REST API Input Validation Project Report

Quynh Nguyen (1001791420)
CSE 4382: Secure Programming

INSTRUCTOR: THOMAS L. JONES
April 21, 2023

1 Setup Instructions

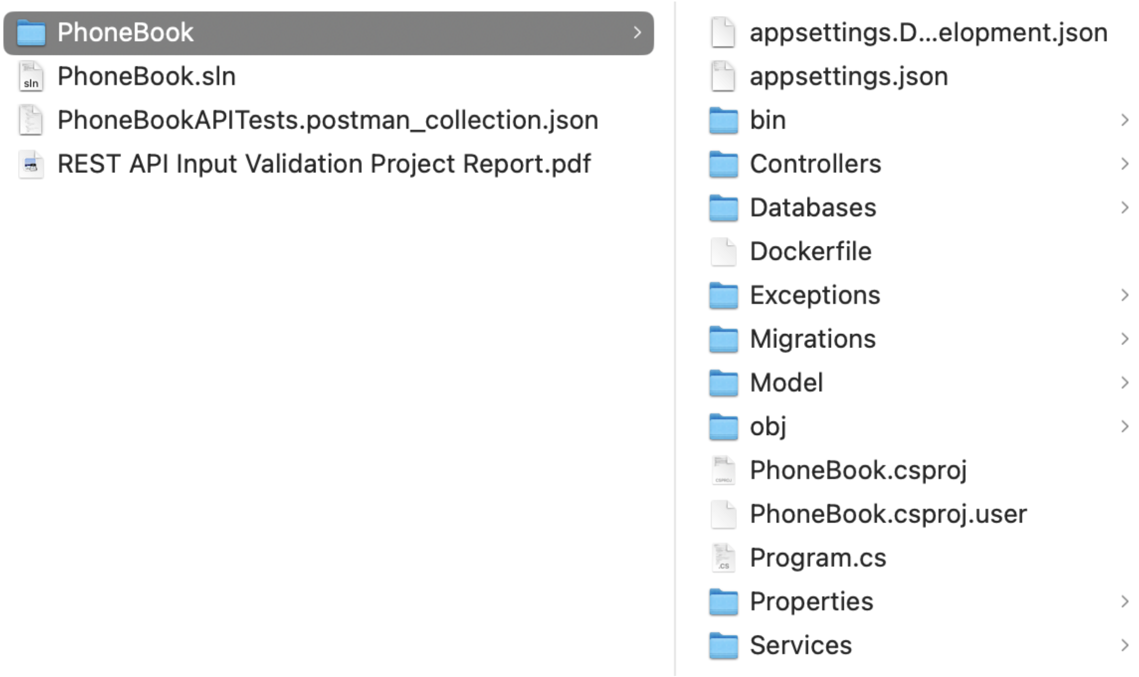
1.1 Build Instructions

Clone the PhoneBook repository or download the project folder.

From the directory that the repository is in, execute:

```
cd PhoneBook/PhoneBook
```

Your current working directory's file structure should look like this (right panel):



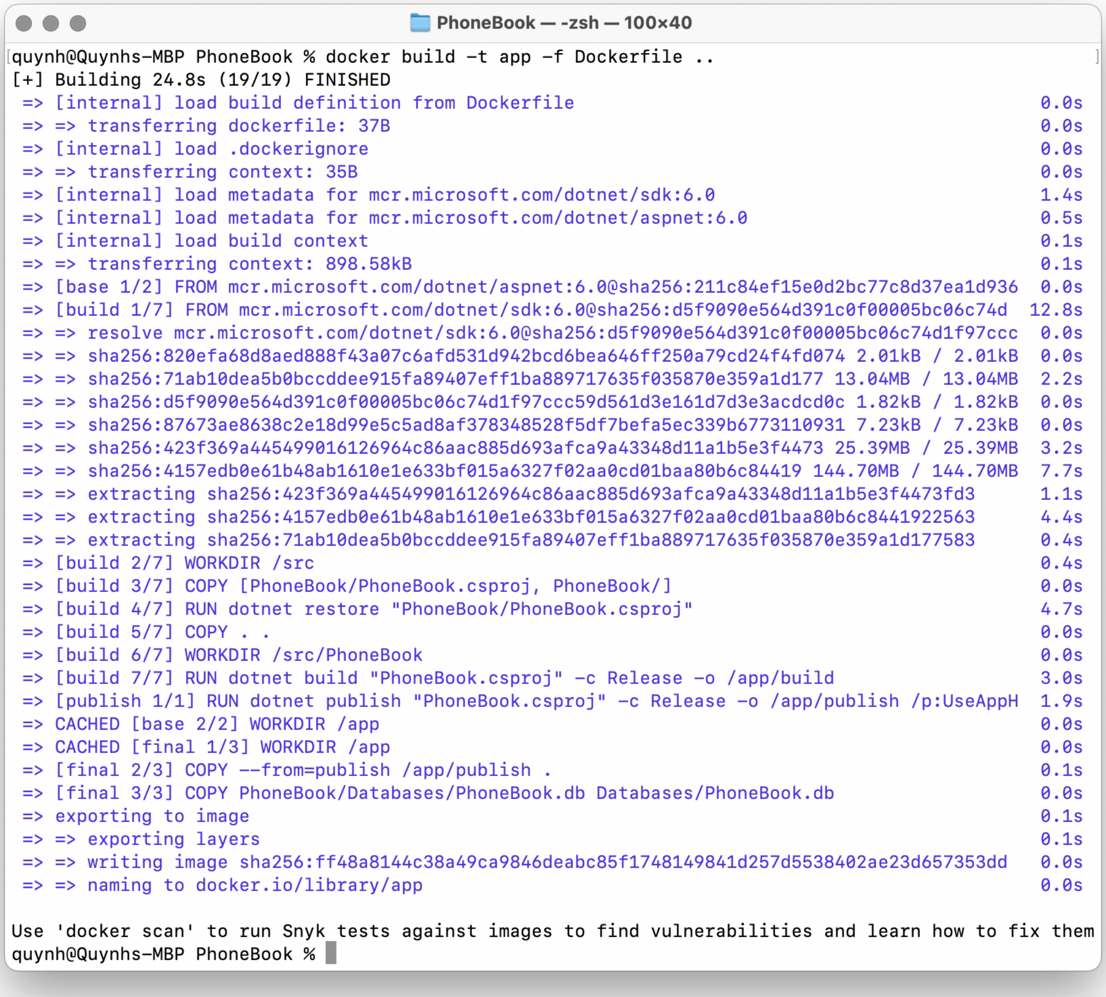
PhoneBook
PhoneBook.sln
PhoneBookAPITests.postman_collection.json
REST API Input Validation Project Report.pdf

appsettings.D...elopment.json
appsettings.json
bin
Controllers
Databases
Dockerfile
Exceptions
Migrations
Model
obj
PhoneBook.csproj
PhoneBook.csproj.user
Program.cs
Properties
Services

Inside a terminal at the current directory, build the Docker image with the command:

```
docker build -t app -f Dockerfile ..
```

The corresponding output should look similar to:



```
[quynh@Quynhs-MBP PhoneBook % docker build -t app -f Dockerfile ..
[+] Building 24.8s (19/19) FINISHED
 => [internal] load build definition from Dockerfile                                           0.0s
 => => transferring dockerfile: 37B                                                            0.0s
 => [internal] load .dockerignore                                                              0.0s
 => => transferring context: 35B                                                               0.0s
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk:6.0                              1.4s
 => [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:6.0                           0.5s
 => [internal] load build context                                                              0.1s
 => => transferring context: 898.58kB                                                          0.1s
 => [base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:6.0@sha256:211c84ef15e0d2bc77c8d37ea1d936  0.0s
 => [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:6.0@sha256:d5f9090e564d391c0f00005bc06c74d  12.8s
 => => resolve mcr.microsoft.com/dotnet/sdk:6.0@sha256:d5f9090e564d391c0f00005bc06c74d1f97ccc  0.0s
 => => sha256:820efa68d8aed888f43a07c6afd531d942bcd6bea646ff250a79cd24f4fd074 2.01kB / 2.01kB  0.0s
 => => sha256:71ab10dea5b0bccddee915fa89407eff1ba889717635f035870e359a1d177 13.04MB / 13.04MB  2.2s
 => => sha256:d5f9090e564d391c0f00005bc06c74d1f97ccc59d561d3e161d7d3e3acdcd0c 1.82kB / 1.82kB  0.0s
 => => sha256:87673ae8638c2e18d99e5c5ad8af378348528f5df7befa5ec339b6773110931 7.23kB / 7.23kB  0.0s
 => => sha256:423f369a445499016126964c86aac885d693afca9a43348d11a1b5e3f4473 25.39MB / 25.39MB  3.2s
 => => sha256:4157edb0e61b48ab1610e1e633bf015a6327f02aa0cd01baa80b6c84419 144.70MB / 144.70MB  7.7s
 => => extracting sha256:423f369a445499016126964c86aac885d693afca9a43348d11a1b5e3f4473fd3      1.1s
 => => extracting sha256:4157edb0e61b48ab1610e1e633bf015a6327f02aa0cd01baa80b6c841922563      4.4s
 => => extracting sha256:71ab10dea5b0bccddee915fa89407eff1ba889717635f035870e359a1d177583      0.4s
 => [build 2/7] WORKDIR /src                                                                   0.4s
 => [build 3/7] COPY [PhoneBook/PhoneBook.csproj, PhoneBook/]                                  0.0s
 => [build 4/7] RUN dotnet restore "PhoneBook/PhoneBook.csproj"                                4.7s
 => [build 5/7] COPY . .                                                                       0.0s
 => [build 6/7] WORKDIR /src/PhoneBook                                                         0.0s
 => [build 7/7] RUN dotnet build "PhoneBook.csproj" -c Release -o /app/build                   3.0s
 => [publish 1/1] RUN dotnet publish "PhoneBook.csproj" -c Release -o /app/publish /p:UseAppH  1.9s
 => CACHED [base 2/2] WORKDIR /app                                                             0.0s
 => CACHED [final 1/3] WORKDIR /app                                                            0.0s
 => [final 2/3] COPY --from=publish /app/publish .                                             0.1s
 => [final 3/3] COPY PhoneBook/Databases/PhoneBook.db Databases/PhoneBook.db                   0.0s
 => exporting to image                                                                         0.1s
 => => exporting layers                                                                        0.1s
 => => writing image sha256:ff48a8144c38a49ca9846deabc85f1748149841d257d5538402ae23d657353dd   0.0s
 => => naming to docker.io/library/app                                                         0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
quynh@Quynhs-MBP PhoneBook %
```

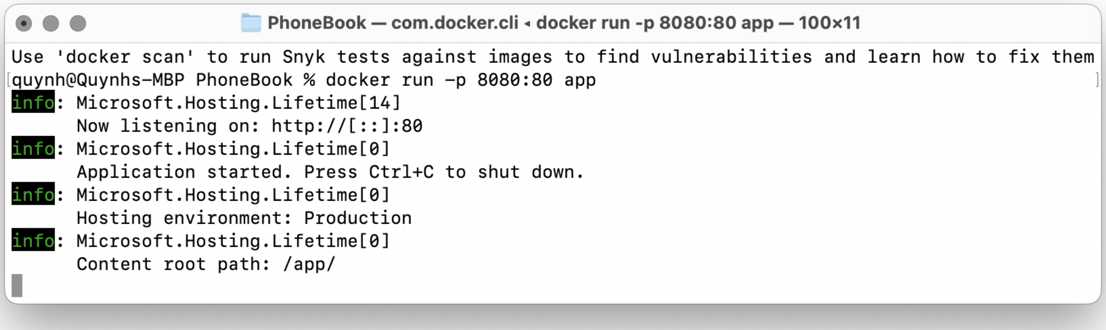
1.2 Run Instructions

To run the web server in the container and be able to access it on your host machine, you need to map port 80 from the container to an available port on your host machine.

To do so, execute the following command:

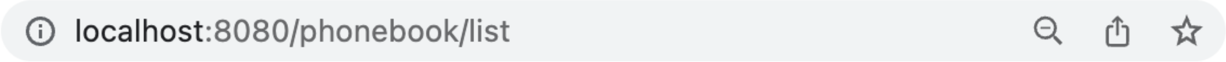
```
docker run -p 8080:80 app
```

The corresponding output should look like this:


PhoneBook — com.docker.cli ◂ docker run -p 8080:80 app — 100×11

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
quynh@Quynhs-MBP PhoneBook % docker run -p 8080:80 app
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app/
```

You should now be able to access the API endpoints at localhost:8080/PhoneBook/{API_ENDPOINT}. For example, to list the phone book entries, visit localhost:8080/PhoneBook/list.



localhost:8080/phonebook/list

2 Architecture

2.1 Overall Architecture

The REST API follows a basic model-view-controller (MVC) pattern but without the view as this backend only serves data in the form of JSON objects, not HTML content. When HTTP requests are sent to the port (80 on the container), the router maps each endpoint to a corresponding “action function.” These “action functions” are defined by the controller, or specifically, the PhoneBookController, and they determine how the server should respond to the request.

2.2 Dependencies

The PhoneBookController has 2 dependencies, an IPhoneBookService and ILogger. At build time, the ASP.NET Core builder injects the 2 dependencies into the controller. Specifically, for this application, the DatabasePhoneBookService is injected as the IPhoneBookService, and ILogger is automatically injected by ASP.NET Core.

2.3 DatabasePhoneBookService

The DatabasePhoneBookService provides the business logic of the application. This service does so via a dependency injection of a DbContext, which does the actual communication with the database via defined API calls, abstracting away the SQL statements from developers. Because the DatabasePhoneBookService is dependent on a DbContext, a scoped object provided by the Entity Framework Core framework, the service itself must also be scoped. The DatabasePhoneBookService is also dependent on an ILogger, which again, is automatically injected by ASP.NET Core.

2.4 Logger

The specific logger used in this REST API is the Serilog logger. This logger is injected as an ILogger object into the PhoneBookController and DatabasePhoneBookService to audit any API calls, such as listing the phone book contents, adding a new entry, and removing entries. The logger logs the timestamp, log level, log source, message, and exception associated with the log (if any).

2.5 Models

1. **PhoneBookEntry**: This is the actual object that users are exposed to via the API. It contains 2 fields, a name and number, both of which are strings. It is in this object where the input validation is applied via regular expression attributes. The regular expressions are matched and enforced automatically by ASP.NET Core's "ModelState" check, which asserts that the data matches the model schema.
2. **PhoneBookEntryDB**: This model is what gets actually stored in the database. It has the same fieldds as the PhoneBookEntry object with an additional GUID field to serve as the primary key. Users are not exposed to this model.

2.6 Persistent Storage

For persistent storage, an SQLite database is used.

3 Assumptions

3.1 Name Assumptions

The follow assumptions are made regarding the name input field:

- There cannot be duplicate names. Any attempt to add an entry with an existing name in the database will result in overwrite of the corresponding phone number.
- Users can have at most one middle name, if they choose to add it.
- Names are case sensitive. "Bob" and "bob" are considered different users.
- There can be at most one apostrophe in the name, and it can only appear after a single letter.

3.2 Phone Number Assumptions

- There cannot be duplicate phone numbers. Any attempt to add an entry with an existing number in the database will result in overwrite of the corresponding name.
- The country code can be space-separated from the start of the phone number by 1 space.
  - e.g. +1 (123)456-7890 and +1(123)456-7890 are both valid
- For Dutch phone numbers, the country code can also be period-separated from the start of the phone number by one period.
  - e.g. 45 12 34 56 78 and 45.12.34.56.78 are both valid
- If the first 3 digits of a U.S. phone number are surrounded by parenthesis, there must not be a space after the close parenthesis.
  - e.g. (123) 456-7890 is not valid, but (123)456-7890 is valid
- Country codes are at most 3 digits long.
- Letters are not valid characters.

- ○ e.g. There should be no “ext” text in the phone number input string

4 Pros/Cons of Approach

4.1 Pros

Using Entity Framework Core significantly helped with saving data to the disk because it essentially removed the need to write explicit SQL statements, which from a security standpoint, help protect against SQL Injection attacks. Additionally, using ASP.NET Core allowed for efficient input validation as it provides data annotations, which include regular expression matching.

4.2 Cons

Because Serilog doesn’t provide an out-of-the-box interface for viewing the logs, an additional package for log viewing may be necessary to have an interactive user interface to filter out specific log levels and search by timestamps, messages, etc.

Additionally, because duplicate names are not allowed, it is not possible to have two different users with the same name in the database.