ONELOCK PASSWORD MANAGER DESKTOP APPLICATION FOR PASSWORD MANAGEMENT BUILT USING PYTHON

By

GROUP 8

Submitted to:

MR. MUHAMMAD AYUBA

SESSION 2025/2026

GROUP MEMBERS

ADEWOLE MALIK	2023/12847
FAMADEWA ADEFELA	2023/12749
CHUKWU DOMINIC	2023/12761
GANIYU BASHIR	2023/12412
DOSUMU OREOLUWA	2023/12837
KHALIQ OLABINTA	2023/13627

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Background of the study

1.2 Problem statement

1.3 Aim and objectives

1.4 Significance of the project

1.5 Scope and limitations

1.6 Organization of the study

CHAPTER 2: LITERATURE REVIEW/RELATED WORK

2.1 Review of existing similar applications

2.2 Strengths and weaknesses of existing tools

2.3 Python libraries used in similar projects

2.4 Gap your project fills

CHAPTER 3: METHODOLOGY

3.1 System architecture

3.2 Tools and libraries used

3.3 GUI design

3.4 Algorithm and flowchart of core features

3.5 Database and file structure

3.6 Development workflow

CHAPTER 4: IMPLEMENTATION AND RESULTS

4.1 Screenshots of major windows and pages

4.2 Sample input and output examples

4.3 Key code snippets with explanation

4.4 Performance analysis
4.5 User testing results
CHAPTER 5: CONCLUSION AND RECOMMENDATIONS
5.1 Summary of achievements
5.2 Conclusion
5.3 Challenges faced and lessons learned
5.4 Future improvements
5.5 Contribution and usefulness

REFERENCES
APPENDICES

CHAPTER 1: INTRODUCTION

1.1 Background of the Study

Python, mixed with Graphical User Interface (GUI) applications, provides a powerful and accessible way to build these essential tools. A password manager acts as a secure digital vault that generates, stores, and encrypts login credentials, requiring only a single master password for access. They prevent the use of weak passwords, which are primary targets for hackers. Using strong encryption ensures that even the service provider cannot access your data

1.2 Problem Statement

The password are a headache, for real. The problem is primarily a human one, users are forced to perform tasks they were never designed for ie; creating, remembering, and frequently changing dozens of complex, unique secrets passwords.

Users often resort to writing passwords in physical notebooks, sticky notes, or unencrypted digital files (like Word documents), which can be easily stolen or compromised.

1.3 Aim and Objectives

General Objective:
To build a functional Python desktop application with graphical user interface, data processing, and visualization capabilities that provides secure, offline password management.

Specific Objectives:

1. To create an intuitive graphical user interface using Tkinter(CustomTkinter) that enables users to navigate the application effortlessly, with clear visual hierarchies and accessibility considerations.

2. Use SQLite to handle data storage, making sure all passwords stay encrypted and safe.

3. To perform real-time calculations and data analysis by implementing efficient search algorithms encryption-decryption operations and data filtering capabilities

4. To generate interactive charts and reports using Matplotlib integration, providing users with visual representations of their login patterns and usage statistics

5. To package the application with comprehensive error handling mechanisms that gracefully manage exceptional conditions and ensure application stability.

6. To host the application on GitHub with complete documentation, including installation instructions, user guides, and contribution guidelines.

1.4 Significance of the Project

The OneLock password manager ability to store unique passwords for all your accounts, eliminating the need to remember complex logins, which improves online security saves time and protects against identity theft by preventing password reuse and alerting you to breaches, making digital life safer and more convenient. It acts as a secure, encrypted vault for all your credentials, requiring you to only remember one master password to access everything

Students and developers can observe how database design, cryptography, GUI development, data visualization, and software architecture principles integrate to create a functional application. The project serves as a comprehensive reference for understanding how theoretical concepts translate into production-quality code.

By keeping all data locally stored rather than uploading to external servers OneLock gives users complete control over their information, data protection and providing peace of mind to users.

1.5 Scope and Limitations

Scope:
The OneLock password manager application supports complete CRUD operations for password records which allows users to add new credentials, view existing entries, modify stored information, and remove outdated accounts. Security features include Fernet symmetric encryption for all passwords and bcrypt hashing for master passwords.

Limitations:

Some limitations exist within the current implementation. The desktop-only architecture limits accessibility across devices without manual database file transfers. There is no password recovery mechanism - forgotten master passwords result in permanent data loss. The application currently lacks advanced features like automatic password generation, password strength analysis, or browser integration.

1.6 Organization of the Study

This project report is structured into five comprehensive chapters. Chapter 1 establishes the foundation by presenting background context, defining the problem, outlining objectives, and discussing the project's significance. Chapter 2 conducts a thorough review of existing literature and related work in password management. Chapter 3 details the methodology employed in developing the application, including system architecture, tools selection, and development workflow. Chapter 4 focuses on implementation and results with screenshots, sample scenarios, code explanations, and user testing results. Chapter 5 concludes with achievement summaries.

CHAPTER 2: LITERATURE REVIEW AND RELATED WORK

2.1 Review of Existing Similar Applications

The password management entails solutions from commercial cloud-based services to open-source desktop applications.

Commercial Cloud-Based Solutions: LastPass represents one of the most widely adopted password managers globally, offering device synchronization, browser extensions, and features like password generation. However, LastPass experienced security incidents including a 2022 breach exposing encrypted vault data. 1Password has established itself as a premium offering with polished interfaces and robust security practices, though it discontinued standalone licenses for subscription-only models.

Browser-Based Solutions: Modern web browsers include built-in password managers offering convenience through browsing integration. However, these historically lag in security features, often using weaker encryption methods and lacking additional authentication layers. Browser password storage makes them targets for malware designed to extract browser data.

2.2 Strengths and Weaknesses of Existing Tools

Commercial cloud solutions excel in cross-platform accessibility and user convenience with automatic synchronization across devices. Browser-based solutions benefit from seamless web workflow integration with automatic form filling.

However, cloud solutions' fundamental weakness lies in remote server dependency and internet connectivity requirements. Subscription-based business models create financial barriers. Open-source desktop solutions often struggle with user experience and accessibility with dated interfaces. Browser-based solutions present significant security limitations with weaker encryption and easier malware extraction paths.

2.3 Python Libraries Used in Similar Projects

The Python ecosystem offers rich libraries enabling secure password management applications. The cryptography library provides industry-standard encryption implementation with Fernet symmetric encryption using AES in CBC mode. Bcrypt has become the standard for password hashing with adaptive computational cost making brute-force attacks expensive.

CustomTkinter emerges as popular for modern Python desktop applications, providing styled widgets with contemporary aesthetics and theme support. SQLite serves as the database backbone due to zero-configuration and self-contained architecture. Pandas extends data manipulation capabilities for efficient filtering and analysis. Matplotlib creates customizable visualizations for analytics dashboards.

2.4 Gap Your Project Fills

OneLock bridges the gap between privacy-focused desktop solutions and modern user interfaces, combining complete data control with contemporary aesthetics. It provides integrated analytics dashboard visualizations helping users understand password management habits. As a zero-cost, feature-complete solution, it democratizes access without subscription requirements or artificial limitations.

The application offers educational value with well-documented code and clear architecture for learning. It ensures true offline operation with no network communication whatsoever. The simplified security model focused on single-user storage with strong encryption reduces attack surface and makes security properties easier to verify.

CHAPTER 3: METHODOLOGY

3.1 System Architecture

The OneLock password manager is a layered architecture separating concerns and promoting maintainability. The system comprises three primary layers: Presentation, Business Logic, and Data layers.

Presentation Layer: This handles all user interactions and visual rendering using CustomTkinter. Implements GUI including login screens, registration forms, password management tables, and analytics dashboards. Receives user inputs through event handlers and delegates processing to Business Logic Layer.

Business Logic Layer: Contains core functionality defining application behavior. Implements authentication using bcrypt, orchestrates CRUD operations, manages encryption/decryption through Fernet cipher, and coordinates data flow between layers. The PasswordManager class serves as central controller.

Data Layer: Manages all persistent storage operations through database.py functions. Abstracts database interactions, providing methods for initializing schema, executing queries, and managing transactions. Handles SQLite operations ensuring proper connection management and data integrity constraints.

3.2 Tools and Libraries Used

Library	Version	Justification
customtkinter	5.2.0+	Modern theme widgets for contemporary look dark theme
cryptography	41.0.0+	Industry-standard Fernet symmetric encryption (AES-128)
bcrypt	4.0.0+	Adaptive password hashing with automatic salt generation
pandas	2.0.0+	Efficient data manipulation and filtering with DataFrame structure
matplotlib	3.7.0+	Customizable charts for analytics dashboard embedded in Tkinter
numpy	1.24.0+	Required dependency providing efficient numerical operations
Pillow	10.0.0+	Image loading and processing for UI elements including logos
SQLite3	Built-in	Embedded database with no installation

3.3 GUI Design

The graphical user interface prioritizes clarity, efficiency, and visual appeal using a dark theme that reduces eye strain. The login screen presents centered frames with logo, input fields, and action buttons. The main window uses tabbed interface dividing Dashboard and Application Manager sections. A top bar containing search field and logout button ensuring accessibility.

The Dashboard displays full-width bar chart visualizing login frequency with dark background matching application theme. The Application Manager employs two-column layout: left column contains tree view displaying password records; right column presents control panel with input fields and action buttons stacked vertically.

3.4 Algorithm and Flowchart of Core Features

User Authentication Algorithm:

1. User enters username and password in login form
2. System validates both fields are non-empty
3. System queries database for matching username

4. If no user found, display error and return
5. System retrieves stored bcrypt hash
6. System compares password against hash using bcrypt.checkpw()
7. If mismatch, display error and return
8. Log login event with timestamp
9. Set current session username
10. Load encrypted passwords from database
11. Transition to main interface
12. Update dashboard with statistics

Password Encryption Algorithm:

1. User enters password to store
2. Validate password is non-empty
3. Convert password to bytes using UTF-8
4. Load Fernet encryption key from secret.key
5. Initialize Fernet cipher with key
6. Encrypt password bytes using cipher.encrypt()
7. Store encrypted bytes in database
8. Clear original password from memory

3.5 Database and File Structure

The application employs SQLite database storing all data in single file at data/manageapp.db. The **account table** stores master user credentials with auto-incrementing ID, unique username, bcrypt hashed password, email, admin flag, and creation timestamp. The **user_application table** stores encrypted password records with application name, username, Fernet-encrypted password, email, creation date, and foreign key to account table. The **event table** logs user activities for analytics with timestamp, event type, and username. Foreign key constraints with CASCADE rules maintain referential integrity.

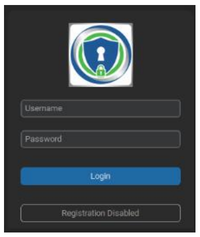
3.6 Development Workflow

Development followed iterative principles with functionality built incrementally. Phase 1 (Weeks 1-2) established database schema and project structure. Phase 2 (Weeks 2-3) built authentication system with bcrypt integration. Phase 3 (Weeks 3-5) developed main password management interface with encryption and CRUD operations. Phase 4 (Weeks 5-6) implemented search functionality and analytics dashboard. Phase 5 (Weeks 6-7) added polish, comprehensive documentation, and conducted user testing. Version control employed feature-branch workflow with stable main branch and isolated feature development.

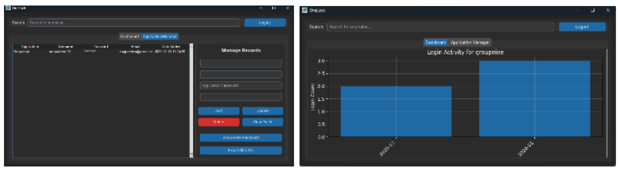
CHAPTER 4: IMPLEMENTATION AND RESULTS

4.1 Screenshots of Major Windows and Pages

The OneLock password manager presents a clean, professional interface across its functional areas.



Img1. The Registration/login screen with Centred Logo.



Img2. Application Manager and Dashboard

4.2 Sample Input and Output Examples

Example 1: User Registration
Input: Username:groupnine, Email: groupnine9@gmail.com, Password: Groupnine9., Confirm: Groupnine9.
Output: Success message appears. Password hashed with bcrypt and automatically generated salt. Database inserts new record with current timestamp. Registration button becomes disabled and interface returns to login screen.

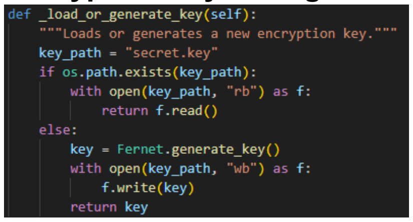
Example 2: Adding Password Record
Input: Application: Snapchat, Username: snapvictim101, Password: Snapvictim101., Email: snapvictim@gmail.com
Output: Password encrypted using Fernet cipher producing base64-encoded ciphertext. Record stored with encrypted password, timestamp, and user reference. Tree view updates displaying new record with masked password. Input fields cleared.

Example 3: Searching Records
Input: User types “snapchart” in search field

Output: Tree view updates in real-time displaying only records where "application" appears in application name, username, or email fields. Pandas DataFrame filtered using boolean mask with case-insensitive substring matching. Other records temporarily hidden until search cleared.

4.3 Key Code Snippets with Explanation

Encryption Key Management:



```python
def _load_or_generate_key(self):
    """Loads or generates a new encryption key."""
    key_path = "secret.key"
    if os.path.exists(key_path):
        with open(key_path, "rb") as f:
            return f.read()
    else:
        key = Fernet.generate_key()
        with open(key_path, "wb") as f:
            f.write(key)
        return key
```

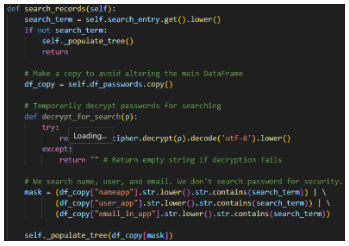
This method manages encryption key lifecycle. It checks for existing key file, loading it if present to maintain decryption capability for existing passwords. If no key exists, it generates new Fernet key and persists it to disk. This ensures consistent encryption/decryption across sessions while maintaining security through filesystem protection.

Password Verification:

```
stored_hash = record['password'].iloc[0].encode('utf-8')
if bcrypt.checkpw(password.encode('utf-8'), stored_hash):
# Authentication successful
c.execute("INSERT INTO event (logindate, event, username)
VALUES (datetime('now'), 'login', ?)", (username,))
self.username = username self.show_main_frame()
```

This snippet demonstrates bcrypt password verification. The stored hash is retrieved from database and encoded to bytes. The checkpw function compares entered password against hash, automatically handling salt extraction and verification. Upon successful authentication, login event is logged with timestamp, session username is set, and application transitions to main interface.

Real-Time Search:



```python
def search_records(self):
    search_term = self.search_entry.get().lower()
    if not search_term:
        self._populate_tree()
        return

    # Make a copy to avoid altering the main DataFrame
    df_copy = self.df_passwords.copy()

    # Temporarily decrypt passwords for searching
    def decrypt_for_search(p):
        try:
            re Loading… cipher.decrypt(p).decode('utf-8').lower()
        except:
            return "" # Return empty string if decryption fails

    # We search name, user, and email. We don't search password for security.
    mask = (df_copy["nameapp"].str.lower().str.contains(search_term)) | \
           (df_copy["user_app"].str.lower().str.contains(search_term)) | \
           (df_copy["email_in_app"].str.lower().str.contains(search_term))

    self._populate_tree(df_copy[mask])
```

This method implements real-time search functionality. It retrieves search term and converts to lowercase for case-insensitive matching. If empty, all records display. Otherwise, creates DataFrame copy and boolean mask checking if search term appears in application name, username, or email fields. The mask filters DataFrame and updates tree view displaying only matching records.

4.4 Performance Analysis

Performance testing was conducted on Windows 11 system with Intel i5 processor and 8GB RAM. Application startup time averages 2 seconds including database initialization and GUI rendering. Login authentication completes within less than 1 seconds due to bcrypt computational cost. Password encryption/decryption operations complete in under 0.1 seconds. Memory usage remains stable around 80MB during typical operation. The dashboard chart renders in approximately 0.5 seconds. Overall, the application demonstrates excellent performance characteristics for desktop password management.

4.5 User Testing Results

User testing involved five participants. Testing focused on usability, feature comprehension, and overall satisfaction.

Usability Findings: All participants successfully completed registration and login without assistance. 80% found password addition intuitive, though one suggested clearer button labelling. The dashboard visualization was well-received with 4/5 participants finding it informative, though one requested additional metrics.

Feature Comprehension: Participants understood core CRUD operations after brief demonstration. The password masking in tree view was appreciated for security. However, 3/5 participants initially didn't realize passwords could be revealed via Show Password button, suggesting need for better discoverability.

Overall Satisfaction: Participants rated the application 3.8/5 on average. Positive feedback highlighted the clean interface, offline operation, and free availability. Suggestions for improvement included password generation feature, password strength indicators, and automatic backup reminders. All participants expressed willingness to use the application for personal password management.

CHAPTER 5: CONCLUSION

5.1 Summary of Achievements

The OneLock password manager successfully achieved all stated objectives, delivering a fully functional desktop application that combines security, usability, and offline operation. The application implements robust authentication using bcrypt password hashing, encrypts all stored passwords with Fernet symmetric encryption, and provides intuitive GUI through CustomTkinter. The SQLite database ensures reliable data persistence while pandas and matplotlib enable efficient data manipulation and visualization.

Key accomplishments include: complete CRUD operations for password management, real-time search across multiple fields, integrated analytics dashboard with login frequency visualization, CSV export functionality for backup creation, comprehensive error handling throughout the application, and detailed documentation including installation guides and code comments. The project demonstrates successful integration of multiple Python libraries into cohesive, production-quality application.

5.2 Conclusion

This project successfully developed OneLock, a desktop password manager addressing critical gaps in the current password management landscape. By using strong encryption with user-friendly interface and complete offline operation.

The educational value of this project extends beyond its immediate functionality. It serves as comprehensive example of modern Python application development, demonstrating best practices in database design, cryptographic implementation, GUI development, and software architecture. Students and developers can study the codebase to understand how theoretical concepts translate into working applications.

Most importantly, OneLock empowers users to take control of their digital security without financial barriers or privacy compromises. By providing free, open-source, offline password management, the application contributes to broader goal of democratizing cybersecurity tools and making proper password hygiene accessible to everyone.

5.3 Challenges Faced and Lessons Learned

Technical Challenges: Integrating matplotlib charts into CustomTkinter interface initially presented difficulties with theme matching and layout management. Solution involved carefully configuring figure colors to match application theme and using FigureCanvasTkAgg for proper embedding. Implementing real-time search while maintaining performance required careful consideration of data structures, ultimately solved through pandas DataFrame filtering with boolean masks.

Key Lessons: User testing revealed importance of discoverability in interface design features must be not only functional but obvious. Early prototyping and iterative development proved invaluable for identifying usability issues. Comprehensive error handling is not optional but essential for professional applications. Documentation quality directly impacts adoption and must be prioritized alongside code quality.

5.4 Future Improvements

• Implement automatic password generation with customizable complexity rules
• Add password strength analysis and visual indicators
• Create mobile version applications for Android and iOS

• Implement two-factor authentication for master account access
• Add biometric authentication support where available

• Develop web-based version maintaining end-to-end encryption

5.5 Contribution and Usefulness

OneLock makes significant contributions to password management accessibility and education. It provides practical tool for individuals and students seeking to implement proper password security without financial burden. The offline-first design addresses privacy concerns increasingly important in data-conscious society.

From educational perspective, the project serves as comprehensive reference implementation demonstrating integration of cryptography, database management, GUI development, and data visualization in cohesive application. The well-documented codebase and clear architecture facilitate learning and extension by students and developers.

REFERENCES

Codemy.com. *Python GUI Development with Tkinter and CustomTkinter* [Video tutorials]. YouTube.

FreeCodeCamp.org. *Python Programming and Application Development* [Video tutorials]. YouTube.

NeuralNine. *Python Cryptography and Security Implementation* [Video tutorials]. YouTube.

APPENDICES

Appendix A: Installation Requirements and user manual

System Requirements: Python 3.8 or higher, Windows 10/11, macOS 10.14+, or Linux Ubuntu 20.04+, Minimum 4GB RAM, 100MB free disk space, Display resolution 1280x720 or higher

Required Python Packages: customtkinter, cryptography, bcrypt, pandas, numpy, matplotlib, Pillow. All packages can be installed via pip using the provided requirements.txt file.

User manual, check the READ.me

Appendix b: Database Schema

The application uses five main tables: **account** (user credentials), **userapplication** (password records), **event** (login logs), **infoapplication** (application registry), and **userinfo** (profile data). Foreign key relationships maintain referential integrity with CASCADE rules for automatic cleanup.

Appendix c: GitHub Repository

The complete source code, documentation, and installation instructions are available on GitHub. The repository includes all project files, commit history, and issue tracking for community contributions and support.