SCHOOL NAME

xxx, xxx

– COMPUTER SCIENCE PROJECT FILE –

Password Manager

ACADEMIC YEAR YYYY-YYYY

Submitted by [Student Name]
Class XII

CERTIFICATE

This is to certify that the project report entitled **“PASSWORD MANAGER”** is an authentic work prepared by **[STUDENT NAME]** of Class XII under my guidance and supervision. This project is in fulfillment of the requirements for appearing in **AISSCE YYYY-YYYY**.

(Internal Examiner) (Principal)

(External Examiner)

DECLARATION

I hereby declare that the project work entitled:

“PASSWORD MANAGER”

was developed under the supervision of our subject teacher for the partial fulfillment of

ALL INDIA SENIOR SCHOOL CERTIFICATE EXAMINATION (AISSCE)

ACKNOWLEDGEMENT

It is with great pleasure I am expressing my gratitude to those who helped me a long way in completing my project.

I wholeheartedly thank my computer teacher **[TEACHER]** for her guidance and support throughout the development of the project and it is of great pleasure to express my deep sense of gratitude to the principal **[PRINCIPAL]**, teachers and friends for their inspiration and constant help during the project.

I thank all my classmates and friends who helped me during the development of this project with their constructive criticism and advice. I would also like to thank my parents for encouraging me during the course of this project.

Finally I would like to thank **CBSE** for this wonderful opportunity.

Overview

PyPass is a terminal based password manager written in Python 3.8 with MySQL database connectivity.

The majority of people use very weak passwords and reuse them on different websites. Password reuse is a serious problem because of the many password leaks that occur each year, even on large websites. When your password leaks, malicious individuals have an email address, username, and password combination they can try on other websites. If you use the same login information everywhere, a leak at one website could give people access to all your accounts. If someone gains access to your email account in this way, they could use password-reset links to access other websites, like your online banking or social media accounts.

What is a Password Manager?

A password manager is a program that houses all your passwords, as well as other information, in one convenient location with one master password. The benefits to using a password manager are:

- **You don’t have to memorize all your passwords** and only need to remember one single password that is used to access the password manager.
    - This lets you use really strong and complex passwords without worrying about forgetting them,
- **The password manager encrypts all the data** that is stored so that it cannot be accessed by anyone else.
    - This is better in comparison with writing down passwords on paper which is considered unsafe.
- **It can be used to keep track of your accounts** so that you are aware of the various services you have signed up for.

What is Python?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It is a very simple language that makes it very attractive for rapid application development, as well as for use as a scripting or glue language to connect existing components together.

Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

It was designed by Guido van Rossum in 1991 and has become one of the most popular programming languages in the world. This password manager is built completely in Python because of its simplicity and flexibility.

What is SQL?

Structured Query Language (SQL) is a standardized programming language that is used to manage relational databases and perform various operations on the data in them. Initially created in the 1970s, SQL is regularly used not only by database administrators, but also by developers writing data integration scripts and data analysts looking to set up and run analytical queries.

SQL queries and other operations take the form of commands written as statements and are aggregated into programs that enable users to add, modify or retrieve data from database tables.

The ease of use of SQL and the availability of several modules that implement SQL functionality into Python (like MySQL, PostgreSQL, SQLite, etc...) makes it an ideal choice for the storage of passwords and information by the password manager.

How are passwords secured?

Storing sensitive information such as passwords and credentials must be done with extreme care so as to avoid it from being compromised. Thus, the proper way to store such data is by encrypting it.

PyPass uses the **Advanced Encryption Standard (AES-256)** for the encryption and decryption of user credentials. AES-256 is an extremely strong encryption algorithm that is implemented in Python through the PyCryptoDome module. The master password is used as the encryption key.

Goals

1. Interface Python with MySQL to allow storage of user credentials..
2. Encrypt passwords using AES-256 to keep the passwords safe from malicious entities.
3. Provide an user-friendly console interface to access the stored credentials using Python.
4. Let the user add, read, update, or delete entries through the console interface.

Specifications

- ➔ [Language] **Python** 3.6.0+
- ➔ [Database] **MySQL** 8.0+
- ➔ [OS] **Microsoft Windows** 7+

Modules

The password manager relies on several python modules as dependencies for functionality. Some of these are part of the standard python library that comes built-in while the others can be installed using PIP (Package Installer for Python).

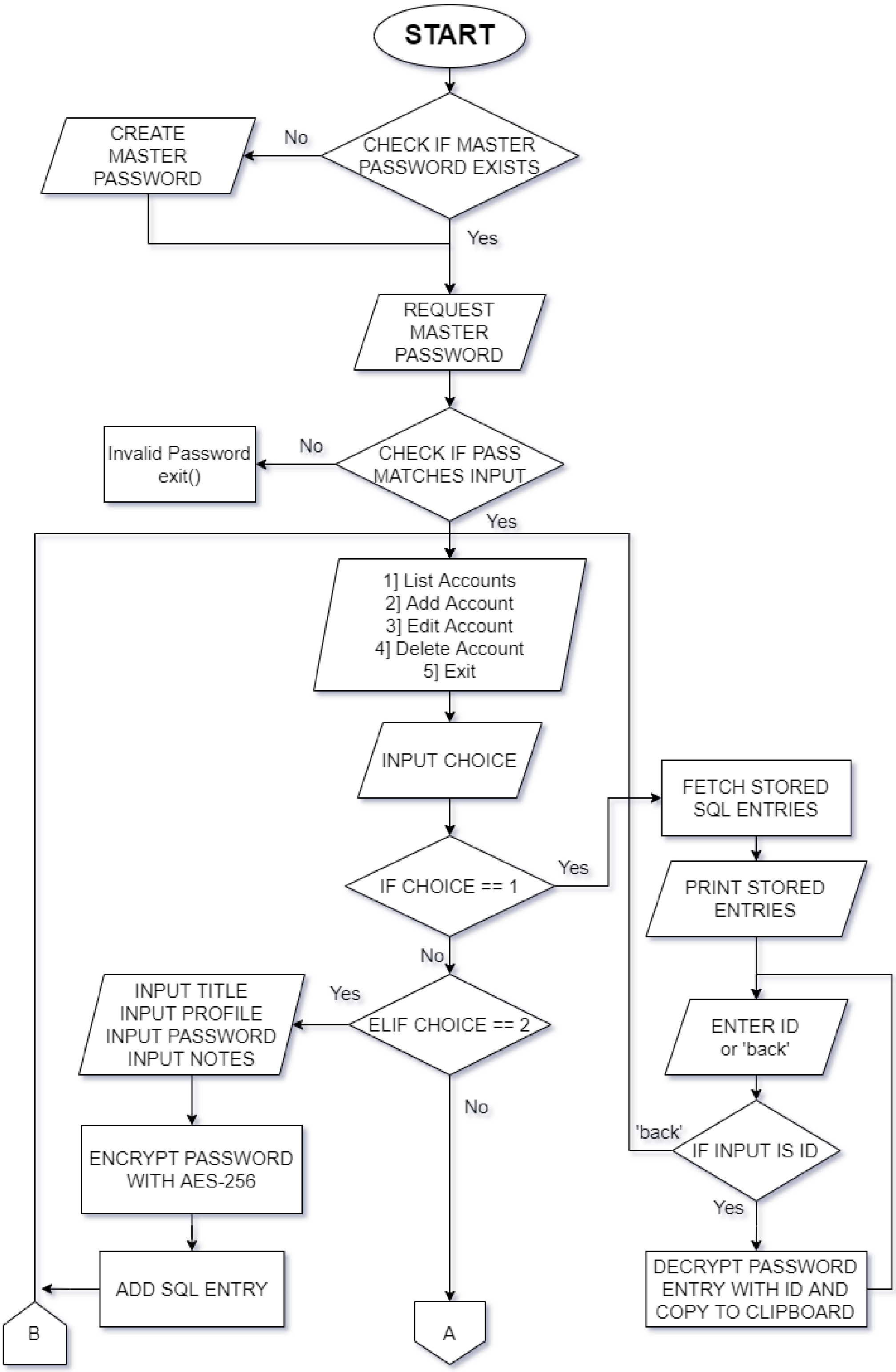
- **os**
  - To manipulate the Windows console by setting window titles, changing colors, and clearing the screen.
- **mysql**
  - To safely store and retrieve user passwords through an SQL database.
- **time**
  - To get accurate time values in order to store information about when passwords were created and/or modified.
- **random**
  - To generate random combinations of characters used for unique IDs.
- **pyperclip**
  - To access and manipulate the clipboard. This is used to copy passwords to clipboard for ease of use when a user requests it.
- **pycryptodome**
  - To securely encrypt passwords using the AES-256 algorithm and to hash the master password using the SHA-256 algorithm.
- **base64**
  - To encode encrypted passwords for proper storage in the SQL database.

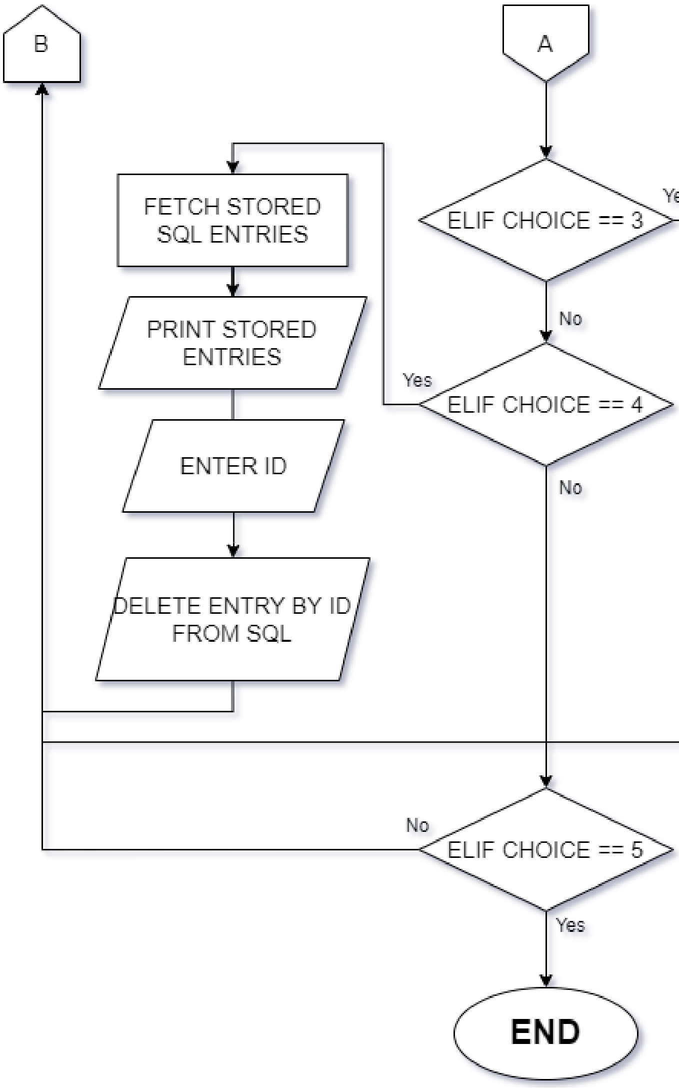
SOURCE CODE

```
## Import Necessary Modules.
import os
import mysql.connector
import time
import random
import pyperclip

## Hashing, Encoding, & Encryption.
from Crypto.Hash import SHA256
from Crypto.Cipher import AES
from Crypto.Protocol.KDF import bcrypt
import base64

# Connect to local database and retrieve cursor.
DATABASE = mysql.connector.connect(
    host="localhost",
    user="user",
    password="password"
)
DB_CURSOR = DATABASE.cursor()

DB_CURSOR.execute("CREATE DATABASE IF NOT EXISTS userpasswds")
DB_CURSOR.execute("USE userpasswds")

# Check if entries table is already defined.
query = DB_CURSOR.execute("SHOW TABLES LIKE 'entries'")
result = DB_CURSOR.fetchone()
if not result:
    # If not, create the table.
    DB_CURSOR.execute("CREATE TABLE entries(id VARCHAR(4), title VARCHAR(32),
profile VARCHAR(32), password VARCHAR(512), notes VARCHAR(512), timestamp
VARCHAR(18))")

# Check if config table is already defined.
query = DB_CURSOR.execute("SHOW TABLES LIKE 'config'")
result = DB_CURSOR.fetchone()
if not result:
    # If not, create the table.
    DB_CURSOR.execute("CREATE TABLE config(keyName VARCHAR(128), keyValue
VARCHAR(512))")
    DB_CURSOR.execute("INSERT INTO config VALUES(%s, %s)", ('masterPassword',
''))
    DATABASE.commit()

# Global Variables
MASTER_PASS = ''
```

```python
# Utility functions to hash, encrypt, and decrypt passwords.
def hash_SHA256(text):
    """Hash an input string using the SHA-256 hashing algorithm."""
    return SHA256.new(text.encode()).hexdigest()

def derive_key(password):
    """Derive a fixed length key from a password string."""
    b64pwd = base64.b64encode(SHA256.new(password.encode()).digest())
    bcrypt_hash = bcrypt(b64pwd, 12, b'salt for bcrypt1')
    return bcrypt_hash[:32]

def encrypt_AES(text, key):
    """Encrypt a string using the AES-256 encryption algorithm."""
    cipher = AES.new(key, AES.MODE_CFB, key[::-1][:16])
    return base64.b64encode(cipher.encrypt(text.encode())).decode()

def decrypt_AES(text, key):
    """Decrypt an AES-256 encrypted string from key."""
    cipher = AES.new(key, AES.MODE_CFB, key[::-1][:16])
    return cipher.decrypt(base64.b64decode(text)).decode()

# Utility functions to manage the windows console.
def cls():
    """Clear the console screen."""
    os.system('cls')

def color(clr):
    """Set the console text color."""
    os.system(f'color {clr}')

def title(text):
    """Set the console window title."""
    os.system(f'title {text}')

def pause(message):
    """Pause the console window and await keyboard input."""
    print(message)
    os.system('pause>nul')

def print_header():
    """Print a large fancy heading text."""
    print()
    print(r" ██████╗ ██╗   ██╗██████╗  █████╗ ███████╗███████╗")
    print(r" ██╔══██╗╚██╗ ██╔╝██╔══██╗██╔══██╗██╔════╝██╔════╝")
    print(r" ██████╔╝ ╚████╔╝ ██████╔╝███████║███████╗███████╗")
    print(r" ██╔═══╝   ╚██╔╝  ██╔═══╝ ██╔══██║╚════██║╚════██║")
    print(r" ██║        ██║   ██║     ██║  ██║███████║███████║")
    print(r" ╚═╝        ╚═╝   ╚═╝     ╚═╝  ╚═╝╚══════╝╚══════╝")

def main():
    """Driver function to enter the application."""
    cls()
    color('c')
    title("Password Manager - Authorization")
    print("=====================================================")
    print("=====================================================")
    print_header()
    print("  A password manager written in pure Python.")
    print("=====================================================")
    print("=====================================================")
    print()

    # Get master password from database.
    query = DB_CURSOR.execute("SELECT keyValue FROM config WHERE keyName =
'masterPassword'")
    master_password = DB_CURSOR.fetchone()[0]

    # If no master password is found. Make user generate one.
    if not master_password:
        entry_password = ''
        while True:
            entry_password = input("  Create Master Password: ")
            entry_passconfirm = input("  Re-Enter New Master Password
(Confirmation): ")
            if entry_password == entry_passconfirm:
                break
            else:
                print("{ERROR} Password Mismatch. Enter Again.\n")

        master_password = hash_SHA256(entry_password)
        DB_CURSOR.execute("UPDATE config SET keyValue=%s WHERE
keyName='masterPassword'", (master_password, ))
        DATABASE.commit()
        print("New Master Password Set.\n")

    mpass = input("  Enter Master Password: ")

    # Verify master password.
    mpass_hashed = hash_SHA256(mpass)
    if mpass_hashed != master_password:  # check against password hash
        print("Access Denied. Incorrect Master Password.")
        if DATABASE.is_connected():
            DB_CURSOR.close()
            DATABASE.close()
        exit()  # --> kick the user out of the application
    else:
        MASTER_PASS = master_password
        menu_showMain()  # --> allow the user into the application

def menu_showMain():
    cls()
    color('c')
    title("Password Manager - Main")
    print_header()
    print()
    print("  1] List Accounts")
    print("  2] Add Account")
    print("  3] Edit Account")
    print("  4] Delete Account")
    print("  5] Exit")
    print()
    while True:
        ch = input(">> ").strip().lower()

        if ch in ('1', 'list', 'l'):
            return menu_showList()
        elif ch in ('2', 'add', 'a'):
            return menu_showAdd()
        elif ch in ('3', 'edit', 'e'):
            return menu_showEdit()
        elif ch in ('4', 'delete', 'd', 'del'):
            return menu_showDelete()
        elif ch in ('5', 'exit', 'quit', 'q'):
            print("Exiting...")
            if DATABASE.is_connected():
                DB_CURSOR.close()
                DATABASE.close()  # safely close connection with database
            return exit()
        else:
            print("{ERROR} Invalid Option. Try Again. [1/2/3/4/5]")
            print()

def menu_showList():
    cls()
    color('c')
    title("Password Manager - List Accounts")
    print()
    print(r"  ╦  ╦╔═╗╔╦╗")
    print(r"  ║  ║╚═╗ ║ ")
    print(r"  ╩═╝╩╚═╝ ╩ ")
    print(r"  ──────────")
    print()

    query = DB_CURSOR.execute("SELECT * FROM entries")
    results = DB_CURSOR.fetchall()
    if len(results) == 0:
        print("  No profiles are saved yet.")
        pause("\n  << Back [press any key]")
        return menu_showMain()

    for c, entry in enumerate(results, 1):
        print(F"[ID #{c}] {entry[1]}")
        print("Profile: ", entry[2])
        print("Password: ", '[Hidden]')
        print("Notes: ", entry[4])
        print(". . . . . ")
    print()
    while True:
        print("Enter ID to copy password to clipboard.")
        print("'back' to go back.")

        inp = input(">> ").strip().lower()
        if inp == 'back':
            print()
            return menu_showMain()
        elif inp.isdigit():
            intinp = int(inp) - 1
            if intinp < len(results) and intinp > -1:
                encrypted_passwd = results[intinp][3]
                # Decrypt the encrypted password using key from master password.
                passwd = decrypt_AES(encrypted_passwd, derive_key(MASTER_PASS))
                # Copy the password to the user's clipboard.
                pyperclip.copy(passwd)
                print(f"Password for {results[intinp][2]} ({results[intinp][1]})
copied to clipboard.")
                print()

def menu_showAdd():
    cls()
    color('c')
    title("Password Manager - Add Account")
    print()
    print(r"  ╔═╗╔╦╗╔╦╗")
    print(r"  ╠═╣ ║║ ║║")
    print(r"  ╩ ╩═╩╝═╩╝")
    print(r"  ──────────")
    print()

    entry_title = input("Enter Title: ")
    entry_profile = input("Enter Username/Profile: ")
    entry_password = ''
    while True:
        entry_password = input("Enter Password: ")
        entry_passconfirm = input("Enter Password (Confirmation): ")
        if entry_password == entry_passconfirm:
            break
        else:
            print("{ERROR} Password Mismatch. Enter Again.\n")
    entry_notes = input("Additional Notes: ")
    last_modified = str(time.time())

    # Encrypt the password for secure storage.
    password_encrypted = encrypt_AES(entry_password, derive_key(MASTER_PASS))
    id = entry_title[0] + str(random.randint(100, 999))
    data = (id, entry_title, entry_profile, password_encrypted, entry_notes, 
last_modified)
    DB_CURSOR.execute("INSERT INTO entries VALUES(%s, %s, %s, %s, %s, %s)", 
data)
    DATABASE.commit()
    print("Successfully added entry.")

    pause("\n  << Back [press any key]")
    menu_showMain()

def menu_showEdit():
    cls()
    color('c')
    title("Password Manager - Edit Account")
    print()
    print(r"  ╔═╗╔╦╗╦╔╦╗")
    print(r"  ║╣  ║║║ ║ ")
    print(r"  ╚═╝═╩╝╩ ╩ ")
    print(r" ──────────")
    print()

    query = DB_CURSOR.execute("SELECT * FROM entries")
    results = DB_CURSOR.fetchall()
    if len(results) == 0:
        print("  No profiles are saved yet.")
        pause("\n  << Back [press any key]")
        return menu_showMain()

    for c, entry in enumerate(results, 1):
        print(f"[ID #{c}] {entry[1]} | {entry[2]}")
    print()

    while True:
        inp = input("Enter ID (to Edit): ").strip()
        if inp.isdigit():
            intinp = int(inp) - 1
            if intinp < len(results) and intinp > -1:
                entry_data = list(results[intinp])
                uid = entry_data[0]

                print(f"Editing Entry {intinp} [#{uid}] (Leave blank on fields 
you do not wish to edit.)")
                print("\nTitle: ", entry_data[1])
                new_title = input("Title (New): ").strip()
                print("\nUsername/Profile:", entry_data[2])
                new_profile = input("Username/Profile (New): ").strip()
                print("\nPassword:", '[hidden]')
                new_password = ''
                while True:
                    new_password = input("Password (New): ")
                    if not new_password:
                        break
                    new_passconfirm = input("Password (Confirmation): ")
                    if new_password == new_passconfirm:
                        break
                    else:
                        print("{ERROR} Password Mismatch. Enter Again.\n")
                print("\nNotes:", entry_data[4])
                new_notes = input("Notes (New):").strip()

                if new_title:
                    entry_data[1] = new_title
                if new_profile:
                    entry_data[2] = new_profile
                if new_password:
                    encrypted_password = encrypt_AES(new_password,
derive_key(MASTER_PASS))
                    entry_data[3] = encrypted_password
                if new_notes:
                    entry_data[4] = new_notes
                entry_data[5] = str(time.time())
                data = tuple(entry_data)

                DB_CURSOR.execute("DELETE FROM entries WHERE id=%s", (uid, ))
                DB_CURSOR.execute("INSERT INTO entries VALUES(%s, %s, %s, %s,
%s, %s)", data)
                DATABASE.commit()

                print("Successfully edited entry.")
                break
            else:
                print("{ERROR} Invalid ID: Out of range.\n")
        else:
            break

    pause("\n  << Back [press any key]")
    menu_showMain()

def menu_showDelete():
    cls()
    color('c')
    title("Password Manager - Delete Account")
    print()
    print(r"  ╔╦╗╔═╗╦  ╔═╗╔╦╗╔═╗")
    print(r"   ║║║╣ ║  ║╣  ║ ║╣ ")
    print(r"  ═╩╝╚═╝╩═╝╚═╝ ╩ ╚═╝")
    print(r"  ──────────────────")
    print()

    query = DB_CURSOR.execute("SELECT * FROM entries")
    results = DB_CURSOR.fetchall()
    if len(results) == 0:
        print("  No profiles are saved yet.")
        pause("\n  << Back [press any key]")
        return menu_showMain()

    for c, entry in enumerate(results, 1):
        print(f"[ID #{c}] {entry[1]} | {entry[2]}")
    print()

    while True:
        inp = input("Enter ID (to Delete): ").strip()
        if inp.isdigit():
            intinp = int(inp) - 1
            if intinp < len(results) and intinp > -1:
                uid = results[intinp][0]
                DB_CURSOR.execute("DELETE FROM entries WHERE id=%s", (uid, ))
                DATABASE.commit()
                print("Successfully deleted entry.")
                break
            else:
                print("Invalid ID: Out of range.\n")
        else:
            break

    pause("\n  << Back [press any key]")
    menu_showMain()

# Call the main() function to enter the application.
main()
```

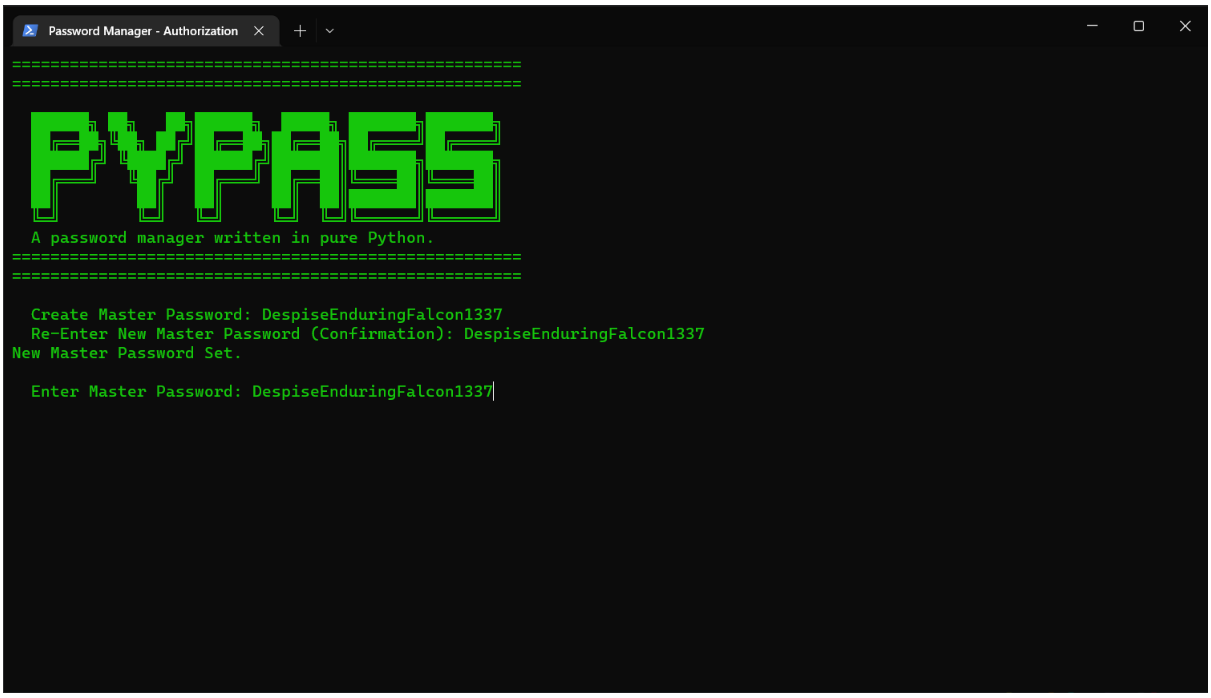
LOGICAL FLOWCHART




B
A
ELIF CHOICE == 3
Yes
FETCH STORED SQL ENTRIES
PRINT STORED ENTRIES
ENTER ID
INPUT TITLE
INPUT PROFILE
INPUT PASSWORD
INPUT NOTES
UPDATE SQL ENTRIES
No
ELIF CHOICE == 4
Yes
FETCH STORED SQL ENTRIES
PRINT STORED ENTRIES
ENTER ID
DELETE ENTRY BY ID FROM SQL
No
ELIF CHOICE == 5
No
Yes
END

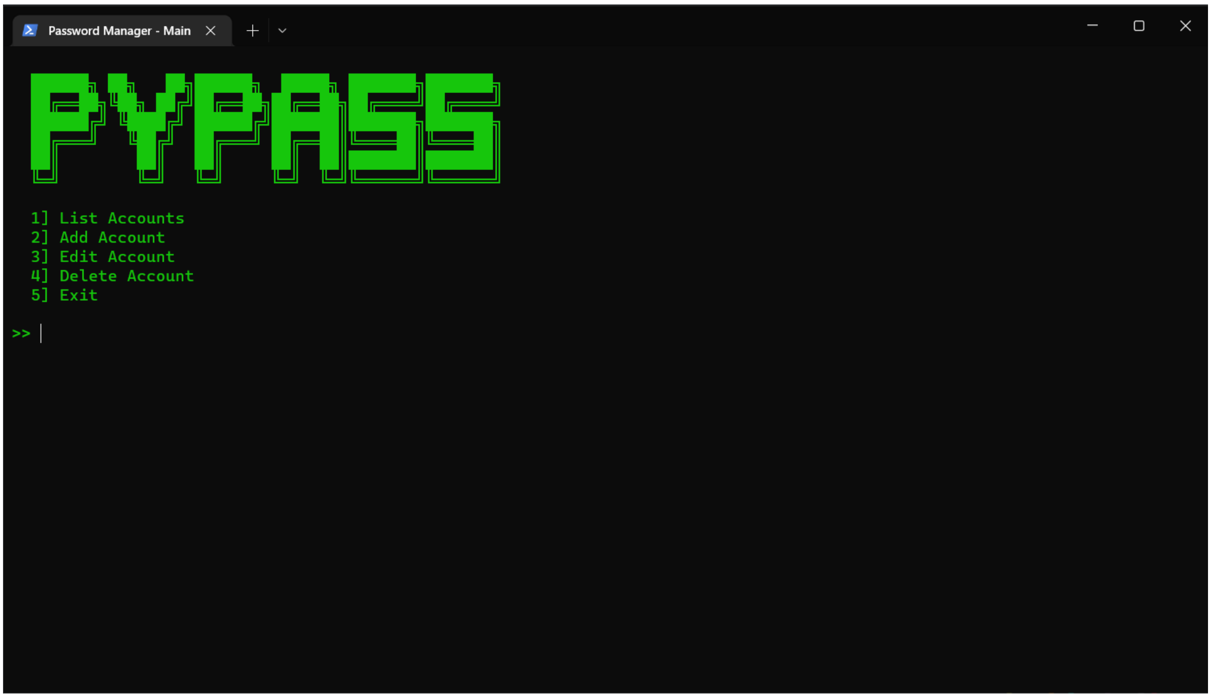
PROGRAM SCREENSHOTS

Master Password Prompt / Creation



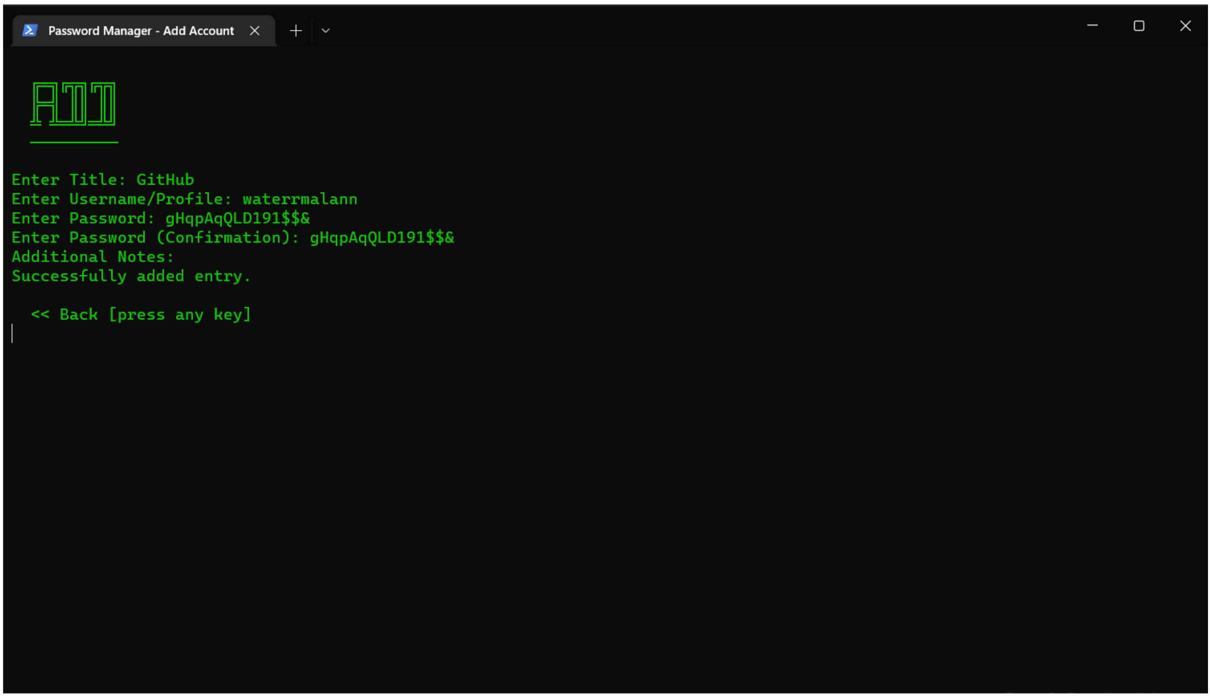
```
=====================================================
=====================================================

  A password manager written in pure Python.
=====================================================
=====================================================

  Create Master Password: DespiseEnduringFalcon1337
  Re-Enter New Master Password (Confirmation): DespiseEnduringFalcon1337
New Master Password Set.

  Enter Master Password: DespiseEnduringFalcon1337
```

Main Menu



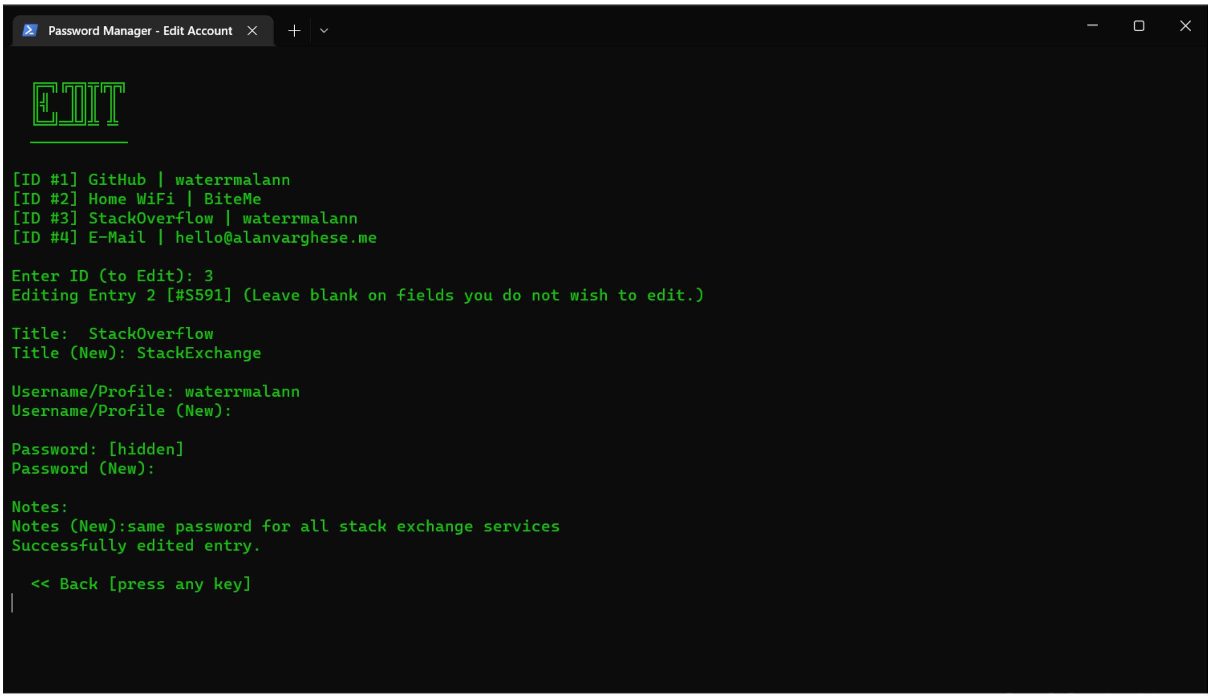
```
  1] List Accounts
  2] Add Account
  3] Edit Account
  4] Delete Account
  5] Exit

>> 
```

Add Account



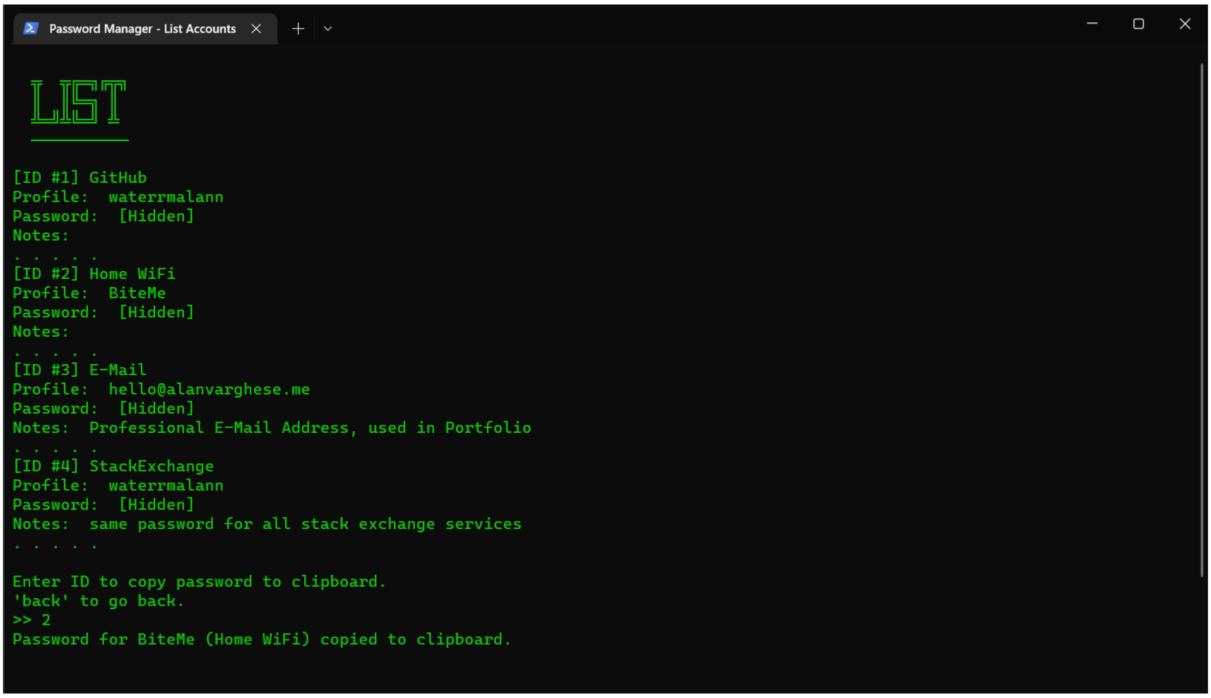
```
Enter Title: GitHub
Enter Username/Profile: waterrmalann
Enter Password: gHqpAqQLD191$$&
Enter Password (Confirmation): gHqpAqQLD191$$&
Additional Notes:
Successfully added entry.

  << Back [press any key]
```

Edit Account



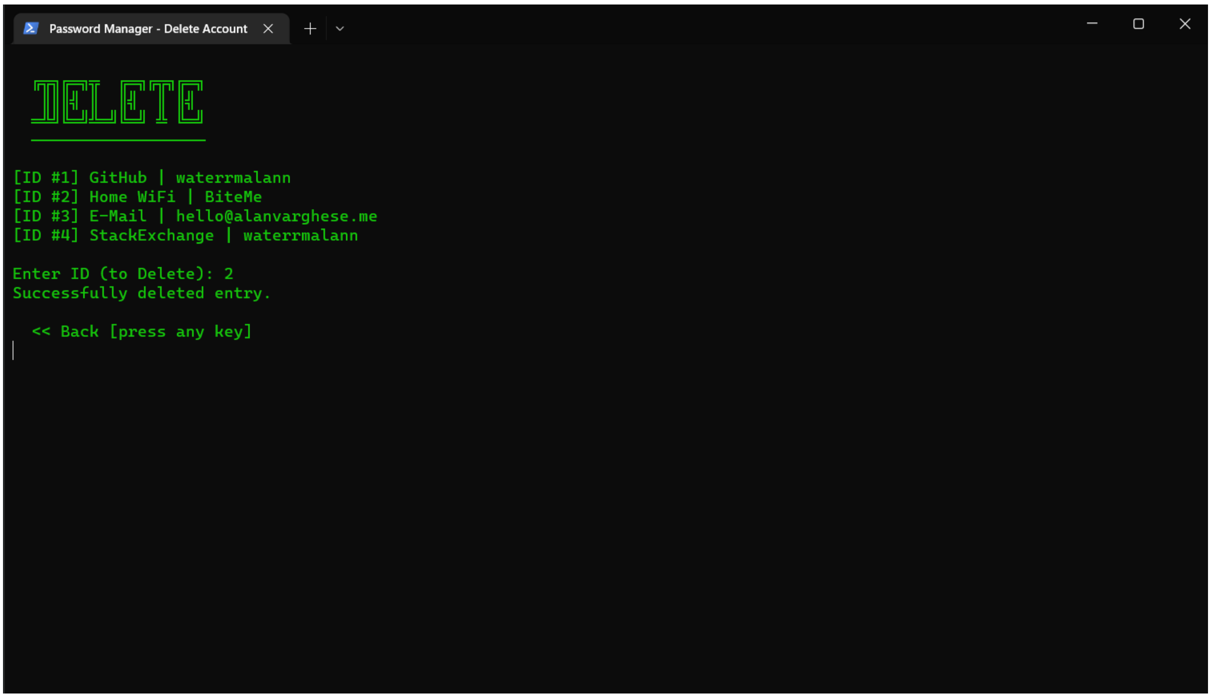
```
[ID #1] GitHub | waterrmalann
[ID #2] Home WiFi | BiteMe
[ID #3] StackOverflow | waterrmalann
[ID #4] E-Mail | hello@alanvarghese.me

Enter ID (to Edit): 3
Editing Entry 2 [#S591] (Leave blank on fields you do not wish to edit.)

Title:  StackOverflow
Title (New): StackExchange

Username/Profile: waterrmalann
Username/Profile (New):

Password: [hidden]
Password (New):

Notes:
Notes (New):same password for all stack exchange services
Successfully edited entry.

  << Back [press any key]
```

List Accounts



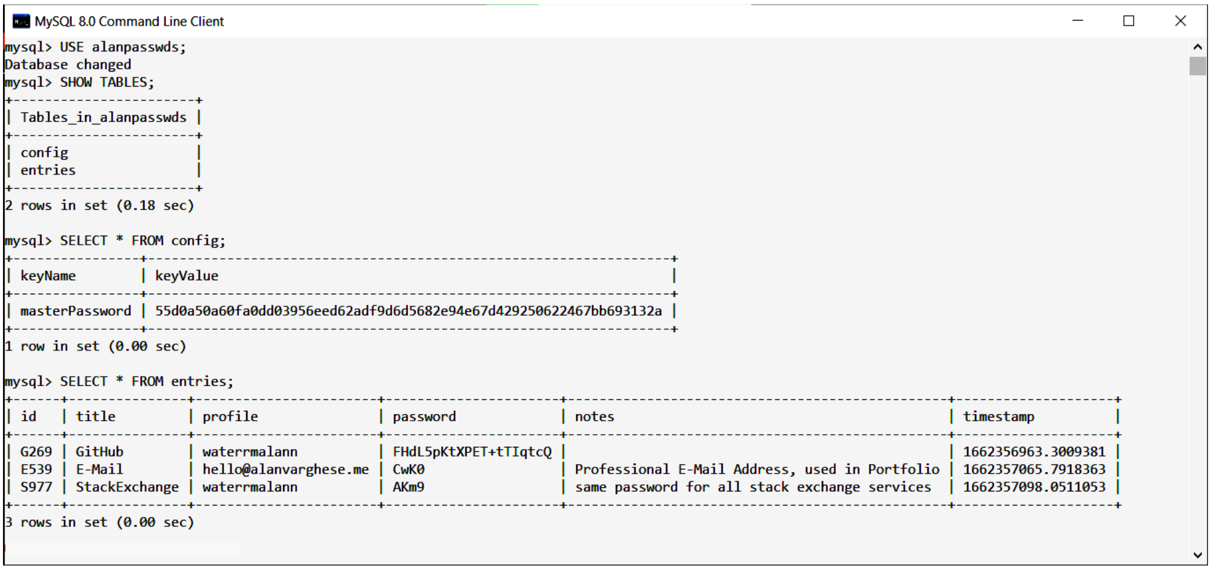
```
[ID #1] GitHub
Profile:  waterrmalann
Password:  [Hidden]
Notes:
. . . . . .
[ID #2] Home WiFi
Profile:  BiteMe
Password:  [Hidden]
Notes:
. . . . . .
[ID #3] E-Mail
Profile:  hello@alanvarghese.me
Password:  [Hidden]
Notes:  Professional E-Mail Address, used in Portfolio
. . . . . .
[ID #4] StackExchange
Profile:  waterrmalann
Password:  [Hidden]
Notes:  same password for all stack exchange services
. . . . . .

Enter ID to copy password to clipboard.
'back' to go back.
>> 2
Password for BiteMe (Home WiFi) copied to clipboard.
```

Delete Account



```
[ID #1] GitHub | waterrmalann
[ID #2] Home WiFi | BiteMe
[ID #3] E-Mail | hello@alanvarghese.me
[ID #4] StackExchange | waterrmalann

Enter ID (to Delete): 2
Successfully deleted entry.

  << Back [press any key]
```

MySQL DATABASE



```
mysql> USE alanpasswds;
Database changed
mysql> SHOW TABLES;
+-----------------------+
| Tables_in_alanpasswds |
+-----------------------+
| config                |
| entries               |
+-----------------------+
2 rows in set (0.18 sec)

mysql> SELECT * FROM config;
+----------------+------------------------------------------------------------------+
| keyName        | keyValue                                                         |
+----------------+------------------------------------------------------------------+
| masterPassword | 55d0a50a60fa0dd03956eed62adf9d6d5682e94e67d429250622467bb693132a |
+----------------+------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM entries;
+------+---------------+-----------------------+----------------------+------------------------------------------------+--------------------+
| id   | title         | profile               | password             | notes                                          | timestamp          |
+------+---------------+-----------------------+----------------------+------------------------------------------------+--------------------+
| G269 | GitHub        | waterrmalann          | FHdL5pKtXPET+tTIqtcQ |                                                | 1662356963.3009381 |
| E539 | E-Mail        | hello@alanvarghese.me | CwK0                 | Professional E-Mail Address, used in Portfolio | 1662357065.7918363 |
| S977 | StackExchange | waterrmalann          | AKm9                 | same password for all stack exchange services  | 1662357098.0511053 |
+------+---------------+-----------------------+----------------------+------------------------------------------------+--------------------+
3 rows in set (0.00 sec)
```

Note: *Usernames and passwords used were purely for the demonstration of this program.*

Tables

- ➔ **config** *[key, value pair used for storing config and meta-data]*
  - ◆ **keyName** `VARCHAR(128)`
    - A dictionary-like key used as an identifier for a value.
  - ◆ **keyValue** `VARCHAR(512)`
    - Stores a corresponding value to the key.
- ➔ **entries** *[table for storage of account and password entries]*
  - ◆ **id** `VARCHAR(4)`
    - Unique identifier for each entry used as a reference.
  - ◆ **title** `VARCHAR(32)`
    - Usually the name of the service or the encrypted device.
  - ◆ **profile** `VARCHAR(32)`
    - The username/account/address for the password entry.
  - ◆ **password** `VARCHAR(512)`
    - The password in AES-256 encrypted form.
  - ◆ **notes** `VARCHAR(512)`
    - Any additional information regarding the specific entry.
  - ◆ **timestamp** `VARCHAR(18)`
    - Reference for when the entry was created/last edited.

REFERENCES

CBSE Text for Class XII: Computer Science with Python

-Sumita Aurora

https://www.python.org/

https://dev.mysql.com/doc/connector-python/en/

https://pycryptodome.readthedocs.io/

https://en.wikipedia.org/wiki/password_manager

https://app.diagrams.net/#