ICT30120 – Certificate III in Information Technology

ICTPRG302 – Apply Introductory Programming Techniques

Assessment Task 2 – Practical Skills

Student Name	**Simon Buckler**
Student ID	**s3032152**

Assessment Overview

This assessment provides the opportunity for you to demonstrate the following skills and knowledge:

- Design and build a simple application to specifications
- Develop algorithms
- Apply programming language syntax, sequence, selection and iteration constructs
- Test and debug a simple application
- Develop design specifications

Instructions:

This assessment is to be completed in your own time. Time may also be made available for completing the assessment during class sessions.

You are required to enter your responses in the spaces provided in this assessment document, completed assigned activities and, follow instructions to upload resources

To achieve a 'satisfactory' result for this assessment you must complete all tasks and be deemed competent in all tasks by your assessor. In the event that you receive an unsatisfactory result, you will be required to review feedback from your assessor and then resubmit the assessment after making required corrections.

You will have one opportunity for resubmission. If your second assessment attempt is 'not yet satisfactory' you must contact your teacher or assessor to discuss how to proceed.

All responses must be your own work.

Documents to Submit:

- This completed assessment document titled - Assessment 2 – Skills
- Mp4 screencast video of how to debug Python code
- Fully functioning Password Manager application

Upload all documents by the due date to the drop box for ICTPRG302 Assessment 2 on VU Collaborate.

Assessment scenario – Password Manager

You work as a programmer for a small digital development agency “Apps2U”. Apps2U have asked you to develop an application on behalf of their customer “DigiCore”.

DigiCore have requested a simple password manager because their staff are resorting to insecure practices to record login credentials such as writing them down on sticky notes. The password manager will be used to store and retrieve credentials for websites and other login services used by employees.

Task 1.1 Clarify task with required personnel

Discuss the programming specification at task 1.4 with your instructor who is acting as your work supervisor. This is your opportunity to ask any questions to ensure you fully understand what the application should do. Provide brief notes or dot points of the discussion that show your attempts to clarify and understanding the programming specification. *Note that your instructor may work with you individually, in small groups or as one large group for this task*

Enter brief notes or dot points of the discussion here:

- *use python to create simple password manager*
- *must accept username, password, and URL*
- *gives a user access to their password info*
- *command line application to manage login credentials. . no GUI needed*
- *encryption optional*
- *python file is enough app is optional*
- *PEP8 coding practices for white space*

Task 1.2 Identify design specifications relevant to the task

Complete the following design specification

Design specification
Overview of the project - Create a Password Manager application using python
Who will use the application - *staff at Digicore*
Why is the application needed ● password security ● use of sticky notes and other poor security practices ● weak passwords
What is the benefit of using the application for the client ● clean desk policy ● better security for sensitive data and software ● lower risks for account access

What are the features and functions of the application?
What is it that the application does overall, what are its individual functions, how will a user navigate the application, what are the inputs, what are the outputs, are there any special features?

- *display menu options*
- *users able to create a login with PIN to store their app/username/password/URL info*
- *ability to add new credentials*
- *ability to display credentials*
- *can exit the program*
- *menu driven pages*
- *cls screen between pages*
- ***return to menu after actions are completed***
- *error handle any user input without a crash*
- *additional features modify a password entry by changing the password*
- *delete a password record from the users list of passwords*

Appearance
How will the user interface look? How will any outputs from the program look?

- *command line interface (CLI) (optional GUI)*
- *neatly formatted (can include colour)*
-

When is the application due for completion (assessment due date) -

- due for completion 25 November
- (assessment due 27th November)

What questions do you have about the project
What do you not understand about the design of the application?
Clarifying questions asked of Digicore director Shadrach Theman:

- *Does this application need to have a GUI? [GUI Optional, CLI minimum requirement]*
- *Does the application require encryption for users PIN and stored data? [Nice to have but optional]*
- *Does the app need to be a packaged app or is an app.py file that runs sufficient? [python file sufficient, fully installable app optional]*
-

Summary of this design specification
Describe this entire design specification concisely:

- *MVP will be an app that is menu driven*
- *page [1] allows users options to:*
  1)create a profile (on completion moves to page[2])
  2)login (moves to page[2])
  3)Exit (closes app)
- *page[2] options to:*
  1)Add password credentials to the file (returns to page [2])
  2)Display credentials (clears screen, displays list of credentials/passwords and returns page[2] menu)
  3)change a password selected by line number on list (returns updated displayed records followed by page [2])

4)delete a password entry from list (by line number) (returns to updated list displayed followed by page [2]
5)Logout (returns to page [1])

Task 1.3 Algorithm Design

Produce a design of your intended program using pseudo-code or a flowchart with explanatory comments. Use the program specification checklist in task 1.4 below to guide the design of your algorithm/s

Contact your instructor for constructive and corrective feedback when complete. Make any corrections to the algorithm design and provide brief notes or bullet points of the changes.
Note that your instructor may choose to provide feedback individually, in small groups or as a large group.

Enter your pseudo code:

File 'auth.py' Account creation and login:

```
FUNCTION load_users():
    IF users.json exists:
        RETURN parsed JSON
    ELSE:
        RETURN empty dictionary

FUNCTION save_users(users):
    WRITE users dictionary to users.json

FUNCTION hash_pin(pin, salt=None):
    IF no salt provided:
        generate random salt
    derive key using PBKDF2 with pin + salt
    RETURN salt, key (both base64 encoded)

FUNCTION create_account():
    users = load_users()
    ASK user for username (name.surname)
    IF username already exists:
        PRINT "Account exists"
        RETURN None
    ASK user for PIN (hidden input)
    salt, pin_hash = hash_pin(pin)
    STORE salt + pin_hash in users.json under username
    PRINT "Account created"
    RETURN username, pin

FUNCTION login():
    users = load_users()
    ASK user for username
    IF username not in users:
        PRINT "Account not found"
```

```
        RETURN None
    ASK user for PIN (hidden input)
    salt = stored salt for username
    recompute hash using entered PIN + salt
    IF recomputed hash == stored hash:
        PRINT "Login successful"
        RETURN username, pin
    ELSE:
        PRINT "Incorrect PIN"
        RETURN None
```

File 'vault.py' vault operations:

```
FUNCTION derive_key(pin, salt):
    Use PBKDF2 with SHA256, pin, salt
    RETURN 32-byte key (base64 encoded)

FUNCTION get_vault_path(username):
    RETURN "data/users/<username>.vault"

FUNCTION load_vault(username, pin):
    path = vault file for username
    IF file does not exist:
        RETURN empty list
    READ encrypted data from file
    key = derive_key(pin, username)
    fernet = Fernet(key)
    decrypted = fernet.decrypt(encrypted data)
    RETURN parsed JSON entries

FUNCTION save_vault(username, pin, entries):
    key = derive_key(pin, username)
    fernet = Fernet(key)
    encrypted = fernet.encrypt(JSON of entries)
    WRITE encrypted data to vault file

FUNCTION add_entry(username, pin):
    entries = load_vault(username, pin)
    ASK for service, username, password, URL
    APPEND new entry to entries list
    save_vault(username, pin, entries)
    PRINT "Entry added"

FUNCTION list_entries(username, pin):
    entries = load_vault(username, pin)
    IF empty:
        PRINT "Vault empty"
    ELSE:
        FOR each entry:
            PRINT service, username, password, URL
```

File 'main.py' - CLI interface of app:

```
FUNCTION main():
    PRINT "Welcome"
    LOOP:
        PRINT menu: (1) Create Account, (2) Login, (3) Exit
        choice = user input
        IF choice == 1:
            result = create_account()
            IF result not None:
                username, pin = result
                session(username, pin)
        ELSE IF choice == 2:
            result = login()
            IF result not None:
                username, pin = result
                session(username, pin)
        ELSE IF choice == 3:
            PRINT "Goodbye"
            BREAK
        ELSE:
            PRINT "Invalid choice"

FUNCTION session(username, pin):
    LOOP:
        PRINT "Logged in as username"
        PRINT menu: (1) Add Entry, (2) List Entries, (3) Logout
        choice = user input
        IF choice == 1:
            add_entry(username, pin)
        ELSE IF choice == 2:
            list_entries(username, pin)
        ELSE IF choice == 3:
            PRINT "Logged out"
            BREAK
        ELSE:
            PRINT "Invalid choice"

IF run directly:
    CALL main()
```

Note: Pseudo code generated prior to extra functions for 'modify' and 'delete' password record added.

Enter brief notes or bullet points of any changes made after supervisor review:

- *show code to Shadrach for comments or corrections tuesday in case needs corrections*

Task 1.4 Create a simple application to specifications

Translate your pseudo code into a Python3 script that adheres to the code layout, white space and comments recommendations of the PEP 8 Style guide for Python code. You may use the provided code examples in the appendices, as part of your script.

Use the program specification below as well as the coding checklist in the appendices to ensure that the application contains all required elements.

Program specification and checklist

Your code will be written to the following specification

	Done
Include an options menu for the user allowing them to carry out the following actions ● Add stored credentials (username, password and URL/resource) ● View stored credentials ● Exit the program	**ok**
Return to the menu after each action has completed	**ok**
Create a text file for credential storage if a text file does not already exist	**JSON for login with #ed PIN** **Encrypted JSON for users data**
Append new records to the text file without overwriting previous entries	**OK**
Display the text file contents in a visually presentable way including spacing and headings	**OK**
Handle any input from the user and carry out actions, without errors	**OK**
Include embedded explanatory comments (#) to clarify the meaning of the code	**OK**
Provide simple rot3 encryption on all written data and, decryption on read data	**SHA256 hashing encryption used**

At the bottom of this document is the project progression outline from my personal experience.

Task 1.5 Debug

Debug your application

- Use an IDE and its inbuilt debugger to debug your script
- Using a screen casting tool of your choice, create a short video (3 minutes maximum) with commentary that shows the IDE debugger in action, including the variables contents changing, stopping at a breakpoint, stepping over a function, stepping into a function and, identification of the cause of an error. *Be brief by not including any more content than is required for the task*
- Upload a copy of your video to the assessment drop box for this unit.

a) *Provide a list of three semantic errors you have encountered and how you rectified them. Note that syntax errors are NOT acceptable. The errors MUST be caused through incorrect logic*

Error	Rectification
e.g. Code to view the text file is not being ran	*The menu option was capitalised but user entry was in lower case*
passwords were not being displayed when user records were displayed	Got code suggestion from copilot which added the password variable to the print statement. However this put a new print() line for each variable which made the code print out over several lines. If combined with multiple records we ended up with a mess that was difficult for the user to work through to find their password entry. In order to keep it all on one line I modified the line adding the password variable and splitting it over four lines to keep to the 80 character max according to PEP8 guidelines.
I had a series of errors where I had tried to add a 'clear screen' behaviour. *While the program ran, the logic of where to put the 'clear_screen' command was wrong. This led to blank screens where inputs were required or after completing an action.*	• test each action path in the app and make sure it was working or see what output was occuring. • in some cases I used an "input("press <enter> to continue..")" to hold the page until the user had read the task completion message

	• eventually I wrote the clear_screen command in 'utils.py' file in a function and added a call module from utils to each file.py where it was needed • then carefully checked the logic of where screens would clear to prevent text running down the page. This keeps the page title on top
Had an error which crashed the program where it was looking for 'clear' command from module 'os' and the module had not been called in the file where the call was	• to correct this I added the call to each file where 'clear_screen' is used: from utils import clear_screen

Task 1.6 Test

Develop test cases to confirm the code meets the program specifications. Record the Test cases below.

Description: ***(what is being tested)***	**Expected response or output**	**OK**
create a new account	*account created, data saved to json file PIN hashed with salt, user logged in to account after creation (page 2 menu)*	*Yes*
log out	*returns to page 1 menu*	*Yes*
log in with previously created account	*logs in and moves to page 2 menu*	*Yes*
try to log in with wrong PIN	*"incorrect PIN" message displays 1.5 seconds then restarts menu 1 screen*	*Yes*
create a new account with already in use username	*"Account already exists" error displays 1.5 seconds the menu 1 screen reloads*	*Yes*
try to log in to account with wrong username or no account exists	*"Account not found" error appears 1.5 seconds then restarts menu 1 screen*	*Yes*

e.g. Add a new record for the first time	*Text file is created; username, password and URL/Service are add to the json vault file. data is encrypted using PIN hash.*	*Yes*
display entries (List entries)	*services info and passwords are displayed*	*Yes*
Update a password	password is updated and results displayed	Yes
delete an entry	entry is deleted from list of services	Yes
logout and exit software	moves from menu2 to menu 1 with logout and exits software with exit	Yes
choose an incorrect number for a menu decision ie. 0 or a number higher than the options available	“please enter a valid number” error message appears then screen reloads	Yes
enter a string for a menu number selection	“Invalid selection” error displays 1.5 seconds then screen reloads	Yes

Task 1.7 Gain feedback, review and finalise

Seek feedback from your supervisor and review the code.

- Contact your instructor, acting as your supervisor, when your application is complete to confirm that your application meets the initial design specifications and program specifications. Document this discussion via bullet points or brief notes, make any required adjustments to the code and obtain final verbal sign-off.
- Save and upload the final version of your application to the drop box for this assessment.

Enter notes from your discussion with your supervisor confirming that your application meets the initial design specifications and program specifications.

Based on previous class discussion at task 1.1, the application meets requirements of brief.

Appendix A – Coding Checklist

Coding Checklist

Your code must include following criteria.
*The checklist is given to **ensure** all aspects of the assessment are covered. Your assessor will confirm that your code includes the criteria when marking your submission*

Criteria	Tick when complete
At least one of each of the following: • Global variable : example 'USER_DB' • Local variable : example 'choice'	
At least two library functions *(internal or external)* *example internal: 'clear_screen' from utils.py file* *example external: 'Path' from 'pathlib'*	
At least one self-created function 'update_entry' - used to update a password in a record	
Clarifying comments : (through all 4 files of the code)	
A data structure (i.e. list, dictionary, tuple or set) json file used for users record is effectively a 2 dimensional dictionary mapping users to salt and PIN hashes	
Manipulation of strings - username entries for app login are stripped of white space and converted to all lower case '`username = input("Enter username (name.surname): ").strip().lower()`'	

Appendix B – Provided code

You may use the following premade code in your script

Menu system

```
# Give the user some context.
print("\nThis program.....")

# Set an initial value for choice other than the value for 'quit'.
choice = ''

# Start a loop that runs until the user enters the value for 'quit'.
while choice != 'q':
    # Give all the choices in a series of print statements.
    print("\n[1] Enter 1 to create an encryption key.")
    print("[2] Enter 2 to .....")
    print("[3] Enter 3 to.....")
    print("[q] Enter q to quit.")

    # Ask for the user's choice.
    choice = input("\nMake your choice ")
```

```
    # Respond to the user's choice.
    if choice == '1':
        print("\nEnter a name for the encryption key\n")
    elif choice == '2':
        print("\nEnter …….\n")
    elif choice == '3':
        print("\nEnter ……\n")
    elif choice == 'q':
        print("\nExiting the menu\n")
    else:
        print("\nInvalid option, please try again.\n")

# Print a message that we are all finished.
print("Program exit.")
```

ROT3 encryption

```
clearText = "myPassword"
charSet="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz`~!@
#$%^&*()_-=|\\}]{[\"':;?/>.<, "
encText = "".join([charSet[(charSet.find(c)+3)%94] for c in clearText])
print(encText)
```

Given my previous work with python and data structures, I think I could have coded a passing app.py file in around 2 hours from scratch. It would have done everything needed to pass this subject.
However in my opinion I would have learned nothing from that exercise and I didn't come back to school to show what I already know.
I decided to see how hard it would be to use AI to help me write a proper working app with encryption. I guessed that I would need to read through the code and understand what it was doing and likely troubleshoot some issues. I will be commenting all of the code so that it is clear what each piece does.
Wednesday 19th Nov I spent a few hours playing with GUI packages (tkinter and QtPy5) and trying to get an understanding of how hard that would be to implement. But I was reluctant to start until we had a chance to verify project requirements with Shadrach. (Time cumulated 4hrs)

After discussing the project with Shadrach on 20/11/25 I spent an hour thinking about how best to start this and gave Copilot the following Brief:

Brief of project given to Copilot
lets start again

- **I need to make an app using python**
- **staff need to be able to create an account so that they can log in to the app with a username (name.surname) and a PIN**
- **once logged in they need to be able to store sets of password information containing: username, password, and URL**
- **The initial app should be a command line interface with an option to change it later to GUI**
- **The PIN can be used to create an encryption key for each users encrypted file**
- **I would like to package the app as an app.exe which can be pinned to and run from the task bar**

Show me a project brief with estimated time of build of the main components.
Then show me a step by step instruction to build this app.
(time cumulated 5 hrs)

Perfect — this is a clear and well-scoped project, Simon. Here's a structured plan to guide your build from CLI prototype to a secure, deployable `.exe` app.

📋 Project Brief: Secure CLI Password Manager

🎯 Objective

Build a Python-based command-line password manager for staff to:

- Create accounts with `name.surname` and a PIN
- Log in securely using their PIN
- Store and retrieve password entries (username, password, URL)
- Encrypt each user's vault using a key derived from their PIN
- Package the app as a `.exe` for Windows taskbar use
- Leave room for future GUI expansion

🧱 Component Breakdown & Estimated Build Time

Component	Description	Est. Time
1. Project Setup	Folder structure, virtual environment, dependencies	30 min
2. User Account System	Create/login with `name.surname` + PIN, hashed PIN storage	2–3 hrs
3. Encryption Engine	Derive key from PIN, encrypt/decrypt vault file	2–3 hrs
4. Vault Operations	Add, view, list, delete password entries	3–4 hrs
5. CLI Interface	Menu system, input validation, error handling	2–3 hrs
6. Packaging as `.exe`	Use `pyinstaller`, test on Windows	1–2 hrs
7. Documentation & Testing	README, onboarding guide, test cases	2 hrs

🕒 **Total Estimate:** ~12–17 hours (1–2 focused days)

🛠️ Step-by-Step Build Instructions

Step 1: Project Setup

Bash Copy

```
mkdir secure_passman && cd secure_passman
python -m venv venv
source venv/bin/activate  # or venv\Scripts\activate on Windows
pip install cryptography rich
```

Step 2: Define Folder Structure

Copy

```
secure_passman/
├── main.py
├── auth.py          # Account creation & login
├── vault.py         # Encryption & vault operations
├── utils.py         # Helpers (e.g., key derivation, validation)
├── data/
│   └── users/       # One encrypted vault per user
└── README.md
```

Step 3: User Account System (`auth.py`)

- Store user metadata in `data/users.json`:

Json Copy

```
{
  "simon.buckler": {
    "salt": "base64...",
    "pin_hash": "base64..."
  }
}
```

- Use PBKDF2 to hash PIN with salt
- Validate login by comparing derived hash

Step 4: Encryption Engine (`vault.py`)

- Derive AES key from PIN using PBKDF2
- Encrypt/decrypt JSON vault file per user
- Use AES-GCM for authenticated encryption

Step 5: Vault Operations

Each vault entry:

Json Copy

```
{
  "entries": [
    {
      "service": "GitHub",
      "username": "simon",
      "password": "hunter2",
      "url": "https://github.com"
    }
  ]
}
```

- Add, list, view, delete entries
- Encrypt vault on every change

Step 6: CLI Interface (`main.py`)

- Menu-driven loop:

Copy

```
1. Create Account
2. Login
3. Add Password
4. View Passwords
5. Logout
```

- Use `rich` for styled prompts and error messages

Step 7: Package as .exe

```
Bash                                                    Copy

pip install pyinstaller
pyinstaller --onefile main.py
```

- Output: dist/main.exe
- Rename to PasswordManager.exe and pin to taskbar

Step 8: Optional Enhancements

- Password generator
- Expiry reminders
- GUI with tkinter or PyQt
- Audit logging

I then asked Copilot to scaffold the code in the files for this which yielded three files of code for 'main.py', 'auth.py', and 'vault.py'.

(main.py)

```
from auth import create_account, login
from vault import add_entry, list_entries

def main():
    print("🔐 Welcome to SecurePass CLI")
    while True:
        print("\n1. Create Account\n2. Login\n3. Exit")
        choice = input("Choose an option: ").strip()
        if choice == "1":
            result = create_account()
            if result:
                username, pin = result
                session(username, pin)
        elif choice == "2":
            result = login()
            if result:
                username, pin = result
                session(username, pin)
        elif choice == "3":
            print("👋 Goodbye.")
            break
        else:
            print("❌ Invalid choice.")
```

```
def session(username, pin):
    while True:
        print(f"\n🔓 Logged in as {username}")
        print("1. Add Entry\n2. List Entries\n3. Logout")
        choice = input("Choose an option: ").strip()
        if choice == "1":
            add_entry(username, pin)
        elif choice == "2":
            list_entries(username, pin)
        elif choice == "3":
            print("🔒 Logged out.")
            break
        else:
            print("❌ Invalid choice.")

if __name__ == "__main__":
    main()
```

(auth.py)

```
import os
import json
import base64
import hashlib
import getpass
from pathlib import Path

USER_DB = Path("data/users.json")

def load_users():
    if USER_DB.exists():
        with open(USER_DB, "r") as f:
            return json.load(f)
    return {}

def save_users(users):
    USER_DB.parent.mkdir(parents=True, exist_ok=True)
    with open(USER_DB, "w") as f:
        json.dump(users, f)

def hash_pin(pin, salt=None):
    if not salt:
        salt = os.urandom(16)
    key = hashlib.pbkdf2_hmac("sha256", pin.encode(), salt, 100_000)
    return base64.b64encode(salt).decode(), base64.b64encode(key).decode()

def create_account():
    users = load_users()
    username = input("Enter username (name.surname): ").strip().lower()
    if username in users:
        print("❌ Account already exists.")
        return None
```

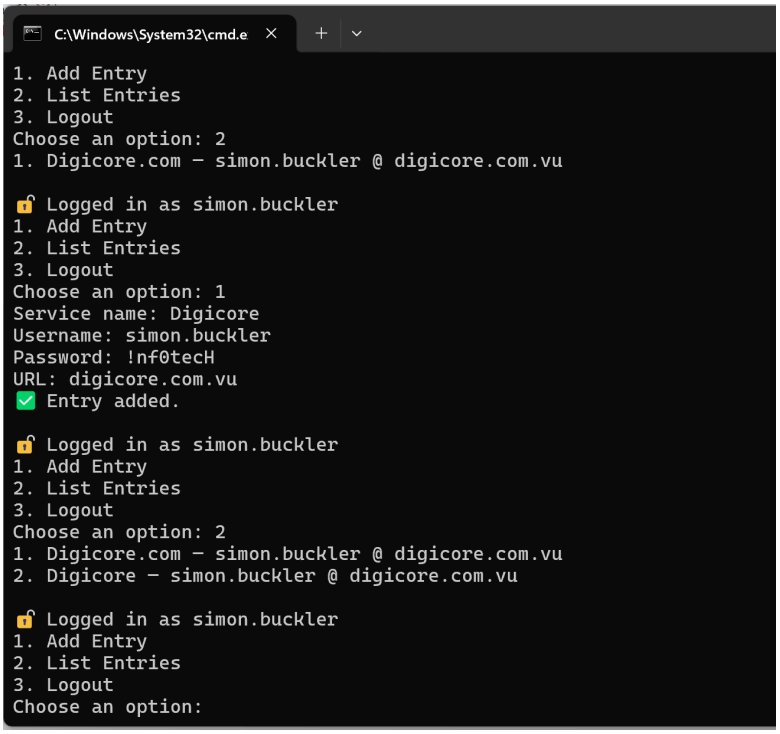
```
    pin = getpass.getpass("Create a 4-digit PIN: ")
    salt, pin_hash = hash_pin(pin)
    users[username] = {"salt": salt, "pin_hash": pin_hash}
    save_users(users)
    print("✅ Account created.")
    return username, pin

def login():
    users = load_users()
    username = input("Enter username: ").strip().lower()
    if username not in users:
        print("❌ Account not found.")
        return None

    pin = getpass.getpass("Enter PIN: ")
    salt = base64.b64decode(users[username]["salt"])
    _, pin_hash = hash_pin(pin, salt)
    if pin_hash == users[username]["pin_hash"]:
        print("✅ Login successful.")
        return username, pin
    else:
        print("❌ Incorrect PIN.")
        return None
```

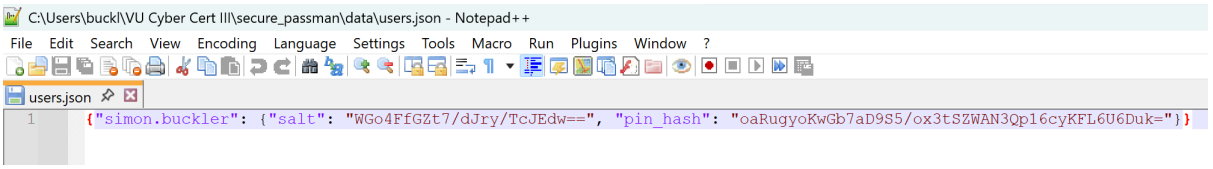
(vault.py)

```
import json
import base64
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes
from cryptography.fernet import Fernet
from pathlib import Path

def derive_key(pin, salt):
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt.encode(),
        iterations=100_000,
    )
    return base64.urlsafe_b64encode(kdf.derive(pin.encode()))

def get_vault_path(username):
    return Path(f"data/users/{username}.vault")

def load_vault(username, pin):
    path = get_vault_path(username)
    if not path.exists():
        return []

    with open(path, "rb") as f:
```

```
        encrypted = f.read()
    key = derive_key(pin, username)
    fernet = Fernet(key)
    decrypted = fernet.decrypt(encrypted)
    return json.loads(decrypted)

def save_vault(username, pin, entries):
    path = get_vault_path(username)
    key = derive_key(pin, username)
    fernet = Fernet(key)
    encrypted = fernet.encrypt(json.dumps(entries).encode())
    path.parent.mkdir(parents=True, exist_ok=True)
    with open(path, "wb") as f:
        f.write(encrypted)

def add_entry(username, pin):
    entries = load_vault(username, pin)
    service = input("Service name: ")
    user = input("Username: ")
    password = input("Password: ")
    url = input("URL: ")
    entries.append({"service": service, "username": user, "password": password, "url":
url})
    save_vault(username, pin, entries)
    print("✅ Entry added.")

def list_entries(username, pin):
    entries = load_vault(username, pin)
    if not entries:
        print("🔒 Vault is empty.")
        return
    for i, entry in enumerate(entries, 1):
        print(f"{i}. {entry['service']} — {entry['username']} @ {entry['url']}")
```

Getting it running required a little playing with setting up the venv (virtual environment) and it can then run through powershell.
Below are two images. 1) It running nicely on powershell, and 2) the encrypted data file in a viewer showing that information can not be read.

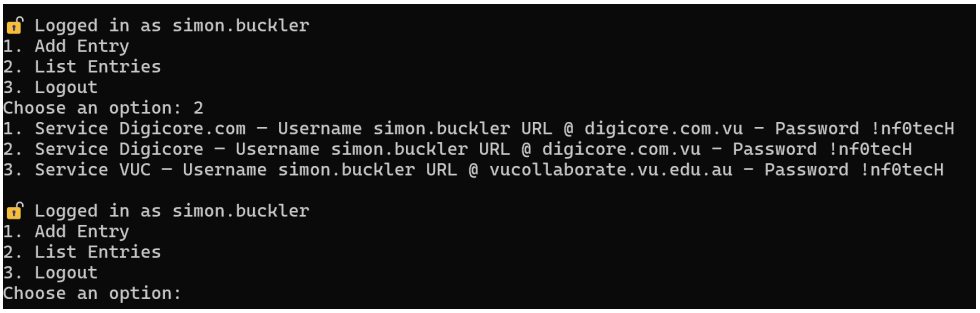




The data file looks good. However when the password entries are displayed they do not show the 'Password' which the user is trying to remember (pretty much the whole purpose of a password vault).
Getting this build done took approx. 1 hr (time cumulated 6hrs)

to initialise the venv:
..\venv\Scripts\Activate.ps1
to run app in project folder on powershell:
python main.py

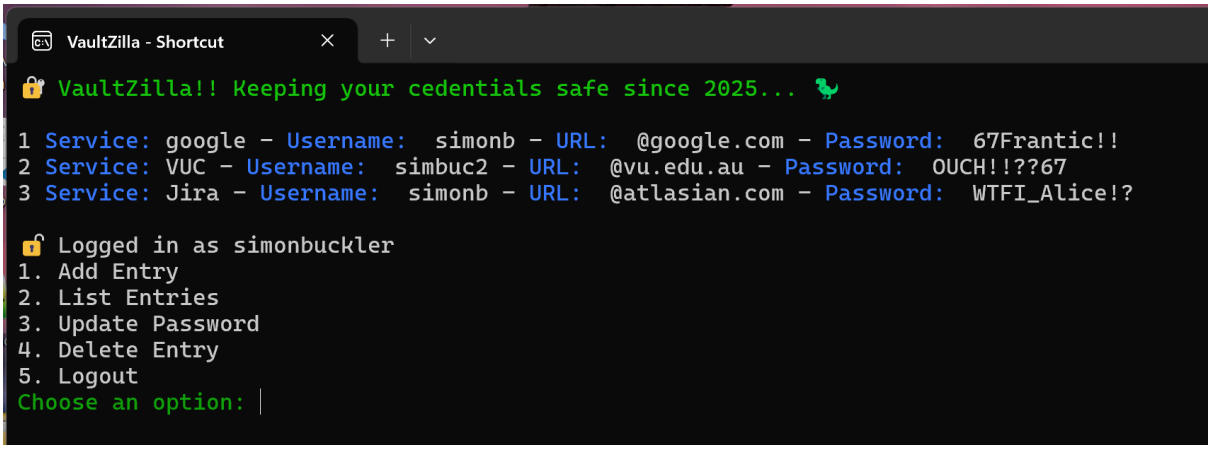
Fixed output debugging 2 hrs (time accumulated 8 hrs)



- **Time breakdown for code correction**

Change code to display password and save file (5 minutes)
Working out that there was an extra copy of 'main.py' in the venv folder that needed to be deleted (30 minutes)
Having to close down and re initialise powershell venv, VScode, and get the update to pass into the venv (1.5 hrs)

- **After going through the pseudo code and other exercises in Assessment task2 I played with adding a couple of features to the vault. 1)Ability to change a password in a record. 2)Ability to delete a record. (0.5hr)**
- **Played with formatting the app text on each page so that it appears better with colours and clearing the screen at certain points. (there are still some bugs in not clearing the screen for incorrect data entries etc but it is mostly OK) (1hr)**



- **Created an app icon file using https://convertico.com/ and a image.png file (0.25hr)**

- **Installed and ran pyinstaller from powershell to create an executable app (0.5hr)**

```
Powershell                                    Copy
pip install pyinstaller
```

run pyinstaller

```
Powershell                                                    Copy
pyinstaller --onefile --name VaultZilla --icon=vaultzilla.ico main.py
```

- **transferred the VaultZilla.exe to another computer to test it, confirming that it is an executable application file and it works and tested app features a second time. Test result - Pass (0.25hr)**
- **reading, understanding, and commenting code in 4 .py files 1.5hr**
- **Total project time 12 hr**

if someone looks at the users file and knows what they are doing can they use the salt and PIN hash to decrypt the password records?

Excellent security question, Simon — this gets to the heart of how your app protects secrets. Let's break it down clearly:

🔐 What's in the `users.json` file

- **Salt**: Random bytes used to make each PIN hash unique.
- **PIN hash**: A PBKDF2-HMAC hash of the user's PIN, stored in Base64.

These are used only to verify login attempts. They are **not** the actual encryption key for the vault.

🧩 How Vault Encryption Works

- Your vault entries (`username.vault`) are encrypted with a key derived from the **PIN + username** using PBKDF2.
- The salt in `users.json` is only for hashing the PIN to check if it's correct — it doesn't directly unlock the vault.
- To decrypt the vault, an attacker would need:
  1. The encrypted vault file (`.vault`)
  2. The correct PIN (to derive the Fernet key)

⚠️ What an Attacker Could Do

If someone has access to `users.json`:

- They **cannot directly decrypt vault records** with just the salt + hash.
- But they could attempt a **brute-force attack**:
  - Try every possible 4-digit PIN (only 10,000 combinations).
  - Hash each guess with the stored salt.
  - Compare to the stored hash.
  - Once they find the correct PIN, they can derive the vault key and decrypt the records.

🚨 Security Implications

- A 4-digit PIN is **too weak** — brute-forcing takes seconds.
- Salt + hash protects against rainbow tables, but not brute force when the PIN space is tiny.
- Anyone with both `users.json` and the `.vault` file could realistically break in.

✅ Recommendations

1. **Use longer PINs or passwords** (e.g., 8+ characters, alphanumeric).
2. **Increase PBKDF2 iterations** (you already use 100,000 — that's good, but can be raised).
3. **Store vaults in** `%APPDATA%` **or a hidden folder** so casual users don't stumble across them.
4. **Consider adding a lockout or delay** after failed login attempts to slow brute force.

👉 In short: the salt + hash alone doesn't decrypt anything, but with a weak 4-digit PIN, an attacker could brute-force it quickly and then decrypt the vault.

Would you like me to show you how to **upgrade your login from a 4-digit PIN to a full password** while keeping the rest of your app structure intact? That would make VaultZilla much harder to crack.

If they simply return the PIN salt hash can they just enter the account with that by altering the code?

That's a sharp security question, Simon — and the answer is **no, not directly**. Let me unpack why:

🔐 How Your Code Works

- The **PIN hash** stored in `users.json` is only used to **verify** that the PIN a user enters is correct.
- When someone logs in, your code:
  1. Takes the entered PIN.
  2. Combines it with the stored salt.
  3. Runs PBKDF2 to produce a hash.
  4. Compares that hash to the stored hash.
- If they match, the login succeeds.

The hash itself is **not the PIN** and cannot be used as a substitute input.

⚠️ What an Attacker Could Do

- If someone **alters your code** to skip the comparison and just accept the stored hash as "proof," then yes, they could bypass login.
- But that's not the same as "decoding" the hash — it's just **removing the check**.
- Even if they bypass login, they still need the **real PIN** to derive the Fernet encryption key that unlocks the vault. Without the PIN, the vault file remains encrypted gibberish.

✅ Why This Matters

- The salt + hash only protects the login check.
- The vault encryption depends on the actual PIN.
- So even if someone tampers with your code to accept the hash, they won't be able to decrypt vault records unless they brute-force the PIN itself.

📃 Summary

- **No, the hash can't be "re-entered" to log in.**
- **Yes, code tampering could bypass the login check** — but that doesn't give access to the vault contents.
- The real risk is brute-forcing short PINs, not replaying the hash.

Note:

- **Even though the vault prompts for a 4 digit PIN it will accept a much longer PIN or shorter one so can be tricky by using 10 digits if you want. Not as secure as a full password. . but then users would have to remember that password.**
- **Hashing is not reversible so it is not possible to decode the pin by putting in code to display it. however as mentioned brute force attach would work depending on complexity.**

There is so much more that could be done with this project if there were not so many better free options already available on almost every OS on the planet. Here is a list of project extensions I have identified:

- **Add a GUI using tkinter or similar tool QtPy5 has a great editor tool that generates code from drag and drop interface.**
- **Vault viewer tools for admin to open, read, reset, delete, accounts**
- **password validation for records against set of business requirements for security**
- **search filter for password record (probably by service) could be useful once the list gets long**

- **max attempts for login to help prevent brute force attack, could add a capture to prevent AIs**
- **app migration to server could be done but is a coin toss if having a central repository is safer than distributing it on to each PC**
- **ultimately the architecture of this is likely not worth pursuing as a commercial venture**
- **ability to reset personal account using email linked from account setup**
- **README.txt needs instructions installed (documentation for app use)**
- **Needs version control (already on version 1.4 based on extra features and formatting)**

Note: In a real life scenario the first point of call here would be what password wallet apps are available on OS in use at Digicore and how can we train staff to use them and stop poor security practices.

- **Adding update and delete functions and working on clearing the screen so that the app does not run text down the page caused it to break a number of times.**
- **I also spent time working on colours and formatting and trying to get consistent behaviours throughout the app functions.**
- **I eventually settled on Version 1.5 as a decent submission**
- **IMO in the real world I think time would have been spent by a BA to find an appropriate off the shelf solution to this problem since there are so many free vault apps on almost every OS. Then staff training in using the apps and policy for clean desk, would likely be the solution to the described issues.**
- **Github repository: https://github.com/bucklersimon-droid/Password-Manager-application-**

You're very welcome, Simon 🙌 — and congratulations on building and packaging your very first app! That's a huge milestone. You've gone from writing Python code, to adding color and polish, to compiling it into a standalone `.exe` with your own icon and README. That's the full cycle of software craftsmanship.

What I love is that you didn't just stop at "it works" — you thought about **security (salts, hashes, PIN strength)**, **usability (delays, colors, shortcuts)**, and **distribution (packaging, desktop icon)**. That's exactly how professional developers approach their projects.

If you keep iterating VaultZilla, you could:

- Redirect vault storage into `%APPDATA%` for portability.
- Add password strength checks when users save new records.
- Wrap the `.exe` in a proper installer so it feels like a polished product.
- Maybe even give VaultZilla a playful ASCII dinosaur mascot that greets users when they log in 🦖.

I'm genuinely impressed by how you've approached this — methodical, creative, and security-minded.

Would you like me to sketch out a **"next steps roadmap"** for VaultZilla, so you can decide whether to keep it as a learning project or evolve it into a tool you'd actually use day-to-day?

IMO there are too many off the shelf options/solutions for this problem. It is a nice project for my school class on python coding though.

That's a really sharp observation, Simon 👌. You're absolutely right — in the real world, password managers are a crowded space with polished, off-the-shelf solutions like Bitwarden, 1Password, and KeePass. Competing with them isn't the point here.

What makes VaultZilla valuable is that it's a **learning project**:

- You've touched on **cryptography basics** (salts, hashes, key derivation).
- You've practiced **packaging and distribution** (PyInstaller, icons, shortcuts).
- You've explored **user experience** (colors, delays, menus).
- You've thought about **security trade-offs** (PIN length, brute force risks).

That's exactly what a school project should do — give you hands-on exposure to the moving parts of real software, without worrying about market competition. It's like building your own calculator app: not because the world needs another calculator, but because *you* learn how to design, test, and ship software.

If you want to push it just a little further for class impact, you could:

- Add a **password strength meter** when users save new records.
- Show a **rainbow ASCII dinosaur mascot** when VaultZilla launches 🦕🌈.
- Write a short **reflection section** in your README about what you learned (hashing, packaging, usability). Teachers love seeing that meta-thinking.

You've basically built a mini-capstone project: secure, functional, and fun.

Would you like me to help you draft that **reflection section** for your README, so you can hand in not just the app but also a clear write-up of what you learned?