**Instructions for building and running software and unit tests:**

Running the code:

* Open Visual Studio.
* Click on "File" in the top left corner and select "Open Folder".
* Navigate to the folder where your Python code is located and select it.
* Open the "Terminal" tab in Visual Studio by clicking on "View" and then "Terminal".
* Install libraries by using command in terminal: **pip install -r requirements.txt**
* In the terminal, navigate to the folder containing your Python code,
* In our case keep the app.py tab open.
* To run the app, type the command in terminal: **uvicorn app:app --reload**
* Navigate to: <http://127.0.0.1:8000/docs>
* Logging in:

I created some accounts ready to show the functionality of the application. Here are the two accounts you can use to authenticate with:

1. Username: johndoe with Password: 12345
2. Username: alice with Password: 56789 //this user will not be able to to do anything this they have the inactive variable set to true.

It is important to choose the “user” scope as it is a requirement for authentication, and you can choose between the “read” or both “read”/”write” scopes in order to test the roles requirements. The reason why I chose this method for authentication will be discussed later in this document.

Creating Docker Container:

* Docker files are created and setup.
* Build it using command: **docker compose up --build**
* Navigate to: <http://localhost:8000/docs/>
* Logging in:

I created some accounts ready to show the functionality of the application. Here are the two accounts you can use to authenticate with:

1. Username: johndoe with Password: 12345
2. Username: alice with Password: 56789 //this user will not be able to to do anything this they have the inactive variable set to true.

It is important to choose the “user” scope as it is a requirement for authentication, and you can choose between the “read” or both “read”/”write” scopes in order to test the roles requirements. The reason why I chose this method for authentication will be discussed later in this document.

How to run unit tests:

* Since the unit testing is in a separate file, being able to run it to conduct the tests will require the use docker desktop.
* Once you go to the container and go to the “exec” section you can use the command python test\_regex.py in order to run the test file.
* Here is a screenshot of where the “exec” section is and it executing on my machine as an example:

A screenshot of a computer

Description automatically generated

* My created test inputs are in the **student\_inputs.txt** file.

**Description of how code works:**

My code was built using the python starter, so it contains code for a RESTful API built with FastAPI that interacts with an SQLite database. The main API allows users to add and retrieve and delete phonebook entries, which consist of a person's full name and phone number. There are also other API endpoints that are used for authentication with Oauth2 which will be explained later. This application uses FastAPI, a Python web framework, for creating the REST API and uses Pydantic for data validation and SQLAlchemy as the ORM as these tools were used in the FastAPI documentation. Whenever one of the 4 actions (list, add, delete by name, delete by number) is done, there is first an auth check that verifies that the user is logged in and has the correct roles to do said action. If they do not have the appropriate roles or are not logged in, there will be appropriate error messages sent to the user. If auth is good, then there will be further checks with the add and delete actions such as verification of valid data and doing some error checking. First, the input is sent to the proper functions to verify user input with regular expressions and then goes through some queries first, for the add function, it verifies that there is not existing entry that matches the user input and then does the proper action of adding the new entry. For the roles and authentication in general, I followed the documentation on the fastAPI website so roles were toggled whenever a user was logging in. So in order to test the roles, it is important to choose the “user” and then “read” or “read” and “write”. For logging, I also used an SQLite database which entry holding a unique id, full name/phone number for add/delete entries, the time the action was performed, and a log message containing either list, add, delete by name, or delete by number. As for the authentication, I used Oauth2 as FastAPI had documentation on how to integrate it to my application and meeting the needs for this assignment. With the documentation I was able to integrate Oauth2 and have hashing for passwords, using JWT tokens, have set expiration times for said tokens, and set scopes which were used for the required roles.

**Assumptions made:**

Since I used Oauth2 for my authentication, I needed to make a database to hold the user accounts and since I did not implement any account creation functionality, I manually added some accounts in the database for testing authentication and the required roles. This was probably not the best option, but it will be talked about in more detail in the next section. There were also some assumptions with the regular expressions I had created. For names, I tried to follow the format of the sample inputs given with some leeway in there possibly being initials in the last name or middle name. As for the phone numbers, I mostly tried to accept North American phone numbers as well as meet the other requirements but did not try to accept all different types of international formats.

**Pros/Cons of my approach:**

While the main API (Phonebook specific)is good in my opinion, the authorization API could have been more polished and used another form of authentication that did not require as much work on my end. Something like Microsoft sign-in or Google sign-in may have been a better option since I did have to create a database myself that would store the user login information. The positives, however, led to me using this approach with authentication because there are more things I can control such as keeping track of the database and being able to modify it myself. Continuing with authentication, because the user logging in can chose their own roles, it can be seen as a security risk and can be fixed by making these choices for them instead of letting then choose. As for other parts of the application, the way I created my regular expressions for user input validation may have been a too restrictive since I tried to be as close to the given sample inputs but being a bit flexible with initials. The regular expressions for names were broad as it was built around the 3 different formats, <first middle last>, <first last> or <last, first MI>), but the phone regular expression can be done better to accept varying types of international formats. In addition, for regex testing, I did not make some regular expressions for the logging as I did not have enough time but it very important to have so that is definitely a con to my approach. The other con is that the way I checked that the user is authenticated, active, and has the right roles is by first making the API calls for the user every time they try executing an action. This can easily take up a lot of resources if there are multiple people accessing the application so there can be a simpler and resource efficient way to do the proper checks only once.