# Dokumentation der Projektarbeit „Erstellung einer Mitarbeiterverwaltungssoftware“

21.03.2024

Christian-Schmidt-Schule Neckarsulm
Fach: BTS
Denis Root
Erik Schuckmann
Sebastian Schwarzer

# Inhaltsverzeichnis:

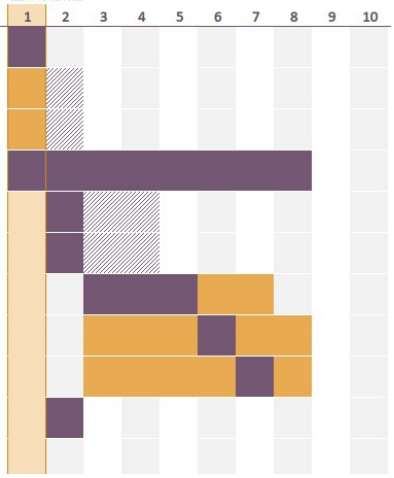
# 1. Motivation und Problemstellung

Die Aufgabe dieses Projekts war es, eine Mitarbeiterverwaltungssoftware zu entwickeln. Als Ausgangssituation wurde ein fiktives Szenario vorgegeben, in dem ein „Herr Pfuscher“ eine Firma übernimmt, die ihre Verwaltung bisher ausschließlich über Mitarbeiterakten durchführte, welche nun durch eine zeitgemäßere Software ersetzt werden sollen.
Grundsätzlich sollte die Software die Möglichkeit haben, Daten von Mitarbeitern zu erfassen und zu speichern. Hierzu wurden bestimmte Daten bereits vorgegeben. Die Mitarbeiterdaten sollten neu angelegt, gesucht, geändert und gelöscht werden können. Zur Speicherung der Daten waren als grundlegende Form Dateien vorgegeben, wie beispielsweise Text-/XML- und Excel-Dateien. Diese Dateien sollten sowohl exportiert als auch wieder importiert werden können.
Zur Durchführung dieses Projekts wurde die Klasse in 2-er und 3-er Gruppen aufgeteilt. Die Mitglieder unserer Gruppe sind Denis Root, Erik Schuckmann und Sebastian Schwarzer. Neben dem Erstellen der Software selbst sollten auch Struktogramme und andere Diagramme erstellt werden, um das Projekt zu dokumentieren. Sinn und Zweck der Aufgabe war es, Projektkompetenz zu entwickeln, die bereits gelernten Programmierkenntnisse in BTS anzuwenden und sich neue Programmier-Themen wie Arrays oder gegebenenfalls objektorientiertes Programmieren (OOP) anzueignen.

# 2. Grundüberlegungen

## 2.1 Zeitlicher Rahmen

| AKTIVITÄT | START DES PLANS | DAUER DES PLANS | TATSÄCHLICHER START | TATSÄCHLICHE DAUER | ABGESCHLOSSEN |
|---|---|---|---|---|---|
| Erstellung Projektplan | 1 | 1 | 1 | 1 | 100% |
| Benutzeroberfläche (GUI) | 2 | 1 | 1 | 1 | 100% |
| Erstellung Use-Cases | 2 | 1 | 1 | 1 | 100% |
| Dokumentieren Erstellung | 1 | 8 | 1 | 8 | 100% |
| Datenbankschnitstelle Modellieren der | 2 | 3 | 2 | 1 | 100% |
| Datenbankmodelle | 2 | 3 | 2 | 1 | 100% |
| Erstellen der Logik | 3 | 3 | 3 | 5 | 100% |
| Testen Verbesserungen | 6 | 1 | 3 | 6 | 100% |
| durchführen Struktogramme | 7 | 1 | 3 | 6 | 100% |
| erstellen | 2 | 1 | 2 | 1 | 100% |



Für die Durchführung des Projekts waren in etwa zwei Monate veranschlagt. Zu Beginn wurde ein Gantt-Diagramm erstellt, um die zeitlichen Meilensteine in etwa vorzugeben. Zunächst sollte mit organisatorischen Punkten angefangen werden, wie der Erstellung des Projektplans, und durch Erstellen von GUI, Strukogrammen und der Datenbankstruktur die Grundlagen für den Hauptteil der Arbeit, dem Erstellen der Logik, gelegt werden. Zum Schluss sollte Zeit für das Testen und Verbesserungen eingeplant und zusätzlich eine Woche Puffer gegeben werden, falls Schwierigkeiten auftreten sollten. Während der gesamten Zeit sollte auch an der Dokumentation gearbeitet werden, sowohl durch das Anfertigen von Notizen als auch durch das Schreiben der Dokumentation selbst. Am Ende wurde die Zeit von acht Wochen eingehalten; wir erhielten zwar mehr Zeit, es gab aber auch Unterrichtsausfall. Es zeigte sich, dass einige der Grundlagenaufgaben wie das Erstellen der GUI und Datenbank weniger Zeit in Anspruch nahmen als geplant. Auf der anderen Seite dauerte das Erstellen der Logik länger. Zudem begannen wir quasi mit Erstellen der Logik gleichzeitig auch mit dem Testen der Anwendung und dem Anpassen und Verbessern der bereits vorhandenen Komponenten. Dies wurde auch bis zum Abschluss der Projektarbeit durchgängig vollzogen.

## 2.2 Auswahl der genutzten Tools

Da wir im Unterricht mit Visual Studio gearbeitet und in C# programmiert haben, bot es sich an, das Projekt ebenfalls in der Form durchzuführen. Die Mitarbeitersoftware sollte als GUI-Programm erstellt werden, um eine einfache Bedienung durch den Nutzer zu ermöglichen. Obwohl wir GUI-Programme im Unterricht in der Regel mit Windows Forms erstellt haben, kam der Vorschlag, stattdessen die Windows Presentation Forms (WPF) zu verwenden. Wir entschieden uns für WPF, da Windows Forms die Eigenart hat, Probleme bei der Ausführung des Programms zu verursachen, wenn beispielsweise Steuerelemente im Programmcode wieder entfernt werden. Bei WPF können Steuerelemente einfach per XML hinzugefügt und wieder entfernt werden.
In den Projektanforderungen wurde gewünscht, dass die Mitarbeiterdaten in Dateien gespeichert und wieder eingelesen werden können, wofür beispielsweise das json – oder CSV-Format genutzt werden könnte. Aufgrund eigener Vorerfahrungen entschieden wir uns jedoch bereits dazu, anstelle eines einfachen Import/Export-Systems eine Datenbank zu nutzen. Hier fiel die Wahl auf das Microsoft Entity Framework. Entity Framework lässt sich in WPF und mit C# nutzen, da es auch von Microsoft entwickelt wurde. Davon erhofften wir uns eine bessere Kompatibilität mit der genutzten IDE und Sprache. Ein weiterer Vorteil des Entity Framework ist es, dass z.B. die SQL-Abfragen alle automatisch im Hintergrund ablaufen, was eine einfachere Bedienung für uns ermöglichte. Dennoch wollten wir zumindest die Möglichkeit bieten, Dateien importieren und

exportieren zu können. Im Laufe des Projekts wechselten wir die IDE von Visual Studio auf JetBrains Rider, um einige Probleme zu umgehen, auf die später näher eingegangen wird.
Damit alle Gruppenmitglieder gleichermaßen am Projekt arbeiten konnten, entschieden wir uns ebenso dazu, Github als Distributionsplattform zu nutzen. Dort haben wir ein Repository angelegt, wo der gesamte Programmcode zu finden ist[1].

## 2.3 Anforderungen & Überlegungen zur Umsetzung

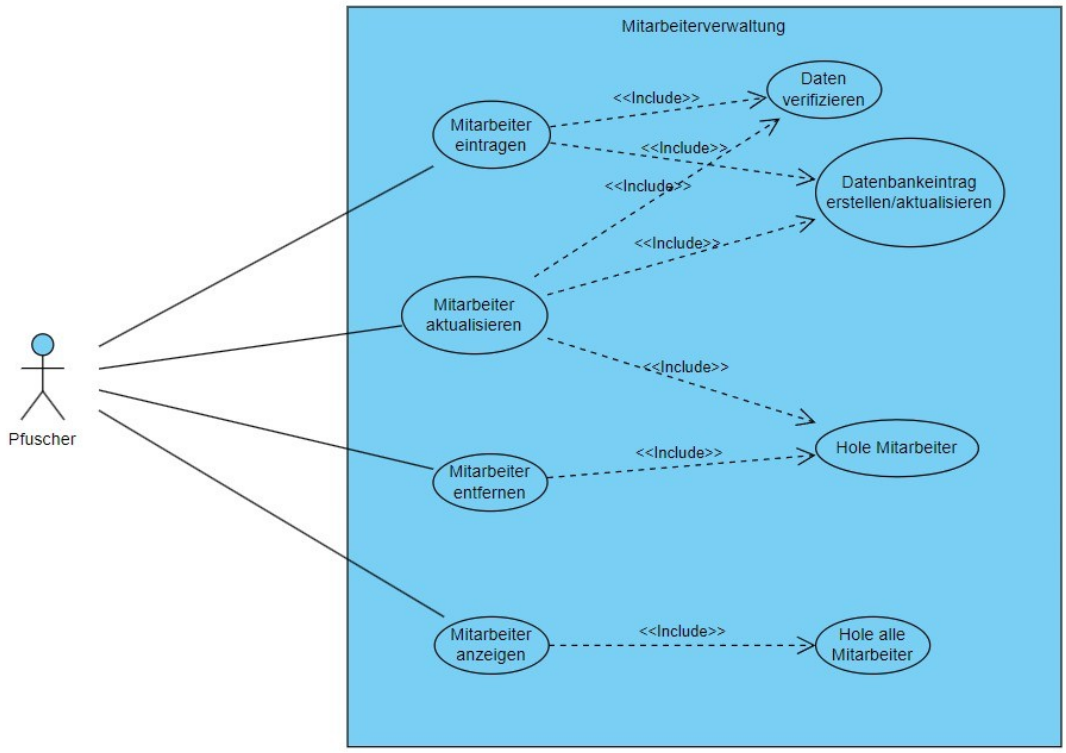
Wie eingangs erwähnt, handelt es sich bei dem Programm um eine Verwaltungssoftware, mithilfe derer Mitarbeiter neu angelegt, gesucht, bearbeitet und gelöscht werden können. Ausgehend von diesen Anforderungen lassen sich vier Basismethoden ableiten, die diese Funktionen übernehmen und idealerweise jeweils durch Klicken eines Buttons ausgeführt werden. Von jedem Mitarbeiter sollen bestimmte Daten erfasst werden. Dazu zählen laut Aufgabenstellung der (Nach-)Name, Vorname, die Adresse, Telefonnummer, Emailadresse, Position, das Eintrittsdatum in die Firma, Gehalt und der Rentenbeginn. Zur Darstellung dieser Daten bot es sich an, eine einfache Liste zur erstellen, in der die einzelnen Daten als Spalten fungieren.
Daraus ergab sich die Idee der groben Darstellung der Oberfläche, die in zwei Bereiche aufgeteilt werden sollte; einem etwas kleineren Bereich, in dem die Buttons zu finden sind, sowie einem größeren Bereich, in dem die Mitarbeiterliste ausgegeben wird. Ersterer Bereich soll zur Möglichkeit der Suche um eine Suchleiste mit Suchbutton erweitert werden. Eine weitere Anforderung an die Software war die Möglichkeit, Mitarbeiterdaten importieren und exportieren zu können. Dazu war es notwendig, zwei weitere Buttons hinzuzufügen, die Methoden zur Umsetzung dieser Anforderung bereitstellen.
Schließlich sollte die Möglichkeit zum Hinzufügen und Bearbeiten von Mitarbeitern gegeben sein. Zur optimierten Nutzung des vorhandenen Platzes kam die Idee auf, den Bereich auf der Oberfläche zu verwenden, der von der Liste eingenommen wird. Daher sollte eine Funktion implementiert werden, die die Mitarbeiterliste ausblendet und stattdessen Textfelder zur Eingabe von Mitarbeiterdaten bereitstellt. Im Falle, dass ein Mitarbeiter bearbeitet wird, würden die Textfelder mit den gespeicherten Information befüllt werden. Es sollten zudem Buttons zum Speichern der Änderungen, sowie zum Abbrechen existieren.
Neben der Umsetzung des Programms selbst, sollten weitere Aufgaben erledigt werden. Dazu zählten die Erstellung eines Gantt-Diagramms, eines Use-Case-Diagramms und eines oder zweier Strukogramme, die die Funktion der erstellten Methoden aufzeigen.

1 Der Link befindet sich im Anhang (S. 15).

# 3. Umsetzung des Programms

## 3.1 Vorbereitung: Use-Case-Diagramm, Strukogramme

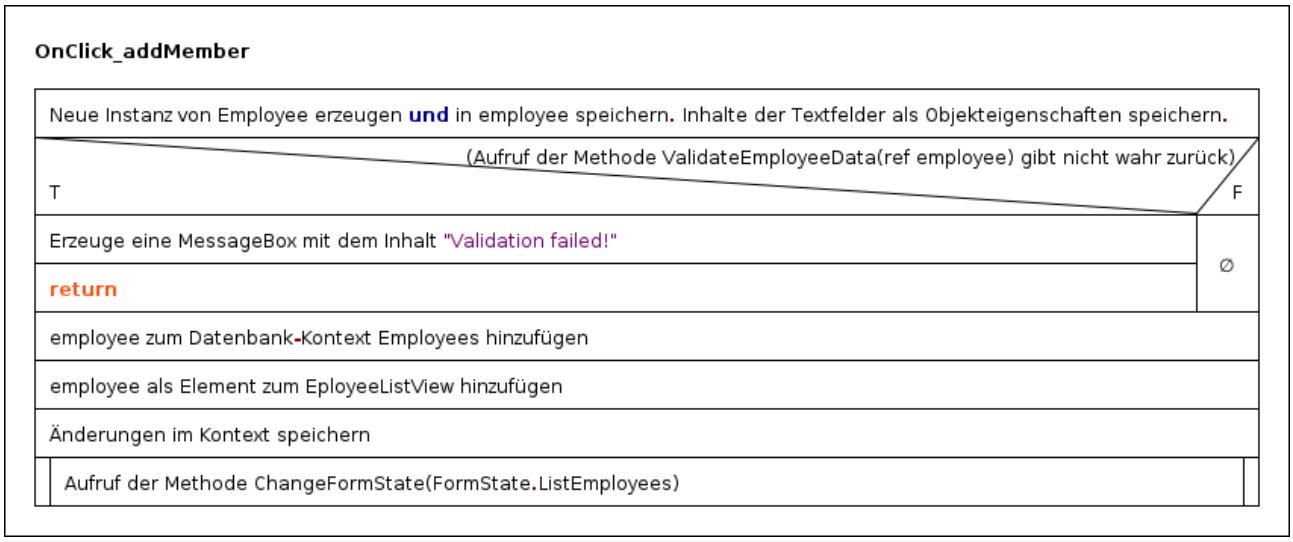
Die Vorbereitung des Programms begann mit der Erstellung eines Use-Case-Diagramms.




Das Use-Case-Diagramm sollte uns helfen, für den Anfang zu visualisieren, welche Anforderungen die Software genau umsetzen soll, sprich, wie man als Nutzer mit ihr interagieren können soll. Zu Beginn hatten wir vier Funktionen im Sinn: Mitarbeiter hinzuzufügen, zu bearbeiten, zu entfernen und anzuzeigen. Diesen „Szenarien“ wurden dann „Ziele“ zugewiesen, die die Software mit ihren Funktionen verwirklichen soll, wie Daten zu verifizieren, die Datenbank zu erstellen und zu aktualisieren, oder einen bestimmten bzw. mehrere Mitarbeiter zu holen. Im Laufe der Arbeit am Projekt stellte sich dann heraus, dass wir für unsere Software weitere Funktionen hinzufügen wollten, die im Use-Case-Diagramm anfangs noch nicht beschrieben waren, wie das Suchen oder der Import und Export von Mitarbeitern.

Für den Entwurf der Methoden können wiederum Struktogramme verwendet werden. Der Begriff ist eine allgemeinere Bezeichnung für das Nassi-Shneidermann-Diagramm und beinhaltet

Programm – und Kontrollstrukturen, damit diese unabhängig von der genutzten Programmiersprache veranschaulicht werden können. Struktogramme können so helfen, Programmcode darzustellen, um diesen entweder einfacher zu verstehen, oder eine Anleitung zu geben, nach welcher der Code erstellt werden soll.
Im Folgenden werden zwei Struktogramme aus unserem Programmcode gezeigt, die jeweils eine Methode abbilden. Beide Struktogramme wurden mit dem Programm „Structorizer“ erstellt, das kostenlos im Internet heruntergeladen werden kann und auf mehreren Betriebssystemen läuft. Unsere Wahl fiel auf diese Methoden, da sie einerseits nicht zu lang sind, andererseits aber auch Kontrollstrukturen beinhalten, die wir im Rahmen von Struktogrammen bereits verwendet haben; eine if-Kontrollstruktur und einen Case-Switch. Dazu beinhalten sie mit dem Methoden-Aufruf ein Element, das im Unterricht noch nicht behandelt wurde. Hierbei zeigte sich jedoch eine Schwierigkeit in Verbindung mit dem objektorientierten Ansatz, den wir in der Aufgabe verfolgen, da zunächst unklar war, wie genau beispielsweise das Erzeugen eines neuen Objekts mit Zuweisung von Eigenschaften in einem Struktogramm dargestellt werden soll. Schließlich sind die Methoden auch wichtige Bestandteile des Programms und bauen aufeinander auf.




Die erste Methode ist die OnClick_addMember-Methode, die aufgerufen wird, wenn der „Hinzufügen“-Button im Mitarbeiterformular geklickt wird.

**ChangeFormState(FormState formState)**

| Aufruf der Methode ClearForm() | | |
|---|---|---|
| (formState)<br>FormState.ListEmployees | FormState.AddNewEmployee | FormState.EditExistingEmployee |
| Visbility des EmployeeListViewContainers auf Visbility.Visible stellen | Visibility des EmployeeListViewContainer auf Visibility.Collapsed stellen | Visbility des EmployeeListViewContainer auf Visbility.Collapsed stellen |
| Visbility der EmployeeForm auf Visbility.Collapsed stellen | Visbility der EmployeeForm auf Visbility.Visible stellen | Visbility der EmployeeForm auf Visibility.Visible stellen |
| | Visibility des AddMemberButton auf Visbility.Visible stellen | Visibility des AddMemberButton auf Visbility.Hidden stellen |
| | Visbility des SaveChangesButton auf Visbility.Hidden stellen | Visbility des SaveChangesButton auf Visbility.Visible stellen |

Die zweite Methode ist die ChangeFormState-Methode, welche die Darstellung des rechten Teils der Oberfläche bestimmt. Diese wird zum Beispiel am Ende der vorherigen Methode aufgerufen.

## 3.2 GUI mit WPF

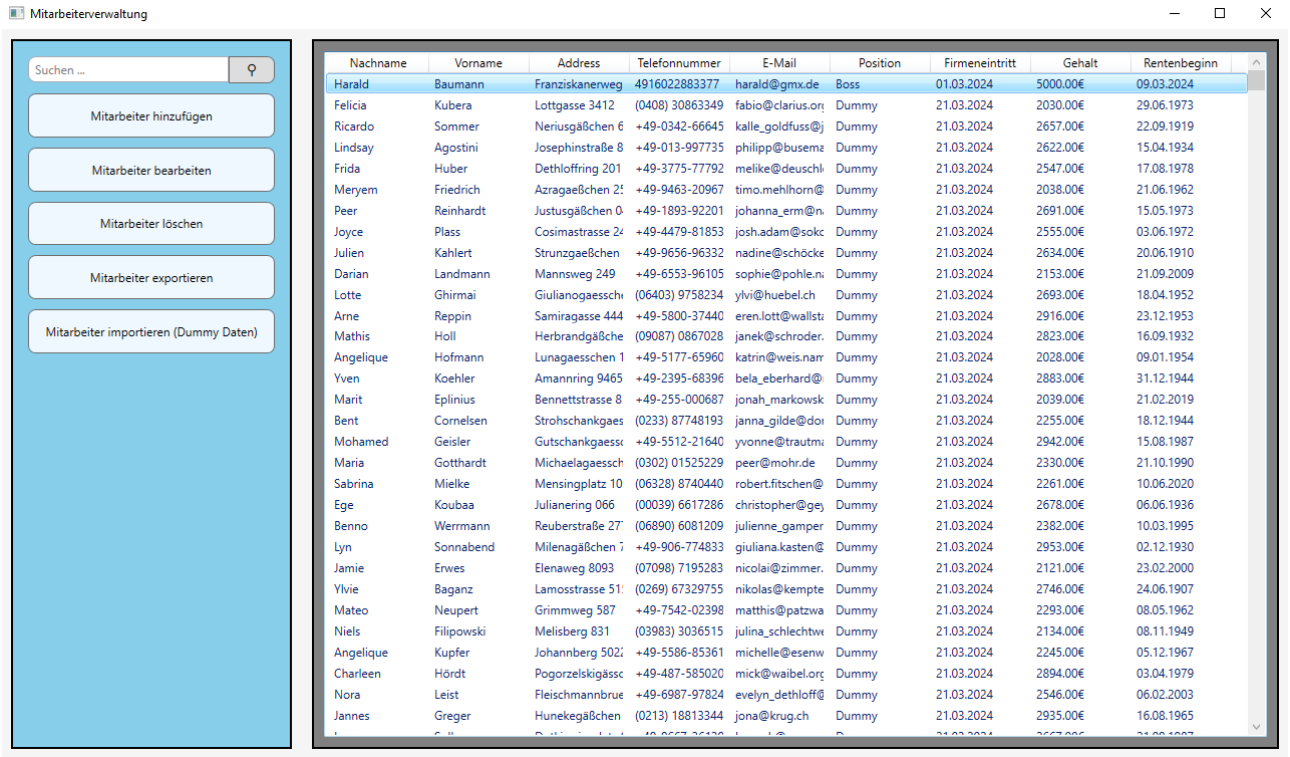
Zur Erstellung der GUI verwendeten wir die Windows Presentation Forms. Der Umgang mit WPF unterschied sich im Vergleich zu Windows Forms vor allem darin, dass Steuerelemente nicht per Drag and Drop auf einer Oberfläche hinzugefügt werden, sondern dass sie in der MainWindow.xaml Datei per XML-Code definiert werden.[2] Der Beginn der Umsetzung der GUI erfolgte bereits ebenfalls am ersten Projekttag.
Die grobe Einteilung der Oberfläche erfolgte in zwei Teile; der linke Teil sollte die Interaktion mit der Mitarbeitertabelle erlauben, während der rechte Teil die Mitarbeiterliste darstellt. Zur Einteilung wurde das Grid-Element verwendet, in dem zunächst die Breite der Spalten definiert wurde. Wir entschieden uns für eine dynamische Breite der Mitarbeiterliste, um die Möglichkeit zur Darstellung von so vielen Informationen wie möglich auf einen Blick zu erlauben. Die Grid-Spalte der Buttons erhielt eine feste Breite. Die Buttons wurden in einem StackPanel-Element definiert, das eine Ausrichtung jener untereinander erlaubt. Ebenso enthält das Element ein DockPanel-Element, in dem die Suchleiste per TextBox und Button definiert wurde. Die Buttons erhielten ein einheitliches Padding und OnClick-Methoden, und die gesamte Spalte eine schwarze Border zur klaren Abtrennung vom Rest des Fensters.
Die rechte Grid-Spalte erhielt ebenfalls eine Border und sollte die Mitarbeiterliste darstellen. Hierzu wurde ein ListView-Element „EmployeeListView“ erstellt, in welchem wiederum ein GridView-Element eingefügt wurde. Diesem GridView-Element können, ähnlich wie dem Grid-Element, Spalten zugewiesen werden. Wir erstellten ein GridViewColumn für alle Mitarbeiterinformationen, die wiederum jeweils ein Binding zu einer Eigenschaft der Mitarbeiterklasse erhielten, um die Daten eines Mitarbeiterobjekts auszugeben.
Da die Mitarbeiterliste selbst beim Hinzufügen oder Bearbeiten eines Mitarbeiters nicht relevant ist,

2 Ein Auschnitt der MainWindow.xaml-Datei befindet sich im Anhang (S. 17).

entschieden wir uns, den Platz, den die Liste in der GUI belegte, ebenfalls für die Darstellung des Formulars zu verwenden. Das Border-Element der ListView hat ein Visbility-Attribut, das standardmäßig auf „Visible“ gesetzt ist. Dieses Attribut machten wir uns zu Nutzen, indem wir ein ScrollViewer-Element für das Formular erstellten, und den Attribut-Wert der Border hier auf „Collapsed“ setzten. Dies erlaubte uns später, eine Methode zu programmieren, die die Sichtbarkeit der Elemente ändert, sodass an gleicher Position in der GUI entweder die Mitarbeiterliste oder das Formular erscheinen.

Für das Formular verwendeten wir erneut ein Grid-Element, in dem die einzelnen TextBox-Elemente ähnlich wie die Suchleiste in DockPanels eingefügt wurden, die wiederum einem StackPanel angehörten, damit die TextBox-Elemente untereinander angeordnet werden. Schließlich wurden unter den Textfeldern drei Buttons hinzugefügt, die das Hinzufügen des Mitarbeiters, Speichern oder Abbrechen des Vorgangs ermöglichen. Für den Hinzufügen – und den Speichern-Button wurde erneut das Visbility-Attribut genutzt, da diese Buttons nur respektive für das Hinzufügen und Bearbeiten eines Mitarbeiters verwendet werden sollen und somit auch unterschiedliche OnClick-Methoden besitzen.

Zum Abschluss wurde das Design angepasst, indem wir Hintergrundfarben der Panels und Buttons bestimmten und die Buttons abrundeten. Dazu verwendeten wir die dazu nötigen Attribute.



## 3.3 Datenbank mit Entity Framework

Die Mitarbeiterdaten sollen in irgendeiner Form persistiert werden. Das bedeutet, dass sie zum Beispiel nach dem Schließen des Programms nicht einfach verschwinden. Zu diesem Zweck entschieden wir uns für die Nutzung einer Datenbank, da dies eine gängige und komfortable Möglichkeit zum Speichern der Daten ist; gerade im Falle des objektorientierten Programmierens, wo ein Import – und Exportsystem mit Textdateien vielleicht etwas umständlicher erscheinen könnte.
Unsere Wahl fiel hier auf das Entity Framework von Microsoft. Dies hatte zwei Gründe. Zum einen ist das Entity Framework relativ einfach zu bedienen. Die meisten Datenbankoperationen wie Abfragen werden intern vom Framework gesteuert und erfordern keinen Userinput. Lediglich die Erstellung eines Datenbank-Kontexts („DbContext“) als eigene Klasse und die Interaktion mit diesem über vordefinierte Methoden wie „Add“, „Remove“ oder „SaveChanges“ ist von Nöten. Auch hier zeigt sich, dass sich das Framework gerade für das objektorientierte Programmieren eignet. Der zweite Grund für die Entscheidung war, dass es von Microsoft entwickelt wurde und lange Zeit Teil des .NET-Frameworks war, was vor dem Hintergrund, dass unsere Projektarbeit mit C# programmiert wurde, eine gute Kompatibilität nahelegte.
Wie bereits erwähnt, muss zunächst eine eigene Klasse vom Typ „DbContext“ erstellt werden, die System.Data.Entity importiert und ein DbSet enthält, für das eine get – und set-Methode definiert werden. Der DbContext stellt die Verbindung zur Datenbank her und bietet Methoden zum Ändern oder Speichern der Datenbank. Den Kontext nannten wir im Programm einfach „EmployeeContext“ und gaben ihm ein DbSet namens „Employees“, das die namensgebenden Entität „Employee“ beinhaltet. „Employee“ ist eine weitere Klasse, die man als Modell-Klasse bezeichnen kann, da sie im Grunde die Modell-Entität der Datenbank darstellt.
Ein Mitarbeiter hat Eigenschaften wie Namen, Adresse oder eine Telefonnummer, die am Anfang des Projekts in der Aufgabenstellung definiert wurden. Diese Eigenschaften lassen sich sehr einfach als Attribute der Employee-Klasse darstellen. Dazu werden sie einfach als public-Elemente in der Klasse angelegt und mit Gettern und Settern versehen. Hier wurde uns empfohlen, die Auto-Property Funktion von C# zu nutzen, mit der die get – und set-Methoden mit einer einfachen Syntax den Attributen („Properties“) hinzugefügt werden können. Da die Zuweisung der Werte zumeist über Textboxen geschieht, sind die Attribute unserer Klasse vom Typ String.[3]
Mit diesen Schritten ist das Datenbank-Modell bereits fertig und kann verwendet werden, worauf im nächsten Punkt der Dokumentation eingegangen werden soll. Anzumerken wäre noch, dass

3 Ein Bild der Datenbankklasse befindet sich im Anhang (S. 17).

Probleme entstehen können, wenn die Modell-Klasse verändert wird, nachdem die Datenbank bereits mit Daten gefüllt wurde.

## 3.4 Methoden

Der Hauptteil des Programmcodes findet sich in der MainWindow.xaml.cs-Datei. Hier werden die Methoden programmiert, die das Programm verwendet. Der Code ist grob in neun OnClick-Methoden und sieben Hilfsmethoden unterteilt. Die OnClick-Methoden werden allesamt über Buttons aufgerufen, die sich auf der GUI befinden. Die Hilfsmethoden dienen wiederum dazu, den Code der OnClick-Methoden zu kürzen und somit die gesamte Struktur des Programms etwas lesbarer zu machen. Im Folgenden soll die Funktionsweise des Programms anhand der Methoden erklärt werden. Ausgewählte Methoden werden zusätzlich näher beleuchtet.
Am Anfang der MainWindow-Klasse werden zunächst zwei Variablen deklariert. Wichtig ist hier die _context-Variable vom Typ EmployeeContext. Über dieses Objekt wird im Grunde die Verbindung zur Datenbank hergestellt, wie im vorherigen Punkt angesprochen, und Methoden zur Interaktion mit der Datenbank bereitgestellt. Die zweite Variable ist ein zu bearbeitender Mitarbeiter, der entweder vom Typ Employee oder null ist, falls kein Mitarbeiter ausgewählt wurde.
Die MainWindow()-Methode initialisiert das Hauptfenster und lädt die Mitarbeiterliste neu.
Die sechs Buttons im linken Bereich der GUI besitzen alle ihre eigene OnClick-Methode. Zum Hinzufügen eines Mitarbeiters wird die OnClick_addNewMember-Methode aufgerufen, die zunächst lediglich die ChangeFormState-Methode aufruft. Diese Methode dient wiederum dazu, die Darstellung des rechten Teils der GUI zu ändern. Hier ist entweder eine Liste von Mitarbeiter zu sehen, was dem Standardwert entspricht, oder ein ausfüllbares Formular zum Hinzufügen oder Bearbeiten eines Mitarbeiters. Die OnClick_addNewMember-Methode macht ersteres Formular sichtbar. Das tatsächliche Hinzufügen eines Mitarbeiters erfolgt über den nun sichtbaren „Hinzufügen“-Button, der die Methode OnClick_addMember aufruft. Diese Methode wurde bereits im Struktogramm-Teil der Dokumentation gezeigt.[4]
Zunächst wird ein neues Employee-Objekt erzeugt und den Attributen Werte aus den verschiedenen Textboxen zugewiesen. Dann wird eine Validierungs-Funktion aufgerufen, die im Falle von Fehlern beim Ausfüllen des Formulars eine Fehlermeldung erzeugt. War die Validierung stattdessen erfolgreich, wird über den _context ein neuer Mitarbeiter der Employees-Tabelle in der Datenbank hinzugefügt, indem die Add-Methode aufgerufen wird. Ebenso wird der Mitarbeiter der Mitarbeiterliste als Element hinzugefügt. Schließlich werden die Datenbank-Änderungen

4 Zusätzlich befindet sich im Anhang ein Bild des tatsächlichen Codes (S. 18).

gespeichert und die ChangeFormState-Funktion zum Ändern der Darstellung zur Mitarbeiterliste aufgerufen. Die Validierung erfolgt über die oben erwähnte Hilfsmethode ValidateEmployeeData. Beim Aufrufen der Methode wird als Argument der erstellte Employee mitgegeben und über eine Reihe von Regex-Bedingungen evaluiert, ob der Rückgabewert wahr oder falsch ist.
Die anderen Buttons im linken Bereich dienen zum Bearbeiten eines Mitarbeiters, zum Löschen von einem oder mehreren Mitarbeitern, die gerade in der Liste ausgewählt sind, sowie zum Importieren und Exportieren von Mitarbeiterlisten. Das Löschen von Mitarbeitern läuft über eine einfache Remove-Interaktion mit der Datenbank, während beim Bearbeiten zunächst die Daten des ausgewählten Mitarbeiters in das Mitarbeiterformular geladen und dann per Aufrufen der OnCLick_saveChanges-Methode gespeichert werden können. Hier wird sich der AddOrUpdate-Methode der Datenbank bedient.
Der Export öffnet zunächst ein Explorer-Fenster, wo der Zielpfad zum Speichern der Datei gesetzt werden kann. Mithilfe von Newtonsoft-Json, einer Erweiterung, die wir installieren mussten, werden die Einträge in der Datenbank in ein json-Format geschrieben und dann als json-Datei im angegebenen Pfad gespeichert. Die Import-Funktion gleicht aktuell mehr einer Dummy-Funktion, die keine Dateien importieren kann, sondern beim Klick auf den Button eine Faker-Erweiterung benutzt, um Testdaten in die Tabelle zu schreiben. Hierbei muss beachtet werden, dass die eingetragenen Daten nicht von der Validierung überprüft werden und mitunter in der Form nicht von einem Benutzer eingetragen werden können. Dies kann beim Bearbeiten von so erstellten Mitarbeitern Probleme verursachen. Der Faker wird zu Testzwecken benutzt, um zu zeigen, dass eine Vielzahl von Daten auf einmal in die Datenbank eingepflegt werden kann. Im Optimalfall würde der Import-Button eine Datei auswählen lassen und die Daten aus dieser Datei in die Datenbank einpflegen. Hierzu lässt sich Newtonsoft-Json nach unserem Wissen ebenfalls verwenden.
Im oberen linken Teil der GUI befindet sich eine Suchleiste, die aus einer Textbox und einem Button besteht. Der Button ruft die OnClick_searchMember-Methode auf. In dieser Methode wird der Inhalt der Textbox in eine Variable gespeichert. Dann wird eine neue Variable foundEmployees deklariert, in der gefundene Mitarbeiter gespeichert werden sollen. Hier wird nun die Datenbank abgefragt und für jeden Eintrag überprüft, ob der in das Suchfeld eingegebene Text in einer beliebigen Spalte zu finden ist. Ist das der Fall, wird der Mitarbeiter in der Datenbank ausgewählt. Hier zeigt sich die Interaktionsmöglichkeit mit der Datenbank über Entity Framework, wo eine Kombination aus einer SQL-Abfrage und C#-Code genutzt wird, um Informationen aus der Datenbank zu erhalten. Nachdem die Abfrage abgeschlossen wurde, wird die Mitarbeiterliste zunächst geleert und dann alle gefundenen Mitarbeiter wieder hinzugefügt. Ferner besitzt die

Suchleiste noch zwei Hilfsmethoden, mit denen ein Placeholder-Text angezeigt und wieder gelöscht werden kann, je nach dem ob die Suchleiste aktiv ist oder nicht. Dies sind die OnLostFocus_fillPlaceholder und GotFocus_emptyBox Methoden.
Eine weitere Hilfsmethode ist die ReloadEmployeeList-Methode, in der die Mitarbeiterliste zunächst geleert wird und dann über die Einträge in der Datenbank iteriert wird, die nach und nach zur Liste hinzugefügt werden.

## 3.5 Probleme und Lösungen

Während der Arbeit am Projekt ließen sich gewisse Probleme nicht vermeiden. Im Folgenden soll eine Auswahl an Problemen gezeigt werden, die aufgetreten sind, und wie sie gelöst wurden.
Eines der Hauptprobleme, mit dem wir immer wieder konfrontiert wurden, war die Arbeit mit Git. Git eignete sich sehr gut dazu, um eine einfache kollaborative Arbeit am Projekt zu ermöglichen. Das Repository wurde sehr früh angelegt und jedes Gruppenmitglied erhielt die Zugänge, um am Projekt arbeiten zu können. Hier gab es anfangs das Problem, dass das Pullen von neuen Änderungen nicht richtig funktionierte. Es gab oft eine Reihe von Merge-Konflikten, die erst beseitigt werden mussten. Dies war bei regulären Projektdateien zwar ein kleineres Problem, jedoch zeigte sich schnell, dass eine Vielzahl von Dateien Merge-Konflikte aufwiesen, die von uns gar nicht bearbeitet wurden. Zunächst lösten wir dieses Problem stumpf, indem wir das Projekt lokal gelöscht, erneut gepullt und dann samt wieder hinzugefügten Änderungen gepusht haben. So konnte zwar das Repository aktualisiert werden, jedoch blieb das Problem mit dem Pullen weiter bestehen. Die Quelle des Problems lag in der IDE, die wir anfangs nutzten.
Zu Beginn des Projekts nutzten wir wie gehabt Visual Studio, da wir im Unterricht ebenfalls damit gearbeitet haben und sich dort auch Projekte mit WPF erstellen lassen. Leider hat Visual Studio die Eigenart, sehr viele automatisch generierte Dateien dem Projekt hinzuzufügen, die alle beim Ändern des Projekts geändert werden. Dieser Umstand war uns anfangs unklar, sodass wir beim Pushen unserer Änderungen in das Git-Repository immer wieder diese Dateien mit hinzugefügt haben. Diese Dateien erzeugten dann die Merge-Konflikte. Nachdem wir dies bemerkt hatten, versuchten wir, das Problem durch Hinzufügen einer .gitignore-Datei zu beheben. Dies funktionierte jedoch nicht zufriedenstellend. Schließlich entschieden wir uns zu einem Wechsel der IDE. Da wir teilweise privat Erfahrung mit JetBrains-Tools hatten, stiegen wir von Visual Studio auf Rider um. Durch Bearbeitung unseres Projekts in Rider wurden nur die wirklich relevanten Dateien geändert und gepusht, wodurch das Pullen letztendlich funktionierte. Ein weiteres Problem zeigte sich beim Umgang mit der Datenbank. Ursprünglich nutzten wir für die Attribute der Employee-Klasse

verschiedene Datentypen, beispielsweise Integer für Zahlenwerte. Nachdem wir uns entschieden, die Datentypen der Attribute auf Strings zu vereinheitlichen, kam das Problem auf, dass das Programm teilweise nicht mehr funktionierte. Der Grund hierfür lag darin, dass die Datenbank bereits Einträge besaß und sich die Datentypen in den Feldern nun von den Datentypen der Entität unterschieden. Wir konnten das Problem lösen, indem wir die Datenbank komplett gewiped haben. Ursprünglich nutzten wir eine Refresh-Methode, um die Mitarbeiterliste neuzuladen. Diese Methode sollte möglichst alle Fälle abdecken, in denen die Liste neugeladen werden müsste, und interagierte direkt mit der Datenbank. Es zeigte sich jedoch schnell, dass die Nutzung der Methode sehr unperformant war, weshalb wir uns entschieden, sie nicht mehr zu verwenden. Später konnten wir eine Refresh-Methode implementieren, die ausschließlich mit der Liste interagiert und so viel performanter ist.
Zusätzlich nutzten wir teils unterschiedliche Betriebssysteme auf den Laptops. Dies war insofern ein Problem, dass WPF unter Linux Systemen nicht lief, weshalb nicht jedes Gruppenmitglied im Unterricht am eigenen Laptop am Programmcode arbeiten konnte und wir uns abwechseln mussten.

# 4. Diskussion

Es sollen nun einige Punkte angesprochen werden, die Raum für eine andere Herangehensweise bieten als die, für die wir uns entschieden haben, oder die verbessert werden können. Der erste Punkt ist die Nutzung von Git als Werkzeug für die Gruppenarbeit. Die Entscheidung für Git fiel, da GitHub eine Plattform ist, die sehr populär ist und auch in professionellen Kreisen genutzt wird. Sofern das Projekt und damit das Repository korrekt angelegt wird, können diejenigen, die an dem Projekt arbeiten, leicht auf den aktuellen Code zugreifen und Änderungen vornehmen. Dies erlaubt ein schnelles und effizientes Arbeiten. Hätten wir alternativ nur an einem PC oder Laptop gearbeitet, würde nur eine Person programmieren und die anderen zuschauen. Daher war die Möglichkeit des Teilens des Projekts wichtig. Eine weitere Möglichkeit wäre die Nutzung eines Exchanges wie Nextcloud gewesen, was jedoch schwieriger gewesen wäre, gerade was das Mergen von Änderungen mehrerer Personen an der gleichen Datei angeht. Auch da bietet Git ein einfaches Feature, um sich um Konflikte zu kümmern.
Der zweite Punkte wäre die Entscheidung für den objektorientierten Ansatz. Ein ausschlaggebendes Argument für diese Entscheidung waren Vorerfahrungen der Gruppenmitglieder. Generell bot sich OOP für die Aufgabenstellung an, da sich ein Mitarbeiter sehr leicht als Klasse mit Attributen modellieren lässt. Dadurch erschien das Konzept auch weniger abstrakt. Die Alternative hierzu wäre

gewesen, die Daten einfach als Elemente in einer Mitarbeiterliste in Form eines Arrays zu speichern, eventuell mit assoziativen Arrays innerhalb des Arrays. Hier spielte aber auch die Entscheidung zur Nutzung einer Datenbank eine Rolle; gerade im Hinblick auf Entity Framework, wo ohnehin schon Modell-Klassen erstellt werden. Dies wäre mit einer Array-basierten Speicherung der Mitarbeiterdaten schwieriger geworden. Anstelle der Datenbank hätte man die Daten natürlich komplett per Import – und Export-Funktion von Textdateien persistieren können, aber da Entity Framework wie bereits in Punkt 3.3 erwähnt leicht zu bedienen ist, erschien uns dies ebenfalls als unnötig kompliziert.
Schließlich bieten die Programmfeatures auch die Möglichkeit zur Verbesserung. Im Falle unserer Validierung könnten die Regex-Bedingungen klar verbessert werden, indem stärker einschränkende Patterns verwendet werden. Da uns die Erfahrung mit Regex an für sich fehlt und manche Patterns, die wir online gefunden haben, nicht korrekt funktionierten und recht komplex waren, entschieden wir uns für eine vereinfachte Variante. Es stellt sich auch die Frage, inwiefern eine Validierung ohnehin notwendig sein muss, da es letztlich am Nutzer liegt, wie er oder sie die Daten aufbereiten möchte. Es lassen sich auch Zusatzfunktionen einbauen, wie zum Beispiel die Möglichkeit, die Mitarbeiterliste nach Spalten zu sortieren, was ursprünglich geplant war, sich jedoch beim Versuch der Implementierung komplexer erwies als erwartet.

# 5. Fazit

Die Projektarbeit ermöglichte uns, im Rahmen der Programmierung mit C# unsere erworbenen Kenntnisse zu vertiefen. So konnten wir Konzepte aus dem Unterricht, angefangen von der Erstellung eines GUI-basierten Programms, bis hin zur Nutzung von Kontrollstrukturen und Schleifen, im Programmcode umsetzen. Gleichzeitig bot uns die Arbeit aber auch die Möglichkeit, neue Konzepte auszuprobieren und umzusetzen, wie das objektorientierte Programmieren, die Erstellung der GUI per Windows Presentation Forms, oder die Nutzung einer Datenbank zum Speichern der Mitarbeiterdaten. Ebenfalls diente die Projektarbeit als Möglichkeit des Arbeitens nach einem vorher definierten Konzept mit klaren Rahmenbedingungen und einer klaren Aufgabenstellung. Zudem war die Auslegung des Projekts als Gruppenarbeit eine Möglichkeit, nicht nur alleine, sondern mit anderen Personen an einer Aufgabe zu arbeiten und zeigte so auf, wie man koordiniert an einer IT-Aufgabe arbeiten soll und wie die Teilaufgaben von den einzelnen Gruppenmitgliedern umzusetzen sind. Gleichzeitig bot das Projekt verschiedene Herausforderungen, mit denen man, gewollt oder ungewollt, konfrontiert wurde, wie die oben

erwähnten Probleme mit Git, dem Zeitmangement oder schlichtweg neueren Themen wie der Umgang mit dem Entity Framework. In Anbetracht dieser Punkte kann geschlussfolgert werden, dass das Projekt in erfolgreicher Weise nicht nur zur Vertiefung des bisher Gelernten, sondern auch zum Erlernen neuer Kenntnisse für alle Gruppenmitglieder beigetragen hat.

# 6. Anhang und Quellenverzeichnis

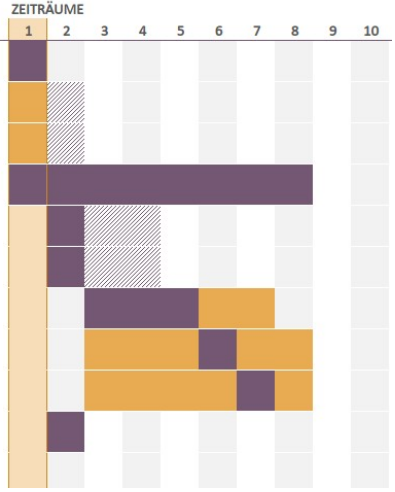
## Anhang

**Link zum GitHub Repository:**

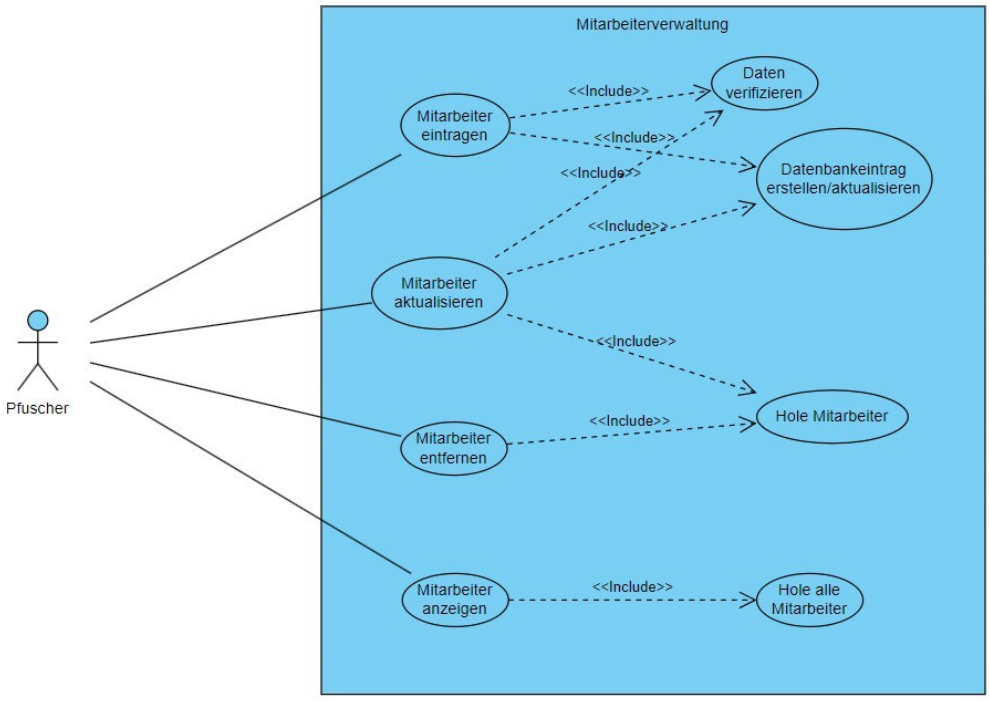
https://github.com/Tenryuubito/css-bts-administration

**Gantt-Diagramm (S. 1)**

| AKTIVITÄT | START DES PLANS | DAUER DES PLANS | TATSÄCHLICHER START | TATSÄCHLICHE DAUER | ABGESCHLOSSEN |
|---|---|---|---|---|---|
| Erstellung Projektplan Benutzeroberfläche (GUI) | 1 | 1 | 1 | 1 | 100% |
| Erstellung Use-Cases | 2 | 1 | 1 | 1 | 100% |
| Dokumentieren Erstellung | 2 | 1 | 1 | 1 | 100% |
| Datenbankschnitstelle Modellieren der | 1 | 8 | 1 | 8 | 100% |
| Datenbankmodelle | 2 | 3 | 2 | 1 | 100% |
| Erstellen der Logik | 2 | 3 | 2 | 1 | 100% |
| Testen Verbesserungen | 3 | 3 | 3 | 5 | 100% |
| durchführen Struktogramme | 6 | 1 | 3 | 6 | 100% |
| erstellen | 7 | 1 | 3 | 6 | 100% |
| | 2 | 1 | 2 | 1 | 100% |




**Use-Case-Diagramm (S. 4):**



## Struktogramme (S. 5)

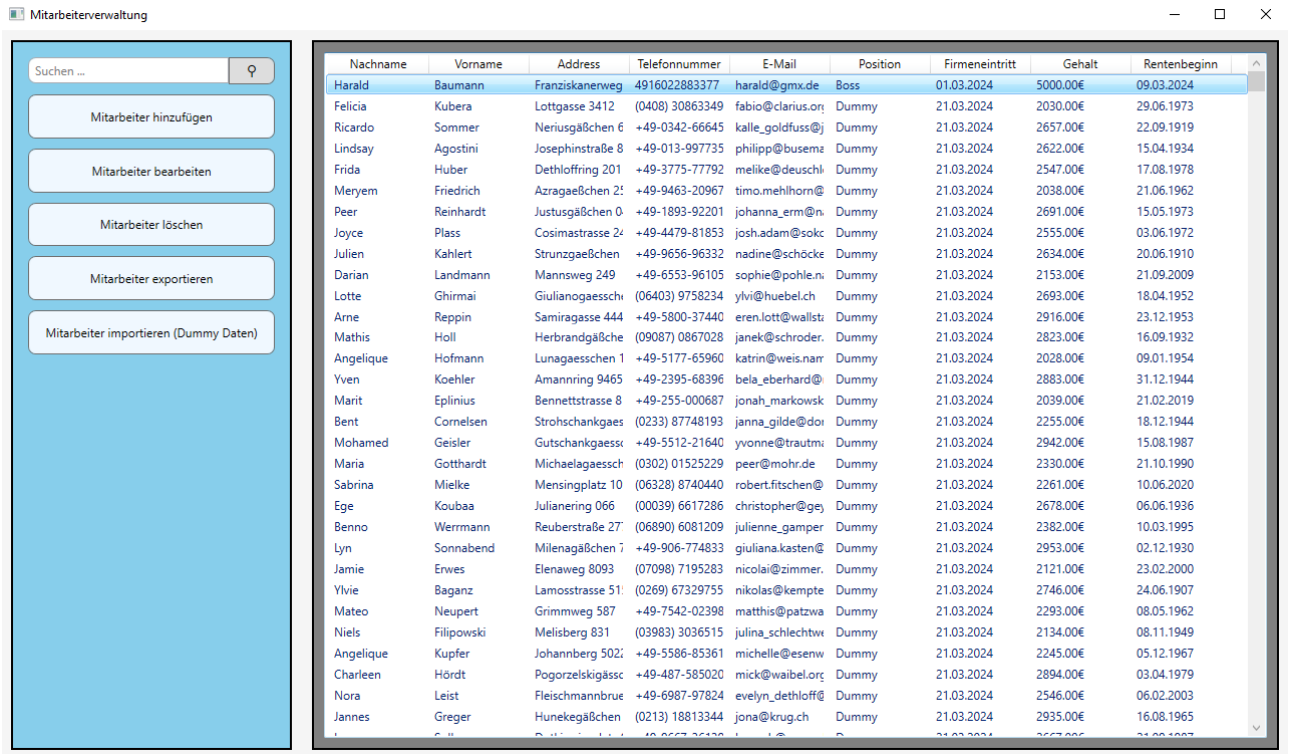
**OnClick_addMember**

| |
|---|
| Neue Instanz von Employee erzeugen **und** in employee speichern. Inhalte der Textfelder als Objekteigenschaften speichern. |
| (Aufruf der Methode ValidateEmployeeData(ref employee) gibt nicht wahr zurück)<br>T / F |
| Erzeuge eine MessageBox mit dem Inhalt "Validation failed!" / ∅ |
| **return** |
| employee zum Datenbank-Kontext Employees hinzufügen |
| employee als Element zum EployeeListView hinzufügen |
| Änderungen im Kontext speichern |
| Aufruf der Methode ChangeFormState(FormState.ListEmployees) |

**ChangeFormState(FormState formState)**

| Aufruf der Methode ClearForm() | | |
|---|---|---|
| (formState) | | |
| FormState.ListEmployees | FormState.AddNewEmployee | FormState.EditExistingEmployee |
| Visbility des EmployeeListViewContainers auf Visbility.Visible stellen | Visbility des EmployeeListViewContainer auf Visbility.Collapsed stellen | Visbility des EmployeeListViewContainer auf Visbility.Collapsed stellen |
| Visbility der EmployeeForm auf Visbility.Collapsed stellen | Visbility der EmployeeForm auf Visbility.Visible stellen | Visbility der EmployeeForm auf Visbility.Visible stellen |
| | Visbility des AddMemberButton auf Visbility.Visible stellen | Visbility des AddMemberButton auf Visbility.Hidden stellen |
| | Visbility des SaveChangesButton auf Visbility.Hidden stellen | Visbility des SaveChangesButton auf Visbility.Visible stellen |

## GUI



## GUI MainWindow.xaml Ausschnitt

```
<Border Background="Gray " Name="EmployeeListViewContainer" Visibility="Visible" Grid.Column="1" BorderThickness="2" BorderBrush="Black " Margin="10" Padding="10">
    <ListView Name="EmployeeListView">

        <ListView.View>

            <GridView AllowsColumnReorder="true" ColumnHeaderToolTip="Employee Information">

                <GridViewColumn DisplayMemberBinding="{Binding Path=FirstName}" Header="Nachname" Width="100"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=LastName}" Header="Vorname" Width="100"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=Address}" Header="Address" Width="100"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=PhoneNumber}" Header="Telefonnummer" Width="100"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=Email}" Header="E-Mail" Width="100"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=Position}" Header="Position" Width="100"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=CompanyEntry}" Header="Firmeneintritt" Width="100"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=Salary}" Header="Gehalt" Width="100"/>
                <GridViewColumn DisplayMemberBinding="{Binding Path=PensionStart}" Header="Rentenbeginn" Width="100"/>
            </GridView>

        </ListView.View>
    </ListView>
</Border>
```

## Datenbank-Modellklasse

```
namespace css_bts_administration
{
    14 usages  smara +1
    class Employee
    {
        public int Id { get; set; }
        6 usages
        public string FirstName { get; set; }
        6 usages
        public string LastName { get; set; }
        6 usages
        public string Address { get; set; }
        5 usages
        public string PhoneNumber { get; set; }
        6 usages
        public string Email { get; set; }
        6 usages
        public string Position { get; set; }
        6 usages
        public string CompanyEntry { get; set; }
        6 usages
        public string Salary { get; set; }
        5 usages
        public string PensionStart { get; set; }
    }
}
```

**OnClick_addMember Methode**

```
private void OnClick_addMember(object sender, RoutedEventArgs e)
{
    Employee employee = new Employee()
    {
        FirstName = InputFirstName.Text,
        LastName = InputLastName.Text,
        CompanyEntry = InputCompanyEntry.Text,
        Address = InputAddress.Text,
        Email = InputEmail.Text,
        Position = InputPosition.Text,
        Salary = InputSalary.Text,
        PensionStart = InputPensionStart.Text,
        PhoneNumber = InputPhoneNumber.Text
    };

    if (!ValidateEmployeeData(ref employee))
    {
        MessageBox.Show( messageBoxText: "Validierung fehlgeschlagen!");
        return;
    }

    _context.Employees.Add(employee);
    EmployeeListView.Items.Add(employee);

    _context.SaveChangesAsync();

    ChangeFormState(FormState.ListEmployees);
}
```

## Quellen


Working with DbContext

https://learn.microsoft.com/en-us/ef/ef6/fundamentals/working-with-dbcontext

Databinding with WPF

https://learn.microsoft.com/en-us/ef/ef6/fundamentals/databinding/wpf

.NET regular expresssions

https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions

Regular expression syntax cheat sheet

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet

Querying and Finding Entities

https://learn.microsoft.com/en-us/ef/ef6/querying/

Newtonsoft JSON

https://www.newtonsoft.com/json

Ignoring files (Git)

https://docs.github.com/en/get-started/getting-started-with-git/ignoring-files

Structorizer

https://structorizer.fisch.lu/

Easy generation of fake/dummy data in C# with Faker.NET

https://blog.elmah.io/easy-generation-of-fake-dummy-data-in-c-with-faker-net/