gender of the entity.
startDate	The start date for the Dates.Date field.
endDate	The end date for the Dates.Date field.
countries	0 or more countries for Addresses.AddressCountry field

All start and end dates are inclusive (e.g. startDate=2000-01-01&endDate=2000-12-31 will return 2000-01-01 <= Dates.Date <= 2000-12-31).

Bonus Challenges:

Challenge 1: Implement support for Pagination and Sorting

- The endpoint for retrieving entities should ideally be paginated and allow for users to specify page number and max page size (up to a reasonable amount).
- Some basic sorting capabilities should also be provided.

Challenge 2: Implementing Retry and Backoff Mechanism

In a real-world scenario, database write operations may occasionally fail due to transient issues, such as network problems or temporary unavailability of the database. To enhance the robustness of your system, implement a retry and backoff mechanism for database write operations in your mocked database interaction layer.

1. Retry Mechanism:

- Implement a mechanism that allows your API to automatically retry a failed database write operation.
- Decide on a reasonable number of retry attempts (e.g., 3) before considering the operation as permanently failed.
- Ensure that the retry attempts are made with a reasonable delay between them.

2. Backoff Strategy:

- Implement a backoff strategy to progressively increase the delay between retry attempts.
- Choose a suitable backoff algorithm (e.g., exponential backoff) to prevent overwhelming the database with repeated requests during transient failures.
- Adjust the parameters of the backoff strategy (initial delay, maximum delay, multiplier, etc.) based on your understanding of an appropriate strategy.

3. Logging:

- Ensure that relevant information about each retry attempt is logged for debugging purposes. Include information such as the number of attempts, the delay before each attempt, and the ultimate success or failure of the operation.

4. Test Cases:

- Create test cases to simulate scenarios where the database write operation fails initially but succeeds after a certain number of retries.
- Verify that the retry and backoff mechanism behaves as expected in different failure scenarios.

Document your implementation in the submission document, explaining the rationale behind your chosen retry and backoff strategy. Consider factors such as system stability, user experience, and the nature of potential transient failures.

Remember that this bonus challenge is an opportunity to showcase your problem-solving and system design skills. Good luck!

KYC360
Comply and Outperform

Submission

Please submit your solution by providing access to a Git repo. Don’t forget to include a document describing your solution and the reasoning behind your approach. Incomplete solutions will not be considered.