<u>Password Manager and Generator</u>

Overview:
What is a password manager? It is a computer program which allows users to securely store, generate and manage all passwords in a local or online application.
Why do we need one? In an era where technology advances and every interaction is digitized, it is important to remember the passwords used for each application. Whether it be a gaming application, a shopping account or even your bank account login.
How should we safely store our passwords? There are numerous applications and services offered today. However, what if we wanted to create one for ourselves? A password manager should be paired with a username and URL. It should be encrypted with one master password unlocking the vault. This vault should contain all stored passwords.

The NIST guidelines require that passwords be salted with at least 32 bits of data and hashed with a one-way key derivation function such as Password-Based Key Derivation Function 2(PBKDF2) or balloon. The Function should be iterated as much as possible (at least 10,000 times) without harming server performance.

An interesting topic which was mentioned online, mentioned that a security technique that most password managers use is simply the refusal of any decryption on the server s.. Instead the encryption blob of the password is stored and has the client decrypt it locally . This way the server does not need a master password.

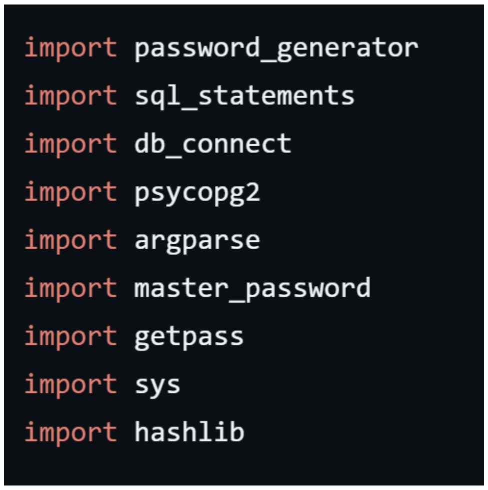
This project will showcase the development of a simple password manager and generator using python and sql
The usage for this case would be a small simple app that will store multiple parts of code which will

Tools and Libraries used:

Tools:	Libraries:
Python	Argparse
PostgreSQL	Psycopg2
	Random
	Arcong2

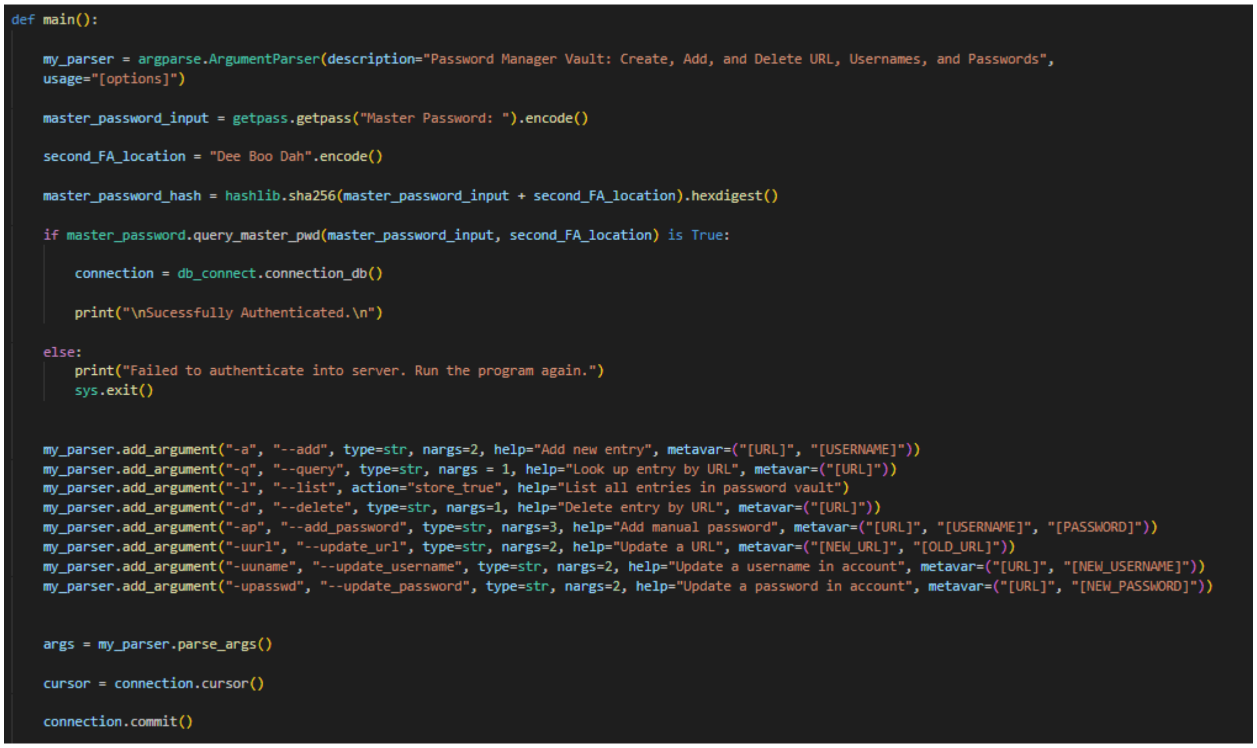
Steps:
Setup environment using PostgreSQL Database Docker Container
Create an Interface with python and psycopg2
Create relational Table Database with URL, Username and Password
Create Password Generator with Random (15 - 20+ character password with uppercase, lowercase, integers and special characters
Manually store username password
Create python functions for INSERT, UPDATE and DELETE
Create terminal interface using argparse library
Create a Master Password with hashing (salt) and combine with an encryption key

Main:
To begin our main python code - we should first add out desired import statements from the list of libraries used



```python
import password_generator
import sql_statements
import db_connect
import psycopg2
import argparse
import master_password
import getpass
import sys
import hashlib
```

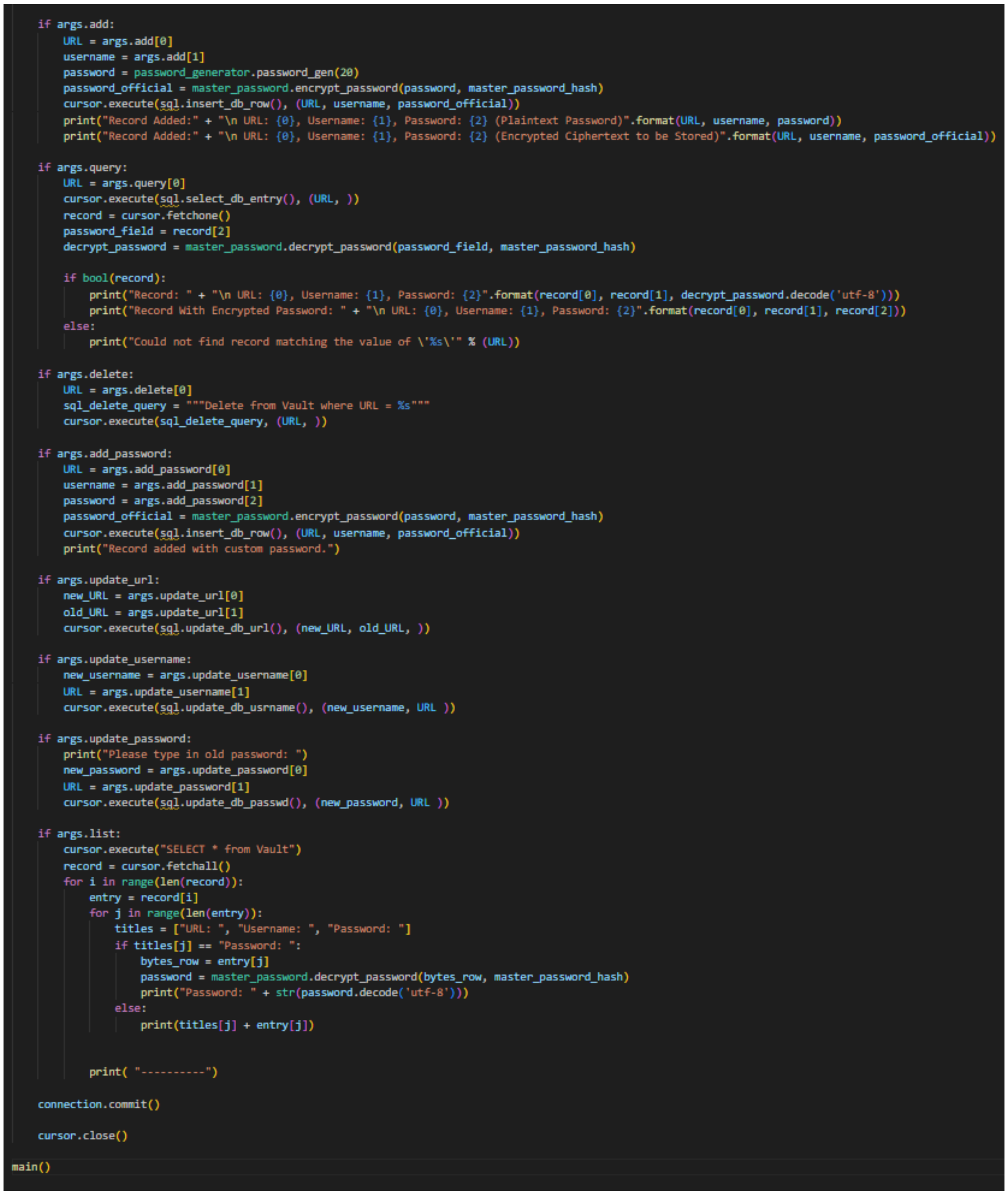
Argument parser:



```python
def main():

    my_parser = argparse.ArgumentParser(description="Password Manager Vault: Create, Add, and Delete URL, Usernames, and Passwords",
    usage="[options]")

    master_password_input = getpass.getpass("Master Password: ").encode()

    second_FA_location = "Dee Boo Dah".encode()

    master_password_hash = hashlib.sha256(master_password_input + second_FA_location).hexdigest()

    if master_password.query_master_pwd(master_password_input, second_FA_location) is True:

        connection = db_connect.connection_db()

        print("\nSucessfully Authenticated.\n")

    else:
        print("Failed to authenticate into server. Run the program again.")
        sys.exit()


    my_parser.add_argument("-a", "--add", type=str, nargs=2, help="Add new entry", metavar=("[URL]", "[USERNAME]"))
    my_parser.add_argument("-q", "--query", type=str, nargs = 1, help="Look up entry by URL", metavar=("[URL]"))
    my_parser.add_argument("-l", "--list", action="store_true", help="List all entries in password vault")
    my_parser.add_argument("-d", "--delete", type=str, nargs=1, help="Delete entry by URL", metavar=("[URL]"))
    my_parser.add_argument("-ap", "--add_password", type=str, nargs=3, help="Add manual password", metavar=("[URL]", "[USERNAME]", "[PASSWORD]"))
    my_parser.add_argument("-uurl", "--update_url", type=str, nargs=2, help="Update a URL", metavar=("[NEW_URL]", "[OLD_URL]"))
    my_parser.add_argument("-uuname", "--update_username", type=str, nargs=2, help="Update a username in account", metavar=("[URL]", "[NEW_USERNAME]"))
    my_parser.add_argument("-upasswd", "--update_password", type=str, nargs=2, help="Update a password in account", metavar=("[URL]", "[NEW_PASSWORD]"))


    args = my_parser.parse_args()

    cursor = connection.cursor()

    connection.commit()
```

The argument parser can be created by adding multiple parsing statements which will help aid the readability of saved username and passwords. Here we have added multiple parsing arguments like add, query, list, and delete. This will allow us to display an output in the command line which would be more readable than everything outputting in a single line.

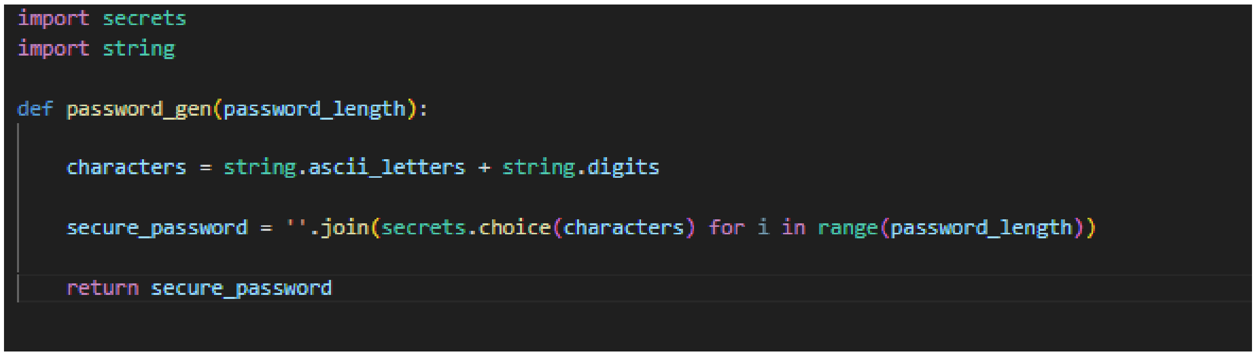
Each parsed argument must be passed through conditional statements to allow us to use each parsing effect correctly. In the project guide, there are multiple if statements which the instructor states as inefficient. However, for this portion of the program it is usable.



```python
    if args.add:
        URL = args.add[0]
        username = args.add[1]
        password = password_generator.password_gen(20)
        password_official = master_password.encrypt_password(password, master_password_hash)
        cursor.execute(sql.insert_db_row(), (URL, username, password_official))
        print("Record Added:" + "\n URL: {0}, Username: {1}, Password: {2} (Plaintext Password)".format(URL, username, password))
        print("Record Added:" + "\n URL: {0}, Username: {1}, Password: {2} (Encrypted Ciphertext to be Stored)".format(URL, username, password_official))

    if args.query:
        URL = args.query[0]
        cursor.execute(sql.select_db_entry(), (URL, ))
        record = cursor.fetchone()
        password_field = record[2]
        decrypt_password = master_password.decrypt_password(password_field, master_password_hash)

        if bool(record):
            print("Record: " + "\n URL: {0}, Username: {1}, Password: {2}".format(record[0], record[1], decrypt_password.decode('utf-8')))
            print("Record With Encrypted Password: " + "\n URL: {0}, Username: {1}, Password: {2}".format(record[0], record[1], record[2]))
        else:
            print("Could not find record matching the value of \'%s\'" % (URL))

    if args.delete:
        URL = args.delete[0]
        sql_delete_query = """Delete from Vault where URL = %s"""
        cursor.execute(sql_delete_query, (URL, ))

    if args.add_password:
        URL = args.add_password[0]
        username = args.add_password[1]
        password = args.add_password[2]
        password_official = master_password.encrypt_password(password, master_password_hash)
        cursor.execute(sql.insert_db_row(), (URL, username, password_official))
        print("Record added with custom password.")

    if args.update_url:
        new_URL = args.update_url[0]
        old_URL = args.update_url[1]
        cursor.execute(sql.update_db_url(), (new_URL, old_URL, ))

    if args.update_username:
        new_username = args.update_username[0]
        URL = args.update_username[1]
        cursor.execute(sql.update_db_usrname(), (new_username, URL ))

    if args.update_password:
        print("Please type in old password: ")
        new_password = args.update_password[0]
        URL = args.update_password[1]
        cursor.execute(sql.update_db_passwd(), (new_password, URL ))

    if args.list:
        cursor.execute("SELECT * from Vault")
        record = cursor.fetchall()
        for i in range(len(record)):
            entry = record[i]
            for j in range(len(entry)):
                titles = ["URL: ", "Username: ", "Password: "]
                if titles[j] == "Password: ":
                    bytes_row = entry[j]
                    password = master_password.decrypt_password(bytes_row, master_password_hash)
                    print("Password: " + str(password.decode('utf-8')))
                else:
                    print(titles[j] + entry[j])


            print( "----------")

    connection.commit()

    cursor.close()

main()
```

Here is the added list of argument parsing, each will provide a different result. For example: the if argos.add statement will ask if the URL argument should be added and if so the password should be generated with a character length of 20. The password will then use the encrypt_password command from the imported library to create a separate hash to the password. Finally it till print two out puts one explaining the plain text of the Username and password along with the formatted URL and secondly it will print out the location of where the encrypted ciphertext is stored.

Password Generator:



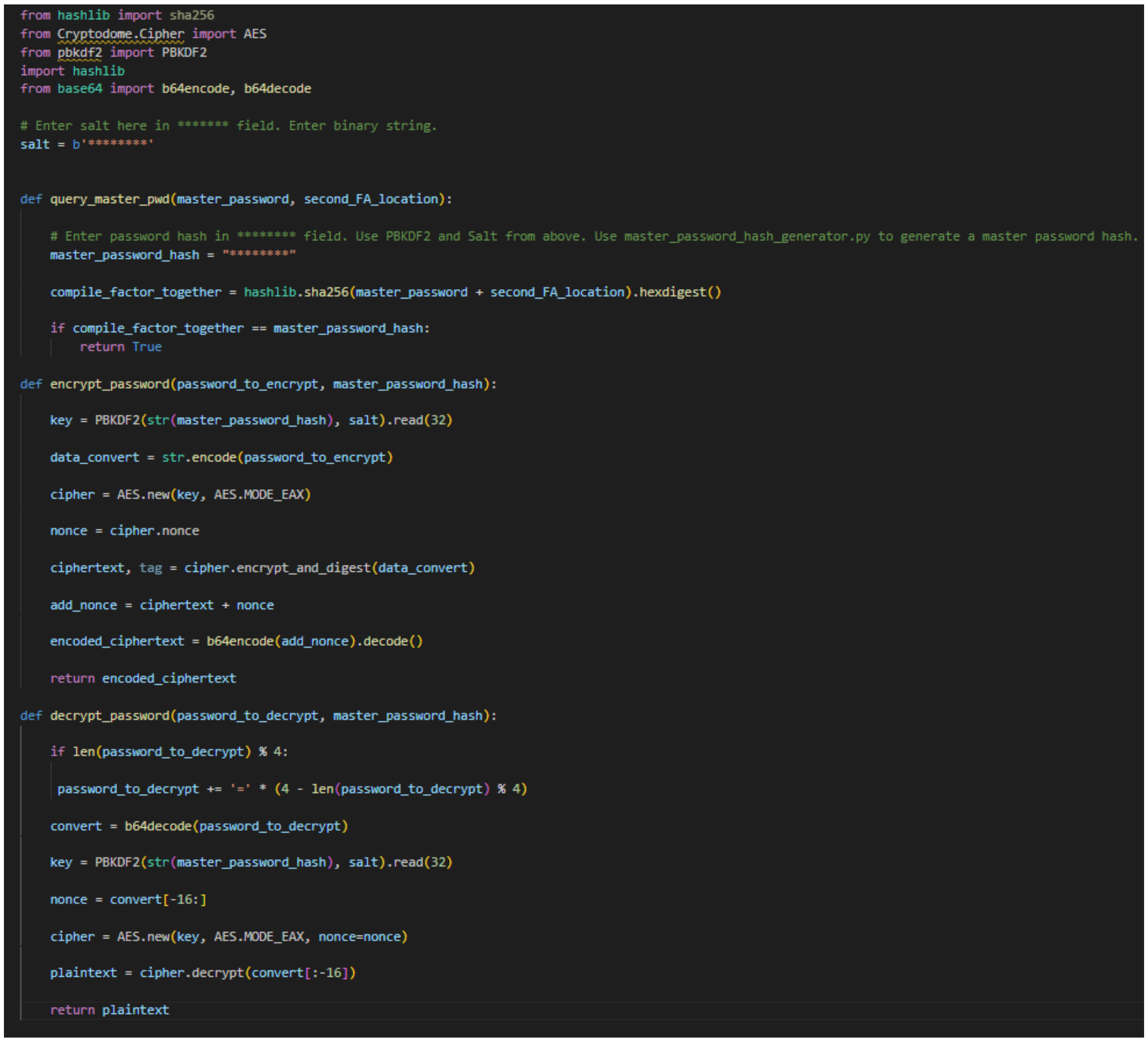
```python
import secrets
import string

def password_gen(password_length):

    characters = string.ascii_letters + string.digits

    secure_password = ''.join(secrets.choice(characters) for i in range(password_length))

    return secure_password
```

Here is a simple look at a password generator. It works by importing secrets and string. We define password_gen with the parameter of password_length
Within this we are able to import characters as a string in ascii letter format. Followed by a string of integers
Then secure_password will join a choice of characters from the secretes library in a range given by the password length
When returned the program will generate a new password using the secrets library.

Continue to next page

Cryptography/Master Password:

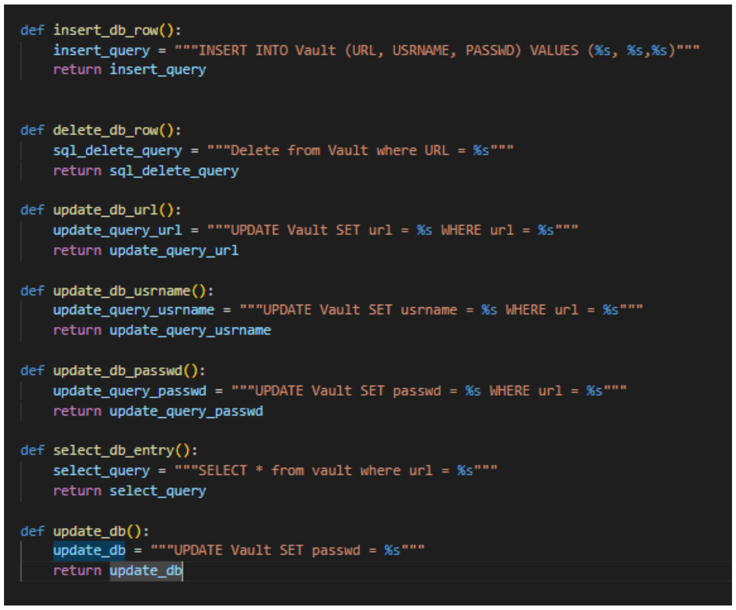
In this portion of the password manager, the code will take a master password into a key derivation algorithm. Here, we used the PBKDF2 algorithm. This will then supply the master password and the salt as the encryption key.When adding your master password input it will verify if it was successfully encrypted.



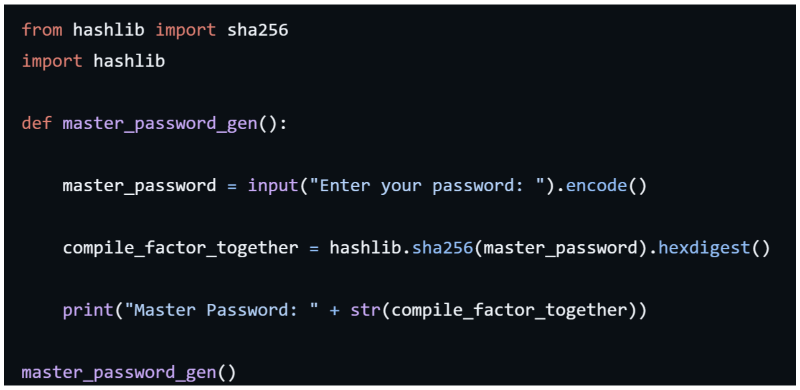
```python
from hashlib import sha256
from Cryptodome.Cipher import AES
from pbkdf2 import PBKDF2
import hashlib
from base64 import b64encode, b64decode

# Enter salt here in ******* field. Enter binary string.
salt = b'********'


def query_master_pwd(master_password, second_FA_location):

    # Enter password hash in ******** field. Use PBKDF2 and Salt from above. Use master_password_hash_generator.py to generate a master password hash.
    master_password_hash = "********"

    compile_factor_together = hashlib.sha256(master_password + second_FA_location).hexdigest()

    if compile_factor_together == master_password_hash:
        return True

def encrypt_password(password_to_encrypt, master_password_hash):

    key = PBKDF2(str(master_password_hash), salt).read(32)

    data_convert = str.encode(password_to_encrypt)

    cipher = AES.new(key, AES.MODE_EAX)

    nonce = cipher.nonce

    ciphertext, tag = cipher.encrypt_and_digest(data_convert)

    add_nonce = ciphertext + nonce

    encoded_ciphertext = b64encode(add_nonce).decode()

    return encoded_ciphertext

def decrypt_password(password_to_decrypt, master_password_hash):

    if len(password_to_decrypt) % 4:

     password_to_decrypt += '=' * (4 - len(password_to_decrypt) % 4)

    convert = b64decode(password_to_decrypt)

    key = PBKDF2(str(master_password_hash), salt).read(32)

    nonce = convert[-16:]

    cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)

    plaintext = cipher.decrypt(convert[:-16])

    return plaintext
```

Master password and password level encryption to individually encrypt each of the passwords stored in the password manager.
This portion of the code will create a secret key and then generate an AES key. This AES key will be used to encrypt and decrypt the two fields.
Written in the code there is a master password hash - however, we should note that the hash or rather master password and password key should never be placed within the code itself. It is considered a vulnerability and can be exploited using the code base history.

Additional Code Lines:



```python
def insert_db_row():
    insert_query = """INSERT INTO Vault (URL, USRNAME, PASSWD) VALUES (%s, %s,%s)"""
    return insert_query


def delete_db_row():
    sql_delete_query = """Delete from Vault where URL = %s"""
    return sql_delete_query

def update_db_url():
    update_query_url = """UPDATE Vault SET url = %s WHERE url = %s"""
    return update_query_url

def update_db_usrname():
    update_query_usrname = """UPDATE Vault SET usrname = %s WHERE url = %s"""
    return update_query_usrname

def update_db_passwd():
    update_query_passwd = """UPDATE Vault SET passwd = %s WHERE url = %s"""
    return update_query_passwd

def select_db_entry():
    select_query = """SELECT * from vault where url = %s"""
    return select_query

def update_db():
    update_db = """UPDATE Vault SET passwd = %s"""
    return update_db
```



```python
from hashlib import sha256
import hashlib

def master_password_gen():

    master_password = input("Enter your password: ").encode()

    compile_factor_together = hashlib.sha256(master_password).hexdigest()

    print("Master Password: " + str(compile_factor_together))

master_password_gen()
```

These additional lines of code are essential for this Manager to work. It allows the sql database to recognize additional effects such as adding, deleting and updating new entries for a set of passwords. It also helps hash the master password to prompt the approval of vault opening.

Conclusion:

Although this may be a non deployable model with many vulnerabilities. It is a great way to start and understand how password managers and password hashing works. In this project, we were able to go over how passwords are generated using a python library, how to connect postgresql using docker container and create a simple sql database to store passwords, introduce methods to encrypt and decrypt password hashes and out put passwords with URLs.