



Mogadu

Umsetzungsarbeit Modul 226B und Modul 120

Estermann Rafael und Cavic Daniel

17.06.2018

Inhalt

Einleitung.....	1
Ideenfindung	1
Vorsätze.....	1
Anforderungen	2
Login Fenster	2
Übersicht.....	3
Teamansicht.....	3
Navigation (Grundgerüst)	3
Navigation (Erweiterung)	4
Auftragsansicht	4
Auftragsdetailansicht.....	5
Aufgabendetail	5
Personendetail.....	5
Responsivität	6
Planung I.....	6
Paketerstellung.....	6
Zeitplan	8
Planung II	9
GUI Entwurf	9
Anmerkung zum GUI.....	11
Klassendiagramm.....	11
Klassendiagramm (nach MVVM)	14
Umsetzung I.....	15
Übersicht.....	15
Aufträge	16
Auftragsdetail	17
Aufgabendetail	18
Team	19
Navigation.....	20
Worte zum GUI	20
GUI Ergonomie	21

Aufgabenangemessenheit	21
Selbstbeschreibungsfähigkeit	21
Steuerbarkeit	21
Erwartungskonformität	22
Fehlertoleranz.....	22
Individualisierbarkeit	22
Lernförderlichkeit	22
Umsetzung II.....	23
MVVM.....	23
GUI Styling	23
Code-Coordination	24
Nhibernate.....	24
Autofac.....	24
Prism	25
Mahapps	25
Umsetzung III.....	25
DataRepository	25
Calculators	26
Ermittler.....	26
Tests.....	27
Fazit	28

Einleitung

Im Rahmen der beiden Module 226 und 120 haben wir den Auftrag erhalten in einer Gruppe eine Umsetzungsarbeit zu realisieren. Bei dieser Umsetzungsarbeit ging es darum, sein Können und Wissen in den beiden Modulen unter Beweis zu stellen.

Für uns war schon von Anfang an klar, dass wir, Rafael Estermann und Daniel Cavic, zusammen arbeiten und ein tolles Projekt realisieren möchten. Dazu haben wir in den letzten Wochen viel Zeit und Kreativität aufgebracht und das Produkt, zu welcher diese Dokumentation als Umschreibung dient, ist entstanden. „mogadu“.

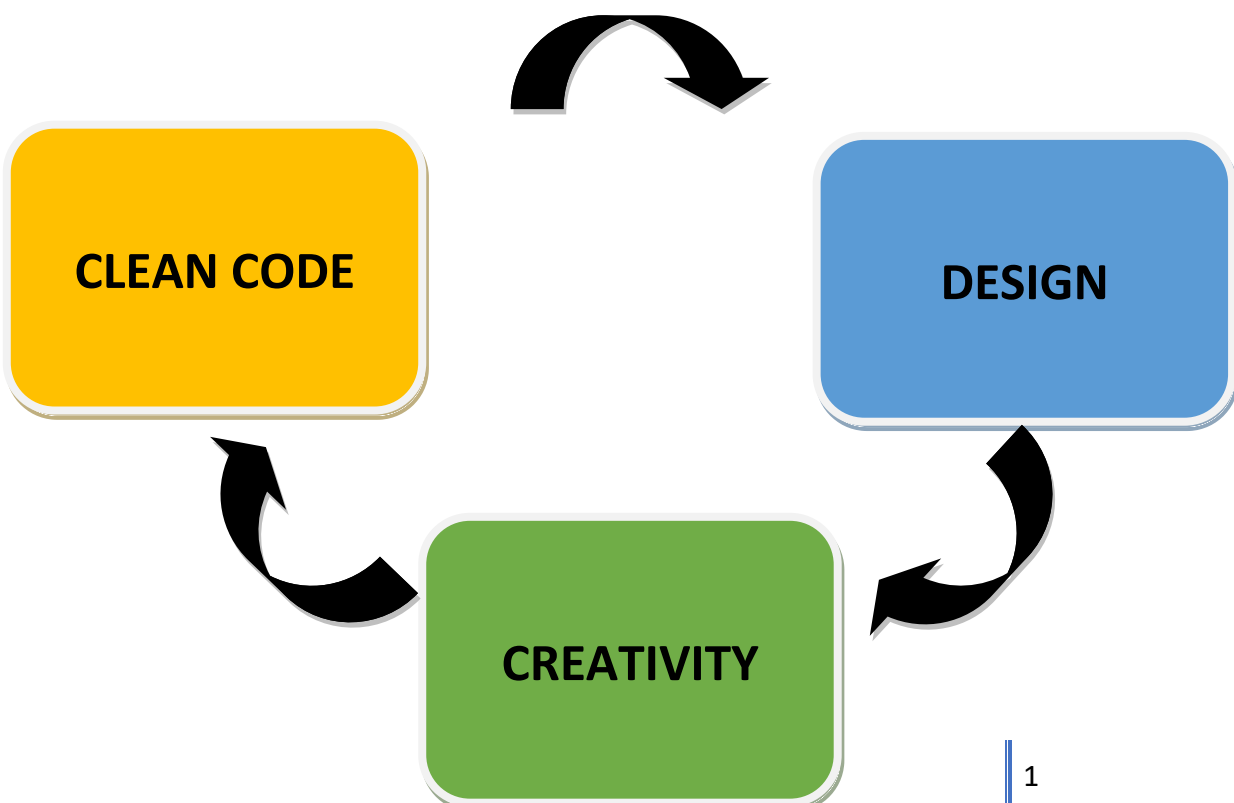
Ideenfindung

Für uns war klar, dass wir etwas Kreatives aufbauen wollen, was aber zugleich aus sehr nützlich ist. Da wir beide mit vielen Projekten zu tun haben und regelmässig um eine vereinfachte, kleine Projektübersicht froh wären, entschieden wir uns dazu „mogadu“ zu entwickeln.

„mogadu“ ist der Name einer Projektmanagement-Applikation. Die Applikation wertet Daten einer Firmen-Datenbank aus und stellt diese für Führungspersonen (wie zum Beispiel Teamleiter, Projektmanager oder auch Kadermitglieder) dar. „mogadu“ überzeugt durch eine einfache Bedienung, einen überschaubaren Möglichkeitsumfang und ein schönes, schlichtes Design.

Vorsätze

Grundlegen haben wir unsere Gedankengänge in der Form einer Grafik zusammengefasst. Während der Ideenfindung haben wir uns andauernd an dieser Grafik orientiert und unser Handeln / unsere Entscheidungen nach dem Inhalt dieser Skizze gerichtet.



Wir waren und von Anfang an über die Intentionen hinter diesem Projekt bewusst und filterten daraus drei wichtige Begrifflichkeiten heraus.

Zum einen wird der Code bewertet. Dabei wollten wir uns auf die Regeln des „**Clean Code**“ stützen. Sauberen Code möchte jeder Entwickler schreiben. Zum einen deswegen, damit er nach einer Weile noch die Belange seines Codes versteht, zum anderen, damit ein Kollege sich leicht in den Code einarbeiten kann. Zudem ist nebst der Art wie wir den Code schreiben auch der Einbezug vom MVVM (Model, View, View Model) Pattern unter diesen Begriff einzuteilen.

Ein Besonderes Augenmerk wollten wir auf das **Design** unserer Software (User Interface) legen. Uns sind die Intuition und das einfache Verständnis unserer Applikation sehr wichtig. Dies beginnt bei der Auswahl passender Controls, einer selbsterklärender Navigation bis hin zur Auswahl passender Farben.

Zu guter Letzt war für uns der Einsatz unserer **Kreativität** von hoher Bedeutung. Dieses Projekt sollt uns Spass machen, und lehren und uns für zukünftige Arbeiten (schulisch, geschäftlich oder auch privat) unterstützen. Wir wollten also auf eine Art kreative Ideen aufbringen, welche dem Benutzer ein anschliessendes „Wow-Gefühl“ bescheren.

Anforderungen

Um die Anforderungen unserer Applikation festzulegen, haben wir Userstorys erstellt. Diese dienen uns als Orientierung und auch als Zielüberprüfung des Projektes. Ausserdem geben Userstorys einen kleinen Einblick in die Funktion von unserer Software „mogadu“.

Login Fenster

Userstory

Als Person mit einer administrativen Teamrolle will ich mich mit meinem Namen und meinem Passwort in die Applikation einloggen und mich als Administrator verhalten.

Priorität

1

Akzeptanzkriterien

- Ein Login Screen wird beim Start geöffnet
- Eine Auswahl der Mitarbeiter kann getroffen werden (Combobox)
- Ein Passwort kann eingegeben werden
- Es kann sich eingeloggt werden
- Validierung wird angestoßen (bei falscher Eingabe des Passwortes)

Grobaufwand

2 Stunden

Übersicht

Userstory

Als Teamleiter will ich eine einfache Übersicht einsehen, welche mir Daten über mein Team, den heutigen Tag oder auch letzte Aktivitäten aufzeigt, damit ich auf einen Blick informiert bin.

Priorität

2

Akzeptanzkriterien

- Eine Kacheldarstellung zeigt mehrere Informationen an
- Informationen sind gut leserlich

Grobaufwand

4 Stunden

Teamansicht

Userstory

Als Teamleiter will ich eine spezielle Übersicht über mein Team einsehen, welche mir Daten über mein Team, die Teamauslastung, die Teammitglieder, ihre Positionen, ihr Salär, und ihr Eintrittsdatum liefert.

Priorität

2

Akzeptanzkriterien

- Auflistung der Teammitglieder
- Teaminformationen zu jeweiligen Team der eingeloggt Person erichtlich

Grobaufwand

3 Stunden

Navigation (Grundgerüst)

Userstory

Als Teamleiter will ich mehrere „Tabs“ auf einmal offen haben könne und zwischen den verschiedenen Tabs beliebig viele Male umherwechseln können, damit ich möglichst Produktiv arbeiten kann.

Priorität

1

Akzeptanzkriterien

- Auswahl der einzelnen Fenster
- Offene Fensterinstanzen werden entsprechend angezeigt

Grobaufwand

3.5 Stunden

Navigation (Erweiterung)

Userstory

Als Teamleiter will ich offene Tabs wieder schließen können, damit eine gewisse Übersichtlichkeit über die Applikation herrscht.

Priorität

3

Akzeptanzkriterien

- Schließen-Button pro Fensterinstanz
 - Nächstes Fenster wird bei der Schließung automatisch ausgewählt
-

Grobaufwand

1.5 Stunden

Auftragsansicht

Userstory

Als Teamleiter will ich alle Aufträge meines Team in einer groben Übersicht einsehen, damit ich allenfalls Schritte zur rechtzeitigen Auftragsabschliessung einleiten kann.

Priorität

1

Akzeptanzkriterien

- Fenster mit Auflistung aller Aufträge von meinem Team
 - Pro Auftrag Möglichkeit, genauere Detail einzulesen
 - Priorität, offene / abgeschlossene Aufgaben etc. wird angezeigt
-

Grobaufwand

2.5 Stunden

Auftragsdetailansicht

Userstory

Als Teamleiter will ich genauere Informationen zu einem Auftrag einlesen, damit ich auftragsspezifische Informationen erhalte und in meinem Team Aufgaben vergeben kann.

Priorität

1

Akzeptanzkriterien

- Fenster mit Auflistung aller Aufgaben (offen / geschlossen) pro Auftrag
 - Auftragsfortschritt wird angezeigt
-

Aufgabendetail

Userstory

Als Teamleiter will ich genauere Informationen zu einer Aufgabe einlesen.
Ich möchte sehen, wie viel Prozent der Aufgabe bereits erledigt wurde, von wem die Aufgabe erledigt wird und diese Daten auch verändern können, damit ich einen direkten Eingriff in die Aufgabenverteilung vornehmen

Priorität

1

Akzeptanzkriterien

- Mitarbeiter, der die Aufgabe ausführt, kann verändert werden
 - Aufgabenfortschritt kann verändert werden
 - Veränderungen können gespeichert werden
 - Informationen zur Aufgabe (Name, Beschreibung) werden angezeigt
-

Grobaufwand

3 Stunden

Personendetail

Userstory

Als Teamleiter möchte ich, während ich in die Aufgabenübersicht bin, direkt auf den Namen des Mitarbeiters klicken können und somit Informationen zu seiner Person einlesen können, damit ich nicht extra in die Teamübersicht muss.

Priorität

3

Akzeptanzkriterien

- Kleine Auflistung der Informationen über einen Mitarbeiter (Name, Funktion, Salär)

Responsivität

Userstory

Als Teamleiter möchte ich die Applikation im Vollbildmodus, oder auch im minimierter Fenstergrösse benützen können, damit ich mehrere Applikationen gleichzeitig auf meinem Bildschirm offen haben kann.

Priorität

3

Akzeptanzkriterien

- Alle Fenster (bis auf die Übersicht (Kachelproblem)) sind bis zu einem gewissen Grad responsiv und passen sich der Fenstergrösse an.

Nachdem wir nun die Grundanforderungen in Form von Userstories geschrieben haben, haben wir uns an die Planung des Projektes gemacht.

Planung I

Wir haben viel Zeit in die sorgfältige Planung unseres Projektes gesteckt. Speziell da wir in den kommenden Wochen sehr viele Projekte und Abschlussprüfungen schreiben und wir deshalb gezwungen sind, die Zeit möglichst gut zu Nutzen.

Paketerstellung

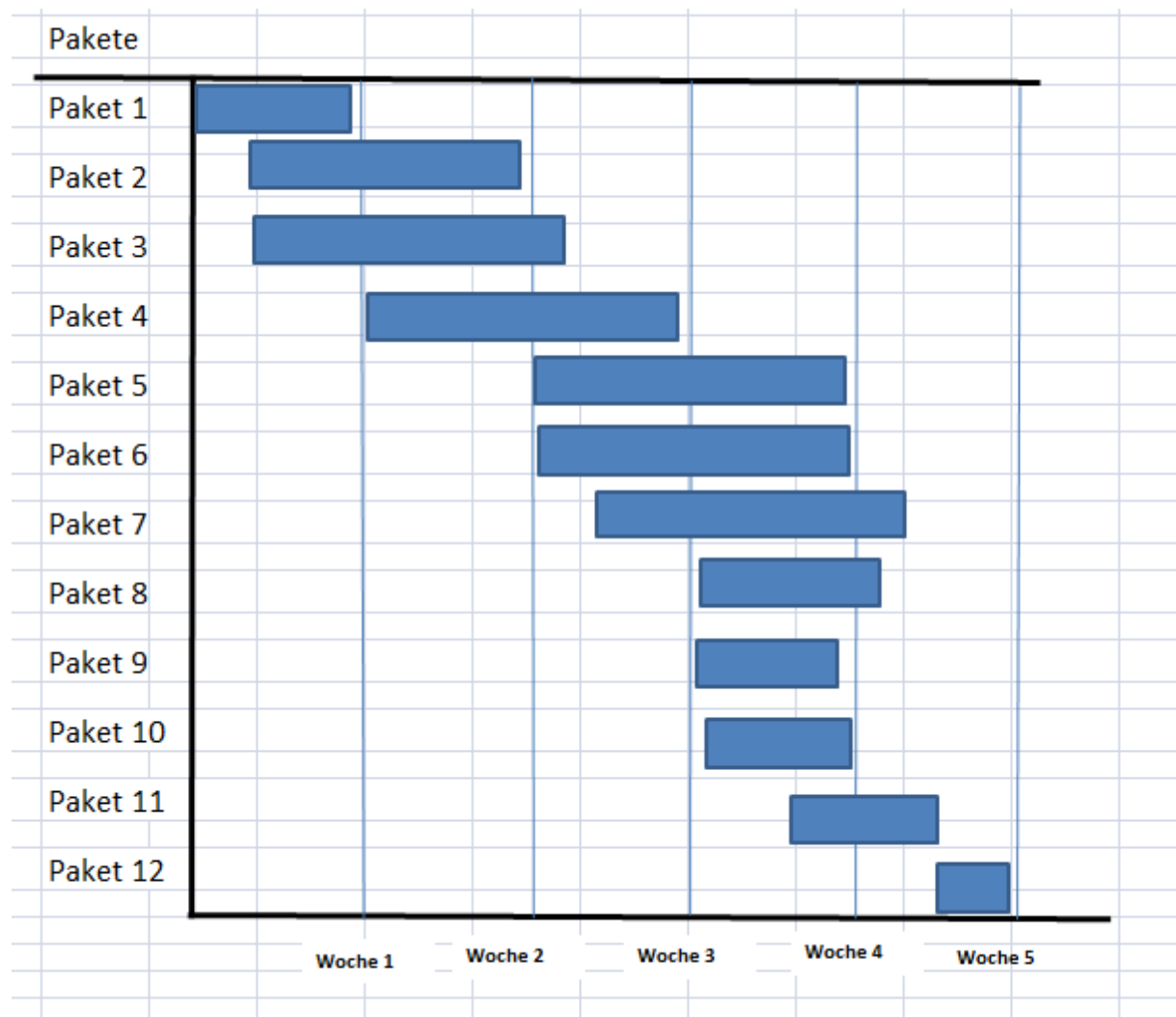
Als erstes haben wir uns Gedanken zu den verschiedenen Paketen, aus welchen die Realisierung von mogadu besteht, gemacht. Dazu haben wir eine Tabelle erstellt, bei der wir jeweils das Paket, die damit verbundenen Aufgaben und ebenfalls wer vom Projektteam das Paket erledigt zusammengefasst.

Aufgabengebiet	Umschreibung	Person
1. Datenbank erstellen	Es muss eine Datenbank gemietet werden, sich Gedanken zu den Tabellen und Schlüsseln gemacht werden und sich über OR-Mapper informiert werden.	Daniel
2. Datenbank befüllen	Es müssen realistische Testdaten erstellt werden, welche anschliessend für Entwicklertests benutzt werden.	Daniel

3. <i>Navigation erstellen</i>	Es wird eine Navigation (ein Konzept) erstellt, welches das Öffnen mehrerer Tabs ermöglicht (und das umherschalten).	Rafael
4. <i>Design</i>	Es wird ein einheitliches und schönes Design (mit Farben, Controls und Verhaltensmustern) erstellt und entsprechend das andere Teammitglied darauf hingewiesen	Daniel
5. <i>Erstellen des Hauptfensters (Design)</i>	Es wird mit UserControl Platzhaltern ein Hauptfenster erstellt. Die erstellte Navigation wird darin eingebettet.	Daniel
6. <i>Datenbank Repository erstellen</i>	Es werden mit dem OR-Mapper und den Mapping Files mehrere Query Services geschrieben, welche Daten von der Datenbank abfragen. (Erstellung Entities und ExpandedEntities)	Rafael
7. <i>Views erstellen (XAML)</i>	Die verschiedenen Views werden mit XAML Code erstellt und die dazugehörigen ViewModels werden ebenfalls erstellt (noch nicht implementiert)	Daniel
8. ViewModels erstellen	<i>Die ViewModels werden implementiert (anhand Services) und der Xaml Code wird mit der Hilfe von DataBindings befüllt</i>	Rafael
9. Businesslogik implementieren	<i>Die Businesskomponenten (welche aus den ViewModels benötigt werden) werden erstellt und ausprogrammiert.</i>	Rafael

10. Testen	<i>Die Applikation wird entsprechend auf ihre Funktionalität getestet</i>	Daniel & Rafael
11. Validieren	<i>Die Applikation wird so programmiert, dass keine Validierungen vorgenommen werden müssen.</i>	Rafael
12. Dokumentation & Abschluss	<i>Die Dokumentation wird entsprechend ergänzt und das Projekt wird abgeschlossen</i>	Daniel

Zeitplan



Wir haben einen groben Zeitplan erstellt, um unsere Zeitressourcen einzuteilen und eine gewisse Kontrolle zu haben, ob wir dem Zeitplan gemäss agieren.

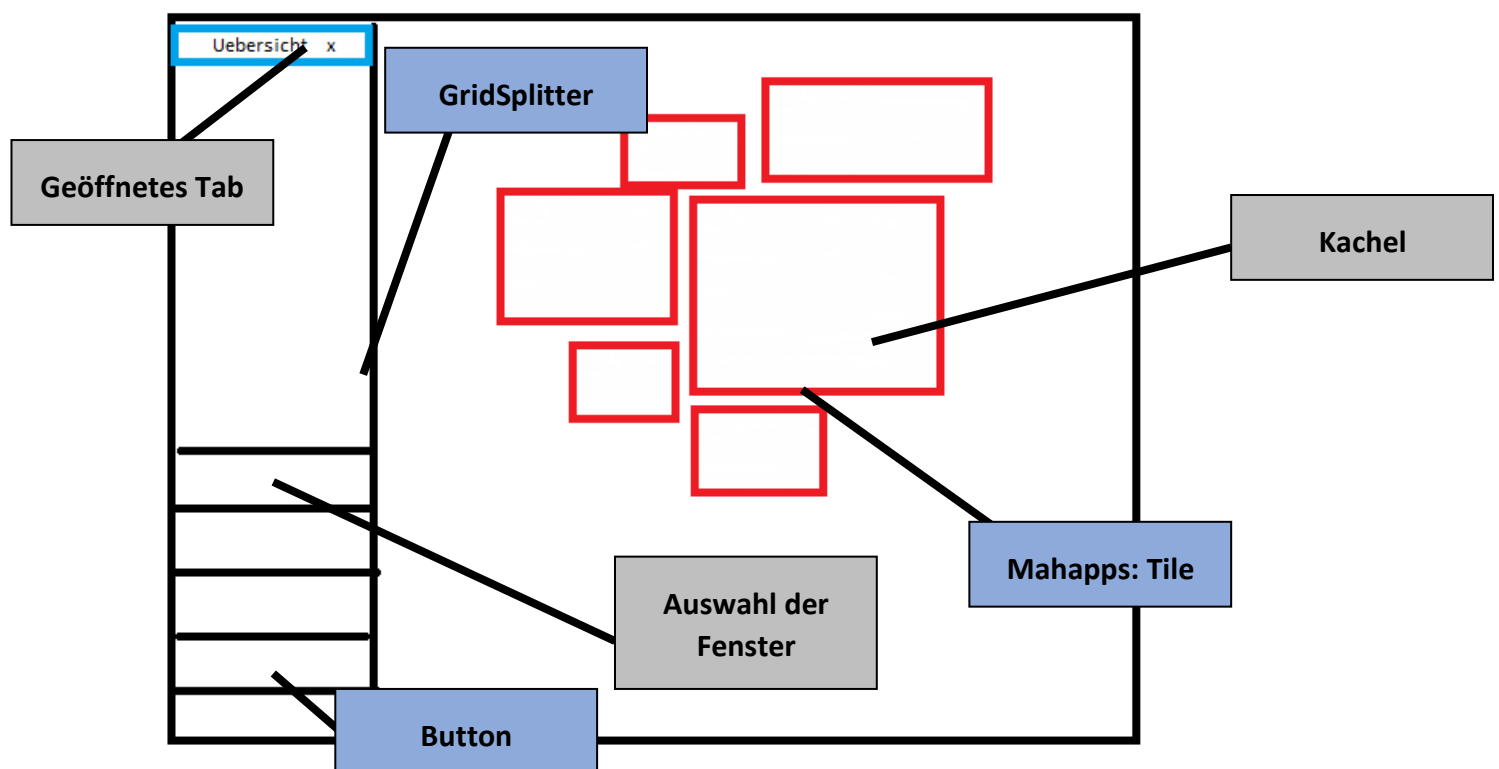
Planung II

Im zweiten Schritt der Planung hat wir ein grobes Klassendiagramm erstellt und dazu ebenfalls ein GUI-Entwurf gemacht. Dies waren anfängliche Überlegungen, welche wir später ebenfalls in die Praktik umsetzen konnten.

GUI Entwurf

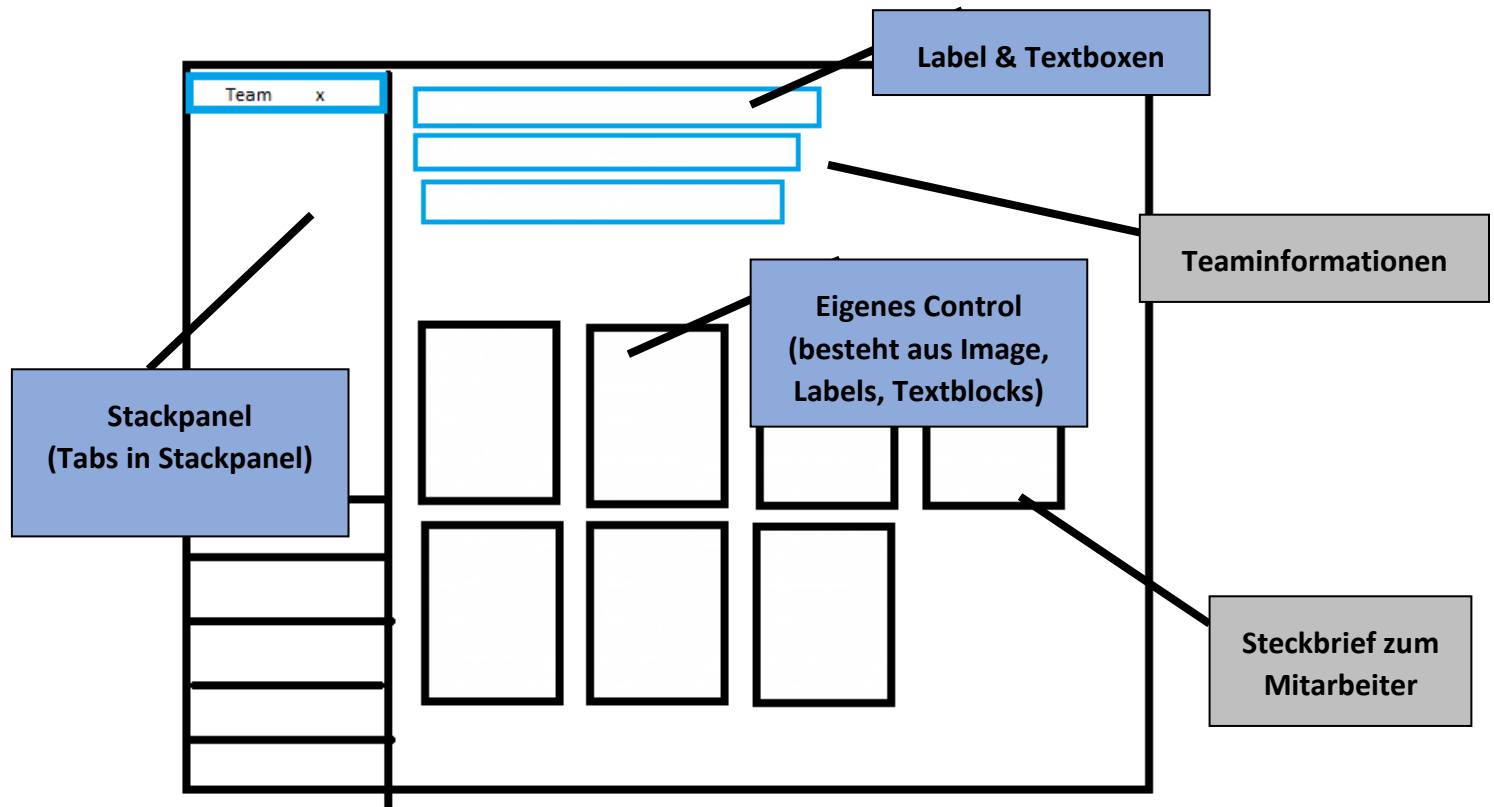
Für die Übersicht haben wir den folgenden GUI Entwurf vorbereitet (MockUp).

Die roten Vierecke stehen dabei für sogenannte Kacheln, welche die bereits in den Userstories beschriebenen Informationen beinhalten. Dies ist eine kreative

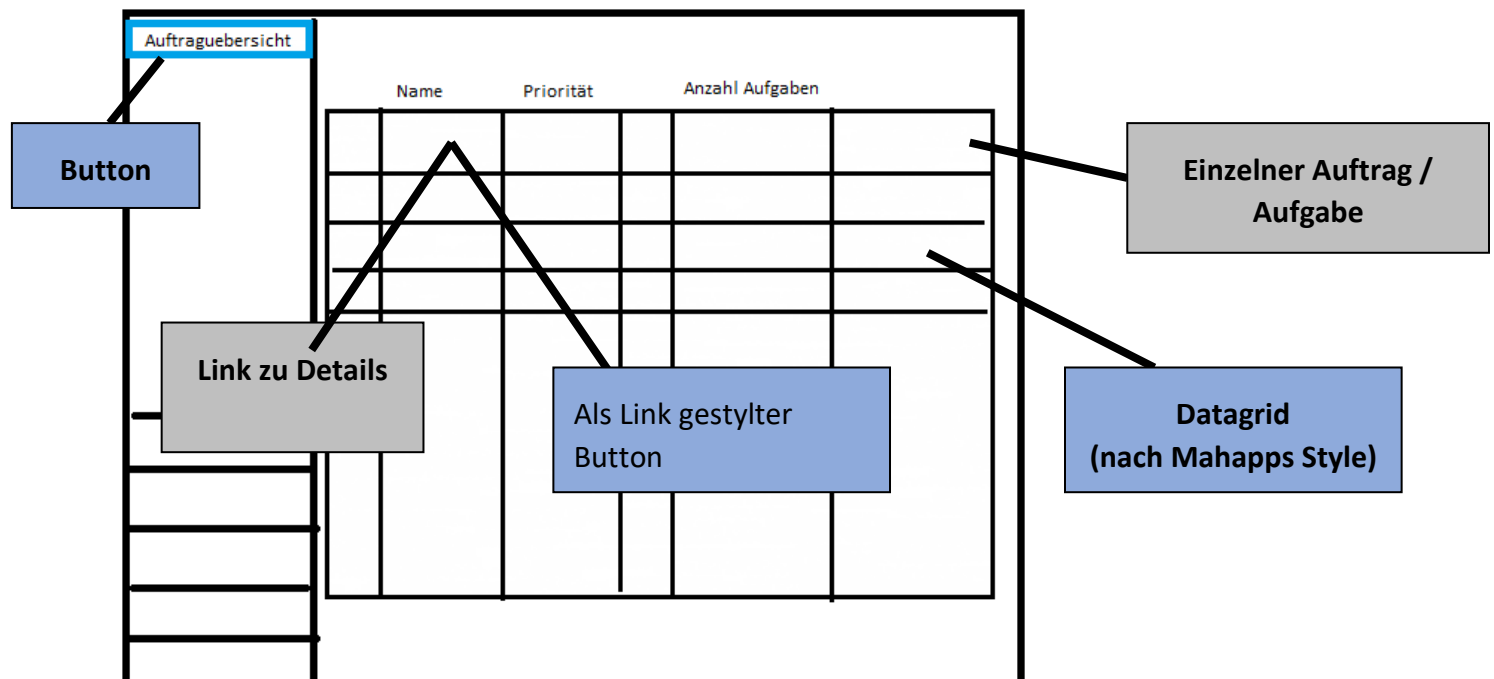


Darstellungsweise, welche immer mehr in modernen ERP-Applikationen verwendet werden. Hier ist ebenfalls gut ersichtlich, wie wir uns die Navigation vorgestellt haben. Nämlich planen wir, auf der linken Seite eine Auswahl an Fenstern zu machen, welche bei einem „Klick“ zu einer geöffneten Instanz werden und als Tab ersichtlich werden. Die Tabs und ihre Werte werden gespeichert, so dass zwischen verschiedenen Tabs um hergewechselt werden kann.

Zusätzlich planen wir für die Teamübersicht eine Art Dashboard, welche in Steckbriefform für jeden Mitarbeiter Informationen bereitstellt (siehe nächste Grafik). Dazu soll der blau Markierte Bereich Informationen zum Team selber bieten.

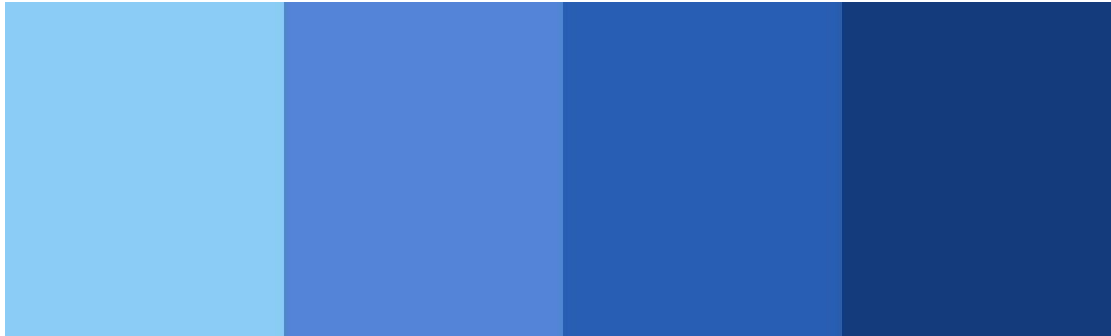


Zu guter Letzt haben wir noch ein GUI Entwurf für die Auftragsübersicht erstellt. Sehr ähnlich planen wir ebenfalls die Aufgabenübersicht zu erstellen (deshalb nur ein Entwurf). Die beiden Ansichten unterscheiden sich nur in den Datagridspalten Spalten.



Anmerkung zum GUI

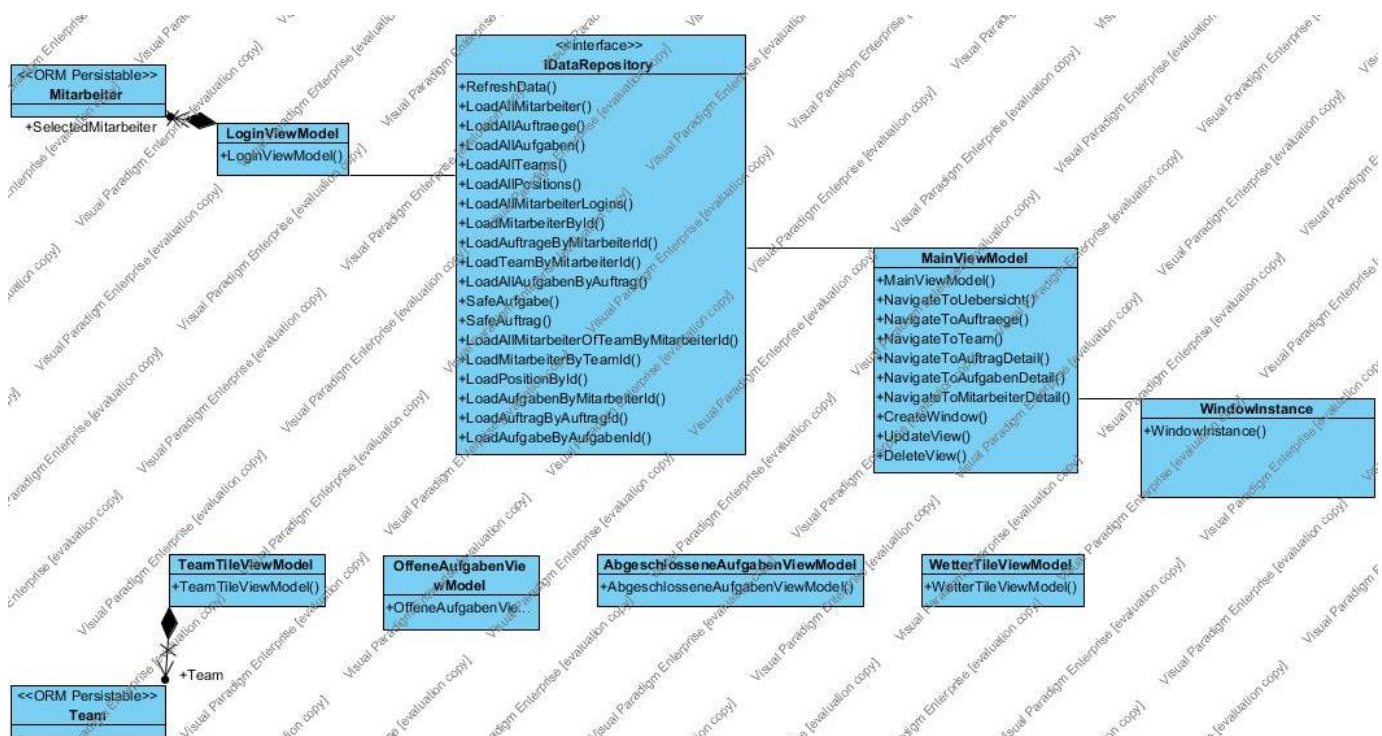
Wir haben uns dazu entschieden mit der Farbe „Blau“ zu spielen. Diese wollen wir als Akzent setzen. Wir haben beispielsweise das folgende Farbschema generiert. Von diesen Farben planen wir speziell viel Gebrauch zu machen (sei es bei Ränderfarben, Hintergrundfarben oder nur bei MouseOver Events).



Dazu ist anzumerken, dass wir bereits im Voraus wussten, dass wir eine externe Library namens „Mahapps Metro“ verwenden werden. Obwohl wir gerne eigene Styles erschaffen, unterstützt die Library in Sachen Styling und bietet eigene Controls dazu.

Die Recherchen zu der Library, sowie auch das Erlernen der Verwendung zählt zu einer unseren Wissensanbauten während es Projektes. Die Dokumentation zu Library findet man unter dem folgenden Link: <https://mahapps.com/>

Klassendiagramm

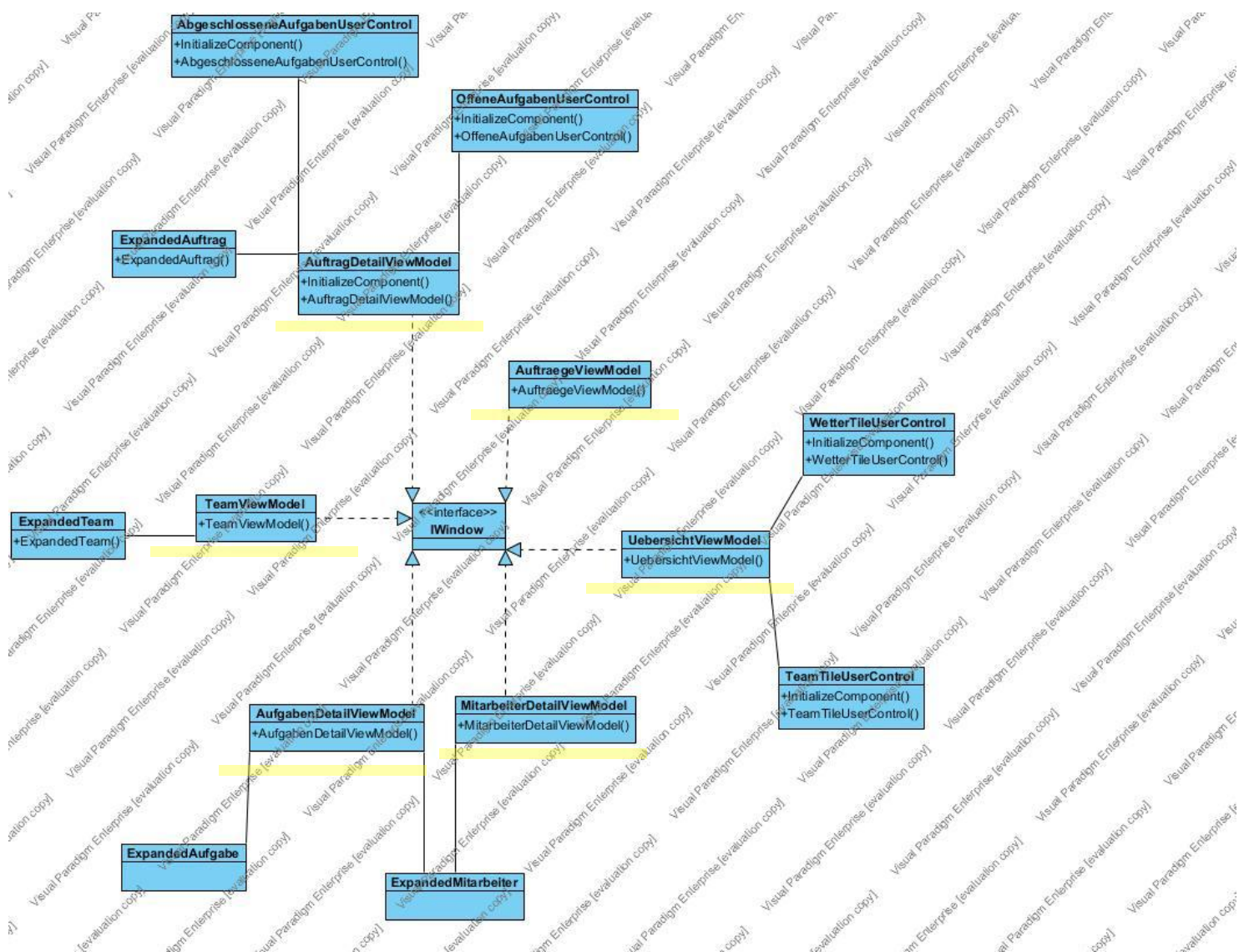


Im Zentrum steht das DataRepository, für das wir das Interface IDataRepository erstellt haben. Das DataRepository stellt eine Verbindung mit der Datenbank her und holt anschliessend Informationen. Ebenfalls speichert (updated) das DataRepository Datensätze. Es wird nur vom MainViewModel gebraucht, welches anschliessend die Instanz an alle anderen UserControl Instanzen weiterreicht.

Anfänglich werden die Daten initial geladen. Die gesamte Laufzeit wird immer auf die von am Anfang geladenen Datensätze zugegriffen. Zu einer Aktualisierung kann man die Methode „RefreshData()“ aufrufen.

Das MainViewModel ist der Kern der Applikation. Es beinhaltet eine Liste der geöffneten Window-Instanzen (alle vom Typ WindowInstance). Dazu enthält das MainViewModel die drei wichtigen Funktionen:

- CreateWindow (neue Instanz eines Fensters)
- UpdateView (wenn zwischen einem Tab gewechselt wird)
- DeleteView (wenn ein Tab gelöscht wird)



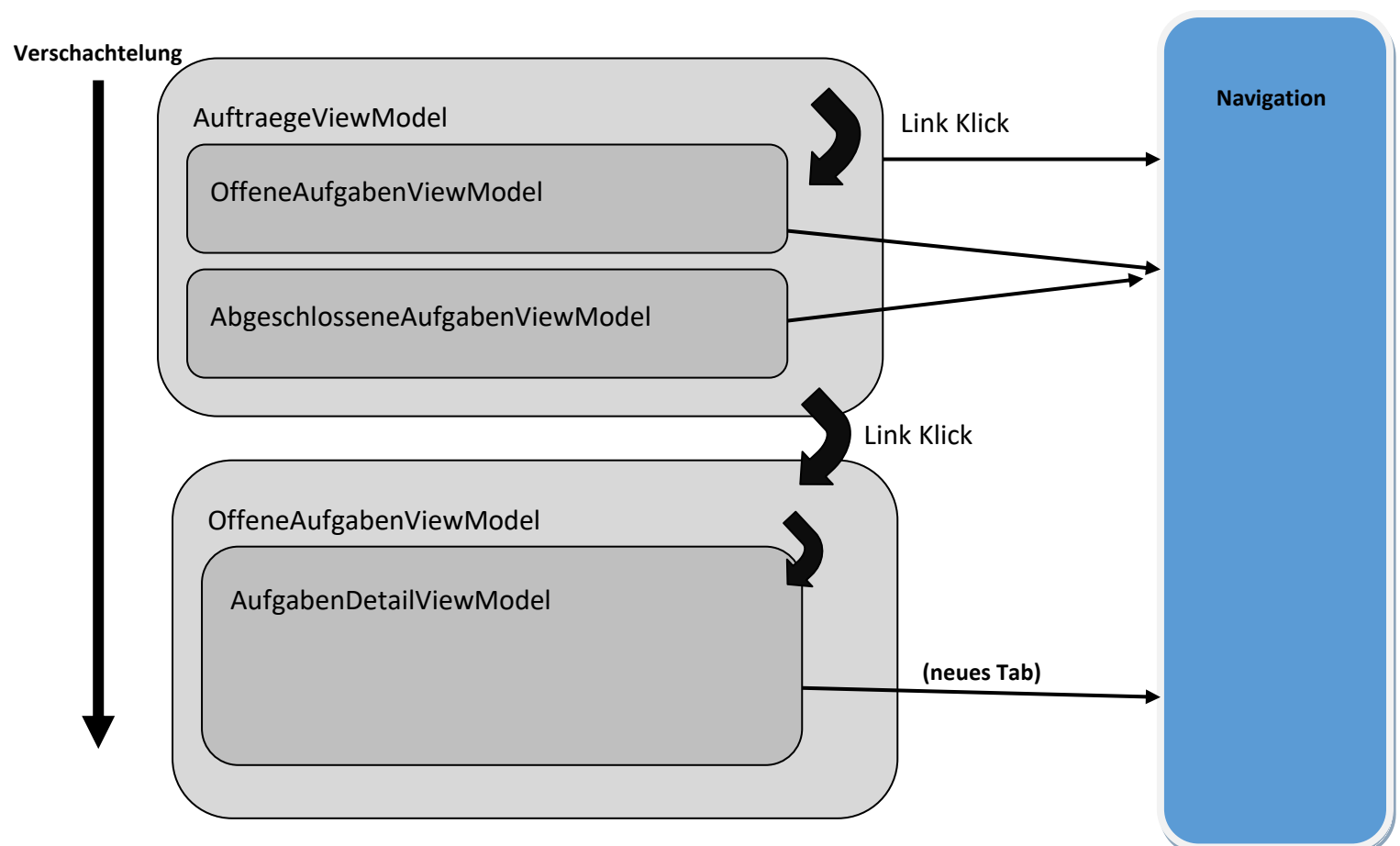
Wichtige Klassen (ViewModels) sind die folgenden:

- TeamViewModel
- UebersichtViewModel
- MitarbeiterDetailViewModel
- AufgabenDetailViewModel
- AuftraegeViewModel
- AuftragDetailViewModel

Zu diesen ViewModels existiert jeweils ein dazugehöriges UserControl. Alle diese ViewModels leiten vom Interface **IWindow** ab. Dieses stellt sicher, dass alle instanzierbaren (als Tab zu gebrauchenden) ViewModels entsprechend markiert sind.

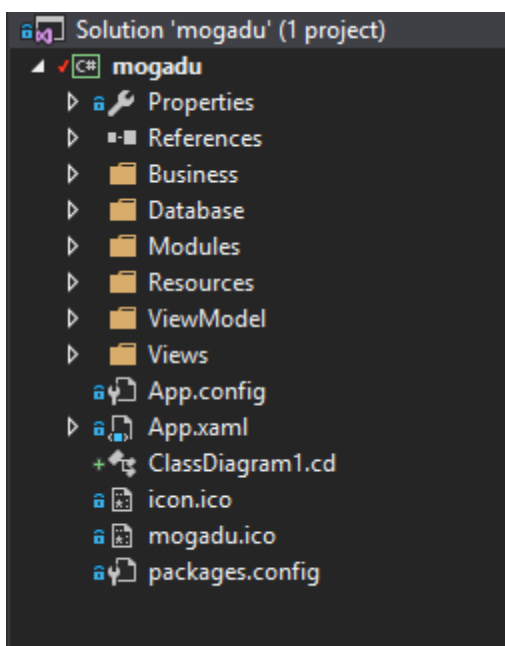
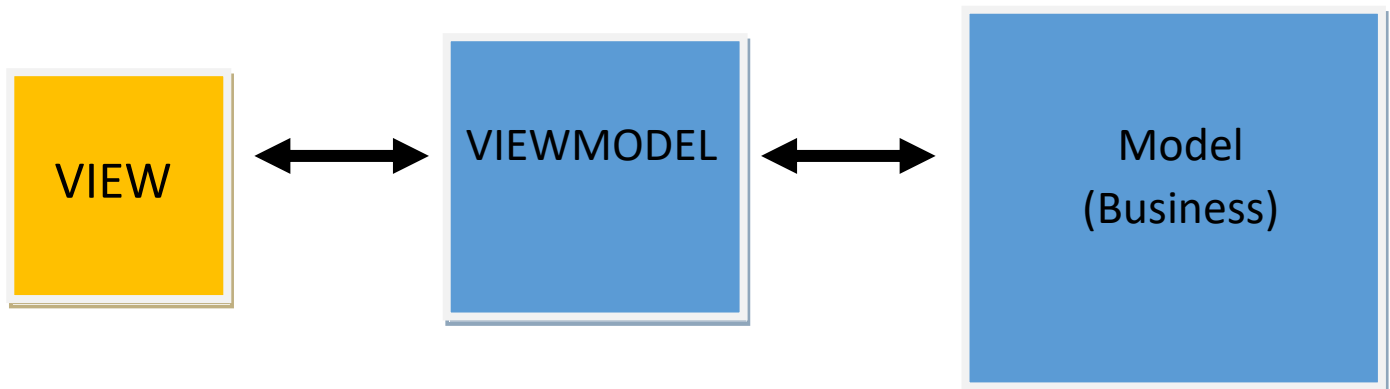
Ebenfalls ist zu erwähnen, dass einige ViewModels in einander verschachtelt sind. Beispielsweise öffnet man zuerst das AuftraegeViewModel, um anschliessend auf einen Auftrag zu klicken. Danach öffnet sich das AuftragDetailViewModel. Dort sieht man anschliessend (wie im UML Diagramm gezeigt -> OffeneAufgabenViewModel / AbgeschlosseneAufgabenViewModel) alle Aufgaben (offen oder abgeschlossen). Klickt man auf eine Aufgabe, so öffnet sich schliesslich das AufgabenDetailViewModel.

Dies ist auch der Grund, weshalb von der Navigation her nur 3 Views (Übersicht, Team, Aufträge) geöffnet werden könne, jedoch 6 ViewModels vom Interface IWindow ableiten.



Klassendiagramm (nach MVVM)

Wir haben uns dafür entschieden nach dem MVVM (Model View ViewModel) Pattern zu arbeiten. Dies ist für uns beide etwas neues. Obwohl Rafael bereits einige Erfahrungen sammeln durfte, ist es für beide Projektteilnehmer sicherlich eine Herausforderung. Das Pattern schreibt regelrecht vor, dass keine Businesslogik in den Viewmodels vorhanden sein darf. Auf dies nahmen wir bei der Planung unseres Codes Rücksicht.



Wir haben zu Beginn eine Ordnerstruktur angelegt, welche uns bei der Implementierung des MVVM-Patterns unterstützen sollte. Im **Views Ordner** sind somit alle Fenster, also alle XAML Files. Dabei benutzen wir sehr viele UserControls. Das Benutzen dieser UserControls verschaffte uns den Vorteil der Wiederverwendbarkeit.

Im **ViewModels Ordner** sind alle ViewModels abgelegt. Zusätzlich zu den ViewModels existiert ebenfalls ein **Windowhandling Ordner**, in dem eine Klasse und ein Interface zur Charakterisierung einer neu Instanzierbaren View (vom Menü aus) ist. Die gesamte Logik (mehrheitlich Daten aus der Datenbank laden / schreiben) und irgendwelche Werte berechnen, befindet sich im Business (nicht im ViewModel!)

Zusätzlich existiert noch ein **Database Ordner**, welche die ganze Nhibernate Thematik (siehe Kapitel Nhibernate) abhandelt und Entities enthält. Dazu haben wir noch ein **Resources Ordner** für die ganzen Bilder und Styles angelegt. Im **Modules Ordner** wäre anschliessend noch die Registrierung der Interfaces gekommen, welche wir aus Zeitgründen nicht implementieren konnten.

Wie im Klassendiagramm gezeigt, ist die Trennung dieser einzelnen Gebiete streng erforderlich. Nur so konnten wir unsere Navigation implementieren, oder ein DataRepository erstellen und gebrauchen.

Umsetzung I

Übersicht

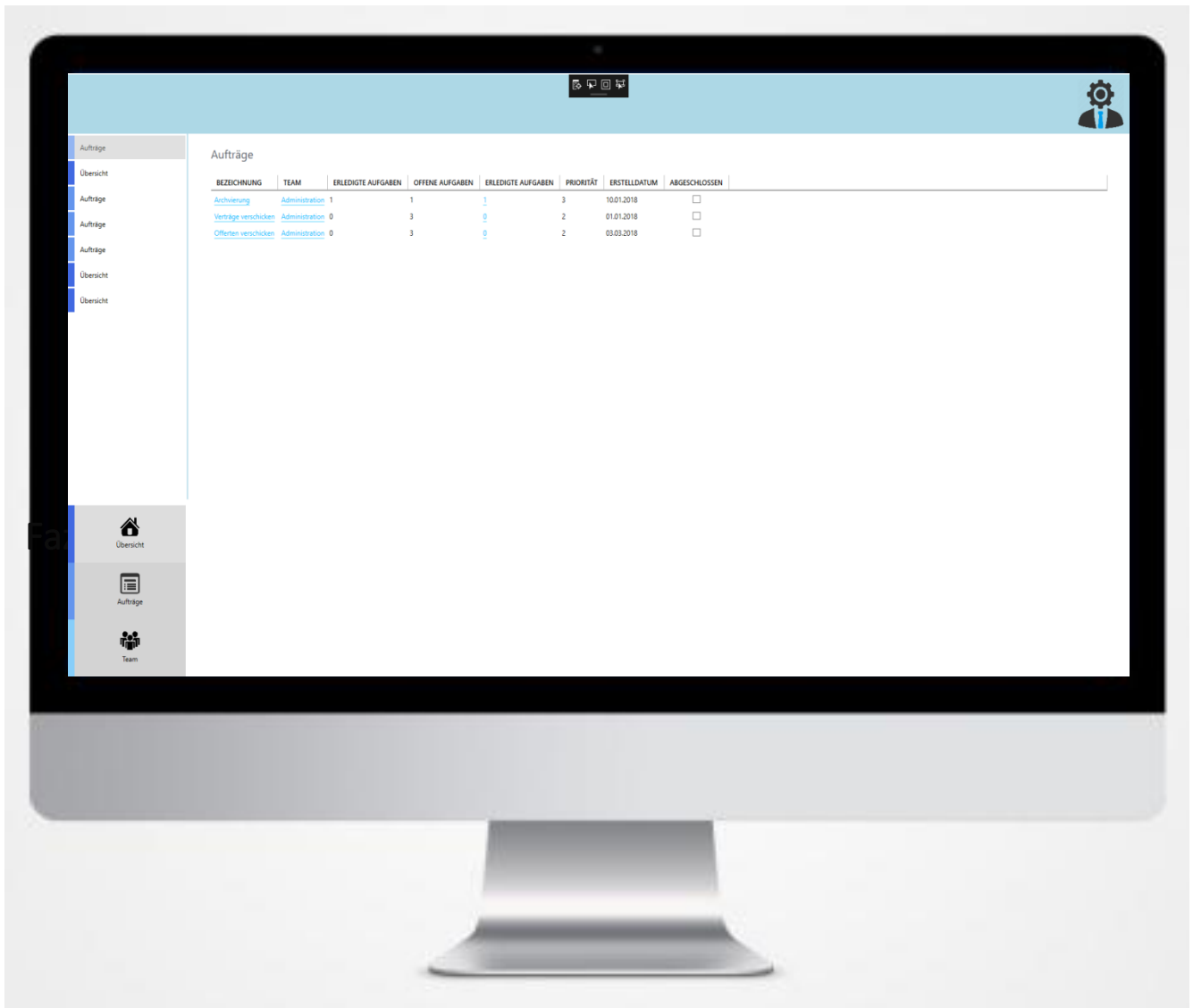


Bedienkonzept

Sobald man sich eingeloggt hat, erscheint die Übersicht (Tab wird automatisch geöffnet). Die Übersicht bietet mehrere Informationen, welche für den Start in den Arbeitstag nützlich sein können. Zum einen wird das Team, in dem sich der mogadu-Nutzer befindet, angezeigt. Der heutige Wochentag sowie auch der heutige Monat wird ebenfalls ersichtlich. Der gekürzte Mitarbeiter des Monats, welcher errechnet wird, das aktuelle Wetter und auch der Firmenname ist ersichtlich. Zudem ist in der Neuigkeiten-Kachel eine Neuigkeit ersichtlich (letzter Abschluss eines Auftrages (firmenweit)).

Auch hier wurde auf die Farbe Blau geachtet. Beim MouseOver über eine Kachel, verändert sich ihre Grösse, was der ganzen View einen ziemlich lebendigen Charakter verleiht und als super Einstiegspunkt in die Software dient.

Aufträge

**Bedienkonzept**

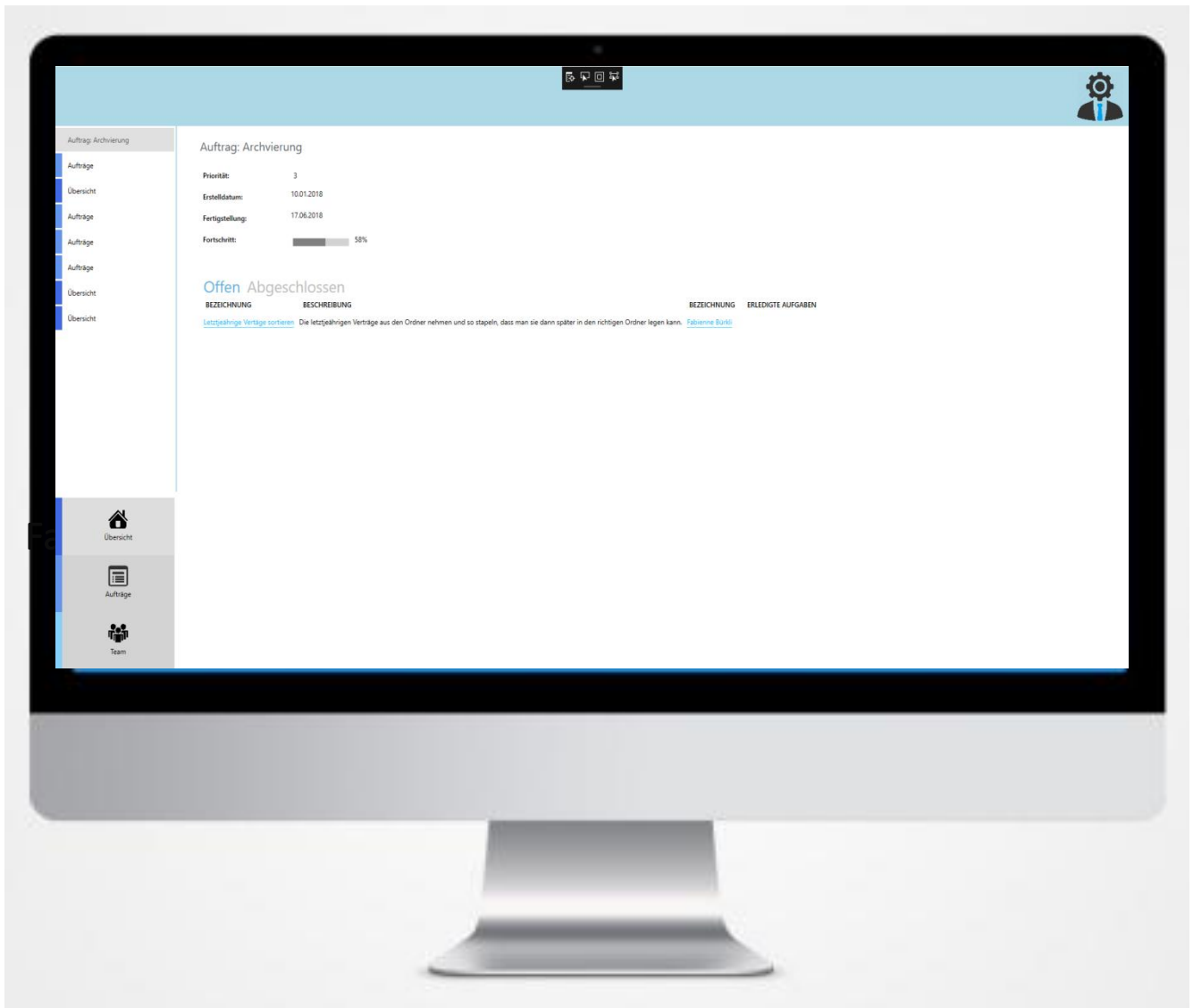
Die Aufträge View zeigt eine Tabelle, welche alle Aufträge des Teams, indem der angemeldete User ist, anzeigt. Dies gilt nur als kleine Übersicht. Die Bezeichnung, das Team, sowie auch die offenen und erledigten Aufgaben (Anzahl) werden angezeigt. Zudem auch die Priorität eines Auftrages, das Erstelldatum und ob er bereits abgeschlossen ist, oder nicht.

Sehr schön ist hier der Vorteil der UserControl und unserer Umsetzung der Navigation zu betrachten. Wenn man nämlich auf den Teamnamen klickt, öffnet sich anschliessend direkt die TeamView. Dies bedeutet, dass man auf mehrere Wege eine View öffnen kann.

Wenn man jedoch mehr Informationen zu einem Auftrag haben möchte, dann kann man auf die Auftragsbezeichnung klicken (ist immer mit einem hellblauen Link gelöst).

Anschliessend wird in die Auftragsdetail View navigiert und ein neues Tab öffnet sich.

Auftragsdetail



Bedienkonzept

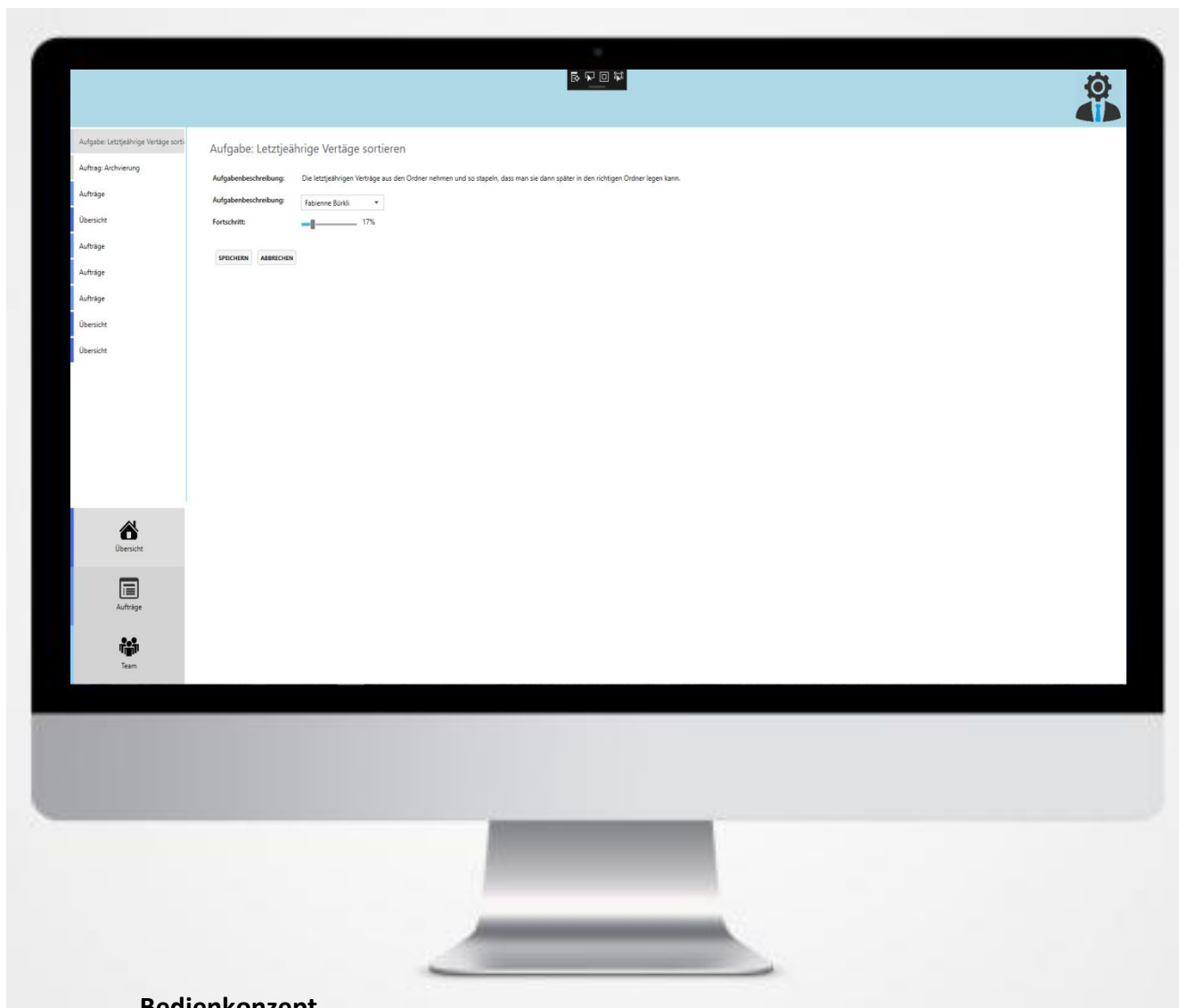
Die Auftragsdetail View bietet eine genauere Übersicht zu einem Auftrag. Auch hier wird wieder die Priorität das Erstelldatum und auch (falls vorhanden) das Fertigstelldatum angezeigt.

Wichtig: das Fertigstelldatum ist nur gesetzt, wenn alle Aufgaben eines Auftrages gelöst sind

Ebenfalls ist der Fortschritt des Auftrages einzulesen. Dieser wird durch eine Komponente im Business berechnet. (Anhand der Prozentualen Fertigstellungstatus der einzelnen Aufgaben).

Im unteren Bereich der View sind anschliessend die Aufgaben (unterteilt in offen und abgeschlossene) zu sehen. Auch dort sind die Informationen (gleich wie bei der Auftragsübersicht) nur oberflächlich angezeigt. Möchte ein mogadu-User beispielsweise Wissen, was für eine Person den Auftrag gerade bearbeitet, dann kann auch hier auf den Namen der Person gedrückt werden, um Informationen zur Person zu sehen. Anderenfalls kann über den Aufgabennamen eine Hierarchie weiter nach unten navigiert werden.

Aufgabendetail



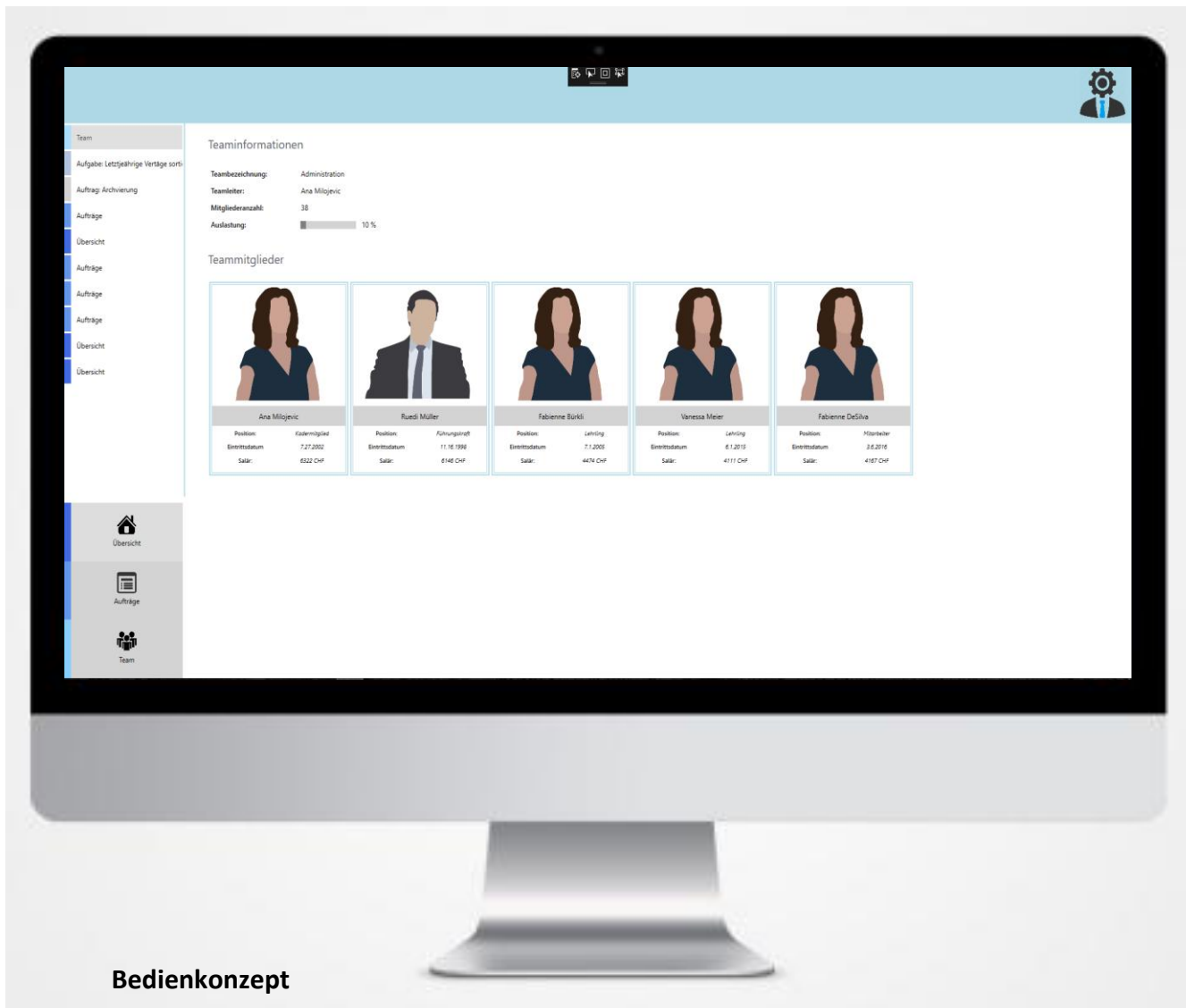
Bedienkonzept

Die Aufgabendetail View ist die einzige View, bei der man auch Daten mutieren kann. Hier kann nämlich für eine Aufgabe ein Mitarbeiter, der die Aufgabe anschliessend lösen muss, ausgewählt werden. Ebenfalls kann ein Teamleiter beispielsweise den Prozentualenstatus (mit einem Slider) verändern. Durch den Speichern und Abbrechen Button kann man anschliessend die gemachten Änderungen in die Datenbank persistieren.

Wichtig: ändert man den Prozentualenstatus auf 100%, wird beim Speichern geprüft ob alle anderen Aufgaben ebenfalls schon fertig gemacht wurden. Falls ja, wird das Fertigstellungsdatum des Auftrages gesetzt und der Auftrag ist abgeschlossen.

Wissen, was für eine Person den Auftrag gerade bearbeitet, dann kann auch hier auf den Namen der Person gedrückt werden, um Informationen zur Person zu sehen. Anderenfalls kann über den Aufgabennamen eine Hierarchie weiter nach unten navigiert werden.

Team



Bedienkonzept

Die Team View schafft eine Übersicht über das Team, indem sich ein mogadu-User befindet. Es wird nebst dem Teamnamen, auch die Anzahl Teammitglieder, der Teamleiter, und die Teamauslastung (wird durch Businesskomponente berechnet) angezeigt.

Dazu werden die einzelnen Teammitglieder und ihre persönlichen Daten (Position, Eintrittsdatum und auch der Salär) angezeigt.

Tipp: Der Salär wird mit der Hilfe einer Businesskomponente berechnet, welche anhand der Position des Mitarbeiters und seiner Arbeitszeit bei der Firma eine Summe zurückgibt.

Navigation

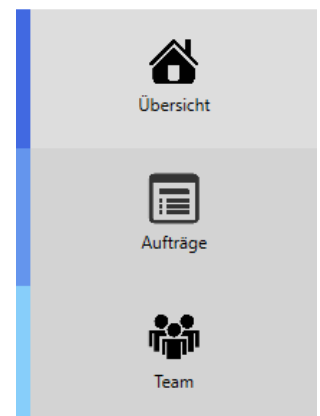
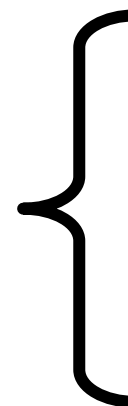
Auf unsere Navigation sind wir besonders Stolz (es gab entsprechend viel zu tun für die Umsetzung der Navigation). Hier wollten wir kurz anmerken, dass pro Klick einer der drei Buttons (Übersicht, Aufträge, Team) ein neues Tab, mit der gleichen Randfarbe (-> dient zur besseren Orientierung) geöffnet wird.

Beim MouseOver über eines der Tabs, verändert sich jeweils der Rand. Ein Angewähltes Tab ist jeweils ein bisschen ausgegraut und entsprechend markiert.

Beim Mouse Over über ein Tab, wird ebenfalls der Löschen Button sichtbar, mit dem man ein Tab aus der Liste der geöffneten Tabs entfernen kann. Ein Algorithmus errechnet anschliessend, welches Tab als nächstes (fall noch welche vorhanden sind) angewählt wird.



Buttons zum öffnen neuer Tabs



Worte zum GUI

Wir sind mit der Umsetzung von unserem GUI sehr zufrieden. Laut Rückmeldungen mach es einen professionellen und erfrischenden Eindruck, was durchaus auch unser Ziel war. Der Einbezug der Mahapps Metro Controls und Styles unterstütze uns dabei. Unsere Applikation ist eine Darstellungsapplikation, welche Daten darstellt und nicht primär zur Veränderung von Daten gedackt ist. Deshalb beschränkten wir uns auf die doch sehr Vielfältige Nutzung von Grids, UserControls, Buttons, Labels, Stackpanels, Tabcontrols, Progressbars, Wrappanels, Datagrids,, Link-Buttons, Comboboxes, Sliders, Checkboxes, Images und Tiles.

GUI Ergonomie

Wir haben unsere Software so entwickelt, dass sie den ergonomischen Grundsätzen entspricht. Dazu haben wir uns an den Block 1 des Modules 120 gehalten.

Aufgabenangemessenheit

Wir haben versucht nicht zu viele Informationen auf unsere Views zu legen. Dies haben wir zum einen mit dem gebrauch von Detail UserControls gemacht (je weiter hinein man navigiert, desto detaillierter wird die Ansicht). Andererseits benutzen wir beispielsweise Platzsparende Controls (TabControl).

Dies ermöglichte uns, relativ schmale Views zu haben, welche trotzdem informativ sind und dem User ein überschaubares UI bieten. Zusätzlich verwendeten wir bei der Auswahl der Mitarbeiter als Vorgabewerte jeweils Comboboxen. Bei der Aufgabedetailmaske, ist beispielsweise auch der Tabindex (so auch im Login Fenster) definiert. Die Abstände zwischen Informationen sind über die gesamte Applikation hin dieselben (immer gleiche Rowdefinitions und Columndefinitions. Genau so die Farben der Titlen und Schriftgrößen der Informationen.

Selbstbeschreibungsfähigkeit

Da es sich bei mogadu um keine Applikation handelt, die für die Daten Mutation gemacht wurde (lediglich Darstellung von Informationen), haben wir versucht, die einzelnen Views sinnvoll zu beschriften (sinnvolle Namensgebung). Ebenfalls haben wir, damit dies dem User klar ist, beim öffnen eines Neuen Tabs, welches man nicht direkt Instanzieren kann vom Menü aus, immer zuerst die Art des Tabs (Aufgabe:, Auftrag:) spezifiziert und im Anschluss, um welchen Auftrag / Aufgabe es sich handelt.

Steuerbarkeit

Unsere Applikation ist insofern Steuerbar, da man einerseits mehrere Tabs offen haben kann und steuern kann, welches aktiv ist, und welche man löschen möchte. Andererseits kann man auch über verschiedene Wege an Informationen gelangen (Beispiel: Teamview über den Menüpunkt Team, oder über die Auftragsübersicht).

Zudem kann man durch das Verändern von Daten (beispielsweise Fortschrittsanpassung Aufgaben) direkten Einfluss auf das Programm nehmen, da somit auch der Auftrag verändert wird.

Erwartungskonformität

Wir haben immer versucht gleiche Schriftarten, gleiche Schriftfarbe, gleiche Zeilen abstände und vor allem ein gleiches Verhalten zu programmieren.

Beispielsweise wird für jeder Menüpunkt (bei Klick) ein neues Tab geöffnet. Zudem wird beim Löschen eines Tabs immer das obere Tab automatisch angewählt (dies ist normalerweise der Fall). Beim Hinzufügen des Tabs, erscheint es immer zuoberst in der List. Angewählte Tabs sind jeweils ausgegraut.

Diese Tabs, welche man nicht von Menü aus öffnen kann, haben entsprechend einen andersfarbigen Border (nicht bläulich), was ihre Eigenheit symbolisieren soll.

Die Controls, welche wir verwendeten, waren zudem auch immer gleich. Die Comboboxen und Buttons waren jeweils von Mahapps, bei Auflistungen benutzen wir immer Stackpanels oder Datagrids, jede View hat einen Titel, im oberen Bereich der View etc.

Fehlertoleranz

Unsere Software, und das ist der Vorteil einer Darstellungsapplikation, ist sehr fehlertolerant. Vorallem weil Fehler nur an zwei Orten passieren können. Zum einen können Fehler beim Login passieren (falsches Passwort) oder auch bei der Mutation der Aufgabendetails.

Bei beiden Orten haben wir eine Validierung (Warnungsicon) eingebaut. Ansonsten ist unsere Software sehr fehlertolerant.

Individualisierbarkeit

Mogadu ist in dem Sinne individualisierbar, da man selber individuell diese Tabs öffnen kann, die man möchte. Ebenfalls kann man aktiven Einfluss auf Auftragsauslastungen, Teamauslastungen und Aufgabenfortschritte nehmen.

Das GUI selber nach seinen Wünschen und Vorzügen gestalten, ist stand heute noch nicht möglich. Bei mehr Zeit wäre dies aber definitiv eines unserer Ziele gewesen.

Lernförderlichkeit

Da unsere Applikation sich keine Werte selber merken muss (durch Benutzereingaben) und sie sehr einfach gehalten ist (zumindest vom GUI Verständnis her), entschieden wir uns keine Hilfe einzubauen oder eine künstliche Intelligenz zu gebrauchen.

Umsetzung II

Während unserer Projektarbeit haben wir uns in viele Themen eingearbeitet. Dies war nicht immer einfach für uns. Wir unterstützten und gegenseitig wo wir nur konnten, fragten im Lehrbetrieb nach, schauten Tutorials und machten Einstiegsübungen.

MVVM

Ein zentraler Punkt, und auch einer der wichtigsten für unser Projekt, war unsere Einarbeitung in die MVVM Thematik. Dazu begannen wir zuerst herumzuprobieren. Als wir mit den Bindings Mühe hatten, arbeiteten wir mehrere Tutorials durch:

- <https://www.tutorialspoint.com/mvvm/index.htm>
- <https://www.codeproject.com/Articles/819294/WPF-MVVM-step-by-step-Basics-to-Advance-Level>
- <https://www.youtube.com/watch?v=U2ZvZwDZmJU> (Videotutorial)

Ebenfalls umstürzen uns unsere Berufsbildner dabei und wir lernten sehr schnell dazu. Es war in gewisser Weise eine Mischung aus Theorie und Learning By Doing. Besonders viele Schwierigkeiten hatten wir mit den Databindings und dem Trennen von ViewModel und Business-Logik. Auch die Verschachtelungen der einzelnen View (im Auftragsbereich) machte uns anfangs Mühe. Mittlerweile sind wir aber im Erschaffen von Views und Viewmodels schon sehr geübt.

GUI Styling

Ein weiterer wichtiger Punkt unserer Arbeit war das GUI Styling. Am Anfang der Dokumentation haben wir bereits erwähnt, dass dies für uns eine sehr hohe Bedeutung, wenn nicht sogar Priorität hat. Obwohl wir von Mahapps bereits schön gestyle Controls bekamen, verhielten sich diese nicht immer unseren Wünschen entsprechend.

Deshalb griffen wir auf das manuelle Styling der Controls zurück. Zu jedem Control kann man nämlich selber einen Style definieren und somit seine Erscheinung und sein Verhalten verändern. Wir haben gelernt, wie man mit Datatrigger und Styletrigger arbeitet, wie man Contenttemplates erstellt und verändert, wie man per Binding auf die verschiedenen Properties eines ViewModels zugreifen kann und wie sich die Verschachtelung komplexer Kontrollstrukturen verhalten.

Es war oft ein mühsames Ausprobieren, doch dank der Visual Studio 2017 Umstellung, dass man im XAML Veränderungen machen kann und diese direkt (ohne Neustart des Debugging Prozesses) angezeigt werden, ging es immer schneller.

Code-Coordination

Da wir die Arbeit in unserer Planung in Pakete aufgeteilt hatten, waren wir oft voneinander abhängig. Wir mussten auf die harte Tour lernen, wie Änderungen verloren gingen, da das andere Teammitglied fehlerhaft gecommited hat.

Ab der Hälfte der Projektzeit waren Daniel und Rafael aber ein eingespieltes Team. Wir setzten uns klare Termine (wann eine Anpassung fertig sein muss), sodass wir uns gegenseitig nicht in die Quere kamen und die Abhängigkeiten untereinander keine Probleme verursachten.

Die Erstellung unseres Zeitplans kam uns in diesem Punkt sehr entgegen.

Nhibernate

Der Datenbankteil unserer Applikation ist sehr gross. Wir wollten jedoch gezielt professionell vorgehen und entschieden uns deshalb für das Framework Nhibernate.

Uns beiden war Nhibernate schon aus dem beruflichen Alltag bekannt, jedoch hatten wir zuvor noch nie von A-Z eine eigene Implementation (mit Mappingfiles, Entities, ExpandedEntities etc.) implementiert.

Für uns war es eine tolle Erfahrung, dass dies zum ersten Mal bereits geklappt hat. Sicherlich hat das Lesen der folgenden Dokumente dazu beigetragen, dass das Mapping der Datenbank zur Software funktioniert.

- https://www.tutorialspoint.com/nhibernate/nhibernate_getting_started.htm
- http://rapidstudent.blogspot.com/2013/02/nhibernate-tutorial-in-net-with-example_5320.html

Von nun an, werden wir immer mit einem OR-Mapper arbeiten, wenn es um eine Datenbank Anbindung geht. Die Einfachheit, nachdem er implementiert ist, überzeugte uns vollkommen.

Autofac

Ein eher schwierigeres Kapitel unseres Lernprozesses war Autofac. Autofac ist ein Framework (per Nuget erhältlich), welches es ermöglicht Interfaces für Klassen zu registrieren. Dies hat die Folge, dass man ein Interface in einem Konstruktor verlangen kann, es jedoch nicht mitgeben muss. Dies sorgt einerseits für eine bessere Übersicht des Codes und andererseits auch für die Kontrolle der Instanzen, welche während der Programmlaufzeit im Umlauf sind.

Unser Ziel war am Anfang, für das Datarepository eine Registrierung vorzunehmen, da wir das Datarepository an sehr vielen Orten in der Software brauchen. Leider war die Zeit sehr knapp und trotz mehreren Anlaufversuchen klappte dies nicht. Die gemachten Schritte zur

Implementierung sind jedoch immer noch in der Solution vorhanden. (dient als Beweis zur versuchten Implementierung). Wir arbeiteten vorallem mit der Autofac-Dokumentation, welche sehr umfangreich, vielleicht etwas zu umfangreich ist:

- <http://autofac.readthedocs.io/en/latest/getting-started/index.html>

Prism

Da wir an vielen Orten mit Buttons arbeiten (Navigation, Links) haben wir uns für die Verwendung des externen Library „Prism“ entschieden. Prism bietet den Vorteil, dass man einen sogenannten DelegateCommand erstellen kann, welchen man an einen Buttoncommand hängt. Beim klicken des Buttons wird anschliessend automatisch die angegebene Methode aufgerufen.

Hier haben wir mit dem Internet und der Hilfe unserer Oberstiften zusammengearbeitet und diesen Tipp erhalten. Für uns beide war diese Library neu, und wir sind sehr froh darüber diese Library kennengelernt zu haben.

Mahapps

Wie bereits mehrfach erwähnt haben wir ebenfalls die externe Library „mahapps“ einbezogen. Diese Library fanden wir per Zufall im Internet. Die Einbindung war nicht besonders schwer.

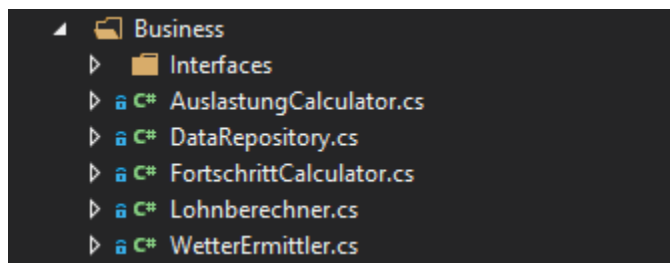
Wir haben schnell gelernt, wie man die einzelnen Controls in eine View einbindet, und wo ihre Vorteil und Nachteile liegen.

Einige der Controls haben wir ihren Nachteilen entsprechend umgestyle, so dass sie unseren Ansprüchen und Vorstellungen gerecht handeln und aussehen.

Umsetzung III

DataRepository

Als dritter Teil der Umsetzung, haben wir eine eigene Businesslogik implementiert. Unsere Businesslogik besteht aus vier Elementen. Für jedes der Elemente wäre ein Interface vorgesehen. Da aber die Autofac Implementierung misslungen ist, dienen die Interfaces lediglich als Abstraktionsvariante und als Zugriffsentkapslung.



Der Grösste Business Teil ist das DataRepository. Das DataRepository funktioniert wie folgt:

Zuerst wird (vom MainViewModel aus) initial eine Datenbank abfrage gemacht, welche alle Daten lädt.

Anschliessend werden diese Daten in Listen gespeichert, welche auf dem DataRepository liegen. Des weiteren existieren sehr viele Servicemethoden, welche vom UI Bereich (also von den ViewModels) aufgerufen werden um die Daten entsprechend zu laden.

Die Benennung der Methoden ist dabei immer gleich:

1. Was möchte man Laden?
2. Anhand von was möchte man etwas laden?
 - ➔ LoadTeamByMitarbeiter(long id);
 - ➔ LoadMitarbeiterByMitarbeiter(long id);
 - ➔ LoadAllMitarbeiter();

Ebenfalls ist dazu anzumerken, dass wir jeweils zwei verschiedene Arten von Entities haben. Für das Mapping relevante Entities sind: Team, Mitarbeiter, Position, Auftrag, Aufgabe

Um im Viewmodel jedoch alle Informationen im direkten Zugriff zu haben, haben wir uns dafür entschieden sogenannte ExpandedEntities zu generieren und als Methodenrückgaben zurückzugeben. Dies hat den Vorteil, dass beispielsweise auf einem Auftrag Objekt ebenfalls alle Aufgaben, sowie die Personen, welche die Aufgaben ausführen im Zugriff.

Calculators





Ebenfalls sind mehrere Calculators implementiert, welche Berechnungen verschiedener Werte durchführen. Der Auslastungscalculator, welche im Teambereich gebraucht wird, berechnet anhand von einem Team, deren Mitarbeiter und Aufgaben eine prozentuale Auslastung aus.



Der FortschrittCalculator berechnet anhand eines Auftrages und deren Aufgaben den Fortschritt eines gesamten Auftrages aus. Wie auch bereits schon in der Bedienungsanleitung von mogadu erwähnt, berechnet die Komponente Lohnberechner den Lohn einer Person anhand seiner Position und Firmendazugehörigkeit (zeitlich) aus.

Ermittler

Der Wetterermittler sollte ursprünglich durch das empfangen eines Signals von einer Google API das momentane Wetter ermitteln und an ein ViewModel im Übersichtsbereich zurückgeben. Aus zeitlichen Gründen haben wir dies jedoch nicht ausimplementiert und es wird lediglich eine Random Zahl zurückgegeben. Anhand dieser Zahl wird entschieden, ob es heute regnet oder sonnig ist.

Tests

Beschrieb	Erwartetes Resultat	OK?
1. Programm starten 2. Einloggen mit falschen Passwort	Eine Validierungsmeldung (Icon) wird angezeigt.	
1. Programm starten 2. Einloggen	Übersicht öffnet sich automatisch und eine Kachelansicht (wie bei der Userstory beschrieben) wird angezeigt	
1. Programm starten 2. Einloggen 3. Menüpunkt „Team“ anwählen	Eine Teamansicht erscheint, auf der mehrere Informationen zum Team (Name, Auslastung, Teamleiter etc.) und die jeweiligen Teammitgliedern in Steckbriefform aufgelistet sind.	
1. Programm starten 2. Einloggen 3. Menüpunkt „Aufträge“ anwählen	Es erscheint ein Datagrid mit allen Aufträgen, welche ein Team bearbeitet. Im Datagrid sind Auftragsname, Teambezeichnung, offene Aufgaben, abgeschlossene Aufgaben vorhanden.	
1. Programm starten 2. Einloggen 3. Menüpunkt „Aufträge“ anwählen 4. Auf die Teambezeichnung klicken	Das entsprechende Team (die Teamansicht) wird angezeigt, und dafür wird ein neues Tab geöffnet.	
1. Program starten 2. Einloggen 3. Menüpunkt „Aufträge“ anwählen 4. Auf die Teambezeichnung klicken	Die Auftragsdetailansicht öffnet sich in einem neuen Tab. Auf der Ansicht werden alle offenen und abgeschlossenen Aufgaben, sowie auch Informationen zum Auftrag selber angezeigt. (Auftragsfortschritt, Bezeichnung etc.)	
1. Programm starten 2. Einloggen 3. Menüpunkt „Aufträge“	Eine kurze Ansicht öffnet sich (in einem neuen Tab), welche Informationen zum	

anwählen 4. Auf die Teambezeichnung klicken 5. Auf Mitarbeiterbezeichnung klicken	Mitarbeiter anzeigt.	
1. Programm starten 2. Einloggen 3. Menüpunkt „Aufträge“ anwählen 4. Auf die Aufgabenbezeichnung klicken	Es wird eine Ansicht in einem neuen Tab (mit dem Namen Aufgabendetail) geöffnet. Im Aufabendetail kann man den ausführenden Mitarbeiter verändern und den Aufgabenfortschritt mutieren. Sobald man eines der beiden Sachen verändert, wird ein Speichern Button Aktiv, welcher die Daten persistiert	
1. Programm starten 2. Einloggen 3. Menüpunkt „Aufträge“ anwählen 4. Alle Aufgabenfortschritte auf 100% setzen	In der Auftragsübersicht ist das das „Abgeschlossen“ Häckchen gesetzt und das n der Auftragsdetail Ansicht ist das Datum eingetragen.	

Fazit

Abschliessend sind wir über die gemachte Erfahrung „mogadu“ sehr froh. Wir haben und sehr in das Projekt hineingesteigert, viele Stunden abseits vom Schulunterricht daran gearbeitet und sicherlich sehr viel Neue Sachen dazugelernt.

Besonders toll war unsere Planung und die anschliessende Zusammenarbeit. Wir haben uns respektvoll behandelt und haben immer auf das gleiche Ziel hingearbeitet. Wir nahmen auf abhängige Aufgabenpakete Rücksicht und sind somit zu einem guten Endziel gekommen.

Folgende Punkte haben wir aufgrund des Zeitmangels nicht mehr fertigstellen können. Diese hätten wir gerne noch im Rahmen des Projektes fertiggestellt. Jedoch war die Zeit sehr knapp.

- Ausprogrammierung des WetterErmittlers
- Erstellung neuer Mitarbeiter / Aufgaben im Programm (nicht nur Datenbank)
- Weitere Kacheln zur Übersicht hinzufügen
- Datenbank Anbindung optimieren (von der Performance her)
- Allgemein Performance verbessern

Wir sind mit unserer geleisteten Arbeit jedoch mehr als Zufrieden und können unserer Meinung nach sehr stolz sein.