PASSWORT-MANAGER

Nils, Joel, Matthis, Nico

INHALT

Projektziele

Datenbank

Speicherung der Daten in einer lokalen .DB-Datei

Sicherheit

Sichere Speicherung und Verwaltung der Daten

Benutzeroberfläche

Speichern, Bearbeiten und Löschen von Einträgen

Sicherheitsbewertung

Bewertung des eingegebenen Passworts

Generator

Generierung eines Sicheren Passworts

Wartbarkeit

Gute Wartbarkeit durch Kommentare und doc-strings

Rollenaufteilung

Zeitwächter

Joel

Überwachung der Zeitplanung

Moderator

Matthis

Koordination der Zusammenarbeit

Protokollant

Nico

Dokumentation der Sitzungen

Statusmanager

Nils

Fortschritts-überwachung

Technisce Aufgabenverteilung

GUI & Navigation

Nils

Verschlüsselung & Speicherung

Nico

Passwort-generierung

Joel

Passwort-Sicherheitscheck

Mathis

Projektmanagement

Meilenstein	Geplantes Datum	Erreicht am	Notizen
Projektstart	08.05.2025	08.05.2025	Kickoff-„Meeting“
Datenbankmodell fertig	15.05.2025	15.05.2025	Entwurf & SQL-Datei
GUI-Prototyp	22.05.2025	23.05.2025	Screenshot HERE
Funktionale Integration	22.05.2025	03.05.2025	Erste Testversion
Testphase	29.05.2025	03.06.2025	Protokoll Testfälle
Präsentation / Abgabe	11.06.2025	11.06.2025	PowerPoint-Datei, Lastenheft & Pflichtenheft

AUFBAU – DATENBANK

VALUES

ValueId	Name	Key

SAVES

SaveId	Name	Value

VORSTELLUNG

GUI & Navigation

Verschlüsselung & Speicherung

```python
class Encryption:
    def __init__(self):
        # Generate a 32-byte (256-bit) AES key from a password using PBKDF2
        # We use a password instead of a randomly generated key so that the key can be consistently recreated when needed.
        # the password is saved in ProgramSettings.CRYPT_KEY

        # Derive a secure key using PBKDF2 (Password-Based Key Derivation Function 2)
        # This allows for secure transformation of a password into a fixed-length cryptographic key.
        self.kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),              # Secure hashing algorithm used in key derivation
            length=32,                              # Length of the AES key: 32 bytes = 256 bits
            salt=b"bd5kqV<5/|N?hyY!AK(9[:-eF.3h",   # Fixed 28-string-salt-value ensures the same key is derived each time for this password
            iterations=100000,                      # Number of iterations increases computation cost to resist brute-force attacks
            backend=default_backend()               # Default backend provides cryptographic primitives
        )

        self.key = self.kdf.derive(ProgramSettings.CRYPT_KEY.encode('utf-8'))  # Derive the AES key from the password and salt using PBKDF2
```

```python
# Class to save all local settings for the program
class ProgramSettings:
    DEFAULT_CRYPT_KEY = "?l4![e_-_~:[C8oZO#Y3K,z53Mb$#2x6"
    DATABASE_PATH = "app_database.db"
    CRYPT_KEY = DEFAULT_CRYPT_KEY
```

Passwort-Generierung

```python
@staticmethod
def Generate(length):
    """
    Generate a secure random password containing letters, digits, and punctuation.
    Parameters:
    length (int): The desired length of the generated password.
    Returns:
    str: A randomly generated password string of the specified length.
    """
    # Define the character pool: uppercase, lowercase letters, digits and punctuation
    characters = string.ascii_letters + string.digits + string.punctuation

    # Use secrets.choice for cryptographically secure random selection
    password = ''.join(secrets.choice(characters) for _ in range(length))

    return password
```

Passwort-Sicherheitscheck

```python
# Check for minimum length
if len(password) < 8:
    messagebox.showwarning(self.DEFAULT_INFO_HEAD, f"{self.DEFAULT_INFO} Passwort zu kurz (mind. 8 Zeichen).")
    return False

# Check for at least one uppercase letter
if not re.search(r"[A-Z]", password):
    messagebox.showwarning(self.DEFAULT_INFO_HEAD, f"{self.DEFAULT_INFO} Mindestens ein Großbuchstabe fehlt.")
    return False

# Check for at least one lowercase letter
if not re.search(r"[a-z]", password):
    messagebox.showwarning(self.DEFAULT_INFO_HEAD, f"{self.DEFAULT_INFO} Mindestens ein Kleinbuchstabe fehlt.")
    return False

# Check for at least one digit
if not re.search(r"[0-9]", password):
    messagebox.showwarning(self.DEFAULT_INFO_HEAD, f"{self.DEFAULT_INFO} Mindestens eine Zahl fehlt.")
    return False

# Check for at least one special character
if not re.search(r"[!@#$%^&*()_+\-=\[\]{};:'\"\\|,.<>/?]", password):
    messagebox.showwarning(self.DEFAULT_INFO_HEAD, f"{self.DEFAULT_INFO} Mindestens ein Sonderzeichen fehlt.")
    return False

# All checks passed; password is strong
return True
```

Zukunftsideen

Backups

Backups der Datenbank erstellen und separat speichern

Zuletzt Benutze / Favoriten

Direkt die letzten Passwörter / Favorieten erkennen können

Standortfilterung

Passwörter nach Abrufstandort abfragen können

UNSER FAZIT

DANKE FÜR DIE AUFMERKSAMKEIT