Project Report for Phone Book API

Aman Hogan-Bailey

The University of Texas at Arlington

Contents

How To Run

Running The Software

In order to run the `PhoneBook` API you need to first download `Visual Studio 2019` or later and `NET 5.0` or later. Below is a guide on how to run the setup `PhoneBook` API in `Visual Studio`.

1. Open Visual Studio: Open the `Visual Studio IDE` on your computer.
2. Open the project: To get started, clone the project, and open it in Visual Studio. Once opened, build the project, and run it. You can do this by going in the file explorer and clicking on the `.sln` file.
3. Build the solution: Once you have the solution opened in Visual Studio, you need to build the solution by clicking on "`Build" > "Build Solution`" in the main menu or by pressing "`Ctrl + Shift + B`".
4. Run the API: You can test the API using a tool such as Postman or by using the Swagger UI. The Swagger UI can be accessed by navigating to `https://localhost:<port>/swagger/index.html`, where `<port>` is the port number on which the API is running.

How to Run Using Docker

1. Make sure the Docker Daemon is running.
2. Open the terminal.
3. Navigate to the folder containing the `Dockerfile`
4. Run the following commands in the terminal:
    a. `docker build -f Dockerfile --force-rm -t myapp/test .`
    b. `docker run -d -p 8080:80 myapp/test`
5. Navigate to the webpage: http://localhost:8080/swagger/index.html
6. `You should then see the swagger UI`

Alternate way to run docker

1. Open up terminal in the directory containing the `myapptest.tar`
2. Run this command: `docker load -i myapptest.tar`
3. Navigate to the webpage: http://localhost:8080/swagger/index.html
4. You should then see the swagger UI.

Running Unit Tests

Brief overview of collection `.json` files: Collection Format JSON file in Postman is a file that contains a collection of HTTP requests, each with its own set of properties and test scripts, organized into folders and sub-folders. It allows you to share your collection of requests with others, backup your requests, or import them into another Postman instance. Below is the process of how to access the selected unit tests in this project:

1. Open the Selected Folder which should contain:
    a. `PhoneBook API.postman_collection.json`
    b. `PhoneBook Tests Demo.postman_collection.json`
2. Open Postman and click on the "`Import`" button located in the top left corner of the window.
3. Select the "`Import from File`" option.
4. Browse your computer to find the Collection JSON file that you want to import.
5. Select the file and click on "`Open`".
6. Postman will import the collection and display it in the "`Collections`" tab.
7. You can now use the imported collection to send requests to the API and run tests on the responses.

Code Description

This `PhoneBook` API is a RESTful web service that allows users to manage a phone book through HTTP requests. The *main functional components that were modified* in this project include:

- `PhoneBookController.cs`
- `DictionaryPhoneBookService.cs`
- `Logger.cs`
- `myDatabase.txt`
- `Logfile.txt`

`PhoneBookController.cs`

The `PhoneBookController.cs` file defines the endpoints for the `PhoneBook API`. It includes four methods:

1. `List` -The List method returns all phone book entries by calling the `List` method of the `IPhoneBookService` interface. It then logs the event using the `Logger.WriteLog` method.
2. `Add` - The Add method adds a new phone book entry by calling the `Add` method of the `IPhoneBookService` interface. If the entry is added successfully, it returns an `HTTP 200 OK` response with a success message. If the entry could not be added, it returns an `HTTP 400 Bad Request` response with an error message.
3. `DeleteByName` - The `DeleteByName` method deletes a phone book entry by name by calling the `DeleteByName` method of the `IPhoneBookService` interface. If the entry is deleted successfully, it returns an `HTTP 200 OK` response with a success

message. If the entry could not be found, it returns an `HTTP 404 Not Found` response with an error message.

4. `DeleteByNumber` - The `DeleteByNumber` method deletes a phone book entry by number by calling the `DeleteByNumber` method of the `IPhoneBookService` interface. If the entry is deleted successfully, it returns an `HTTP 200 OK` response with a success message. If the entry could not be found, it returns an `HTTP 404 Not Found` response with an error

`DictionaryPhoneBookService.cs`

The `DictionaryPhoneBookService` class implements the `IPhoneBookService` interface and provides an in-memory phone book service using a dictionary to store phone book entries. It also writes the phone book dictionary to a text file for persistence. The class has a private field called `phoneBook` which is a dictionary that stores phone book entries. The keys of the dictionary are the phone numbers of the entries, and the values are the corresponding names. The class also has a constructor that takes a string parameter representing the path to the text file where the phone book dictionary will be persisted. In the constructor, the class reads the phone book dictionary from the text file and populates the `phoneBook` field. The class implements four different methods:

1. `Add(PhoneBookEntry phoneBookEntry)`: This method takes a `PhoneBookEntry` object as a parameter and adds it to the phone book dictionary. If the name and phone number are valid and do not already exist in the dictionary, `AddSuccess` is set to "`SUCCESS`". If the name and phone number are invalid,

`AddSuccess` is set to "`FAIL`". The method also writes the updated phone book dictionary to the text file.

2. `Add(string name, string phoneNumber)`: This method adds a new phone book entry to the dictionary, but does not write the dictionary to the text file. If the name and phone number are not null, they are added to the dictionary.
3. `List()`: This method returns a list of all phone book entries as `PhoneBookEntry` objects.
4. `DeleteByName(string name)`: This method deletes a phone book entry with the given name from the dictionary and writes the dictionary to the text file. If the entry is successfully deleted, `DeleteSuccess` is set to "`SUCCESS`". If the entry does not exist in the dictionary, `DeleteSuccess` is set to "`FAIL`".
5. `DeleteByNumber(string number)`: This method deletes a phone book entry with the given phone number from the dictionary and writes the dictionary to the text file. If the entry is successfully deleted, `DeleteSuccess` is set to "`SUCCESS`". If the entry does not exist in the dictionary, `DeleteSuccess` is set to "`FAIL`".

`Logger.cs`

This code defines a static Logger class with a single method called `WriteLog`. This method takes a string message as input and writes it to a log file at the location specified by `log_path`.

The `log_path` variable is initialized with the path of a file named "`LogFile.txt`" in the current directory. The `Path.Combine` method is used to concatenate the directory path and the file name.

This Logger class is used by other parts of the code to log events for auditing purposes. For example, in the `PhoneBookController.cs` file, the `Logger.WriteLog` method is called to log events when the `List, Add, and Delete` methods are called.

`myDatabase.txt`

`myDatabase.txt` is a text file that serves as a data store for the phone book entries. It contains the names and phone numbers of the users, with each entry on a separate line. The data in the file is in a plain text format.

`Logfile.txt`

`Logfile.txt` is a text file that is used to log events that occur during the operation of the phone book API. It contains a timestamp and a message for each event, with each entry on a separate line. The file is created and updated by the Logger class using standard file I/O operations.

Regular Expression Design

The regular expressions are used in the code to validate the input values of name and phone number. The regular expressions are designed to match various formats of names and phone numbers, making the code more robust and flexible. The name for a person's name has one regular expression. The phone number has nine different regular expressions to cover the wide variety of phone numbers that could be used.

1. `regexName`: This regular expression matches a person's name. It allows for various formats of names such as first and last name separated by spaces or punctuation, first and last name separated by a hyphen, or first, middle and last names separated by spaces or punctuation. The regular expression uses character classes, groups and quantifiers to match the different formats of names.
2. `regexPhone`: This regular expression matches a North American phone number. It allows for various formats of phone numbers such as (xxx) xxx-xxxx, xxx-xxx-xxxx, xxx.xxx.xxxx, xxx xxx xxxx, and international numbers that start with a plus sign. It also allows for extensions. The regular expression uses character classes, groups, quantifiers, and optional characters to match the different formats of phone numbers.
3. `regexForeignPhone1`: This regular expression matches a 5-digit foreign phone number.
4. `regexForeignPhone2`: This regular expression matches a 9-digit foreign phone number with optional separators such as spaces, periods, or hyphens.
5. `regexForeignPhone3`: This regular expression matches a 5-digit foreign phone number with optional separators such as periods, hyphens, or spaces.

6. `regexForeignPhone4`: This regular expression matches an international phone number with 3-digit country code and optional separators such as spaces, periods, or hyphens.
7. `regexForeignPhone5`: This regular expression matches an international phone number with 1 or 3-digit country code and optional separators such as spaces, periods, or hyphens.
8. `regexForeignPhone6`: This regular expression matches a Danish phone number with optional country code.
9. `regexForeignPhone7`: This regular expression matches a 7-digit foreign phone number with optional separators such as spaces, periods, or hyphens.
10. `regexForeignPhone8`: This regular expression matches a 10-digit foreign phone number with separators such as periods, spaces or hyphens.

Assumptions

This program was built on a few assumptions:

1. Buffer Overflow: The user would not enter an exceedingly long name. Currently there are no name size restrictions.
2. Memory Leak/Overflow: The entire phone number and name database is stored in a variable then reprinted into the text file. This means each time the database file is opened, it is wiped, then reprinted with the value of the list variable. This method is inefficient, and that piece of data can easily overflow.
3. Regular Expression Edge Cases: The regular expressions should handle most cases, but they are functioning on the assumption that the names are not exceedingly unique.
4. Unique Names/Phone Number: Two people can have the exact same name and phone number. When deleting by name or phone number, it chooses the first entry in the list. This program assumes that everyone's phone number and name are unique.

Pros/Cons

There are several potential downsides or pitfalls with the approach I have taken:

1. Scalability: The use of a text file to persist phone book data may not be scalable for very large datasets. If the number of entries in the phone book grows significantly, it may become impractical to store all the data in a single text file.
2. Security: The current implementation does not include any authentication or authorization mechanisms, which could be a security risk in certain contexts. If the API is exposed to the public internet, it could be vulnerable to attacks such as SQL injection or cross-site scripting.
3. Error handling: The current implementation does not include robust error handling and may not handle all possible edge cases. For example, if the phone book file is locked by another process, the application may not be able to write to it and could crash.
4. Code maintainability: The current implementation is relatively simple and easy to understand, but as the codebase grows and new features are added, it may become more difficult to maintain and modify. The lack of modularity and separation of concerns could make it harder to extend the application in a clean and organized way.