MITARBEITERVERWALTUNG
PROJEKTBERICHT

Kurs: Software Engineering

Zeitraum: Sommersemester 2021
Betreuer: Prof. Dr. Petrlic

Teammitglieder:

Name:	**Matrikelnummer:**	**E-Mail-Adresse:**
Manuel Röhrer	3291284	roehrerma75432@th-nuernberg.de
Christoph Engelhardt	3581544	engelhardtch84254@th-nuernberg.de
Petra Grüner	2635718	gruenerpe60472@th-nuernberg.de
Tobias Kaiser	3587814	kaiserto84660@th-nuernberg.de
Samuel Borshch	3585670	borshchsa84469@th-nuernberg.de

Inhaltsverzeichnis

1. Projektziel

Das Verwalten von Mitarbeitern kann sich auch bereits in kleineren Unternehmen als Herausforderung erweisen. Es müssen verschiedenste Daten von allen Mitarbeitern so gespeichert werden, dass sie nachträglich leicht einsehbar und wenn nötig auch bearbeitbar sind. Dabei gibt es einen wichtigen Aspekt des Datenschutzes. Es liegt sowohl im Interesse der Mitarbeiter als auch des Unternehmens, dass die Daten sicher verwahrt werden. Als Team haben wir es uns deswegen zur Aufgabe gemacht, ein Programm zu entwickeln, welches Daten von Mitarbeitern in einer Online-Datenbank speichert und in einer grafischen Desktop Nutzeroberfläche anschaulich darstellen kann. Dabei gibt es für jeden Nutzer eine Anmeldung mit einem Benutzernamen und einem Passwort, wodurch eine interne Rechteverwaltung gewährleistet wird. Hiermit wird dafür gesorgt, dass jeder nur auf die Daten zugreifen kann, die ihn betreffen.

2. Aufgabenbereiche jedes Teammitglieds

Am Anfang des Projekts wurden die Stärken und Schwächen der jeweiligen Mitglieder besprochen. So konnte sichergestellt werden, dass sich jeder in der Gruppenarbeit voll einbringen kann. Das Team wurde in zwei Gruppen aufgeteilt, weil wir feststellen konnten, dass zwei bereits Erfahrung mit SQL hatten. So entstand die erste Gruppe, die sich mit der Datenbank inklusive der Verknüpfung zur Oberfläche beschäftigen sollte. Die zweite Gruppe programmierte an der Benutzeroberfläche des Mitarbeiterverwaltungsprogramms. Das Team Datenbank bestand aus Petra Grüner und Tobias Kaiser, während sich das Team Oberfläche aus Samuel Borshch, Christoph Engelhardt und Manuel Röhrer zusammensetzte.

In den Sprint Backlogs (im Anhang), welche wir zu Beginn jedes Sprints mit Hilfe unseres Product Backlogs neu anfertigten, ist zu erkennen, welche Aufgaben welches Teammitglied bearbeiten sollte. Da die meisten von uns weder in SQL noch in WPF bereits erfahren waren, mussten wir uns erstmal mit dem Thema richtig befassen, bevor es die Arbeit losging. Wie wir schnell erkannten, war eine strikte Einteilung der Gruppen nicht die optimale Lösung, denn so kam es oft dazu, dass die eine Gruppe auf die anderen warten musste oder andersherum. Im Laufe des Projekts arbeitete deswegen auch öfters jemand vom Datenbank-Team an der Oberfläche und andersherum.

3. Anforderungsliste

Anforderungen im Product Backlog

Die Liste der Anforderungen im Projekt lässt sich grob aufteilen. Diese entsprechen den einzelnen Ansichten des User Interfaces und den damit verbundenen Funktionalitäten. Genauere Beschreibungen der Anforderungen befinden sich im Anhang im Product Backlog.

Login:

- Loginbereich mit Benutzername und Passwort
- Ein extra Fenster

Dashboard:

- Übersicht der verschiedenen möglichen Seiten
- Logoutbutton

Adressbuchseite:

- Liste aller eingetragenen Mitarbeiter
- Möglichkeit die Liste zu filtern
- Sämtliche Daten einsehbar mit dem richtigen Recht
- Verschiedene Daten ändern zu können mit der richtigen Berechtigung (Private Daten und geschäftliche Daten)

Einstellungen:

- Möglichkeit das eigene Passwort zu ändern
- Eigene Daten, die Firma von einem hat einsehen zu können (nicht geschafft)

Benutzer erstellen:

- Anlegen eines neuen Accounts
- Account Rechte zuteilen

Mitarbeiter erstellen:

- Erstellen eines Mitarbeiters
- Mitarbeiter soll nach hinzufügen in der Adressliste auftauchen

Urlaubsplaner (nicht im Programm integriert):

- Möglichkeit Urlaub einzutragen
- Urlaub für alle sichtbar, damit Überblick vorhanden ist, wann wer Urlaub nimmt, um sich gegenseitig abzusprechen (Der Urlaubsplaner wurde leider zeitlich nicht mehr geschafft)

Nachträgliche zusätzliche Sicherheits-Anforderungen

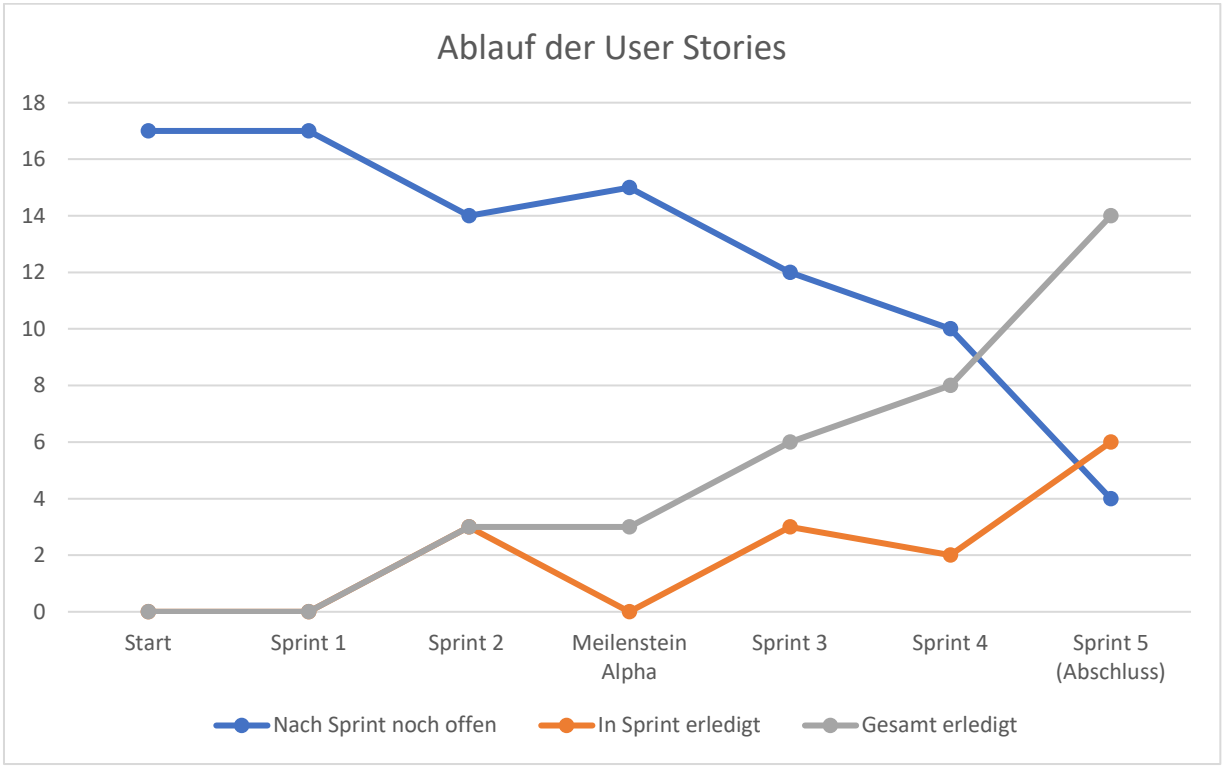
Im Laufe des Projekts bekam unser Team am 19.05.2021 weitere Anforderungen von einem zweiten Product Owner. Wir sollten zusätzliche Daten von einem Mitarbeiter speichern können, diese wären: Einstellungsdatum, Kündigungsfrist, Abmahnungen, Gehalt, Bonuszahlungen und Kommentarfeld.

Priorisierung der User Stories

Bei der Priorisierung der User Stories musste auf zwei manchmal gegensätzliche Faktoren geachtet werden. Zum einen sollten die unbedingt nötigen Grundfunktionalitäten zuerst fertig gestellt werden, um schnell einen funktionierenden Prototyp zu haben. Andererseits war aber die Anzahl an Personen begrenzt, die gleichzeitig an demselben Feature arbeiten konnten, da sonst Probleme bei der Zusammenführung entstanden wären.

4. Soll Ist Auswertung

Es wurden alle zur grundlegenden Funktionalität dringend nötigen Anforderungen von uns umgesetzt. Im unteren Abbild ist die Fertigstellung der User Stories im zeitlichen Verlauf dargestellt. Unsere Sprints hatten dabei normalerweise eine Dauer von zwei Wochen. Lediglich als das Projekt langsam Richtung Ende ging, haben wir unsere Sprints etwas verkürzen müssen, um mehr Punkt erledigen zu können. Zum Alpha Meilenstein am 19.Mai hatten wir eine lauffähige Demo implementiert, welche dem Dozenten vorgeführt werden konnte. Beim Meilenstein Alpha kam vom Masterstudenten Julian Hof, der die Rolle eines weiteren Product Owners spielte, eine weitere zusätzliche User Story hinzu, die wir beim Product Backlog aufnehmen mussten. Deswegen hat sich in untenstehender Grafik die Anzahl der offenen Storys beim Meilenstein Alpha um eine erhöht. Dem Prototyp fehlte noch die ein oder andere Funktionalität, doch diese wurde bis zum Abschluss Meilenstein noch fertig gestellt, wie man auch in der Grafik gut erkennen kann.



5. Quellcode-Konventionen

Wir verwendeten für Klassennamen, Methoden, Events, Commands und öffentliche Eigenschaften die Pascal Case Konvention. Für Variablen und privaten Eigenschaften wurde hier die Camel Case genutzt. Außerdem wurde eine Einrückung von vier Leerzeichen verwendet. Die öffnenden Klammern für Anweisungsblöcke wurden jeweils am Ende der vorherigen Zeile gesetzt und die schließenden Klammern am Ende des Blocks jeweils in einer einzelnen Zeile. Es wurde auch Wert auf sprechende Bezeichner gelegt, sodass der Quellcode gut lesbar ist.

Zusätzlich wurde auch dort der Code kommentiert, wo es in unseren Augen nötig war. Sprich es wurden nur sinnvolle Kommentare geschrieben. Bei zu vielen Kommentaren würde dies die Wartbarkeit verschlechtern, da bei Quellcodeänderungen die zugehörigen Kommentare oft nicht mit angepasst werden. Somit wird fehlerhafte Information zwischen den Zeilen vorgebeugt.

Diese Quellcode-Konventionen halfen uns dabei den Quellcode einheitlich zu halten und somit die Arbeit im Team zu erleichtern.

6. Tests

Bei unserem Projekt wurden Unit und Integration Tests durchgeführt. Dies war nötig, um eine hohe Softwarequalität sicherzustellen und während der zukünftigen Entwicklung neue Bugs zu vermeiden. Die Vorteile bei diesen Tests sind, dass man während der Entwicklung stets ein schnelles Feedback nach Änderungen am Code bekommt. Wenn die bestehenden Tests erfolgreich waren, konnte man sich sicher sein, dass keine bestehenden Funktionen nicht mehr wie erwartet funktionieren.

Die Testphasen wurden während oder nach dem Programmieren durchgeführt. Unser neuer Code wurde erst dann zur Main Branch zusammengefügt, wenn die geforderten Tests erfolgreich waren. Durch dieses Vorgehen konnte die angestrebte Testabdeckung erreicht werden.

Unit Tests wurden bei uns durchgeführt, um die Codelogik auf Funktionalität zu überprüfen und ob der Code auch angemessen ist. Es verbessert die Lesbarkeit des Programmcodes, somit war es leichter neuen Code zu implementieren. Die Integration Tests wurden hingegen genutzt, um sicherzustellen, dass die Methoden des Repository Layers korrekt mit der Datenbank interagieren, aber auch die korrekte Kommunikation zwischen den verschiedenen Komponenten einzuschließen.

Beim Integrieren eines Branches in die Main Branch hat uns GitHub unter die Arme gegriffen. Denn GitHub überprüft automatisch, ob es Konflikte zwischen den Branches gibt. Wenn es keine gab, konnte man erst dann alles mergen – dies konnte auch nur geschehen, wenn alle erforderlichen Tests erfolgreich durchgeführt wurden. Wenn GitHub grünes Licht zum Mergen gab, gab es immer nochmal mindestens eine/n aus unserem Team, der/die grob über den Code schaute.

Bei unseren gemeinsamen Meetings wurden auch die neuen hinzugefügten Funktionen vor allen Mitgliedern gezeigt und bewertet. Nachdem dies geschehen worden ist, wurde die User Story als „done“ markiert.

Durch die Durchführung der Tests war es möglich eine qualitative und möglichst fehlerfreie Software zu entwickeln. Zudem wurden vor allem im Bereich des Erstellens von Unit- und Integrationstests viele neue Erfahrungen gesammelt.

7. Überführung in die Praxis

Da wir mit Visual Studio gearbeitet haben und mit dem Vorgehensmodell MVVM bei WPF, ist die Wartung der Software kein Problem. Falls die Software an sich einen Fehler haben sollte, könnte man dies mit einem Update leicht beheben. Hierfür könnte noch eine „Updater-Funktionalität“ implementiert werden. Die gesamten Daten sind auf der Datenbank verschlüsselt gesichert, somit gehen keine Daten während des Updates verloren. Der Benutzer bräuchte einen eigenen Server, auf dem die Datenbank des Programms liegt. Es wird empfohlen die Datenbank 24/7 laufen zu lassen. Natürlich könnte man den Server auch erst dann starten, wenn das Programm gebraucht wird. Das Projekt ist in unseren Augen sehr skalierbar, weil das Programm beliebig erweiterbar ist.

8. Zusätzliche Aspekte in der Praxis

Aktuell wäre das Programm für eine kleine bis mittelgroße Firma (ca. 50 Personen) ausgelegt. Für größere Unternehmen wäre es eher nicht geeignet. Hier ist das Problem, dass es zu Synchronisierungsfehlern kommen kann, wenn zu viele Personen auf einmal Daten ändern. Das heißt daher, dass die Datenbank nicht mit so einem großen Datenfluss arbeiten kann.

Da wir Secure Scrum verwendet haben, ist die grundsätzliche IT-Sicherheit von uns gegeben. Empfohlen wird, dass der Server lokal in der Firma eingerichtet ist. Dazu kommt noch, dass der Server sicher vor DDoS-Attacken (= Distributed Denial of Service – Attacken) geschützt werden soll, da sonst eine Benutzung der Software nicht mehr möglich wäre.

9. Projektorganisation

9.1 Unser Vorgehensmodell

Wir als Team haben uns entschieden ein agiles Vorgehensmodell für unser Projekt zu nutzen, da diese, anders als klassische, offen für Änderungen in der Anforderung ist. Bei klassischen Modellen ist es schwieriger auf mögliche Änderungen zu reagieren, die im Laufe der Zeit auf ein Entwicklerteam zukommen könnten. Die klassischen Vorgehensmodelle wie die Wasserfall-Methode oder die V-Methode wären außerdem für unser Team ungeeignet gewesen, da es durch die Berufstätigkeit mehrerer Teammitglieder zu unterschiedlichen Zeitverfügbarkeiten gekommen ist und dies die Planung erschweren würde.

Geplant war das agile Vorgehensmodell Scrum zu wählen. Unser Team war eins von 6 ausgewählten Teams, die ein Teil der Masterarbeit von Julian Hof waren, deswegen wurden wir gebeten eins von drei Scrum-Varianten zu wählen. Zur Auswahl standen das klassische Scrum, das VAHTI-Scrum und Secure-Scrum. Wir entschieden uns für Secure-Scrum, da dies uns als Team am meisten angesprochen hat.

Wie im Namen schon erkennbar, dreht sich Secure-Scrum auch stark um die Sicherheit der Software. Das Besondere daran ist, dass es Entwickler hilft sichere Software zu entwickeln, auch wenn diese keine Experten im Thema Security sind. In unserem Team kannte sich keiner richtig aus mit Security und wir waren erstaunt, wie wichtig Sicherheit eigentlich ist. Software kann viele Sicherheitslücken besitzen und mit dem ausgewählten Vorgehensmodell wirft man erstrecht ein Auge auf diese Lücken.

9.2 Verwendete Programme und Tools

Im nachfolgenden gibt es eine Übersicht, welche Tools, Programme und Technologien zu der erfolgreichen Umsetzung unseres Projektes nötig waren und auf welche Weise sie zum Erfolg beigetragen haben:

- **Microsoft Visual Studio Community 2019:**
  Die IDE wurde von uns zum Implementieren unserer Software verwendet. Wir haben in C# programmiert und die Erweiterung WPF mit dem MVVM Modell angewendet. Bei dem äußerlichen Design benutzen wir das ModernWPFui-NuGet-Paket von Yimeng Wu, ein spezielles Paket für die WPF-Anwendung. Des Weiteren verwendeten wir das NuGet-Paket Data.SqlClient von Microsoft.

- **Microsoft SQL-Server Management Studio (SSMS):**
  Die Anwendung Microsoft SQL-Server Management Studio wurde verwendet, um die Datenbank zu erstellen und zu verwalten.

- **Microsoft SQL-Server 2019:**
  Die Datenbank wurde mit Microsoft SQL-Server 2019 realisiert. Durch die Verwendung von Microsoft Visual Studio Community 2019, war es uns möglich, dass wir ohne Probleme die integrierte Schnittstelle zur Datenbank verwenden konnten.

- **GitHub:**
  Zur Verwaltung des Quellcodes wurde die Plattform GitHub eingesetzt, welche einen Git Server bietet, um die verteilte Entwicklung zu stützen. Darüber hinaus bietet GitHub aber noch mehr als die reine Quellcode Verwaltung. Beispielsweise können über GitHub auch Code Reviews durchgeführt werden, was im Laufe des Projektes ein fester Bestandteil unseres Vorgehens war.

- **Microsoft Excel:**
  Mit Excel wurde sowohl unser Product Backlog erstellt als auch die einzelnen Sprint Backlogs. Mit dieser Software ist es sehr einfach Tabellen anzufertigen und den Überblick nicht zu verlieren

- **Microsoft Word:**
  Mit Word wurde der Projektbericht geschrieben und Notizen angefertigt.

- **Microsoft Teams:**
  Während dem Online-Semester benutzten wir Microsoft Teams. Dies ermöglichte uns untereinander Besprechungen abzuhalten und Dateien auszutauschen. Jeder von uns ist im Besitzt von einem Konto, da dies von der Technischen Hochschule bereitgestellt wird.

- **WhatsApp:**
  WhatsApp nutzten wir damit wir uns kurzfristig austauschen können. Die Kommunikation auf WhatsApp ist schneller und direkter als auf Teams, weil jeder die Nachrichten sofort auf dem mobilen Endgerät bekommt.

- **GlobalProtect (VPN):**
  Um Zugriff auf die Datenbank zu bekommen, wurde der VPN GlobalProtect verwendet.

9.3 Verwendeter Server für die Datenbank

Das Programm verwendet eine Datenbank zur Speicherung der Mitarbeiterdaten, weswegen wir einen Server von der Fakultät Informatik beim Dipl.-Ing. (FH) R. Fischer angefragt haben. Somit hatte jeder Zugriff auf die Datenbank aus unserem Team.
Der große Vorteil hierbei war, dass niemand von uns die Datenbank lokal speichern musste. Auf die Datenbank konnte über den VPN der Technischen Hochschule Nürnberg zugegriffen werden.

9.4 Struktur im Projekt mit MVVM

In unserem Projekt verwendeten wir das MVVM-Modell (=Model-View-ViewModel). Dies dient zur Trennung von Darstellung und Logik der Benutzerschnittstelle. MVVM zielt auf moderne UI-Plattformen ab, da ein Datenbindungsmechanismus erforderlich ist. Wir verwendeten die UI-Plattform Windows Presentation Foundation (WPF), weswegen wir dieses Modell gewählt haben. Es erleichtert die Übersicht durch eine klare Ordnerstruktur und es verbessert die Testbarkeit, da die “ViewModel” die UI-Logiken enthalten und unabhängig von der “View”(=Oberfläche) instanziiert werden können.

10. UML-Diagramme

Im Anhang befinden sich die zum Projekt zugehörigen UML-Diagramme. Das Klassendiagramm, die Zustands- und die Sequenzdiagramme.

10.1 Klassendiagramm

Im Klassendiagramm sind die einzelnen Klassen und deren Beziehungen zueinander dargestellt. Die Strukturierung des Diagramms basiert auf dem Entwurfsmuster Model View ViewModel (MVVM).

Die einzelnen Klassen sind jeweils in die Namespaces MAV.Login, MAV.Client, MAV.Base und MAV.Helper gruppiert.

Im MAV.Login befinden sich die Klassen, die für den Login in die Anwendung benötigt werden. MAV.Client beinhaltet alle Klassen die für die Hauptanwendung wichtig sind. Die Namespaces MAV.Base und MAV.Helper enthalten Hilfs- und Basisklassen.

10.2 Zustandsdiagramme

Für das Projekt haben wir mehrere Zustandsdiagramme erstellt.

Diese stellen u.a. folgende Vorgänge dar:

- Mitarbeiter hinzufügen
- Benutzer hinzufügen
- Passwort ändern
- Mitarbeiterdaten anzeigen und bearbeiten
- Login
- Logout
- Impressum anzeigen

10.3 Sequenzdiagramme

Das Verhalten des Programms ist in mehreren Sequenzdiagrammen abgebildet:

- sd Login: Login eines Nutzers
- sd AddUser: Benutzer hinzufügen (nur Admin)
- sd AddEmployee: Mitarbeiter hinzufügen (Admin/HR)
- sd EmployeeEdit: Bearbeiten von Mitarbeiterdaten durch den Admin bzw. HR
- sd EmployeeInfo: Anzeigen von Informationen des einzelnen Mitarbeiters
- sd ChangePassword: Ändern des Passworts eines Nutzers durch den Admin bzw. HR

11. Datenbank

Die Datenbank der Anwendung befindet sich auf einem Server der TH-Nürnberg unter der IP-Adresse 141.75.150.78. Der Server läuft auf einer Distribution von Windows 10.

Die Tabellen für das Programm sind in der Datenbank „dbMAV“ hinterlegt. Dort sind auch die für die Implementierung notwendigen Stored Procedures gespeichert.

Für die graphische Darstellung des Datenbankentwurfs haben wir das Entity-Relationship-Modell (ERM) angewandt. Dort sind alle Tabellen und deren zugehörigen Attribute mit den entsprechenden Datentypen aufgelistet. Die Beziehungen zwischen den Tabellen sind in der Krähenfuß-Notation dargestellt.

12. Voraussetzungen zum Programmtest

Da unser Tool auf den Server der Hochschule zum Speichern und Abrufen der Mitarbeiterdaten angewiesen ist, wird bei der Ausführung eine VPN-Verbindung über GlobalProtect zum Hochschulnetz vorausgesetzt. Zudem muss das Betriebssystem Windows sein, da unser Programm eine reine Windows App ist. Der Zugriff auf den Server erfolgt über die IP-Adresse 141.75.150.78. Für die verschiedenen Zugriffsebenen gibt es eigene Logins und Passwörter. Der Admin kann sich dabei mit dem Nutzernamen: „Admin“ und Passwort „Admin123“ anmelden, der HR-Mitarbeiter mit „user_hr“ und „123456789“ und der normale Mitarbeiter mit „user_normalo“ und „987654321“.

11. Anhänge

Product Backlog

ID	Beschreibung	Sub-Beschreibung	S-Mark	Status	LossValue	Prio
US.01	Als Benutzer möchte ich mich einloggen können	Extra Fenster mit Username und Passwort eingabe	**ST01**	done	HOCH	1
US.02	Ich als Benutzer möchte ein Fenster haben, wovon ich überall drauf zugreifen kann	Eine Übersicht für die verschiedenen Benutzer-Oberflächen		done	NIEDRIG	2
US.03	Es soll eine Liste vorhanden sein, bei der ich alle Mitarbeiter des Unternehmens sehen kann	Mitarbeiter gelistet mit den wichtigen Daten (Abteilung, Telefonnummer, Name usw.)		done	NIEDRIG	3
US.18	Ich als Personalmitarbeiter (oder Admin) möchte die Möglichkeit haben, die Daten eines Mitarbieters zu bearbeiten	Durch Button auf Informationsseite erreichbar	**ST02 und ST03**	done	HOCH	4
US.16	Als Mitarbeiter der Personalabteilung will ich zu Mitarbeitern Personaldaten speichern (Einstellungsdatum, Kündigungsfrist, Abmahnungen, Gehalt, Bonuszahlungen, Kommentarfeld) um diese Daten im System zu verwalten.		**ST02 und ST03**	done	HOCH	5
US.05	Die Kollegen von mir sollen nur meine geschäftlichen Daten einsehen können	Geschäftliche Daten: Name, gesch. Telefonnummer, Abteilung, Beruf, Geburtstag (ohne Jahr)	**ST02**	done	HOCH	6
US.08	Es soll eine Seite geben, auf der man einen Mitarbeiter hinzufügen kann	Extra Seite um einen Mitarbeiter zu erstelllen	**ST02 und ST03**	done	HOCH	7

US.04	Ich als User möchte meine eigenen Daten einsehen können	Geschäftliche und Private Daten einsehbar	**ST02**	nicht möglich	MITTEL	8
US.06	Die Liste soll filterbar sein, damit ich schneller jemanden finden kann	Filterbar durch eingabe des Suchsbegriffs	**ST03**	done	HOCH	9
US.07	Als Admin soll ich die Möglichkeit haben Benutzerkonten hinzufügen	Extra Seite zum Erstellen eines neuen Benutzers	**ST02**	done	HOCH	10
US.10	Wenn ich mich eingeloggt habe, möchte ich das Passwort ändern können	Eine Seite für die eigenen Einstellungen. Altes Passwort für die Bestätigung und 2x das neue Passwort eingeben als Bestätigung	**ST01**	done	HOCH	11
US.11	Als Admin kann ich entscheiden, wer was einsehen oder bearbeiten kann	Vergabe der verschiedenen Rechte möglich (Admin, HR oder normaler Mitarbeiter)	**ST02**	done	HOCH	12
US.12	Eine einfache Übersicht soll vorhanden sein			done	NIEDRIG	13
US.13	Bei Inaktivität soll ich mich automatisch ausloggen	Nach 5 bis 10 Minuten inaktivität ausloggen	**ST04**	done	MITTEL	14
US.14	Wenn ich mich auslogge, kann ich mich gleich wieder einloggen ohne das Programm neu zu starten	Beim einfachen Ausloggen das Loginfenster wieder anzeigen	**ST04**	done	MITTEL	15
US.17	Es soll eine Liste vorhanden sein, bei der ich alle Benutzer zum Anmelden im Programm sehen kann			offen	MITTEL	16
US.09	Wenn jemand sein Passwort vergessen haben sollte, kann ich als Admin das	Admin wählt Person in Liste aus und hat die Möglichkeit das Passwort zu ändern	**ST01**	offen	HOCH	17

	ohne Probleme ändern	ohne das alte zu wissen			
US.15	Als Arbeiter möchte ich sehen wann wer Urlaub hat.	Urlaubsplaner erstellen mit einsicht, wann jemand Urlaub hat.	offen	NIEDRIG	18
ST01	Als User möchte ich einen sicheren Umgang mit Passwörtern	SSL/TLS - Sichere Verbindung - Verschlüsselung in der Datenbank	done		
ST02	Die Daten sollen nicht für jeden einsehbar sein	Zugriffseinschränkung	done		
ST03	Eingabemöglichkeit keine Manipulation möglich	SQL-Injection -> Eingabe überprüfen	done		
ST04	Ein sicheres Ausloggen, damit ich mir keine Sorgen machen muss	Logout Button hinzufügen, damit der Benutzer eine Bestätigung hat, dass er sich sicher ausgeloggt hat	done		

Sprint Backlog 01 von 22.04. bis 06.05.

ID	Beschreibung	Sub-Beschreibung	S-Mark	Bearbeiter	Code Review	Done
US.01 (Login)	Als Benutzer möchte ich mich einloggen können	Loginbereich - Passwort, Benutzername				
US.01_1	Fenster erstellen	Username-Eingabe, Passwort-Eingabe, Extra Fenster, Erkennbar dass es ein Loginbereich ist.	-	Christoph		
US.02 (Client)	Ich als Benutzer möchte ein Fenster haben, wovon ich überall drauf zugreifen kann	Ein Client bei dem man nach dem erfolgreichen Login ankommt				
US.02_1	Statusleiste, Einstellungszahnrad	Ein "Rahmen" für Usercontrols	-	Manuel		
US.02_2	Usercontrol-Möglichkeiten	Usercontrol Schnittstelle. Anzeigen von verschiedenen Seiten möglich	-	Manuel		
Base	Basisklassen Vererbung/Struktur					
Base_01	PropertyChanged	Event Implementierung	-	Tobi		
Helper	Hilfsklassen	Benötigt der Client				
Helper_01	Assemblyloader	Laden von Usercontrols + abspeichern	-	Manuel		
Helper_02	Datenbankprovider	Schnittstelle zur Datenbank	-	Samuel		
Helper_03	RelayCommand	Hilfsklassen um Buttons auszuführen	-	Christoph		
DB	Start vom Aufsetzen					
DB_01	SQL-Server aufsetzen		-	Tobi		
DB_02	Tabellen aufsetzen		-			
DB_03	Tabellen planen		-			
DB_04	Testdaten aufsetzen		-			
Solution	Struktur schaffen					
Solution_01	MAV:Login		-	Christoph		
Solution_02	MAV:Client		-	Tobi		
Solution_03	MAV:Base		-	Tobi		
Solution_04	MAV:Helper		-	Tobi		

Sprint Backlog 02 von 07.05. bis 20.05.

ID	Beschreibung	Sub-Beschreibung	S-Mark	Bearbeiter	Code	Review	Done
US.01 (Login)	Als Benutzer möchte ich mich einloggen können	Loginbereich - Passwort, Benutzername	-				
US.01_1	Fenster erstellen	Username-Eingabe, Passwort-Eingabe, Extra Fenster, Erkennbar dass es ein Loginbereich ist.	-	Christoph			
US.02 (Client)	Ich als Benutzer möchte ein Fenster haben, wovon ich überall drauf zugreifen kann	Ein Client bei dem man nach dem erfolgreichen Login ankommt					
US.02_1	Usercontrol-Möglichkeiten	Usercontrol Schnittstelle. Anzeigen von verschiedenen Seiten möglich	-	Manuel			
Helper	Hilfsklassen	Benötigt der Client	-				
Helper_1	Assemblyloader	Laden von Usercontrols + abspeichern	-	Manuel			
Helper_2	Datenbankprovider	Schnittstelle zur Datenbank	-	Samuel			
Helper_3	RelayCommand	Hilfsklassen um Buttons auszuführen	-	Christoph			
DB	Start vom Aufsetzen		-				
DB_1	SQL-Server aufsetzen		-	Tobi			
DB_2	Tabellen aufsetzen		-	Tobi			
DB_3	Tabellen planen		-	Tobi			
DB_4	Testdaten aufsetzen		-	Tobi			
US.03	Es soll eine Liste vorhanden sein, bei der ich alle Mitarbeiter sehen kann, die auch ein Konto haben	Mitarbeiter gelistet mit den wichtigen Daten (Abteilung, Telefonnummer, Name usw.)	-				
US.03_01	ListView erstellen	Spalten: Abteilung, Name, Vorname, gesch. Telefonnummer, gesch. Handynummer	-	Manuel			
US.04	Ich als User möchte meine eigenen Daten einsehen können	Geschäftliche und Private Daten einsehbar	**ST02**	Samuel			

US.04_1	Sonderrecht auf eigene Daten						
US.05	Die Kollegen von mir sollen nur meine geschäftlichen Daten einsehen können	Geschäftliche Daten: Name, gesch. Telefonnummer, Abteilung, Beruf, Geburtstag (ohne Jahr)	**ST02**	Samuel			
US.05_1	Anzeigemöglichkeiten von Mitarbeitern	Geschäftliche Daten werden angezeigt. Extra Button für besondere Rechte zu privaten Daten	-	Samuel			
US.06	Die Liste soll filterbar sein, damit ich schneller jemanden finden kann		**ST03**				
US.06_01	Filtermöglichkeiten	Mit einem Textfeld	-				
US.12	Eine einfache Übersicht soll vorhanden sein		-				
US.14	Wenn ich mich auslogge, kann ich mich gleich wieder einloggen ohne das Programm neu zu starten	Beim einfachen Ausloggen das Loginfenster wieder anzeigen	**ST04**	Tobi			

Sprint Backlog 03 von 21.05. bis 03.06.

ID	Beschreibung	Sub-Beschreibung	S-Mark	Bearbeiter	Code	Review	Done
Helper	Hilfsklassen	Benötigt der Client	-				
Helper_2	Datenbankprovider	Schnittstelle zur Datenbank	-	Samuel / Petra			
DB	Start vom Aufsetzen		-				
DB_1	SQL-Server aufsetzen		-	Tobi			
DB_2	Tabellen aufsetzen		-	Tobi			
DB_4	Testdaten aufsetzen		-	Tobi			
US.03	Es soll eine Liste vorhanden sein, bei der ich alle Mitarbeiter sehen kann, die auch ein Konto haben	Mitarbeiter gelistet mit den wichtigen Daten (Abteilung, Telefonnummer, Name usw.)	-				
US.03_01	ListView erstellen	Spalten: Abteilung, Name, Vorname, gesch. Telefonnummer, gesch. Handynummer	-	Manuel			
US.04	Ich als User möchte meine eigenen Daten einsehen können	Geschäftliche und Private Daten einsehbar	**ST02**	Samuel			
US.04_1	Sonderrecht auf eigene Daten						
US.05	Die Kollegen von mir sollen nur meine geschäftlichen Daten einsehen können	Geschäftliche Daten: Name, gesch. Telefonnummer, Abteilung, Beruf, Geburtstag (ohne Jahr)	**ST02**	Samuel			
US.05_1	Anzeigemöglichkeiten von Mitarbeitern	Geschäftliche Daten werden angezeigt. Extra Button für besondere Rechte zu privaten Daten	-	Samuel			
US.06	Die Liste soll filterbar sein, damit ich schneller jemanden finden kann		**ST03**				

US.06_01	Filtermöglichkeiten	Mit einem Textfeld, ComboBox	-	Manuel	
US.12	Eine einfache Übersicht soll vorhanden sein		-	Manuel	
US.13	Bei Inaktivität soll ich mich automatisch ausloggen	Nach 5 bis 10 Minuten inaktivität ausloggen	-	Tobi	
US.13_01	Timer hinzufügen		-	Tobi	
US.07	Als Admin soll ich das Recht haben die Benutzerkonten hinzuzufügen	Extra Seite zum Erstellen eines neuen Benutzers	-		
US.07_01	Extra Seite - Benutzer hinzufügen	Hinzufügen von sämtlichen Daten und der Berechtigung	-	Christoph	
US.07_02	Prozedur zum Anlegen des Benutzers in Datenbank	Speichern des neuen Benutzers	-	Tobi	
US.09	Wenn jemand sein Passwort vergessen haben sollte, kann ich als Admin das ohne Probleme ändern	Admin wählt Person in Liste aus und hat die Möglichkeit das Passwort zu ändern ohne das alte zu wissen	-		
US.09_01	Hinzufügen zur Seite zum Bearbeiten		-	Christoph	

Sprint Backlog 04 von 10.06. bis 22.06.

ID	Beschreibung	Sub-Beschreibung	S-Mark	Bearbeiter	Code	Review	Done
Helper	Hilfsklassen	Benötigt der Client	-				
Helper_2	Datenbankprovider	Schnittstelle zur Datenbank	-	Petra			
US.04	Ich als User möchte meine eigenen Daten einsehen können	Geschäftliche und Private Daten einsehbar	ST02	Manuel			
US.04_1	Sonderrecht auf eigene Daten						
US.05	Die Kollegen von mir sollen nur meine geschäftlichen Daten einsehen können	Geschäftliche Daten: Name, gesch. Telefonnummer, Abteilung, Beruf, Geburtstag (ohne Jahr)	ST02	Manuel			
US.05_1	Anzeigemöglichkeiten von Mitarbeitern	Geschäftliche Daten werden angezeigt. Extra Button für besondere Rechte zu privaten Daten	-	Manuel			
US.06	Die Liste soll filterbar sein, damit ich schneller jemanden finden kann		ST03	Tobi			
US.06_1	Datenbank Skript	Skript schreiben für Filter		Tobi			
US.03	Es soll eine Liste vorhanden sein, bei der ich alle Mitarbeiter des Unternehmens sehen kann	Mitarbeiter gelistet mit den wichtigen Daten (Abteilung, Telefonnummer, Name usw.)		Manuel			
US.11	Als Admin möchte ich nicht alles alleine managen.	Verschiedene Rechte mit verschiedenen Möglichkeiten		Christoph			
US.11_1	Rechte vergeben beim erstellen des Benutzers			Christoph			
US.10	Wenn ich mich eingeloggt habe, möchte ich das Passwort ändern können	Eine Seite für die eigenen Einstellungen. Altes Passwort für die Bestätigung und 2x das neue Passwort eingeben als Bestätigung		Christoph			

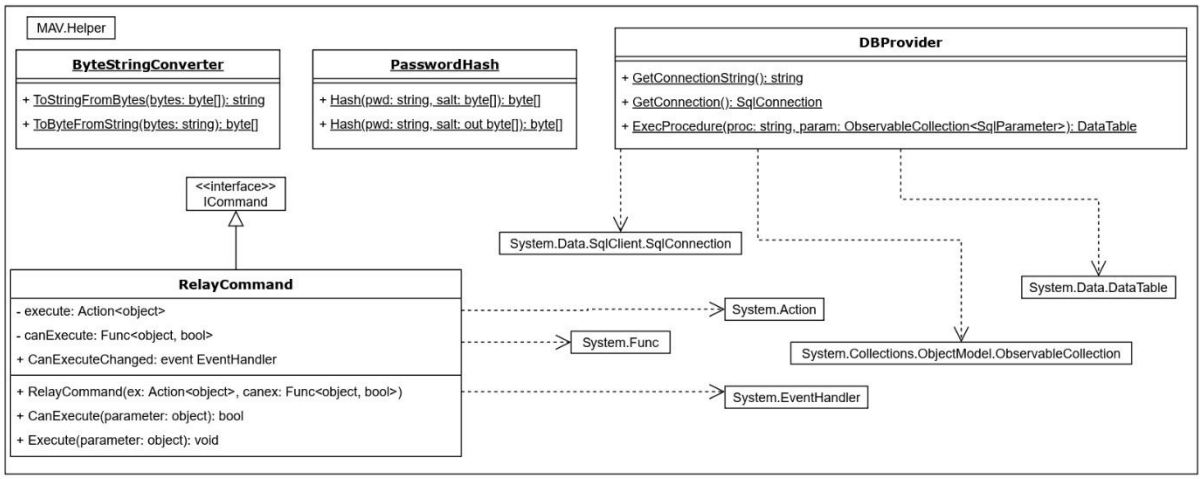
US.10_1	Radiobutton Einstellungen Passwort ändern hinzufügen		Christoph			
US.10_2	Reihenfolge der RadioButtons ändern		Christoph			
US.16	Als Mitarbeiter der Personalabteilung will ich zu Mitarbeitern Personaldaten speichern (Einstellungsdatum, Kündigungsfrist, Abmahnungen, Gehalt, Bonuszahlungen, Kommentarfeld) um diese Daten im System zu verwalten.		Manuel			
US.18	Ich als Personalmitarbeiter (oder Admin) möchte die Möglichkeit haben, die Daten eines Mitarbieters zu bearbeiten	Durch Button auf Informationsseite erreichbar	Manuel			
EXTRA	Dokumentation		Samuel			

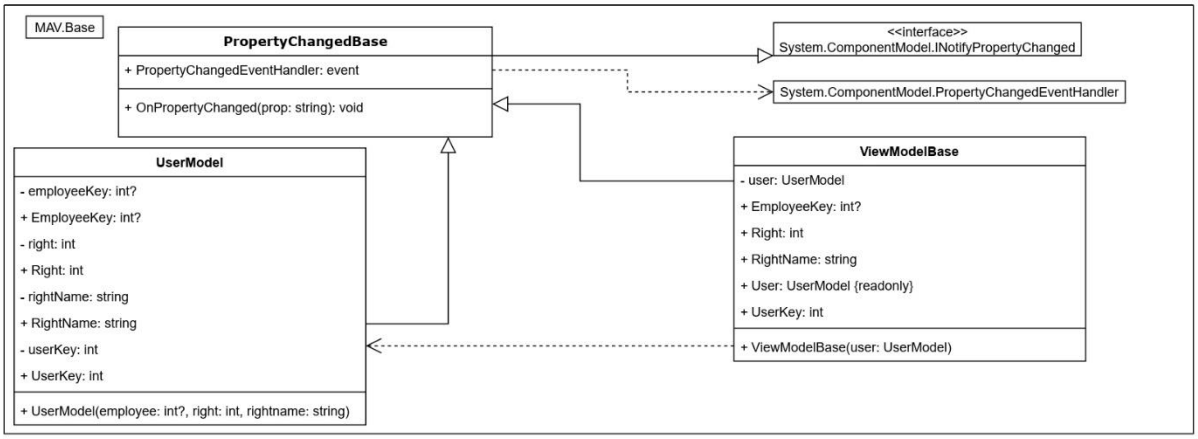
Sprint Backlog 05 von 25.06 - 30.06

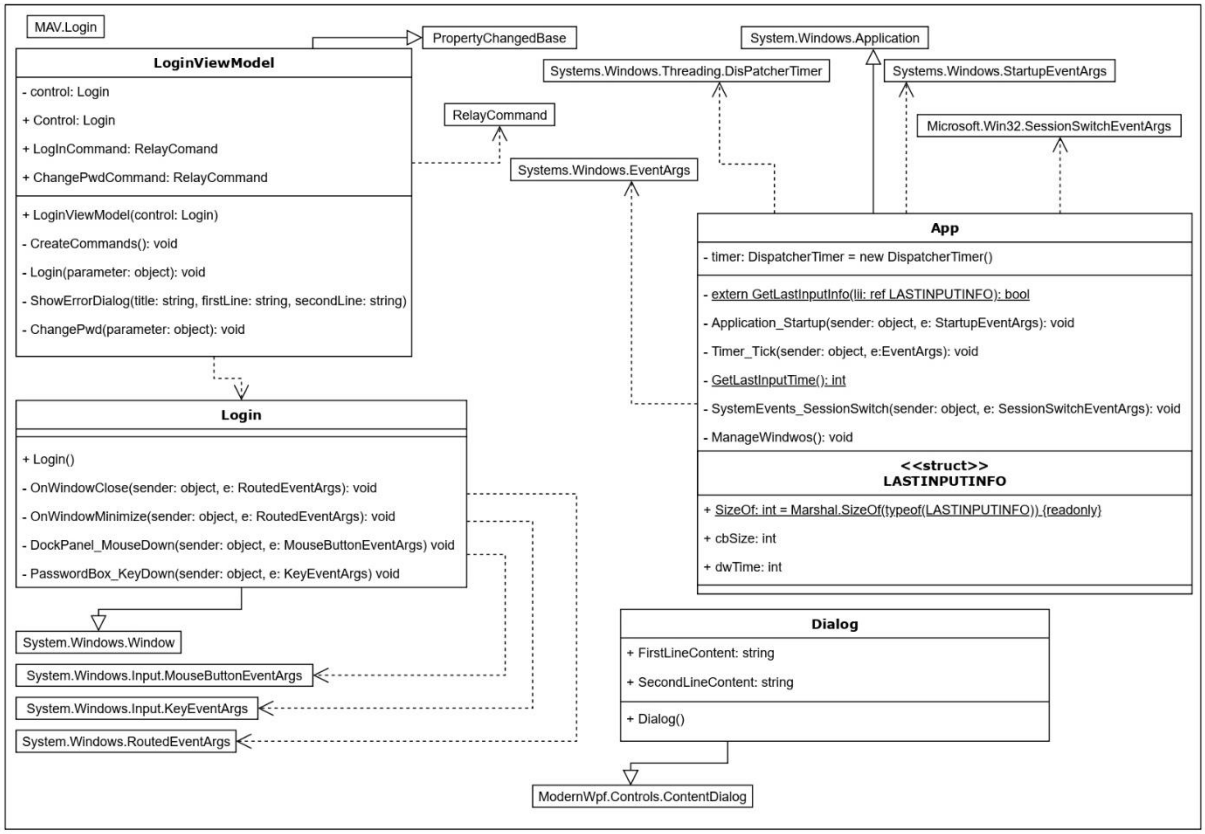
ID	Beschreibung	Sub-Beschreibung	S-Mark	Bearbeiter	Code	Review	Done
US.04	Ich als User möchte meine eigenen Daten einsehen können	Geschäftliche und Private Daten einsehbar	ST02	Manuel			
US.04_1	Sonderrecht auf eigene Daten			Manuel			
US.05	Die Kollegen von mir sollen nur meine geschäftliche n Daten einsehen können	Geschäftliche Daten: Name, gesch. Telefonnummer, Abteilung, Beruf, Geburtstag (ohne Jahr)	ST02	Manuel			
US.05_1	Anzeigemögli chkeiten von Mitarbeitern	Geschäftliche Daten werden angezeigt. Extra Button für besondere Rechte zu privaten Daten	-	Manuel			
US.11	Als Admin möchte ich nicht alles alleine managen.	Verschiedene Rechte mit verschiedenen Möglichkeiten		Manuel			
US.11_1	Verschiedene Möglichkeiten mit den Rechten			Manuel			
US.10	Wenn ich mich eingeloggt habe, möchte ich das Passwort ändern können	Eine Seite für die eigenen Einstellungen. Altes Passwort für die Bestätigung und 2x das neue Passwort eingeben als Bestätigung		Tobi			
US.16	Als Mitarbeiter der Personalabteil ung will ich zu Mitarbeitern Personaldate n speichern (Einstellungsd atum,			Manuel			

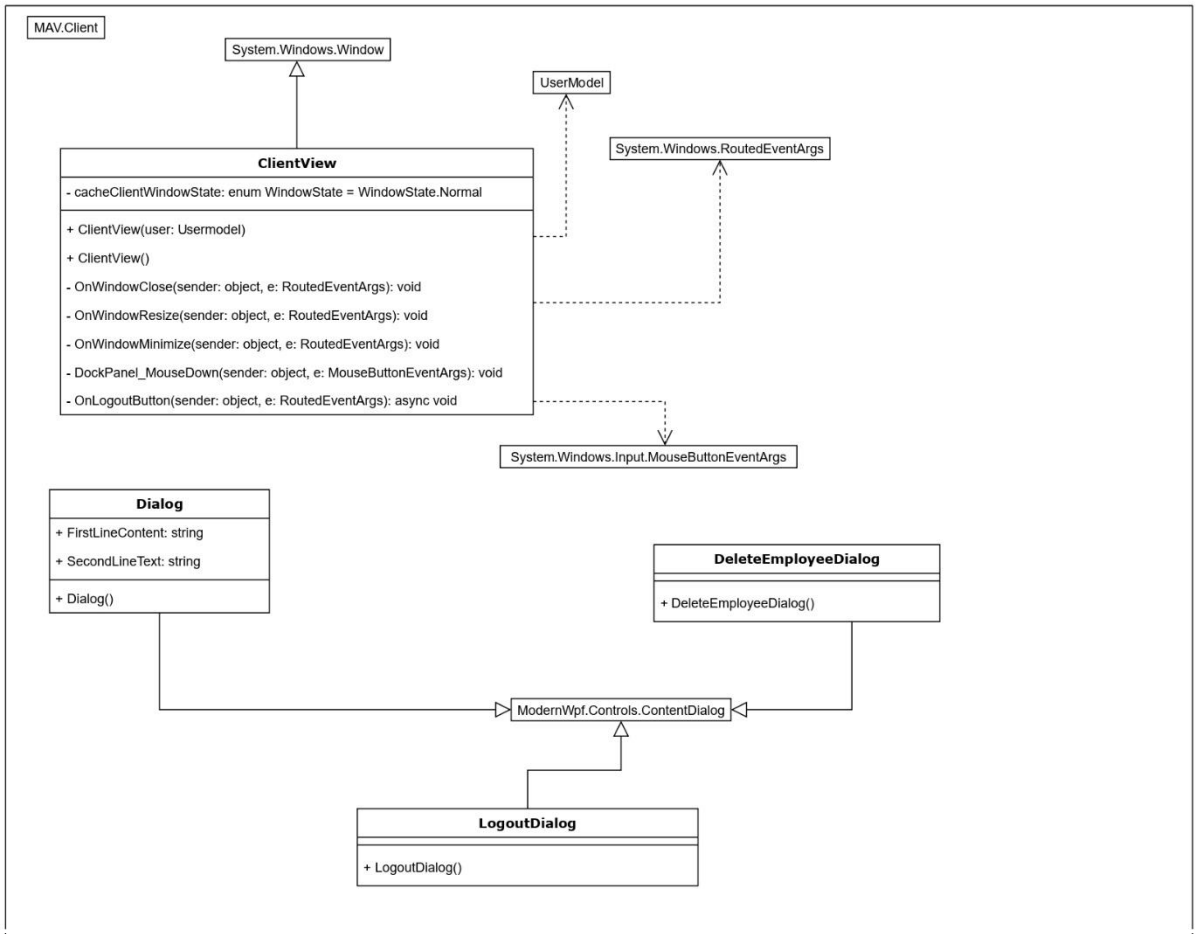
	Kündigungsfrist, Abmahnungen, Gehalt, Bonuszahlungen, Kommentarfeld) um diese Daten im System zu verwalten.						
US.08	Es soll eine Seite geben, auf der man einen Mitarbeiter hinzufügen kann	Extra Seite um einen Mitarbeiter zu erstelllen		Manuel			
US.18	Ich als Personalmitarbeiter (oder Admin) möchte die Möglichkeit haben, die Daten eines Mitarbieters zu bearbeiten	Durch Button auf Informationsseite erreichbar		Manuel			
EXTRA	Dokumentation			Samuel/Petra/ Christoph			

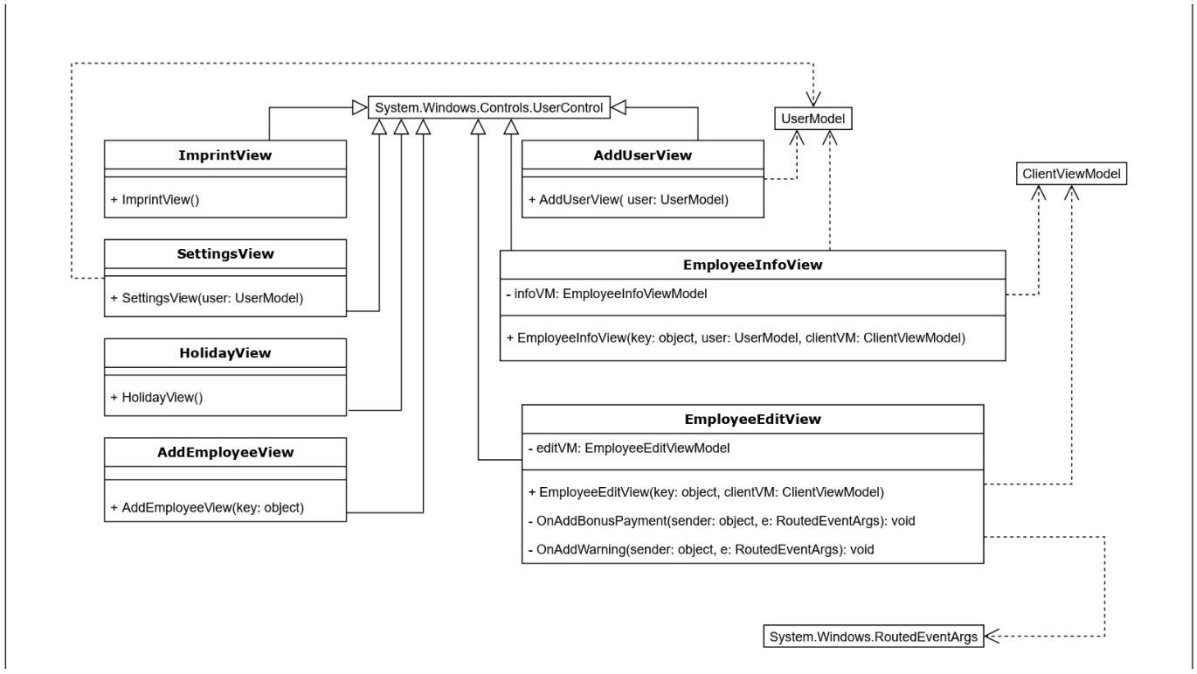
Klassendiagramme

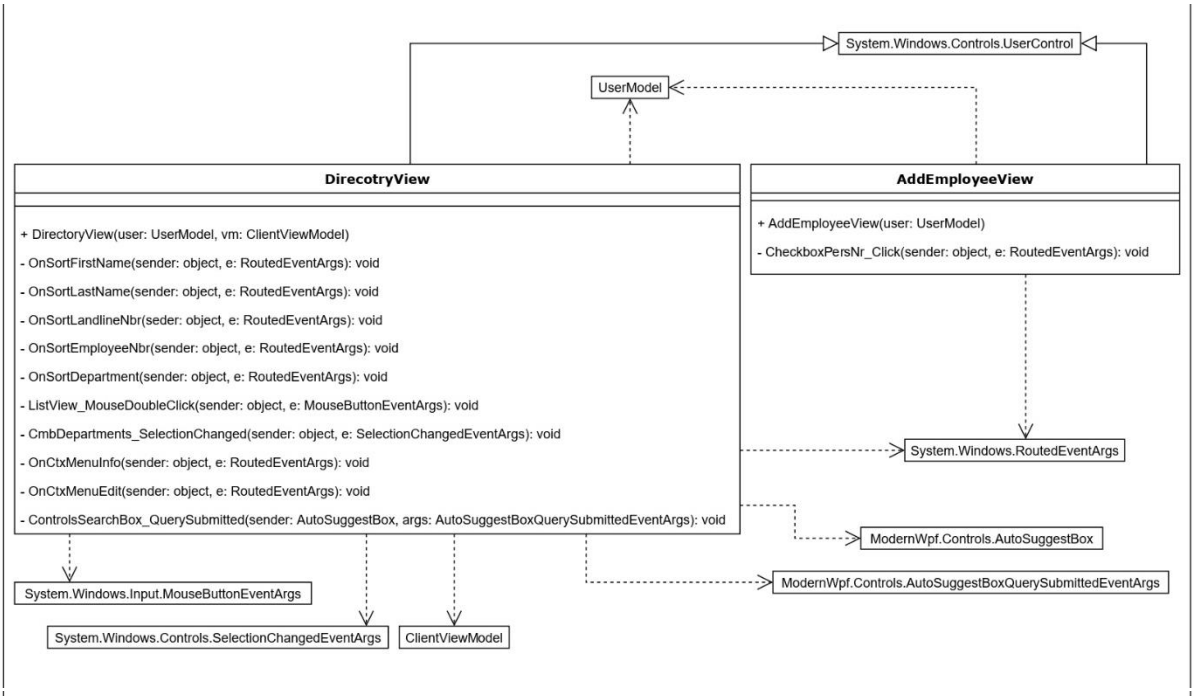


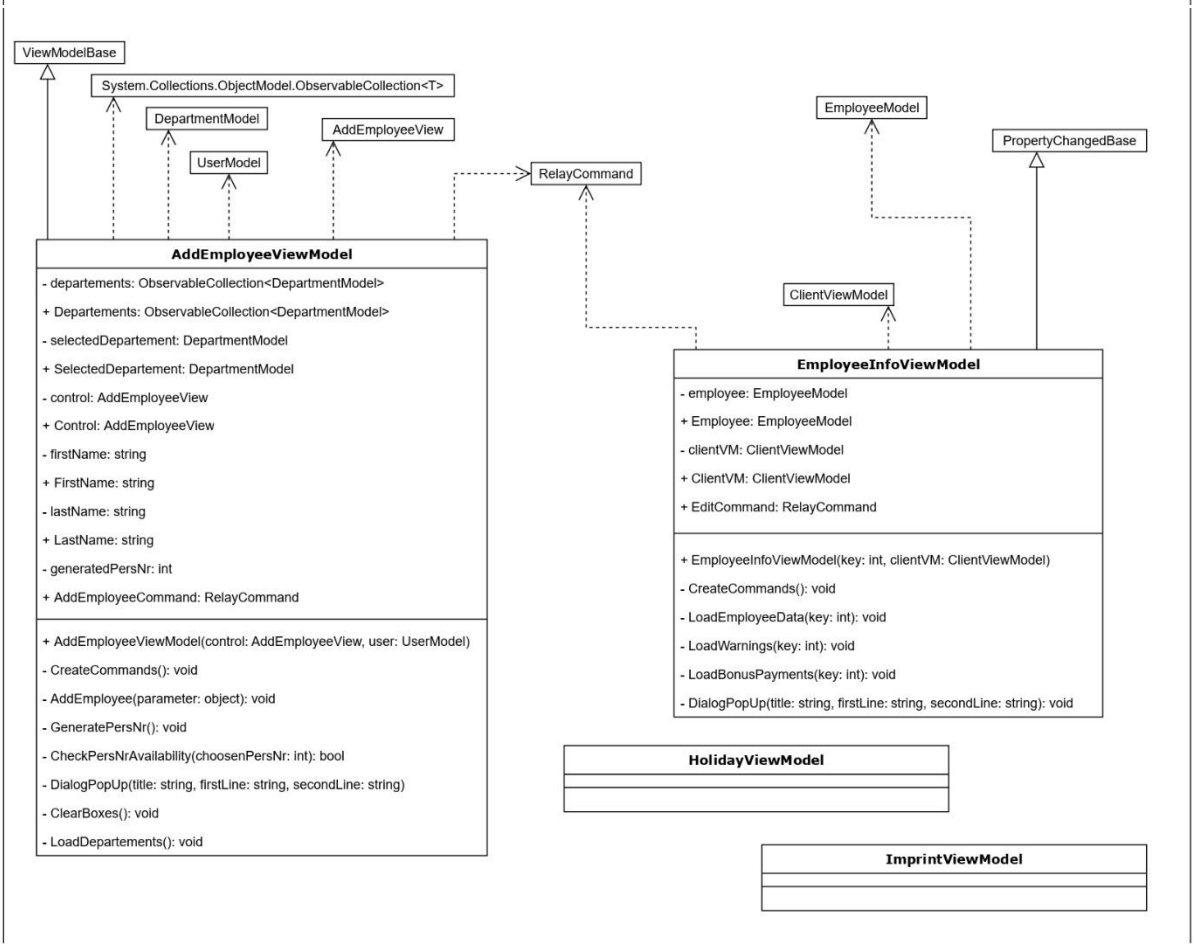


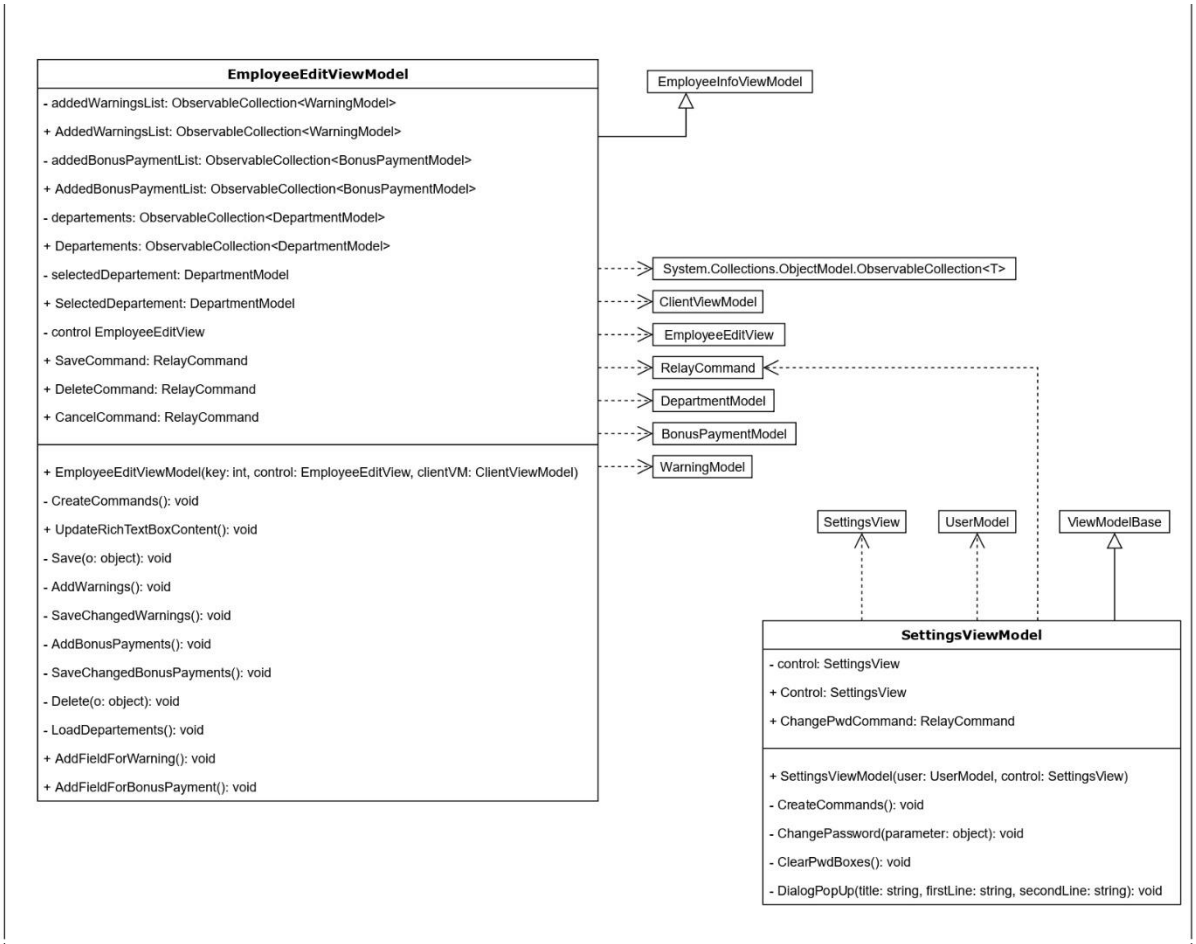


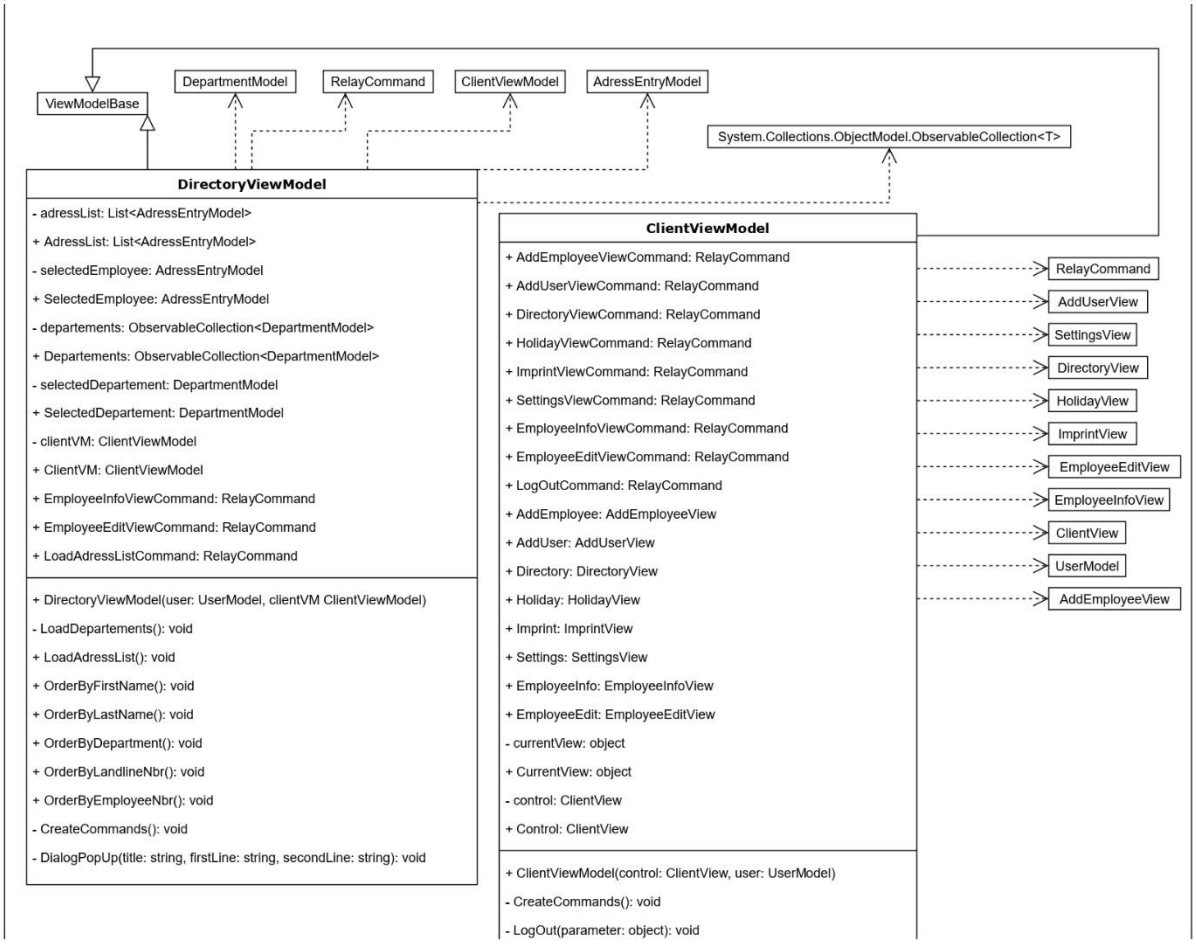

MAV.Client
System.Windows.Window
UserModel
System.Windows.RoutedEventArgs
ClientView
- cacheClientWindowState: enum WindowState = WindowState.Normal
+ ClientView(user: Usermodel)
+ ClientView()
- OnWindowClose(sender: object, e: RoutedEventArgs): void
- OnWindowResize(sender: object, e: RoutedEventArgs): void
- OnWindowMinimize(sender: object, e: RoutedEventArgs): void
- DockPanel_MouseDown(sender: object, e: MouseButtonEventArgs): void
- OnLogoutButton(sender: object, e: RoutedEventArgs): async void
System.Windows.Input.MouseButtonEventArgs
Dialog
+ FirstLineContent: string
+ SecondLineText: string
+ Dialog()
DeleteEmployeeDialog
+ DeleteEmployeeDialog()
ModernWpf.Controls.ContentDialog
LogoutDialog
+ LogoutDialog()

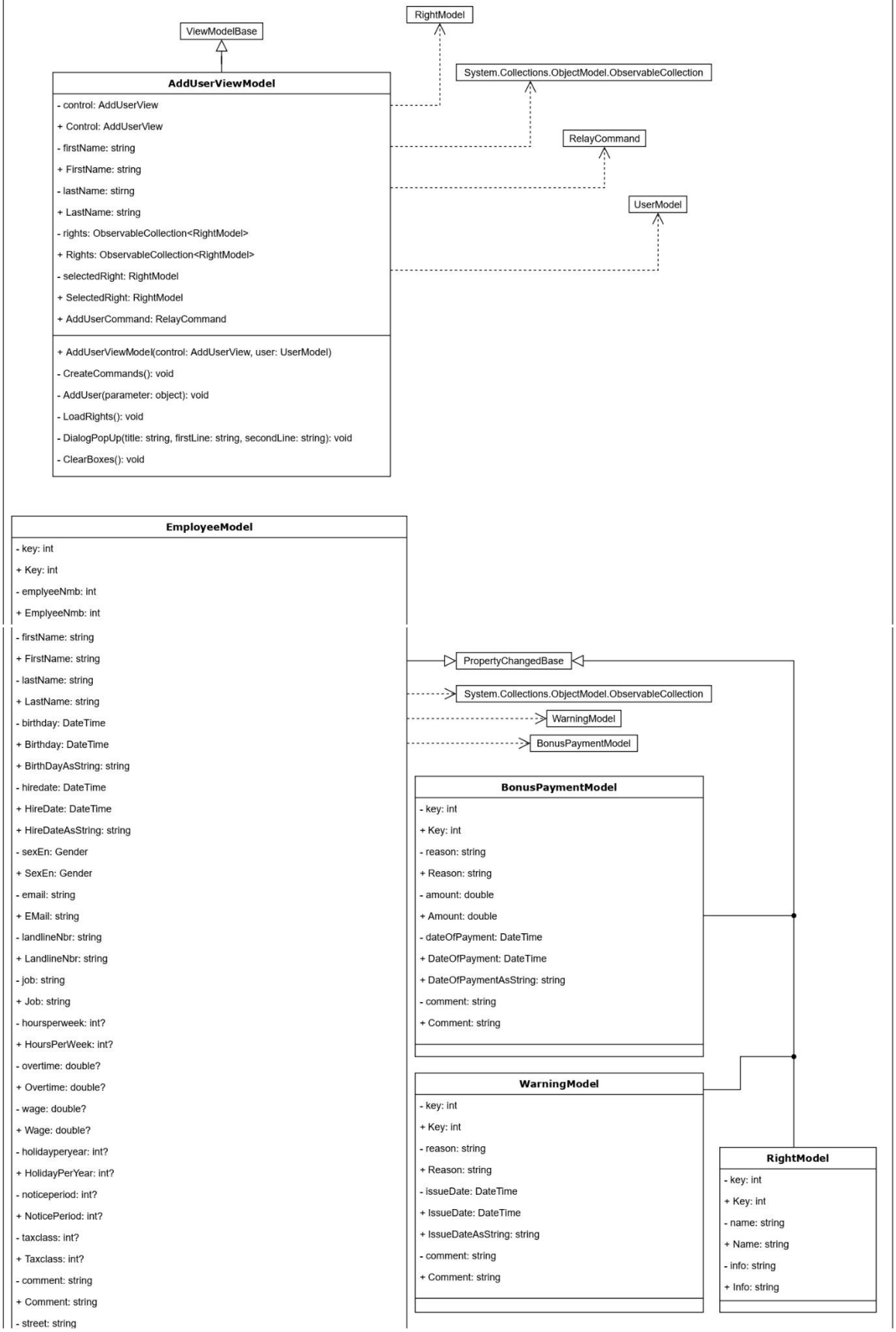

System.Windows.Controls.UserControl
UserModel
ClientViewModel
ImprintView
+ ImprintView()
AddUserView
+ AddUserView(user: UserModel)
SettingsView
+ SettingsView(user: UserModel)
EmployeeInfoView
- infoVM: EmployeeInfoViewModel
+ EmployeeInfoView(key: object, user: UserModel, clientVM: ClientViewModel)
HolidayView
+ HolidayView()
EmployeeEditView
- editVM: EmployeeEditViewModel
+ EmployeeEditView(key: object, clientVM: ClientViewModel)
- OnAddBonusPayment(sender: object, e: RoutedEventArgs): void
- OnAddWarning(sender: object, e: RoutedEventArgs): void
AddEmployeeView
+ AddEmployeeView(key: object)
System.Windows.RoutedEventArgs

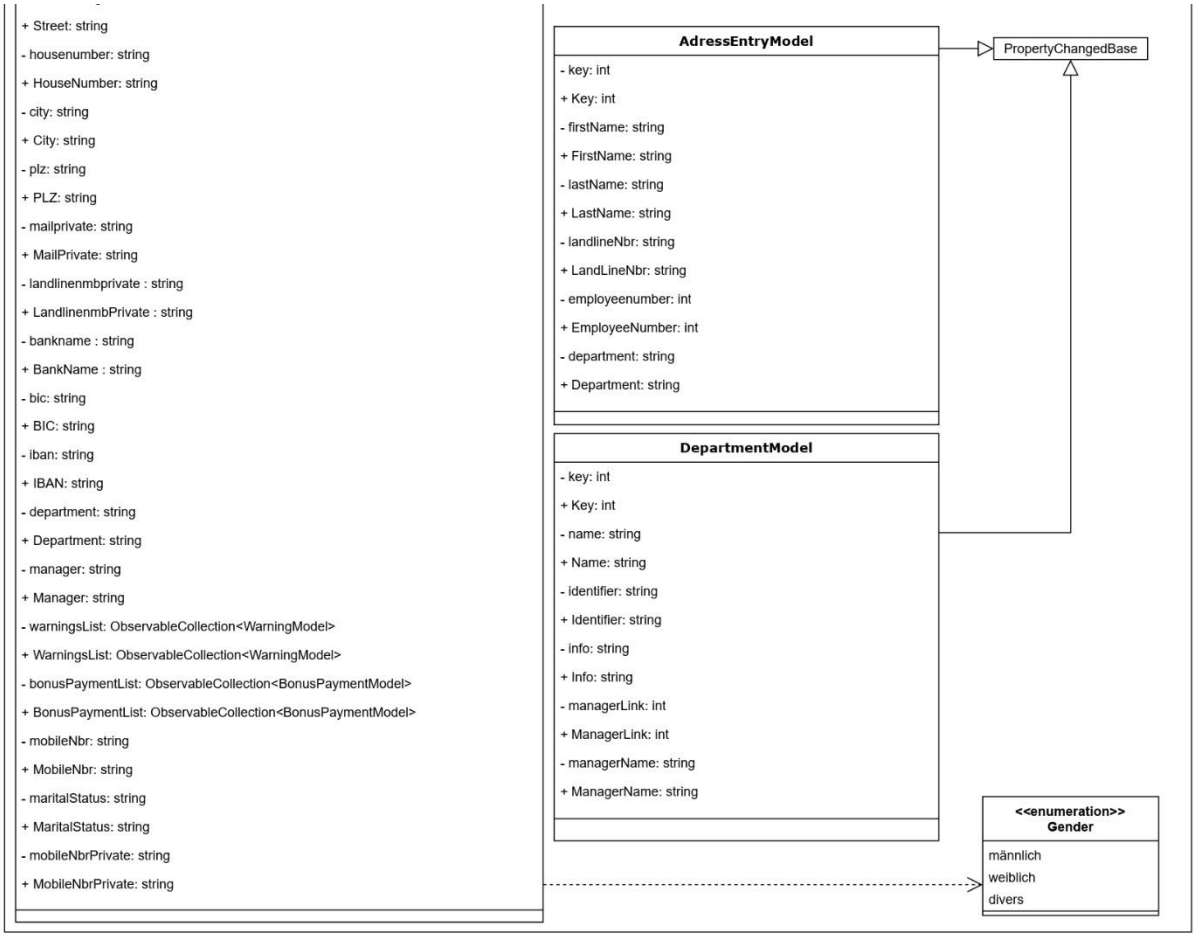

System.Windows.Controls.UserControl
UserModel
DirecotryView
+ DirectoryView(user: UserModel, vm: ClientViewModel)
- OnSortFirstName(sender: object, e: RoutedEventArgs): void
- OnSortLastName(sender: object, e: RoutedEventArgs): void
- OnSortLandlineNbr(seder: object, e: RoutedEventArgs): void
- OnSortEmployeeNbr(sender: object, e: RoutedEventArgs): void
- OnSortDepartment(sender: object, e: RoutedEventArgs): void
- ListView_MouseDoubleClick(sender: object, e: MouseButtonEventArgs): void
- CmbDepartments_SelectionChanged(sender: object, e: SelectionChangedEventArgs): void
- OnCtxMenuInfo(sender: object, e: RoutedEventArgs): void
- OnCtxMenuEdit(sender: object, e: RoutedEventArgs): void
- ControlsSearchBox_QuerySubmitted(sender: AutoSuggestBox, args: AutoSuggestBoxQuerySubmittedEventArgs): void
AddEmployeeView
+ AddEmployeeView(user: UserModel)
- CheckboxPersNr_Click(sender: object, e: RoutedEventArgs): void
System.Windows.RoutedEventArgs
ModernWpf.Controls.AutoSuggestBox
ModernWpf.Controls.AutoSuggestBoxQuerySubmittedEventArgs
System.Windows.Input.MouseButtonEventArgs
System.Windows.Controls.SelectionChangedEventArgs
ClientViewModel


ViewModelBase
System.Collections.ObjectModel.ObservableCollection<T>
DepartmentModel
UserModel
AddEmployeeView
RelayCommand
EmployeeModel
PropertyChangedBase
ClientViewModel
AddEmployeeViewModel
- departements: ObservableCollection<DepartmentModel>
+ Departements: ObservableCollection<DepartmentModel>
- selectedDepartement: DepartmentModel
+ SelectedDepartement: DepartmentModel
- control: AddEmployeeView
+ Control: AddEmployeeView
- firstName: string
+ FirstName: string
- lastName: string
+ LastName: string
- generatedPersNr: int
+ AddEmployeeCommand: RelayCommand
+ AddEmployeeViewModel(control: AddEmployeeView, user: UserModel)
- CreateCommands(): void
- AddEmployee(parameter: object): void
- GeneratePersNr(): void
- CheckPersNrAvailability(choosenPersNr: int): bool
- DialogPopUp(title: string, firstLine: string, secondLine: string)
- ClearBoxes(): void
- LoadDepartements(): void
EmployeeInfoViewModel
- employee: EmployeeModel
+ Employee: EmployeeModel
- clientVM: ClientViewModel
+ ClientVM: ClientViewModel
+ EditCommand: RelayCommand
+ EmployeeInfoViewModel(key: int, clientVM: ClientViewModel)
- CreateCommands(): void
- LoadEmployeeData(key: int): void
- LoadWarnings(key: int): void
- LoadBonusPayments(key: int): void
- DialogPopUp(title: string, firstLine: string, secondLine: string): void
HolidayViewModel
ImprintViewModel


EmployeeEditViewModel
- addedWarningsList: ObservableCollection<WarningModel>
+ AddedWarningsList: ObservableCollection<WarningModel>
- addedBonusPaymentList: ObservableCollection<BonusPaymentModel>
+ AddedBonusPaymentList: ObservableCollection<BonusPaymentModel>
- departements: ObservableCollection<DepartmentModel>
+ Departements: ObservableCollection<DepartmentModel>
- selectedDepartement: DepartmentModel
+ SelectedDepartement: DepartmentModel
- control EmployeeEditView
+ SaveCommand: RelayCommand
+ DeleteCommand: RelayCommand
+ CancelCommand: RelayCommand
+ EmployeeEditViewModel(key: int, control: EmployeeEditView, clientVM: ClientViewModel)
- CreateCommands(): void
+ UpdateRichTextBoxContent(): void
- Save(o: object): void
- AddWarnings(): void
- SaveChangedWarnings(): void
- AddBonusPayments(): void
- SaveChangedBonusPayments(): void
- Delete(o: object): void
- LoadDepartements(): void
+ AddFieldForWarning(): void
+ AddFieldForBonusPayment(): void
EmployeeInfoViewModel
System.Collections.ObjectModel.ObservableCollection<T>
ClientViewModel
EmployeeEditView
RelayCommand
DepartmentModel
BonusPaymentModel
WarningModel
SettingsView
UserModel
ViewModelBase
SettingsViewModel
- control: SettingsView
+ Control: SettingsView
+ ChangePwdCommand: RelayCommand
+ SettingsViewModel(user: UserModel, control: SettingsView)
- CreateCommands(): void
- ChangePassword(parameter: object): void
- ClearPwdBoxes(): void
- DialogPopUp(title: string, firstLine: string, secondLine: string): void

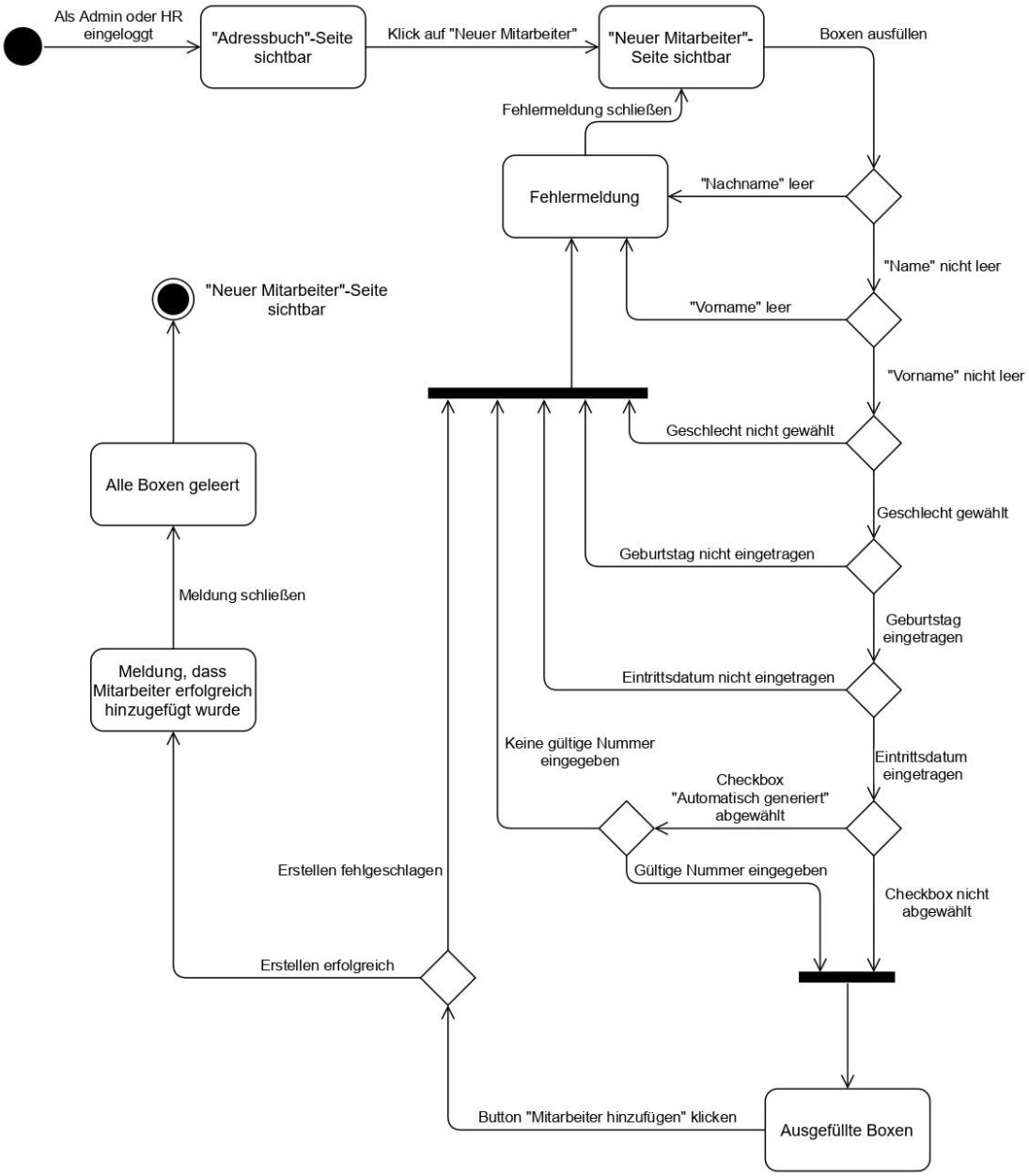

ViewModelBase
DepartmentModel
RelayCommand
ClientViewModel
AdressEntryModel
System.Collections.ObjectModel.ObservableCollection<T>
DirectoryViewModel
- adressList: List<AdressEntryModel>
+ AdressList: List<AdressEntryModel>
- selectedEmployee: AdressEntryModel
+ SelectedEmployee: AdressEntryModel
- departements: ObservableCollection<DepartmentModel>
+ Departements: ObservableCollection<DepartmentModel>
- selectedDepartement: DepartmentModel
+ SelectedDepartement: DepartmentModel
- clientVM: ClientViewModel
+ ClientVM: ClientViewModel
+ EmployeeInfoViewCommand: RelayCommand
+ EmployeeEditViewCommand: RelayCommand
+ LoadAdressListCommand: RelayCommand
+ DirectoryViewModel(user: UserModel, clientVM ClientViewModel)
- LoadDepartements(): void
+ LoadAdressList(): void
+ OrderByFirstName(): void
+ OrderByLastName(): void
+ OrderByDepartment(): void
+ OrderByLandlineNbr(): void
+ OrderByEmployeeNbr(): void
- CreateCommands(): void
- DialogPopUp(title: string, firstLine: string, secondLine: string): void
ClientViewModel
+ AddEmployeeViewCommand: RelayCommand
+ AddUserViewCommand: RelayCommand
+ DirectoryViewCommand: RelayCommand
+ HolidayViewCommand: RelayCommand
+ ImprintViewCommand: RelayCommand
+ SettingsViewCommand: RelayCommand
+ EmployeeInfoViewCommand: RelayCommand
+ EmployeeEditViewCommand: RelayCommand
+ LogOutCommand: RelayCommand
+ AddEmployee: AddEmployeeView
+ AddUser: AddUserView
+ Directory: DirectoryView
+ Holiday: HolidayView
+ Imprint: ImprintView
+ Settings: SettingsView
+ EmployeeInfo: EmployeeInfoView
+ EmployeeEdit: EmployeeEditView
- currentView: object
+ CurrentView: object
- control: ClientView
+ Control: ClientView
+ ClientViewModel(control: ClientView, user: UserModel)
- CreateCommands(): void
- LogOut(parameter: object): void
RelayCommand
AddUserView
SettingsView
DirectoryView
HolidayView
ImprintView
EmployeeEditView
EmployeeInfoView
ClientView
UserModel
AddEmployeeView


ViewModelBase
RightModel
System.Collections.ObjectModel.ObservableCollection
RelayCommand
UserModel
AddUserViewModel
- control: AddUserView
+ Control: AddUserView
- firstName: string
+ FirstName: string
- lastName: stirng
+ LastName: string
- rights: ObservableCollection<RightModel>
+ Rights: ObservableCollection<RightModel>
- selectedRight: RightModel
+ SelectedRight: RightModel
+ AddUserCommand: RelayCommand
+ AddUserViewModel(control: AddUserView, user: UserModel)
- CreateCommands(): void
- AddUser(parameter: object): void
- LoadRights(): void
- DialogPopUp(title: string, firstLine: string, secondLine: string): void
- ClearBoxes(): void
EmployeeModel
- key: int
+ Key: int
- emplyeeNmb: int
+ EmplyeeNmb: int
- firstName: string
+ FirstName: string
- lastName: string
+ LastName: string
- birthday: DateTime
+ Birthday: DateTime
+ BirthDayAsString: string
- hiredate: DateTime
+ HireDate: DateTime
+ HireDateAsString: string
- sexEn: Gender
+ SexEn: Gender
- email: string
+ EMail: string
- landlineNbr: string
+ LandlineNbr: string
- job: string
+ Job: string
- hoursperweek: int?
+ HoursPerWeek: int?
- overtime: double?
+ Overtime: double?
- wage: double?
+ Wage: double?
- holidayperyear: int?
+ HolidayPerYear: int?
- noticeperiod: int?
+ NoticePeriod: int?
- taxclass: int?
+ Taxclass: int?
- comment: string
+ Comment: string
- street: string
PropertyChangedBase
System.Collections.ObjectModel.ObservableCollection
WarningModel
BonusPaymentModel
BonusPaymentModel
- key: int
+ Key: int
- reason: string
+ Reason: string
- amount: double
+ Amount: double
- dateOfPayment: DateTime
+ DateOfPayment: DateTime
+ DateOfPaymentAsString: string
- comment: string
+ Comment: string
WarningModel
- key: int
+ Key: int
- reason: string
+ Reason: string
- issueDate: DateTime
+ IssueDate: DateTime
+ IssueDateAsString: string
- comment: string
+ Comment: string
RightModel
- key: int
+ Key: int
- name: string
+ Name: string
- info: string
+ Info: string

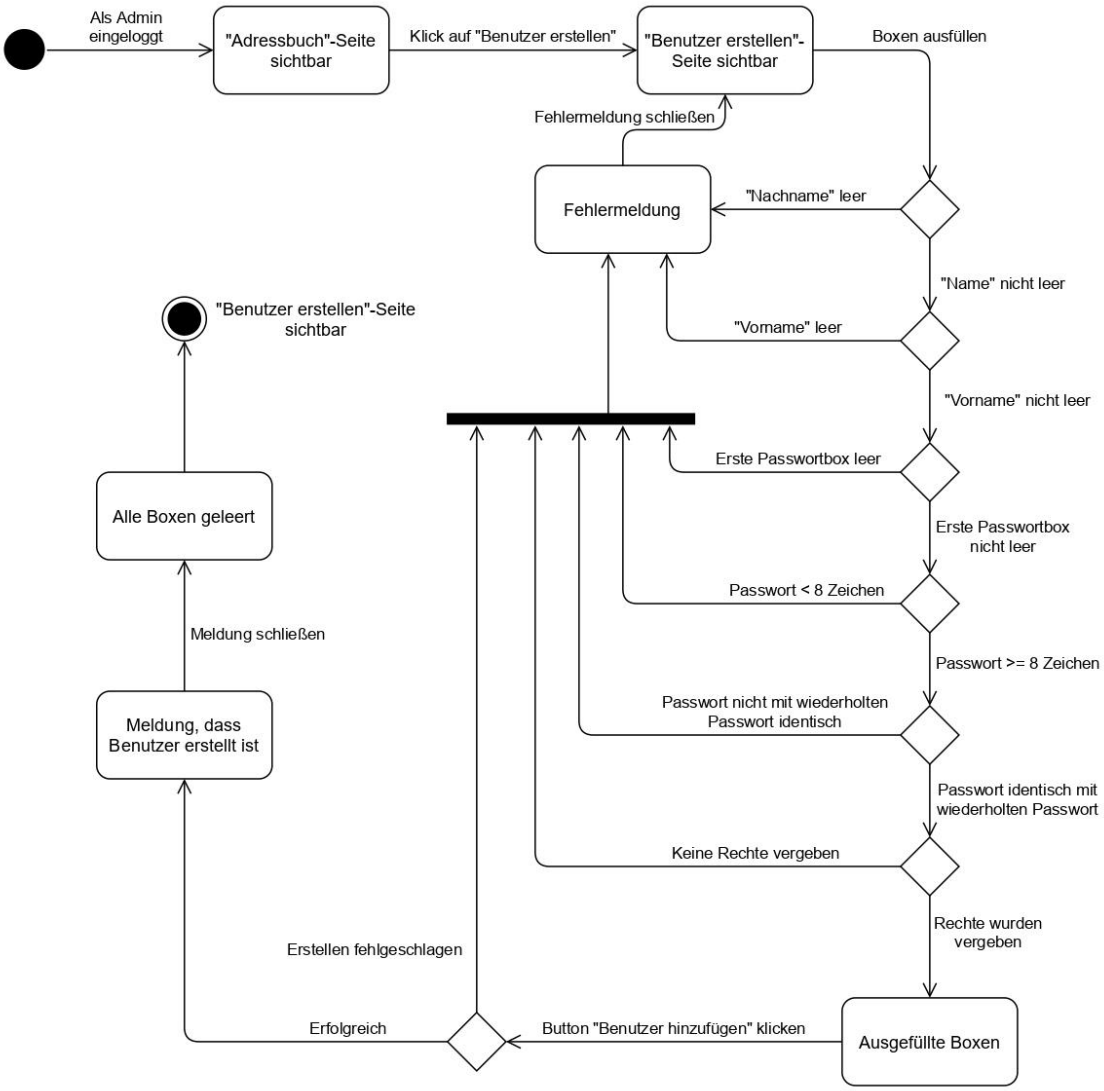

+ Street: string
- housenumber: string
+ HouseNumber: string
- city: string
+ City: string
- plz: string
+ PLZ: string
- mailprivate: string
+ MailPrivate: string
- landlinenmbprivate : string
+ LandlinenmbPrivate : string
- bankname : string
+ BankName : string
- bic: string
+ BIC: string
- iban: string
+ IBAN: string
- department: string
+ Department: string
- manager: string
+ Manager: string
- warningsList: ObservableCollection<WarningModel>
+ WarningsList: ObservableCollection<WarningModel>
- bonusPaymentList: ObservableCollection<BonusPaymentModel>
+ BonusPaymentList: ObservableCollection<BonusPaymentModel>
- mobileNbr: string
+ MobileNbr: string
- maritalStatus: string
+ MaritalStatus: string
- mobileNbrPrivate: string
+ MobileNbrPrivate: string
AdressEntryModel
- key: int
+ Key: int
- firstName: string
+ FirstName: string
- lastName: string
+ LastName: string
- landlineNbr: string
+ LandLineNbr: string
- employeenumber: int
+ EmployeeNumber: int
- department: string
+ Department: string
PropertyChangedBase
DepartmentModel
- key: int
+ Key: int
- name: string
+ Name: string
- identifier: string
+ Identifier: string
- info: string
+ Info: string
- managerLink: int
+ ManagerLink: int
- managerName: string
+ ManagerName: string
<<enumeration>>
Gender
männlich
weiblich
divers

Zustandsdiagramme

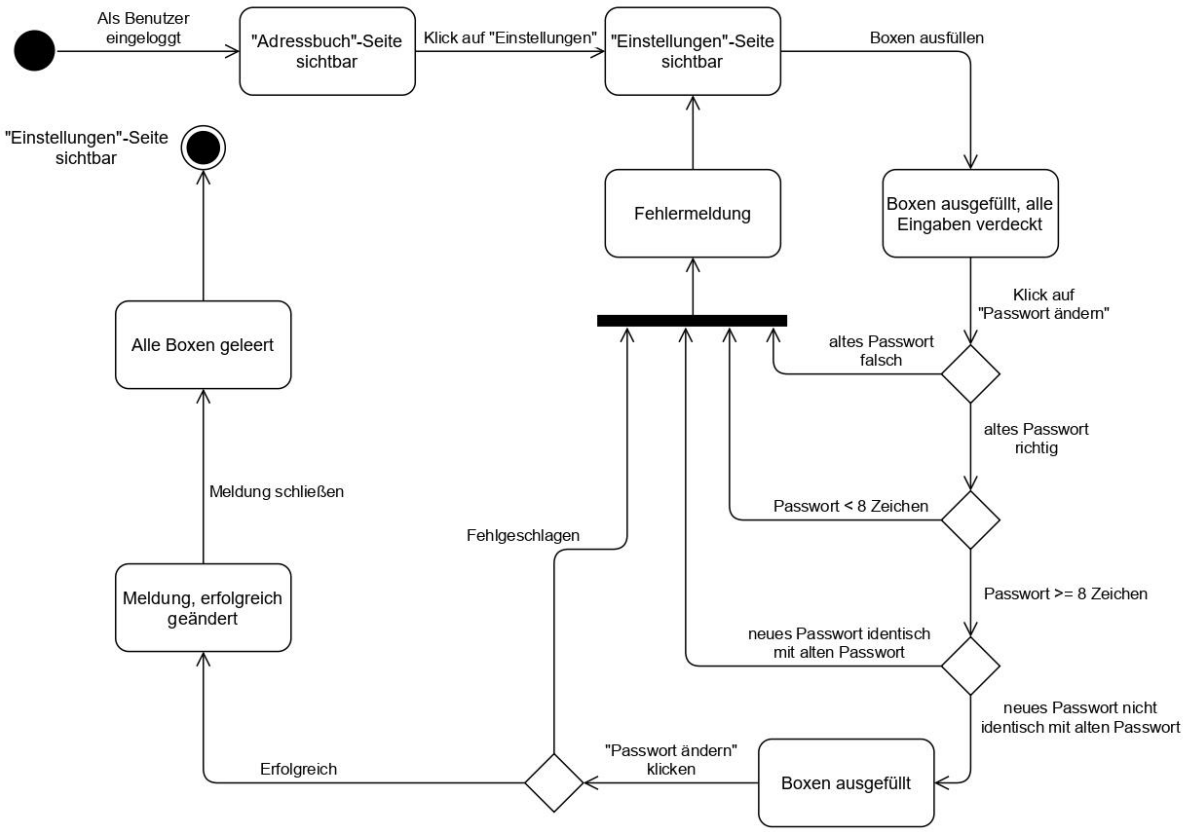
Mitarbeiter hinzufügen:



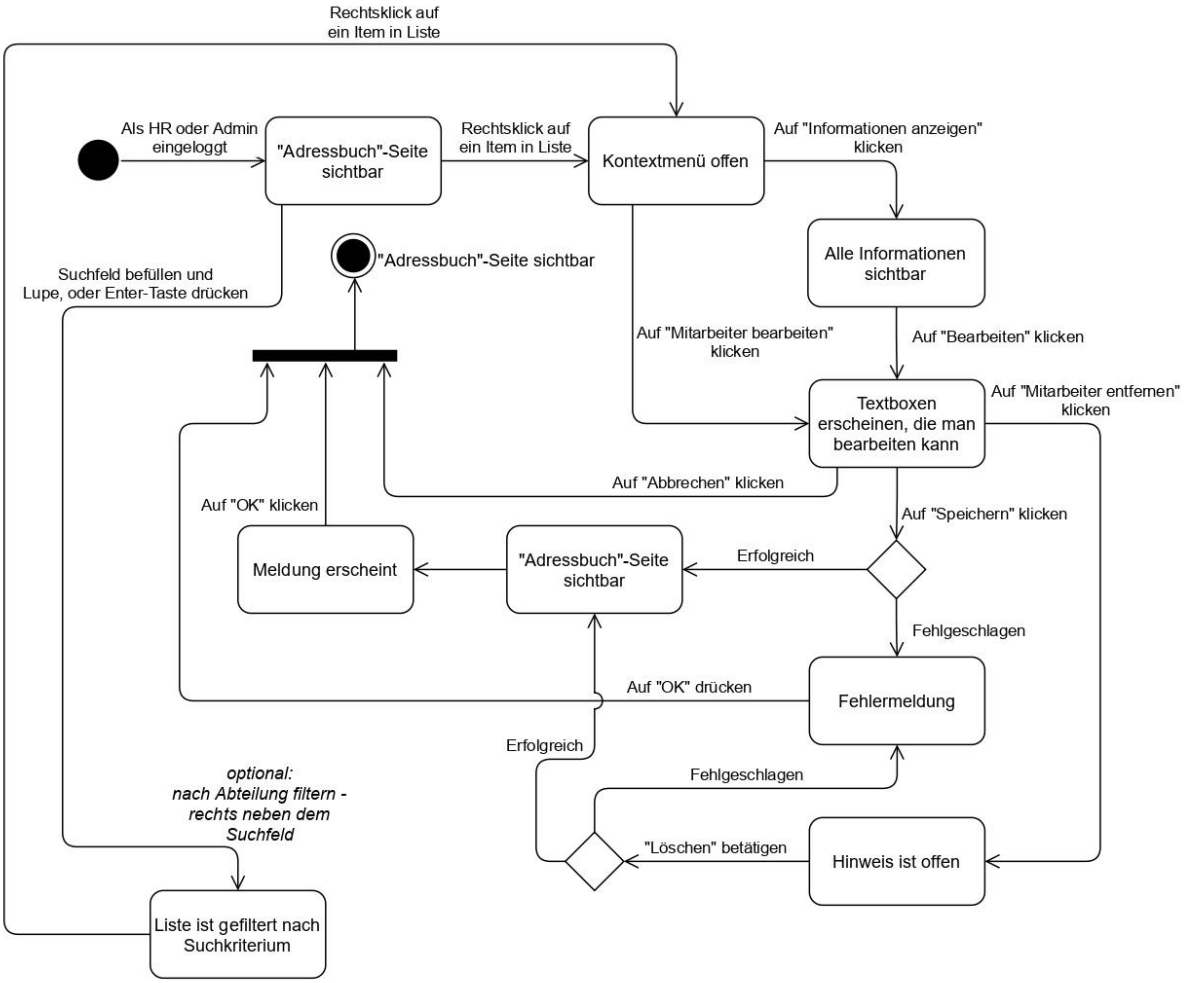
Benutzer hinzufügen:



Passwort ändern:

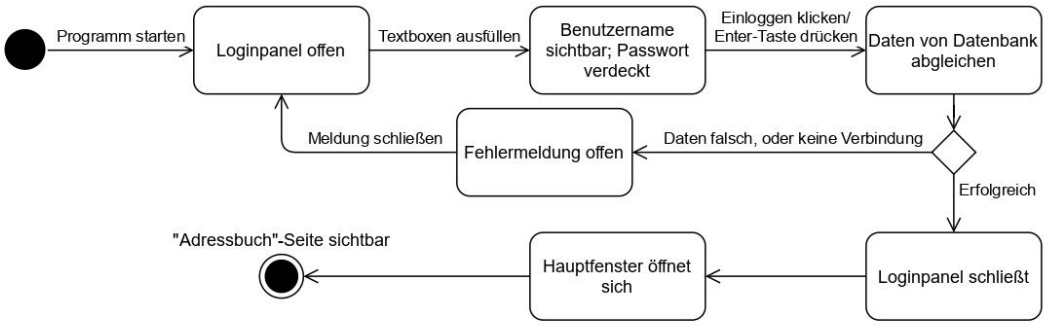


Mitarbeiter Informationen anzeigen / ändern:

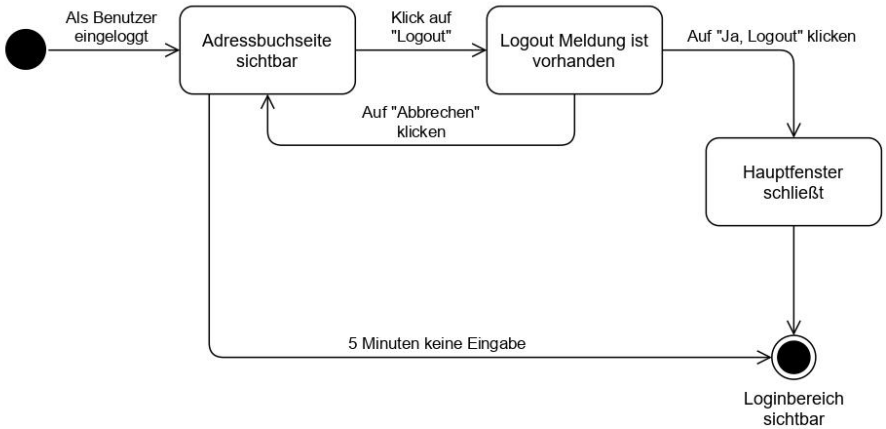


Impressum:

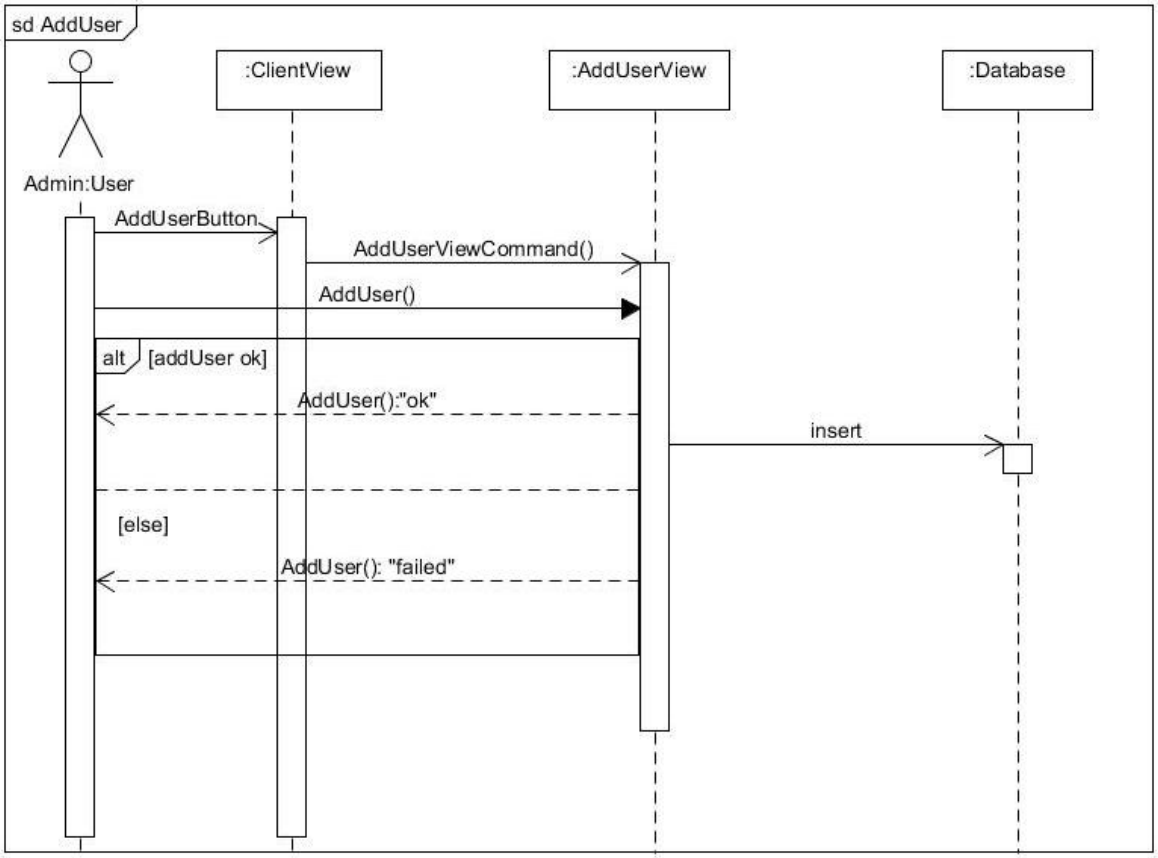
Einloggen:

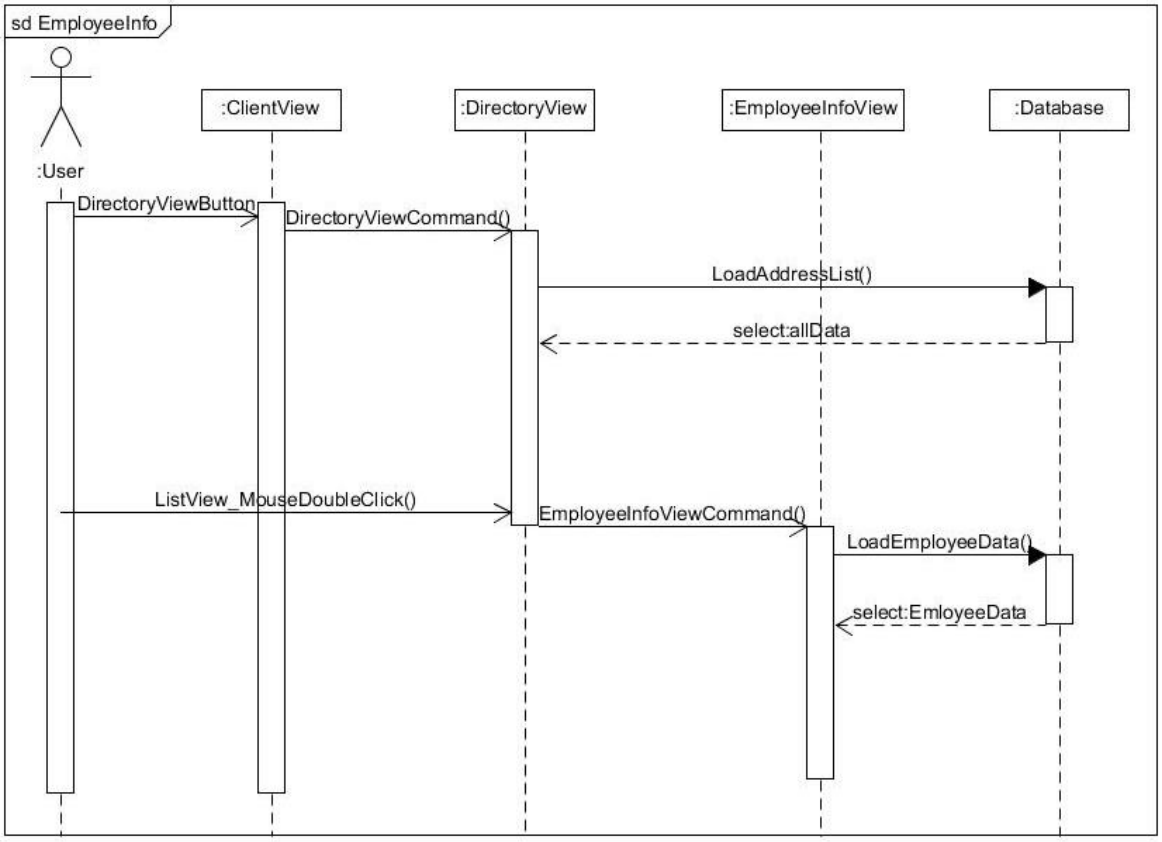


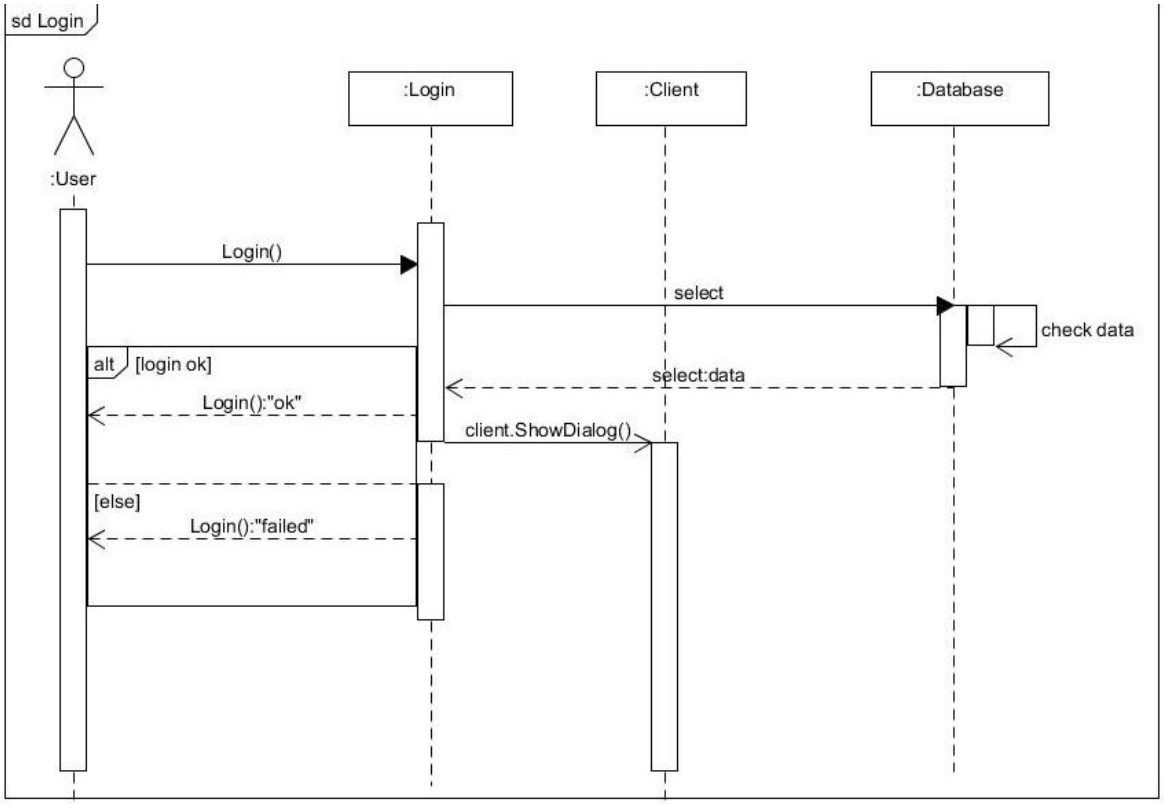
Ausloggen:

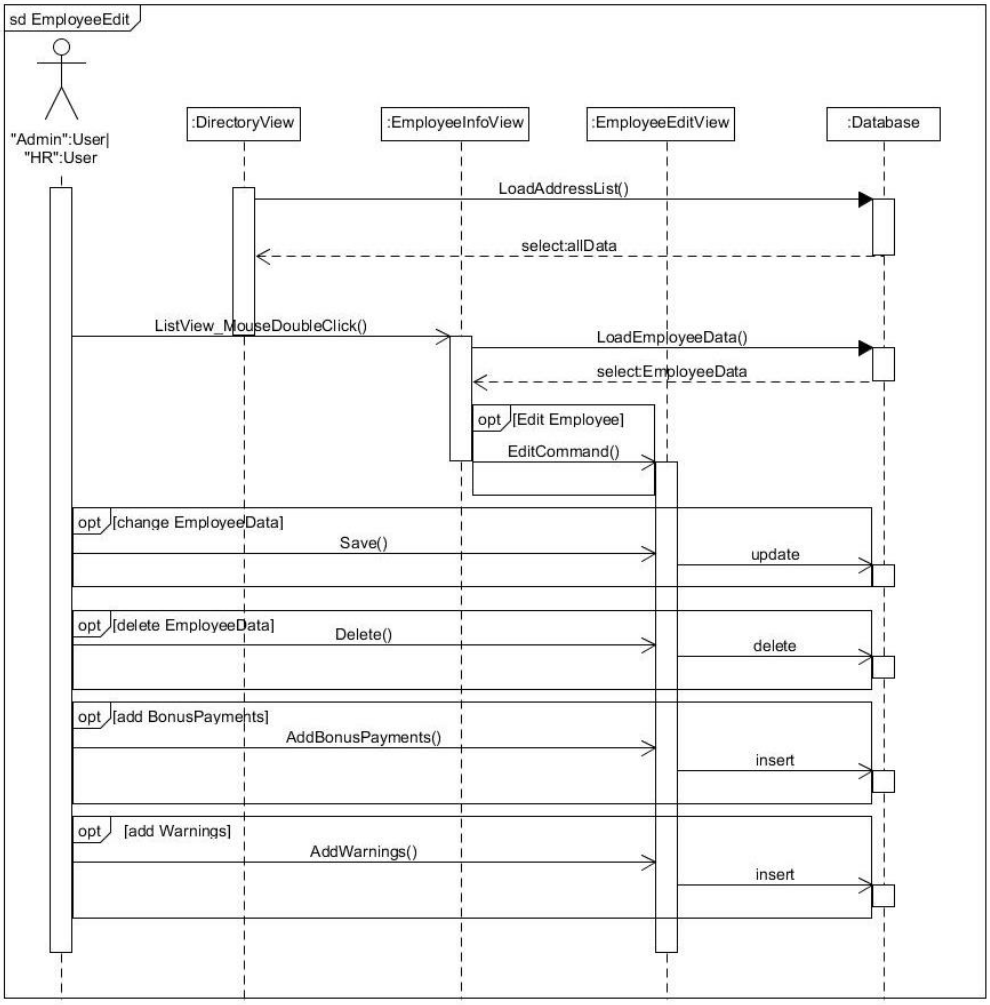


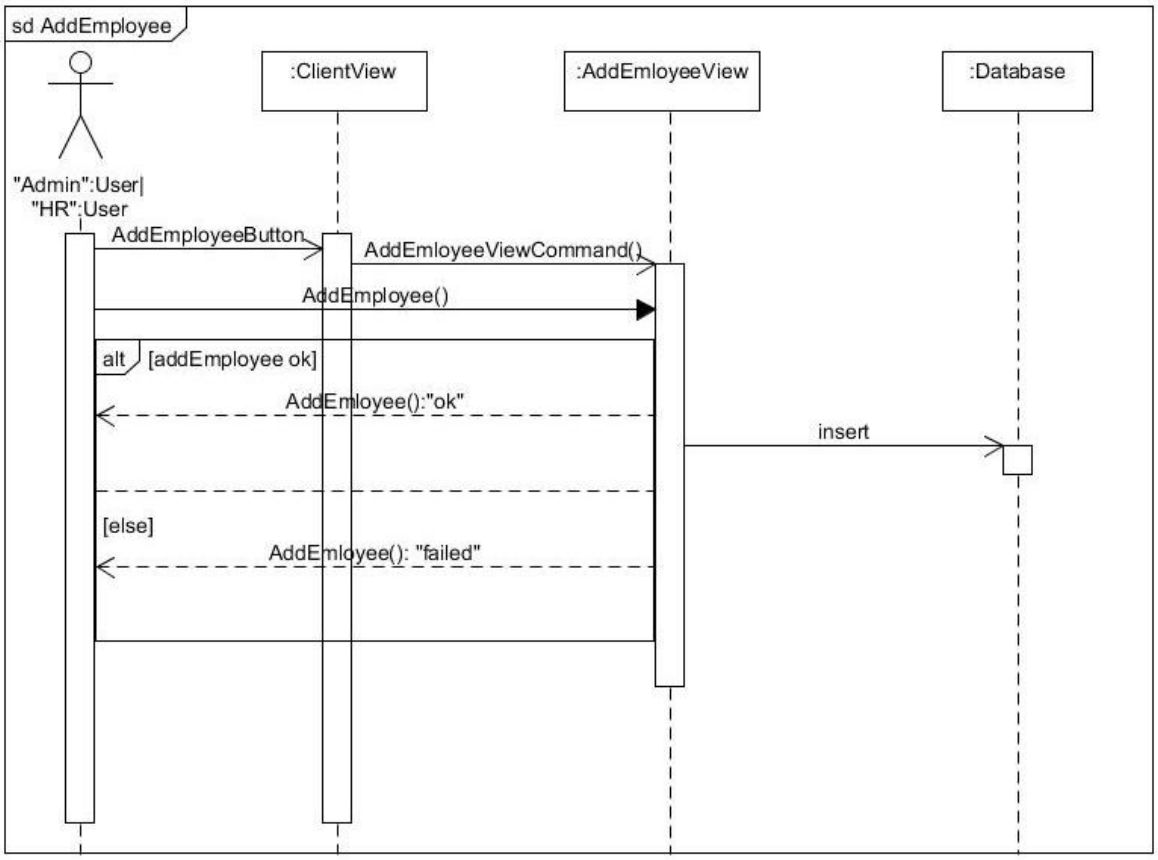
Sequenzdiagramme

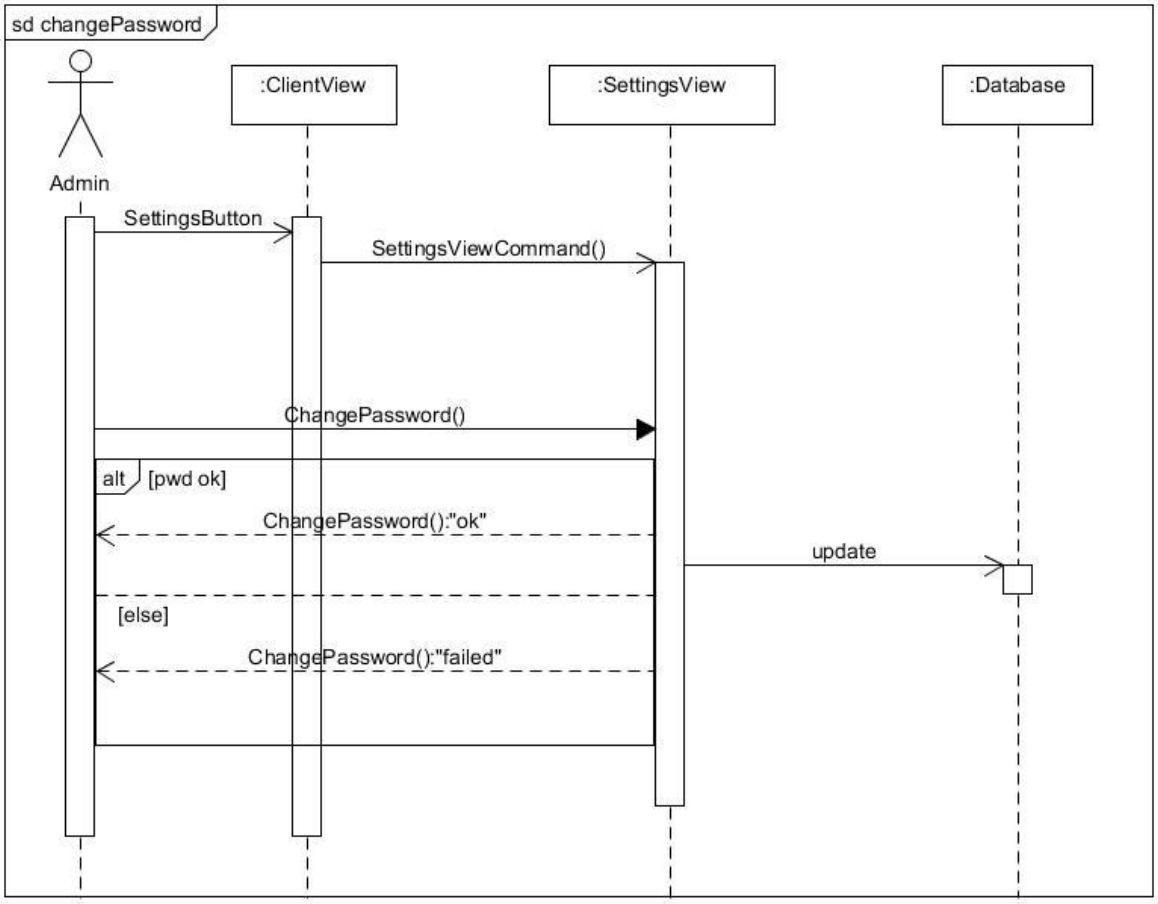





sd Login
:User
:Login
:Client
:Database
Login()
select
check data
alt
[login ok]
select:data
Login():"ok"
client.ShowDialog()
[else]
Login():"failed"


sd EmployeeEdit
"Admin":User|
"HR":User
:DirectoryView
:EmployeeInfoView
:EmployeeEditView
:Database
LoadAddressList()
select:allData
ListView_MouseDoubleClick()
LoadEmployeeData()
select:EmployeeData
opt
[Edit Employee]
EditCommand()
opt
[change EmployeeData]
Save()
update
opt
[delete EmployeeData]
Delete()
delete
opt
[add BonusPayments]
AddBonusPayments()
insert
opt
[add Warnings]
AddWarnings()
insert


sd AddEmployee
:ClientView
:AddEmloyeeView
:Database
"Admin":User|
"HR":User
AddEmployeeButton
AddEmloyeeViewCommand()
AddEmployee()
alt
[addEmployee ok]
AddEmloyee():"ok"
insert
[else]
AddEmloyee(): "failed"


sd changePassword
:ClientView
:SettingsView
:Database
Admin
SettingsButton
SettingsViewCommand()
ChangePassword()
alt
[pwd ok]
ChangePassword():"ok"
update
[else]
ChangePassword():"failed"

Datenbankmodell