

SimplePasswordManager – Dokumentation

1. Projektbeschreibung, Zielsetzung, Mockup

Dieses Projekt ist ein einfacher Passwortmanager mit Login-Authentifizierung über ein Masterpasswort und einem eingebauten Passwortgenerator. Der Benutzer kann damit seine Zugangsdaten sicher speichern.

1.1 Geplante Hauptfunktionen:

- Masterpasswortschutz beim Start
- Speicherung von Zugangsdaten (Passwörter in verschlüsselter Form)
- Steuerung über eine GUI
- Speichern, Anzeigen und Löschen von Zugangsdaten
- Konfigurierbarer Passwortgenerator (Länge, Zeichensatz)

Mockup 1 (Login):

Passwortmanager


Masterpasswort:

☐ Passwort Anzeigen

Mockup 2 (Hauptanwendung):

Passwortmanager

Passwortgenerator

Passwortlänge 

☐ Großbuchstaben
☐ Zahlen
☐ Sonderzeichen

Neue Zugangsdaten speichern

Benutzername

Passwort

☐ Passwort Anzeigen

Beschreibung

Gespeicherte Passwörter

Benutzername	Passwort	Beschreibung
user123	*****	Ebay
rudi.knaller@web.de	*****	Amazon
Nickname23	*****	Netflix

2. Verwendete Technologien

Programmiersprache:

C# (Version 10 / .NET 6)

Objektorientierte Programmiersprache mit starker Typisierung, geeignet für Desktop- und Webanwendungen.

Framework:

.NET (Core) / .NET 6

Modernes, plattformübergreifendes Framework von Microsoft für die Entwicklung leistungsfähiger Anwendungen.

GUI-Framework:

Windows Forms

Klassisches Framework zur Erstellung von Desktop-Oberflächen unter Windows.

ORM:

Entity Framework Core

Objekt-relationaler Mapper zur einfachen Datenbankbindung und -manipulation in C#.

Datenbank:

SQLite

Leichtgewichtiges, dateibasiertes Datenbanksystem ohne Server.

Testing-Framework:

xUnit

Weit verbreitetes Unit-Test-Framework für .NET-Anwendungen.

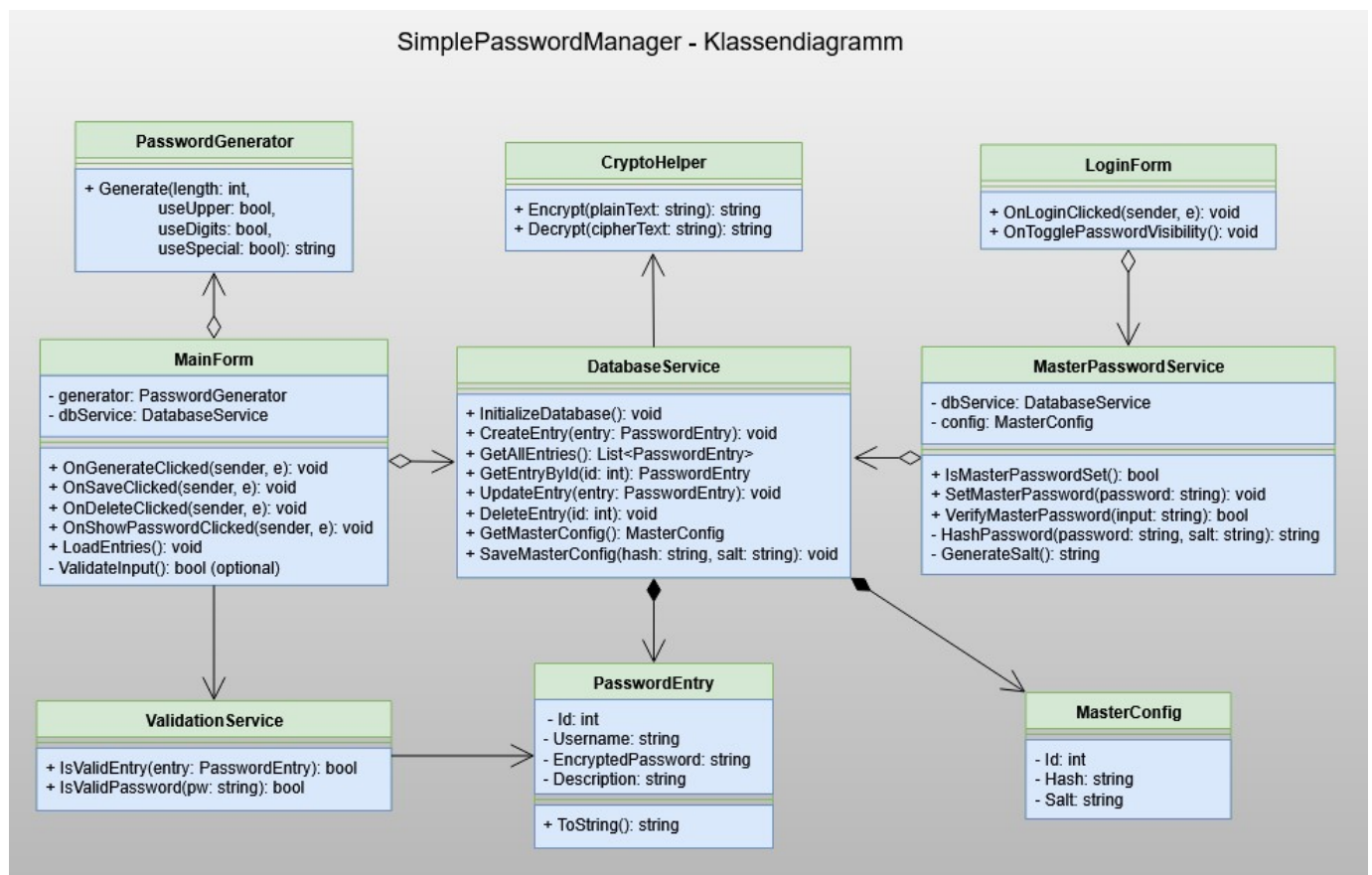
Kryptografie:

AES-256 (Advanced Encryption Standard), PBKDF2

Für sichere Verschlüsselung und Schlüsselableitung sensibler Daten (Passwörter).

2. Programmlogik und verwendete Klassen

Klassendiagramm:



2.1 Model-Klassen

Einfache Klassen, die Datenstrukturen abbilden und von Entity Framework Core zur Erstellung und Verwaltung der zugehörigen Datenbanktabellen verwendet werden.

1. **MasterConfig**
Speichert den Hash und Salt des Masterpassworts zur Authentifizierung.
2. **PasswordEntry**
Repräsentiert einen gespeicherten Zugang mit Benutzername, verschlüsseltem Passwort und Beschreibung.
3. **PasswordEntryDisplay**
Wird verwendet, um sensible Daten wie Benutzername und Passwort maskiert oder entschlüsselt im **DataGridView** der **MainForm** anzuzeigen – ohne direkt das verschlüsselte Original (**PasswordEntry**) zu verändern oder offenzulegen.

2.2 Persistenz (Datenbank) -Klassen

Klassen zur Datenpersistenz mit Entity Framework Core und SQLite. Der zentrale “DbContext” ist in Entity Framework Core die zentrale Klasse, die die Verbindung zur Datenbank verwaltet und den Zugriff ermöglicht.

4. **PasswordDbContext**
Entity Framework Core **DbContext**, verwaltet die Datenbankverbindung und enthält **DbSets** für Passwörter und Master-Config.

2.3 Service-Klassen (Logik & Sicherheit)

Kapseln zentrale Logik wie Verschlüsselung, Passwortgenerierung und Eingabevalidierung. Diese Klassen sind unabhängig von der Benutzeroberfläche und dienen als wiederverwendbare Hilfsdienste im Hintergrund.

5. **CryptoHelper**
Bietet Methoden zur AES-Verschlüsselung und -Entschlüsselung mit Salt, IV (Initialisierungsvektor) und PBKDF2-Schlüsselableitung.
6. **DatabaseService**
Kümmert sich um CRUD-Operationen für **PasswordEntry** und die Speicherung der **MasterConfig**.

7. **MasterPasswordService**

Verwaltet das Masterpassword: setzt, validiert und prüft es. Nutzt SHA256 + Salt zum Hashen.

8. **PasswordGenerator**

Generiert sichere Passwörter nach Vorgaben (Länge, Großbuchstaben, Ziffern, Sonderzeichen).

9. **ValidationService**

Validiert Eingaben für neue Passwort-Einträge (z. B. Mindestlänge, leere Felder, max. Länge Beschreibung).

2.4 Benutzeroberfläche (WinForms)

Die grafische Benutzeroberfläche ist auf Basis von Windows Forms (WinForms). Die Forms ermöglichen Bedienung und Anzeige über visuelle Steuerelemente wie Textfelder, Buttons und DataGridViews.

10. **LoginForm**

UI-Formular für die Anmeldung. Setzt anfangs das Masterpassword und prüft es bei zukünftigen Logins.

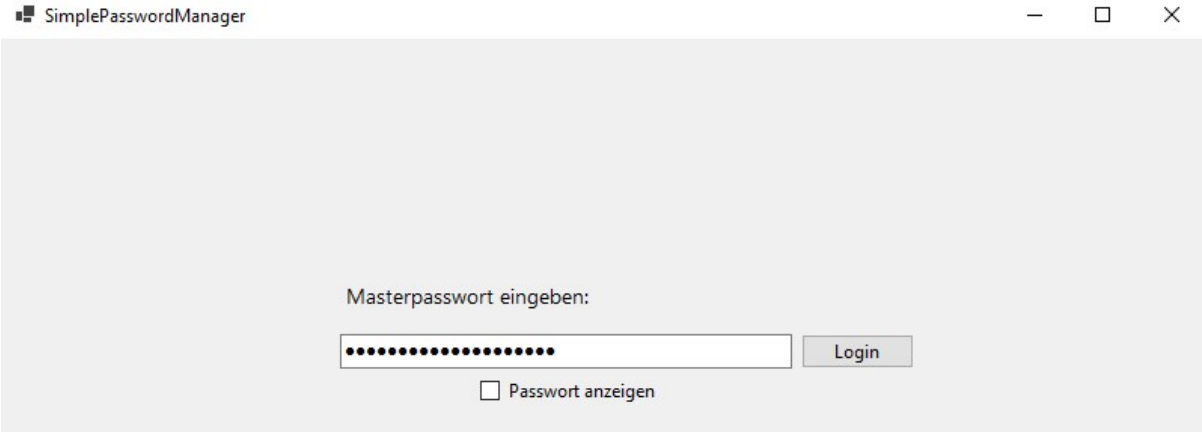
11. **MainForm**

Hauptoberfläche: Zeigt gespeicherte Passwörter an, erlaubt das Hinzufügen, Löschen, Entschlüsseln, Verbergen und Generieren von Passwörtern.

3. Grafische Benutzeroberfläche (GUI)

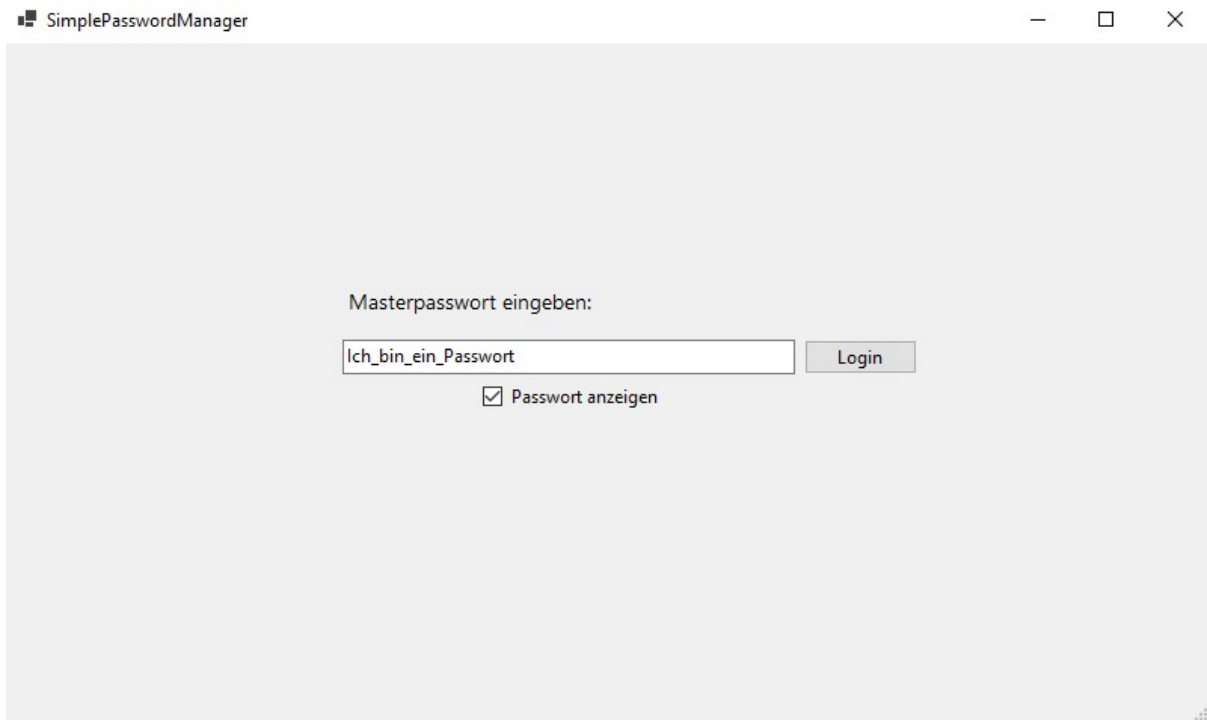
3.1 LoginForm:

Um die Anwendung zu starten muss der Benutzer sich zuerst einloggen. Bei erstmaliger Benutzung wird der Benutzer gebeten ein Masterpassword zu erstellen. Von diesem wird ein Hash in der Datenbank gespeichert, Zukünftig wird beim Login dieses Masterpassword verlangt. Eine Veränderung des Masterpasswords ist in dieser Version der Anwendung noch nicht implementiert.



The screenshot shows a Windows Forms application window titled "SimplePasswordManager". The window has a standard Windows title bar with minimize, maximize, and close buttons. The main content area is light gray. At the bottom, there is a label "Masterpassword eingeben:" above a text input field. The input field contains a series of black dots, indicating a password is entered. To the right of the input field is a "Login" button. Below the input field, there is a checkbox labeled "Passwort anzeigen".

Ein Klick auf **“Passwort anzeigen”** macht die Eingabe lesbar:



SimplePasswordManager

Masterpasswort eingeben:

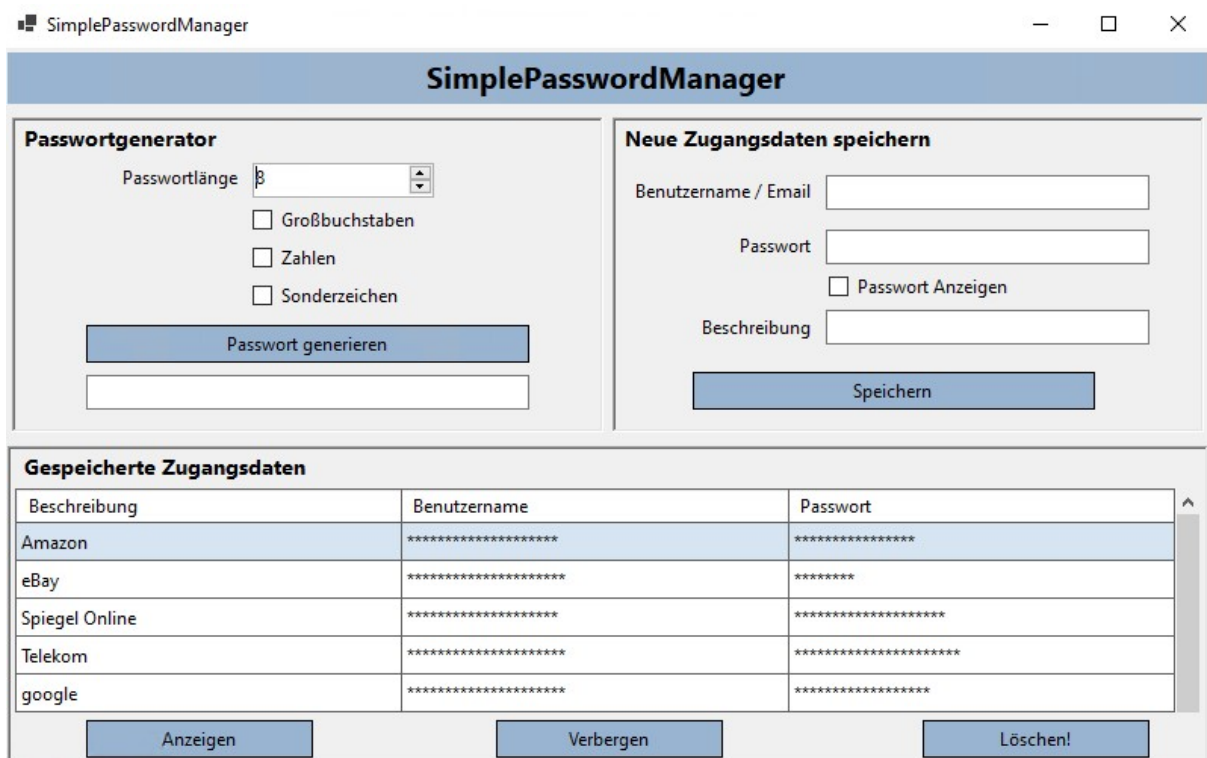
☒ Passwort anzeigen

Durch den Button **“Login”** wird bei korrektem Masterpasswort die **MainForm** gestartet.

3.2 MainForm:

MainForm ist in drei Bereiche unterteilt:

- Passwortgenerator
- Neue Zugangsdaten speichern
- Gespeicherte Zugangsdaten



SimplePasswordManager

Passwortgenerator

Passwortlänge

☐ Großbuchstaben

☐ Zahlen

☐ Sonderzeichen

Neue Zugangsdaten speichern

Benutzername / Email

Passwort

☐ Passwort Anzeigen

Beschreibung

Gespeicherte Zugangsdaten

Beschreibung	Benutzername	Passwort
Amazon	*****	*****
eBay	*****	*****
Spiegel Online	*****	*****
Telekom	*****	*****
google	*****	*****

Mit **“Anzeigen”** werden die ausgewählten Daten sichtbar, mit **“Verbergen”** wieder ausgeblendet:

SimplePasswordManager

SimplePasswordManager

Passwortgenerator
Passwortlänge: 8
☐ Großbuchstaben
☐ Zahlen
☐ Sonderzeichen
Passwort generieren

Neue Zugangsdaten speichern
Benutzername / Email:
Passwort:
☐ Passwort Anzeigen
Beschreibung:
Speichern

Gespeicherte Zugangsdaten

Beschreibung	Benutzername	Passwort
Amazon	*****	*****
eBay	*****	*****
Spiegel Online	*****	*****
Telekom	rudi.knaller@email.de	bte2dXOZtLVUvI0CYLmj5R
google	*****	*****

Anzeigen Verbergen Löschen!

Mit **“Löschen!”** können Einträge nach Bestätigung entfernt werden:

SimplePasswordManager

SimplePasswordManager

Passwortgenerator
Passwortlänge: 8
☐ Großbuchstaben
☐ Zahlen
☐ Sonderzeichen
Passwort g

Neue Zugangsdaten speichern
Benutzername / Email:
Passwort:
☐ Passwort Anzeigen
Beschreibung:
ern

Gespeicherte Zugangsdaten

Beschreibung	Benutzername	Passwort
Amazon	*****	*****
eBay	*****	*****
Spiegel Online	*****	*****
Telekom	*****	*****
google	*****	*****

Anzeigen Verbergen Löschen!

Löschen bestätigen
Sind Sie sicher, dass Sie diesen Eintrag "Telekom" löschen möchten?
Ja Nein

3.3 Passwortgenerator:

Passwörter können mit Auswahlmöglichkeiten zu Zeichensatz und Länge generiert werden, um sie dann als Zugangsdaten zu verwenden:

Passwortgenerator	Neue Zugangsdaten speichern
Passwortlänge: 10	Benutzername / Email: max.mustermann@muster.de
<input checked="" type="checkbox"/> Großbuchstaben	Passwort: fT&*_WPG^({
<input type="checkbox"/> Zahlen	<input checked="" type="checkbox"/> Passwort Anzeigen
<input checked="" type="checkbox"/> Sonderzeichen	Beschreibung: Beispiel 1
<button>Passwort generieren</button>	<button>Speichern</button>
fT&*_WPG^({	

Sportverein	*****	*****
Beispiel 1	max.mustermann@muster.de	fT&*_WPG^({
Beispiel 2	*****	*****

Passwörter können aber auch manuell eingegeben werden:

SimplePasswordManager		
Passwortgenerator	Neue Zugangsdaten speichern	
Passwortlänge: 10	Benutzername / Email: max.mustermann@muster.de	
<input checked="" type="checkbox"/> Großbuchstaben	Passwort: Ich_bin_ein_Passwort	
<input type="checkbox"/> Zahlen	<input checked="" type="checkbox"/> Passwort Anzeigen	
<input checked="" type="checkbox"/> Sonderzeichen	Beschreibung: Beispiel 2	
<button>Passwort generieren</button>	<button>Speichern</button>	
fT&*_WPG^({		

Beispiel 1	*****	*****
Beispiel 2	max.mustermann@muster.de	Ich_bin_ein_Passwort

4. Probleme und Lösungen

4.1 Login-Form blockiert exe-Datei

Problem:

Im ersten Entwurf wurde die App mit "*Application.Run(new LoginForm())*" gestartet. Dabei blieb LoginForm im Hintergrund aktiv, nachdem das Hauptfenster "MainForm" geöffnet wurde. Das führte dazu, dass beim Schließen der App der Prozess nicht korrekt beendet wurde und bei erneutem Build die .exe-Datei gesperrt war.

Lösung:

Die "LoginForm" wird nun als modales* Dialogfenster ("*ShowDialog()*") geöffnet. Nach erfolgreichem Login wird sie explizit geschlossen ("*this.Close()*"), und erst dann wird das Hauptfenster (MainForm) mit "*Application.Run()*" gestartet.

*Modal: Zustand oder Fenster, das den User zwingt, sich damit zu beschäftigen, bevor es weiter geht.

Ergebnis:

Die Anwendung wird vollständig beendet, sobald MainForm geschlossen wird – ohne hängenbleibende Prozesse.

5. Tests

Im Rahmen der Entwicklung wurden folgende Klassen mittels Unit-Tests überprüft:

- **CryptoHelper**
- Testklasse: **CryptoHelperTests**
- Überprüft die korrekte Funktionsweise der Verschlüsselung und Entschlüsselung sowie das Verhalten bei falschem Passwort.

```
0 Verweise
public class CryptoHelperTests
{
    [Fact]
    0 Verweise
    public void EncryptDecrypt_ReturnsOriginalText()
    {
        string toEncrypt = "123: Ich bin unverschlüsselt!!!";
        string encryptionPassword = "Verschlüsselungs_Passwort_123";
        string encrypted = CryptoHelper.Encrypt(toEncrypt, encryptionPassword);
        string decrypted = CryptoHelper.Decrypt(encrypted, encryptionPassword);
        Assert.Equal(toEncrypt, decrypted);
    }

    [Fact]
    0 Verweise
    public void Decrypt_WithWrongPassword_ThrowsException()
    {
        string toEncrypt = "Geheimer Text";
        string encryptionPassword = "Richtig123";
        string wrongEncryptionPassword = "Falsch456";

        string encrypted = CryptoHelper.Encrypt(toEncrypt, encryptionPassword);

        Assert.Throws<CryptographicException>(() => CryptoHelper.Decrypt(encrypted, wrongEncryptionPassword));
    }
}
```

- **PasswordGenerator**
- Testklasse: **PasswordGeneratorTests**
- Testet die Erzeugung von Passwörtern auf Länge und Zeichentypen (Klein-, Großbuchstaben, Zahlen, Sonderzeichen) und Fehlerbehandlung bei zu kurzer Länge.

```

0 Verweise
public class PasswordGeneratorTests
{
    private PasswordGenerator _generator = new PasswordGenerator();

    [Fact]
    0 Verweise
    public void Generate_ReturnsPasswordWithCorrectLength()
    {
        string password = _generator.Generate(12, true, true, true);
        Assert.Equal(12, password.Length); // Richtige Länge?
    }

    [Fact]
    0 Verweise
    public void Generate_WithOnlyLowercase_ReturnsCorrectLengthAndOnlyLowercase()
    {
        string result = _generator.Generate(length: 12, useUpper: false, useDigits: false, useSpecial: false);

        Assert.Equal(12, result.Length); // Richtige Länge?
        Assert.All(result, c => Assert.True(char.IsLower(c))); // Alle Elemente Kleinbuchstaben?
    }

    [Fact]
    0 Verweise
    public void Generate_WithUppercase_ContainsAtLeastOneUpper()
    {
        string result = _generator.Generate(length: 12, useUpper: true, useDigits: false, useSpecial: false);

        Assert.Contains(result, c => char.IsUpper(c)); // Mindestens ein Großbuchstabe?
    }

    [Fact]
    0 Verweise
    public void Generate_WithDigitsAndSpecial_ContainsAtLeastOneDigitAndSpecial()
    {
        string result = _generator.Generate(12, useUpper: false, useDigits: true, useSpecial: true);

        Assert.Contains(result, c => char.IsDigit(c)); // Mindestens eine Zahl?
        Assert.Contains(result, c => "!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~\".Contains(c));
    }
}

```

```

[Fact]
0 Verweise
public void Generate_TooShort_ThrowsArgumentException()
{
    // 3 Zeichengruppen aktiviert, aber Länge 2 ausgewählt.
    Assert.Throws<ArgumentException>(() => _generator.Generate(2, true, true, true));
}

[Theory]
[InlineData(8, true, true, false)]
[InlineData(16, true, true, true)]
[InlineData(20, false, true, true)]
0 Verweise
public void Generate_ReturnsStringWithCorrectLength(int length, bool upper, bool digits, bool special)
{
    string result = _generator.Generate(length, upper, digits, special);
    Assert.Equal(length, result.Length); // Stimmen eingegebene und tatsächliche Länge überein?
}
}

```

- **ValidationService**
- Testklasse: **ValidationServiceTests**
- Validiert Benutzereingaben auf Vollständigkeit, Mindest- und Maximallänge sowie korrekte Fehlerausgaben bei ungültigen Eingaben.

```

0 Verweise
public class ValidationServiceTests
{
    [Theory]
    [InlineData("", "abc123", "Notiz", "Benutzername leer.")]
    [InlineData("user", "", "Notiz", "Passwort leer.")]
    [InlineData("user", "abc123", "", "Beschreibung leer.")]
    [InlineData("user", "abc", "Notiz", "Passwort zu kurz (mind. 6 Zeichen).")]
    [InlineData("user", "abc123", "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
        "Beschreibung zu lang.(max. 40 Zeichen)")]
    0 Verweise
    public void IsValidEntry_InvalidInput_ReturnsFalseAndErrorMessage(
        string username, string password, string description, string expectedError)
    {
        // Act
        bool result = ValidationService.IsValidEntry(username, password, description, out string error);

        // Assert
        Assert.False(result);
        Assert.Equal(expectedError, error);
    }

    [Fact]
    0 Verweise
    public void IsValidEntry_ValidInput_ReturnsTrue()
    {
        bool result = ValidationService.IsValidEntry("user", "securePass123", "Beschreibung", out string error);

        Assert.True(result);
        Assert.Equal("", error);
    }
}

```

Testergebnisse (alle bestanden):

The screenshot shows the Test Explorer window with the following details:

- Test Explorer Header:** Includes icons for running tests (play, stop, skip) and a summary bar showing 16 passed tests (green checkmarks), 16 failed tests (red X's), and 0 skipped tests.
- Bereit (Ready):** A section header indicating the state of the tests.
- Test Results Table:**

Test	Dauer (Duration)	Merkmale (Features)	Fehlermeldung (Error Message)
✓ PasswordManagerApp.Tests (16)	477 ms		
✓ PasswordManagerApp.Tests (16)	477 ms		
✓ CryptoHelperTests (2)	273 ms		
✓ Decrypt_WithWrongPassword_ThrowsException	241 ms		
✓ EncryptDecrypt_ReturnsOriginalText	32 ms		
✓ PasswordGeneratorTests (8)	204 ms		
✓ Generate_ReturnsPasswordWithCorrectLength	< 1 ms		
✓ Generate_ReturnsStringWithCorrectLength (3)	< 1 ms		
✓ Generate_ReturnsStringWithCorrectLength(length: 16, upper: True, digits: True, special: True)	< 1 ms		
✓ Generate_ReturnsStringWithCorrectLength(length: 20, upper: False, digits: True, special: True)	< 1 ms		
✓ Generate_ReturnsStringWithCorrectLength(length: 8, upper: True, digits: True, special: False)	< 1 ms		
✓ Generate_TooShort_ThrowsArgumentException	200 ms		
✓ Generate_WithDigitsAndSpecial_ContainsAtLeastOneDigitAndSpecial	< 1 ms		
✓ Generate_WithOnlyLowercase_ReturnsCorrectLengthAndOnlyLowercase	1 ms		
✓ Generate_WithUppercase_ContainsAtLeastOneUpper	3 ms		
✓ ValidationServiceTests (6)	< 1 ms		
✓ IsValidEntry_InvalidInput_ReturnsFalseAndErrorMessage (5)	< 1 ms		
✓ IsValidEntry_InvalidInput_ReturnsFalseAndErrorMessage(username: "", password: "abc123", descri...	< 1 ms		
✓ IsValidEntry_InvalidInput_ReturnsFalseAndErrorMessage(username: "user", password: "", descriptio...	< 1 ms		
✓ IsValidEntry_InvalidInput_ReturnsFalseAndErrorMessage(username: "user", password: "abc", descri...	< 1 ms		
✓ IsValidEntry_InvalidInput_ReturnsFalseAndErrorMessage(username: "user", password: "abc123", de...	< 1 ms		
✓ IsValidEntry_InvalidInput_ReturnsFalseAndErrorMessage(username: "user", password: "abc123", de...	< 1 ms		
✓ IsValidEntry_ValidInput_ReturnsTrue	< 1 ms		

6. Mögliche Erweiterungen

Für zukünftige Weiterentwicklungen der Anwendung bieten sich unter anderem folgende Funktionen an:

- Funktion zum Ändern des Masterpassworts: Ermöglicht dem Benutzer, sein Masterpasswort sicher zu aktualisieren.
- Unterstützung für mehrere Benutzerkonten: Erlaubt die Verwaltung von getrennten Passwortdatenbanken für verschiedene Nutzer.
- Anzeige der Passwortstärke: Gibt visuelles Feedback zur Sicherheit eingegebener Passwörter (z. B. mit Farbcodes oder Balken).
- Bearbeiten bestehender Einträge
Hinzufügen einer Funktion, um bereits gespeicherte Passworteinträge direkt zu ändern.