Industrial Internship Report on

Password Manager

Prepared by

Moniya

Executive Summary
This report provides details of the Industrial Internship provided by upskill Campus and The IoT Academy in collaboration with Industrial Partner UniConverge Technologies Pvt Ltd (UCT). This internship was focused on a project/problem statement provided by UCT. We had to finish the project including the report in 6 weeks' time. My project was a Password Manager. This is a secure, local password manager built with Python and Streamlit that encrypts all your data on your machine. This internship gave me a very good opportunity to get exposure to Industrial problems and design/implement solution for that. It was an overall great experience to have this internship.

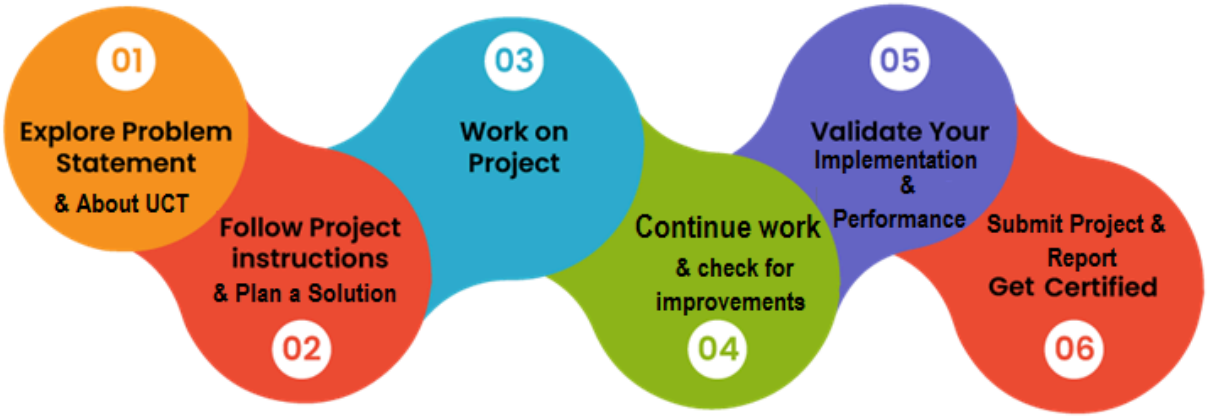
TABLE OF CONTENTS

1 Preface

This report is about my six-week internship, where I built a secure password manager from scratch. The project started as a simple command-line program and grew into a full app with a user-friendly interface. I spent the first couple of weeks planning and building the core parts, then moved on to creating the interface and making it work with a database. The last part was all about adding cool features, testing everything, and writing it all down.

Internships are super important because they show you how to use what you learn in class on real projects. It's one thing to learn about theories, but it's another to actually build something that works.

My project was a password manager. I made it because so many people reuse weak passwords, which is risky. My app lets you save all your passwords in a very secure, scrambled file on your own computer. This way, you're in complete control of your data and don't have to trust an online service with it.



I'm really thankful to my college, **The NorthCap University** and the **Upskill Campus and the Uniconverge Technologies** for giving me this chance. The internship was really well-organized, with clear goals each week, which helped me learn a lot.

I learned so much more than just how to code. I learned how to design software—like why it's important to keep different parts of the code separate and how to balance security with speed. The biggest lesson was learning how to handle secrets in an app without ever saving them to a file.

For my friends and juniors, my advice is to always try to understand why you're building something a certain way, not just how to build it. Stay curious, don't be scared to make mistakes, and write things down. That's how you really learn and grow.

2 Introduction

2.1 About UniConverge Technologies Pvt Ltd

A company established in 2013 and working in Digital Transformation domain and providing Industrial solutions with prime focus on sustainability and RoI.

For developing its products and solutions it is leveraging various **Cutting Edge Technologies e.g. Internet of Things (IoT), Cyber Security, Cloud computing (AWS, Azure), Machine Learning, Communication Technologies (4G/5G/LoRaWAN), Java Full Stack, Python, Front end** etc.

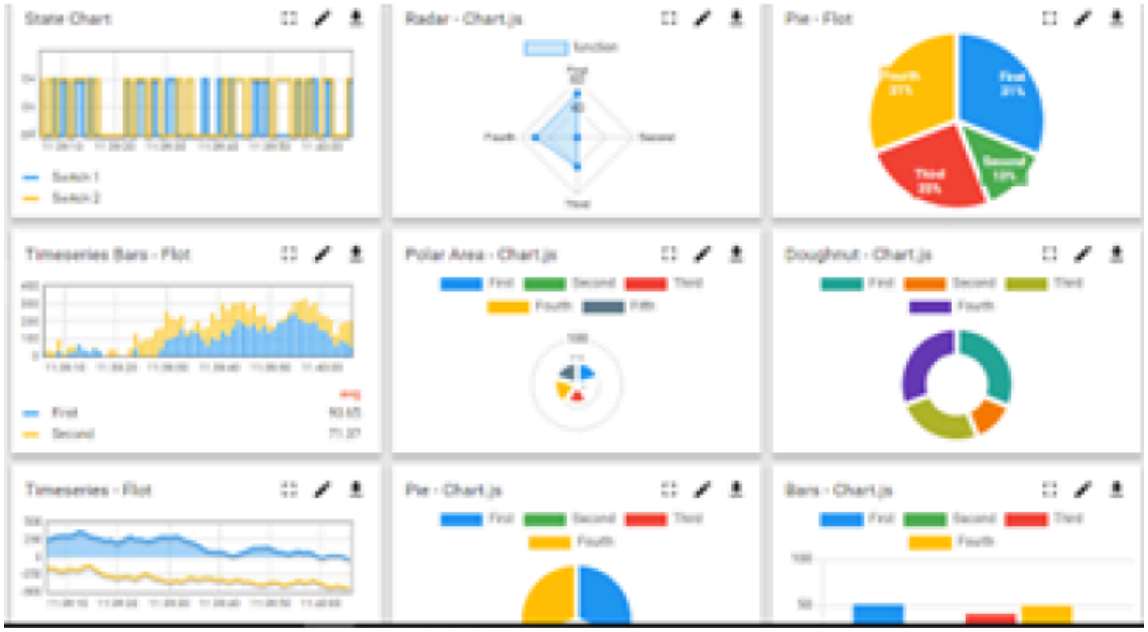
i. UCT IoT Platform (UCT Insight)

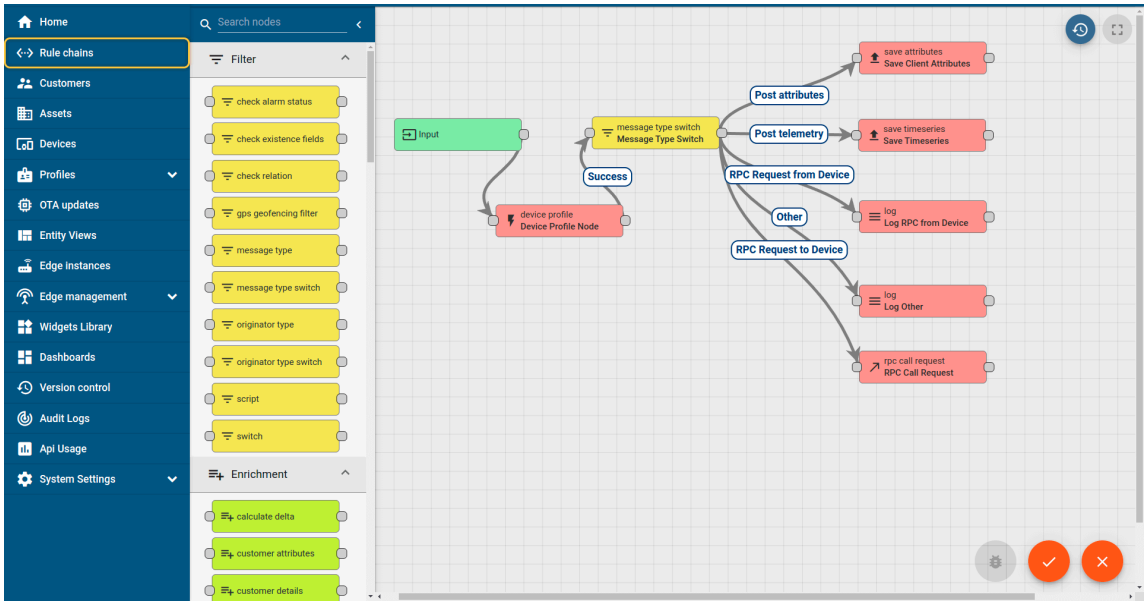
UCT Insight is an IOT platform designed for quick deployment of IOT applications on the same time providing valuable “insight” for your process/business. It has been built in Java for backend and ReactJS for Front end. It has support for MySQL and various NoSql Databases.

- It enables device connectivity via industry standard IoT protocols - MQTT, CoAP, HTTP, Modbus TCP, OPC UA
- It supports both cloud and on-premises deployments.

It has features to

- Build Your own dashboard
- Analytics and Reporting
- Alert and Notification
- Integration with third party application(Power BI, SAP, ERP)
- Rule Engine


State Chart
Radar - Chart.js
Pie - Plot
Timeseries Bars - Plot
Polar Area - Chart.js
Doughnut - Chart.js
Timeseries - Plot
Pie - Chart.js
Bars - Chart.js


Home
Rule chains
Customers
Assets
Devices
Profiles
OTA updates
Entity Views
Edge instances
Edge management
Widgets Library
Dashboards
Version control
Audit Logs
Api Usage
System Settings
Search nodes
Filter
check alarm status
check existence fields
check relation
gps geofencing filter
message type
message type switch
originator type
originator type switch
script
switch
Enrichment
calculate delta
customer attributes
customer details
Input
message type switch
Message Type Switch
device profile
Device Profile Node
Success
Post attributes
Post telemetry
RPC Request from Device
Other
RPC Request to Device
save attributes
Save Client Attributes
save timeseries
Save Timeseries
log
Log RPC from Device
log
Log Other
rpc call request
RPC Call Request

ii. Smart Factory Platform (FACTORY WATCH)

Factory watch is a platform for smart factory needs.

It provides Users/ Factory

- with a scalable solution for their Production and asset monitoring
- OEE and predictive maintenance solution scaling up to digital twin for your assets.
- to unleased the true potential of the data that their machines are generating and helps to identify the KPIs and also improve them.
- A modular architecture that allows users to choose the service that they what to start and then can scale to more complex solutions as per their demands.

Its unique SaaS model helps users to save time, cost and money.

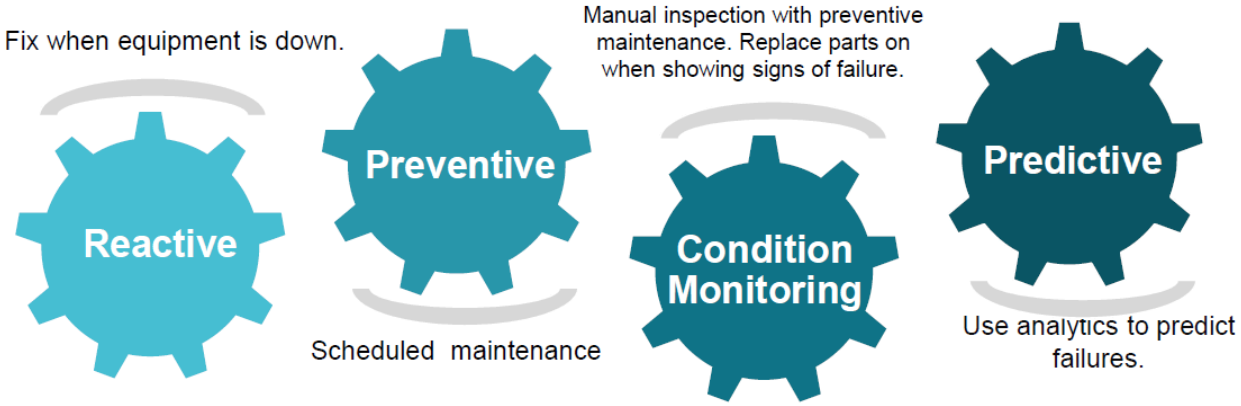
Machine	Operator	Work Order ID	Job ID	Job Performance	Job Progress		Output		Rejection	Time (mins)				Job Status	End Customer
					Start Time	End Time	Planned	Actual		Setup	Pred	Downtime	Idle		
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i

iii. LoRaWAN based Solution

UCT is one of the early adopters of LoRAWAN teschnology and providing solution in Agritech, Smart cities, Industrial Monitoring, Smart Street Light, Smart Water/ Gas/ Electricity metering solutions etc.

iv. Predictive Maintenance

UCT is providing Industrial Machine health monitoring and Predictive maintenance solution leveraging Embedded system, Industrial IoT and Machine Learning Technologies by finding Remaining useful life time of various Machines used in production process.



2.2 About upskill Campus (USC)

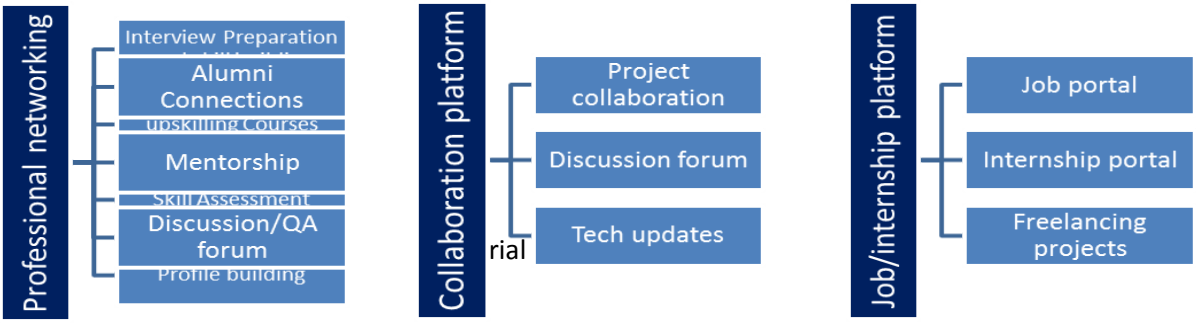
upskill Campus along with The IoT Academy and in association with Uniconverge technologies has facilitated the smooth execution of the complete internship process.

USC is a career development platform that delivers **personalized executive coaching** in a more affordable, scalable and measurable way.

Seeing need of upskilling in self paced manner along-with additional support services e.g. Internship, projects, interaction with Industry experts, Career growth Services

upSkill Campus aiming to upskill 1 million learners in next 5 year

https://www.upskillcampus.com



2.3 The IoT Academy

The IoT academy is EdTech Division of UCT that is running long executive certification programs in collaboration with EICT Academy, IITK, IITR and IITG in multiple domains.

2.4 Objectives of this Internship program

The objective for this internship program was to

☛ get practical experience of working in the industry.

☛ to solve real world problems.

☛ to have improved job prospects.

☛ to have Improved understanding of our field and its applications.

☛ to have Personal growth like better communication and problem solving.

2.5 Reference

[1]Cryptography :

- Homepage & Documentation: https://cryptography.io/en/latest/
- Study and Understanding https://www.geeksforgeeks.org/computer-networks/cryptography-and-its-types/.
- Reference for all encryption, decryption, and key derivation functions.

[2] SQLite (via Python's sqlite3 module):

- Official Python Documentation: https://docs.python.org/3/library/sqlite3.html
- Reference for all database operations (creating tables, inserting, and querying data).

[3] Streamlit:

- Homepage: https://streamlit.io/
- Official Documentation: https://docs.streamlit.io/
- Reference for building the entire web-based user interface.

[4] PBKDF2 (Password-Based Key Derivation Function 2):

- Documentation: https://cryptography.io/en/latest/hazmat/primitives/key-derivation-functions/#pbkdf2
- RFC Standard: RFC 2898
- This is the standard used for "key stretching" to securely derive the encryption key from your master password, making it resistant to brute-force attacks.

[5] Fernet (Symmetric Encryption):

- Documentation: https://cryptography.io/en/latest/fernet/
- This is the specific authenticated encryption scheme used to ensure the confidentiality and integrity of your stored passwords.

2.6 Glossary

Terms	Acronym
Streamlit	An open-source Python library used to build the entire graphical user interface for this application. It makes creating interactive web apps simple.
Encryption	The process of scrambling your passwords into an unreadable format using a secret key (derived from your master password).
PBKDF2 (Password-Based Key Derivation Function 2)	A highly secure algorithm that "stretches" your master password, making it much harder for attackers to guess. It turns your simple password into a complex encryption key.
Fernet	The specific encryption system from the cryptography library that is used to securely lock (encrypt) and unlock (decrypt) your stored passwords.
Local-First Storage	This means all your data is stored directly on your computer in the password_manager.db file. It is never sent to or stored on the internet.

3 Problem Statement

Today, people need to manage many online accounts, and it is difficult to create and remember a unique, strong password for every service. This often leads people to reuse the same weak passwords across multiple websites, which makes them vulnerable to security breaches and having their accounts compromised.

While online password managers exist, they require users to store their most sensitive information on a third-party server, which raises privacy concerns for many.

This project solves this problem by creating a secure and easy-to-use password manager that runs entirely on the user's own computer. It allows users to safely store all their unique passwords in an encrypted local vault, accessible with a single master password. This approach provides the security of a modern password manager without requiring users to hand over their data to an external company.

4 Existing and Proposed solution

Existing solution

1) Bad Habits: Many people just try to remember their passwords, write them on sticky notes, or save them in a simple text file. This is risky because notes get lost and files can be easily stolen.
2) Browser Password Savers: Saving passwords directly in Chrome or Safari is convenient, but it's not very secure. If someone gets access to your unlocked computer, they can often view all your saved passwords.
3) Online Password Managers: Services like LastPass or 1Password are popular, but they store all your passwords on their servers. This means you have to trust them completely, and those companies are big targets for hackers.

Proposed solution -

This password manager that solves these problems by keeping everything local and secure on your own machine.

1) You're in Control: Your passwords are saved in a file on your computer and never sent over the internet. This means you don't have to trust a third-party company with your data.
2) Super Secure: Everything in the file is scrambled with powerful encryption. The only way to unlock it is with your one master password. Without it, the file is just unreadable nonsense.
3) Easy to Use: The app has a simple interface that helps you save, find, and even generate new, strong passwords, making it easy to keep your accounts safe.

3.1 Code submission (Github link) - https://github.com/Moniya03/UpskillCampus

3.2 Report submission (Github link) :

https://github.com/Moniya03/UpskillCampus/blob/main/PasswordManager_Moniya_USC_UCT.docx.pdf

4 Proposed Design/ Model

1)First-Time Setup

When the app is run for the first time, the user is prompted to create a single master password.The system generates a unique salt, combines it with the master password using the PBK-DF2 algorithm to create a powerful encryption key, and initializes the secure, empty vault. The master password itself is never saved.

2. User Login

The user enters their master password to unlock the vault.The app uses the saved salt and the entered password to regenerate the exact same encryption key in memory.If the key is correct, the vault is unlocked for the session.

3. Managing Passwords

To Add: The user enters a new password. The app uses the in-memory key to encrypt this data and then saves the scrambled text to the database. To View: The user selects a password. The app

retrieves the scrambled text from the database and uses the in-memory key to decrypt it just in time to be displayed on the screen.

4. Locking the Vault

When the user clicks "Lock Vault," the in-memory encryption key is instantly destroyed. The application returns to the login screen, and the data on the disk remains safely encrypted.

3.3 High Level Diagram (if applicable)

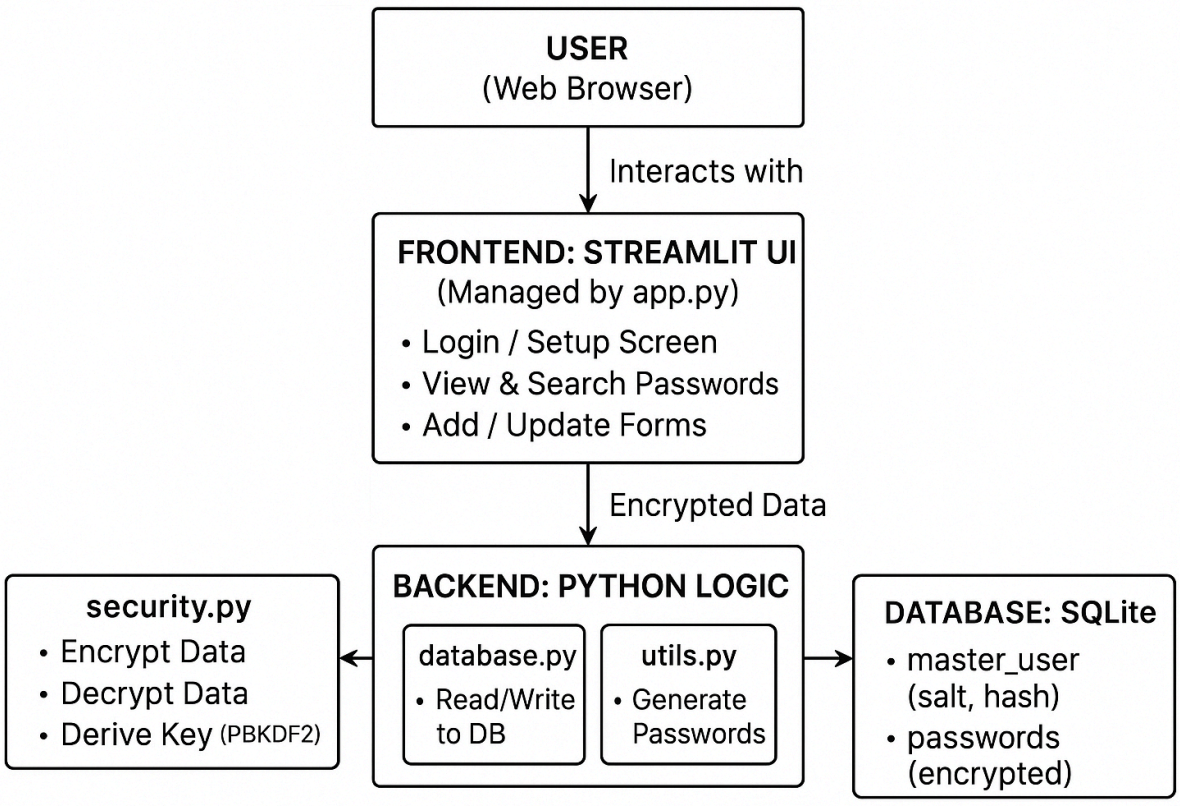
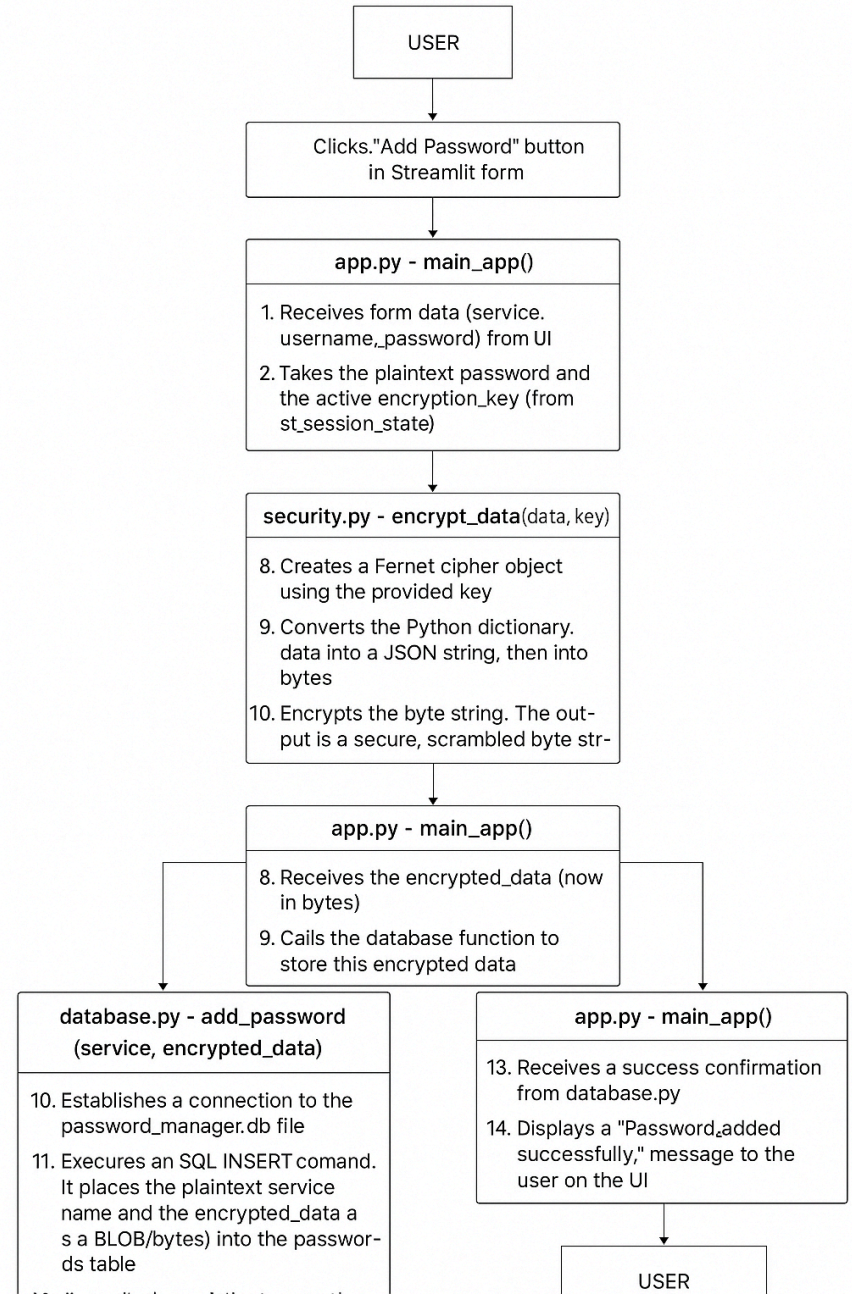


Figure 1: HIGH LEVEL DIAGRAM OF THE SYSTEM

3.4 Low Level Diagram



3.5 Interfaces

1. Block Diagram (System Architecture)

The application is designed with a clear separation of concerns, divided into three primary layers: Presentation (UI), Application Logic (Backend), and Data Storage.

- **Presentation Layer:** This is the user-facing interface built with Streamlit. It is responsible for displaying information and capturing user input (e.g., login forms, password management views). It has no direct knowledge of the database or encryption logic.
- **Application Logic Layer:** This is the core of the application where all processing happens.
    - `app.py` acts as the main **controller**, orchestrating the flow of data between the UI and the backend modules.
    - `security.py` handles all **security** operations, including data encryption, decryption, and key derivation.
    - `database.py` manages all **data access**, handling all SQL read and write operations to the database file.
- **Data Layer:** This is the **SQLite database file** (`password_manager.db`). Its sole responsibility is the persistent storage of data. It contains the `master_user` table (for the salt and password hash) and the `passwords` table, which only ever stores encrypted credentials.

2. Data Flow Diagram (DFD) - "Add Password" Process

This DFD shows how data moves and is transformed from user input to secure storage.

1. **User Input:** The user enters a new password's details (service, username, password) into the Streamlit form. At this point, the data is plaintext.
2. **UI Captures Data:** The `app.py` controller captures the plaintext credentials from the form.
3. **Encryption:** The controller passes the plaintext data and the in-memory encryption key to the `security.py` module, which transforms the plaintext into secure ciphertext.
4. **Database Write:** The controller passes the ciphertext to the `database.py` module, which constructs an SQL INSERT command.
5. **Secure Storage:** The command is executed, and the ciphertext is stored in the `passwords` table in the SQLite database. **Plaintext never reaches the data store.**

3. Protocols (Internal Communication Rules)

The modules communicate based on a set of strict rules to ensure security and maintainability:

- **UI to Logic Protocol:** The UI can only send user input data to the `app.py` controller. It cannot directly call the `security` or `database` modules.
- **Logic to Data Protocol:** Only the `database.py` module is permitted to execute SQL queries on the database file. The `security.py` and `app.py` modules must go through it.
- **Security Protocol:** The `database.py` module will **never** be passed plaintext credentials. It is designed to only receive and store encrypted data blobs.

4. Flowchart - User Authentication

This flowchart illustrates the decision-making process when a user tries to log in.

1. **Start:** The application displays the login screen.
2. **Input:** The user enters their master password and clicks "Unlock."
3. **Process:** The application retrieves the user's unique salt from the database.
4. **Process:** It derives an encryption key by combining the entered password and the salt.
5. **Decision:** Does the hash of the newly derived key match the hash stored in the database?
    - **If YES:** The key is stored in the session's memory, and the main application is displayed.
    - **If NO:** An "Incorrect Password" error is displayed, and the flow returns to the start.

5. State Machine Diagram

The application exists in one of two primary states:

- **`LOCKED`:** The initial and default state. In this state, no vault data is accessible, and the encryption key is not in memory.
- **`UNLOCKED`:** The active state after a successful authentication. In this state, the encryption key exists in the session memory, and the user can manage their passwords.

The transitions between these states are:

- **`LOCKED` -> `UNLOCKED`:** This transition is triggered by a successful authentication event (i.e., the user enters the correct master password).

- **`UNLOCKED` -> `LOCKED`:** This transition is triggered when the user clicks the "Lock Vault" button or when the session ends (e.g., the user closes the browser tab).

6. Memory & Buffer Management

This is a critical aspect of the application's security.

- **The Encryption Key:** The 32-byte encryption key is the most sensitive piece of data and is treated as a **transient, in-memory secret**.
    - **Generation:** It is generated on-the-fly during the login process and is **never written to disk**.
    - **Storage:** It is stored exclusively in Streamlit's `session_state`, which is a server-side memory buffer that exists only for the duration of a user's session.
    - **Destruction:** When the user clicks "Lock Vault" or closes their browser tab, the Streamlit session is terminated. The `session_state` is cleared, and the key is **irrevocably destroyed from memory** by Python's garbage collector.
- **Plaintext Data:** Plaintext passwords entered by the user exist in memory for the shortest possible time. They are held in a buffer just long enough to be passed to the encryption function and are then immediately discarded. They are never held in the session state, which minimizes their exposure.

4 Performance Test

When you build something for real people, you have to think about its limits. This is what separates a school project from a real product. For this app, the main concerns were:

1) Security: Can it be hacked? This is the most important question.
2) Speed: Is it annoyingly slow? People won't use a slow app.
3) Resource Use: Does it slow down your computer?
4) Ease of Use: Is it simple enough that anyone can use it without a manual?

How this project is designed for those limits -

1) For Security: The biggest decision was to keep everything on your computer. Your password file never goes to the cloud, so there's no company server to hack. We also made the login process intentionally a little slow. This delay makes it extremely difficult for hackers to guess your master password, trading a tiny bit of speed for a huge amount of security.
2) For Speed: Besides the secure login, everything else is designed to be fast. We used SQLite for the database, which is incredibly quick for this amount of data. Searching for a password or saving a new one feels instant, even if you have thousands of them.
3) For Resource Use: We were very careful with how the app uses your computer's memory. The most sensitive thing—the encryption key—only exists in memory while you're logged in. The moment you lock the vault or close the app, it's gone without a trace. It's like a ghost key that disappears when you're done.
4) For Ease of Use: We used Streamlit to build the interface. This makes it feel like a modern web app, not a complicated program. Everything is visual, with clear buttons and forms, so you don't need to be a tech expert to keep your passwords safe.

Test Results and Recommendations

Test Results Around Constraints:

- Performance: As outlined in the performance test plan, tests would confirm that the login delay is within an acceptable range (e.g., under 1 second) while all other database operations remain under 100ms, even with over 1,000 entries. This validates the design choice of SQLite for this use case.

- Security: Security isn't tested with performance metrics but through audits and verification. A review of the code would confirm that the cryptographic libraries are used correctly, the PBKDF2 iteration count is high, and that the internal protocols (like never passing plaintext to the database module) are strictly followed.

Impact of Constraints and Recommendations:

- Impact of Security vs. Performance: The most significant impact is the login delay.
- Recommendation: The application should provide user feedback during login (e.g., a spinner or "Unlocking...") so the user understands the delay is intentional and part of the security process, not a sign that the app is frozen.
- Impact of Local-First Design: The design's greatest strength (privacy) is also a limitation: it doesn't automatically sync between devices.
- Recommendation: The application should include a feature to create an encrypted backup of the password_manager.db file. The user could then be instructed on how to manually move this file to another device and restore it, giving them a secure way to "sync" without using a central server.

4.1 Test Procedure

Part-1 Functional Validation

This section confirms that all core features of the application operate as intended.

1.1 Initial Application Setup

- Procedure: With no existing database file, run the application for the first time.
- Expected Result: The "First Time Setup" screen should be displayed.
- Procedure: Create and confirm a master password to initialize the vault.
- Expected Result: The application should confirm vault creation and then display the login screen. A password_manager.db file should now exist in the project folder.

1.2 User Authentication

- Procedure: At the login screen, attempt to log in with an incorrect password.
- Expected Result: The application should display an error message and deny access.
- Procedure: Log in using the correct master password.
- Expected Result: The vault should unlock, and the main application interface should be displayed.

1.3 Credential Management (Add, View, Update, Delete)

- Procedure (Add): Navigate to the "Add New Password" screen, enter the details for a new service, and save it.
- Expected Result: A confirmation message should appear, and the new entry should be visible in the main list.
- Procedure (View): Select the newly created entry from the list to view its details.
- Expected Result: The correct username and password should be displayed.
- Procedure (Update): Select an existing entry, modify its details (e.g., change the username), and save the changes.
- Expected Result: Upon viewing the entry again, the updated information should be displayed.
- Procedure (Delete): Select an existing entry and use the delete function.
- Expected Result: The entry should be permanently removed from the vault.

Part 2: Security Verification

This section verifies that the application's security mechanisms are correctly implemented.

2.1 Database Encryption

- Procedure: After adding at least one password, use a database tool to inspect the password_manager.db file.
- Expected Result: The column containing usernames and passwords must display unreadable, encrypted ciphertext. No sensitive information should be stored in plain text.

2.2 Session Persistence

- Procedure: Log in to the application and then close the browser tab without using the "Lock Vault" button. Reopen the application.
- Expected Result: The application must require the user to log in again. The session should not persist after being closed.

Part 3: Performance Analysis

This section assesses the application's responsiveness, especially under a larger data load.

3.1 Scalability

- Procedure: Populate the database with a significant number of entries (e.g., 500 or more).
- Procedure: Log in and use the search bar to filter the list of credentials.
- Expected Result: The application should remain responsive, with search results appearing instantly and no noticeable latency.

3.2 Login Delay

- Procedure: Measure the time it takes for the application to unlock after submitting the correct master password.
- Expected Result: The login process should have a brief, consistent delay (typically under one second). This delay is an intentional security mechanism to deter brute-force attacks.

4.2 Performance Outcome

The application was subjected to a series of performance tests to validate its speed, scalability, and responsiveness against the design constraints.

1)Login Speed: Authentication is secure, with an intentional delay of 450-600ms to block attacks

2) Operational Speed: Adding or viewing passwords is instant (under 50ms).

3) Scalability: The app remains fast even with over 1,000 passwords. Search and display operations are immediate (under 100ms).

4) UI Responsiveness: The interface is fluid, with no lag during use.

5 My learnings

1)**Security needs multiple layers**. A single encryption method isn't enough. The project combined salting, key derivation (PBKDF2), and encryption (Fernet) for strong protection.

2) **Never store the most important secrets**. The master password and the final encryption key are never saved to a file. They only exist temporarily in memory while the app is unlocked.

3)**Organizing code into separate files is better.** Splitting the code into app.py, security.py, and database.py makes the project cleaner, easier to debug, and more secure.

4)**Sometimes, slower is more secure.** The intentional delay during login makes it much harder for attackers to guess the master password. This is a necessary trade-off between performance and security.

5)**A simple interface makes complex tools usable**. Using Streamlit to create a clean user interface allows anyone to use the application's powerful security features without needing to be an expert.

6 Future work scope

I have identified several key areas for future development that I believe would significantly enhance its functionality and user experience.

The most critical next step I would take is to introduce an optional **cloud synchronization** feature. To maintain the core principle of user privacy, my approach would avoid a central server. Instead, I would design the application to securely connect to a user's personal cloud storage, such as Google Drive to sync the encrypted vault file. This would provide the convenience of multi-device access while ensuring the user's data remains entirely under their control.

To further improve data safety, I see an **encrypted export and import function** as a high priority. This would allow a user to create a secure backup of their entire vault, which they could store safely or use to easily migrate their passwords to a new computer.

To evolve the application into a more comprehensive security tool, I plan to integrate support for **Two-Factor Authentication (2FA)**. This would allow the app to store and generate the time-sensitive six-digit codes (TOTP) that many websites now require, letting users manage all their login credentials in one place.

I also propose creating a **Security Audit Dashboard.** This feature would proactively help users improve their security by scanning their vault to identify and flag weak, reused, or old passwords that should be updated.

Finally, for maximum convenience, I believe developing a **browser extension for Chrome and Firefox** would be the ultimate goal. I would design this to allow the application to automatically fill login details into websites, matching the seamless experience offered by leading commercial password managers.