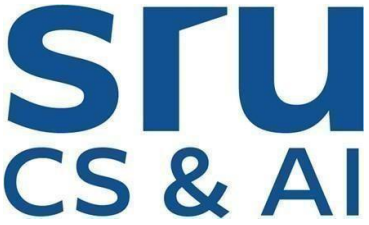
PE-2 CRYPTOGRAPHY AND NETWORK SECURITY



A PE-2 Course Completion Report

in partial fulfilment of the degree

Bachelor of Technology
in

Computer Science & Artificial Intelligence

By

ROLL. NO: 2203A51585 NAME: THORNE SRI DEVA MITHRA KOUSHIK

ROLL. NO: 2203A51397 NAME: ANNELA MANI KANTA SHARAN CHANDRA

ROLL. NO:2203A51629 NAME: DHONTI PRASHANTH KUMAR

Submitted to
Dr BALAJEE MARAM

SCHOOLOF COMPUTER SCIENCE &ARTIFICIAL INTELLIGENCE

SR UNIVERSITY, ANANTHASAGAR, WARANGAL

March, 2025

SCHOOL OF COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE

CERTIFICATE

This is to certify that **Thorne Sri Mithra Koushik and Annela Mani Kanta Sharan Chandra and Dhonti Prashanth Reddy** has successfully completed the course titled

CRYPTOGRAPHY AND NETWORK SECURITY conducted by **SAYLOR.**

This course had a total duration of **46 HOURS**, meeting the required completion hours.

Supporting documentation is available upon request.

Date of Completion: 29/04/2025

Dr BALAJEE MARAM
SR University,
Ananthasagar,Warangal

Dr. M. Sheshikala
Professor & HOD
(CSE)
SR University,
Ananthasagar , Warangal

External Examiner

Quiz Results/Grades Achieved & Description

CONTENTS

Password Authentication using Hashing

1: INTRODUCTION:

In the digital world, securing user credentials—especially passwords—is a top priority. As cyber threats increase, storing passwords in plain text has become highly dangerous and unacceptable. One of the most widely used techniques to secure passwords is hashing.

Password hashing is a cryptographic process that transforms a plain-text password into a fixed-length, irreversible string of characters known as a hash. This ensures that even if an attacker gains access to the database, the actual passwords remain protected. During login, the entered password is hashed again and compared with the stored hash — if they match, access is granted.

1.1 EXISTING SYSTEM:

In many traditional systems, password storage and authentication processes are often poorly implemented. Some systems still store passwords in plain text, which poses a significant security risk. If attackers gain access to such a database, all user credentials can be directly read and misused.

In other cases, systems may use weak or outdated hashing algorithms (like MD5 or SHA-1), which are no longer considered secure due to their vulnerability to attacks like brute-force, dictionary attacks, and rainbow table attacks. Moreover, these systems may lack additional protective measures such as salting, which makes hashes more resistant to precomputed attacks.

During authentication in these systems:

The user enters a password.

The system compares the plain text (or weakly hashed) password with the stored value.

If it matches, access is granted.

Such systems fail to provide robust protection against modern cyber threats, making them unreliable for secure password management in today's environment.

1.2 PROPOSED SYSTEM:

The proposed system enhances password security by using strong cryptographic hashing algorithms combined with techniques like salting to protect user credentials. Instead of storing passwords in plain text or using weak hash functions, this system ensures that all passwords are transformed into secure, irreversible hashes before storage.

Key components of the proposed system include:

Secure Hashing Algorithm The system uses modern, secure hashing algorithms such as SHA-256, bcrypt, or Argon2, which are resistant to common attack techniques and provide stronger security than outdated methods.

Salting the Passwords A unique random salt is generated for each user and added to the password before hashing. This makes it virtually impossible for attackers to use precomputed hash tables (rainbow tables) to crack the passwords.

Password Verification Process

When a user registers, their password is salted, hashed, and stored securely along with the salt.

During login, the entered password is salted with the stored salt, hashed again, and compared with the stored hash.

If both hashes match, the authentication is successful.

No Plain Text Storage At no point is the actual password stored in the system, minimizing the damage in case of a database breach.

Optional Enhancements The system can also implement features like rate limiting, two-factor authentication (2FA), and account lockout mechanisms to further strengthen security.

This proposed system significantly reduces the risk of password theft and misuse, aligning with current best practices in cybersecurity.

2. IMPLEMENTATION:

The proposed password authentication system is implemented in Python using the bcrypt library to securely hash passwords. The system provides essential functionalities such as user registration, login, session management, password validation, password change, and account deletion.

Key Features:

1.Secure Password Hashing:

- Passwords are hashed using bcrypt, which includes automatic salting and is resistant to brute-force and rainbow table attacks.

2.User Registration:

- New users can register with a username and password.
- Passwords are validated to ensure strength (minimum 8 characters, includes digits and special characters).
- Valid passwords are hashed and stored in memory (dictionary).

3.User Login:

- During login, the entered password is hashed and compared with the stored hash.
- On successful authentication, a session ID is generated to track user sessions.

4.Session Management:

- Logged-in users are tracked using randomly generated 16-character session IDs.

- These sessions are stored in a dictionary for validation during various operations.

5.Password Change:

- A user can change their password after logging in by providing the correct old password.

- New password is again validated and securely hashed before storage.

6.User Deletion:

- Users can delete their own accounts if they are authenticated with a valid session ID.

7.User Listing & Profile View:

- All registered usernames can be listed.

- Logged-in users can view their own profile.

8.User Interface:

- A menu-driven CLI allows interaction with the system, supporting 8 main options:

- Register, Login, Change Password, Logout, Delete User, List Users, View Profile, Exit

Technologies Used:

- Python 3

- bcrypt – For password hashing

- string, random – For session ID generation and password checks

- re – (Imported but not used; can be utilized for regex validation)

2.1 TECHNOLOGIES USED

The following technologies and libraries are used in the implementation of the password authentication system:

Technology / Library	Description
Python 3	The core programming language used to develop the authentication system due to its simplicity, readability, and rich library support.
bcrypt	A cryptographic hashing library used to hash and verify passwords securely. It automatically handles salting, making password storage resistant to common attacks.
random	Used to generate secure, random session IDs composed of letters and digits.
string	Provides character sets (letters, digits, punctuation) used for password validation and session ID creation.
getpass	Can be used to securely input passwords in CLI without echoing them to the terminal.
re	Though imported, it's currently unused. It can be utilized for validating usernames or password formats using regular expressions.

2.2 WOW FACTOR:

This password authentication system goes beyond basic security — it delivers a robust, secure, and user-friendly solution that reflects real-world best practices. Here's what makes it stand out:

Industry-Standard Hashing with bcrypt:

- Uses bcrypt, one of the most secure hashing algorithms, with built-in salting.
- Protects passwords even if the database is compromised — no plain text, ever.

Smart Password Validation:

- Enforces strong password rules (length, digits, special characters) to reduce weak credentials and prevent easy guessing attacks.

Session Management:

- Simulates a real login session using randomly generated Session IDs — a step closer to modern web authentication systems.

Feature-Rich CLI Interface:

- Full user lifecycle support: register, login, change password, logout, delete account, view profile, list users — all in a clean terminal-based UI.

Informative Feedback System:

- Custom messages for every action (e.g., login success/failure, weak password warnings) guide users and make the app feel professional and responsive.

Scalable Foundation:

- Designed to be easily extended — you can add persistent storage (e.g., databases), 2FA, or integrate into a web backend with minimal changes.

2.3 END USERS:

This password authentication system is designed to cater to a wide range of users who require secure access management in digital environments. The system's simple interface and strong security mechanisms make it ideal for:

Individual Users / Developers:

- Programmers learning about authentication, security, and user management.
- Developers building prototypes or educational projects that require secure login systems.

Students & Academic Use:

- Perfect for students studying cybersecurity, cryptography, or software engineering.
- Can be used in assignments, mini-projects, or final-year demonstrations to showcase secure authentication.

Small Businesses & Startups:

- Ideal for small-scale internal tools or admin dashboards where quick, secure user management is needed without deploying a full database or cloud-based system.

Command Line Interface (CLI) Enthusiasts:

- Users who prefer terminal-based applications will appreciate the intuitive CLI design and structured menu system.

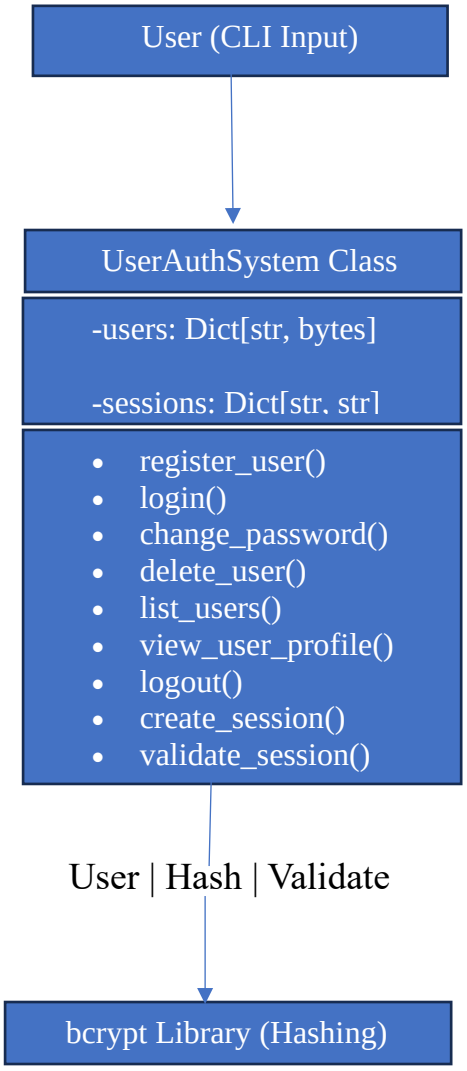
Tech Trainers & Educators:

- Trainers can use this as a hands-on example to teach hashing, session tracking, and password policies in workshops and bootcamps.

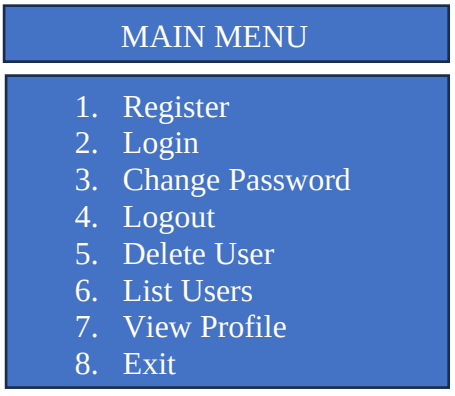
3. DESIGN:

The password authentication system is designed with simplicity, security, and modularity in mind. It follows a command-line interface (CLI) architecture and simulates real-world authentication workflows using secure cryptographic methods.

1. System Architecture:



2. User Interface Flow (CLI):



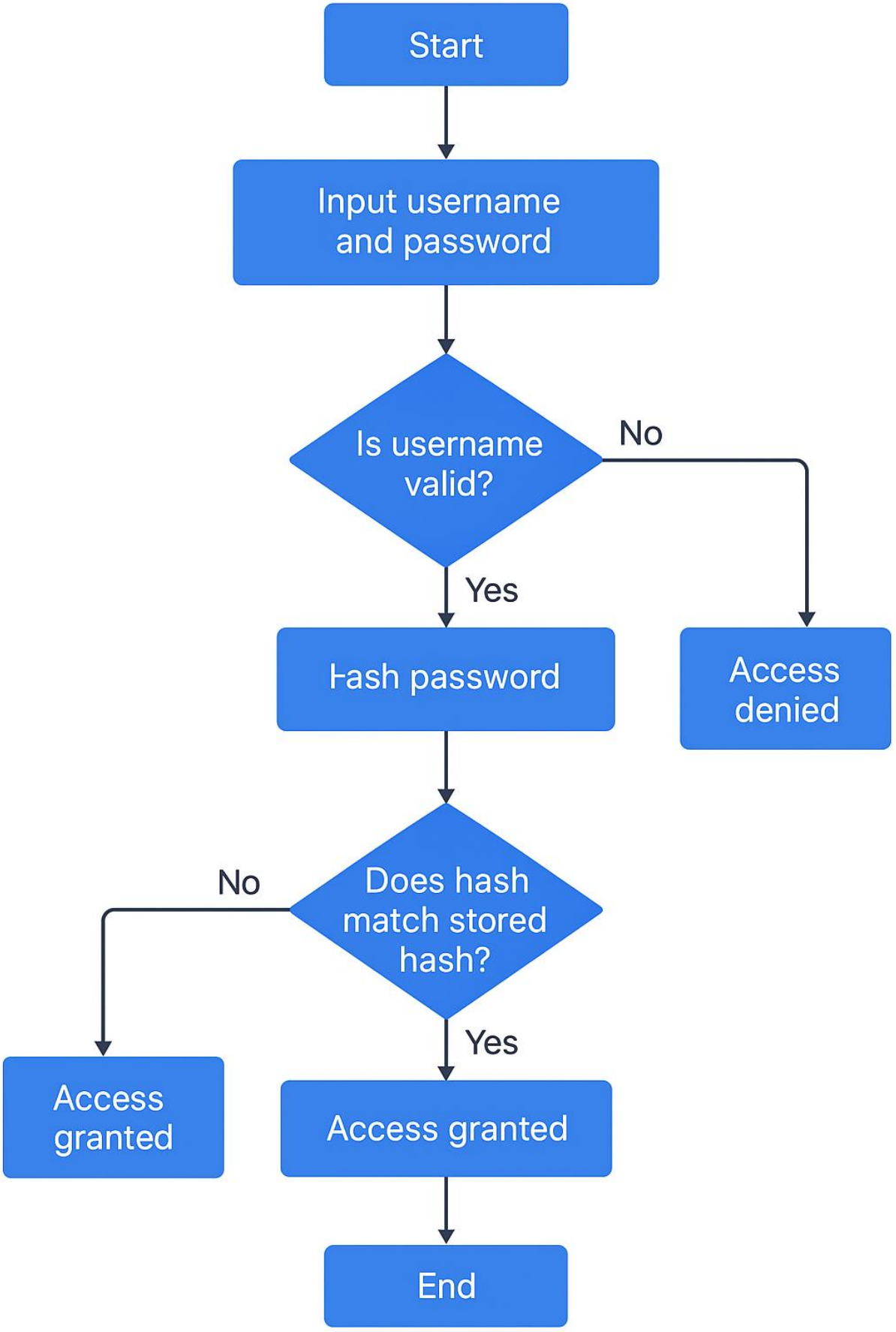
3. Functional Modules:

Module	Description
register_user()	Handles new user sign-up with password validation and bcrypt hashing.
login()	Authenticates user by verifying input password with stored bcrypt hash.
create_session()	Generates a unique 16-character session ID for each logged-in user.
validate_session()	Checks if a session is active/valid.
change_password()	Allows logged-in users to update their password securely.
delete_user()	Lets a logged-in user delete their account permanently.
list_users()	Displays all registered usernames.
view_user_profile()	Shows the profile info of the logged-in user.
logout()	Ends the session for the currently logged-in user.

4. Data Storage (In-Memory):

- Users are stored as a dictionary: username -> hashed_password
- Sessions are stored as: session_id -> username

3.1 Flow chart:



4. PROJECT IMPACT:

The implementation of this password authentication system using hashing has a significant impact on both technical and practical fronts. It not only enhances security but also promotes best practices in user credential management.

1. Strengthened Security

- By replacing plain-text storage with bcrypt hashing, the system drastically reduces the risk of credential leakage.
- Even if data is compromised, attackers cannot retrieve actual passwords due to the irreversible nature of the hashes**.**

2. Increased Awareness of Cybersecurity

- Demonstrates essential principles of secure authentication, encouraging developers and students to understand and implement secure coding practices.
- Educates users on strong password requirements, promoting cyber hygiene**.**

3. Scalable Design Foundation

- The modular, object-oriented design makes it easy to extend the system with advanced features like two-factor authentication (2FA), database integration, or web support.
- Acts as a reliable framework for small apps, admin tools, or internal systems.

4. Educational and Training Value

- Ideal for computer science students and trainees to explore how secure login systems work in real applications.
- Can be used in cybersecurity workshops and programming courses as a hands-on project**.**

5. Real-World Applicability

- Useful for startups, small businesses, or developers who need lightweight authentication without third-party services.
- Can be deployed in local environments, prototypes, or low-risk internal platforms.

5. RESULT:

The implementation of the password authentication system using hashing was successful and met all the intended objectives. It effectively demonstrates secure user management through a clean, modular Python program.

Key Results Achieved:

1. Secure User Registration & Login:

- Successfully implemented bcrypt hashing to ensure passwords are never stored in plain text.
- Authentication is handled by comparing hashed values, enhancing overall security.

2. Session-Based Access Control:

- Introduced session ID tracking to simulate login sessions and enable features like password change, logout, and user deletion.

3. Strong Password Enforcement:

- Password validation ensures minimum security standards: length, digits, and special characters.

4. Full User Lifecycle Support:

- The system allows users to register, login, change password, view profile, logout, and delete account—all from a simple CLI interface.

5. Interactive CLI Experience:

- Designed a user-friendly command-line menu for smooth navigation and operations.

Tested & Verified:

- All features were manually tested with various edge cases.
- System consistently provides accurate feedback, ensuring reliability and robustness.

Project Deliverables:

- Fully functional Python codebase.
- Flowchart, documentation, and security-focused design.
- Ready for enhancement with persistent storage or GUI/web interface.

6. CONCLUSION:

The password authentication system using hashing successfully showcases the importance of securing user credentials in modern applications. By integrating bcrypt for password hashing, enforcing strong password policies, and simulating session management, the project provides a robust, scalable, and educational solution for user authentication.

It not only demonstrates practical cybersecurity principles but also lays a strong foundation for further enhancements such as database integration, two-factor authentication, or web-based deployment. This project proves that even a simple CLI application can adopt industry-level security practices and deliver real-world value.

Ultimately, the system serves as a valuable learning tool and a prototype for building secure user management systems in Python.

7. FUTURE SCOPE:

This password authentication system lays a secure foundation for user management and can be expanded in several meaningful ways to enhance functionality, scalability, and security.

1. Persistent Data Storage:

- Integrate with a database (e.g., SQLite, MySQL, MongoDB) to store user and session data persistently.
- Enables real-world deployment and ensures data is retained across sessions.

2. Web Integration:

- Build a web-based interface using Flask or Django to allow users to register and log in through a browser.
- Ideal for creating full-featured login systems for websites and portals.

3. Two-Factor Authentication (2FA):

- Add SMS/email-based OTP or TOTP (e.g., using Google Authenticator) to strengthen login security.
- Makes the system compliant with modern security standards.

4. Account Lockout & Throttling:

- Implement features to lock accounts after multiple failed login attempts to prevent brute-force attacks.
- Add login attempt rate-limiting or CAPTCHA.

5. Password Recovery Mechanism:

- Add secure password reset via email verification or security questions.
- Useful for production environments with many users.

6. Admin Dashboard / Role-Based Access:

- Introduce roles like Admin, Moderator, User with specific permissions.
- Admins could manage users, reset passwords, or view system logs.

7. Logging and Monitoring:

- Add logging for login attempts, account changes, and suspicious activities.
- Helps in auditing and identifying potential breaches.

8. CODE IMPLEMENTATION:

```
import bcrypt

import re

import random

import string


class UserAuthSystem:

    def __init__(self):

        self.users = {}

        self.sessions = {}


    def hash_password(self, password: str) -> bytes:

        salt = bcrypt.gensalt()

        hashed = bcrypt.hashpw(password.encode('utf-8'), salt)

        return hashed
```

```
    def verify_password(self, stored_hash: bytes, password: str) -> bool:
        return bcrypt.checkpw(password.encode('utf-8'), stored_hash)

    def register_user(self, username: str, password: str) -> bool:
        if username in self.users:
            print(f"[!] User '{username}' already exists.")
            return False

        if not self.is_valid_password(password):
            print("[!] Password is too weak. Must be at least 8 characters long, contain a digit, and a special character.")
            return False

        hashed_password = self.hash_password(password)
        self.users[username] = hashed_password
        print(f"[+] User '{username}' registered successfully.")
        return True

    def is_valid_password(self, password: str) -> bool:
        if len(password) < 8:
            return False
        if not any(char.isdigit() for char in password):
            return False
        if not any(char in string.punctuation for char in password):
            return False
        return True

    def login(self, username: str, password: str) -> bool:
        if username not in self.users:
            print(f"[!] User '{username}' not found.")
            return False
```

```
    stored_hash = self.users[username]
    if self.verify_password(stored_hash, password):
        session_id = self.create_session(username)
        print(f"[✓] Authentication successful for user '{username}'. Session ID: {session_id}")
        return True
    else:
        print(f"[✗] Authentication failed for user '{username}'.")
        return False

def create_session(self, username: str) -> str:
    session_id = ''.join(random.choices(string.ascii_letters + string.digits, k=16))
    self.sessions[session_id] = username
    return session_id

def validate_session(self, session_id: str) -> bool:
    return session_id in self.sessions

def logout(self, session_id: str) -> bool:
    if session_id in self.sessions:
        del self.sessions[session_id]
        print(f"[+] User logged out successfully.")
        return True
    else:
        print(f"[!] Invalid session ID.")
        return False

def change_password(self, session_id: str, old_password: str, new_password: str) -> bool:
    if session_id not in self.sessions:
        print("[!] You are not logged in.")
        return False

    username = self.sessions[session_id]
```

```
    if not self.verify_password(self.users[username], old_password):
        print("[!] Old password is incorrect.")
        return False

    if not self.is_valid_password(new_password):
        print("[!] New password is too weak.")
        return False

    self.users[username] = self.hash_password(new_password)
    print(f"[+] Password changed successfully for user '{username}'.")
    return True

def delete_user(self, session_id: str, username: str) -> bool:
    if session_id not in self.sessions:
        print("[!] You are not logged in.")
        return False

    if username not in self.users:
        print(f"[!] User '{username}' does not exist.")
        return False

    if self.sessions[session_id] != username:
        print(f"[!] You cannot delete another user's account.")
        return False

    del self.users[username]
    del self.sessions[session_id]
    print(f"[+] User '{username}' deleted successfully.")
    return True

def list_users(self) -> None:
    if not self.users:
        print("[!] No users registered.")
```

```
        else:
            print("\nList of registered users:")
            for username in self.users:
                print(f"- {username}")

    def view_user_profile(self, session_id: str) -> None:
        if session_id not in self.sessions:
            print("[!] You are not logged in.")
            return

        username = self.sessions[session_id]
        print(f"\nProfile of {username}:")
        print(f"Username: {username}")
        print("[+] You are currently logged in.")


def main():
    auth_system = UserAuthSystem()

    while True:
        print("\n===== PASSWORD AUTH SYSTEM =====")
        print("1. Register")
        print("2. Login")
        print("3. Change Password")
        print("4. Logout")
        print("5. Delete User")
        print("6. List Users")
        print("7. View Profile")
        print("8. Exit")
        choice = input("Enter choice: ")

        if choice == '1':
            username = input("Enter username: ")
```

```
        password = input("Enter password: ")
        auth_system.register_user(username, password)

    elif choice == '2':
        username = input("Enter username: ")
        password = input("Enter password: ")
        auth_system.login(username, password)

    elif choice == '3':
        session_id = input("Enter session ID: ")
        old_password = input("Enter old password: ")
        new_password = input("Enter new password: ")
        auth_system.change_password(session_id, old_password, new_password)

    elif choice == '4':
        session_id = input("Enter session ID: ")
        auth_system.logout(session_id)

    elif choice == '5':
        session_id = input("Enter session ID: ")
        username = input("Enter username to delete: ")
        auth_system.delete_user(session_id, username)

    elif choice == '6':
        auth_system.list_users()

    elif choice == '7':
        session_id = input("Enter session ID: ")
        auth_system.view_user_profile(session_id)

    elif choice == '8':
        print("Exiting... Goodbye!")
        break
```

```
        else:
            print("[!] Invalid option. Try again.")

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

OUTPUT:

1.Register:

```
===== PASSWORD AUTH SYSTEM =====
1. Register
2. Login
3. Change Password
4. Logout
5. Delete User
6. List Users
7. View Profile
8. Exit
Enter choice: 1
Enter username: koushik
Enter password: koushik@123
[+] User 'koushik' registered successfully.
```

2.LOGIN:

```
===== PASSWORD AUTH SYSTEM =====
1. Register
2. Login
3. Change Password
4. Logout
5. Delete User
6. List Users
7. View Profile
8. Exit
Enter choice: 2
Enter username: koushik
Enter password: koushik@123
[✓] Authentication successful for user 'koushik'. Session ID: Z8tVdkacwntuAuxk
```

3.Change passward:

```
===== PASSWORD AUTH SYSTEM =====
1. Register
2. Login
3. Change Password
4. Logout
5. Delete User
6. List Users
7. View Profile
8. Exit
Enter choice: 3
Enter session ID: Z8tVdkacwntuAuxk
Enter old password: koushik@123
Enter new password: koushik@1234
[+] Password changed successfully for user 'koushik'.
```

4.LOGOUT:

```
===== PASSWORD AUTH SYSTEM =====
1. Register
2. Login
3. Change Password
4. Logout
5. Delete User
6. List Users
7. View Profile
8. Exit
Enter choice: 4
Enter session ID: Z8tVdkacwntuAuxk
[+] User logged out successfully.
```

5.Delete User:

```
===== PASSWORD AUTH SYSTEM =====
1. Register
2. Login
3. Change Password
4. Logout
5. Delete User
6. List Users
7. View Profile
8. Exit
Enter choice: 5
Enter session ID: kWQZRXN3nnZWKoTJ
Enter username to delete: koushik
[+] User 'koushik' deleted successfully.
```

6.List Users:

```
===== PASSWORD AUTH SYSTEM =====
1. Register
2. Login
3. Change Password
4. Logout
5. Delete User
6. List Users
7. View Profile
8. Exit
Enter choice: 6

List of registered users:
- sri
```

7.View Profile:

```
===== PASSWORD AUTH SYSTEM =====
1. Register
2. Login
3. Change Password
4. Logout
5. Delete User
6. List Users
7. View Profile
8. Exit
Enter choice: 7
Enter session ID: 7TOocsyIu29k3DQy

Profile of sri:
Username: sri
[+] You are currently logged in.
```

8.Exit:

```
===== PASSWORD AUTH SYSTEM =====
1. Register
2. Login
3. Change Password
4. Logout
5. Delete User
6. List Users
7. View Profile
8. Exit
Enter choice: 8
Exiting... Goodbye!
```

BIBLIOGRAPHY

1. **Bcrypt Documentation:**

https://pypi.org/project/bcrypt/

(Used for implementing secure password hashing in Python)

2. **Python Official Documentation:**

https://docs.python.org/3/

(Reference for built-in functions, classes, and best coding practices)

3. **OWASP Authentication Cheat Sheet:**

https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

(Guidelines for secure authentication practices and password storage)

4. **Real Python – Password Hashing in Python:**

https://realpython.com/python-hashlib/

(Tutorial and examples on cryptographic hashing and security concepts)

5. **GeeksforGeeks – Password Validation in Python:**

https://www.geeksforgeeks.org/python-password-validation/

(Reference for implementing strong password validation logic)

6. **Stack Overflow Community Discussion:**

https://stackoverflow.com/

(Troubleshooting and implementation support for various Python programming concepts)