



# myMD

## IMPLEMENTIERUNG

Praxis der Softwareentwicklung WS2017/2018

*Philipp Pelcz, Philipp Karcher, Jan-Luca Vettel*

supervised by  
Marc Aurel Kiefer

12. März 2018

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>3</b>
<b>2 Vorangegangene Planung</b>	<b>4</b>
2.1 Kriterien (Planungsphase) . . . . .	4
2.2 Klassendiagramm (Entwurfsphase) . . . . .	7
2.3 Gantt-Diagramm (Entwurfsphase) . . . . .	8
<b>3 Geänderte Daten</b>	<b>9</b>
3.1 Geänderte Entwürfe . . . . .	9
3.2 Geänderte Klassen . . . . .	13
3.3 Nicht umgesetzte Kriterien . . . . .	23
3.4 Klassendiagramm (Implementierungsphase) . . . . .	26
<b>4 Ergebnisse der Phase</b>	<b>28</b>
4.1 Ablauf . . . . .	29
4.2 Zukunftspläne . . . . .	30

# 1 Einleitung

"myMD" ist eine mobile Anwendung, die es Patienten erlaubt, ihre Arztbriefe digital über Bluetooth von ihren Ärzten zu erhalten, um dann in der Applikation die darin enthaltenen Informationen, wie die Diagnose oder verschriebene Medikationen, einsehen zu können.

Die App wurde dabei mit Xamarin.Forms in C# und XAML für Android- und iOS-Geräte entwickelt.

Aufbauend auf dem vorangegangenen Pflichtenheft und dem Entwurfsdokument, wird in diesem Dokument der Fortschritt der Implementierung und die dabei vorgenommenen Änderungen an in Planung und Entwurf festgelegten Kriterien beschrieben.

Zunächst werden noch einmal die wichtigsten Ergebnisse aus Planungs- und Entwurfsphase aufgelistet und danach die daran in dieser Phase vorgenommenen Änderungen. Dazu zählen geänderte Entwürfe, Klassen und nicht umgesetzt Kriterien.

Letztlich folgen noch abschließende Worte über den Verlauf der Phase und Pläne für die Zukunft.

## 2 Vorangegangene Planung

### 2.1 Kriterien (Planungsphase)

*Hinweis: Der Übersicht wegen sind hier Pflicht- und Wunschkriterien zusammen aufgeführt. Sie sind weiterhin anhand der Kürzel voneinander unterscheidbar (PK = Pflichtkriterium, WK = Wunschkriterium).*

#### Patientenseitige Datenübertragung

- [PK1010] Arztbriefe können von der Desktop Anwendung auf die myMD App des Patienten übertragen werden.
- [PK1020] Die Übertragung erfolgt verschlüsselt über Bluetooth.
- [PK1030] Eine laufende Datenübertragung kann manuell abgebrochen werden.
- [WK1010] Ein Arztbrief kann von der myMD App des Patienten auf die Desktop Anwendung übertragen werden.
- [WK1020] Der Patient kann mehrere Arztbriefe gleichzeitig senden.
- [WK1030] NFC steht als weitere Übertragungsmöglichkeit zur Verfügung.
- [WK1040] Der Patient wird vor dem Senden von sensiblen Daten darauf hingewiesen, dass er sensible Daten versendet.
- [WK1050] Ein Profil auf einem Mobilgerät kann auf ein anderes übertragen werden.

## Darstellung

- [PK2010] Arztbriefe werden chronologisch absteigend im Tab *Übersicht* dargestellt.
- [PK2020] Der Nutzer kann überflüssige/unerwünschte Arztbriefe löschen.
- [PK2030] Die Darstellung eines Arztbriefes umfasst die Diagnose, verordnete Medikamente, das Datum und den Namen des behandelnden Arztes.
- [WK2010] Eingenommene Medikamente werden in einem extra Tab chronologisch absteigend sortiert dargestellt.
- [WK2020] Laborwerte des Patienten werden in einem extra Tab chronologisch absteigend sortiert dargestellt.
- [WK2030] Ein Arztbrief kann Bilddateien enthalten und die myMD App kann diese originalgetreu darstellen und einem Arztbrief zuordnen.
- [WK2040] Es gibt die Möglichkeit, Arztbriefe nach eigenen Kriterien (Arzt, Krankheit o.ä.) zu gruppieren.
- [WK2050] Es gibt eine Suchfunktion, die alle Arztbriefe nach Daten durchsucht.

## Einstellungen

- [PK3010] Der Nutzer kann ein eigenes Profil anlegen, welches Daten wie den Namen, Versicherungsnummer, Blutgruppe und Allergien enthält.
- [PK3020] Die myMD App wird in Deutsch angeboten.
- [WK3010] Auf einer myMD App können mehrere Nutzer verwaltet werden.
- [WK3020] Die myMD App wird zusätzlich auch auf Englisch angeboten.
- [WK3030] Der Nutzer kann einzelne Arztbriefe oder ganze Gruppen als sensibel markieren.
- [WK3040] Die myMD App kann den Nutzer an regelmäßige Arzttermine (z.B. Zahnarzt, Augenarzt) erinnern.

## **Desktop Anwendung**

- [PK4010] Die Desktop Anwendung kann die Geräte in der Nähe anzeigen.
- [PK4020] Der Arzt kann unter den Geräten in der Nähe das Mobilgerät des Patienten als Empfänger auswählen.
- [PK4030] Digitale Arztbriefe können entweder per Drag and Drop oder über einen Explorer in die Desktop Anwendung geladen werden.
- [PK4040] Die Daten werden auf Knopfdruck an das Mobilgerät des Patienten gesendet.
- [WK4010] Die Versichertennummer, die in einem Arztbrief auf dem Computer des Arztes eingetragen ist, wird vor dem Senden mit der in der myMD App des Patienten hinterlegten Versichertennummer abgeglichen und nur bei Übereinstimmung wird der Arztbrief gesendet.

## **Kompatibilität**

- [PK5010] Die Desktop Anwendung wird von Microsoft Windows 10 unterstützt.
- [PK5020] Die myMD App wird von Android 6.0 (und höher) unterstützt.
- [PK5030] Die myMD App kann Arztbriefe im .hl7 Dateiformat anzeigen.
- [WK5010] Die myMD App wird von iOS 10 (und höher) unterstützt.
- [WK5020] Die Desktop Anwendung wird zusätzlich von macOS 10.12 (und höher) unterstützt.
- [WK5030] Die myMD App kann Arztbriefe im .pdf Dateiformat anzeigen.
- [WK5040] Die myMD App kann Arztbriefe im .csv Dateiformat anzeigen.

## 2.2 Klassendiagramm (Entwurfsphase)

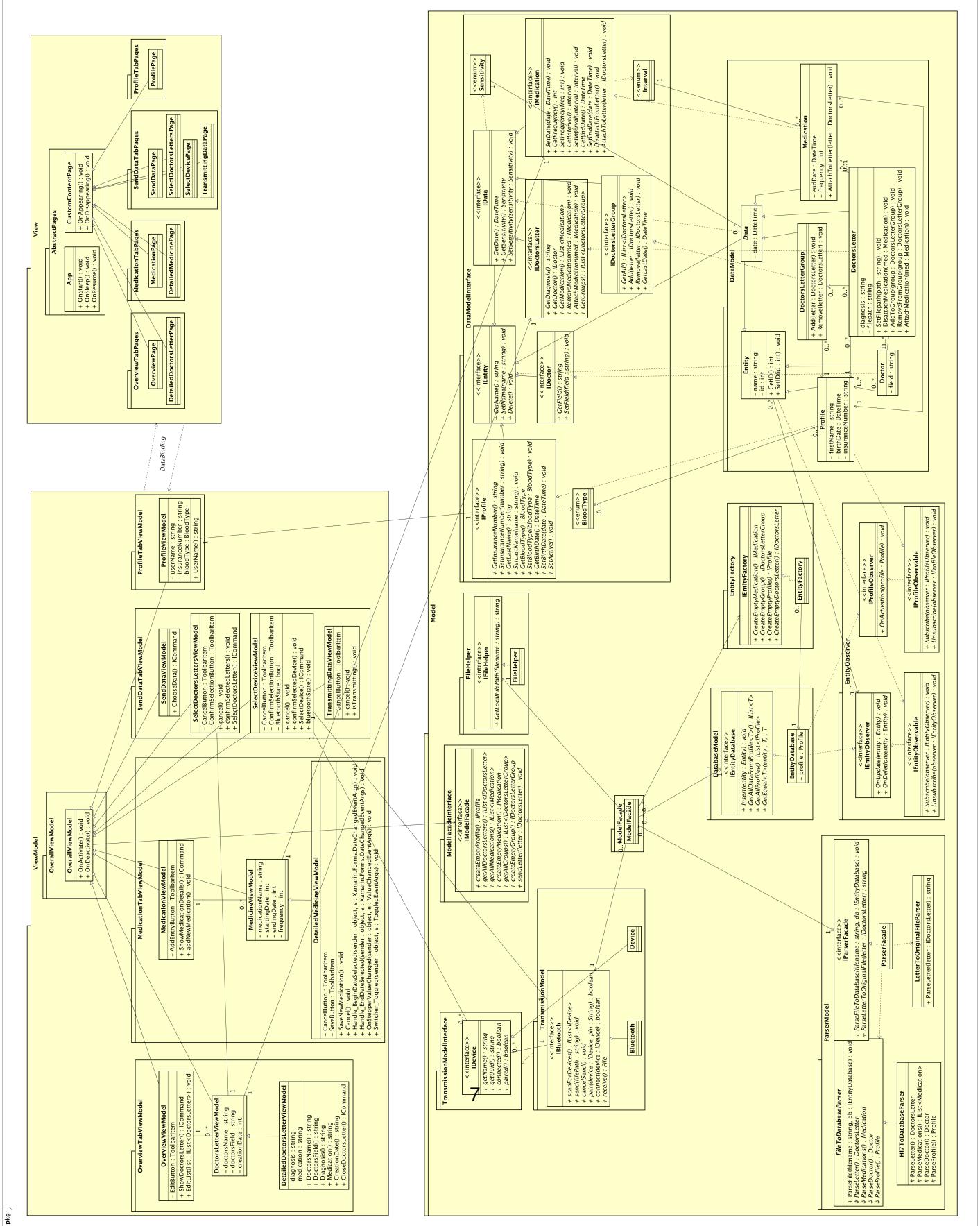


Abbildung 2.1: myMD Klassendiagramm der Entwurfsphase

## 2.3 Gantt-Diagramm (Entwurfsphase)

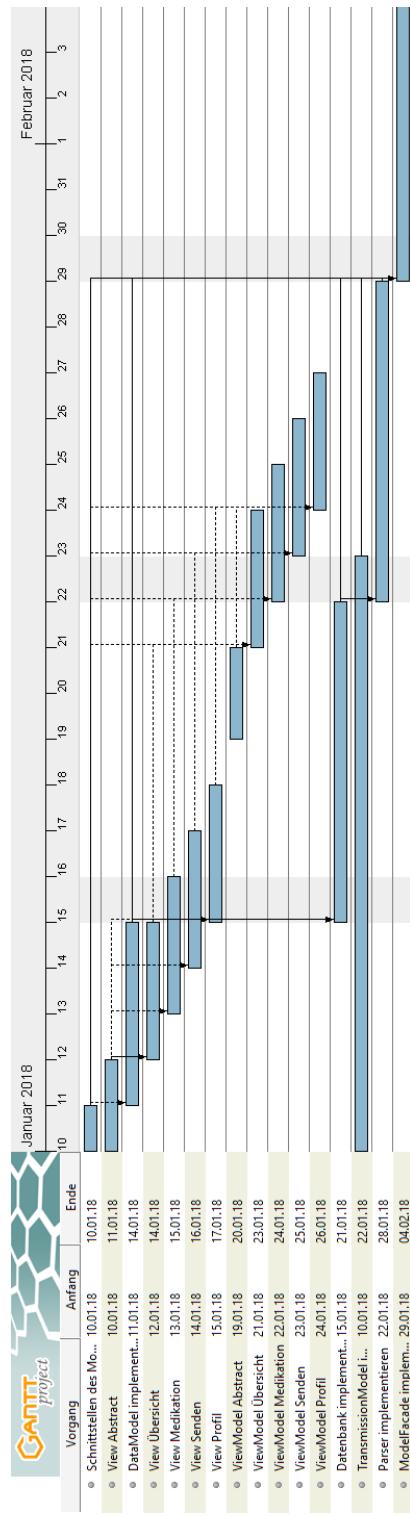


Abbildung 2.2: Gantt Diagramm der Entwurfsphase

# 3 Geänderte Daten

## 3.1 Geänderte Entwürfe

### 3.1.1 Model

#### Beobachter-Entwurfsmuster

In der Entwurfsphase entschieden wir uns für das Beobachter-Entwurfsmuster, um die Datenbank über Änderungen an den Instanzen der Klassen aus dem **DataModel** Paket zu benachrichtigen.

Das **EntityObserver** Paket, das die dafür vorgesehenen Schnittstellen enthält, wurde mitamt diesen komplett entfernt. Stattdessen muss das Aktualisieren einer Entität nun per explizitem Aufruf an die **ModelFacade** durchgeführt werden.

Die Gründe dafür sind zweigeteilt:

Erstens werden durch die Benutzung der *SQLite-Net-Extensions* Library beim rekursiven Auflösen von Relationen Objekte aus der Datenbank geladen, ohne dass bei ihnen von uns die Datenbank als Beobachter angemeldet werden kann. Um das Laden von in Relation stehender Entitäten aus der Datenbank zu vereinfachen, ist es sinnvoll das Beobachterprinzip nicht weiter zu verwenden.

Zweitens ist es von der GUI-Seite auch intuitiver, Änderungen dann erst zu übernehmen, wenn dies explizit vom Benutzer so veranlasst wird. Damit bietet sich die *Update* Methode in der **ModelFacade** als Alternative an.

#### Datenbanktabellen

Alle Klassen aus dem **DataModel** Paket stellen Vorlagen für eine Datenbanktabelle dar. Im ursprünglichen Entwurf waren nur die Getter und Setter für den Zugriff vom ViewModel aus

vorgesehen (mit den Schnittstellen aus **DataModellInterface** als Parameter). Um Objekte vollständig in die Datenbank schreiben und wieder daraus laden zu können, werden Getter und Setter für alle konkreten Attribute benötigt.

Um unnötiges Casten zu vermeiden, erhalten die Schnittstellen aus **DataModellInterface** noch eine Konvertierungsmethode zu der in **DataModel** definierten konkreten Implementierung (für die Klassen aus **DataModel** gibt diese Methode lediglich die Instanz der Klasse selbst zurück.)

### Plattformspezifisches Parsen

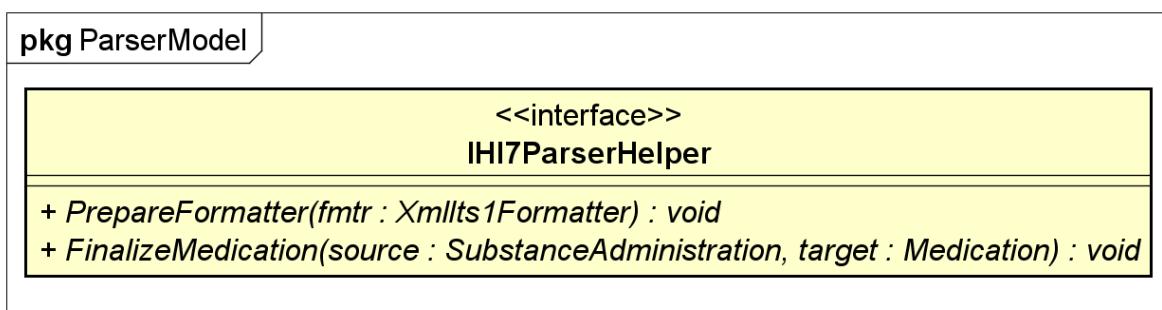


Abbildung 3.1: IHI7ParserHelper Schnittstelle

Bestimmte Klassen des *Everest Frameworks*, das zum Parsen von .hl7 Dateien benutzt wird, implementieren die **ICloneable** Schnittstelle, die in einem *Xamarin-Cross-Platform* Projekt nicht verfügbar ist.

Manche dieser Klassen werden jedoch benötigt um die Dateien vollständig zu parsen, daher kann der **Hl7ToDatabaseParser** nicht mehr wie geplant komplett plattformunabhängig implementiert werden.

Operationen mit diesen Klassen werden nun über die **IHI7ParserHelper** Schnittstelle aufgerufen und im **Hl7ParserHelper** plattformspezifisch implementiert.

## Dateiformatunabhängiges Parsen

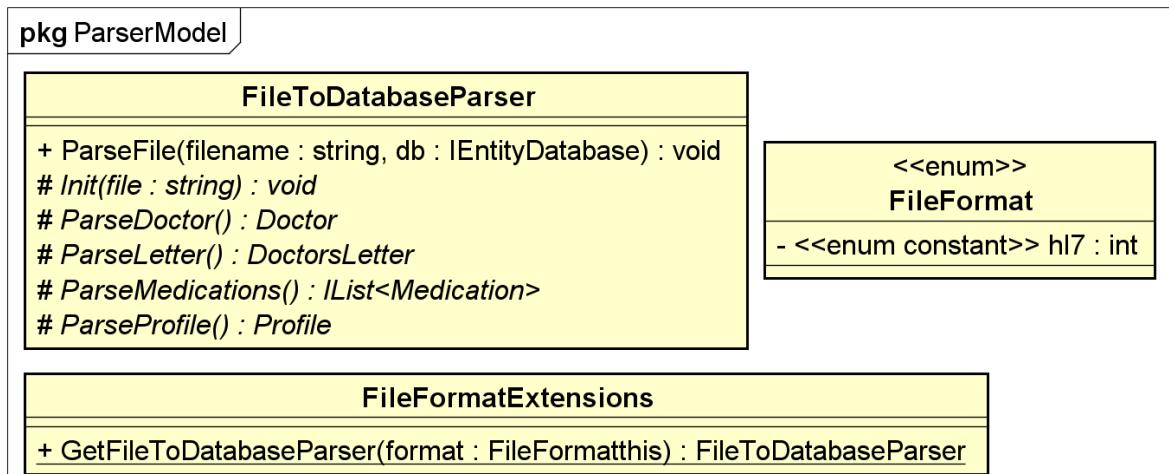


Abbildung 3.2: Klassen zum dateiformatunabhängigen Parsen

Um in Zukunft die Implementierung der Unterstützung mehrere Dateiformate zu vereinfachen, wurden mehrere Änderungen am **FileToDatabaseParser** vorgenommen:

Das Auswählen des richtigen Parsers erfolgt nun über die Enumeration **FileFormat** der unterstützten Dateiformate und eine Erweiterungsmethode in **FileFormatExtensions** die einem Dateiformat aus **FileFormat** den passenden **FileToDatabaseParser** zuordnet.

Weiterhin wurden auch die dateiformatspezifischen Einschubmethoden des **FileToDatabaseParser** modifiziert:

Die zu parsende Datei wird nun über die neu hinzugefügte *Init* Methode übergeben, die weiteren *Parse* Methoden sind dafür nun parameterlos. Dies erlaubt es flexibler aus einer Datei lesen zu können.

Beim vorherigen Ansatz musste für jeden Aufruf einer *Parse* Methoden neu aus der Datei gelesen werden. Jetzt können stattdessen gemeinsame Vorbereitungen getroffen werden, wie z.B. beim **Hl7ToDatabaseParser**. Hier wird mit *Init* die gesamte Datei mit *Everest* in den frameworkeigenen Datentyp *ClinicalDocument* geparsst, der dann wiederum mit den restlichen *Parse* Methoden in die Datentypen aus dem **DataModel** Paket geparsst wird.

### 3.1.2 ViewModel

#### Enum-Picker

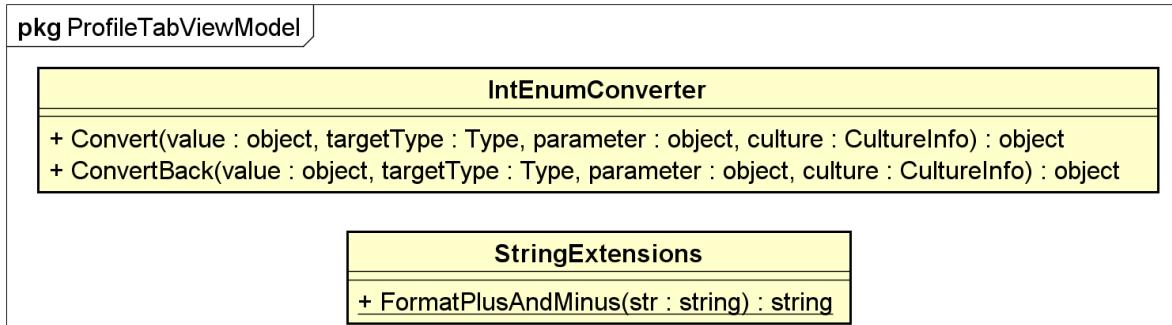


Abbildung 3.3: Enum Picker Klassen

Um Elemente aus einem Enum in einem Picker von *Xamarin* darstellen zu können, wird ein Konvertierer (**IntEnumConverter**) benötigt, da die dafür von *Xamarin* angebotene Funktionalität eine Liste von Elementen erwartet. Mit dem Konvertierer kann dann jedem Listeneintrag über *DataBinding* ein Enum-Wert zugewiesen werden.

Um die Namen von Enumwerten zum Anzeigen formatieren zu können, wurden für diesen Zweck auch noch die Klasse **StringExtensions** mit Erweiterungsmethoden für strings hinzugefügt.

#### ProfileItemViewModel

Um die Daten eines Profils beim Wechseln vom **ProfileViewModel** zum **ProfileEditViewModel** (und zurück) konsistent zu halten, mussten wir ein neues ViewModel einführen, dass diese Daten kapselt.

### 3.1.3 View

#### Benutzeroberflächenlogik

In der Entwurfsphase orientierten wir uns daran, dass die Benutzeroberflächenlogik nach dem *MVVM-Prinzip* in den jeweiligen *ViewModels* implementiert wird. Jedoch fiel uns auf,

dass dies sehr umständlich ist. Deshalb haben wir die Benutzeroberflächenlogik im Code-Behind zu jeder entsprechenden Seite in der *View* implementiert.

### **ProfileEditPage**

Während dem Entwurf stellten wir uns vor, dass es auf einer **ProfilePage** einen Toggle zum Bearbeiten der Daten im Profil gibt, der die Labels, in denen die Daten stehen, durch Entries ersetzt. Da sich diese Funktion, aber durch die Gestaltung der Seite in .xaml nur suboptimal implementieren lässt, haben wir uns dazu entschieden, die neue Seite **ProfileEditPage** einzuführen. Diese wird geöffnet, wenn man den *Bearbeiten-Button* der **ProfilePage** antippt. Die **ProfileEditPage** gleicht der **ProfilePage** im Design. Die Unterschiede begrenzen sich auf die zwei *Statusbar-Elemente* "Abbrechen", "Fertig" und die Ersetzung der Labels (in der **ProfilePage**) durch Entries.

## **3.2 Geänderte Klassen**

*Hinweis: Nur leicht (z.B. in der Namensgebung) geänderte Klassen oder solche, die bereits in den geänderten Entwürfen erwähnt sind hier nicht aufgelistet. Weiterhin wurden die meisten Getter- und Setter-Methoden durch die in C# üblichen Properties ersetzt. Da dies mehr eine optische als tatsächlich inhaltliche Änderung darstellt, wird auch dies hier übergegangen.*

### 3.2.1 ModellInterface

IModelFacade

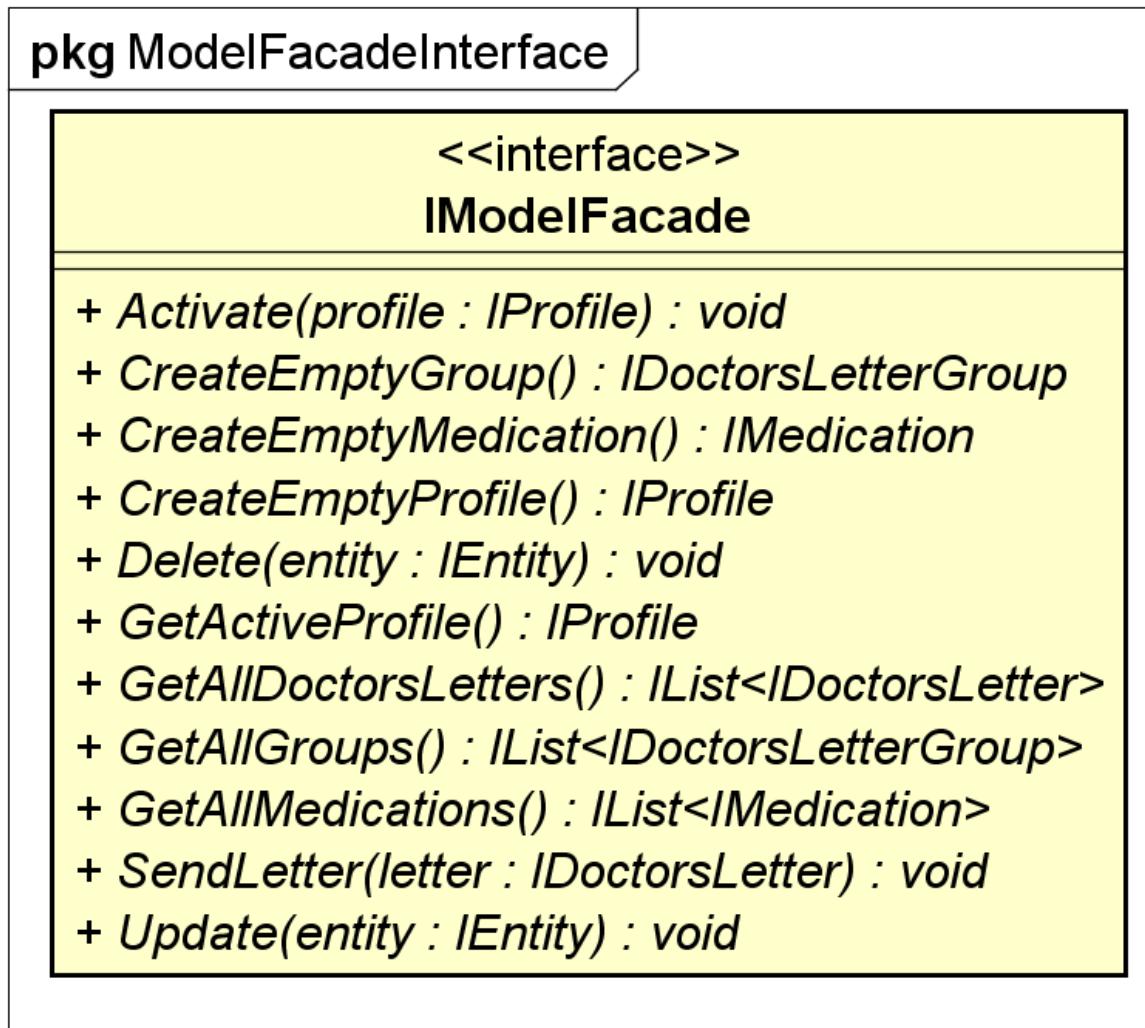


Abbildung 3.4: IModelFacade Schnittstelle

**Art der Änderung:** Neue Methoden

**Beschreibung:** Methode zum Abfragen des aktuell aktiven Profils hinzugefügt.

**Paket:** ModellInterface.ModelFacadeInterface

## IMedication

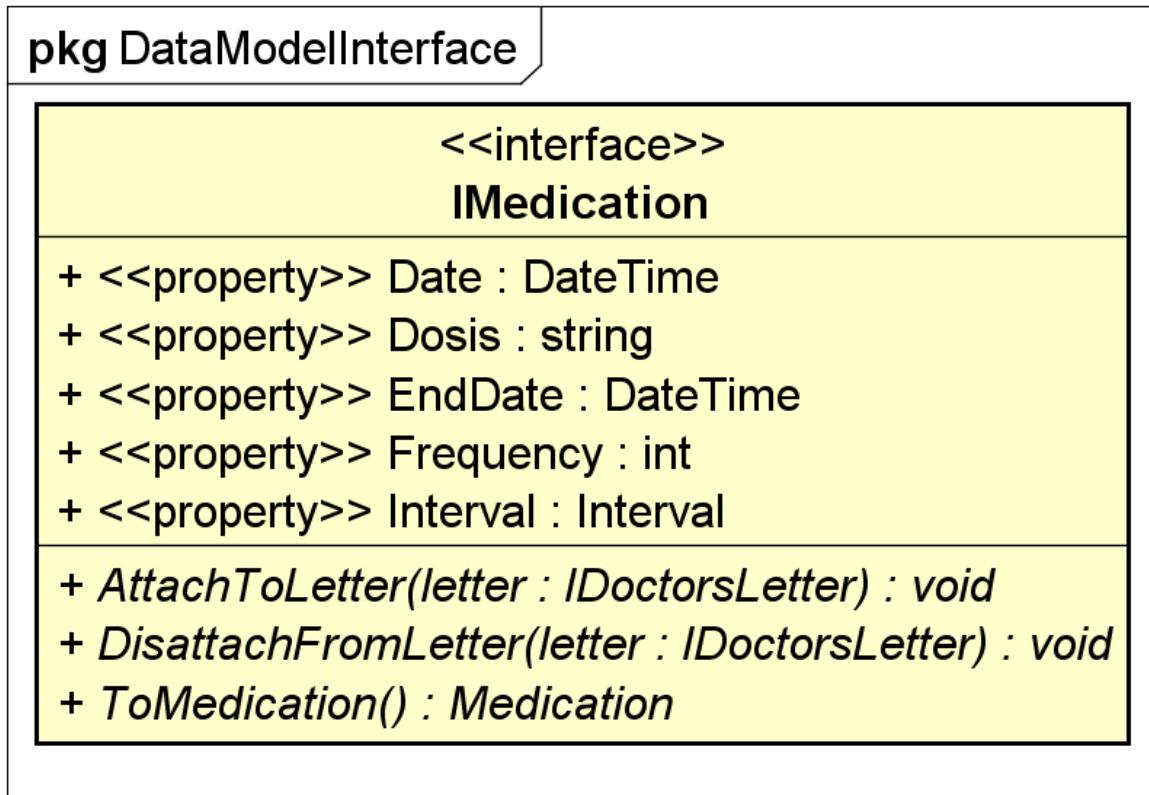


Abbildung 3.5: IMedication Schnittstelle

**Art der Änderung:** Neues Attribut

**Beschreibung:** Medikation haben jetzt auch eine Dosis (Getter und Setter als string)

**Paket:** ModellInterface.DataModellInterface

### 3.2.2 Model

*Hinweis: Klassen aus dem **DataModel** Paket wurden Vergleichsmethoden wie Compare und Equals hinzugefügt, die hier nicht für jede Klasse einzeln aufgezählt werden.*

## IEntityDatabase

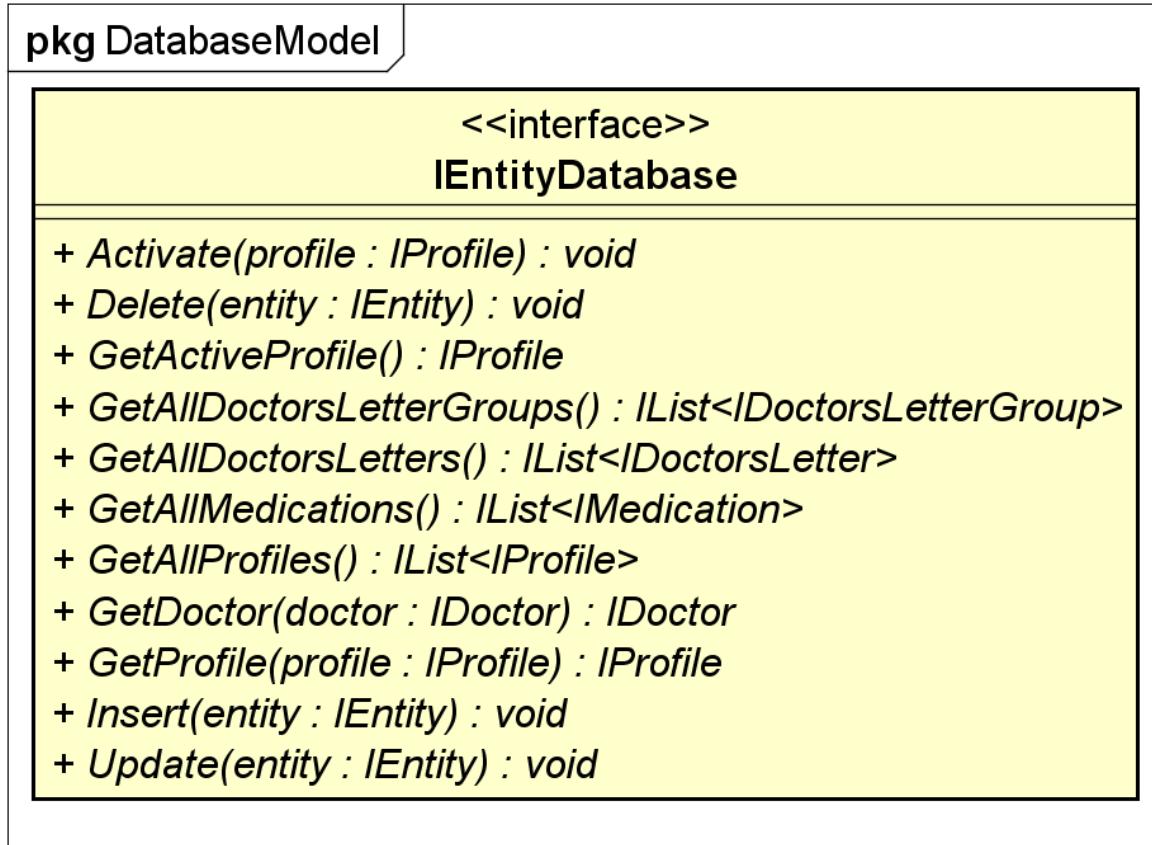


Abbildung 3.6: IEntityDatabase Schnittstelle

**Art der Änderung:** Neue/geänderte Methoden

**Beschreibung:** Die Methoden der Observer-Schnittstellen aus dem **EntityObserver** Paket befinden sich nun in dieser Schnittstelle.

Außerdem wurde die generischen Methoden durch typspezifische Methoden ersetzt, eine Methode zum Abfragen des aktuell aktiven Profils hinzugefügt und die Methodenparameter sind nun konsistent aus dem **DataModellInterface** Paket.

**Paket:** Model.EntityDatabase

## EntityFactory

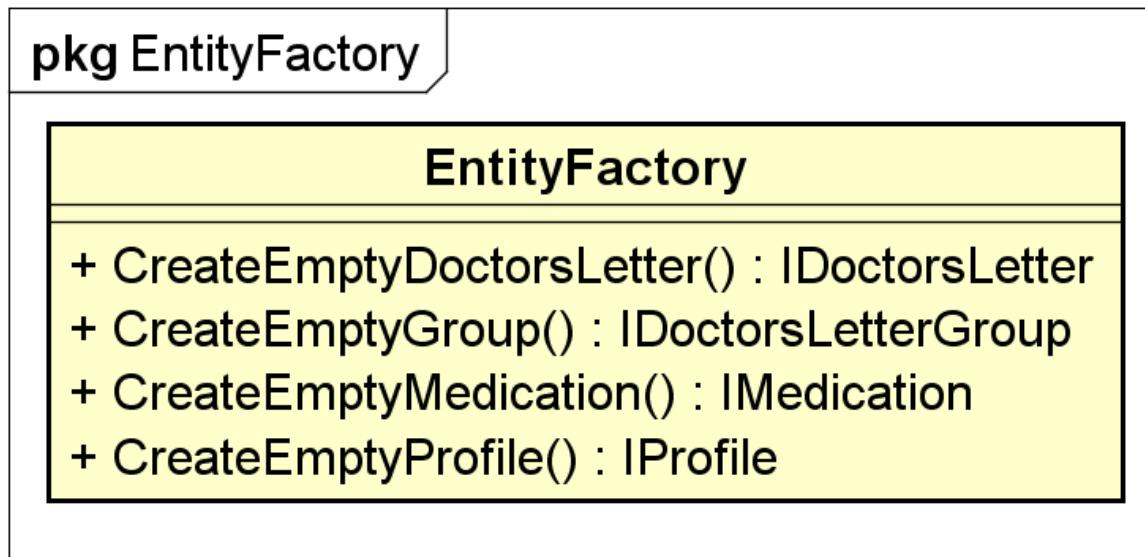


Abbildung 3.7: EntityFactory Klasse

**Art der Änderung:** Entferntes Attribut

**Beschreibung:** Im ursprünglichen Entwurf hatte **EntityFactory** noch eine **IEntityDatabase** als Attribut um erstellte Entitäten direkt in die Datenbank einzufügen. Um Abhängigkeiten zwischen Klassen zu verringern wurde das Attribut entfernt und das Einfügen in die Datenbank wird von der **ModelFacade** übernommen, die sowieso bereits die Datenbank als Attribut enthält.

**Paket:** Model.EntityFactory

## DoctorsLetterGroupDoctorsLetter

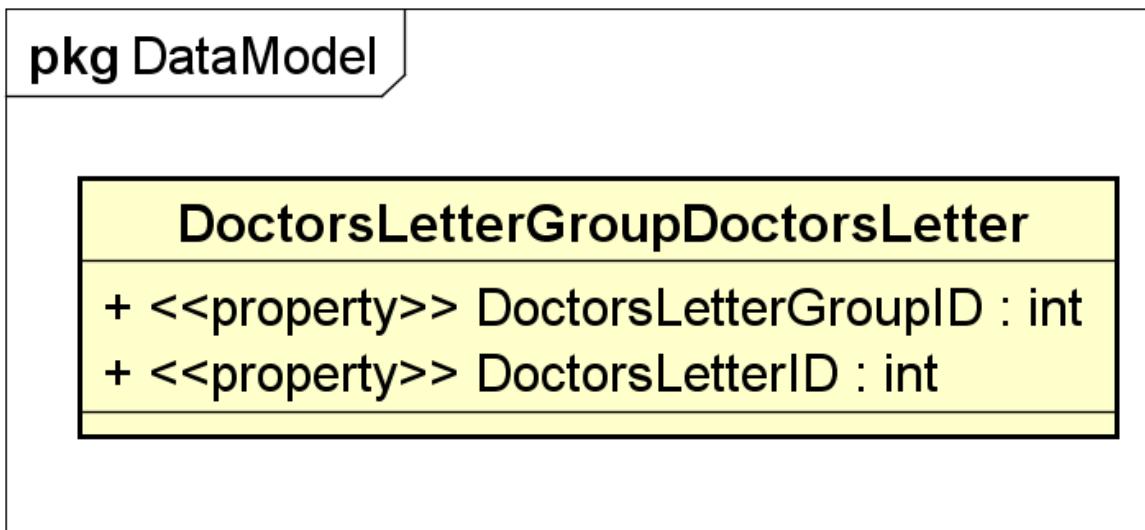


Abbildung 3.8: DoctorsLetterGroupDoctorsLetter Klasse

**Art der Änderung:** Neue Klasse

**Beschreibung:** Um Many-To-Many Relationen darstellen zu können, benötigt das SQLite-net-extensions Framework eine separate Klasse für die Relation. Da Arztbriefe und Gruppen in einer solchen Relation stehen, wird diese Klasse benötigt.

**Paket:** Model.DataModel

## IFileHelper

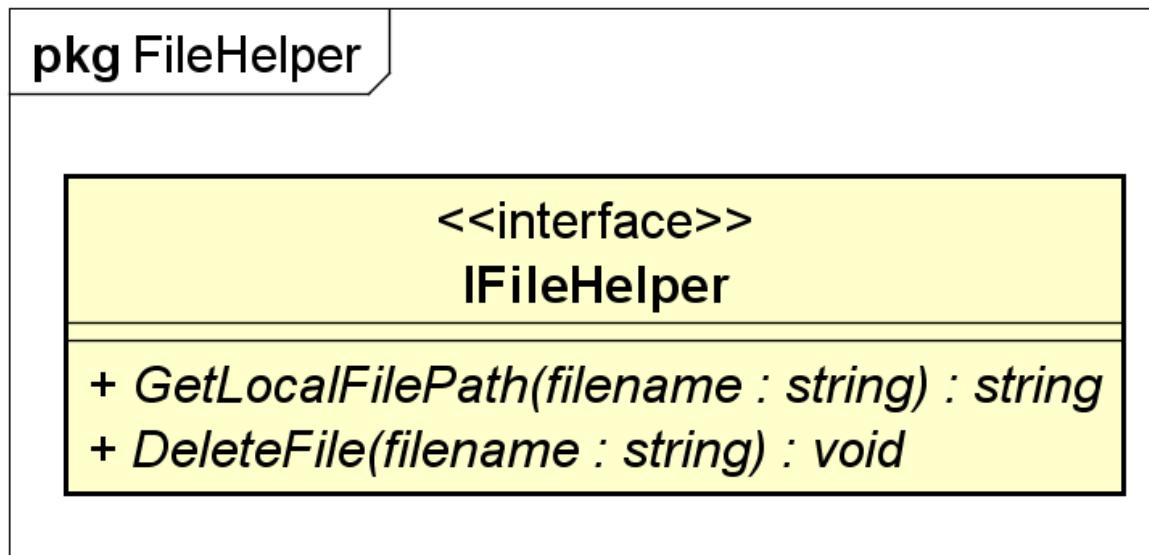


Abbildung 3.9: IFileHelper Schnittstelle

**Art der Änderung:** Neue Methode

**Beschreibung:** Methode zum Löschen von Dateien hinzugefügt.

**Paket:** Model.FileHelper

### 3.2.3 ViewModel

#### OverviewViewModel

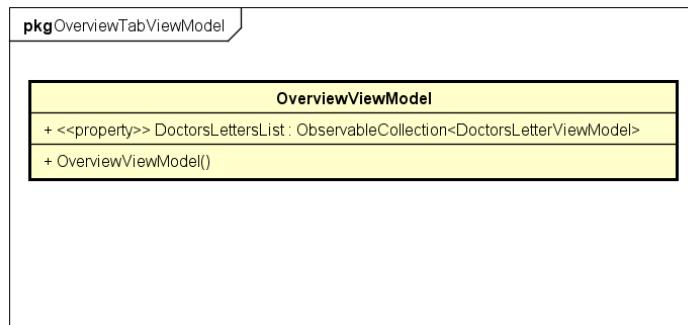


Abbildung 3.10: OverviewViewModel Klasse

**Art der Änderung:** Attribut hinzugefügt

**Beschreibung:** Um eine Liste an DoctorsLetter zu modellieren, muss diese dem ViewModel vorliegen. Dieser Fehler wurde im Entwurf übersehen.

**Paket:** ViewModel.OverviewTabViewModel

#### DetailedDoctorsletterViewModel

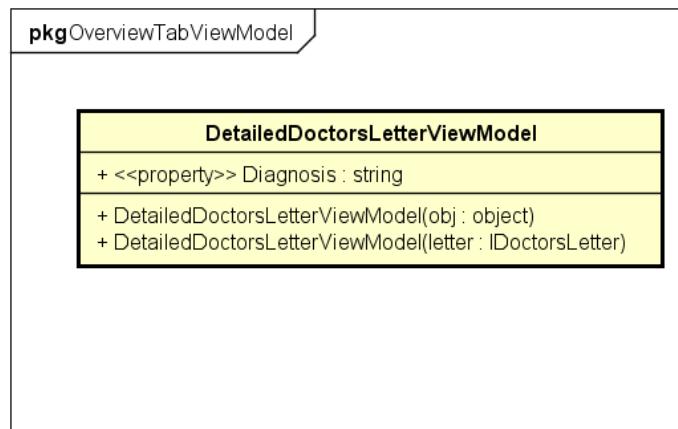


Abbildung 3.11: DetailedDoctorsLetterViewModel Klasse

**Art der Änderung:** Methoden entfernt

**Beschreibung:** Im Entwurf wurden im DetailedDoctorsletterViewModel Methoden eingeführt, um aus einem DoctorsLetter Daten zu extrahieren und in einem String zu speichern. Während der Implementierung stellten wir fest, dass diese Methoden nicht nötig sind.

**Paket:** ViewModel.OverviewTabViewModel

### DoctorsLetterViewModel

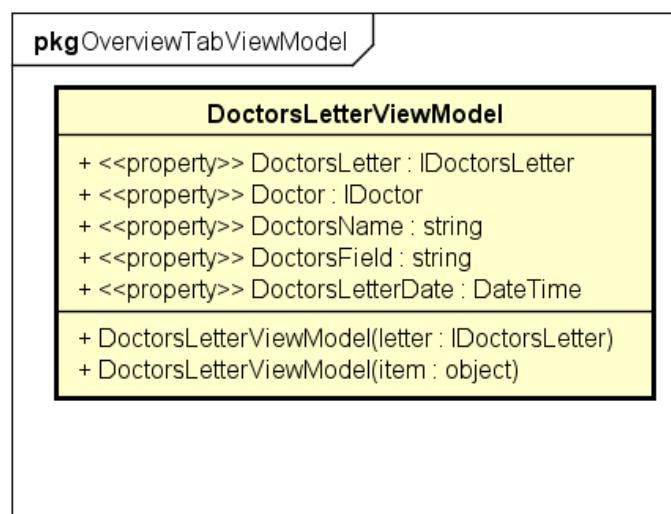


Abbildung 3.12: DoctorsLetterViewModel Klasse

**Art der Änderung:** Attribut hinzugefügt

**Beschreibung:** Da ein Arztbrief alle Informationen zu einem Arzt anzeigt, bietet es sich an, dass das DoctorsLetterViewModel ein Attribut vom IDoctor besitzt.

**Paket:** ViewModel.OverviewTabViewModel

## MedicationViewModel

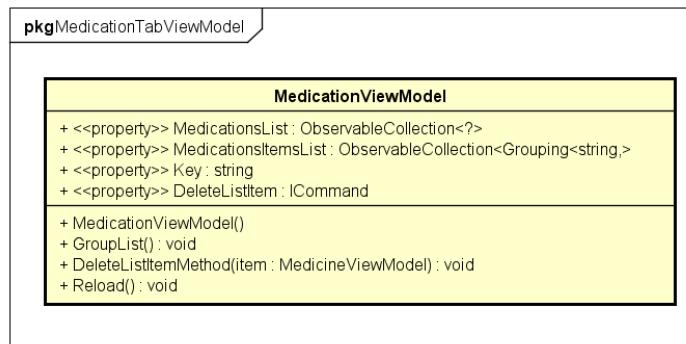


Abbildung 3.13: MedicationViewModel Klasse

**Art der Änderung:** Methoden und Attribute hinzugefügt

**Beschreibung:** Während dem Entwurf war uns nicht klar, dass man eine für das Gruppieren und Sortieren von Medikationen eigene Attribute und Methoden benötigt. Diese wurden bei der Implementierung hinzugefügt.

**Paket:** ViewModel.MedicationTabViewModel

## MedicineViewModel

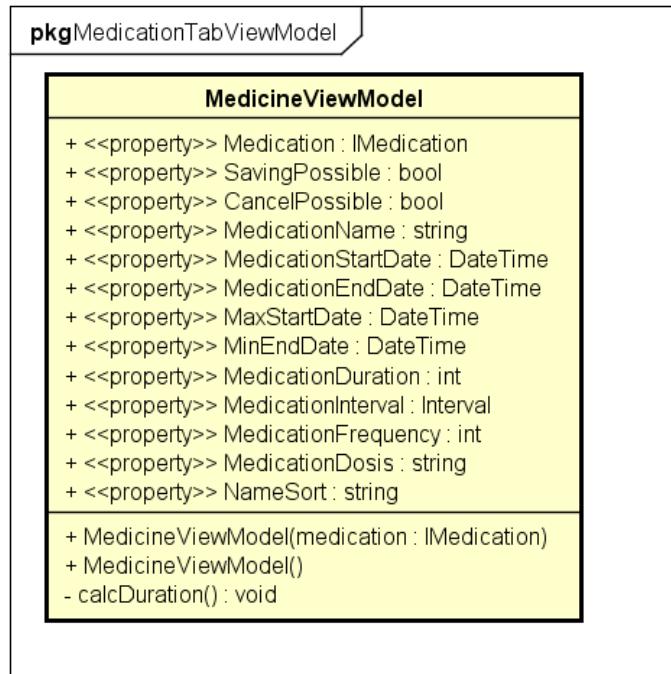


Abbildung 3.14: MedicineViewModel Klasse

**Art der Änderung:** Methoden und Attribute hinzugefügt

**Beschreibung:** Um eine Eingabe auf ihre Korrektheit zu überprüfen, haben wir neue Attribute eingeführt und Methoden um die Attribute auf korrekte Werte zu setzen.

**Paket:** ViewModel.MedicationTabViewModel

### 3.3 Nicht umgesetzte Kriterien

*Hinweis: Manche dieser Kriterien (z.B. [PK2030]) wurden nur teilweise umgesetzt. D.h. nur manche Teile unseres Projektes unterstützen diese Kriterien, aber aus Zeitgründen wurde die vollständige Implementierung verschoben. Die halbfertigen Kriterien werden durch ein "!" gekennzeichnet.*

## **Patientenseitige Datenübertragung**

- [WK1010] Ein Arztbrief kann von der myMD App des Patienten auf die Desktop Anwendung übertragen werden.
- [WK1020] Der Patient kann mehrere Arztbriefe gleichzeitig senden.
- [WK1030] NFC steht als weitere Übertragungsmöglichkeit zur Verfügung.
- [WK1040] Der Patient wird vor dem Senden von sensiblen Daten darauf hingewiesen, dass er sensible Daten versendet.
- [WK1050] Ein Profil auf einem Mobilgerät kann auf ein anderes übertragen werden.

## **Darstellung**

- ![PK2030] Die Darstellung eines Arztbriefes umfasst verordnete Medikamente.
- [WK2020] Laborwerte des Patienten werden in einem extra Tab chronologisch absteigend sortiert dargestellt.
- [WK2030] Ein Arztbrief kann Bilddateien enthalten und die myMD App kann diese originalgetreu darstellen und einem Arztbrief zuordnen.
- [WK2040] Es gibt die Möglichkeit, Arztbriefe nach eigenen Kriterien (Arzt, Krankheit o.ä.) zu gruppieren.

## **Einstellungen**

- ![WK3010] Auf einer myMD App können mehrere Nutzer verwaltet werden.
- ![WK3030] Der Nutzer kann einzelne Arztbriefe oder ganze Gruppen als sensibel markieren.
- [WK3040] Die myMD App kann den Nutzer an regelmäßige Arzttermine (z.B. Zahnarzt, Augenarzt) erinnern.

## **Desktop Anwendung**

- [PK4010] Die Desktop Anwendung kann die Geräte in der Nähe anzeigen.
- [PK4020] Der Arzt kann unter den Geräten in der Nähe das Mobilgerät des Patienten als Empfänger auswählen.
- [PK4030] Digitale Arztbriefe können entweder per Drag and Drop oder über einen Explorer in die Desktop Anwendung geladen werden.
- [PK4040] Die Daten werden auf Knopfdruck an das Mobilgerät des Patienten gesendet.
- [WK4010] Die Versichertennummer, die in einem Arztbrief auf dem Computer des Arztes eingetragen ist, wird vor dem Senden mit der in der myMD App des Patienten hinterlegten Versichertennummer abgeglichen und nur bei Übereinstimmung wird der Arztbrief gesendet.

## **Kompatibilität**

- [WK5020] Die Desktop Anwendung wird zusätzlich von macOS 10.12 (und höher) unterstützt.
- [WK5030] Die myMD App kann Arztbriefe im .pdf Dateiformat anzeigen.
- [WK5040] Die myMD App kann Arztbriefe im .csv Dateiformat anzeigen.

### 3.4 Klassendiagramm (Implementierungsphase)



Aufgrund der enormen Größe der Grafik lässt sich diese über folgenden QR-Code online abrufen, um bequem alle Details des Klassendiagrammes begutachten zu können.



Abbildung 3.16: QR Code zur Online-Variante des myMD Klassendiagrammes



# 4 Ergebnisse der Phase

## 4.1 Ablauf

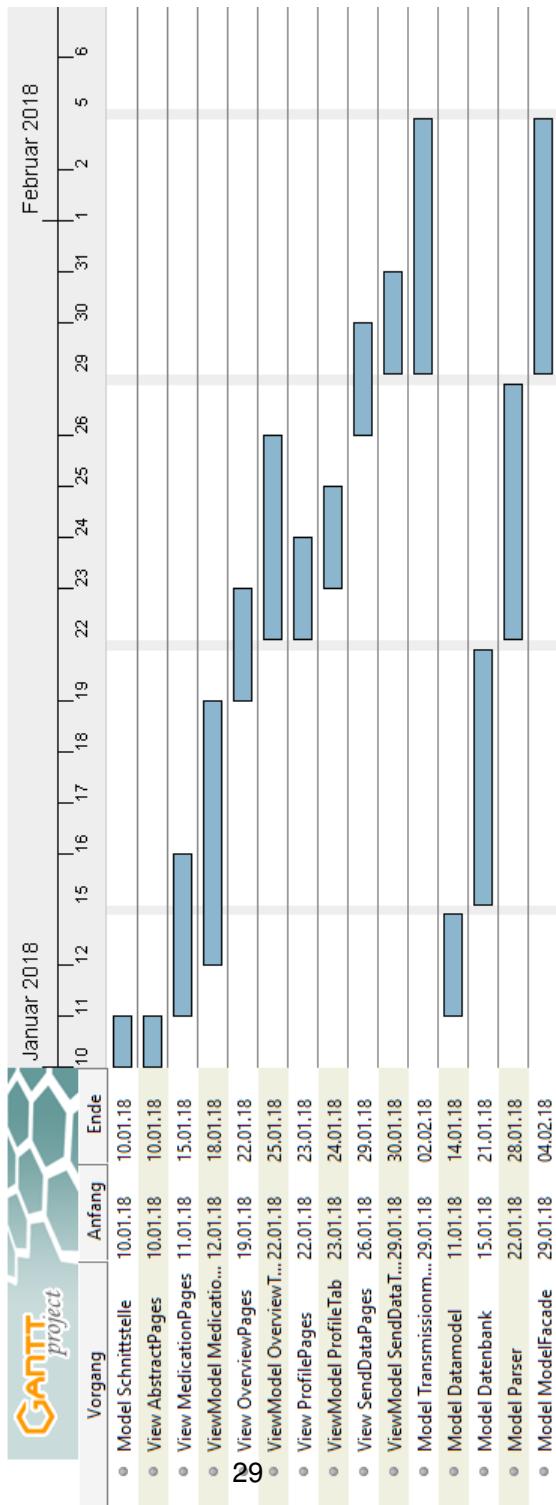


Abbildung 4.1: Gantt Diagramm der Implementierungsphase

Von dem, im Entwurf, geplanten Ablauf wurde hauptsächlich in der **View** und dem **ViewModel** abgewichen. Denn hier, stellten wir fest, ist es sinnvoller eine Page der **View** parallel zum entsprechenden **ViewModel** zu implementieren. Da wir mit dem **MedicationTab** anfingen, brauchten wir hier länger als geplant. Jedoch holten wir die verlorene Zeit bei der Implementierung des **OverviewTabs** und des **ProfileTabs** wieder zurück. Denn hier konnten wir das Know-How, das wir uns beim **MedicationTab** angeeignet haben, gut anwenden.

Im **Model** gab es keine nennenswerten Abweichungen vom Zeitplan.

## 4.2 Zukunftspläne

Unsere höchste Priorität ist es die Desktop Anwendung funktionsfähig zu machen. Da durch sie unsere App erst vollständig benutzbar wird.

Dann folgt die Vorbereitung für den *Microsoft ImagineCup*, für den wir noch ein paar Bugs aus der App beseitigen und Englisch als zweite Sprache anbieten wollen. Außerdem sollen für die Vorbereitung ein paar der nicht umgesetzten Kriterien (z.B. Such/Sortierfunktion im **OverviewTab** und **MedicationTab**) implementiert werden.

Danach würden wir die App neu strukturieren. D.h. der **OverviewTab** wird mit dem **MedicationTab** zusammengelegt. Dies soll eine höhere Informationsdichte bewirken.

# Glossar

**Anamnese** Die Anamnese (von altgriechisch ἀνάμνησις, deutsch ‚Erinnerung‘) ist die professionelle Erfragung von potenziell medizinisch relevanten Informationen durch Fachpersonal (z. B. einen Arzt). 30

**App** Als Mobile App (auf Deutsch meist in der Kurzform die App, eine Abkürzung für den Fachbegriff Applikation) wird eine Anwendungssoftware für Mobilgeräte beziehungsweise mobile Betriebssysteme bezeichnet. 4, 5, 6, 23, 24, 25, 30

**Arztbrief** Der Arztbrief, oft synonym als Epikrise, Entlassungsbefund, Patientenbrief oder Befundbericht bezeichnet, ist ein Transferdokument für die Kommunikation zwischen Ärzten. Der Arztbrief gibt einen zusammenfassenden Überblick über den Status des Patienten bei der Entlassung, einen Rückblick über den Krankheitsverlauf, die veranlasste Therapie, eine Interpretation des Geschehens zum Krankheitsverlauf im speziellen Fall. 4, 5, 6, 23, 24, 25, 30

**Bluetooth** Bluetooth ist ein in den 1990er Jahren durch die Bluetooth Special Interest Group (SIG) entwickelter Industriestandard gemäß IEEE 802.15.1 für die Datenübertragung zwischen Geräten über kurze Distanz per Funktechnik (WPAN). Dabei sind verbindungslose sowie verbindungsbehaftete Übertragungen von Punkt zu Punkt und Ad-hoc- oder Piconetze möglich. 4, 30

**Cloud** Die Cloud ist keine physische Größe, sondern ein riesiges Netzwerk aus Remote-servers, die über die ganze Welt verteilt aber miteinander verbunden sind, damit sie als ein einziges großes Ökosystem funktionieren können. 30

**Desktop Anwendung** Als Desktop Anwendungen (auch Anwendungsprogramm, kurz Anwendung oder Applikation; englisch application software, kurz App) werden Computerprogramme bezeichnet, die genutzt werden, um eine nützliche oder gewünschte nicht systemtechnische Funktionalität zu bearbeiten oder zu unterstützen. Sie dienen der „Lösung von Benutzerproblemen“. 4, 5, 6, 23, 24, 25, 30

**Drag and Drop** Drag and Drop, oft auch Drag'n'Drop, deutsch „Ziehen und Ablegen“, ist eine Methode zur Bedienung grafischer Benutzeroberflächen von Rechnern durch das Bewegen grafischer Elemente mittels eines Zeigegerätes. Ein Element wie z. B. ein Piktogramm kann damit gezogen und über einem möglichen Ziel losgelassen werden. Dieses kann zum Beispiel markierter Text oder das Symbol einer Datei sein . 5, 24, 30

**Medikament** Arzneimittel oder gleichbedeutend Medikamente (lateinisch medicamentum „Heilmittel“) sind Stoffe oder Stoffzusammensetzungen, die „zur Heilung oder zur Verhütung menschlicher oder tierischer Krankheiten bestimmt sind“ oder sich dazu eignen, physiologische Funktionen zu beeinflussen oder eine medizinische Diagnose zu ermöglichen. 4, 24, 30

**NFC** Die Nahfeldkommunikation (Near Field Communication, abgekürzt NFC) ist ein auf der RFID-Technik basierender internationaler Übertragungsstandard zum kontaktlosen Austausch von Daten per elektromagnetischer Induktion mittels loser gekoppelter Spulen über kurze Strecken von wenigen Zentimetern. 4, 23, 30

**Nutzer** Ein Benutzer (auch Endbenutzer, Bediener oder kurz Nutzer genannt sowie englisch User) ist eine Person, die ein Hilfs- oder Arbeitsmittel zur Erzielung eines Nutzens verwendet, beispielsweise für eine Zeitersparnis oder Kostensenkung. 4, 5, 24, 30

**Pop-Up** Ein Pop-up (von englisch to pop up, „plötzlich auftauchen“) ist ein Element einer grafischen Benutzeroberfläche. In der Regel werden Pop-ups eingesetzt, um zusätzliche Inhalte anzuzeigen oder eine bestimmte Interaktion abzufragen. Typischerweise „springen“ Pop-ups auf und überdecken dabei andere Teile der Benutzeroberfläche. 30

**Server** Ein Server (englisch server, wörtlich Diener oder Bediensteter, im weiteren Sinn auch Dienst) ist ein Computerprogramm oder ein Computer, der Computerfunktionalitäten wie Dienstprogramme, Daten oder andere Ressourcen bereitstellt, damit andere Computer oder Programme („Clients“) darauf zugreifen können. 30

**Tab** Eine Registerkarte, auch Reiter oder Tab genannt, ist ein Steuerelement einer grafischen Benutzeroberfläche, das einem Registerblatt aus Aktenschranken nachempfunden wurde . 4, 24, 30

**Versichertennummer** Die Krankenversichertennummer dient der Identifikation des Versicherten bei einer Krankenversicherung. Die Krankenversichertennummer wird benötigt, damit Leistungserbringer, z. B. Ärzte oder Zahnärzte ihre Leistungen mittels der Krankenversicherungskarte, über die Kassenärztlichen Vereinigungen, mit der zuständigen Krankenkasse abrechnen können. 5, 24, 30

# Abbildungsverzeichnis

2.1	myMD Klassendiagramm der Entwurfsphase . . . . .	7
2.2	Gantt Diagramm der Entwurfsphase . . . . .	8
3.1	IHI7ParserHelper Schnittstelle . . . . .	10
3.2	Klassen zum dateiformatunabhängigen Parsen . . . . .	11
3.3	Enum Picker Klassen . . . . .	12
3.4	IModelFacade Schnittstelle . . . . .	14
3.5	IMedication Schnittstelle . . . . .	15
3.6	IEntityDatabase Schnittstelle . . . . .	16
3.7	EntityFactory Klasse . . . . .	17
3.8	DoctorsLetterGroupDoctorsLetter Klasse . . . . .	18
3.9	IFileHelper Schnittstelle . . . . .	19
3.10	OverviewViewModel Klasse . . . . .	20
3.11	DetailedDoctorsLetterViewModel Klasse . . . . .	20
3.12	DoctorsLetterViewModel Klasse . . . . .	21
3.13	MedicationViewModel Klasse . . . . .	22
3.14	MedicineViewModel Klasse . . . . .	23
3.15	myMD Klassendiagramm der Implementierungsphase . . . . .	26
3.16	QR Code zur Online-Variante des myMD Klassendiagrammes . . . . .	27
4.1	Gantt Diagramm der Implementierungsphase . . . . .	29