Python Project 14: Authentication System

1. Project Overview

Build a simple desktop **Authentication System** with user registration and login.

The system stores usernames and **salted password hashes** instead of storing plain text passwords. Python's `hashlib.pbkdf2_hmac()` is designed for password key derivation, and Python recommends using a salt from a secure random source.

2. Features

- User registration
- Username validation
- Password validation
- Confirm password
- Secure password hashing
- Unique salt for each user
- User login
- Login validation
- Logout
- SQLite database
- Password visibility toggle
- Simple Tkinter interface
- No external packages

3. Technologies

- Python 3
- Tkinter
- SQLite
- `sqlite3`

- hashlib
- secrets
- hmac
- Object Oriented Programming

SQLite queries use parameterized placeholders instead of string concatenation, which Python's documentation recommends to avoid SQL injection vulnerabilities.

4. Folder Structure

```
authentication_system/
□□□ main.py
□□□ ui.py
□□□ auth.py
□□□ database.py
□□□ README.md
```

5. How the Project Works

Registration

```
User enters username + password
        □
Validate input
        □
Generate random salt
        □
Hash password with PBKDF2
        □
Save username + salt + hash
        □
Registration complete
```

Login

```
User enters username + password
        ↓
Find user in database
        ↓
Retrieve stored salt + hash
        ↓
Hash entered password with same salt
        ↓
Compare hashes
        ↓
Login successful / failed
```

Passwords are not stored as plain text. The `secrets` module provides cryptographically strong randomness, while `hmac.compare_digest()` can be used for safer hash comparison.

6. Complete Authentication Backend

auth.py

```
import hashlib
import hmac
import secrets


class AuthManager:

    ITERATIONS = 500_000
    SALT_LENGTH = 16
    HASH_LENGTH = 32

    def __init__(self, database):
        self.database = database

    def validate_registration(self, username, password, confirm):
```

```
      username = username.strip()

      if not username:
         raise ValueError("Username is required.")

      if len(username) < 3:
         raise ValueError(
            "Username must contain at least 3 characters."
         )

      if len(password) < 8:
         raise ValueError(
            "Password must contain at least 8 characters."
         )

      if password != confirm:
         raise ValueError(
            "Passwords do not match."
         )

   def hash_password(self, password, salt=None):
      if salt is None:
         salt = secrets.token_bytes(
            self.SALT_LENGTH
         )

      password_bytes = password.encode("utf-8")

      password_hash = hashlib.pbkdf2_hmac(
         "sha256",
         password_bytes,
         salt,
         self.ITERATIONS,
         dklen=self.HASH_LENGTH
```

```
        )

        return salt, password_hash

    def register(self, username, password, confirm):
        username = username.strip()

        self.validate_registration(
            username,
            password,
            confirm
        )

        if self.database.user_exists(username):
            raise ValueError(
                "Username already exists."
            )

        salt, password_hash = self.hash_password(
            password
        )

        self.database.create_user(
            username,
            salt.hex(),
            password_hash.hex()
        )

    def login(self, username, password):
        username = username.strip()

        if not username or not password:
            return False
```

```
        user = self.database.get_user(username)

        if user is None:
            return False

        stored_salt = bytes.fromhex(user["salt"])
        stored_hash = bytes.fromhex(user["password_hash"])

        _, password_hash = self.hash_password(
            password,
            stored_salt
        )

        return hmac.compare_digest(
            password_hash,
            stored_hash
        )
```

7. Complete Database Code

database.py

```
import sqlite3


class UserDatabase:

    def __init__(self, database_name="users.db"):
        self.database_name = database_name
        self.create_table()

    def connect(self):
        return sqlite3.connect(self.database_name)

    def create_table(self):
```

```
        connection = self.connect()
        connection.row_factory = sqlite3.Row

        cursor = connection.cursor()

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE NOT NULL,
                salt TEXT NOT NULL,
                password_hash TEXT NOT NULL
            )
        """)

        connection.commit()
        connection.close()

    def create_user(
        self,
        username,
        salt,
        password_hash
    ):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            INSERT INTO users
            (username, salt, password_hash)
            VALUES (?, ?, ?)
        """, (
            username,
            salt,
            password_hash
```

```
        ))

        connection.commit()
        connection.close()

    def user_exists(self, username):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            SELECT id
            FROM users
            WHERE username = ?
        """, (username,))

        result = cursor.fetchone()

        connection.close()

        return result is not None

    def get_user(self, username):
        connection = self.connect()
        connection.row_factory = sqlite3.Row

        cursor = connection.cursor()

        cursor.execute("""
            SELECT id, username, salt, password_hash
            FROM users
            WHERE username = ?
        """, (username,))

        result = cursor.fetchone()
```

```
        connection.close()

        if result is None:
            return None

        return dict(result)
```

8. Complete Frontend Code

ui.py

```
import tkinter as tk
from tkinter import messagebox

from auth import AuthManager
from database import UserDatabase


class AuthenticationUI:

    def __init__(self, root):
        self.root = root
        self.root.title("Authentication System")
        self.root.geometry("500x560")
        self.root.configure(bg="#0f172a")

        self.database = UserDatabase()
        self.auth = AuthManager(self.database)

        self.username_var = tk.StringVar()
        self.password_var = tk.StringVar()
        self.confirm_var = tk.StringVar()

        self.create_login_screen()
```

```
    def clear_screen(self):
        for widget in self.root.winfo_children():
            widget.destroy()

    def create_login_screen(self):
        self.clear_screen()

        title = tk.Label(
            self.root,
            text="Welcome Back",
            font=("Arial", 25, "bold"),
            bg="#0f172a",
            fg="white"
        )
        title.pack(pady=(55, 5))

        subtitle = tk.Label(
            self.root,
            text="Login to your account",
            font=("Arial", 11),
            bg="#0f172a",
            fg="#94a3b8"
        )
        subtitle.pack(pady=(0, 30))

        form = tk.Frame(
            self.root,
            bg="#1e293b",
            padx=30,
            pady=30
        )
        form.pack(
            padx=55,
```

```
            fill="x"
        )

        tk.Label(
            form,
            text="Username",
            bg="#1e293b",
            fg="white",
            font=("Arial", 10, "bold")
        ).pack(anchor="w")

        username_entry = tk.Entry(
            form,
            textvariable=self.username_var,
            font=("Arial", 12),
            bg="#334155",
            fg="white",
            insertbackground="white",
            relief="flat"
        )
        username_entry.pack(
            fill="x",
            pady=(7, 18),
            ipady=8
        )

        tk.Label(
            form,
            text="Password",
            bg="#1e293b",
            fg="white",
            font=("Arial", 10, "bold")
        ).pack(anchor="w")
```

```
        password_entry = tk.Entry(
            form,
            textvariable=self.password_var,
            show="*",
            font=("Arial", 12),
            bg="#334155",
            fg="white",
            insertbackground="white",
            relief="flat"
        )
        password_entry.pack(
            fill="x",
            pady=(7, 10),
            ipady=8
        )

        show_password = tk.BooleanVar()

        tk.Checkbutton(
            form,
            text="Show password",
            variable=show_password,
            command=lambda: self.toggle_password(
                password_entry,
                show_password
            ),
            bg="#1e293b",
            fg="#94a3b8",
            activebackground="#1e293b",
            activeforeground="white",
            selectcolor="#334155"
        ).pack(anchor="w")

        tk.Button(
```

```
            form,
            text="Login",
            command=self.login,
            bg="#2563eb",
            fg="white",
            font=("Arial", 11, "bold"),
            relief="flat",
            padx=20,
            pady=10
        ).pack(
            fill="x",
            pady=(25, 10)
        )

        tk.Button(
            form,
            text="Create Account",
            command=self.create_register_screen,
            bg="#334155",
            fg="white",
            font=("Arial", 10),
            relief="flat",
            pady=8
        ).pack(fill="x")

        username_entry.focus()

        self.root.bind(
            "<Return>",
            lambda event: self.login()
        )

    def create_register_screen(self):
        self.clear_screen()
```

```
        title = tk.Label(
            self.root,
            text="Create Account",
            font=("Arial", 25, "bold"),
            bg="#0f172a",
            fg="white"
        )
        title.pack(pady=(40, 5))

        subtitle = tk.Label(
            self.root,
            text="Register a new user",
            font=("Arial", 11),
            bg="#0f172a",
            fg="#94a3b8"
        )
        subtitle.pack(pady=(0, 25))

        form = tk.Frame(
            self.root,
            bg="#1e293b",
            padx=30,
            pady=25
        )
        form.pack(
            padx=55,
            fill="x"
        )

        self.create_field(
            form,
            "Username",
            self.username_var,
```

```
            False
        )

        self.create_field(
            form,
            "Password",
            self.password_var,
            True
        )

        self.create_field(
            form,
            "Confirm Password",
            self.confirm_var,
            True
        )

        tk.Button(
            form,
            text="Register",
            command=self.register,
            bg="#2563eb",
            fg="white",
            font=("Arial", 11, "bold"),
            relief="flat",
            pady=10
        ).pack(
            fill="x",
            pady=(20, 10)
        )

        tk.Button(
            form,
            text="Back to Login",
```

```
            command=self.create_login_screen,
            bg="#334155",
            fg="white",
            relief="flat",
            pady=8
        ).pack(fill="x")

    def create_field(
        self,
        parent,
        label,
        variable,
        password
    ):
        tk.Label(
            parent,
            text=label,
            bg="#1e293b",
            fg="white",
            font=("Arial", 10, "bold")
        ).pack(anchor="w")

        entry = tk.Entry(
            parent,
            textvariable=variable,
            show="*" if password else "",
            font=("Arial", 12),
            bg="#334155",
            fg="white",
            insertbackground="white",
            relief="flat"
        )

        entry.pack(
```

```
            fill="x",
            pady=(7, 15),
            ipady=8
        )

    def toggle_password(
        self,
        entry,
        variable
    ):
        entry.config(
            show="" if variable.get() else "*"
        )

    def register(self):
        try:
            self.auth.register(
                self.username_var.get(),
                self.password_var.get(),
                self.confirm_var.get()
            )

            messagebox.showinfo(
                "Success",
                "Account created successfully."
            )

            self.password_var.set("")
            self.confirm_var.set("")

            self.create_login_screen()

        except ValueError as error:
            messagebox.showerror(
```

```
                "Registration Error",
                str(error)
            )

        except Exception as error:
            messagebox.showerror(
                "Error",
                str(error)
            )

    def login(self):
        username = self.username_var.get()
        password = self.password_var.get()

        if self.auth.login(username, password):
            self.create_dashboard(username)
        else:
            messagebox.showerror(
                "Login Failed",
                "Invalid username or password."
            )

    def create_dashboard(self, username):
        self.root.unbind("<Return>")
        self.clear_screen()

        title = tk.Label(
            self.root,
            text="Dashboard",
            font=("Arial", 27, "bold"),
            bg="#0f172a",
            fg="white"
        )
        title.pack(pady=(80, 10))
```

```
        welcome = tk.Label(
            self.root,
            text=f"Welcome, {username}",
            font=("Arial", 16),
            bg="#0f172a",
            fg="#60a5fa"
        )
        welcome.pack(pady=10)

        description = tk.Label(
            self.root,
            text="You are successfully authenticated.",
            font=("Arial", 11),
            bg="#0f172a",
            fg="#94a3b8"
        )
        description.pack(pady=10)

        tk.Button(
            self.root,
            text="Logout",
            command=self.create_login_screen,
            bg="#334155",
            fg="white",
            font=("Arial", 11, "bold"),
            relief="flat",
            padx=30,
            pady=10
        ).pack(pady=30)
```

9. Main Code

main.py

```
import tkinter as tk

from ui import AuthenticationUI


def main():
    root = tk.Tk()
    AuthenticationUI(root)
    root.mainloop()


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

10. README

README.md

```
# Authentication System

A Python desktop authentication system with
registration, login, logout, password hashing,
and SQLite storage.

## Features

- Register
- Login
- Logout
- Password hashing
- Random salt
- SQLite database
- Input validation

## Requirements
```

Python 3

No external packages are required.

Run

python main.py

11. How Frontend + Backend Connect

ui.py

AuthManager

Validate credentials

Hash / verify password

UserDatabase

SQLite

Authentication result

Login or Dashboard

12. How to Run

Open the project directory:

cd authentication_system

Run:

```
python main.py
```

Click **Create Account**, register a user, then return to the login screen.

13. Basic Testing

Registration

Test:

```
Username: hafsa
Password: password123
Confirm: password123
```

Expected:

```
Account created successfully.
```

Invalid Registration

Test:

```
Password: 123
```

Expected:

```
Password must contain at least 8 characters.
```

Test mismatched passwords:

```
Password: password123
Confirm: password456
```

Expected:

```
Passwords do not match.
```

Login

Use the registered credentials.

Expected:

```
Welcome, hafsa
You are successfully authenticated.
```

Invalid Login

Use an incorrect password.

Expected:

```
Invalid username or password.
```

Logout

Click **Logout**.

Expected:

```
Login screen appears.
```

14. Security Notes

This project demonstrates the core concepts of authentication, but it is still an educational desktop application.

Passwords should never be stored as plain text. Python's documentation specifically recommends salted, one-way password hashing rather than storing recoverable passwords.

For a production web application, additional protections would be required, including secure sessions, HTTPS, rate limiting, account recovery, email verification, CSRF protection, and appropriate password-hashing parameters.

15. Small Improvements

- Add email authentication
- Add password reset
- Add email verification
- Add account lockout/rate limiting
- Add user roles
- Add profile management
- Add session management
- Convert the desktop application to Flask or FastAPI
- Add secure web cookies
- Add two factor authentication

Visit haas.dev for more resources and guides.