Novibet Interview Assignment
EXPRESSYOURSELF

Bakas Giorgos

September 24, 2023

Contents

1 Intro-PLEASE READ FIRST

This document offers nothing more than a (almost) detailed explanation of the code I created. Therefore reading it doesn't come as any help to anyone who knows and understands the language. It was created as part of my journey in learning c sharp and WebAPIs. So please in any question raised by a miss-approach or something that I didn't develop correctly, consider contacting me (bakasgiorgos976@gmail.com) instead of searching for answers here. That clearly states that my solution is neither perfect nor right (without

mistakes), just a honest try. In case you were searching for Swagger responses, or the inputs format, they can be found in the other pdf I attached.
with best Regards,
G.Bakas

2 Report

Assumption: I implemented the database in SQLite to facilitate communication. The application is working correctly, providing the three functionalities, defined in the assignment, within the Swagger environment. Below, you will find brief information regarding the code, its logic, and an ideally optimized enhancement.

3 Controller

3.1 getData

3.1.1 Method Signature

public async Task¡IActionResult¿ GetData([FromQuery] string ipAddress):
This is a method within a controller that accepts an IP address (ipAddress) as a parameter and returns an IActionResult. The async keyword indicates that this method contains asynchronous operations.
Note:
1-**Task**: It represents an asynchronous operation that returns a result. In this case, the method returns a Task that will eventually yield an IActionResult when the operation is complete.
1-**IActionResult**: This is an interface representing an HTTP response. It allows you to return different types of results (e.g., OkObjectResult, NotFoundResult, BadRequestResult, etc.) depending on the outcome of the method's execution. Using IActionResult provides flexibility in returning appropriate HTTP status codes and data in response to client requests.
3-[**FromQuery**]: This attribute is used to specify that the value of the ipAddress parameter should be retrieved from the query string of the HTTP request. In the context of a GET request, query string parameters are typically passed in the URL after a question mark, e.g., ?ipAddress=value. By using [FromQuery], you are indicating that the value of ipAddress should be bound from the query string.

3.1.2 Asynchronous Call

Ip2cService ip2cService = new Ip2cService(new HttpClient());:
Here, an instance of the Ip2cService class is created, which is responsible for interacting with an external web service. **var (twoLetterCode, threeLetterCode, name) = await ip2cService.GetCountryCodesByIpAsync(ipAddress);**:

This line makes an asynchronous call to the GetCountryCodesByIpAsync method of the ip2cService object. It passes the ipAddress as a parameter to the method and awaits the response.

3.1.3 await Keyword

await is a keyword that signifies that the code execution will pause at this point until the awaited asynchronous operation (GetCountryCodesByIpAsync) is completed. During this pause, the application's main thread is free to handle other tasks, enhancing responsiveness.

3.1.4 Handling the Response:

After the asynchronous call returns a response (in this case, twoLetterCode, threeLetterCode, and name), you can proceed with any necessary processing or handling of the data.

3.1.5 Returning an IActionResult

Finally, the method returns an IActionResult using return Ok(data);. The Ok method indicates a successful HTTP response, and data would contain the processed or retrieved information.

3.2 getIps

3.2.1 TwoLetterCodes Class

TwoLetterCodes is a custom class with a property TwoLetterCodesFilter, representing a list of two-letter codes received as input.

3.2.2 Controller Action

GetIps is an HTTP GET action within a controller. It is annotated with [HttpGet("getAnIPAddresses/rawinput")] to specify the route. The action accepts a parameter rawInput of type TwoLetterCodes, which is deserialized from the query parameters.

3.2.3 Database Connection

A connection string for a SQLite database is defined (connectionString).

3.2.4 Data Containers

Lists (dataList and dataCorrectResponseList) are initialized to store retrieved data and transformed responses.

3.2.5 Data Retrieval Loop

A foreach loop iterates through the TwoLetterCodesFilter list. Inside the loop, it calls the Retriever.Return IPs method to retrieve IP data based on the two-letter code. If data is retrieved (data != null), it is added to both dataList and dataCorrectResponseList after transforming it into the appropriate format.

3.2.6 Response Handling

If there is data in dataList, the dataCorrectResponseList is converted to an array (toReturn). The action returns an HTTP OK response with the array of data (toReturn) as the response body. If no data is found, it returns a "Not Found" response.

3.2.7 Data

I personally used to return all the IP Addresses found in a specific country, to ensure that there is no error (no IP forgotten).
For this case, i left my structures in the code (forbidding them to be returned by comment-ing them), so if anyone should want to retrieve the addresses also, he shouldn't get in much trouble.

3.3 UpdateIpsBatch

3.3.1 Controller Action

UpdateIpsBatch is an HTTP PUT action within a controller. It is annotated with [HttpPut("UpdateIpsBatch")] to specify the route.

3.3.2 Try-Catch Block

The code is wrapped in a try-catch block to handle exceptions that might occur during the batch update.

3.3.3 Database Update

An instance of the update db class is created as Update. The total number of IPs is calculated using Update.GetMaxIps().

3.3.4 Batch Processing

The total number of batches is calculated, with each batch containing 100 IPs. A for loop iterates through the batches.

3.3.5 IP Retrieval and Update

Inside the loop, a list ips is created to store IPs for the current batch, retrieved using Update.GetIPsByPage(i). Another loop iterates through the IPs in ips. For each IP, data is retrieved and updated using Update.IP2C(ip) and Update.Update(ip, ip2cResponse). A Thread.Sleep(1000) is used to pause execution for 1 second (to control the rate of processing).

3.3.6 Response Handling

If the batch update process completes without errors, an HTTP OK response is returned with the message "Batch update completed." If an exception occurs during processing, a 500 Internal Server Error response is returned with an error message.

4 DataBase

4.1 Read

Information and Explanation about the method used to search an IP address in the Database.

4.1.1 DbRead Class

This class represents a database reader utility responsible for reading data from a SQLite database.

4.1.2 Constructor

The class has a constructor that accepts a connectionString. The connectionString is used to specify the database file location and version.

4.1.3 ReadDataFromDatabase Method

1-This method takes an ipAddress as input and returns a DataDTO object.
2-It establishes a connection to the SQLite database using the provided connectionString.
3-Inside a using block, it creates an SQL command to retrieve data from the database.
4-The SQL query performs a JOIN operation between the Countries and IPAddresses tables, matching records where the IP address in the IPAddresses table matches the provided ipAddress.
5-The command.Parameters.AddWithValue("@ipAddress", ipAddress); line adds the ipAddress as a parameter to the SQL query to prevent SQL injection.
6-It then executes the query and reads the result using a SQLiteDataReader.
7-If a matching record is found, it populates a DataDTO object (myData) with

data from the database, including the country name, two-letter code, and three-letter code.
8-If no matching record is found, it returns null.

4.2 Write

4.2.1 DbWrite Class

This class represents a database writer utility responsible for writing data to a SQLite database.

4.2.2 Constructor

The constructor initializes the connectionString field, which specifies the database file location and version.

4.2.3 GetCountryId Method

This method takes a two-letter country code as input and returns the corresponding country's ID from the Countries table in the database.

4.2.4 GetNewIpId and GetNewCId Methods

These methods retrieve the next available unique ID for the IPAddresses and Countries tables, respectively.

4.2.5 TwoLetterCodeExists Method

This method checks if a two-letter country code already exists in the Countries table and returns the corresponding country's ID. 0 is standard nothing found value.

4.2.6 InsertData Method

This method inserts data into the database, including IP addresses and country information. It first checks if the country with the provided two-letter code exists in the database. If the country exists, it inserts data into the IPAddresses table with the existing country ID. If the country does not exist, it inserts data into both the IPAddresses and Countries tables, generating new unique IDs.

4.3 Update

This class provides methods to fetch, update, and insert data into the SQLite database. It handles various database operations, including checking for the existence of countries, updating IP addresses, and making external HTTP requests to gather IP2C data.

4.3.1 update db Class

This class represents a utility for updating the database with IP address data.

4.3.2 GetNewCId Method

Retrieves the next available unique Country ID.

4.3.3 get max ips Method

Retrieves the total count of IP addresses in the IPAddresses table.

4.3.4 GetIPsByPage Method

Retrieves a list of IP addresses based on the provided page number and a batch size of 100.

4.3.5 Country exists Method

Checks if a country with a given two-letter code exists in the Countries table.

4.3.6 IP2C Method

Makes an HTTP request to "http://ip2c.org" to retrieve IP2C data for a given IP address. Parses the response and returns a DataDTO object with relevant data.

4.3.7 Update Method

Updates IP address data in the database:
Checks if the country with the two-letter code from ip2cresponse exists.
If not, it creates a new country entry in the Countries table and retrieves the new country ID.
Updates the CountryId and UpdatedAt fields in the IPAddresses table for the given IP address.

5 Data Structures

5.1 DataDTO

Object having:
1-Country Name: name of the country,
2-TwoLetterCode: two letter code for the country, for example: GR -¿ Greece,
3-ThreeLetterCode: three letter code for the country, for example: GRC -¿ Greece
Used to transfer data of a country, from web requests to database.

5.2 Data

Object having:
1-Country: two letter code of the country,
2-Ips found: number of total IP Addresses found associated with a country,
3-Ips List: List of all IP Addresses associated with a country
4- last update: date time of last updated IP in the above list.
Used to return data from the Search in database. Currently not used, as the ips list was not asked.

5.3 Data correct response

Object having:
All of the above except the IP addresses list. Used to return data from the Search in database.

6 Ip2c Request

6.1 Ip2cService Class

This class provides a service to retrieve country codes by IP address using an external API.

6.2 Constructor

The constructor takes an HttpClient as a parameter and initializes the http-Client field for making HTTP requests.

6.3 GetCountryCodesByIpAsync Method

This method asynchronously fetches country codes (TwoLetterCode, ThreeLetterCode, and name) for a given IP address.
It makes an HTTP GET request to "https://ip2c.org/ipAddress" using http-Client.
Ensures the response indicates success with response.EnsureSuccessStatusCode().
Reads and splits the response content by ';'.
Checks if the response contains at least 3 parts and the first part is "1" (indicating success).
Returns a tuple with country codes if successful; otherwise, returns null values.
Handles exceptions and unexpected responses by logging messages to the console.

7 Optimized Solution

Knowing that my solution to the task provided isn't the optimized, i marked a few points would upgrade the solution, being in code management, data management, methods management and so on.

7.1 Static Helper Class for Common Functions

Creating a static helper class for common functions like GetNewIpId or GetNewCountryId would be a good point. This can help reduce code duplication and centralize common functionality.
Instead of defining these functions in multiple classes, a single static class with static methods for these operations can do the job. This can make the project more maintainable and avoid redundancy.

7.2 Database Connection String

Externalizing the database connection string to a configuration file or using Dependency Injection to inject it into classes is considerable. This makes it easier to change the connection string without modifying the code.

7.3 Error Handling

The error handling is somewhat limited to writing messages to the console. It would be better to log errors and exceptions using a proper logging framework. This allows for better tracking and debugging of issues.

7.4 Hardcoded Values

Some hardcoded values exist, such as the batch size (e.g., 100) in the GetIPsByPage method. Making these values configurable or passing them as parameters to methods to make the code more flexible is considerable.

7.5 Redundant Variable Declarations

In some methods, variables at the beginning of the method get declared, but aren't used until later. Declaring variables closer to where they are used to improve code readability is considerable.

7.6 Naming Conventions

Consistent naming conventions for variables, methods, and classes should be used.