

Programmentwurf PWManager

Name: Hagl, Alexander & Hübner, Jasmin
Matrikelnummer: 7371122, 8566500

Abgabedatum: 07.06.2024

Allgemeine Anmerkungen:

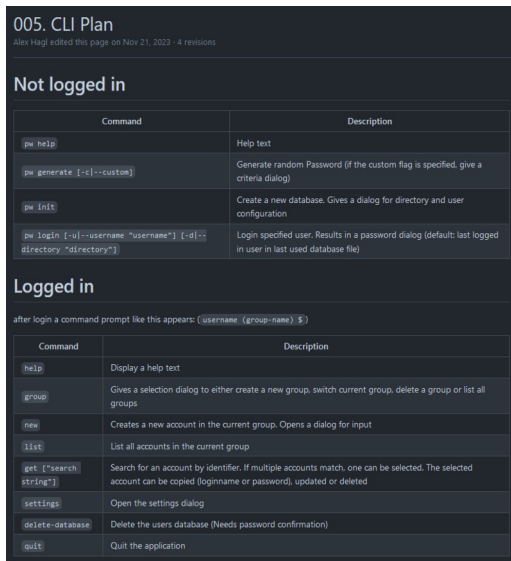
- *es darf nicht auf andere Kapitel als Leistungsnachweis verwiesen werden (z.B. in der Form “XY wurde schon in Kapitel 2 behandelt, daher hier keine Ausführung”)*
- *alles muss in UTF-8 codiert sein (Text und Code)*
- *sollten mündliche Aussagen den schriftlichen Aufgaben widersprechen, gelten die schriftlichen Aufgaben (ggf. an Anpassung der schriftlichen Aufgaben erinnern!)*
- *alles muss ins Repository (Code, Ausarbeitung und alles was damit zusammenhängt)*
- *die Beispiele sollten wenn möglich vom aktuellen Stand genommen werden*
 - *finden sich dort keine entsprechenden Beispiele, dürfen auch ältere Commits unter Verweis auf den Commit verwendet werden*
 - *Ausnahme: beim Kapitel “Refactoring” darf von vorne herein aus allen Ständen frei gewählt werden (mit Verweis auf den entsprechenden Commit)*
- *falls verlangte Negativ-Beispiele nicht vorhanden sind, müssen entsprechend mehr Positiv-Beispiele gebracht werden*
 - *Achtung: werden im Code entsprechende Negativ-Beispiele gefunden, gibt es keine Punkte für die zusätzlichen Positiv-Beispiele*
 - *Beispiele*
 - *“Nennen Sie jeweils eine Klasse, die das SRP einhält bzw. verletzt.”*
 - *Antwort: Es gibt keine Klasse, die SRP verletzt, daher hier 2 Klassen, die SRP einhalten: [Klasse 1], [Klasse 2]*
 - *Bewertung: falls im Code tatsächlich keine Klasse das SRP verletzt: volle Punktzahl ODER falls im Code mind. eine Klasse SRP verletzt: halbe Punktzahl*
- *verlangte Positiv-Beispiele müssen gebracht werden*
- *Code-Beispiel = Code in das Dokument kopieren*

Kapitel 1: Einführung

Übersicht über die Applikation

[Was macht die Applikation? Wie funktioniert sie? Welches Problem löst sie/welchen Zweck hat sie?]

Die Applikation ist ein Password Manager, den man in seiner CLI bedienen kann. Es hat folgende Commands (<https://github.com/CUMGroup/PWManager/wiki/005.-CLI-Plan>):



005. CLI Plan
Alex Hagl edited this page on Nov 21, 2023 · 4 revisions

Not logged in

Command	Description
<code>pw help</code>	Help text
<code>pw generate [-c --custom]</code>	Generate random Password (if the custom flag is specified, give a criteria dialog)
<code>pw init</code>	Create a new database. Gives a dialog for directory and user configuration
<code>pw login [-u --username "username"] [-d --directory "directory"]</code>	Login specified user. Results in a password dialog (default: last logged in user in last used database file)

Logged in
after login a command prompt like this appears: `(username (group-name)) $`

Command	Description
<code>help</code>	Display a help text
<code>group</code>	Gives a selection dialog to either create a new group, switch current group, delete a group or list all groups
<code>new</code>	Creates a new account in the current group. Opens a dialog for input
<code>list</code>	List all accounts in the current group
<code>get ["search string"]</code>	Search for an account by identifier. If multiple accounts match, one can be selected. The selected account can be copied (loginname or password), updated or deleted
<code>settings</code>	Open the settings dialog
<code>delete-database</code>	Delete the users database (Needs password confirmation)
<code>quit</code>	Quit the application

Man kann an einem beliebigen Pfad eine Passwort-Datenbank initialisieren, dabei wird nach dem Namen des Benutzers und einem Master Passwort gefragt.

Wird der login Command ausgeführt, wird ohne angegebenen Argumente die zuletzt benutzte Datenbank geöffnet. Falls bisher noch keine Datenbank geöffnet wurde, wird nach dem Pfad, der geöffnet werden soll gefragt.

Im Programm selbst kann man Accounts in Gruppen verwalten, zwischen Gruppen wechseln und neue Accounts erstellen, mit entweder eigenem Passwort oder generierten Passwort.

Die Daten werden verschlüsselt auf der Festplatte in einer SQLite Datei gespeichert.

Für mehr Informationen:

- README (<https://github.com/CUMGroup/PWManager>)
- (außerhalb der Session) ``$ PWManager.exe help``
- (innerhalb der Session) ``$ help``

Wie startet man die Applikation?

[Wie startet man die Applikation? Welche Voraussetzungen werden benötigt? Schritt-für-Schritt-Anleitung]

1. Schritt: Requirements

Zum starten der Applikation werden folgende Voraussetzungen benötigt:

- .NET 7 SDK (<https://dotnet.microsoft.com/en-us/download/dotnet/7.0>)
- Das geklonte Repo (<https://github.com/CUMGroup/PWManager.git>)

Getestet wurde die Applikation auf Windows 10.

2. Schritt: Kompilieren

Um die Applikation zu kompilieren, liegt ein Script im root Ordner des Repos `build.bat`.

Diese führt 2 Operationen im PWManager.CLI Ordner aus:

1. `dotnet restore` → Downloadet und kompiliert alle Dependencies
2. `dotnet build --configuration Release` → Baut die CLI Applikation in Release configuration

→ Die erstellte `PWManager.CLI.exe` kann im Ordner `PWManager.CLI\bin\Release\net7.0` gefunden werden.

3. Schritt: Ausführen

Führe die PWManager.CLI.exe in der Konsole aus.

Die Anwendung wurde im `Windows Terminal` mit einer Powershell 7 getestet.

Starte `PWManager.CLI.exe help` für mehr Informationen.

Wie testet man die Applikation?

[Wie testet man die Applikation? Welche Voraussetzungen werden benötigt? Schritt-für-Schritt-Anleitung]

Um die Anwendung zu testen werden die Requirements aus dem vorherigen Kapitel benötigt:

- .NET 7 SDK (<https://dotnet.microsoft.com/en-us/download/dotnet/7.0>)
- Das geklonte Repo (<https://github.com/CUMGroup/PWManager.git>)

Die UnitTests liegen in PWManager.UnitTests. Um diese auszuführen liegt ein Script im root Ordner des Repos `test.bat`. Diese führt folgende Schritte im PWManager.UnitTests Ordner aus aus:

1. `dotnet restore` → Downloadet und kompiliert alle Dependencies
2. `dotnet tool install -g dotnet-reportgenerator-globaltool` → Installiert den dotnet report generator
3. `dotnet test --logger "console;verbosity=detailed" --collect:"XPlat Code Coverage"`
→ Ausführung der Unit Tests
4. `reportgenerator "-reports:TestResults/*/coverage*" "-targetdir:coverage" "-reporttypes:Html;TextSummary"` → Erstellen des coverage reports
5. `type coverage\Summary.txt` → Ausgeben der coverages in der Konsole
6. `start coverage\index.html` → Öffnen des ausführlichen Reports im Webbrowser

Kapitel 2: Clean Architecture

Was ist Clean Architecture?

[allgemeine Beschreibung der Clean Architecture in eigenen Worten]#

Die Aufteilung einer Software in auf sich aufbauenden Zwiebelschichten. Je weiter innen die Schicht liegt, desto stabiler muss diese sein. Generell sollten sich die Schichten weiter innen seltener ändern als Schichten, die weiter außen liegen.

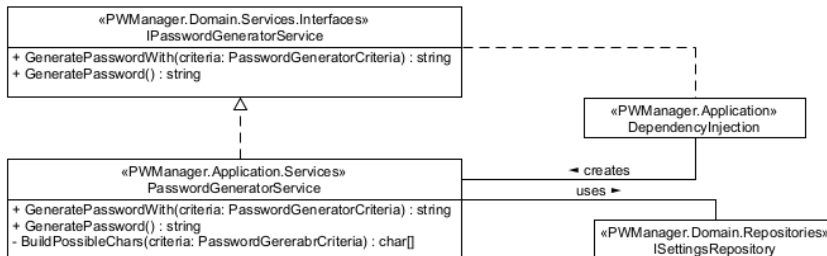
Häufigste Schichten:

1. Domain Schicht – Beinhaltet die Business Logik, die generell für die Problemdomäne gilt.
2. Applikations-Schicht – Beinhaltet Logik für die spezielle Applikation und hängt von der Domain Schicht ab.
3. Adapter Schicht – Gilt als Kommunikator zwischen den äußeren Schichten und den inneren Schichten.
4. Plugin Schicht – Ändert sich am häufigsten und hängt von den inneren Schichten ab. Hier sind die technischen Implementationsdetails.

Analyse der Dependency Rule

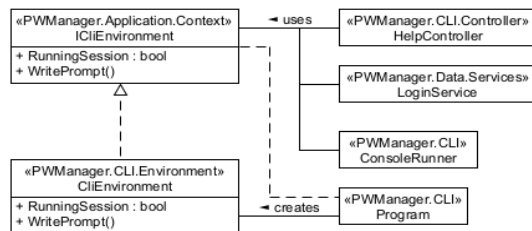
[(1 Klasse, die die Dependency Rule einhält und eine Klasse, die die Dependency Rule verletzt); jeweils UML der Klasse und Analyse der Abhängigkeiten in beide Richtungen (d.h., von wem hängt die Klasse ab und wer hängt von der Klasse ab) in Bezug auf die Dependency Rule]

Positiv-Beispiel: Dependency Rule



Der Service PasswordGeneratorService implementiert das Interface IPasswGeneratorService und hängt nur von dem ISettingsRepository Interface ab, welche beide im Domain-Layer liegen. Keine Klasse hängt von dem Service direkt ab, sondern von dem implementierten Interface. Die Klasse wird per Dependency Injection bereitgestellt.

Negativ-Beispiel: Dependency Rule

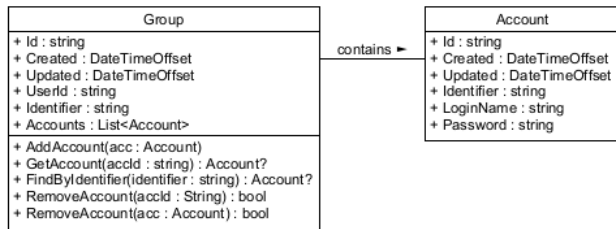


Das Interface ICliEnvironment hat zwar keine Code-Abhängigkeiten selbst, die die Dependency Rule brechen. Allerdings impliziert dieses Interface, welche Art von UI in der äußeren Schicht benutzt wird. Wird die UI mit einer Desktop Anwendung getauscht, muss die innere Schicht angepasst werden.

Analyse der Schichten

[jeweils 1 Klasse zu 2 unterschiedlichen Schichten der Clean-Architecture: jeweils UML der Klasse (ggf. auch zusammenspielenden Klassen), Beschreibung der Aufgabe, Einordnung mit Begründung in die Clean-Architecture]

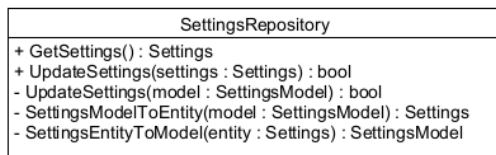
Schicht: Domain (PWManager.Domain)



Group ist die Entität des PWManagers, das eine Kollektion von verschiedenen Passwort-Einträgen (Accounts) darstellt. Darüber hinaus hat es die Aufgabe der Aggregat-Root-Entität im Aggregat Group – Account.

Diese Klasse ist in der Domain Schicht positioniert, da es als Entität ein Grundstein der Business Logik ist.

Schicht: Plugin Schicht (PWManager.Data)



Das SettingsRepository ist die Implementierung des ISettingsRepository Interfaces. Es ist dafür zuständig die Benutzer Einstellungen zu lesen und zu schreiben/speichern.

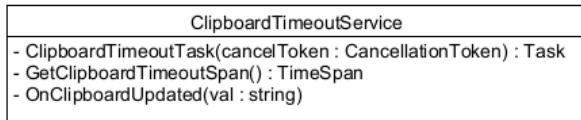
Da es von der speziellen Implementierung der Datenpersistenz abhängt – also wissen muss wie genau es die Daten speichert – liegt es ganz außen in der Plugin Schicht.

Kapitel 3: SOLID

Analyse Single-Responsibility-Principle (SRP)

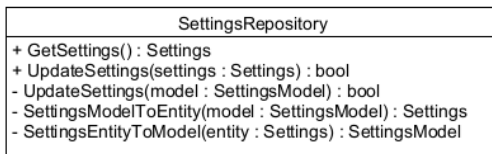
[jeweils eine Klasse als positives und negatives Beispiel für SRP; jeweils UML der Klasse und Beschreibung der Aufgabe bzw. der Aufgaben und möglicher Lösungsweg des Negativ-Beispiels (inkl. UML)]

Positiv-Beispiel



Der ClipboardTimeoutService ist ein Subscriber der Clipboard API und ist dafür zuständig bei einer Notification einen Task zu starten, der nach einer definierten Zeit (aus den Benutzer Einstellungen) das Clipboard wieder leert. Da das die einzige Aufgabe der Klasse ist, erfüllt diese das SRP.

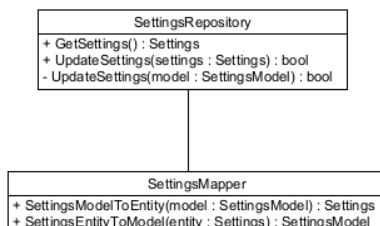
Negativ-Beispiel



Das SettingsRepository hat 2 Aufgaben. Sie ist dafür zuständig die User Settings zu schreiben und zu lesen. Als zweite Aufgabe hat die Klasse das Mappen zwischen Entitäten und Datenmodellen.

Da diese Klasse zwei disjunkte Aufgaben erfüllt, hält sie nicht das SRP.

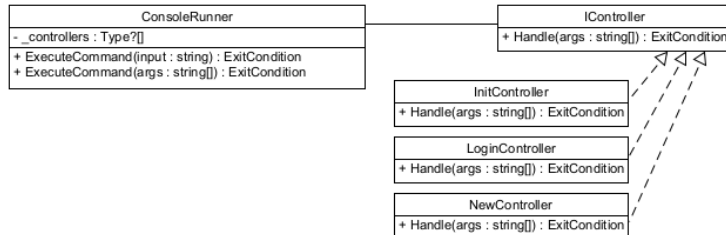
Um das SRP einzuhalten, müsste man das SettingsRepo aufspalten in ein pures Repository und einen dedizierten Mapper zwischen den Objekten (siehe UML).



Analyse Open-Closed-Principle (OCP)

[jeweils eine Klasse als positives und negatives Beispiel für OCP; jeweils UML der Klasse und Analyse mit Begründung, warum das OCP erfüllt/nicht erfüllt wurde – falls erfüllt: warum hier sinnvoll/welches Problem gab es? Falls nicht erfüllt: wie könnte man es lösen (inkl. UML)?]

Positiv-Beispiel

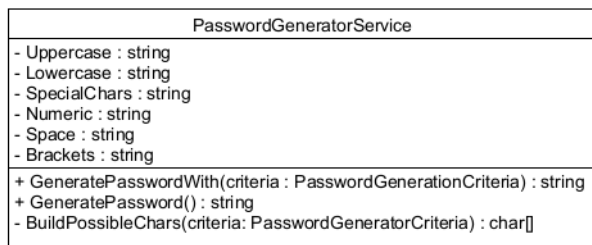


Der **ConsoleRunner** verteilt Anfragen dynamisch an die dedizierten Controller. Um einen neuen Controller zu erstellen, muss nur das **IController** Interface implementiert werden.

Der **ConsoleRunner** ist also offen für Erweiterung und geschlossen für Modifikation und erfüllt daher das OCP.

Es wurde hier speziell darauf geachtet, dass neue Controller einfach hinzugefügt werden können.

Negativ-Beispiel



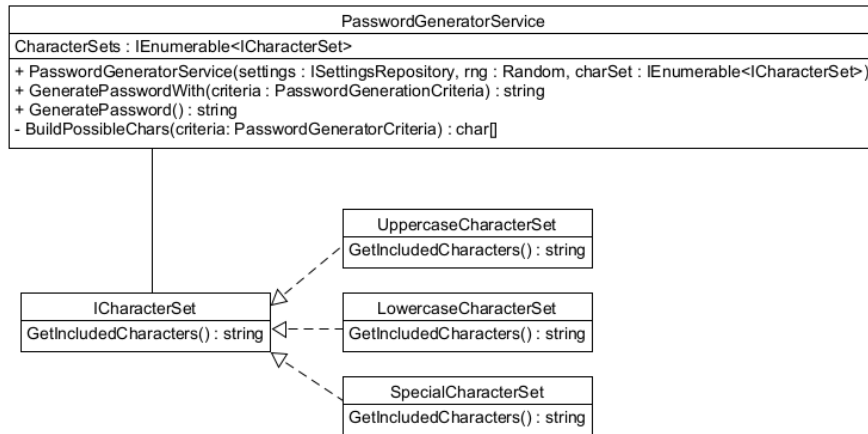
Der Passwort Generator benutzt ein vordefiniertes Character-Set (Hier Uppercase, Lowercase,...).

In der Methode **BuildPossibleChars** wird die übergebene criteria betrachtet und die definierten Character-Sets zusammengebaut, je nachdem was aktiviert ist in der criteria.

Will man nun aber neben den vorhandenen Zeichen auch Emojies als Character-Set hinzufügen muss ich das Character-Set als Attribut hinzufügen und die **BuildPossibleChars** Methode erweitern.

→ Bruch von Open/Closed

Verbesserung:



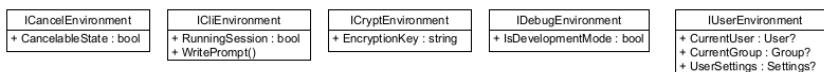
Um es zu verbessern, kann man die Character-Sets in ein Interface auslagern und die möglichen Character-Sets im Konstruktor übergeben. In der BuildPossibleChars Methode wird dann nur über die Liste iteriert und per GetIncludedCharacters() zusammengebaut.

Analyse Liskov-Substitution- (LSP), Interface-Segregation- (ISP), Dependency-Inversion-Principle (DIP)

[jeweils eine Klasse als positives und negatives Beispiel für entweder LSP oder ISP oder DIP); jeweils UML der Klasse und Begründung, warum man hier das Prinzip erfüllt/nicht erfüllt wird]

[Anm.: es darf nur ein Prinzip ausgewählt werden; es darf NICHT z.B. ein positives Beispiel für LSP und ein negatives Beispiel für ISP genommen werden]

Positiv-Beispiel (ISP)



Diese Interfaces wurden aufgeteilt in minimale Interfaces, dass jede Klasse, die Informationen über das Environment benötigt, das benötigte Interface injizieren kann.

Braucht ein Service den aktuellen User aus IUserEnvironment, darf dieser nicht auf den EncryptionKey zugreifen können usw.

Negativ-Beispiel (ISP)

ACHTUNG ALTER STAND (COMMIT ID 98fcddb903dcd140b499ac64ec35c9958cf2aaf0):

<https://github.com/CUMGroup/PWManager/blob/98fcddb903dcd140b499ac64ec35c9958cf2aaf0/PWManager.Application/Context/IApplicationEnvironment.cs>

IApplicationEnvironment
IsDevelopmentMode : bool
RunningSession : bool
CurrentUser : User?
CurrentGroup : Group?
EncryptionKey : string?

Das Interface IApplicationEnvironment erfüllt nicht das ISP, da z.B. der CryptService auf den EncryptionKey zugreifen muss, allerdings über das Interface auch Informationen über den User oder die Session bekommt.

Um ISP einzuhalten wurde das Interface aufgespalten zum Stand des positiv Beispiels.

Kapitel 4: Weitere Prinzipien

Analyse GRASP: Geringe Kopplung

[jeweils eine bis jetzt noch nicht behandelte Klasse als positives und negatives Beispiel geringer Kopplung; jeweils UML Diagramm mit zusammenspielenden Klassen, Aufgabenbeschreibung und Begründung für die Umsetzung der geringen Kopplung bzw. Beschreibung, wie die Kopplung aufgelöst werden kann]

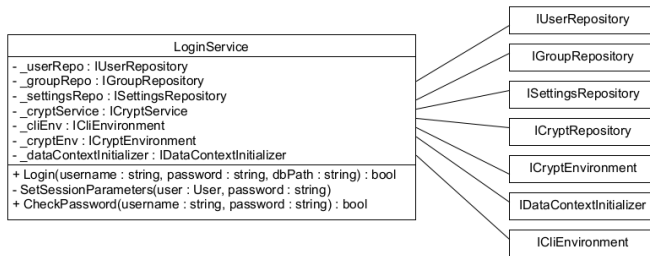
Positiv-Beispiel

CommandParser
+ ParseCommandWithArguments(cmd : string) : IEnumerable<string>
- LookaheadTo(cmd : string, c : char, cursorStart : int) : int
+ ParseCommand(cmd : string) : AvailableCommands

Der CommandParser ist dafür zuständig Input zu nehmen und daraus den richtigen Command zu parsen oder Argumente zu separieren.

Die Klasse hat eine niedrige Kopplung. Die einzige Klasse (bis auf .NET Klassen), die sie referenziert ist ein Enum, welcher keine Logik implementiert und sehr stabil ist.

Negativ-Beispiel

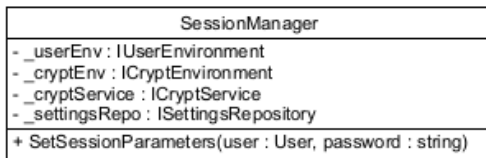
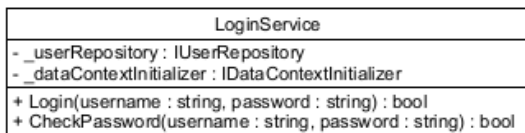


Der LoginService ist dafür Zuständig den User einzuloggen (dabei das Passwort zu überprüfen) und die Session Parameter zu setzen.

Zwar hat der LoginService nur Abhängigkeiten zu Interfaces, allerdings bedingt die Masse der Abhängigkeiten eine durchaus erhöhte Kopplung.

Die Abhängigkeit zum ICLiEnvironment ist die schlimmste. Diese koppelt den Service stark zu einem „CLI“ Environment. Wird ein anderes Frontend verwendet, muss hier auch angefasst werden und sorgt für eine geringere Wiederverwendbarkeit.

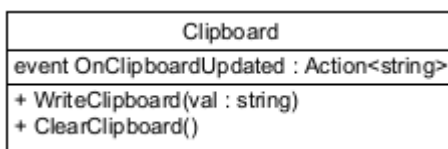
Die Kopplung könnte aufgelöst werden, indem der LoginService in mehrere Services aufgespalten wird. Nicht jede Methode benötigt alle Abhängigkeiten. So könnte es verringert werden.



Das Sessionmanagement ist nun ausgelagert und soll vom Caller übernommen werden. Die Dependency ICLiEnvironment ist komplett weggefallen. Ist der Caller ein CliController, soll dieser selbst das ICLiEnvironment benutzen um die CLI-Session endgültig zu starten. So ist die ICLiEnvironment-Abhängigkeit nicht mehr so schlimm wie im LoginService, da sie dann sowieso nur noch innerhalb des CLI namespaces benutzt wird.

Analyse GRASP: Hohe Kohäsion

[eine Klasse als positives Beispiel hoher Kohäsion; UML Diagramm und Begründung, warum die Kohäsion hoch ist]



Die Clipboard Klasse ist eine Abstraktion für die Interaktion mit der System Zwischenablage. Sie bietet 2 Funktionen zum Schreiben und ein Event, zu dem sich andere Klassen subscriben können um updates des Clipboards zu erhalten.

Die Kohäsion dieser Klasse ist sehr hoch, da alle bereitgestellten Funktionen stark verwandt sind und eine klar definierte Aufgabe im gleichen Kontext erfüllen.

Don't Repeat Yourself (DRY)

[ein Commit angeben, bei dem duplizierter Code/duplizierte Logik aufgelöst wurde; Code-Beispiele (vorher/nachher); begründen und Auswirkung beschreiben]

Commit:

<https://github.com/CUMGroup/PWManager/commit/a96ce83e55e9d5763fc37074a1e716c41d1ac301>

[CommitID: a96ce83e55e9d5763fc37074a1e716c41d1ac301]

Alter Code (vor a96ce83e55e9d5763fc37074a1e716c41d1ac301)

```
[Fact]
public void Login_Should_SetEnviroment() {
    _wrapper.DatabaseExists(Arg.Any<string>()).Returns(true);
    var user = new User(Guid.NewGuid().ToString(), DateTimeOffset.Now, DateTimeOffset.Now, "TestUserName");
    _userRepo.CheckPasswordAttempt(Arg.Any<string>(), Arg.Any<string>()).Returns(user);
    var group = new Group("TestGroup", user.Id);
    _settingsRepository.GetSettings().Returns(
        new Settings(user.Id, null, null, new MainGroupSetting(group.Identifier))
    );
    _groupRepo.GetGroup(Arg.Any<string>()).Returns(group);

    _sut = new LoginService(_wrapper, _userRepo, _groupRepo, _cryptService, _settingsRepository, _cliEnv, _userEnv,
        cryptEnv);
    _sut.Login("TestUserName", "WhatAPasswort", ".");

    Assert.Equal(user, _userEnv.CurrentUser);
    Assert.NotNull(_cryptEnv.EncryptionKey);
    Assert.NotNull(_userEnv.CurrentGroup);
    Assert.True(_cliEnv.RunningSession);
}

[Fact]
public void Login_ShouldNot_IfDatabaseDoesntExists() {
    _wrapper.DatabaseExists(Arg.Any<string>()).Returns(false);
    var user = new User(Guid.NewGuid().ToString(), DateTimeOffset.Now, DateTimeOffset.Now, "TestUserName");
    _userRepo.CheckPasswordAttempt(Arg.Any<string>(), Arg.Any<string>()).Returns(user);

    _sut = new LoginService(_wrapper, _userRepo, _groupRepo, _cryptService, _settingsRepository, _cliEnv, _userEnv,
        cryptEnv);

    var ex = Assert.Throws<UserFeedbackException>(() => _sut.Login("TestUserName", "WhatAPasswort", "."));
    Assert.Equal("Database not found.", ex.Message);
}
```

Neuer Code (nach a96ce83e55e9d5763fc37074a1e716c41d1ac301)

```
[Fact]
public void Login_Should_SetEnvironment() {
    SetupDatabaseExists(true);
    var user = SetupPasswordCheckReturnsUser();
    var group = SetupGroupRepoReturnsGroup(user.Id);
    SetupSettingsRepo(user.Id, group.Identifier);

    _sut = new LoginService(_userRepo, _groupRepo, _cryptService, _settingsRepository, _cliEnv, _userEnv, _cryptEnv,
_dataContextInitializer);

    _sut.Login("TestUserName", "WhatAPasswort", ".");

    Assert.Equal(user, _userEnv.CurrentUser);
    Assert.NotNull(_cryptEnv.EncryptionKey);
    Assert.NotNull(_userEnv.CurrentGroup);
    Assert.True(_cliEnv.RunningSession);
}

[Fact]
public void Login_ShouldNot_IfDatabaseDoesntExists() {
    SetupDatabaseExists(false);
    SetupPasswordCheckReturnsUser();

    _sut = new LoginService(_userRepo, _groupRepo, _cryptService, _settingsRepository, _cliEnv, _userEnv, _cryptEnv,
_dataContextInitializer);

    var ex = Assert.Throws<UserFeedbackException>(() => _sut.Login("TestUserName", "WhatAPasswort", "."));

    Assert.Equal("Database not found.", ex.Message);
}

private void SetupDatabaseExists(bool exists) {
    _dataContextInitializer.DatabaseExists(Arg.Any<string>()).Returns(exists);
}

private User SetupPasswordCheckReturnsUser() {
    var user = new User(Guid.NewGuid().ToString(), DateTimeOffset.Now, DateTimeOffset.Now, "TestUserName");
    _userRepo.CheckPasswordAttempt(Arg.Any<string>(), Arg.Any<string>()).Returns(user);
    return user;
}

private void SetupSettingsRepo(string userId, string mainGroup) {
    _settingsRepository.GetSettings().Returns(
        new Settings(userId, null, null, new MainGroupSetting(mainGroup))
    );
}

private Group SetupGroupRepoReturnsGroup(string userId) {
    var group = new Group("TestGroup", userId);
    _groupRepo.GetGroup(Arg.Any<string>()).Returns(group);
    return group;
}
```

Das Beispiel DRY refactoring kommt aus den Unit Tests. Zuvor hat jeder Test selbst Mocks erstellt und Rückgabewerte konfiguriert, was zu viel duplizierten Code geführt hat.

Danach wurde diese Mock-Initialisierung in eigene Methoden ausgelagert, um den Erstellungsschritt wiederverwendbar zu gestalten und das P von ATRIP einzuhalten.

Kapitel 5: Unit Tests

10 Unit Tests

[Nennung von 10 Unit-Tests und Beschreibung, was getestet wird]

Unit Test (<i>Klasse#Methode</i>)	Beschreibung
<i>PasswordBuilderTest#PasswordBuilder_Should_IncludeLowercase</i>	Überprüft, ob PasswordGeneratorCriteria beim setzen von IncludeLowerCase, das auf true gesetzt wird.
AccountServiceTest#AccountService_ShouldNot_AddExistingAccount	Überprüft, ob beim erneuten Einfügen eines bereits existierenden Accounts in eine Group eine UserFeedbackException mit der passenden Nachricht geworfen wird.
LoginServiceTest#Login_Should_SetEnvironment	Überprüft, ob nach einem erfolgreichen Login, alle Environments (richtig) gesetzt wurden.
GroupServiceTest#GroupService_Should_SwitchGroup	Überprüft, ob die Gruppe auch wirklich gewitched wird.
CryptServiceTest#CryptService_Should_HashTheSamePassordIdentical	Überprüft, ob das gleiche Passwort mit dem gleichen Salt gleich gehashed wird.
GroupTest#Group_Should_GetValidAccount	Überprüft, ob der gewünschte Account zurückgegeben wird.
LoginControllerTest#Arguments_Should_Return_From_Prompts	Überprüft, ob bei einer eingabe der Parameter -d und -u diese richtig geparkt werden und vom Benutzer abgefragt werden.
SettingsServiceTest#SettingsService_Should_ChangeMainGroup	Überprüft, ob bei Änderung des MainGroupSettings, dies auch erfolgt.
DatabaseInitService#Init_ShouldNot_InitExistingDb	Stellt sicher, dass eine Datenbank nicht zweimal initiiert wird.
CommandParserTest#ArgParser_Should_ParseWithSpacesInQuotes	Stellt sicher, dass ein Argument, welches in Anführungszeichen gesetzt ist und Leerzeichen Enthält als ein Argument interpretiert wird.

ATRIP: Automatic

[Begründung/Erläuterung, wie 'Automatic' realisiert wurde]

Durch die Verwendung des xUnit-Frameworks von .NET ist es möglich alle Tests automatisiert mit einem Konsolenbefehl auszuführen (`dotnet test`). Dies wird in beim Ausführen der `test.bat`. Zusätzlich werden durch den GitHub Workfow `test_on_pr.yaml` alle Unit-Tests bei einer Pull-Request ausgeführt. Dabei ist keine manuelle Dateneingabe notwendig, da die Testdaten bereits im Test mit angegeben sind. Schlägt ein Test fehl, so zeigt GitHub dies an und verweigert den Merge des Pull-Requests.

ATRIP: Thorough

[jeweils 1 positives und negatives Beispiel zu ‘Thorough’; jeweils Code-Beispiel, Analyse und Begründung, was professionell/nicht professionell ist]

Positiv-Beispiel

```
[Fact]
public void Login_Should_SetEnvironment() {
    SetupDatabaseExists(true);
    var user = SetupPasswordCheckReturnsUser();
    var group = SetupGroupRepoReturnsGroup(user.Id);
    SetupSettingsRepo(user.Id, group.Identifier);

    _sut = new LoginService(_userRepo, _groupRepo, _cryptService, _settingsRepository, _cliEnv, _userEnv, _cryptEnv,
_dataContextInitializer);

    _sut.Login("TestUserName", "WhatAPasswort", ".");

    Assert.Equal(user, _userEnv.CurrentUser);
    Assert.NotNull(_cryptEnv.EncryptionKey);
    Assert.NotNull(_userEnv.CurrentGroup);
    Assert.True(_cliEnv.RunningSession);
}
```

Bei diesem Test handelt es sich um ein positiv-Beispiel, da es alle Punkte zum setzten des Environments bei einem erfolgreichen Login abdeckt.

Negativ-Beispiel

```
[Fact]
public void GroupService_Should_SwitchGroup() {
    var env = Substitute.For<IUserEnvironment>();
    var repo = MockGroupRepo();
    var group = new Group("a", "asd");
    repo.GetGroup(Arg.Any<string>()).Returns(group);

    _sut = new GroupService(env, repo);

    _sut.SwitchGroup("a");

    env.Received().CurrentGroup = group;
}
```

Die Testklasse GroupServiceTest sollte alle Funktionen des GroupService abdecken. Bei dem Codebeispiel oben wird zum Beispiel überprüft, ob das Wechseln der aktuellen Gruppe funktioniert. Allerdings wird in der Testklasse nicht überprüft, ob das Wechseln auf eine nicht-existente Gruppe unterbunden wird. Damit erfüllt die Testklasse GroupServiceTest nicht die Thorough-Eigenschaft.

ATRIP: Professional

[jeweils 1 positives und negatives Beispiel zu 'Professional'; jeweils Code-Beispiel, Analyse und Begründung, was professionell/nicht professionell ist]

Positiv-Beispiel

```
[Fact]
public void AccountService_ShouldNot_AddExistingAccount() {
    var env = MockUserEnvironmentWithGroup();
    _sut = new AccountService(env, null, null, null);

    var ex = Assert.Throws<UserFeedbackException>(() => _sut.AddNewAccount("AccountId", "Name", "password"));

    Assert.Equal(MessageStrings.AccountAlreadyExist("AccountId"), ex.Message);
}

private IUserEnvironment MockUserEnvironmentWithGroup() {
    var env = Substitute.For<IUserEnvironment>();
    env.CurrentGroup.Returns(new Group("GroupId", new List<Account>() {new Account("AccountId", "Name1", "Password1"), new Account("AccountId2", "Name2", "Password2")} ));
    return env;
}
```

Dies ist ein positives Beispiel für die Eigenschaft Professional, da dieser Test die Hilfsmethode MockUserEnvironmentWithGroup verwendet (welche ebenfalls von 12 weiteren Tests in dieser Testklasse verwendet wird), sowie einen wichtigen Aspekt des AccountServices überprüft. Dabei wird ebenfalls im Code dokumentiert, dass in einer Group kein Account Identifier doppelt verwendet werden kann.

Negativ-Beispiel

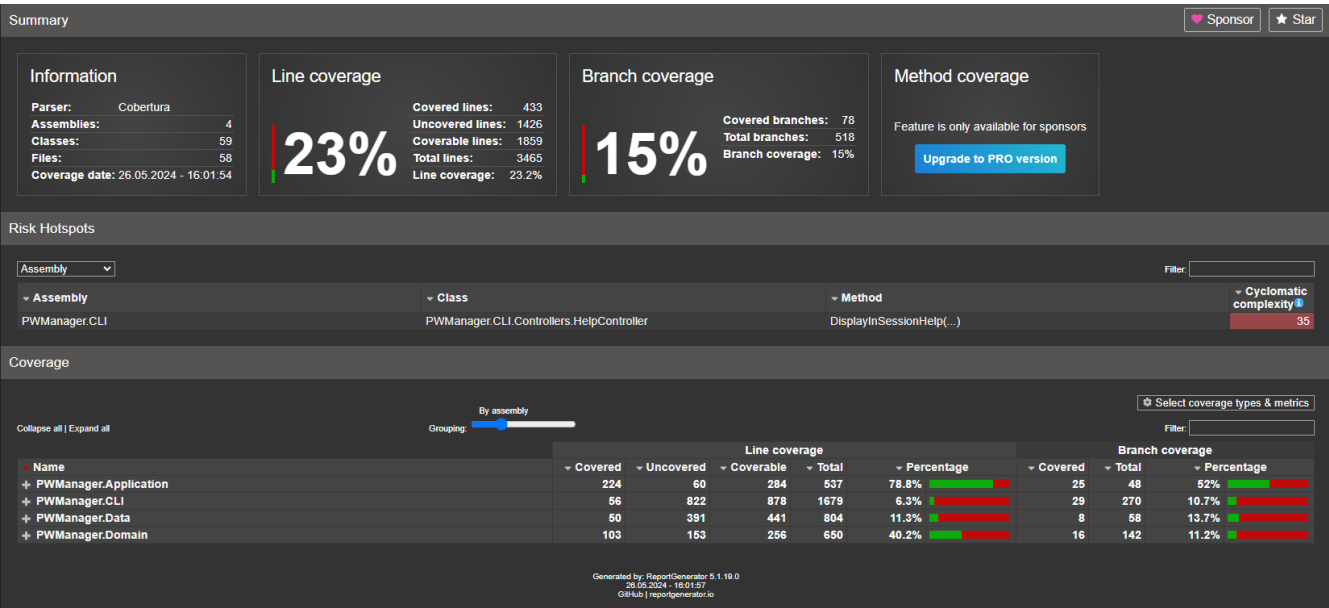
```
[Fact]
public void PasswordBuilder_Should_IncludeLowercase() {
    var criteria = PasswordBuilder.Create()
        .IncludeLowercase()
        .BuildCriteria();

    Assert.True(criteria.IncludeLowerCase);
}
```

Bei diesem Beispiel wird überprüft, ob PasswordGeneratorCriteria beim setzen von IncludeLowerCase, das auf true gesetzt wird. Dieser Test erfüllt nicht die Eigenschaft Professional, da dieser Test auch hätte erspart werden können. Hier wird lediglich geprüft ob, wenn etwas auf true gesetzt wird auch auf true ist (ähnlicher Mehrwert, wie ein „Getter“ zu testen).

Code Coverage

[Code Coverage im Projekt analysieren und begründen]



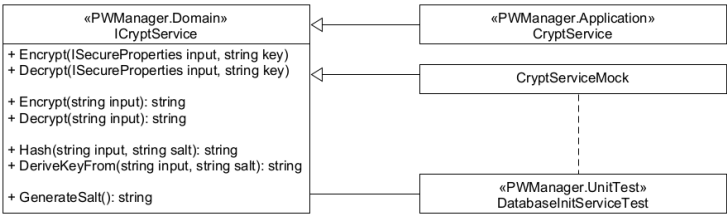
Betrachtet man den Code Coverage der Tests, so fällt auf, dass insgesamt 23% aller Zeilen und 15% aller Verzweigungen abgedeckt sind. Dies ist dem geschuldet, dass beim Testen die Aufmerksamkeit auf der Applikationsschicht lag, da dort die ganze Logik der Anwendung verbaut ist. In der Schicht sind 78.8% der Zeilen und 52% der Verzweigungen abgedeckt.

Fakes und Mocks

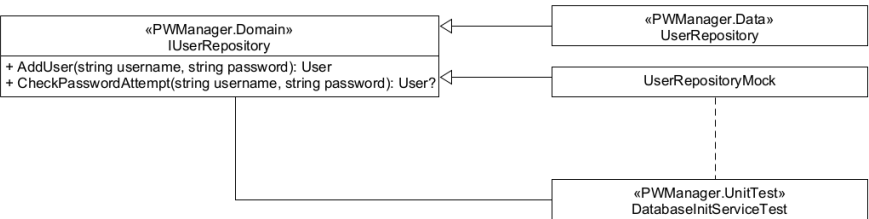
[Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten; zusätzlich jeweils UML Diagramm der Klasse]

In der Testklasse `DatabaseInitServiceTest` werden unter anderem die Mock-Objekte `CryptServiceMock` sowie `UserRepositoryMock` eingesetzt. Dies ist erforderlich, da bei Unit Tests eine Einheit getestet werden soll und daher kein direkter Zugang zu den anderen Komponenten bestehen soll. Die Mock-Klasse wird dann zur Laufzeit von der Library NSubstitute generiert.

CryptServiceMock:



UserRepositoryMock:



Kapitel 6: Domain Driven Design

Ubiquitous Language

[4 Beispiele für die Ubiquitous Language; jeweils Bezeichnung, Bedeutung und kurze Begründung, warum es zur Ubiquitous Language gehört]

Bezeichnung	Bedeutung	Begründung
Account Identifier	Die eindeutige Kennung, durch die ein Account in der angegebenen Group identifiziert wird. Die Kennung kann der Name des externen Dienstes sein, bei dem das Passwort verwendet wird (z. B. Gmail, Dropbox, GitHub).	Hierbei ist es wichtig zwischen der Bezeichnung eines Accounts und dem damit verbundenen `Login Name` zu unterscheiden. Account Name wäre hier falsch, da eine Verwechslung mit dem Anmeldenamen beim externen Dienst möglich wäre. Manche Benutzer haben den gleichen Anmeldenamen bei verschiedenen externen Diensten
Login Name	Die E-Mail-Adresse oder der Benutzername, der für die Authentifizierung beim angegebenen externen Dienst verwendet wird.	Auch hier sollte der Begriff `Account` in der Bezeichnung vermieden werden, um keine Verwechslung mit dem Begriff `Account Identifier` in der Domäne zu verursachen.
Password	Das Passwort für den bestimmten Account.	Eindeutiger Begriff aus der Domäne.
Master Password	Das Passwort für den Login in die Nutzerdatenbank.	Zur Differenzierung zu `Password`.

Entities

[UML, Beschreibung und Begründung des Einsatzes einer Entity; falls keine Entity vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

Account
- Identifier: string - LoginName: string - Password: string

Der Hauptfokus eines Passwortmanagers ist es, Passwörter zu speichern. Um diese zu den jeweiligen externen Diensten zuordnen zu können müssen diese in einem Objekt gespeichert werden, ein `Account`. Es handelt sich dabei um ein Entity, da es eine eindeutige Zuordnung `Identifier` gibt und die anderen Attribute `LoginName` und `Password` sich über die Zeit ändern können (dabei handelt es sich dann aber immer noch um den gleichen Account).

Value Objects

[UML, Beschreibung und Begründung des Einsatzes eines Value Objects; falls kein Value Object vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

PasswordGeneratorCriteria
- IncludeLowerCase: bool - IncludeUpperCase: bool - IncludeNumeric: bool - IncludeSpecial: bool - IncludeSpaces: bool - IncludeBrackets: bool - MinLength: int - MaxLength: int

Als zusätzliches Sicherheitsfeature eines Passwortmanagers, ist es hilfreich sich Passwörter auch zufällig generieren zu lassen. Dabei spielt die `PasswordGeneratorCriteria` eine wichtige Rolle, da manche externe Dienste unterschiedliche Ansprüche an Passwortkriterien haben. Hierbei handelt es sich um ein Value Object, da hier keine eigene Identität aufzuweisen ist und über die Eigenschaften (den unterschiedlichen Kriterien) definiert ist.

Repositories

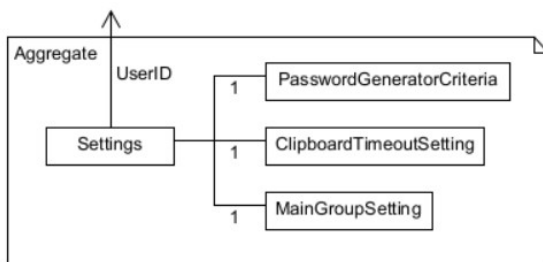
[UML, Beschreibung und Begründung des Einsatzes eines Repositories; falls kein Repository vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

IGroupRepository
+ GetGroup(string groupName): Group + GetAllGroupNames(): List<string> + AddGroup(Group group): bool + UpdateGroup(Group group): bool + AddAccountToGroup(Account account, Group group): bool + UpdateAccountInGroup(Account account, Group group): bool + DeleteAccountInGroup(Account account, Group group): bool + RemoveGroup(string groupName): bool

Zur Group muss man regelmäßig Daten abfragen, daher ist an dieser Stelle ein Repository sinnvoll. Dieses Repository handelt sowohl das Lesen, Suchen als auch das Schreiben und Löschen einer Group.

Aggregates

[UML, Beschreibung und Begründung des Einsatzes eines Aggregates; falls kein Aggregate vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]



Hierbei handelt es sich um eine Arbeitseinheit rund um die Einstellungen. Individuelle Einstellungen sind, z. B. das Einstellen der PasswordGeneratorCriteria ist nur durch das Settings Entity möglich.

Kapitel 7: Refactoring

Code Smells

[jeweils 1 Code-Beispiel zu 2 Code Smells aus der Vorlesung; jeweils Code-Beispiel und einen möglichen Lösungsweg bzw. den genommen Lösungsweg beschreiben (inkl. (Pseudo-)Code)]

Long Method

Commit:

(<https://github.com/CUMGroup/PWManager/commit/54868cecc166b425d23575baabda83d35aeed1df>)

Vor dem Commit (ID: 54868cecc166b425d23575baabda83d35aeed1df):

```
public ExitCondition Handle(string[] args) { // TODO: eventuell refactoring
    var option = Prompt.Select(UIstrings.SELECT_ACTION, new[] { newGroup, switchGroup, listAllGroups,
deleteGroup, exit });

    if (option.Equals(exit)) {
        return ExitCondition.CONTINUE;
    }

    if (option.Equals(newGroup)) {
        CreateNewGroupAndSwitchToIt();
        return ExitCondition.CONTINUE;
    }

    if (option.Equals(switchGroup)) {
        SwitchGroup(_groupService.GetAllGroupNames());
        return ExitCondition.CONTINUE;
    }

    if (option.Equals(listAllGroups)) {
        var groups = _groupService.GetAllGroupNames();
        PromptHelper.PrintPaginated(groups);
        return ExitCondition.CONTINUE;
    }

    if (option.Equals(deleteGroup)) {
        var currentgroup = _userEnv.CurrentGroup!.Identifier;
        HandleDeletion(currentgroup);
    }

    return ExitCondition.CONTINUE;
}
```

Zur Lösung dieses Code Smells, wurden die möglichen Aktionen mithilfe eines Enums zur `ExecuteAction()` ausgelagert. In der Handle-Methode ist demnach nur noch eine kleine Do-While-Schleife, welche überprüft, ob es ein Grund gibt aus den Menü auszusteigen.

Nach dem Commit (ID: 54868cecc166b425d23575baabda83d35aeed1df):

```
public ExitCondition Handle(string[] args) {  
  
    GroupAction action;  
    var executed = false;  
    do {  
        action = GetGroupAction();  
        executed = ExecuteAction(action);  
    } while ((action != GroupAction.DELETE_GROUP || !executed) && action != GroupAction.RETURN);  
  
    return ExitCondition.CONTINUE;  
}  
  
private GroupAction GetGroupAction() {  
    throw new NotImplementedException();  
}  
  
private bool ExecuteAction(GroupAction action) {  
    return action switch {  
        GroupAction.NEW_GROUP => HandleCreateNewGroupAndSwitchToIt(),  
        GroupAction.SWITCH_GROUP => HandleSwitchGroup(_groupService.GetAllGroupNames()),  
        GroupAction.LIST_GROUPS => HandleListAllGroups(),  
        GroupAction.DELETE_GROUP => HandleDeletion(),  
        GroupAction.RETURN => true,  
        _ => false  
    };  
}
```

Die Methode GetGroupAction() wurde anschließend im nachfolgenden Commit (<https://github.com/CUMGroup/PWManager/commit/368a5df45e873bb6cd276fd22f5f40466e510e53>) implementiert:

```
private GroupAction GetGroupAction() {  
    return Prompt.Select<GroupAction>(UIstrings.SELECT_ACTION);  
}
```

Switch Statements

```
private bool ExecuteAction(GroupAction action) {  
    return action switch {  
        GroupAction.NEW_GROUP => HandleCreateNewGroupAndSwitchToIt(),  
        GroupAction.SWITCH_GROUP => HandleSwitchGroup(_groupService.GetAllGroupNames()),  
        GroupAction.LIST_GROUPS => HandleListAllGroups(),  
        GroupAction.DELETE_GROUP => HandleDeletion(),  
        GroupAction.RETURN => true,  
        _ => false  
    };  
}
```

Die Switch Statements in der Methode `ExecuteAction` können durch einführen einer neuen Oberklasse `GroupAction` (anstelle des Enums), die eine Handle- Methode vorgibt. Von der Klasse können dann die entsprechenden Unterklassen, NewGroup etc., erben und eine eigenen Implementation der Handle-Methode umsetzen.

Pseudocode:

```
Class GroupAction {
    public GroupAction(){}
    public Handle(){}
}

Class NewGroupAction : GroupAction {
    public NewGroupAction(){}
    public Handle(){
        // create new group
    }
}

Class SwitchGroupAction : GroupAction {
    private string[] GroupNames;
    public SwitchGroupAction(string[] groupNames){
        this.GroupNames = groupNames;
    }
    public Handle(){
        // switch group
    }
}

...
```

Restliche sind äquivalent zu NewGroupAction.

2 Refactorings

[2 unterschiedliche Refactorings aus der Vorlesung anwenden, begründen, sowie UML vorher/nachher liefern; jeweils auf die Commits verweisen]

Extract Method

Commit:

(<https://github.com/CUMGroup/PWManager/commit/a96ce83e55e9d5763fc37074a1e716c41d1ac301>)

Um Codeduplizierungen zu verringern, wurden in der LoginServiceTest Klasse einige sich wiederholenden Zeilen in zusätzlichen Hilfsmethoden ausgelagert. Dadurch wird der Code verständlicher. Dadurch sind neue private Methoden in der Klasse entstanden.

UML vor Refactor:

LoginServiceTest
- _groupRepo: IGroupRepository - _cryptService: ICryptService - _settingsRepository: ISettingsRepository - _wrapper: DataContextWrapper - _cliEnv: ICliEnvironment - _userEnv: IUserEnvironment - _cryptEnv: ICryptEnvironment - _userRepo: IUserRepository
+ Login_Should_SetEnvironment() + Login_ShouldNot_IfDatabaseDoesntExists() + Login_ShouldNot_IfUserNotFound()

UML nach dem Refactor:

LoginServiceTest
- _groupRepo: IGroupRepository - _cryptService: ICryptService - _settingsRepository: ISettingsRepository - _wrapper: DataContextWrapper - _cliEnv: ICliEnvironment - _userEnv: IUserEnvironment - _cryptEnv: ICryptEnvironment - _userRepo: IUserRepository
+ Login_Should_SetEnvironment() + Login_ShouldNot_IfDatabaseDoesntExists() + Login_ShouldNot_IfUserNotFound()
- SetupDatabaseExists(bool exists) - SetupPasswordCheckReturnsUser(): User - SetupPasswordCheckThrowsException() - SetupSettingsRepo(string userId, string mainGroup) - SetupGroupRepoReturnsGroup(string userId): Group

Rename Method

Commit:

(<https://github.com/CUMGroup/PWManager/commit/54868cecc166b425d23575baabda83d35aeed1df>)

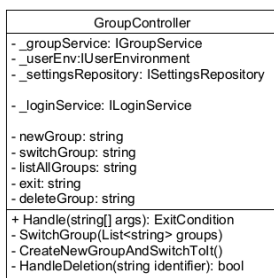
```

66 - private void SwitchGroup(List<string> groups) {
67     if (!groups.Any()) {
68         throw new UserFeedbackException("There are no groups in your database. Something is
69         really wrong!"); // TODO: Exception Message auslagern
70     }
71     var groupIdentifier = Prompt.Select(UiStrings.SWITCH_GROUP_PROMPT, groups);
72     _groupService.SwitchGroup(groupIdentifier);
73 }
74
75 - private void CreateNewGroupAndSwitchToIt() {
76     var groupIdentifier = PromptHelper.GetInput(UiStrings.NEW_GROUP_NAME);
77     _groupService.AddGroup(_userEnv.CurrentUser!.Id, groupIdentifier);
78     _groupService.SwitchGroup(groupIdentifier);
79     PromptHelper.PrintColoredText(ConsoleColor.Green, UiStrings.GROUP_SWITCH_CONFIRM);
80 }
81
82 }
83
68 + private bool HandleSwitchGroup(List<string> groups) {
69     if (!groups.Any()) {
70         throw new UserFeedbackException("There are no groups in your database. Something is
71         really wrong!"); // TODO: Exception Message auslagern
72     }
73     var groupIdentifier = Prompt.Select(UiStrings.SWITCH_GROUP_PROMPT, groups);
74     _groupService.SwitchGroup(groupIdentifier);
75     return true;
76 }
77
78 + private bool HandleCreateNewGroupAndSwitchToIt() {
79     var groupIdentifier = PromptHelper.GetInput(UiStrings.NEW_GROUP_NAME);
80     _groupService.AddGroup(_userEnv.CurrentUser!.Id, groupIdentifier);
81     _groupService.SwitchGroup(groupIdentifier);
82     PromptHelper.PrintColoredText(ConsoleColor.Green, UiStrings.GROUP_SWITCH_CONFIRM);
83     return true;
84 }
85

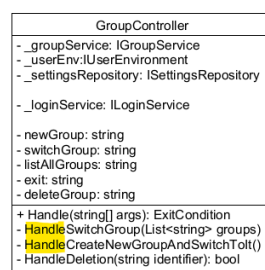
```

Zur besseren Lesbarkeit wurden die Methoden `SwitchGroup` und `CreateNewGroupAndSwitchToIt` der Klasse `GropController` zu `HandleSwitchGroup` und `HandleCreateNewGroupAndSwitchToIt` umbenannt. Dies ist dem Bedingt, da in den Klassen `SettingsController` sowie `AccountController` eine ähnliche Logik mit der Nomenklatur `Handle...`. Durch dieses Refactoring ist die Nomenklatur einheitlicher.

UML vor Refactor:



UML nach dem Refactor:

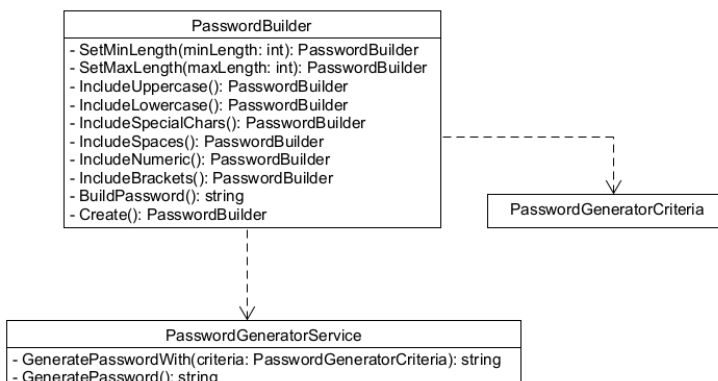


Kapitel 8: Entwurfsmuster

[2 unterschiedliche Entwurfsmuster aus der Vorlesung (oder nach Absprache auch andere) jeweils sinnvoll einsetzen, begründen und UML-Diagramm]

Entwurfsmuster: Erzeugungsmuster (Erbauer)

Der PasswordBuilder erzeugt anhand der PasswordGeneratorCriteria mithilfe des PasswordGeneratorServices ein neues Passwort. Das Erzeugungsmuster hilft bei der Erzeugung eines Passworts zu unterschiedlichen Einstellungen.



Entwurfsmuster: Verhaltensmuster (Beobachter)

Bei der Umsetzung dieses Entwurfsmuster bietet die Clipboard Klasse ein Event (Sprachfeature von C#) an, den andere Klassen beobachten können. Ein Beobachter ist der ClipboardTimeoutService. Dadurch bekommt dieser mit, wenn sich das Clipboard updatet (z.B. wenn ein Passwort in die Zwischenablage kopiert werden soll). Der ClipboardTimeoutService kann dadurch eine gewissen Zeit nachdem in das Clipboard geschrieben wurde, ClearClipboard aufrufen.

