BI Developer Technical Assessment

Purpose

The purpose of this test is to assess how well a potential candidate performs based on the following criteria:

- Ability to write efficient and comprehensible code.
- Ability to use Git.
- Understanding of coherent Database schemas.
- Ability to create good API's.
- Ability to write Unit Tests.

This technical test will give you the opportunity to showcase your knowledge, so feel free to have fun with it and implement things how you see best – Good Luck!

Time

There is no set time for this test, spend as much time as you need. It is however, in your best interest to finish the test as soon as possible.

Prerequisites

In order to successfully complete this assessment, you will need to install the following:

- Microsoft Visual Studio 2019 (any edition) with the workloads:
  - ASP.Net and web development
  - Azure development
  - .NET desktop development
- Dotnet Core 3.1
- SQL Server 2019 Developer Edition

You should also receive a Skeleton Visual Studio solution called "DataImporter", a description of what is contained in this can be found further down in this document.

Scenario

The business wants to implement a new feature called the Data Importer. The Data Importer will automate the importing of product information that is provided by our clients in a file format to a Database. The Data Importer should have an API which will allow consumers to retrieve product data on request by supplying the Company and Feed Ids.

Requirements

An SQL Server Database

This Database must contain three tables: Company, Feed and Product.

There are two CSV files: Company and Feed. Use the data in these to populate the Company and Feed tables.

The Product table must contain a composite key made up of the Company, Feed and Product Ids.

You may use either Entity Framework Core or SQL to create, seed and update the tables.

Data Importer

The data importer should import Product Feed data in the CSV file form and persist them to the Product table in the database.

The test product data can be found in the folder "DataImporter.ConsoleApp\TestData". The folder structure is as follows:

Company_[**Company Id**] -> Feed_[**Feed Id**] -> Data.csv

Web API

A single web API endpoint must be exposed, use the ProductController to do this.

This endpoint when consumed should return the Products for the given Company and Feed Ids.

The response should be an Array of Product objects, each of which contain the following properties:

- Unique Id
- Name
- Brand
- Description

Make sure this is serialized from a class, instead of an anonymous object type.

Unit Tests

You will be required to write a few unit tests for the DataImporterService only. This can be on as many use cases as you feel necessary. Do not feel the need to go overboard and cover every single possible use case. What matters most is that you understand how to write a good suite of Unit Tests, enough to ensure the logic stays intact during future changes.

Git

Use Git for version control. Ensure your commits are small and comprehensible. Git has not been initialized on the project; you will need to do this. There is no requirement to push your commits to a remote repository, you may do this to safeguard your work if you wish to do so.

Code

Ensure your code is efficient and easily comprehendible by humans, in real life scenarios there can be feeds that have rows in the hundreds of thousands. Your importer should be able to handle this many rows and more.

Skeleton Project Description

The Visual Studio Solution provided contains skeleton projects and gives you the structure you will need to implement this feature.

DataImporter.Core

All the core code should be contained in here, having this layer of abstraction is beneficial as we can simply extract this library if we plan on implementing this feature in a different way, e.g., run the data importer as an Azure Function.

DataImporter.Core.Tests

This should contain all tests used to ensure that no breaking changes are introduced in the Core project.

Note: This project uses MSTest, if you feel more comfortable using NUnit or xUnit then feel free to substitute this project with one of those.

DataImporter.ConsoleApp

Use this app to Run the DataImporterService, this should allow for easy development and debugging. The test data can be found in the folder “TestData”.

DataImporter.Api

This project contains the API endpoint; where consumers can send requests for the product data based on the Company and Feed Id’s.

DataImporter Database

There is currently no project for the database side of things, you may use either SQL or Entity Framework Core to handle this. if you use SQL, please provide the script(s) you used to create and seed the Database tables.

Optional Requirements

These are optional requirements that you can implement, however are not necessary for the assessment itself.

- **Deploy the Data Importer application to Azure.**
  This can be done as either an Azure Function app or by using Azure Data Factory. The app can then import the feed files from Blob Storage and store them in an Azure SQL Database.
- **Deploy the API in a service in Azure.**
  This can be any service of your choosing. Ensure that you can make requests to the API either via Swagger (OpenAPI) or Postman.

The above are examples of how to approach these optional requirements. You may implement them in other ways if they make more sense to you.