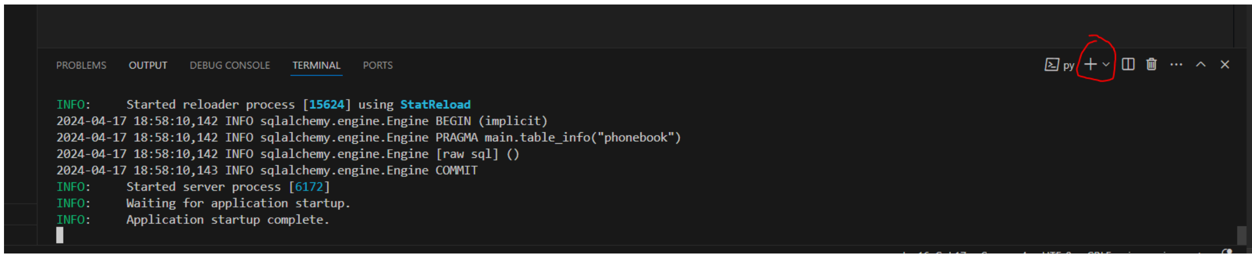
Name: Raed Ali

Mav ID: 1001567598

Report

1. **Instructions for building and running software and unit tests:(copied most of this from the python starter, but I made a few modifications)**
    a. Running the code
        i. Unzip the contents of the zip file I submitted
        ii. There should be a folder named “Code”, this has the python files, text files, and db files you need to make my code run
        iii. Open Visual Studio
        iv. Click on "File" in the top left corner and select "Open Folder".
        v. Navigate to the folder “Code” where the content of my code is
        vi. I added some additional dependencies to “requirements.txt”, so we have install some additional libraries
        vii. Open the "Terminal" tab in Visual Studio by clicking on "View" and then "Terminal".
        viii. Install libraries by using command in terminal: pip install -r requirements.txt
        ix. Open Docker Desktop
        x. To run the app, in a docker container type “docker build -t phonebook .”
        xi. Then type “docker run -p 8000:8000 phonebook”
        xii. Make another terminal to run our unit test script by hitting the plus button button

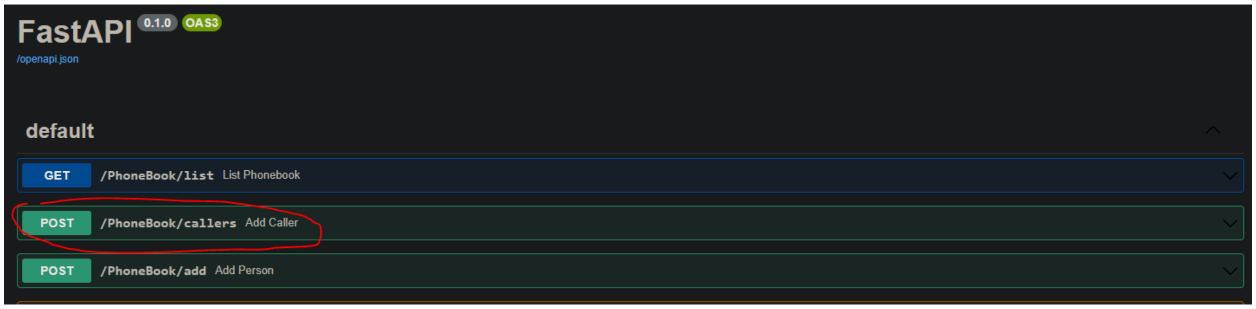


```
INFO:     Started reloader process [15624] using StatReload
2024-04-17 18:58:10,142 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-04-17 18:58:10,142 INFO sqlalchemy.engine.Engine PRAGMA main.table_info("phonebook")
2024-04-17 18:58:10,142 INFO sqlalchemy.engine.Engine [raw sql] ()
2024-04-17 18:58:10,143 INFO sqlalchemy.engine.Engine COMMIT
INFO:     Started server process [6172]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

        xiii. Now we will run our unit tests outside the docker container with our app running on the docker container. Before we move onto the next step my unit tests use the requests library so pip install the requests library by typing “pip install requests” in the newly opened terminal
        xiv. Now type “py test_app.py”, this will run all our unit tests. You should get back an okay, there may be some warnings related to the encryption library but everything should still work
        xv. If you wish to perform tests on my functions outside of the ones provided in my unit test you can use swagger to do so, after starting the server go to this link http://127.0.0.1:8000/docs
        xvi. If you want to add a username, password, and privilege for a person that makes requests to the phonebook database then do the following below in swagger

Name: Raed Ali

Mav ID: 1001567598

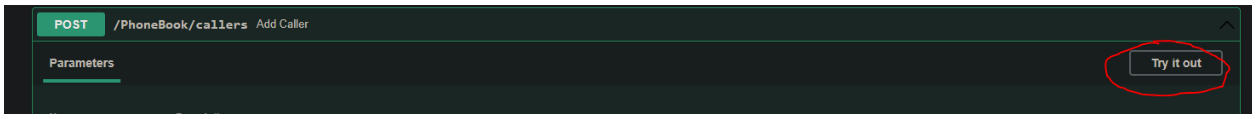

FastAPI 0.1.0 OAS3
/openapi.json

default

GET /PhoneBook/list List Phonebook

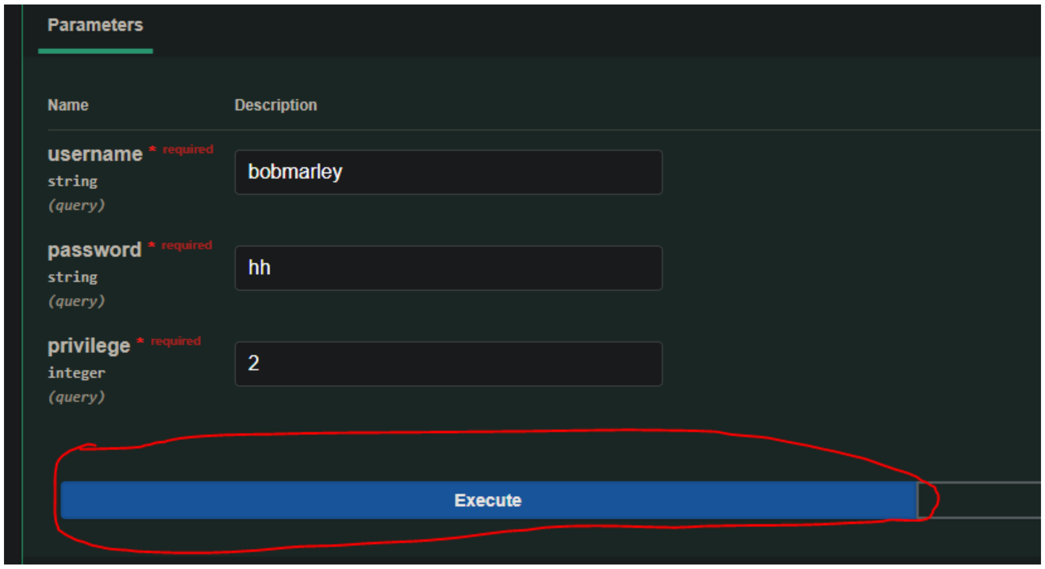
POST /PhoneBook/callers Add Caller

POST /PhoneBook/add Add Person


POST /PhoneBook/callers Add Caller

Parameters

Try it out


Parameters

Name	Description
username * required string (query)	bobmarley
password * required string (query)	hh
privilege * required integer (query)	2

Execute

The password will be encrypted and privilege can be either 1 or 2, 1 indicating the user can read only and 2 indicating they can read and write

2. Description of how my code works

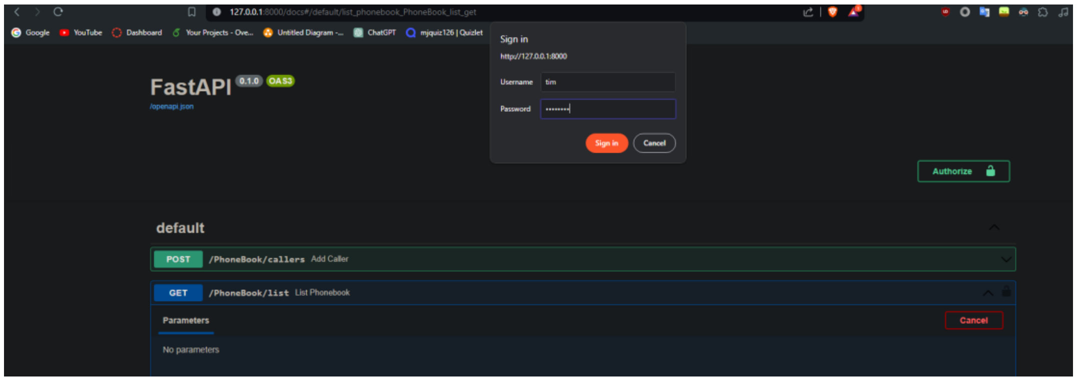
a. app.py (main file)

i. We are using the passlib library to do password hashing and verification. We are using the bcrypt algorithm to hash our passwords. We are creating a CryptContext instance initially using the bcrypt algorithm which will be saved as the variable “password_context” which will later be used to verify a password with its hashed version.

ii. To do authentication we are using HTTPBasicCredentials which is a basic way to do authentication in fast api. So basically we are passing the password and username in the header for each request.

Name: Raed Ali

Mav ID: 1001567598


127.0.0.1:8000/docs#/default/list_phonebook_PhoneBook_list_get

Sign in
http://127.0.0.1:8000
Username: tim
Password: ••••••••

As you can see after I tried to make a get request to list the phonebook it's requiring to enter my user name and password

iii. In order to make a list,add, deleteByName, or deleteByNumber request you need to have a entry in the caller.db table. Not only that but you should also have the appropriate privileges. In swagger you can use the post method of add caller to add a username, password and privilege of a user. When we insert in a new caller the query is parametrized to protect against sql injections and other vulnerabilities

iv. So, the functions list_phonebook, add_person, delete_by_name, delete_by_number now require you to pass in the username and password in the header field, in addition to whatever you were inputting in the original requests. The code searches for the user name in the caller db using a parameterized query of the inputted username and gets the hashed password stored in the database if that user exists. If not it throws and error, but if that user does exist we call verify_password and give it as arguments the user inputted password and the hashed password that was just returned. We use that password_context variable that I talked about earlier to verify the inputted and hashed password.

v. I created the PersonName class for deleting my name, which inherits from BaseModel and has a field for the full name. Similarly I created the PersonNumber class for deleting my number which also inherits from BaseModel and has a field for the phone number.I did this because I noticed I was getting a 422 Unprocessable Entity error after I added the authentication to these 2 requests. I noticed if an object is passed in instead that inherits from BaseModel, this error does not happen.

vi. I added an enum for the Roles, either Read correlated to 1 or ReadandWrite correlated to a 2. This enum is used the caller's privilege for whether they can add or delete entries

vii. I added some regular expression to verify that valid phone numbers and names are inputted into the phonebook.db. I added these regular expressions in the add_person function. I am using 1 regular expression to verify the name and 2 regular expressions to verify the phone number.

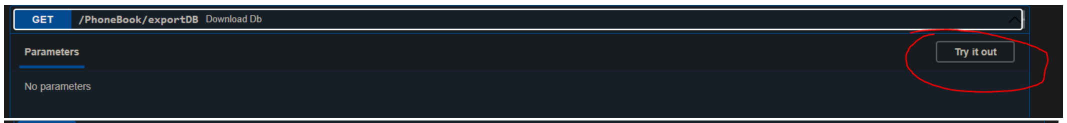
Name: Raed Ali

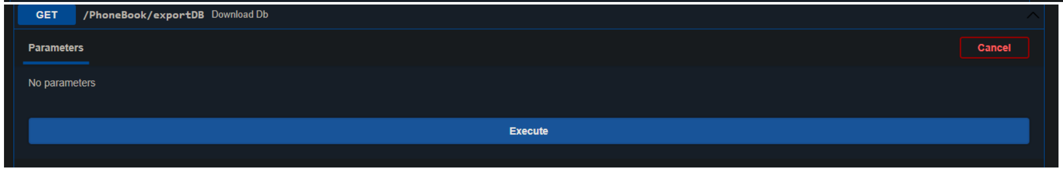
Mav ID: 1001567598

viii. I added a function to log the user name, time stamp, and what action was performed whenever a person adds a person, lists the phonebook, and deletes an entry. I am storing the logs in a file called "AuditLog.csv" with a column for each of those 3 things. The time is being saved in the format Time Stamp(Year-Month-Day Hour:Minute:Seconds).

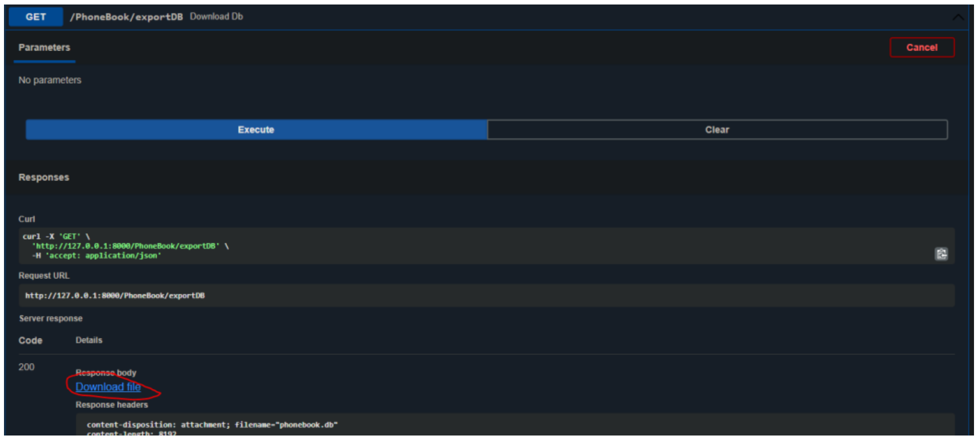
ix. I need to clear the database sometimes to properly test some of the inputs so I created a additional node called clear to wipe the database of the phonebook clear

x. To persist the phonebook db to disk we have to use a get request I created. So we basically open swagger with http://127.0.0.1:8000/docs and then we go to the /PhoneBook/exportDB request. And try it out and then hit execute





Then we click download file to save the database to the disk



Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/PhoneBook/exportDB' \
  -H 'accept: application/json'
```

Request URL

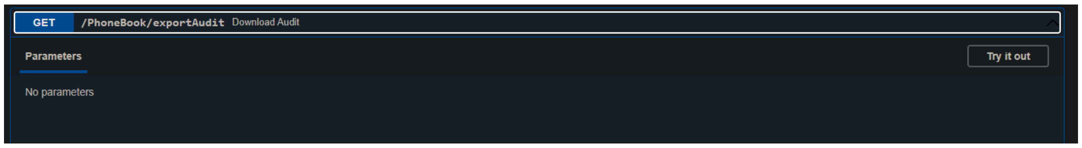
```
http://127.0.0.1:8000/PhoneBook/exportDB
```

Server response

Code	Details
200	Response body Download file Response headers `content-disposition: attachment; filename="phonebook.db"`

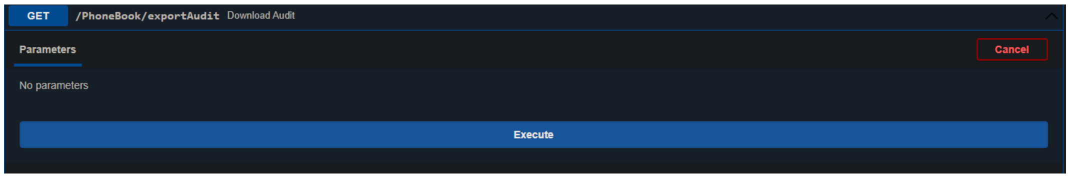
So now we can export our db from our docker container to our system

xi. We use a similar approach for exporting the audit log, so again in swagger we go to the /PhoneBook/exportAudit request. And try it out and then hit execute

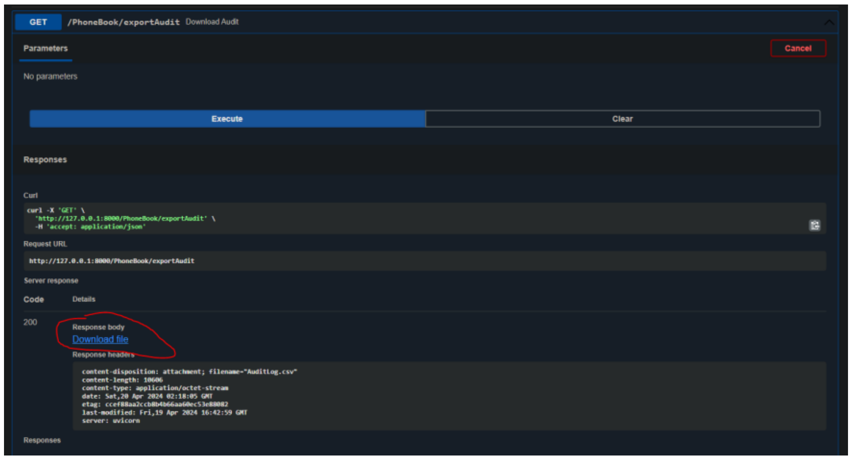


Name: Raed Ali

Mav ID: 1001567598



Then we click download file to save the database to the disk



```
curl -X 'GET' \
  'http://127.0.0.1:8000/PhoneBook/exportAudit' \
  -H 'accept: application/json'
```

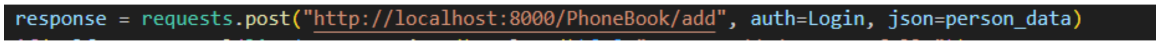
```
http://127.0.0.1:8000/PhoneBook/exportAudit
```

```
content-disposition: attachment; filename="AuditLog.csv"
content-length: 10606
content-type: application/octet-stream
date: Sat,20 Apr 2024 02:18:05 GMT
etag: ccef88aa2ccb8b4b66aa60ec53e88082
last-modified: Fri,19 Apr 2024 16:42:59 GMT
server: uvicorn
```

Now to be able to download the phonebook database file and the audit log to the disk you need to be a user in the database with valid credentials entered. If not you won't be allowed to read this information since you won't have any read permission since you aren't a user.

b. test_app.py (unit test file)

   i. I am using the built in unittest library to do my unit tests for app.py. I am also using the requests library to make specific get and post requests and pass in the appropriate arguments as a json argument. Which also passing in the login information in the authentication field in the header. Like below



```
response = requests.post("http://localhost:8000/PhoneBook/add", auth=Login, json=person_data)
```

   ii. In total there are 9 tests

   iii. I test 3 types of users in my inputs. One a valid username and password ('tim',"password") with read write privleges, an valid username and invalid password ('tim',"notpassword"), a valid username and valid password with only read privileges ('bob',"123"). The usernames and hashed passwords for these users are in the caller.db database.

   iv. I have a couple text files in the directory which will be utilized the unit tests to tests certain types of input

   v. For most of the unit tests I am clearing the phonebook database at the beginning to prevent in errors with any future insertions of data and so there won't be any duplicate entries.

   vi. It should noted with every request that's made you need to pass in the user name and password in the auth field for the request and that username must be in the caller.db database.

Name: Raed Ali

Mav ID: 1001567598

vii. For most of the unit tests we are using the user "tim" and inputting his password which is just "password". Here's how the users in the database are stored.

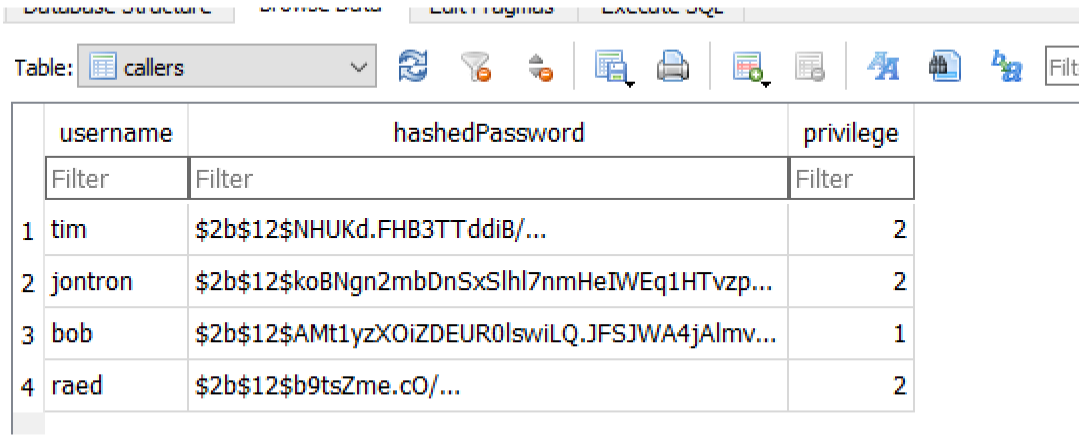


Table: callers

	username	hashedPassword	privilege
1	tim	$2b$12$NHUKd.FHB3TTddiB/...	2
2	jontron	$2b$12$koBNgn2mbDnSxSlhl7nmHeIWEq1HTvzp...	2
3	bob	$2b$12$AMt1yzXOiZDEUR0lswiLQ.JFSJWA4jAlmv...	1
4	raed	$2b$12$b9tsZme.cO/...	2

As you can see our password has been hashed.

viii. The first unit test, test_listBook checks if we get a 200 ok request back when we call it. It also checks the user calling the request has valid login information too and an error would be thrown if the login information was invalid. Almost all the unit tests do this login check to see if the login was valid I won't mention it for the other unit tests unless if we are checking for incorrect login information.

ix. The next unit test test_add_ValidNumbers, tests all the valid phone numbers provided in the assignment pdf. Here we go through a loop and add a person named "Test Name" (since the name doesn't really matter for this test) to the phonebook database, but each entry would have a phone numbers from the valid phone numbers text file. And we check that a person was successfully added each time.

x. Next we check that we can add valid names using the test_add_ValidNames unit test. And again we got these valid names from the assignment pdf. Since the phone numbers have to be unique we have an array of valid phone numbers and assign each of them a name from the names we are testing. And we are checking that a person was successfully added with each entry from the valid names text files.

xi. Then we have the test_InValidNames function which test invalid names provided to us in the assignment pdf. We make sure a error code 400 occurs when we try to add each name.

xii. Similarly, we have the test_InValidNumbers function which tests the invalid phone numbers provided to us in the assignment pdf. We make sure a error code 400 occurs when we try to add each name

xiii. Then we tested the test_delete_by_name function. Here we basically add a person to the phonebook and make sure we did that properly. Then after that we call the put request to delete by name and make sure the person was deleted successfully. Then we add another person and try to delete

Name: Raed Ali

Mav ID: 1001567598

with in correct parameters and make sure that no user was found and thus nobody was deleted.

xiv. Similarly, we had our test_delete_by_number unit test. Where basically did the same thing we did in the previous paragraph but for phone numbers.

xv. Then we have our test_incorrectLogin unit test. Basically here we provide an incorrect password to all the request functions and make sure an Incorrect username or password response was returned for all our request functions.

xvi. And finally, we have our test_user_privleges unit test. Here we use the user bob which only read only privileges and we make sure we can't add or remove any users to the phone book.

3. Assumptions I made

i. I assumed the TA enters the username "tim" and the password "password" when testing his input. Or he will add an entry to the caller.db table and use that.

ii. I assumed that the TA will pip install the requests library to test out my unit test file. Since my unit tests won't work without that library.

iii. I made some assumptions when it came to what a valid phone number is. I assumed the country code can not start with 0 if it's only 2 digits long.

iv. I assumed the country code can at most be 3 digits

v. I assumed that each word in the full name needs to be capitalized

vi. I also noticed for one of the valid phone number inputs there's a 1 in between the country code and area code. I'm not sure why that's allowed but I'm assuming it's okay to do that

vii. I assumed every username in the database can at least read files.

viii. I'm assuming the phone numbers and names provided in the pdf are sufficient enough to test my regular expressions

ix. I assumed it was okay to export the phonebook db by using swagger and the exportDB request

x. I assumed you would have to be a user to be able to export the phonebook to disk. Since you read permissions to read the entries in the phonebook.

4. Pros/Cons of my approach

i. A con from my approach is that I feel like my unit tests could have been more specific in identifying the error. Like whenever the code does things that it's not suppose to I basically divide a number 0 to force an error to stop the rest of the code from the running, however I feel like there's a better way to do that can more specific.

ii. A con from my approach is that I could have used more test input for the phone numbers and names, that could be seen as a limitation to my unit tests.

Name: Raed Ali

Mav ID: 1001567598

iii. A con to my approach is that I feel like my system could be more secure if I was using API keys or bearer tokens. Like my approach just does basic authentication.
iv. One con maybe that we are temporarily storing the unhashed password as local variables in the functions. Now from a lot of the stuff we learned in this class, some other vulnerability may be used to access these local variables in the stack some how. This may or may not create some vulnerabilities.
v. A con to my approach is that I am storing my logs as a csv file. Which may not be the most secure way to store log files.
vi. A con to my approach is that I think if I did more research there are better ways of persisting the data from the docker container to the disc, but I could not figure it out. My method is less automated which is a downside for getting both the log file and the phonebook file.
vii. A pro to my approach is that that it's pretty simple to implement. Just using the basic authentication built into fast api and hashing the passwords.
viii. Another pro to my approach is the user's password's are stored securely since the passwords are encrypted.
ix. Another pro to my approach is that whenever I make an sql query I am using prepared statements to prevent any sql injection attacks.