



myMD

ENTWURF

Praxis der Softwareentwicklung WS2017/2018

Philipp Pelcz, Philipp Karcher, Jan-Luca Vettel

supervised by
Marc Aurel Kiefer

17. März 2018

Inhaltsverzeichnis

1 Einleitung	3
2 Systemaufbau	4
2.1 Systemarchitektur	4
2.2 Subsysteme	5
2.3 Subsystem-Schnittstellen	8
3 Paketführer	18
3.1 Paketbeschreibung	18
3.2 Benutzt-Relation	26
4 Klassenbeschreibung	27
4.1 View	27
4.2 ViewModel	38
4.3 Model	52
5 Daten	77
5.1 Datenspeicherung	77
5.2 Datenübertragung	77
6 Dynamik und Ablauf	79
6.1 Aktionen	79
6.2 UI-Navigation	84
6.3 UI-Seiten	86
7 Anwendung für Ärzte	92
8 Klassendiagramm	93

1 Einleitung

Dieses Dokument für den Entwurf der Applikation **myMD** ist im Rahmen des Softwarepraktikums PSE am Karlsruher Institut für Technologie entstanden.

Der Entwurf wird dabei nach dem Top-Down Prinzip vorgestellt: zunächst die Systemarchitektur , die die Applikation in mehrere Subsysteme unterteilt (*Abschnitt 2*). Diese Subsysteme bestehen wiederum aus Paketen(*Abschnitt 3*), die dann die einzelnen Klassen und Schnittstellen enthalten(*Abschnitt 4*). In den jeweiligen Abschnitten werden dann alle Teile des Systems genau beschrieben

Daraufhin wird die Speicherung und Übertragung der in der Applikation verwendeten Daten beschrieben (*Abschnitt 5*).

Die Dynamik der Applikation, also das Zusammenspiel der verschiedenen Subsysteme, Pakete und Klassen, wird durch Sequenz- und Zustandsdiagramme dargestellt (*Abschnitt 6*). Hier ist auch eine Beschreibung der Benutzeroberfläche und der Navigation dieser zu finden.

Darauf folgt eine Beschreibung der Variante der Applikation für Ärzte statt Patienten (*Abschnitt 7*).

Abschließend wird noch eine Ansicht des gesamten Klassendiagramms der Applikation geboten (*Abschnitt 8*).

2 Systemaufbau

2.1 Systemarchitektur

Die Logik des Systems verteilt sich auf beliebig viele Partner, die unter Umständen miteinander kommunizieren können. Der Aufbau eines solchen Partners wird nun im Folgenden beschrieben:

Ein Partner ist fest an ein von *Xamarin.Forms* unterstütztes Gerät gebunden. Die Architektur trennt sich dabei in drei so unabhängig wie möglich agierende Subsysteme nach dem MVVM-Prinzip, um die Anwendung möglichst plattformübergreifend zu gestalten:

Das **Model** enthält die Anwendungslogik und ist komplett unabhängig von den restlichen Subsystemen. Hier findet auch die Kommunikation mit anderen Partnern statt.

Die **View** beinhaltet die Gestaltung der Benutzeroberfläche. Dabei wird das Seitenlayout jeder UI-Seite in .XAML modelliert.

Das **ViewModel** ist das Verbindungsstück zwischen Model und View. Es führt Zugriffe auf das Model über dessen Schnittstellen aus, reicht dem View anzuzeigende Informationen über das *Xamarin.Forms* Data-Binding Framework und implementiert die Benutzeroberflächenlogik.

2.2 Subsysteme

2.2.1 View

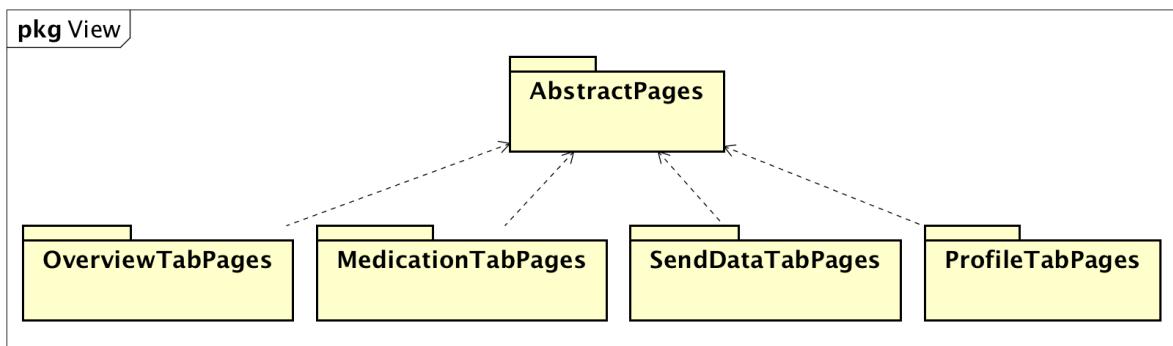


Abbildung 2.1: Pakete des Subsystems View

Der View besteht aus allen Klassen und Dateien, die benötigt werden, um die Benutzeroberfläche zu modellieren. Er dient als Eingabe- und Ausgabeschnittstelle der Anwendung. Hierbei erfolgt die Modellierung hauptsächlich (Xamarin.Forms konform) in XAML-Dateien, wodurch sich leicht eine einheitliche, plattformübergreifende Oberfläche erstellen lässt. Da es sich bei XAML jedoch um eine "*Markup Language*" handelt, finden diese Dateien im obigen UML-Diagramm keine Darstellung, lediglich die einer XAML-Datei zugehörigen C#-Dateien sind dort repräsentiert. Die einzelnen Ansichten der Anwendung, sprich die UI-Zustände, werden in separaten Klassen erstellt und, um für eine übersichtlichere Struktur zu sorgen, werden diese abhängig ihrer jeweiligen Tab-Zugehörigkeit, in Paketen eingesortiert.

OverviewTabPage enthält all jene Ansichten (Pages), die sich innerhalb des Tabs *Übersicht* befinden. Dies umfasst beispielsweise die Übersicht aller gespeicherter Arztbriefe.

Das Paket **MedicationTabPage** verfügt über alle Pages, die innerhalb des Tabs *Medikation* vom Nutzer erreichbar sind. Dazu zählt unter anderem die Auflistung der gespeicherten Medikationen.

Alle Ansichten innerhalb des Tabs *Senden* werden in Klassen innerhalb des Pakets **SendDataTabPage** modelliert.

Zu guter Letzt werden alle dem Tab *Profil* zugehörigen Ansichten innerhalb des Pakets **ProfileTabPage** verwaltet. Dies betrifft Ansichten wie die Darstellung des eigenen Nutzerprofils.

Um Gemeinsamkeiten aller Ansichten nicht redundant definieren zu müssen, existiert zusätzlich das Paket **AbstractPages**, in welchem beispielsweise grundlegende Attribute aller Pages definiert werden können. Text- oder Hintergrundfarben wären hierfür ebenso Beispiele wie auch die Organisation der Tabbar.

2.2.2 ViewModel

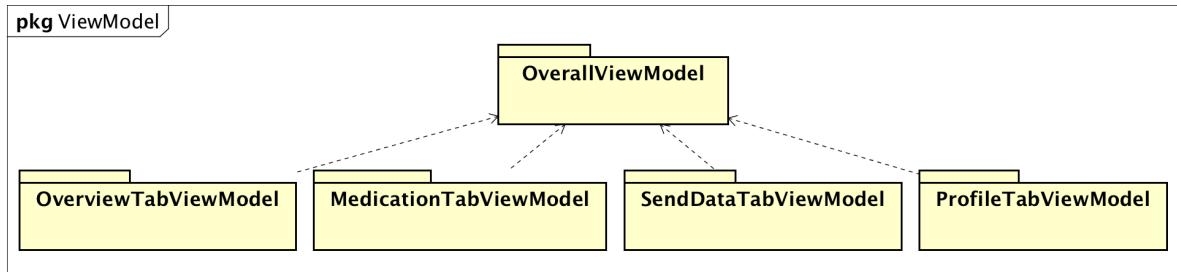


Abbildung 2.2: Pakete des Subsystems ViewModel

Das ViewModel stellt den Vermittler zwischen View und Model dar. Es sorgt für die Aktualisierung sowohl der View als auch des Models. Der Datenaustausch zwischen View und ViewModel erfolgt über das sogenannte *Data Binding*, die Kommunikation mit dem Model erfolgt über die Model-eigenen Schnittstellen.

Ähnlich wie im Aufbau des Views, werden auch im ViewModel aus Gründen der Übersichtlichkeit die einzelnen Klassen je nach Tab-Zugehörigkeit gegliedert.

OverviewTabViewModel kümmert sich hauptsächlich um alle Bestandteile eines Arztbriefes, fordert darzustellende Arztbriefe an und meldet Änderungen an bestehenden Daten.

Für Ähnliches ist auch **MedicationTabViewModel** zuständig: Hier werden alle hinterlegte, neu erzeugte oder geänderte Medikationen übermittelt.

Das Model über zu übertragende Dateien und den gewünschten Empfänger zu informieren, ist die Hauptaufgabe des Pakets **SendDataTabViewModel**.

ProfileTabViewModel letztlich ist unter anderem dafür zuständig, dem Model Änderungen im Profil des Nutzers und dem View das darzustellende Profil zu übermitteln.

2.2.3 Model

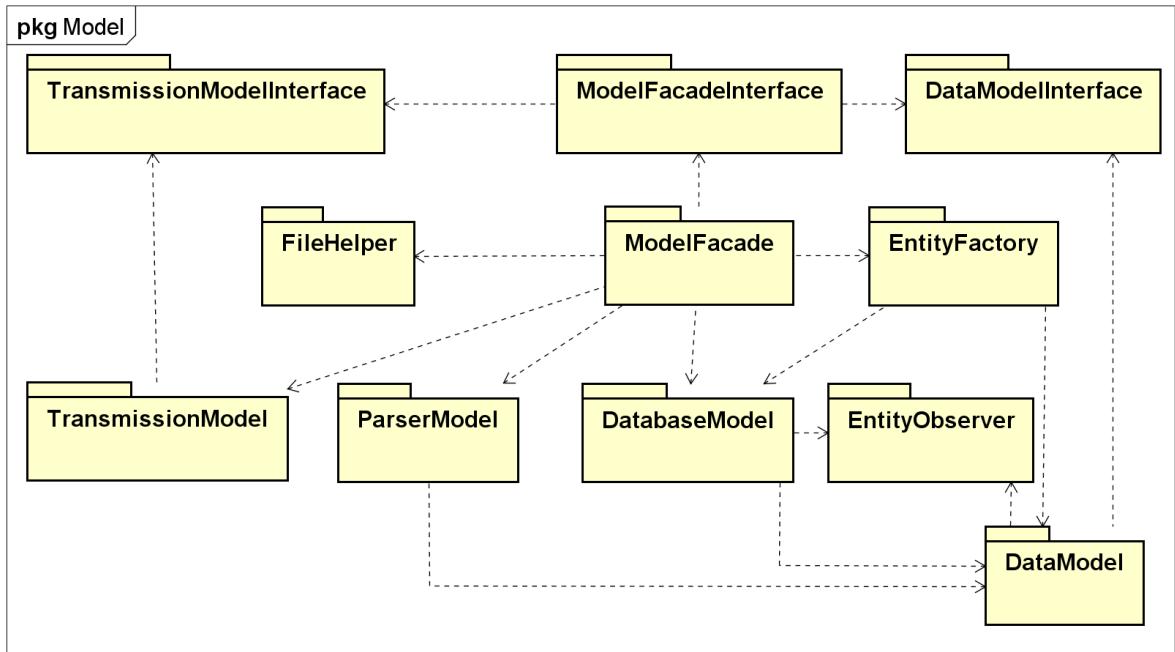


Abbildung 2.3: Pakete des Subsystems Model

Das Model enthält alle Klassen und Methoden zum Speichern, Laden, Ändern, Senden und Empfangen der medizinischen Daten des Benutzers.

Das Paket **DataModel** enthält dabei die Objekte zur Modellierung der in der Applikation enthaltenen Daten. Um über Änderungen an diesen Daten informiert zu werden, kann sich eine Klasse durch Implementierung der **EntityObserver** Schnittstelle anmelden. Die Speicherung dieser Daten geschieht über das Paket **DatabaseModel**. Außerdem können Daten über dieses Paket gesucht und abgefragt werden.

Für das Senden und Empfangen von Dateien ist das **TransmissionModel** Paket zuständig. Da die Daten in einem anderen Format übertragen, als sie intern in der Applikation verwendet werden, muss das Paket **ParserModel** benutzt um zwischen den Formaten zu konvertieren.

Letztlich wird noch, um plattformübergreifend auf das lokale Dateisystem zugreifen zu können, das Paket **FileHelper** benötigt.

Um den Zugriff auf das Model zu vereinfachen wird das Fassaden Muster verwendet. Das

Paket **ModelFacade** nimmt Anfragen an das Model entgegen und delegiert dann an die restlichen Pakete im Model.

2.3 Subsystem-Schnittstellen

2.3.1 ModellInterface

TransmissionModellInterface

Das TransmissionModellInterface beinhaltet alle Klassen und Methoden, die zur Darstellung eines Empfängers benötigt werden.

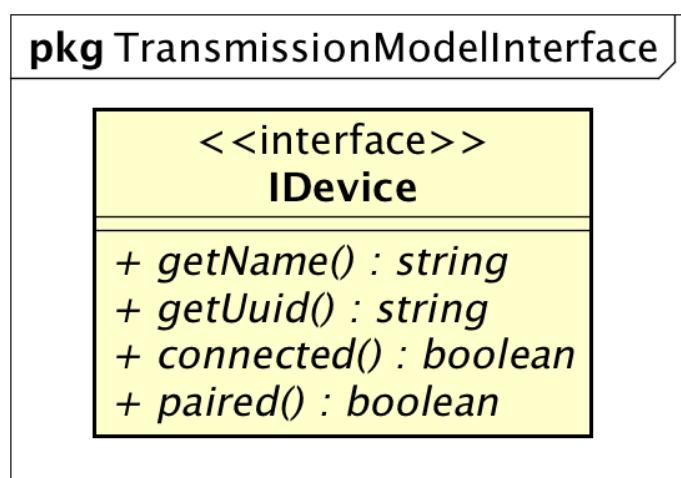


Abbildung 2.4: TransmissionModel Schnittstellen

getName() : string

Gibt den Namen der Entität zurück.

getUuid() : string

Gibt die UUID der Entität zurück.

connected() : boolean

Gibt den Verbindungsstatus der Entität zurück.

paired() : boolean

Gibt den Kopplungsstatus der Entität zurück.

DataModelInterface

Hier sind die Schnittstellen der medizinischen Daten enthalten, die angezeigt werden sollen. Die hier aufgeführten Methoden umfassen daher das Abfragen von Informationen über die Daten aber auch einfache Operationen die auf diesen Daten ausgeführt werden, wie zum Beispiel das Ändern dieser Informationen.

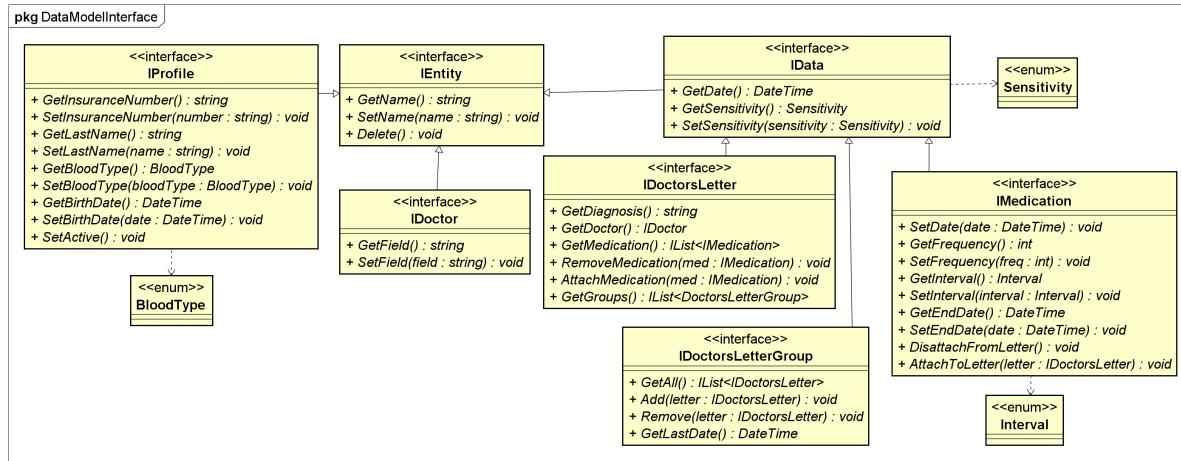


Abbildung 2.5: DataModel Schnittstellen

IEntity

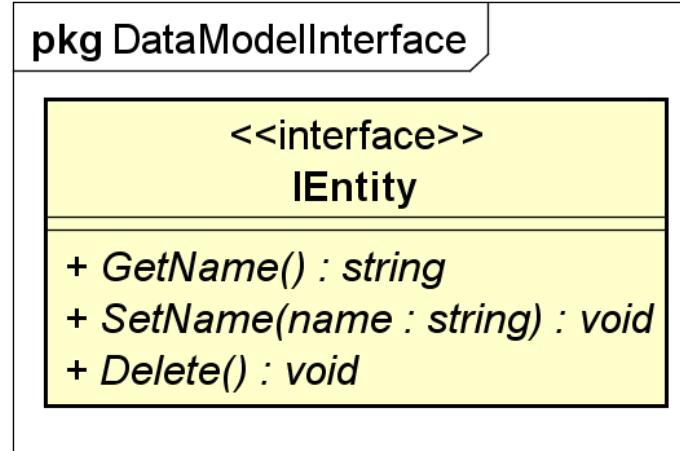


Abbildung 2.6: IEntity Schnittstelle

GetName() : string

Gibt den Namen der Entität zurück.

SetName(name : string) : void

Setzt den Namen der Entität auf *name*.

Delete() : void

Löscht die Entität und löst alle Assoziationen von ihr auf.

IData

Die Schnittstelle **IData** erweitert **IEntity**.

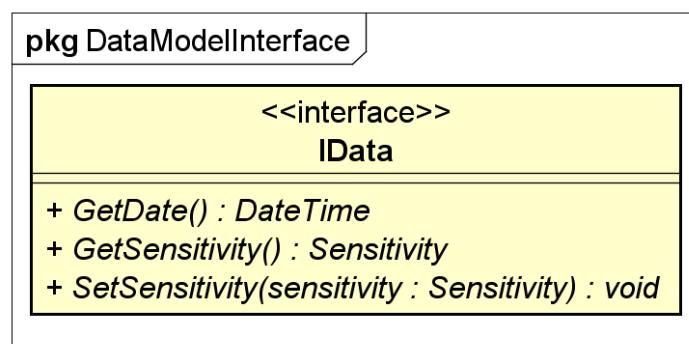


Abbildung 2.7: IData Schnittstelle

GetDate() : DateTime

Gibt das Datum zurück, von dem die Daten stammen.

GetSensitivity() : Sensitivity

Gibt die Sensitivitätsstufe der Daten zurück.

SetSensitivity(sensitivity : Sensitivity) : void

Setzt die Sensitivitätsstufe der Daten auf *sensitivity*.

IDoctorsLetter

Die Schnittstelle **IDoctorsLetter** erweitert **IData**.

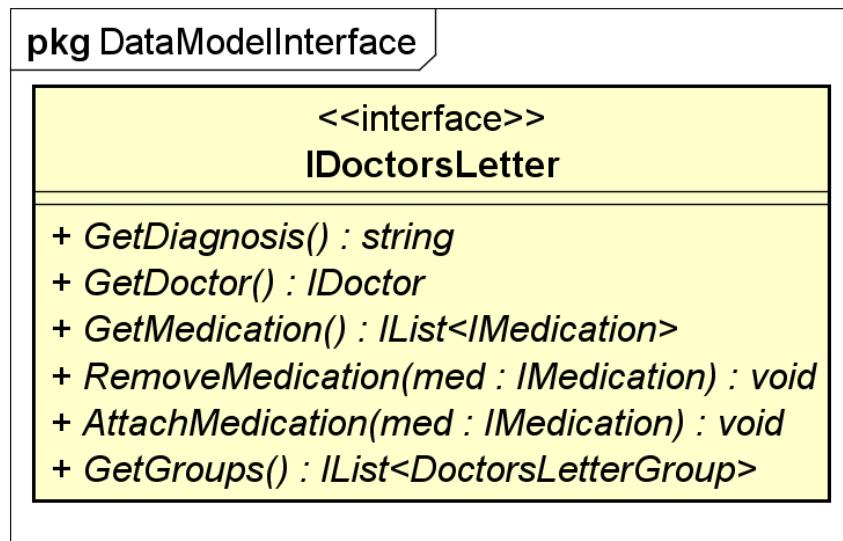


Abbildung 2.8: IDoctorsLetter Schnittstelle

GetDiagnosis() : string

Gibt die Diagnose im Arztbrief zurück.

GetDoctor() : IDoctor

Gibt den Doktor zurück, von dem der Arztbrief stammt.

GetMedication() : IList<IMedication>

Gibt alle Medikationen zurück, die von diesem Arztbrief stammen.

RemoveMedication(med : IMedication) : void

Entfernt die Medikation *med* aus der Liste der Medikationen die von diesem Arztbrief stammen.

AttachMedication(med : IMedication) : void

Verbindet die Medikation *med* mit diesem Arztbrief.

GetGroups() : IList<DoctorsLetterGroup>

Gibt alle Gruppen von Arztbriefen zurück, in denen dieser Arztbrief enthalten ist.

RemoveFromGroup(group : IDoctorsLetterGroup) : void

Entfernt diesen Arztbrief aus der Gruppe *group*.

GetFilename() : string

Gibt den Namen der Originaldatei dieses Arztbriefs zurück.

IDoctorsLetterGroup

Die Schnittstelle **IDoctorsLetter** erweitert **IData**.

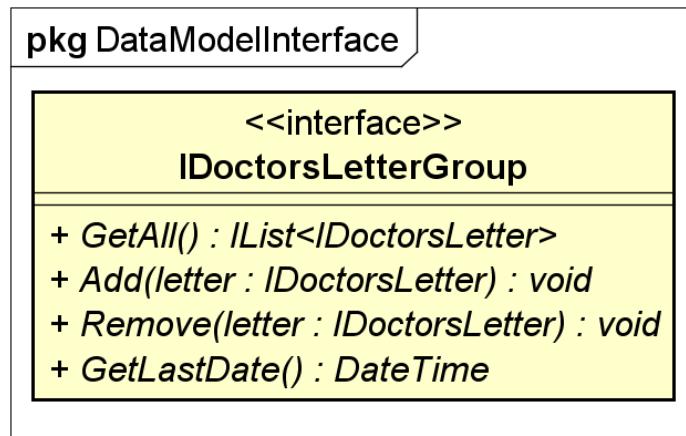


Abbildung 2.9: IDoctorsLetterGroup Schnittstelle

GetAll() : IList<IDoctorsLetter>

Gibt alle Arztbriefe in der Gruppe zurück.

Add(letter : IDoctorsLetter) : void

Fügt den Arztbrief *letter* zu dieser Gruppe hinzu.

Remove(letter : IDoctorsLetter) : void

Entfernt den Arztbrief *letter* aus dieser Gruppe.

GetLastDate() : DateTime

Gibt das aktuellste Datum aller Arztbriefe in dieser Gruppe zurück.

IMedication

Die Schnittstelle **IMedication** erweitert **IData**.

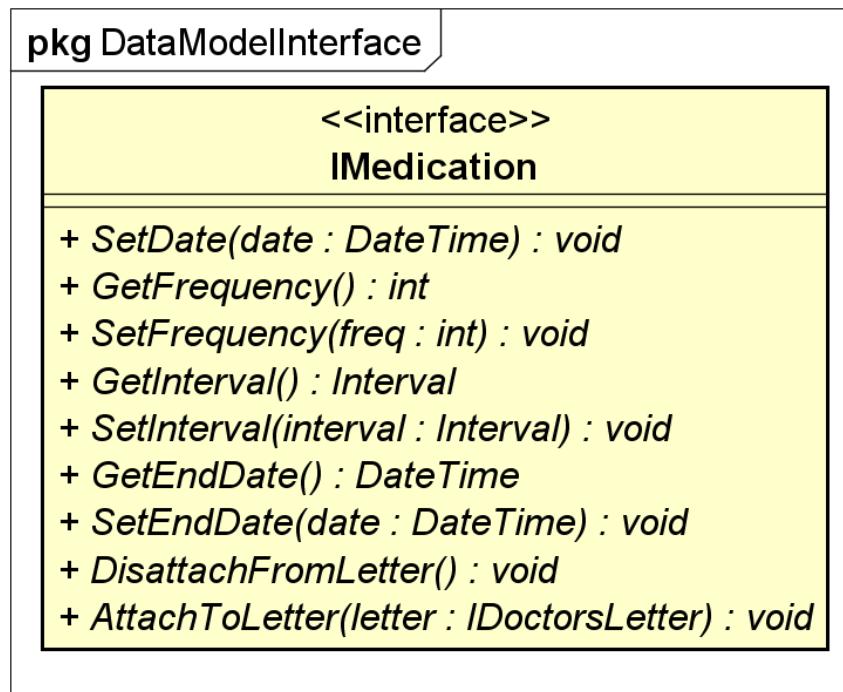


Abbildung 2.10: IMedication Schnittstelle

SetDate(date : DateTime) : void

Setzt das Datum an dem diese Medikation angefangen wurde auf das Datum *date*.

GetFrequency() : int

Gibt zurück wie oft die Medikation in ihrem Intervall genommen werden soll.

SetFrequency(freq : int) : void

Setzt die Häufigkeit in der die Medikation in ihrem Intervall genommen werden soll auf *freq*.

GetInterval() : Interval

Gibt zurück in welchem Zeitintervall die Medikation eingenommen werden soll. Das Intervall und die Häufigkeit zusammen ergeben dann eine tatsächliche Häufigkeit in der die Medikation eingenommen werden soll (z.B. 3 mal pro Tag).

SetInterval(interval : Interval) : void

Setzt das Zeitintervall in dem die Medikation eingenommen werden soll auf *interval*.

GetEndDate() : DateTime

Gibt das Datum an dem die Medikation abgesetzt werden soll zurück.

SetEndDate(date : DateTime) : void

Setzt das Datum an dem die Medikation abgesetzt werden soll auf *date*.

DisattachFromLetter() : void

Löst die Verbindung der Medikation zu ihrem Arztbrief auf.

AttachToLetter(letter : IDoctorsLetter) : void

Verbindet diese Medikation mit dem Arztbrief *letter*.

IProfile

Die Schnittstelle **IProfile** erweitert **IEntity**.

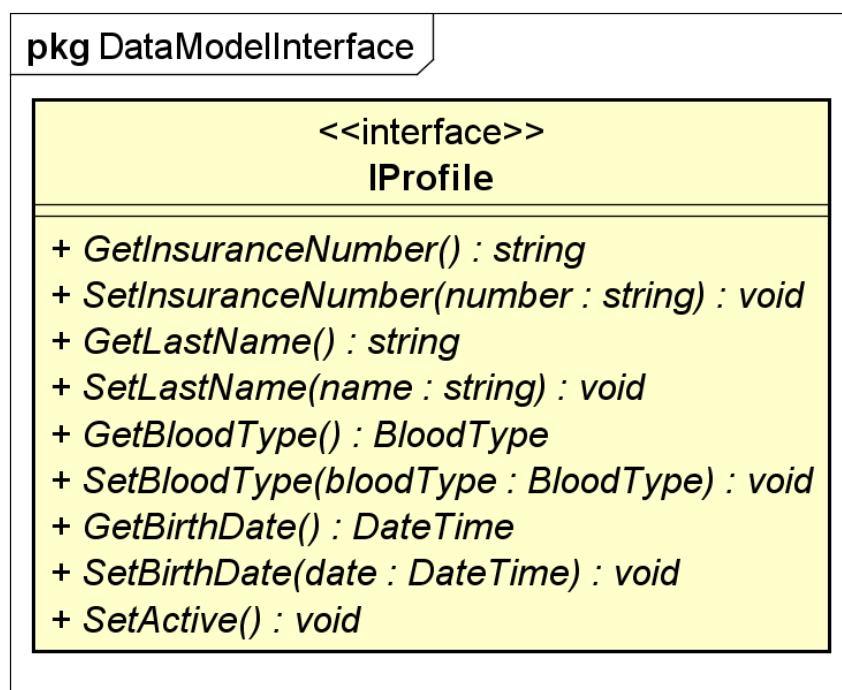


Abbildung 2.11: IProfile Schnittstelle

GetInsuranceNumber() : string

Gibt die Versicherungsnummer dieses Profils zurück.

SetInsuranceNumber(number : string) : void

Setzt die Versicherungsnummer dieses Profils auf *number*.

GetLastName() : string

Gibt den Nachnamen dieses Profils zurück.

SetLastName(name : string) : void

Setzt den Nachnamen dieses Profils auf *name*.

GetBloodType() : BloodType

Gibt die Blutgruppe dieses Profils zurück.

SetBloodType(bloodType : BloodType) : void

Setzt die Blutgruppe dieses Profils auf *bloodType*.

GetBirthDate() : DateTime

Gibt das Geburtsdatum dieses Profils zurück.

SetBirthDate(date : DateTime) : void

Setzt das Geburtsdatum dieses Profils auf *date*.

SetActive() : void

Setzt dieses Profil als das aktive Profil.

IDoctor

Die Schnittstelle **IDoctor** erweitert **IEntity**.

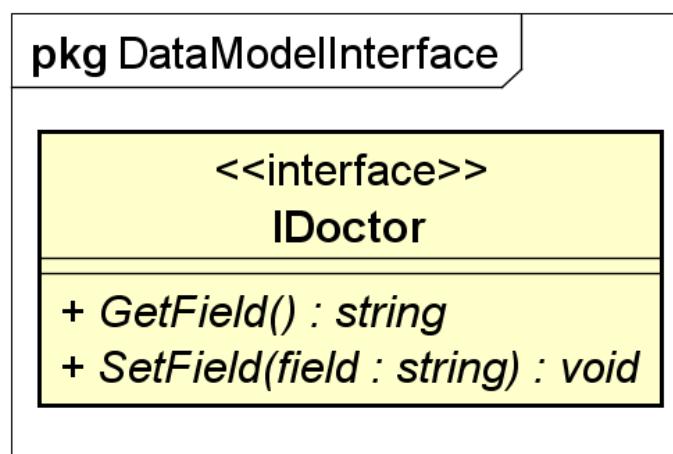


Abbildung 2.12: IDoctor Schnittstelle

GetField() : string

Gibt das Fachgebiet dieses Doktors zurück.

SetField(field : string) : void

Setzt das Fachgebiet dieses Doktors auf *field*.

Enumerations

Die Enumerationen *Sensitivity*, *Interval* und *BloodType* müssen Subsystemen die auf die hier aufgeführten Daten zugreifen ebenfalls bekannt sein, um diese anzeigen zu können.

ModelFacadeInterface

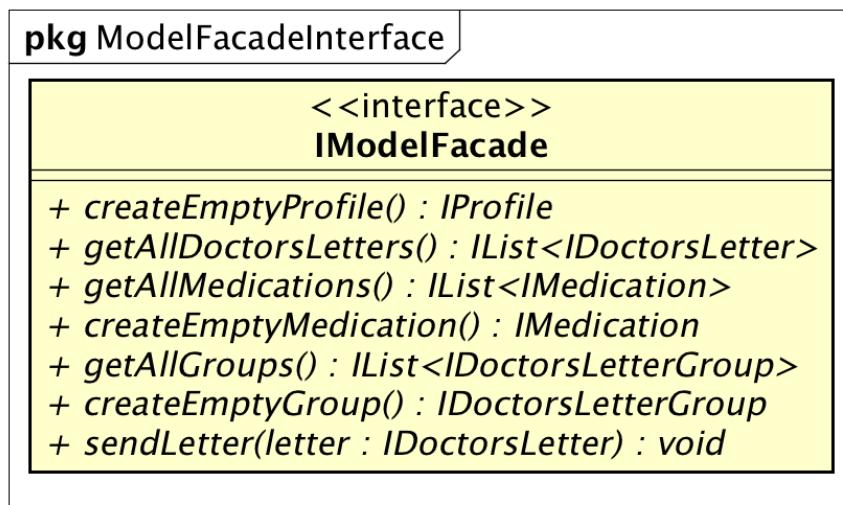


Abbildung 2.13: ModelFacade Schnittstelle

Dies ist die Haupteinstiegsstelle für andere Subsysteme (hier nur das ViewModel) in das Model. Als Ein- und Ausgabeparameter werden die zuvor aufgeführten Schnittstellen verwendet. Alle komplexeren Operationen die nicht schon von diesen Schnittstellen übernommen werden, sind hier enthalten:

SetActiveProfile(profile : IProfile) : void

Wechselt das aktive Profil zum Profil *profile*.

CreateEmptyProfile() : IProfile

Legt ein neues Profil an, das noch keine Informationen und Daten enthält und gibt dieses

zurück. Wechselt außerdem das aktive Profil zu diesem Profil.

GetAllDoctorsLetters() : IList<IDoctorsLetter>

Fordert alle Arztbriefe des aktiven Profils an und gibt diese zurück.

GetAllMedications() : IList<IMedication>

Fordert alle Medikationen des aktiven Profils an und gibt diese zurück.

CreateEmptyMedication() : IMedication

Legt eine neue Medikation an, die noch keine Informationen enthält und gibt diese zurück.

GetAllGroups() : IList<IDoctorsLetterGroup>

Fordert alle Gruppen von Arztbriefen des aktiven Profils an und gibt diese zurück.

CreateEmptyGroup() : IDoctorsLetterGroup

Legt eine neue Gruppe von Arztbriefen an, die noch keine Arztbriefe enthält und gibt diese zurück.

SendLetter(letter : IDoctorsLetter) : void

Bereitet den gewünschten Arztbrief für die Datenübertragung vor und veranlasst diese anschließend.

3 Paketführer

3.1 Paketbeschreibung

3.1.1 View

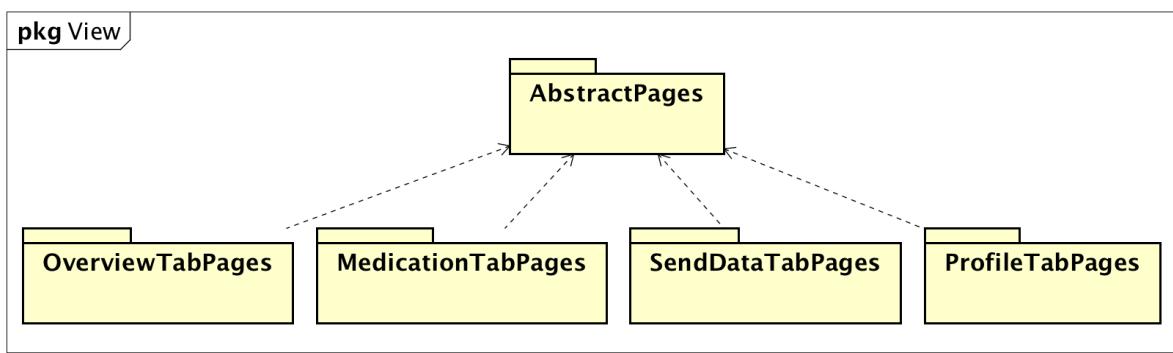


Abbildung 3.1: Pakete der View

OverviewTabPages

Funktion: Dieses Paket enthält alle Pages des Overview Tabs.

Entwurfsentscheidung: Gestaltung der Ansichten des Tabs *Übersicht*

Klassendiagramm: siehe S.94

Enthaltene Klassen: OverviewPage, DetailedDoctorsLetter

MedicationTabPages

Funktion: Dieses Paket enthält alle Pages des Medication Tabs.

Entwurfsentscheidung: Gestaltung der Ansichten des Tabs *Medikation*

Klassendiagramm: siehe S.94

Enthaltene Klassen: MedicationPage, DetailedMedicinePage

SendDataTabPages

Funktion: Dieses Paket enthält alle Pages des SendData Tabs.

Entwurfsentscheidung: Gestaltung der Ansichten des Tabs *Senden*

Klassendiagramm: siehe S.94

Enthaltene Klassen: SendTabPage, SelectDoctorsLettersPage, SelectDevicePage, TransmittingTabPage

ProfileTabPages

Funktion: Dieses Paket enthält alle Pages des Profile Tabs.

Entwurfsentscheidung: Gestaltung der Ansichten des Tabs *Profil*

Klassendiagramm: siehe S.94

Enthaltene Klassen: ProfilePage

AbstractTabPages

Funktion: Dieses Paket enthält alle Klassen zur Verwaltung genereller Eigenschaften der Benutzeroberfläche und der grundsätzlichen Verhaltensweise einiger UI-Elemente.

Entwurfsentscheidung: Kapselung systemumfassender UI-Entscheidungen

Klassendiagramm: siehe S.94

Enthaltene Klassen: CustomContentPage, App

3.1.2 ViewModel

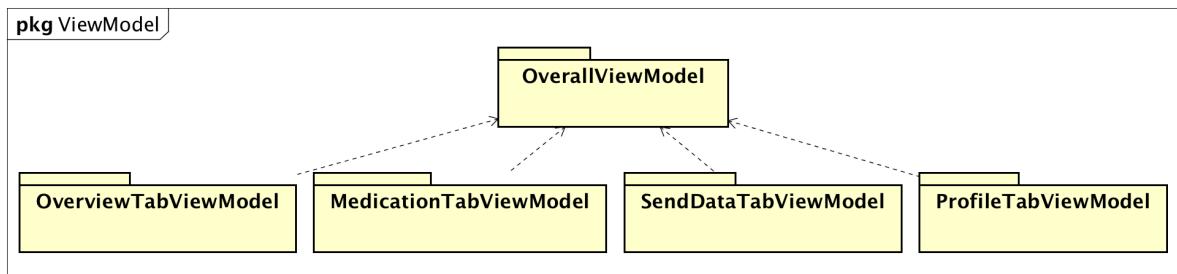


Abbildung 3.2: Pakete des ViewModels

OverallViewModel

Funktion: Dieses Paket dient als Überklasse aller anderen ViewModel Klassen, um redundanten Code zu vermeiden.

Entwurfsentscheidung: Kapselung gemeinsamer Funktionalitäten des ViewModels

Klassendiagramm: siehe S.94

Enthaltene Klassen: OverallViewModel

OverviewTabViewModel

Funktion: Kommunikation zwischen Model und View ermöglichen

Entwurfsentscheidung: Funktions- und Kommunikationsweise der Ansichten des Tabs *Übersicht*

Klassendiagramm: siehe S.94

Enthaltene Klassen: OverviewViewModel, DoctorsLetterViewModel, DetailedDoctorsLetterViewModel

MedicationTabViewModel

Funktion: Kommunikation zwischen Model und View ermöglichen

Entwurfsentscheidung: Funktions- und Kommunikationsweise der Ansichten des Tabs

Medikation

Klassendiagramm: siehe S.94

Enthaltene Klassen: MedicationViewModel, MedicineViewModel, DetailedMedicineViewModel

SendDataTabViewModel

Funktion: Kommunikation zwischen Model und View ermöglichen

Entwurfsentscheidung: Funktions- und Kommunikationsweise der Ansichten des Tabs *Senden*

Klassendiagramm: siehe S.94

Enthaltene Klassen: SendDataViewModel, SelectDoctorsLettersViewModel, SelectDeviceViewModel, TransmittingDataViewModel

ProfileTabViewModel

Funktion: Kommunikation zwischen Model und View ermöglichen

Entwurfsentscheidung: Funktions- und Kommunikationsweise der Ansichten des Tabs *Profil*

Klassendiagramm: siehe S.94

Enthaltene Klassen: ProfileViewModel

3.1.3 Model

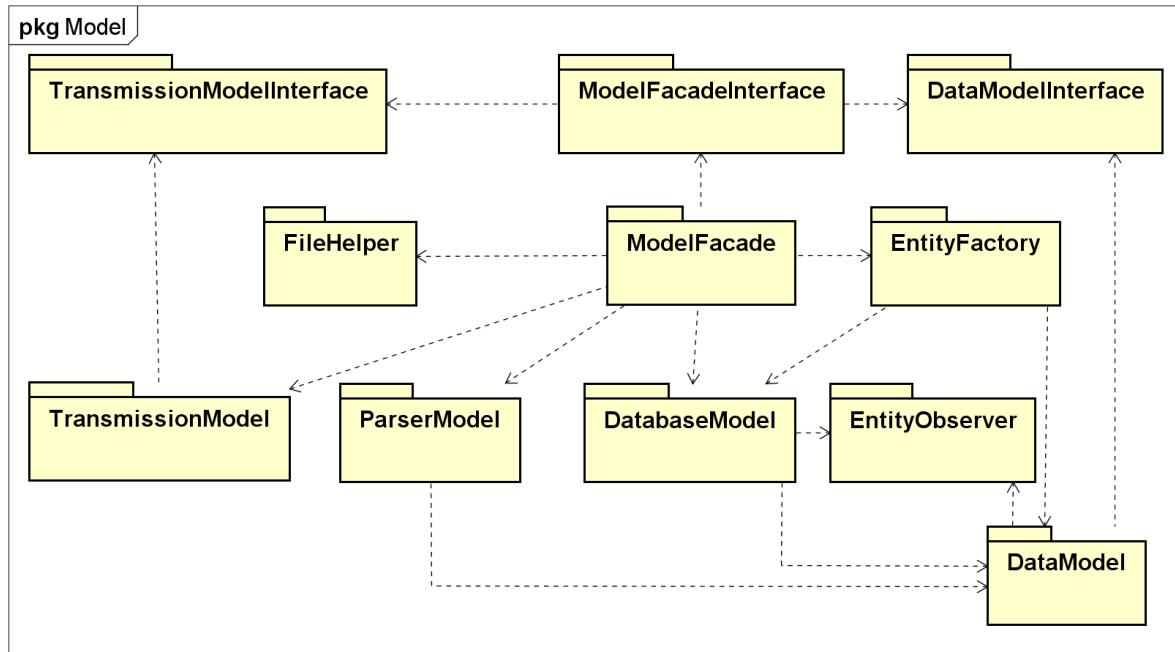


Abbildung 3.3: Pakete des Models

ModelFacade

Funktion: Dieses Paket implementiert die Hauptschnittstelle `IModelFacade` des Models und delegiert Aufgaben an die restlichen Pakete innerhalb des Subsystems. Die `ModelFacade` agiert unabhängig von den anderen Subsystemen und nimmt nur Anfragen entgegen.

Entwurfsentscheidung: Organisation des Models

Subsystem: Model

Klassendiagramm: siehe S.94

Enthaltene Klassen: `ModelFacade`

TransmissionModel

Funktion: Dieses Paket implementiert die Schnittstelle der Datenübertragung. Hauptsächlich werden Daten entgegengenommen, sei es von einem anderen Gerät, welches Daten

übermittelt, oder dem Nutzer, der Daten zur Übertragung freigeben möchte.

Entwurfsentscheidung: Übertragung der Daten

Subsystem: Model

Klassendiagramm: siehe S.94

Enthaltene Klassen: IBluetooth, Bluetooth, Device

DatabaseModel

Funktion: Dieses Paket bietet eine Schnittstelle zur Interaktion mit der Datenbank in der die Daten gespeichert werden. Die Implementierung dieser Schnittstelle leitet größtenteils die Anfragen an die tatsächlich verwendete Datenbank (hier: lokale SQLite Datenbank) weiter.

Entwurfsentscheidung: Art der Datenspeicherung und -abfrage

Subsystem: Model

Klassendiagramm: siehe S.94

Enthaltene Klassen: IEntityDatabase, EntityDatabase

DataModel

Funktion: Dieses Paket enthält die Implementierung der verschiedenen Datentypen, die in **DataModellInterface** beschrieben werden. Diese Klassen dienen jeweils als Vorlage für eine Tabelle der Datenbank und können über das **EntityObserver** Paket beobachtet werden.

Entwurfsentscheidung: Kapselung der verschiedenen Datentypen

Subsystem: Model

Klassendiagramm: siehe S.94

Enthaltene Klassen: Entity, Profile, Doctor, Data, DoctorsLetter, DoctorsLetterGroup, Medication

EntityFactory

Funktion: Dieses Paket enthält eine Schnittstellen um Objekte aus **DataModellInterface** zu erzeugen. Die Implementierung der Schnittstelle erzeugt Objekte aus **DataModel** und fügt sie in die Datenbank ein.

Entwurfsentscheidung: Kapselung der Implementierung und Erstellung der Datentypen

Subsystem: Model

Klassendiagramm: siehe S.94

Enthaltene Klassen: IEntityFactory, EntityFactory

EntityObserver

Funktion: Dieses Paket enthält die Schnittstellen die implementiert werden müssen um Entitäten zu beobachten.

Entwurfsentscheidung: Benachrichtigung der Datenbank über Änderung der Daten

Subsystem: Model

Klassendiagramm: siehe S.94

Enthaltene Klassen: IEntityObservable, IEntityObserver

ParserModel

Funktion: Dieses Paket enthält die Schnittstelle IParseFacade um zwischen Datenformaten zu konvertieren. Die Implementierung der Schnittstelle wählt dann den richtigen Parser aus und delegiert die Konvertierung an diesen.

Entwurfsentscheidung: Unabhängigkeit der lokaler Datenspeicherung und Datenübertragung

Subsystem: Model

Klassendiagramm: siehe S.94

Enthaltene Klassen: IParseFacade, ParseFacade, FileToDatabaseParser, HI7ToDatabaseParser, LetterToFileParser

FileHelper

Funktion: Dieses Paket bietet eine Schnittstelle um plattformübergreifend auf Dateien zugeifen zu können und implementiert diese plattformspezifisch.

Entwurfsentscheidung: Kapselung des plattformspezifischen Dateizugriffs

Subsystem: Model

Klassendiagramm: siehe S.94

Enthaltene Klassen: IFileHelper, FileHelper

3.2 Benutzt-Relation

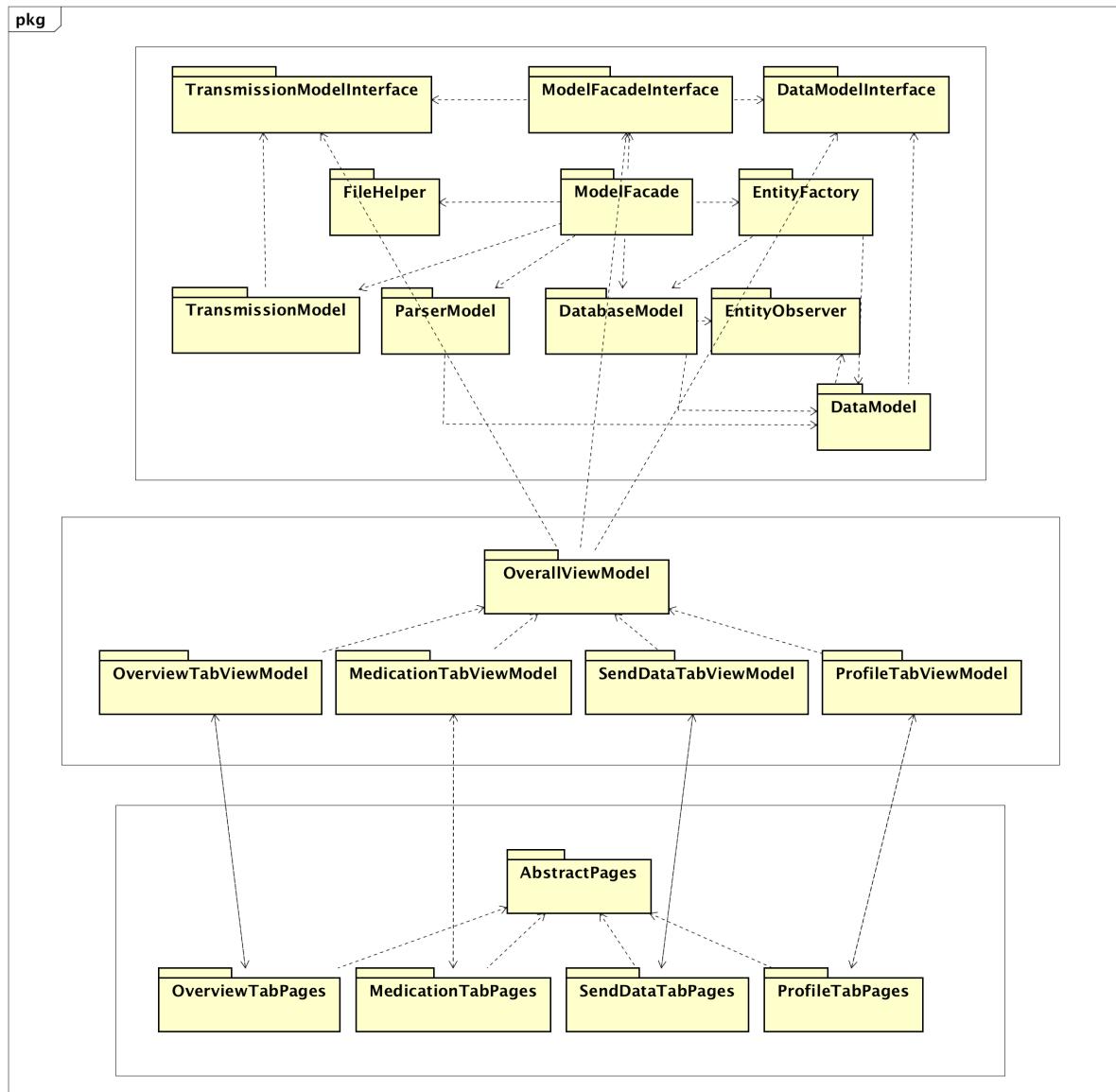


Abbildung 3.4: Benutztrelation des Systems

4 Klassenbeschreibung

Hinweis: Alle hier aufgelisteten Klassendiagramme sind Ausschnitte des Systemklassendiagramms. Viele Attribute und implementierte Methoden aus Interfaces sind deswegen nicht in dem jeweiligen Diagramm zu finden, sondern als Relationen zwischen den Klassen bzw. in dem implementierten Interface.

4.1 View

Hinweis: Die im folgenden aufgelisteten Klassen bestehen, ganz im Sinne der Xamarin.Forms Cross-Plattform Idee, nicht nur aus C#-Klassen, sondern besitzen stets auch eine assoziierte XAML-Datei, die im Normalfall den vollständigen Aufbau der Ansicht definiert.

4.1.1 AbstractPages

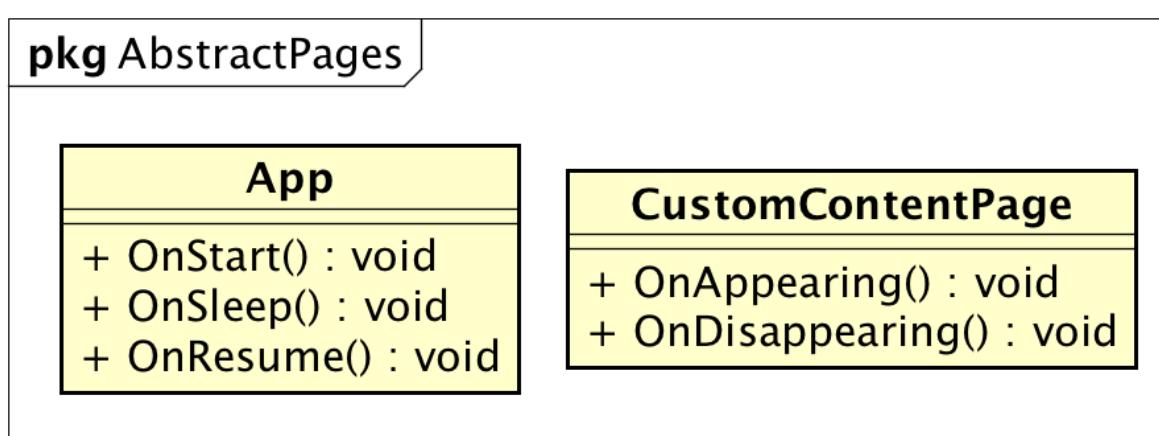


Abbildung 4.1: Klassen des AbstractPages Paket

CustomContentPage

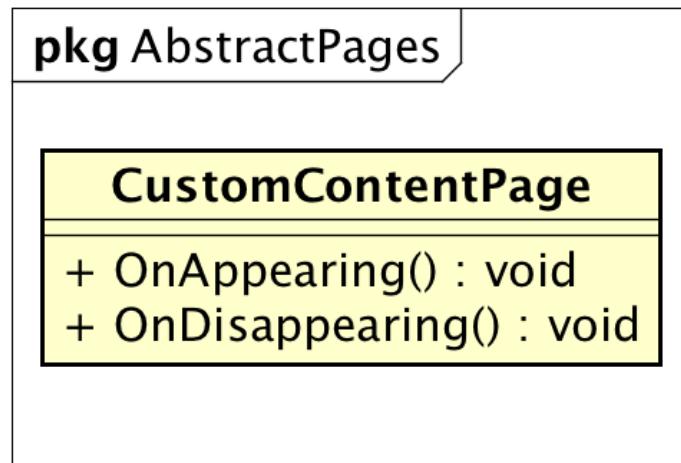


Abbildung 4.2: CustomContentPage Klasse

Beschreibung: Die Klasse **CustomContentPage** legt grundlegende Eigenschaften der gesamten Benutzeroberfläche fest. Dies betrifft beispielsweise Farbgestaltung verschiedener Elemente wie des Hintergrundes, Texte, Tab- und der Statusbar.

App

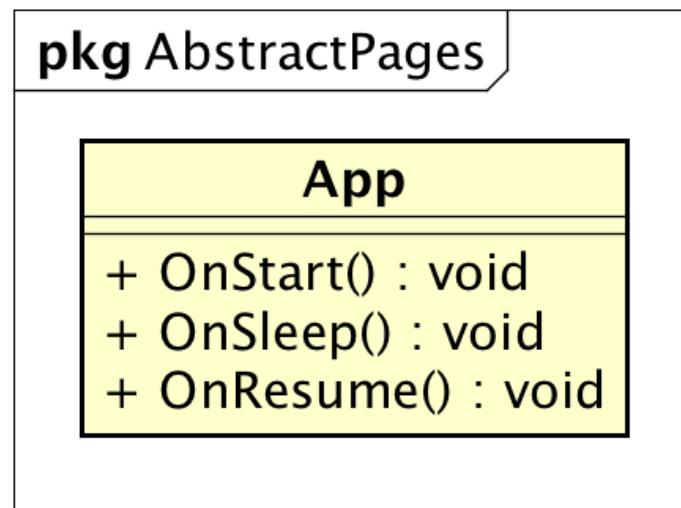


Abbildung 4.3: App Klasse

Beschreibung: Die Klasse **App** koordiniert den Aufbau der Tabbar und die zu den entsprechenden Tabs gehörenden Ansichten und Tab-Icons.

4.1.2 OverviewTabPage

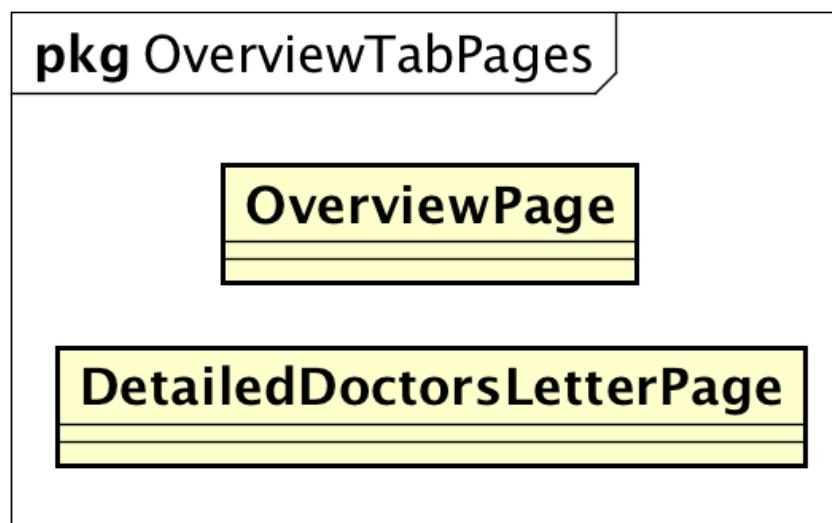


Abbildung 4.4: Klassen des OverviewTabPage Paket

OverviewPage : CustomContentPage

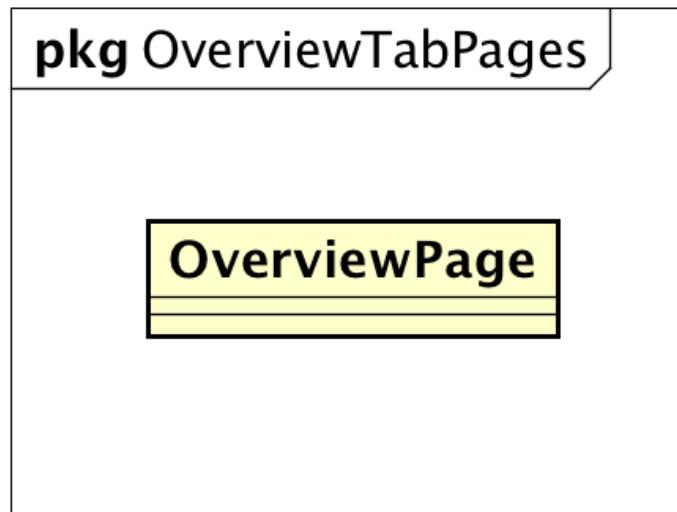


Abbildung 4.5: OverviewPage Klasse

Beschreibung: Die Klasse **OverviewPage** definiert die Eigenschaften und Darstellung der Startseite des Tabs *Übersicht*.

DetailedDoctorsLetterPage : CustomContentPage

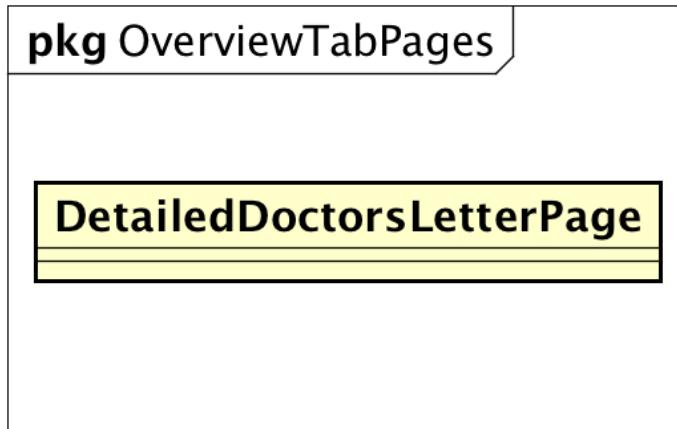


Abbildung 4.6: DetailedDoctorsLetterPage Klasse

Beschreibung: Die Klasse **DetailedDoctorsLetterPage** definiert die Eigenschaften und Darstellung der Detailansicht eines Arztbriefes.

4.1.3 MedicationTabPage

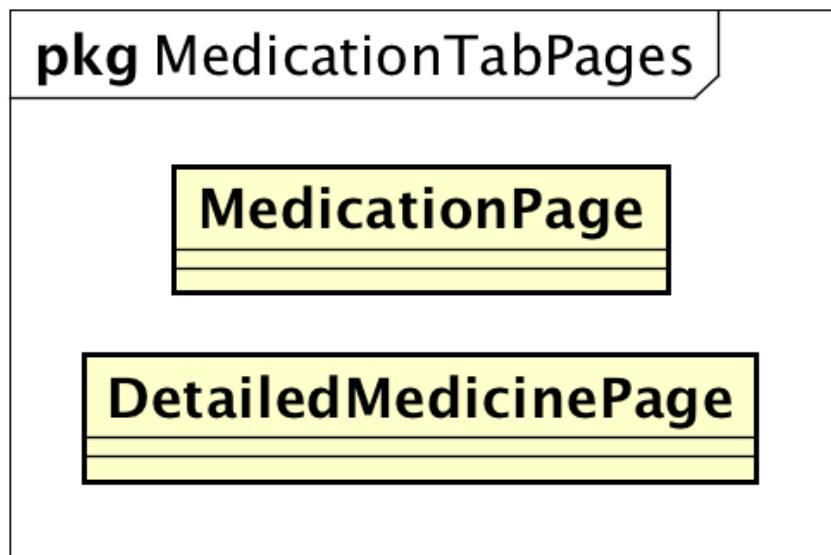


Abbildung 4.7: Klassen des MedicationTabPage Paket

MedicationPage : CustomContentPage

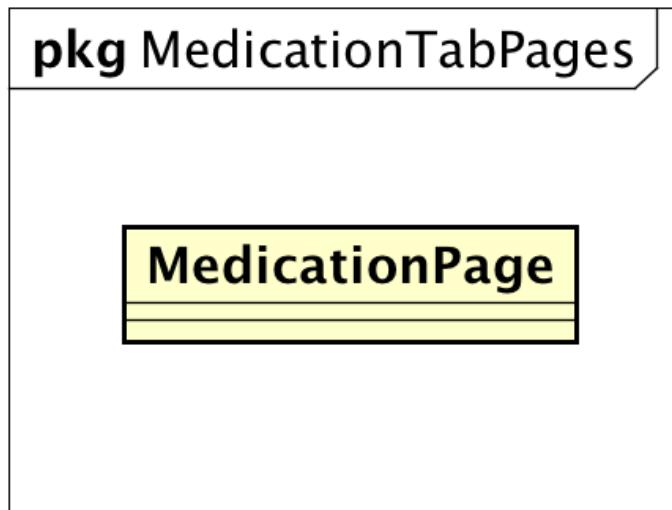


Abbildung 4.8: MedicationPage Klasse

Beschreibung: Die Klasse **MedicationPage** definiert die Eigenschaften und Darstellung der Startseite des Tabs *Medikation*.

DetailedMedicinePage : CustomContentPage

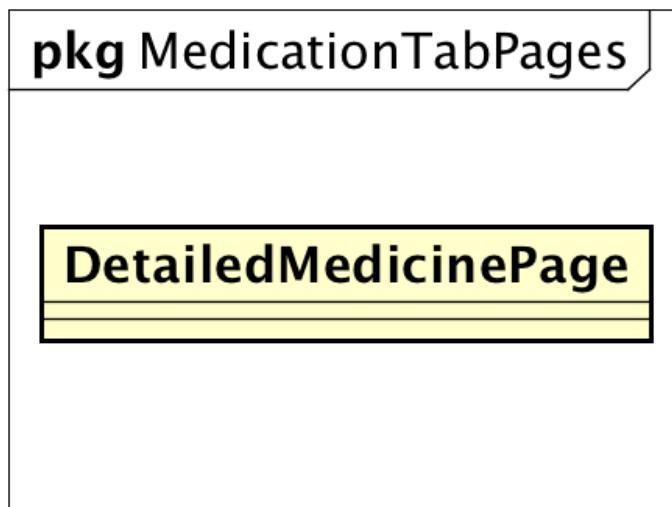


Abbildung 4.9: DetailedMedicinePage Klasse

Beschreibung: Die Klasse **DetailedMedicinePage** definiert die Eigenschaften und Darstellung der Detailansicht einer Medikation.

4.1.4 SendDataTabPages

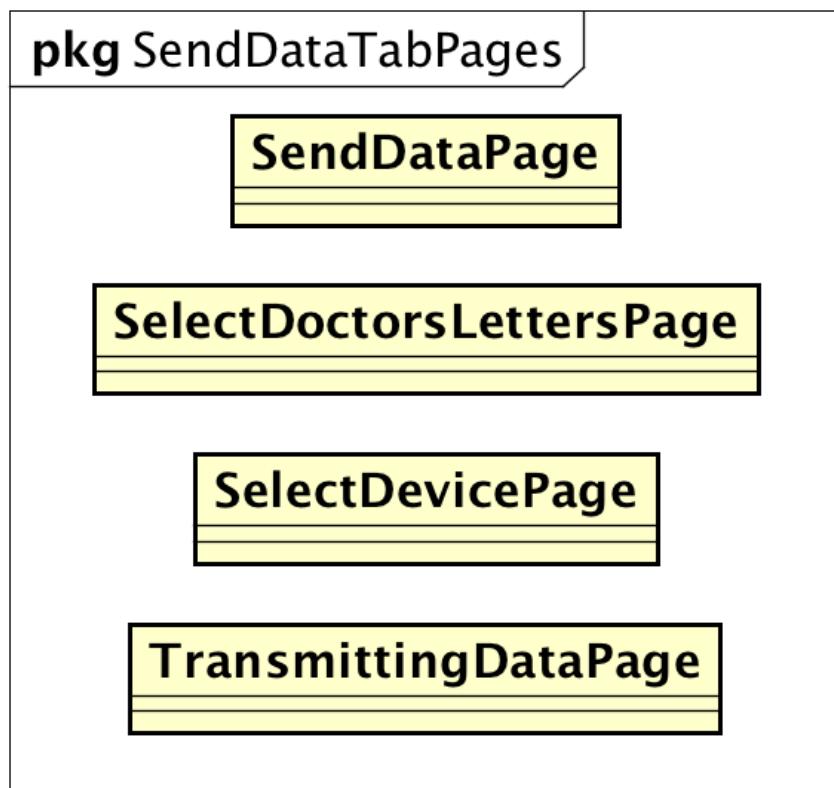


Abbildung 4.10: Klassen des SendDataTabPages Paket

SendDataPage : CustomContentPage

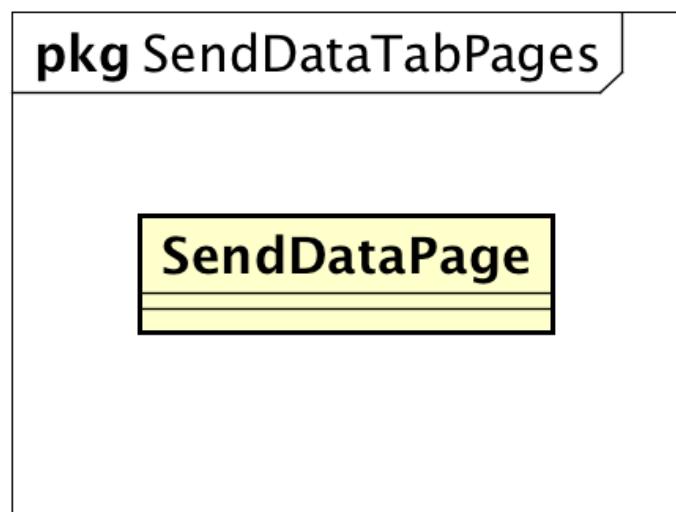


Abbildung 4.11: SendDataPage Klasse

Beschreibung: Die Klasse **SendDataPage** definiert die Eigenschaften und Darstellung der Startseite des Tabs *Senden*.

SelectDoctorsLettersPage : CustomContentPage

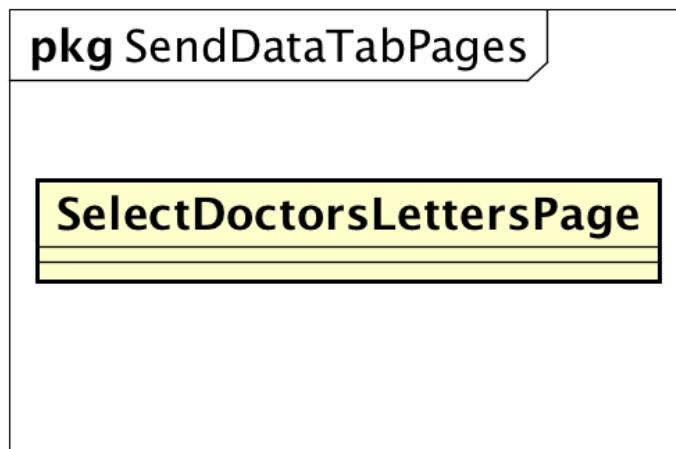


Abbildung 4.12: SelectDoctorsLettersPage Klasse

Beschreibung: Die Klasse **SelectDoctorsLettersPage** definiert die Eigenschaften und Darstellung der Ansicht zur Auswahl der zu übertragenden Menge an Arztbriefen.

SelectDevicePage : CustomContentPage

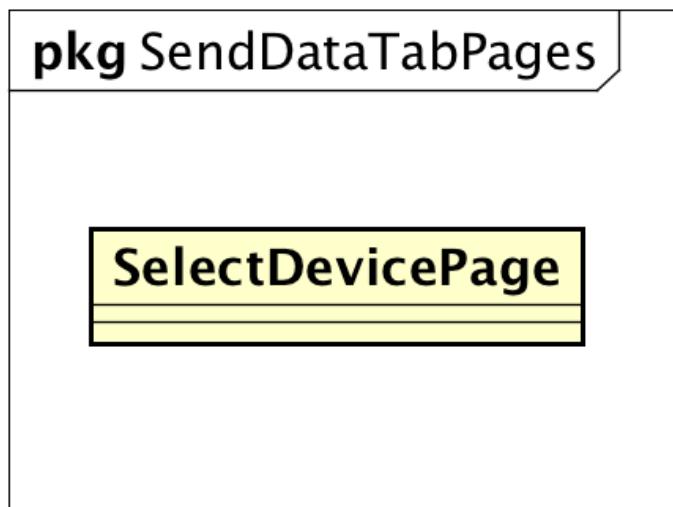


Abbildung 4.13: SelectDevicePage Klasse

Beschreibung: Die Klasse **SelectDevicePage** definiert die Eigenschaften und Darstellung der Ansicht zur Auswahl des Empfängergerätes der zuvor ausgewählten Menge an Arztbriefen.

TransmittingDataPage : CustomContentPage

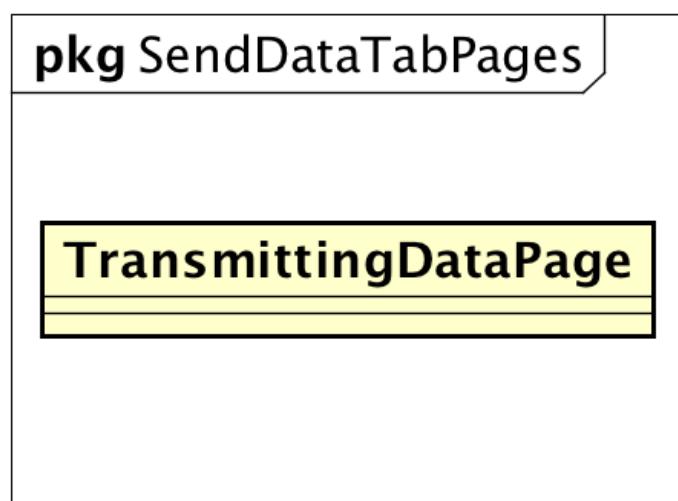


Abbildung 4.14: TransmittingDataPage Klasse

Beschreibung: Die Klasse **TransmittingDataPage** definiert die Eigenschaften und Darstellung der Ansicht eines aktiven Sendevorganges. Sie beschreibt die dritte und letzte Ansicht eines selbstinitiierten Sendevorganges.

4.1.5 ProfileTabPage

ProfilePage : CustomContentPage

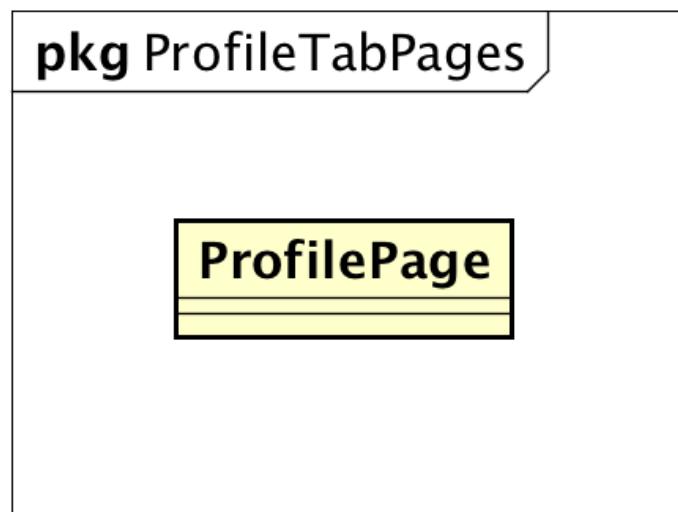


Abbildung 4.15: ProfilePage Klasse

Beschreibung: Die Klasse **ProfilePage** definiert die Eigenschaften und Darstellung der Startseite des Tabs *Profil*.

4.2 ViewModel

4.2.1 OverallViewModel

OverallViewModel

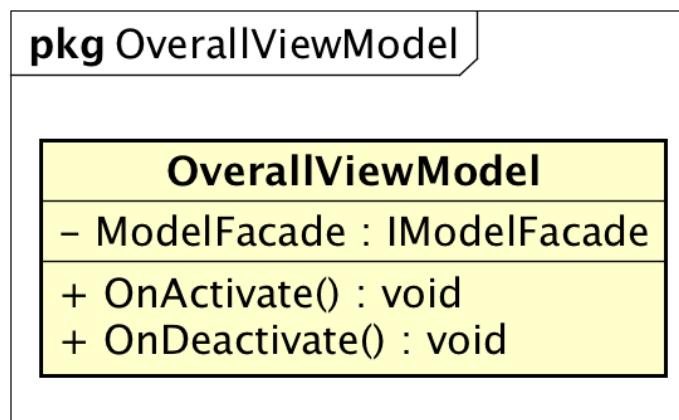


Abbildung 4.16: OverallViewModel Klasse

Beschreibung: Die Klasse **OverallViewModel** kapselt Methoden und Funktionen, die in (beinahe) allen ViewModel-Klassen benötigt werden.

Attribute:

- *ModelFacade : IModelFacade* - Ermöglicht den Klassen des ViewModels die Kommunikation mit dem Model über die *ModelFacade*

4.2.2 OverviewTabViewModel

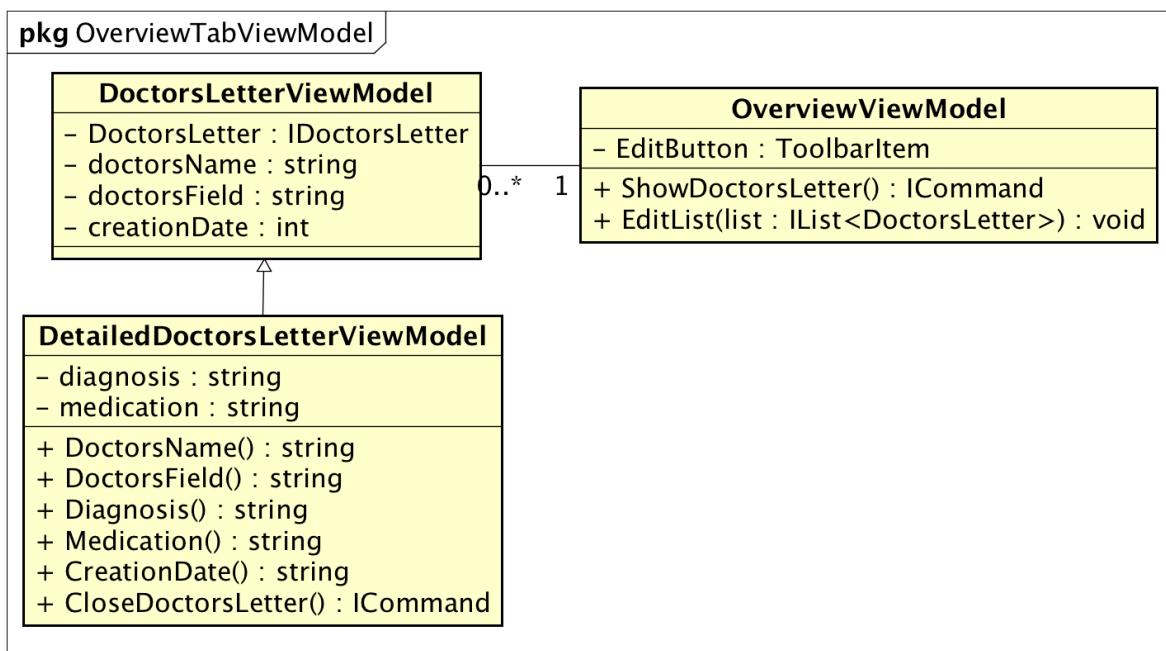


Abbildung 4.17: Klassen des OverviewTabViewModel Paket

OverviewViewModel : OverallViewModel

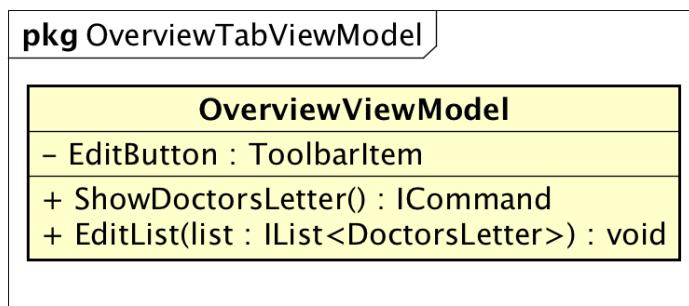


Abbildung 4.18: OverviewViewModel Klasse

Beschreibung: Die Klasse **OverviewViewModel** verwaltet die Liste der anzuzeigenden Arztbriefe der View *OverviewPage* und die Funktionsweise der dort angelegten UI-Elemente.

Attribute:

- *DoctorsLettersList : IList<DoctorsLetterViewModel>* - Die Menge der anzuzeigenden Arztbriefe
- *EditButton : ToolbarItem* - Knopf in der Toolbar, um die Liste der Arztbriefe editieren zu können

Methoden:

- *ShowDoctorsLetter() : ICommand* - Die Methode des EditButton (ICommand), der unsichtbar auf jedem Listeneintrag liegt. Ein Click auf diesen Button öffnet die Detailansicht des angetippten Arztbriefes.
- *EditList(list : IList<IDoctorsLetter>) : void* - Methode, um eine Auswahl an Arztbriefen zu löschen.

Paket: OverviewTabViewModel

DoctorsLetterViewModel : OverallViewModel

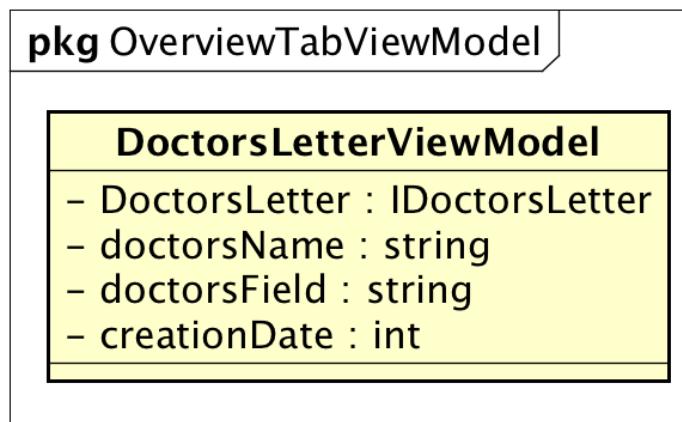


Abbildung 4.19: DoctorsLetterViewModel Klasse

Beschreibung: Die Klasse **DoctorsLetterViewModel** ist für die Zusammensetzung eines Arztbrieflisteneintrages zuständig.

Attribute:

- *DoctorsLetter : DoctorsLetter* - Ein anzuzeigender Arztbrief
- *doctorsName : string* - Der Name des behandelnden Arztes
- *doctorsField : string* - Das Fachgebiet des behandelnden Arztes

- *creationDate : DateTime* - Das Erstellungsdatum des Arztbriefes

Paket: OverviewTabViewModel

DetailedDoctorsLetterViewModel : DoctorsLetterViewModel

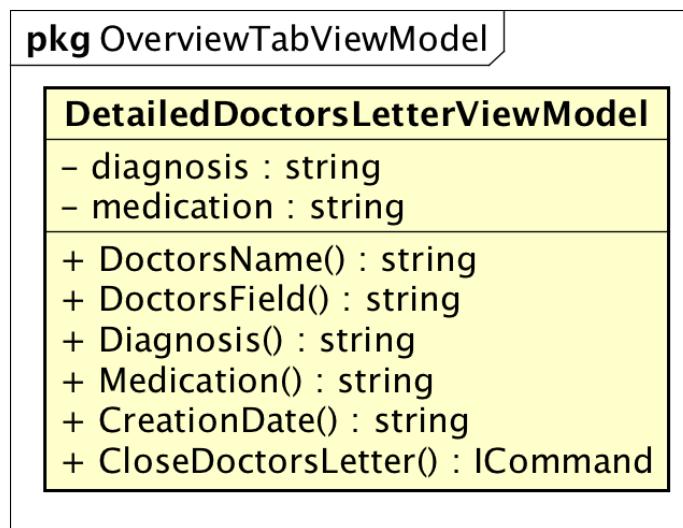


Abbildung 4.20: DetailedDoctorsLetterViewModel Klasse

Beschreibung: Die Klasse **DetailedDoctorsLetterViewModel** kümmert sich um die Zusammensetzung eines ausführlichen Arztbriefes, dargestellt in der *DetailedDoctorsLetter-Page*. Um Redundanz zu vermeiden, erbt diese Klasse von *DoctorsLetterViewModel*.

Attribute:

- *diagnosis : string* - Die vom behandelnden Arzt ausgestellte Diagnose
- *medication : string* - Die vom behandelnden Arzt empfohlene Therapie

Methoden:

- *DoctorsName() : string* - Methode um den Namen des Arztes als String aus einem DoctorsLetter zu erhalten
- *DoctorsField() : string* - Methode um den Fachbereich des Arztes als String aus einem DoctorsLetter zu erhalten
- *Diagnosis() : string* - Methode um die Diagnose als String aus einem DoctorsLetter zu

erhalten

- *Medication() : string* - Methode um die verordnete Medikation als String aus einem DoctorsLetter zu erhalten
- *CreationDate() : string* - Methode das Erstellungsdatum als String aus einem DoctorsLetter zu erhalten
- *CloseDoctorsLetter() : ICommand* - Button, um zurück in die OverviewTabPage zu wechseln

Paket: OverviewTabViewModel

4.2.3 MedicationTabViewModel

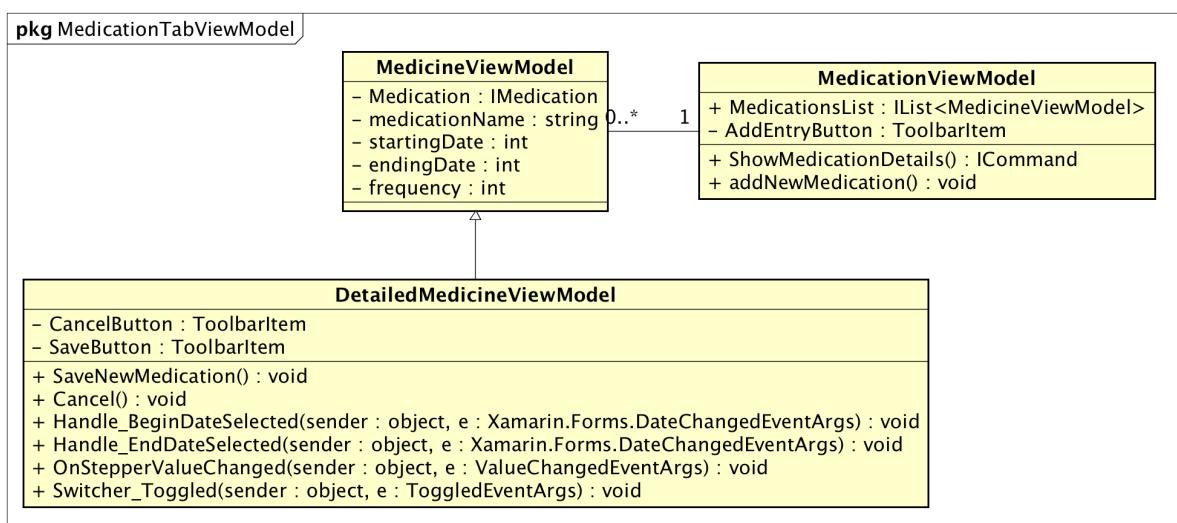


Abbildung 4.21: Klassen des MedicationTabViewModel Paket

MedicationViewModel : OverallViewModel

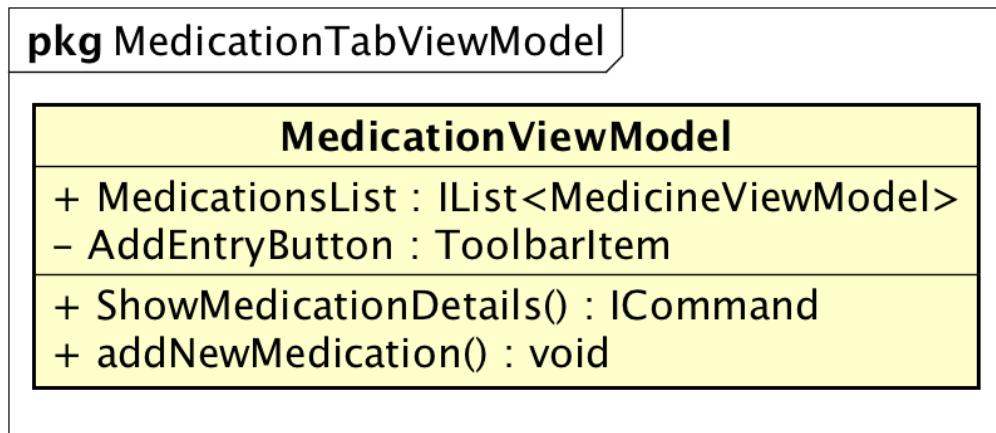


Abbildung 4.22: MedicationViewModel Klasse

Beschreibung: Die Klasse **MedicationViewModel** verwaltet die Liste der anzuzeigenden Medikationen der View *MedicationPage* und die Funktionsweise der dort angelegten UI-Elemente.

Attribute:

- *MedicationsList : IList<MedicineViewModel>* - Die Menge der anzuzeigenden Medikationen
- *AddEntryButton : ToolbarItem* - Knopf in der Toolbar, um neue Medikationen anlegen zu können

Methoden:

- *ShowMedicationDetail() : ICommand* - Methode, um die Detailansicht einer Medikation zu öffnen.
- *EditList() : void* - Methode um eine neue, mit default-Werten gefüllte, Medikation anzulegen.

Paket: MedicationTabViewModel

MedicineViewModel : OverallViewModel

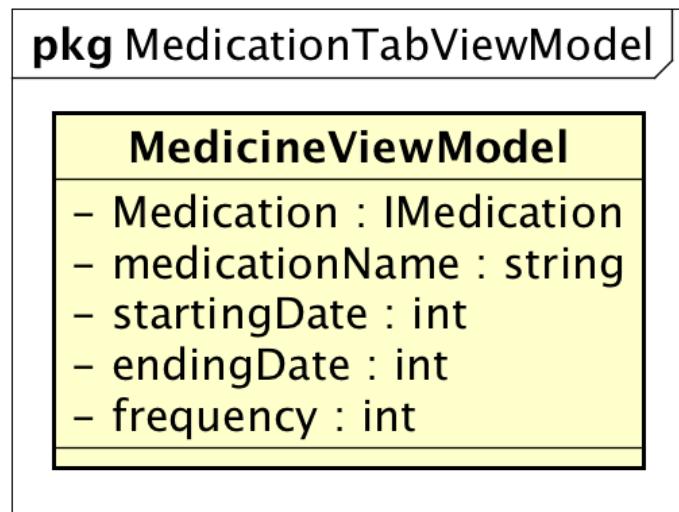


Abbildung 4.23: MedicineViewModel Klasse

Beschreibung: Die Klasse **MedicineViewModel** ist für die Zusammensetzung eines Medikationlisteneintrages zuständig.

Attribute:

- *Medication : Medication* - Eine anzuzeigende Medikation
- *medicationName : string* - Die Bezeichnung des eingenommenen Medikaments oder Wirkstoffes
- *startingDate : DateTime* - Das Datum des Einnahmebeginns
- *endingDate : DateTime* - Das Datum des Einnahmeendes
- *frequency : int* - Die Anzahl der täglichen Einnahmen

Paket: MedicationTabViewModel

DetailedMedicineViewModel : MedicineViewModel

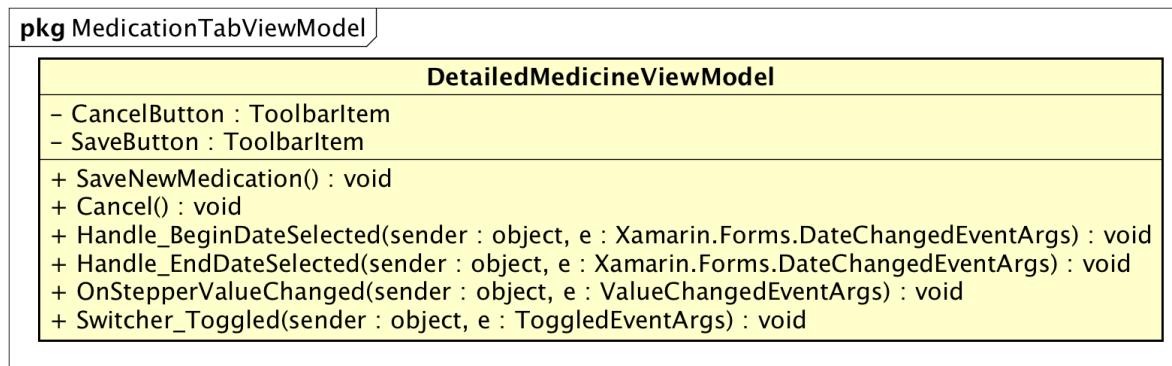


Abbildung 4.24: DetailedMedicineViewModel Klasse

Beschreibung: Die Klasse **DetailedMedicineViewModel** implementiert, wie bereits bestehende Medikationen in ihren Parametern angepasst oder neue Medikationen angelegt werden können. Um Redundanz zu vermeiden, erbt diese Klasse von *MedicineViewModel*.

Attribute:

- *CancelButton : ToolbarItem* - Knopf in der Toolbar, um das Anlegen einer neuen Medikation abzubrechen
- *SaveButton : ToolbarItem* - Knopf in der Toolbar, um die neue/geänderte Medikation abzuspeichern

Methoden:

- *SaveNewMedication() : void* - Methode um die gerade geöffnete Medikation zu speichern.
- *Cancel() : void* - Methode um ohne zu Speichern auf die MedicationTabPage zurück zu kehren.
- *Handle_BeginDateSelected(sender : object, e : Xamarin.Forms.DateChangedEventArgs) : void* - Methode um ein ausgewähltes Startdatum zu verarbeiten.
- *Handle_EndDateSelected(sender : object, e : Xamarin.Forms.DateChangedEventArgs) : void* - Methode um ein ausgewähltes Enddatum zu verarbeiten.
- *OnStepperValueChanged(sender : object, e : ValueChangedEventArgs) : void* - Methode um eine Änderung am Wert des Steppers zu verarbeiten.

- *Switcher_Toggled(sender : object, e : ToggledEventArgs) : void* - Methode um eine Änderung am Wert des Switches zu verarbeiten.

Paket: MedicationTabViewModel

4.2.4 SendDataTabViewModel

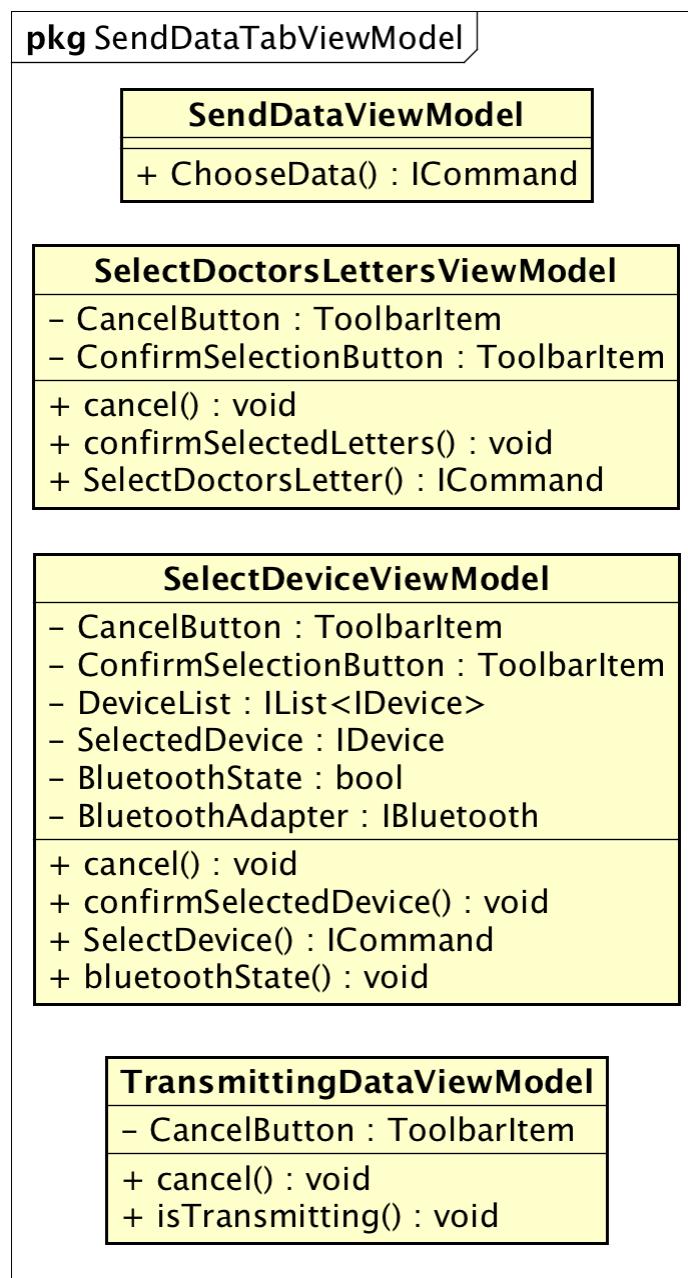


Abbildung 4.25: Klassen des SendDataTabViewModel Paket

SendDataViewModel : OverallViewModel

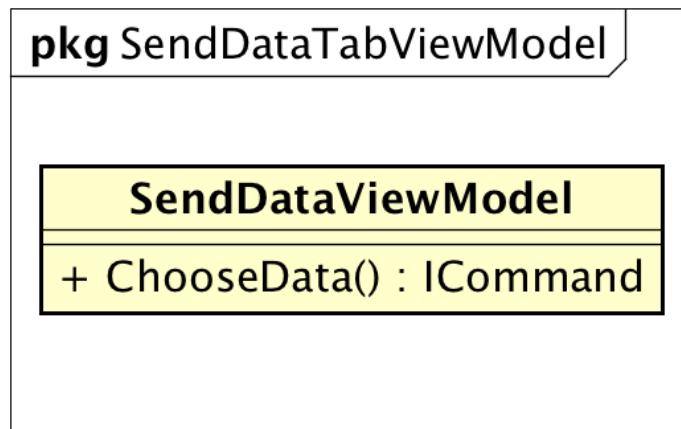


Abbildung 4.26: SendDataViewModel Klasse

Beschreibung: Die Klasse **SendDataViewModel** verwaltet die Funktionsweise der in *Send-DataPage* angelegten UI-Elemente.

Methoden:

- *ChooseData() : ICommand* - Methode um die gerade geöffnete Medikation zu speichern.

Paket: SendDataTabViewModel

SelectDoctorsLettersViewModel : OverallViewModel

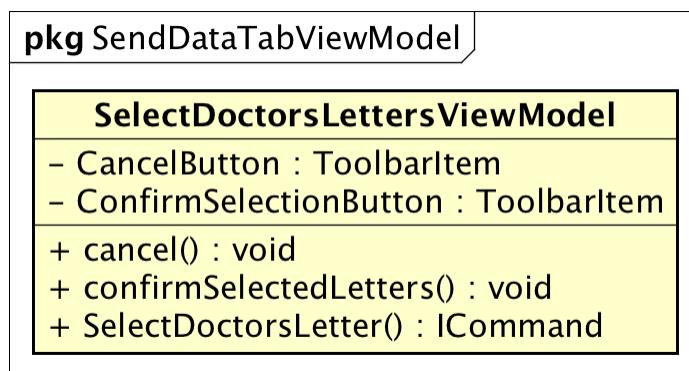


Abbildung 4.27: SelectDoctorsLettersViewModel Klasse

Beschreibung: Die Klasse **SelectDoctorsLettersViewModel** koordiniert die in *SelectDoctorsLettersPage* darzustellenden Arztbriefe sowie die Menge der vom Nutzer ausgewählten Arztbriefe.

Attribute:

- *CancelButton : ToolbarItem* - Knopf in der Toolbar, um den Vorgang abzubrechen
- *ConfirmSelectionButton : ToolbarItem* - Knopf in der Toolbar, um mit der ausgewählten Menge an Arztbriefen fortzufahren
- *DoctorsLettersList : IList<DoctorsLetterViewModel>* - Die Menge der anzuzeigenden Arztbriefe
- *SelectedDoctorsLettersList : IList<DoctorsLetterViewModel>* - Die Menge der vom Nutzer ausgewählten Arztbriefe

Methoden:

- *cancel() : void* - Methode um die Auswahl der Arztbriefe abzubrechen.
- *confirmSelectedLetters() : void* - Methode um weiter zur Auswahl des Empfangsgerätes zu navigieren.
- *SelectDoctorsLetter() : ICommand* - Methode um einen Arztbrief auszuwählen

Paket: SendDataTabViewModel

SelectDeviceViewModel : OverallViewModel

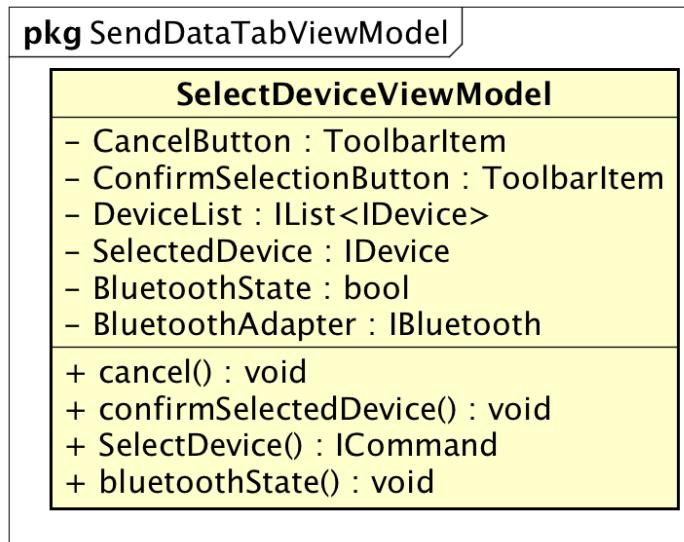


Abbildung 4.28: SelectDeviceViewModel Klasse

Beschreibung: Die Klasse **SelectDeviceViewModel** koordiniert die in *SelectDevicePage* darzustellenden Geräte in Reichweite sowie die Auswahl des Empfängers durch den Nutzer

Attribute:

- *CancelButton : ToolbarItem* - Knopf in der Toolbar, um den Vorgang abzubrechen
- *ConfirmSelectionButton : ToolbarItem* - Knopf in der Toolbar, um mit dem ausgewählten Empfänger fortzufahren
- *DeviceList : IList<Device>* - Die Menge der Empfangsgeräte in Reichweite
- *SelectedDevice : IDevice* - Der vom Nutzer gewählte Empfänger
- *BluetoothState : bool* - Zustand des Bluetooth Moduls (Bluetooth an/aus)
- *BluetoothAdapter : Bluetooth* - Kommunikation mit der Bluetooth-Schnittstelle

Methoden:

- *cancel() : void* - Methode um die Auswahl des Empfangsgerätes abzubrechen.
- *confirmSelectedDevice() : void* - Methode um zum Senden weiter zu navigieren.
- *SelectDevice() : ICommand* - Methode um ein Empfangsgerät auszuwählen.
- *bluetoothState() : void* - Methode um den Status der eigenen Bluetoothverfügbarkeit zu erhalten.

Paket: SendDataTabViewModel

TransmittingDataViewModel : OverallViewModel

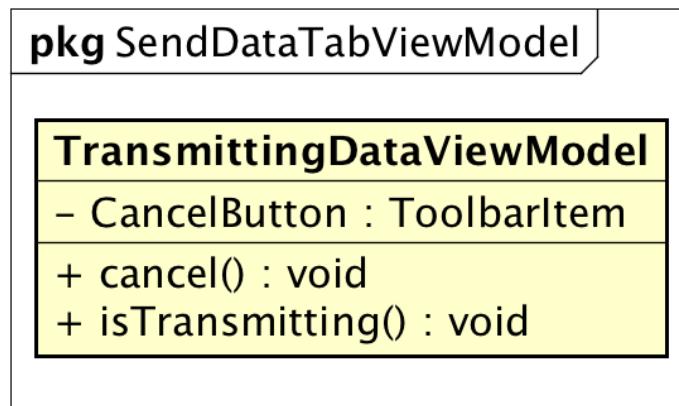


Abbildung 4.29: TransmittingDataViewModel Klasse

Beschreibung: Die Klasse **TransmittingDataViewModel** verteilt die vom Nutzer in den Schritten zuvor ausgewählten Daten an die verschiedenen zuständigen Modelklassen. Neben den zu übertragenden Arztbriefen umfasst dies auch den zu kontaktierenden Empfänger.

Attribute:

- *CancelButton : ToolbarItem* - Knopf in der Toolbar, um den Vorgang abzubrechen

Methoden:

- *cancel() : void* - Methode um den Sendevorgang abzubrechen.
- *isTransmitting() : void* - Methode um den Status des Sendevorgangs zu erhalten.

Paket: SendDataTabViewModel

4.2.5 ProfileTabViewModel

ProfileViewModel : OverallViewModel

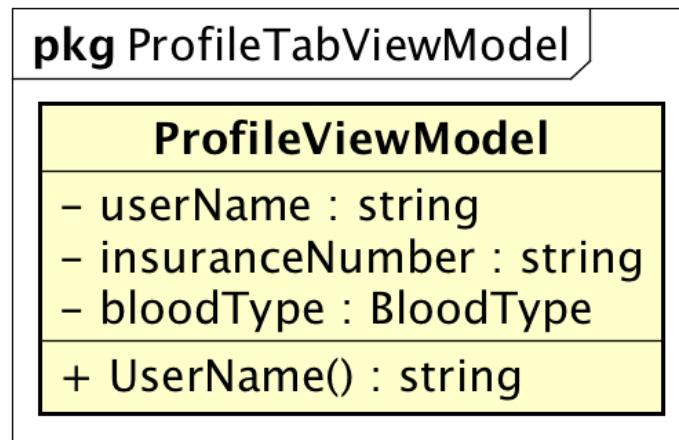


Abbildung 4.30: ProfileViewModel Klasse

Beschreibung: Die Klasse **ProfileViewModel** verwaltet das in *ProfilePage* anzuseigende Profil und die Funktionsweise der dort angelegten UI-Elemente.

Attribute:

- `userName : string` - Knopf in der Toolbar, um den Vorgang abzubrechen
- `insuranceNumber : string` - Knopf in der Toolbar, um den Vorgang abzubrechen
- `bloodType : BloodType` - Knopf in der Toolbar, um den Vorgang abzubrechen

Methoden:

- `UserName() : string` - Methode um den Username des Nutzers als String zu erhalten.

Paket: ProfileTabViewModel

4.3 Model

*Hinweis: Die Schnittstellen des Models (**ModellInterface**) wurden bereits im Abschnitt **Subsystem-Schnittstellen** beschrieben und werden hier nicht erneut aufgeführt.*

4.3.1 ModelFacade

ModelFacade : IModelFacade

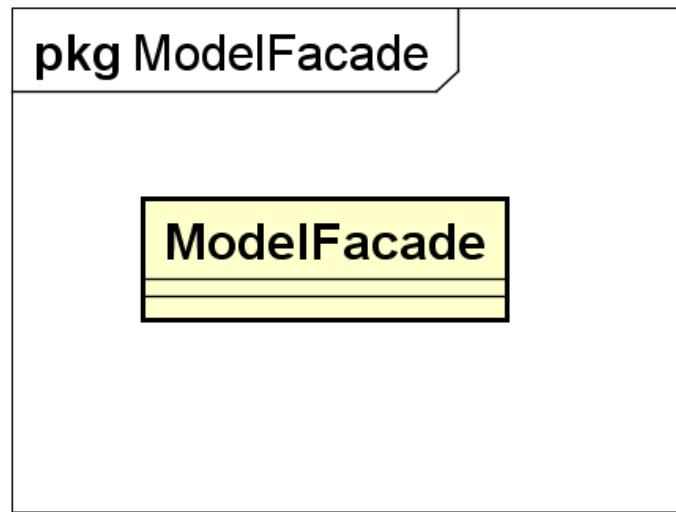


Abbildung 4.31: ModelFacade Klasse

Beschreibung: Die **ModelFacade** implementiert die Fassaden-Schnittstelle **IModelFacade**. Sie delegiert bei einem Aufruf an eine oder mehrere Schnittstellen der restlichen Pakete des Subsystems.

Attribute:

- *database : IEntityDatabase* - Die Datenbank, in der Entitäten gespeichert werden
- *factory : IEntityFactory* - Enthält die Logik um Entitäten zu erstellen
- *parser : IParserFacade* - Enthält die Logik um zwischen Datenformaten zu konvertieren
- *fileHelper : IFileHelper* - Helferklasse zum Dateizugriff
- *transmission : IBluetooth* - Enthält die Logik der Initialisierung einer Datenübertragung

Paket: ModelFacade

4.3.2 TransmissionModel

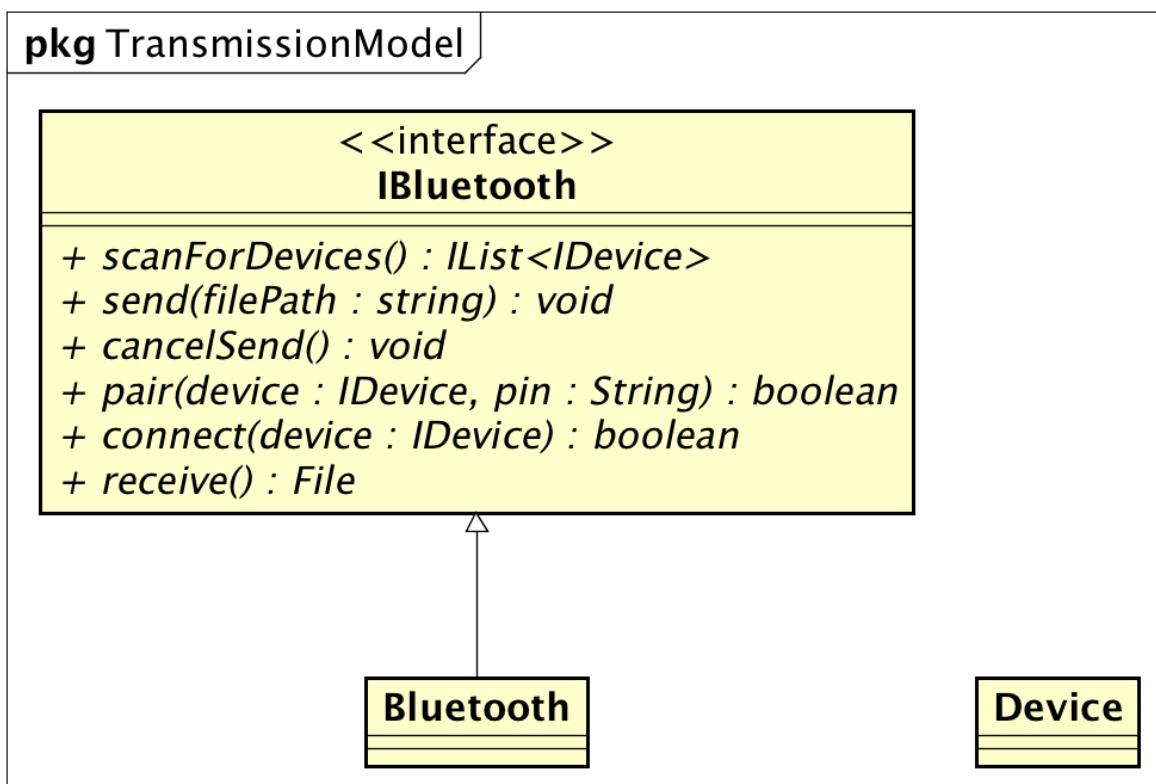


Abbildung 4.32: Klassen des TransmissionModel Paket

IBluetooth

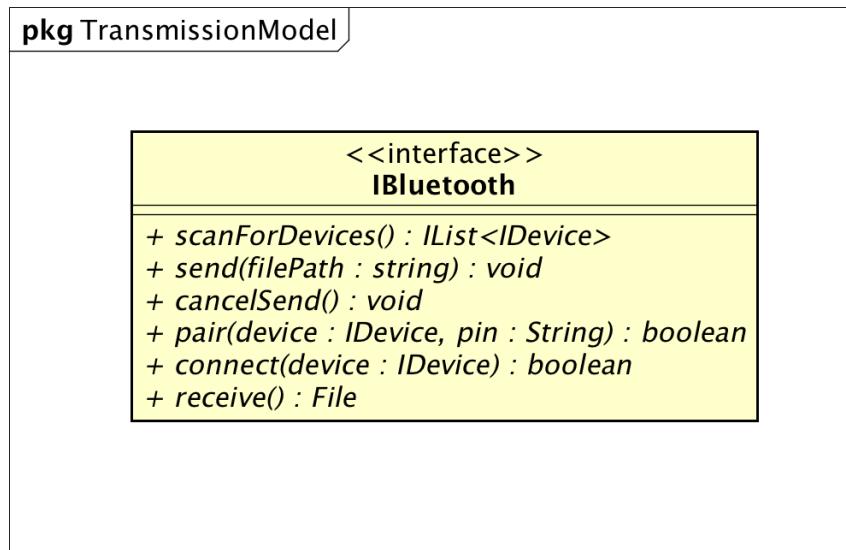


Abbildung 4.33: IBluetooth Schnittstelle

Beschreibung: Die Klasse **IBluetooth** stellt einen Adapter zum verwendeten Bluetooth-Framework dar.

Methoden: + `scanForDevices() : IList<IDevice>` - Methode, um Geräte in der Umgebung zu finden

+ `send(filePath : string) : void` - Methode, um eine spezifische Datei zu übertragen

+ `cancelSend() : void` - Methode, um eine aktive Übertragung abzubrechen

+ `pair(device : IDevice, pin : string) : boolean` - Methode, um mit einem spezifischen Gerät zu koppeln

+ `connect(device : IDevice) : boolean` - Methode, um sich mit einem spezifischen Gerät zu verbinden

+ `receive() : File` - Methode, um eine Datei zu empfangen

Bluetooth : IBluetooth

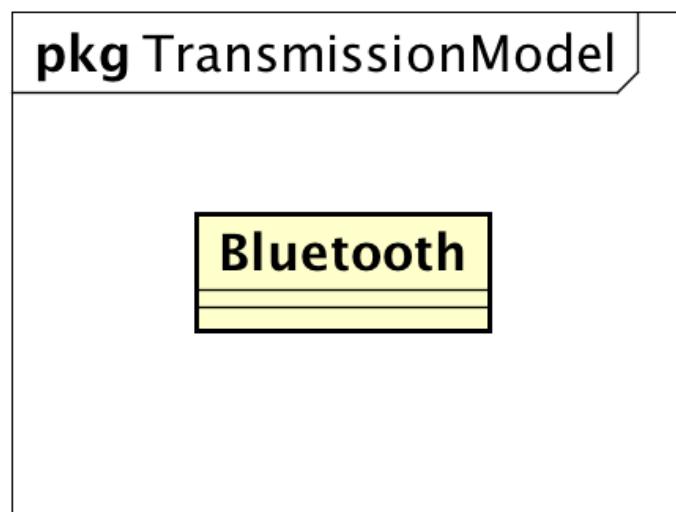


Abbildung 4.34: Bluetooth Klasse

Beschreibung: Implementierung der Schnittstelle *IBluetooth*

Device : IDevice

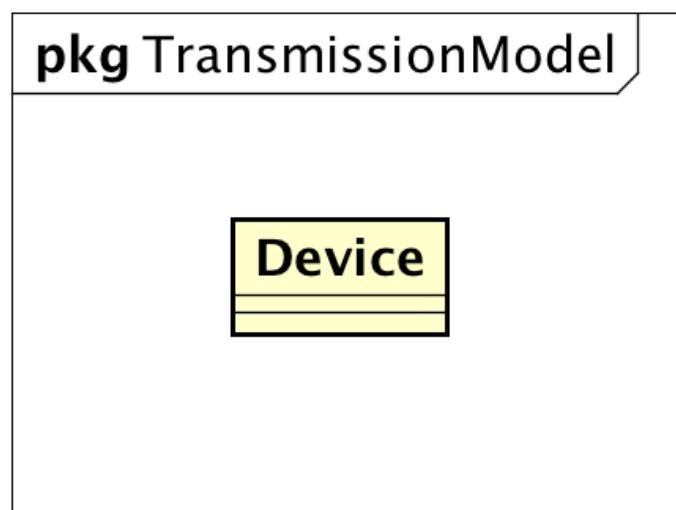


Abbildung 4.35: Device Klasse

Beschreibung: Implementierung der Schnittstelle *IDevice*

4.3.3 DatabaseModel

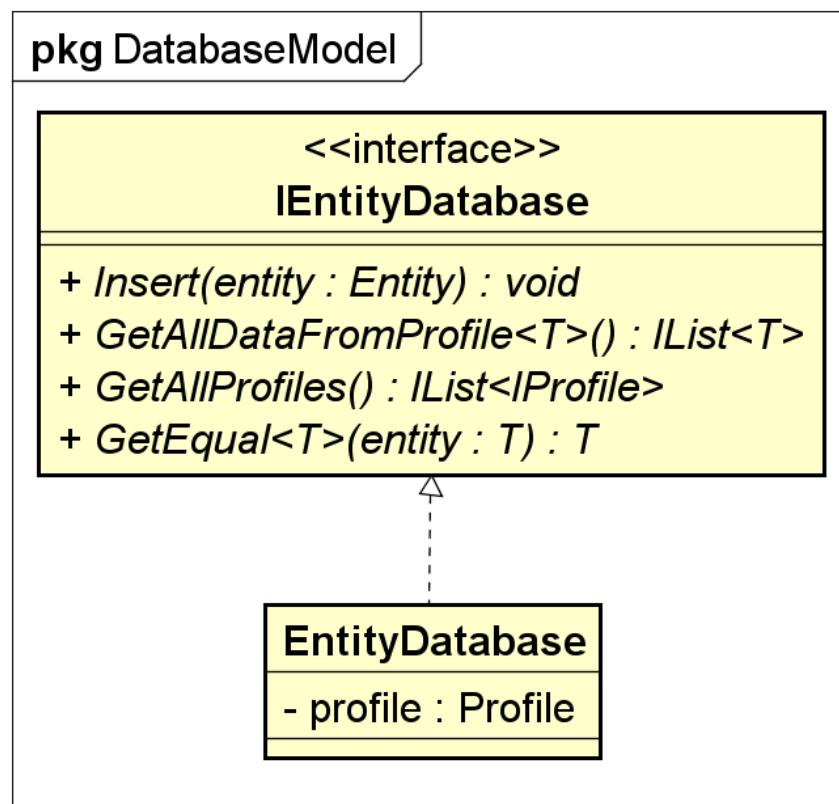


Abbildung 4.36: Klassen des DatabaseModel Paket

IEntityDatabase

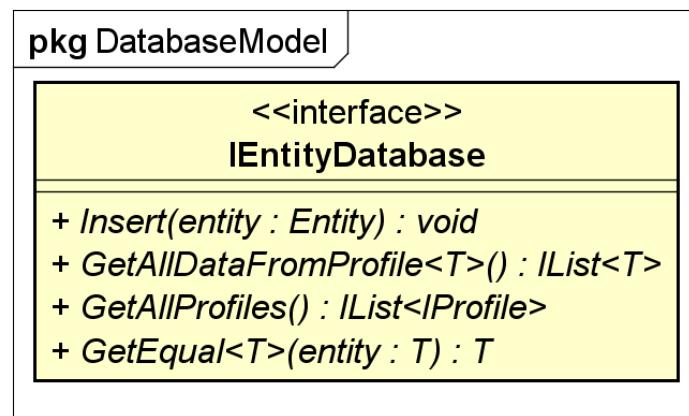


Abbildung 4.37: IEntityDatabase Schnittstelle

Beschreibung: Die Schnittstelle **IEntityDatabase** bietet eine Zugriffsmöglichkeit auf die verwendete Datenbank an.

Hier kann das aktuelle Profil gewechselt werden **SetActiveProfile** und sonstige Datenabfragen darauf durchgeführt werden.

Paket: ModelFacade

EntityDatabase : IEntityDatabase, IEntityObserver, IProfileObserver

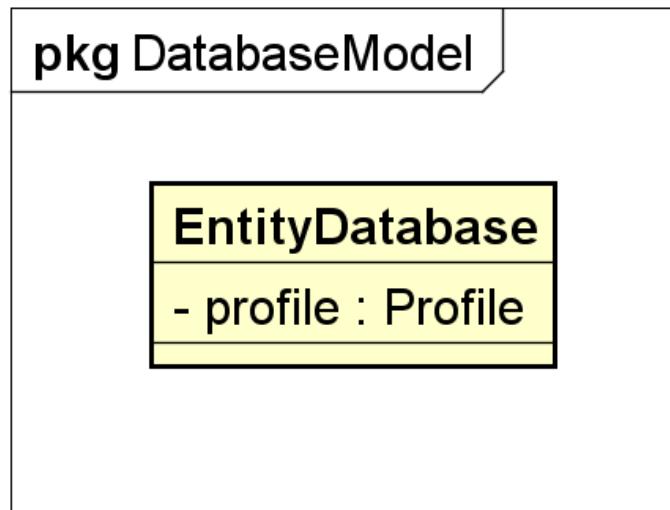


Abbildung 4.38: EntityDatabase Klasse

Beschreibung: Die Klasse **EntityDatabase** implementiert die **IEntityDatabase** Schnittstelle. Dabei werden Aufrufe von Methoden aus dieser Schnittstelle größtenteils an die hier verwendete SQLite-Library weitergeleitet.

Weiterhin wird die **IEntityObserver** Schnittstelle implementiert, sodass sich die Datenbank über Änderungen an den in ihr gespeicherten Entitäten informieren lassen kann.

Paket: ModelFacade

4.3.4 DataModel

Hinweis: Um Assoziationen zwischen Entitäten aus diesem Paket, also Entitäten, die in einer SQLite Datenbank gespeichert werden können, ordnungsgemäß in der Datenbank speichern zu können, reicht es nicht die Schnittstelle der Entitäten zu kennen. Stattdessen müssen diese konkreten Entitäten auch also solche angegeben werden. Dafür werden Methoden aus DataModelInterface die solche Assoziationen beeinflussen, zusätzlich zu ihrer Implementierung noch für die jeweiligen Parameter aus DataModel überladen.

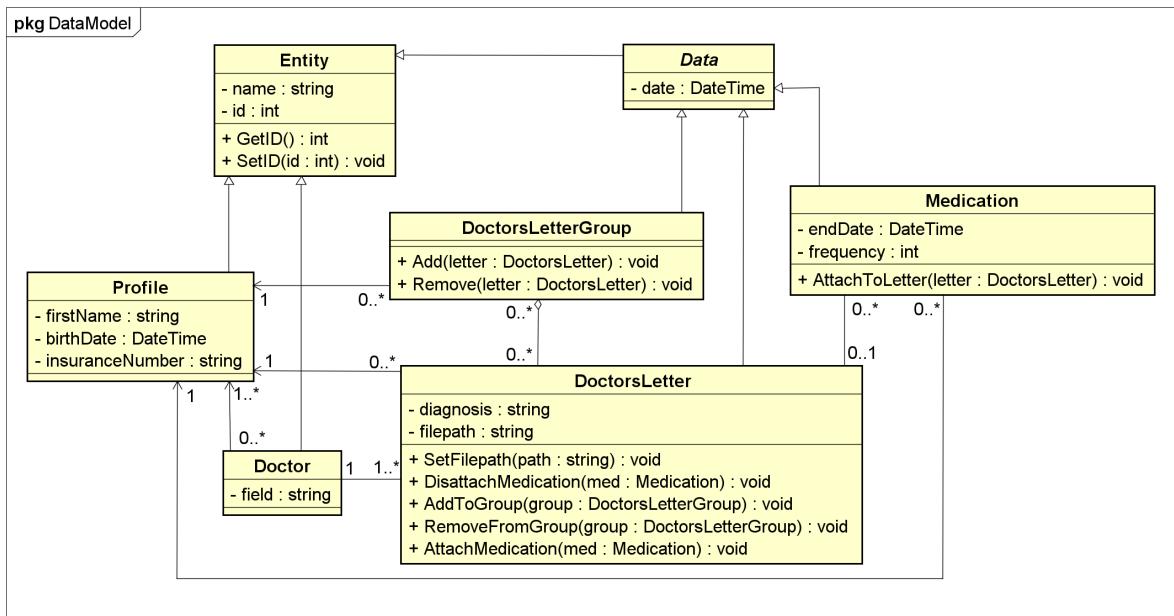


Abbildung 4.39: Klassen des DataModel Paket

Entity : IEntity, IEntityObservable

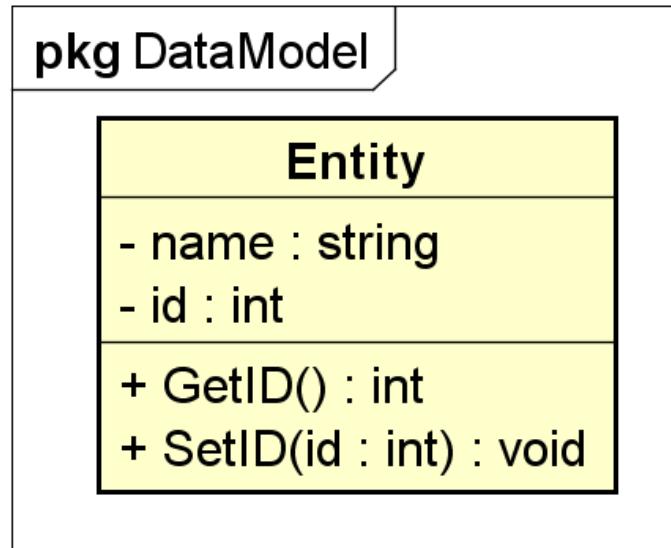


Abbildung 4.40: Entity Klasse

Beschreibung: Die abstrakte Klasse **Entity** implementiert die **IEntity** Schnittstelle als Vorlage für eine SQLite-Datenbanktabelle.

Weiterhin wird die **IEntityObservable** Schnittstelle implementiert, sodass eine Datenbank über Änderungen informiert werden kann.

Diese Klasse dient als Strukturelement in der Vererbungshierarchie im **DataModel** Paket, und kann deshalb nicht selbst instanziert werden, dies ist nur für ihre Unterklassen möglich. Bevor eine Entität aus ihrer Datenbank gelöscht wird, sollten all ihre Assoziationen zu anderen Entitäten aufgelöst werden (*Dispose*).

Attribute:

- *id : int* - Die ID über die Entitäten in einer Datenbank identifiziert werden können
- *name : string* - Der Name der Entität

Paket: DataModel

Profile : Entity, IProfile

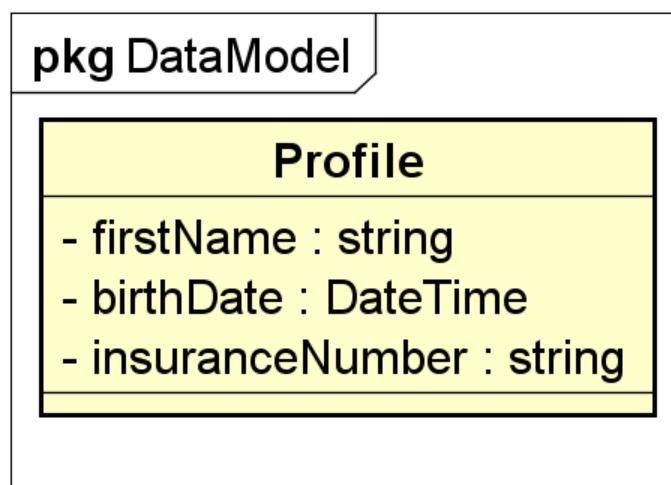


Abbildung 4.41: Profile Klasse

Beschreibung: Die Klasse **Profile** implementiert die **IProfile** Schnittstelle und erweitert die **Entity** Klasse um in einer Datenbank gespeichert werden zu können.

Ein Profil enthält dabei verschiedene Informationen über einen Nutzer, die in den Attributen aufgelistet werden.

Attribute:

- *birthDate : DateTime* - Das Geburtsdatum des Nutzers
- *bloodGroup : BloodGroup* - Die Blutgruppe des Nutzers
- *insuranceNumber : string* - Die Versicherungsnummer des Nutzers
- *firstName : string* - Der Vorname des Nutzers

Paket: DataModel

Doctor : Entity, IDoctor

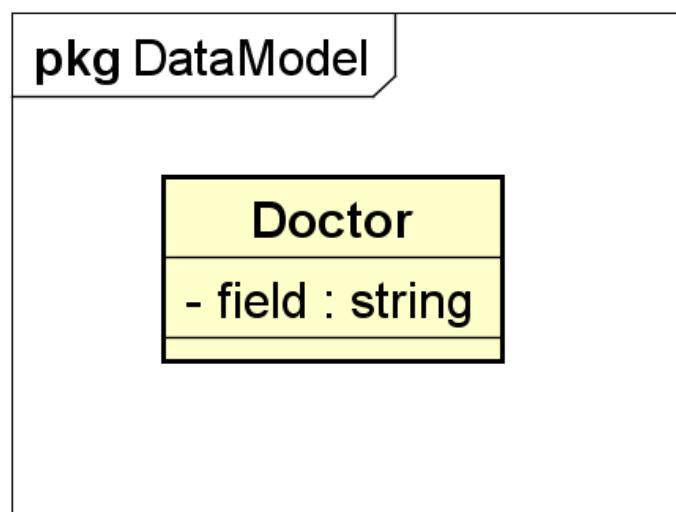


Abbildung 4.42: Doctor Klasse

Beschreibung: Die Klasse **Doctor** implementiert die **IDoctor** Schnittstelle und erweitert die **Entity** Klasse um in einer Datenbank gespeichert werden zu können.
Hier werden die Informationen über einen Arzt gekapselt.

Attribute:

- *profile : IProfile* - Das Profil, dem dieser Arzt zugeordnet ist
- *field : string* - Das Fachgebiet des Arztes

Paket: DataModel

Data : Entity, IData

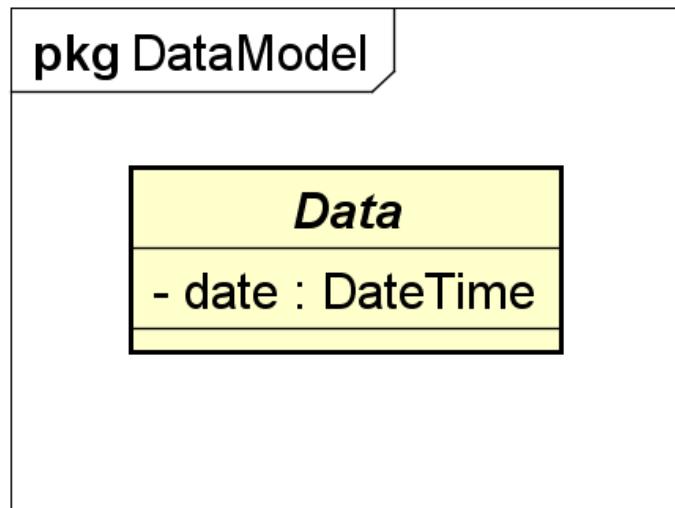


Abbildung 4.43: Data Klasse

Beschreibung: Die abstrakte Klasse **Data** implementiert die **IData** Schnittstelle und erweitert die **Entity** Klasse um in einer Datenbank gespeichert werden zu können.

Diese Klasse dient als Strukturelement in der Vererbungshierarchie im **DataModel** Paket, und kann deshalb nicht selbst instanziert werden, dies ist nur für ihre Unterklassen möglich.

Attribute:

- *date : DateTime* - Das Datum von dem diese Daten stammen
- *profile : IProfile* - Das Profil, dem diese Daten zugeordnet sind
- *sensitivity : Sensitivity* - Die Sensitivität der Daten

Paket: DataModel

DoctorsLetter : Data, IDoctorsLetter

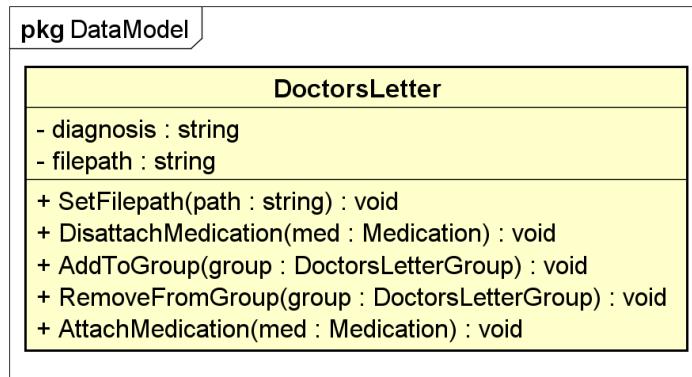


Abbildung 4.44: DoctorsLetter Klasse

Beschreibung: Die Klasse **DoctorsLetter** implementiert die **IDoctorsLetter** Schnittstelle und erweitert die **Data** Klasse.

Hier werden die Informationen über einen Arztbrief gekapselt.

Attribute:

- *diagnosis : string* - Die Diagnose, die in diesem Arztbrief enthalten ist
- *filepath : string* - Der Dateipfad in der der originale Arztbrief gespeichert ist
- *groups : IList<IDoctorsLetterGroup>* - Die Gruppen in denen dieser Arztbrief enthalten ist
- *meds : IList<IMedication>* - Die Medikationen, die in diesem Arztbrief verschrieben wurden
- *doctor : IDoctor* - Der Arzt der diesen Arztbrief erstellt hat

Paket: DataModel

DoctorsLetterGroup : Data, IDoctorsLetterGroup

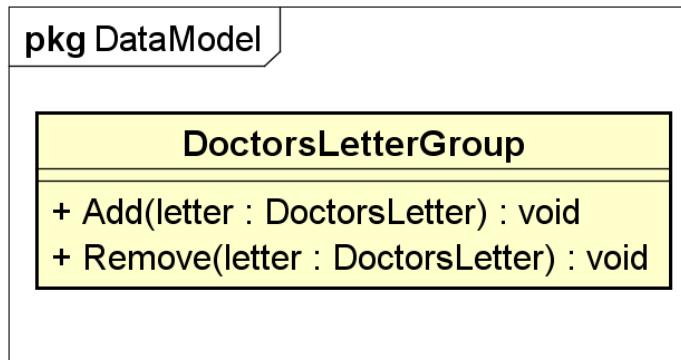


Abbildung 4.45: DoctorsLetterGroup Klasse

Beschreibung: Die Klasse **DoctorsLetterGroup** implementiert die **IDoctorsLetterGroup** Schnittstelle und erweitert die **Data** Klasse.
Hier wird eine Ansammlung von Arztbriefen gekapselt.

Attribute:

- *groups* : *IList<IDoctorsLetter>* - Die Arztbriefe die in dieser Gruppe enthalten sind

Paket:

DataModel

Medication : Data, IMedication

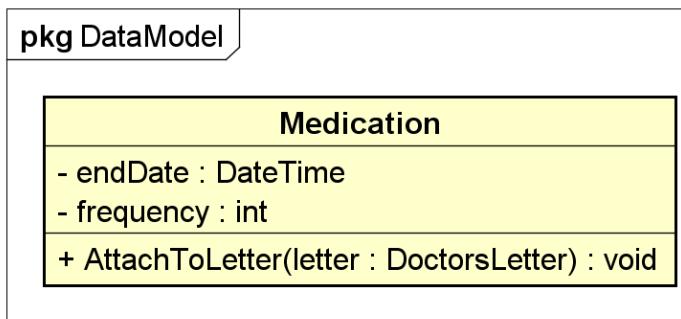


Abbildung 4.46: Medication Klasse

Beschreibung: Die Klasse **Medication** implementiert die **IMedication** Schnittstelle und erweitert die **Data** Klasse.

Hier werden alle Informationen über eine Medikation gekapselt.

Attribute:

- *letter* : *IDoctorsLetter* - Der Arztbrief in dem diese Medikation verschrieben wurde
- *frequency* : *int* - Gibt an wie oft die Medikation in ihrem *interval* eingenommen werden soll
- *interval* : *Interval* - Gibt an in welchem Intervall die Medikation eingenommen werden soll
- *endDate* : *DateTime* - Das Datum an dem die Medikation abgesetzt werden soll/sollte

Paket: DataModel

4.3.5 EntityFactory

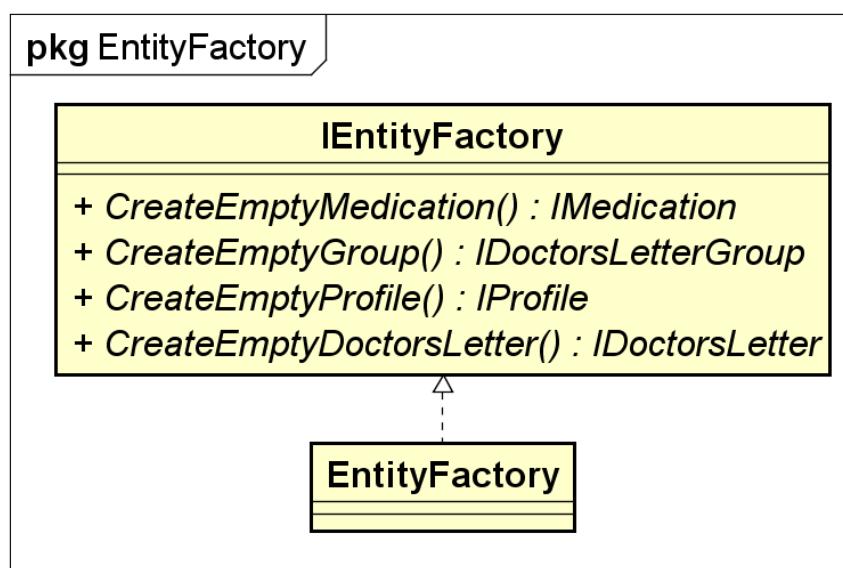


Abbildung 4.47: Klassen des EntityFactory Paket

IEntityFactory

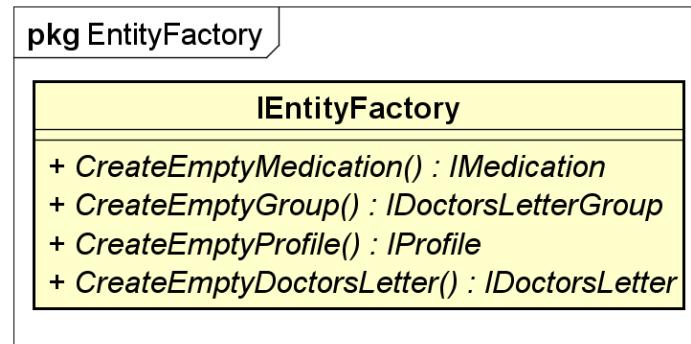


Abbildung 4.48: IEntityFactory Schnittstelle

Beschreibung: Schnittstelle zum Erstellen von Entitäten, also Arztbriefen (*CreateEmptyDoctorsLetter*), Gruppen von Arztbriefen (*CreateEmptyGroup*), Medikationen (*CreateEmptyMedication*) und Profilen (*CreateEmptyProfile*).

Paket: EntityFactory

EntityFactory : IEntityFactory

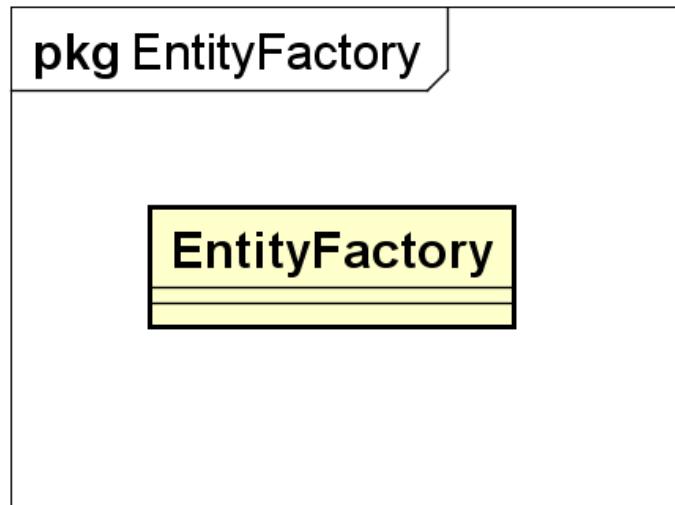


Abbildung 4.49: EntityFactory Klasse

Beschreibung: Implementierung der **IEntityFactory** Schnittstelle. Fügt Entitäten bei deren Erstellung in die Datenbank *database* ein und meldet sie als Beobachter an.

Attribute:

- *database* : *IEntityDatabase* - Die Datenbank in der die Entitäten gespeichert werden

Paket: EntityFactory

4.3.6 EntityObserver

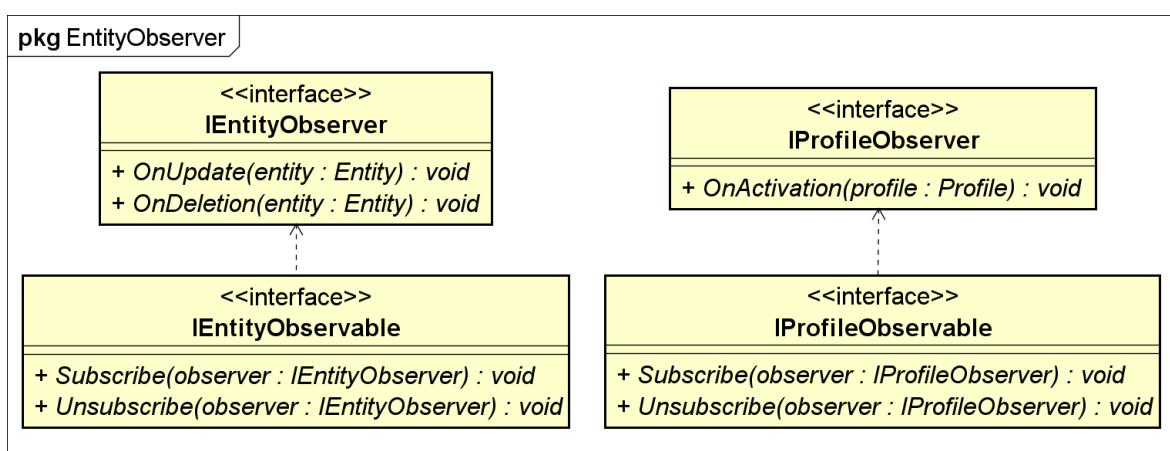


Abbildung 4.50: Klassen des EntityObserver Paket

IEntityObservable

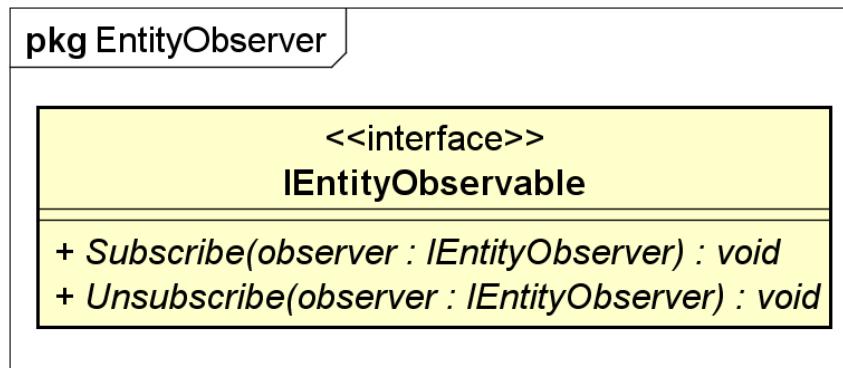


Abbildung 4.51: EntityObservable Schnittstelle

Beschreibung: Beobachter können sich über diese Schnittstelle bei Datenstrukturen anmelden (*Subscribe*) und wieder abmelden (*Unsubscribe*), um sich über Änderungen an diesen informieren zu lassen.

Paket: EntityObserver

IEntityObserver

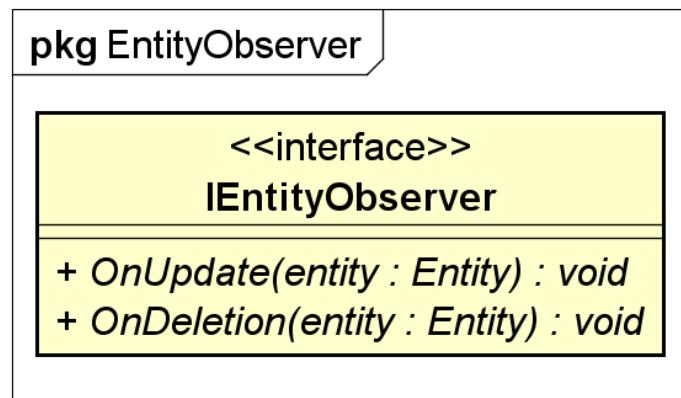


Abbildung 4.52: IEntityObserver Schnittstelle

Beschreibung: Schnittstelle um IEntityObservable-Objekte beobachten zu können.

Diese können dann, wenn sie geändert (*OnUpdate*) oder gelöscht (*OnDeletion*) werden, die angemeldeten Beobachter über diese Schnittstelle benachrichtigen.

Paket: EntityObserver

IProfileObservable

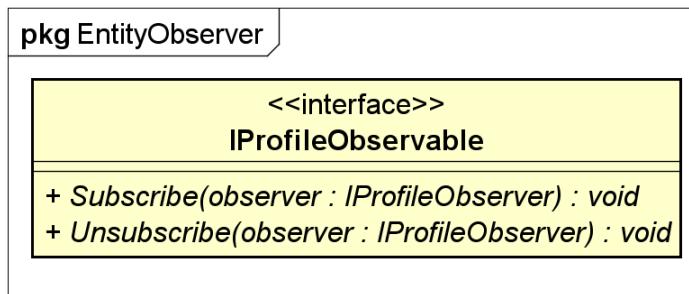


Abbildung 4.53: IProfileObservable Schnittstelle

Beschreibung: Beobachter können sich über diese Schnittstelle bei Datenstrukturen anmelden (*Subscribe*) und wieder abmelden (*Unsubscribe*), um sich über Änderungen an diesen informieren zu lassen.

Paket: EntityObserver

IProfileObserver

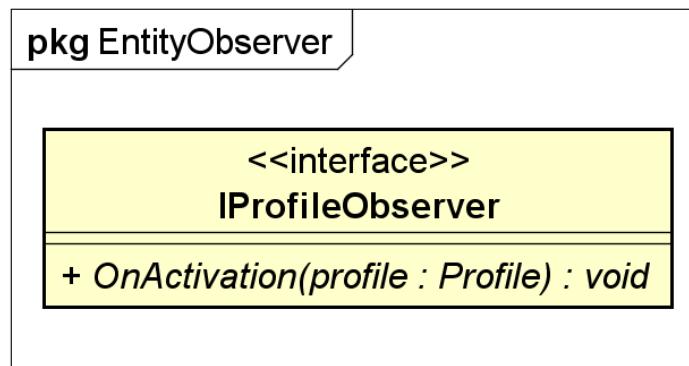


Abbildung 4.54: IProfileObserver Schnittstelle

Beschreibung: Die Schnittstelle **IProfileObserver** erweitert die **IIEntityObserver** Schnittstelle.

Über diese Schnittstelle können angemeldete Beobachter über die Aktivierung eines Objekts benachrichtigt werden.

Paket: EntityObserver

4.3.7 ParserModel

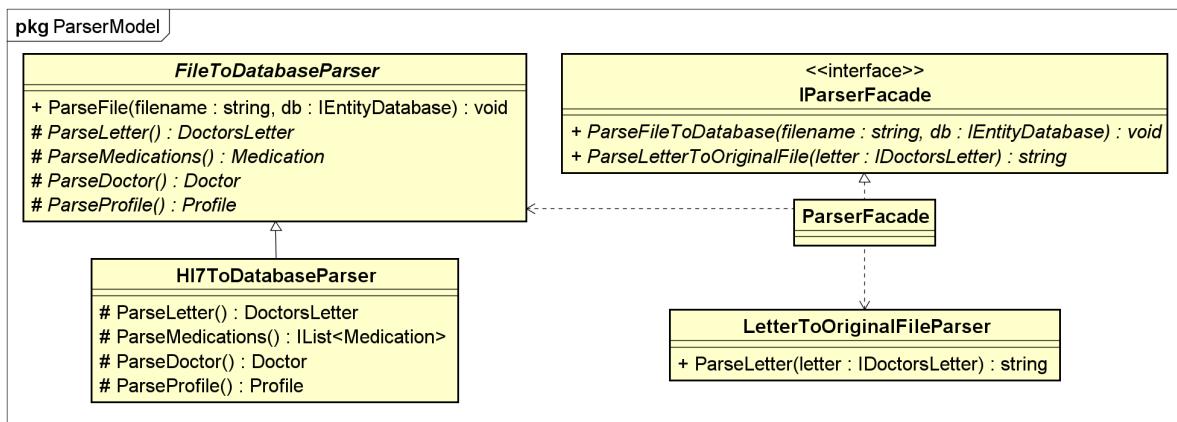


Abbildung 4.55: Klassen des ParserModel Paket

IParserFacade

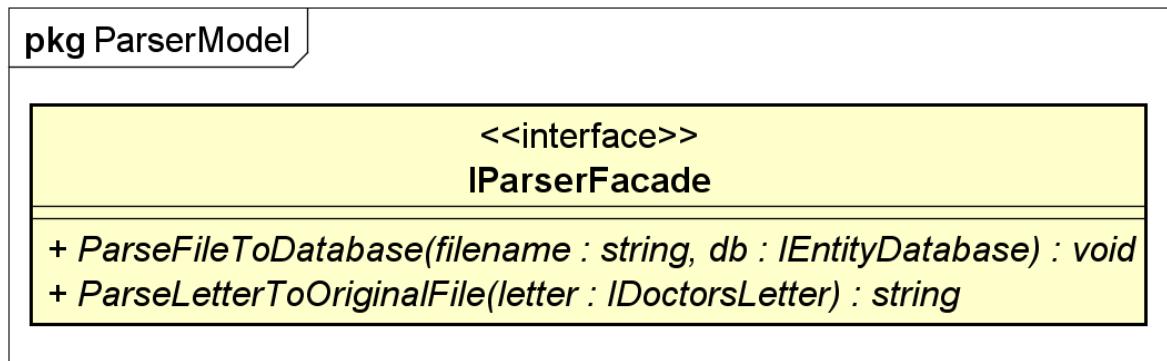


Abbildung 4.56: IParserFacade Schnittstelle

Beschreibung: Diese Schnittstelle ist die Einstiegsstelle in das **ParserModel** Paket. Hier können Arztbriefe aus einer Datei in eine Datenbank eingefügt werden (*ParseFileToDatabase*) oder in eine Datei konvertiert werden (*ParseLetterToOriginalFile*).

Paket: ParserModel

ParserFacade : IParserFacade

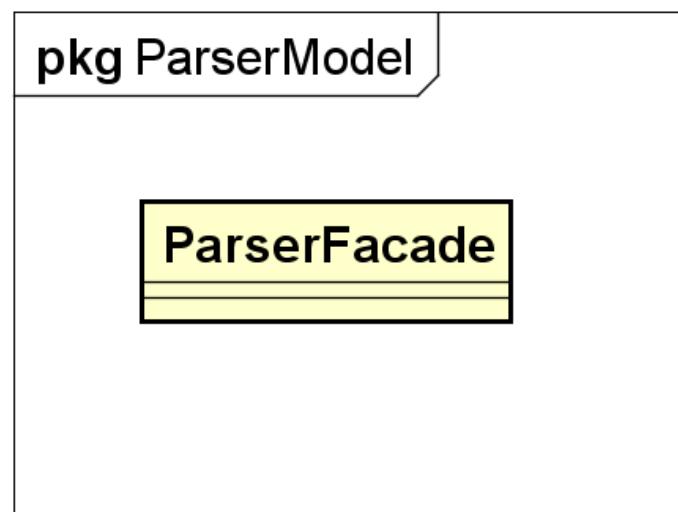


Abbildung 4.57: ParserFacade Klasse

Beschreibung: Implementierung der **IParserFacade** Schnittstelle. Wählt bei Aufruf den passenden Parser aus diesem Paket aus und delegiert an diesen.

Paket: ParserModel

FileToDatabaseParser

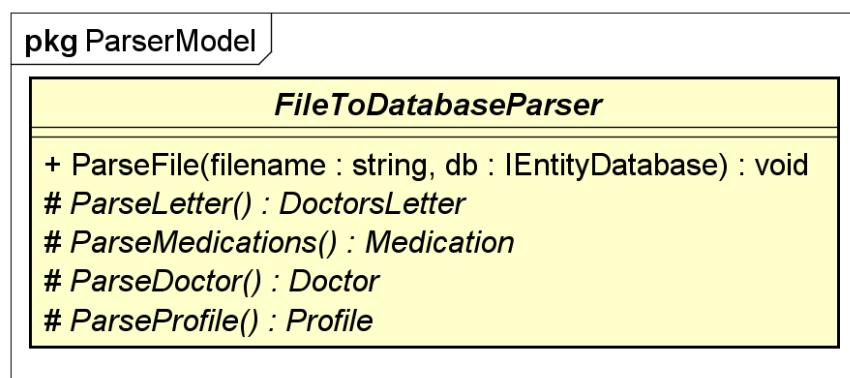


Abbildung 4.58: FileToDatabaseParser Klasse

Beschreibung: Abstrakte Klasse zum Konvertieren eines in einer Datei gespeicherten Arztbriebs in ein **DoctorsLetter** Objekt, dass in einer **IEntityDatabase** gespeichert wird (*ParseFile*).

Die Implementierung der Methoden um die nötigen Informationen, also die Attribute des **DoctorsLetter** Objektes aus der Datei zu extrahieren (*ParseLetter*, *ParseMedications*, *ParseDoctor*, *ParseProfile*), wird den dateiformatspezifischen Unterklassen nach dem Schablonenmuster überlassen.

Paket: ParserModel

HI7ToDatabaseParser : FileToDatabaseParser

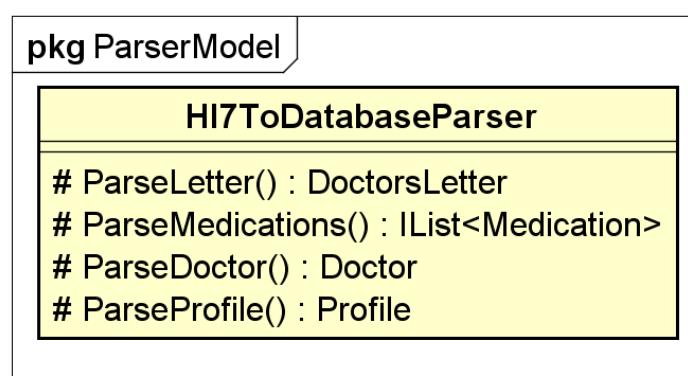


Abbildung 4.59: HI7ToDatabaseParser Klasse

Beschreibung: Implementierung der abstrakten Klasse **FileToDatabaseParser** und ihrer abstrakten Einschubmethoden (*ParseLetter*, *ParseMedications*, *ParseDoctor*, *ParseProfile*) für das .hl7 Dateiformat.

Paket: ParserModel

LetterToOriginalFileParser

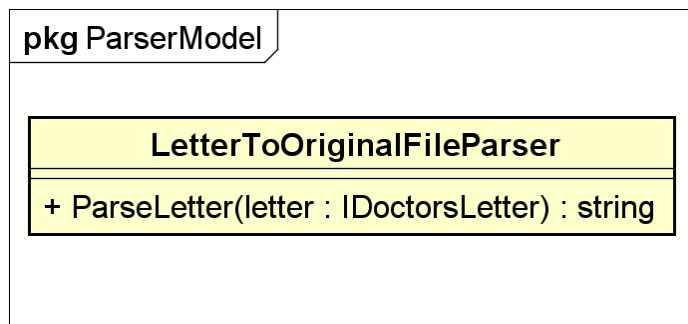


Abbildung 4.60: LetterToOriginalFileParser Klasse

Beschreibung: Konvertiert einen Arztbrief zu dem Dateiformat in dem er vor seiner Erstellung gesendet wurde (*ParseLetter*).

Da die Originaldatei auch nach der Erstellung des Arztbriefs gespeichert bleibt, wird lediglich der Dateipfad mithilfe der **IFileHelper** Schnittstelle zurückgegeben.

Paket: ParserModel

4.3.8 FileHelper

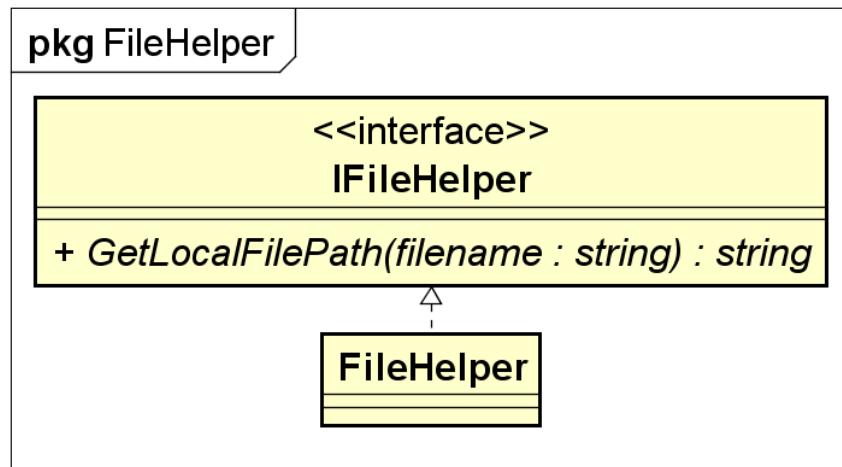


Abbildung 4.61: Klassen des FileHelper Paket

IFileHelper

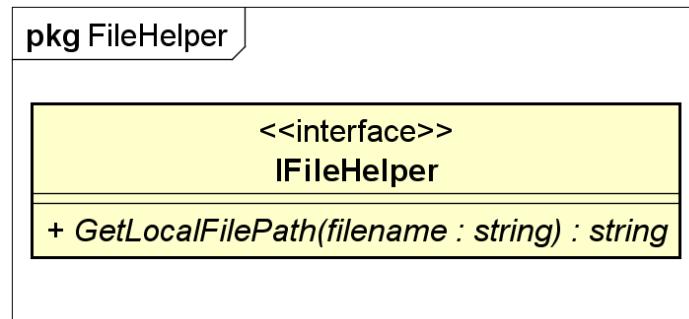


Abbildung 4.62: IFileHelper Schnittstelle

Beschreibung: Schnittstelle um den Namen einer Datei im Verzeichnis der Anwendung in einen vollständigen Dateipfad umzuwandeln (*GetLocalFilepath*).

Paket: FileHelper

FileHelper : IFileHelper

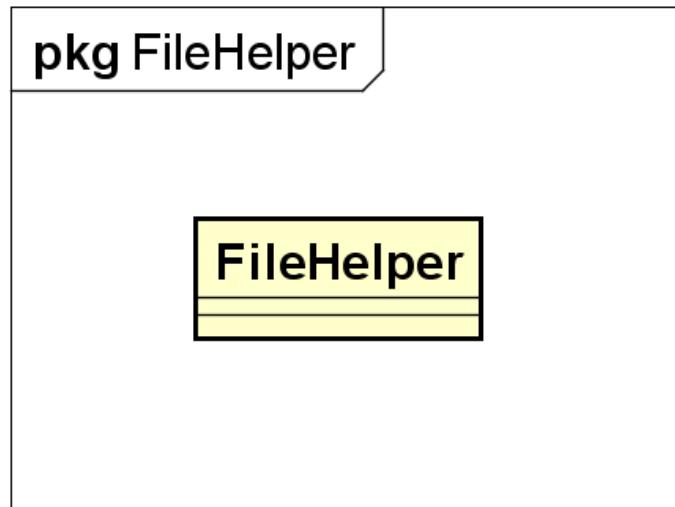


Abbildung 4.63: FileHelper Klasse

Beschreibung: Direkte Implementierung der **IFileHelper** Schnittstelle. Muss plattformspezifisch implementiert werden.

Paket: FileHelper

5 Daten

5.1 Datenspeicherung

Die in der Applikation vorhandenen Daten (also die Objekte aus **DataModel**) werden in einer lokalen SQLite-Datenbank gespeichert.

Der Aufbau der Datenbank ergibt sich dabei direkt aus den Klassen in **DataModel**, d.h. **Profile**, **Doctor**, **DoctorsLetter**, **DoctorsLetterGroup** und **Medication** sind jeweils Vorlagen für eine Tabelle in der Datenbank.

Der Zugriff auf die Datenbank erfolgt über eine Xamarin Cross-Platform Library, die über **DatabaseModel** aufgerufen wird. Weiterhin beobachtet die Datenbank, die in ihr enthaltenen Objekte um über Änderungen an diesen informiert zu werden und sich aktuell zu halten.

Jeder in der Datenbank enthaltene Arztbrief wurde dabei aus einer über das **Transmission-Model** gesendeten Datei (zunächst nur im XML-basierten .hl7 Format) erstellt. Da solche Dateien in der Regel mehr Informationen enthalten als zum Anzeigen nötig sind und in der Datenbank gespeichert werden, wird die Originaldatei ebenfalls gespeichert um diese Informationen nicht zu verlieren.

5.2 Datenübertragung

Die drahtlose Datenübertragung stellt eine Kernfunktion der Anwendung dar. Um diese auf einer möglichst großen Anzahl an Mobilgeräten möglichst nutzerfreundlich umsetzen zu können, ist die Wahl auf den in zahlreichen Mobilgeräten verwendeten Bluetooth-Standard gefallen.

Zu betonen ist, dass in diesem Anwendungsfall das Bluetooth-Profil *GATT* zum Einsatz kommt, da die überwiegende Mehrheit der Mobilgeräte dieses Profil unterstützt.

Die Kommunikation zweier Geräte lässt sich folgendermaßen skizzieren: Der Empfänger

agiert als sogenannter GATT-Server, welcher über beschreibbare Charakteristiken verfügt. Das sendende Gerät verbindet sich mit diesem Server, und schreibt die zu übertragenden Daten in diese Charakteristiken ein, die anschließen vom Empfänger ausgelesen und ausgewertet werden können.

6 Dynamik und Ablauf

6.1 Aktionen

6.1.1 Gesamte Anwendung

6.1.2 Model

Hinweis: Um die Sequenzdiagramme der gesamten Anwendung übersichtlich zu halten, sind hier komplexere Operationen aus dem Model separat vom Rest der Anwendung aufgeführt. Werden solche oder ähnliche Operationen an anderer Stelle referenziert, so wird in diesen Sequenzdiagrammen nur der Aufruf an die Schnittstelle des Models und nicht erneut der genaue Ablauf aufgeführt.

Ändern einer Entität

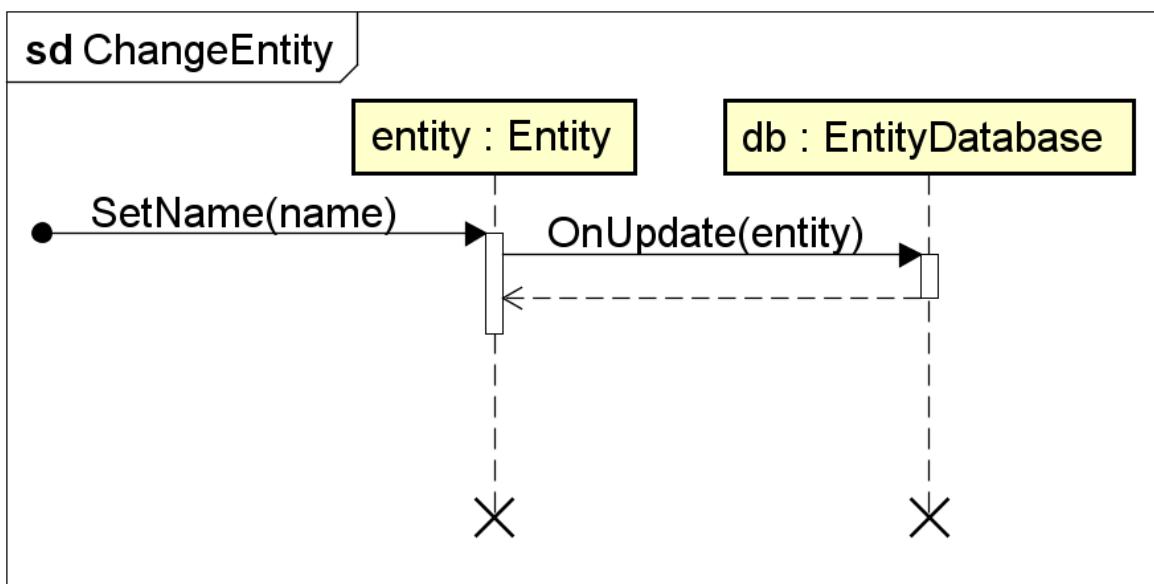


Abbildung 6.1: Ändern einer Entität

Über die Schnittstelle der Entität wird eine Methode(z.B. `SetName` aufgerufen die eines ihrer Attribute verändert.

Die Entität benachrichtigt, nachdem sie die Änderung durchgeführt hat, die Datenbank `db`, die sich zuvor (z.B. bei der Erstellung der Entität über **EntityFactory**) als Beobachter anmeldet hat.

Dieser Aufruf an die Datenbank veranlasst dann die Speicherung der Änderung darin.

Löschen eines Arztbriefs

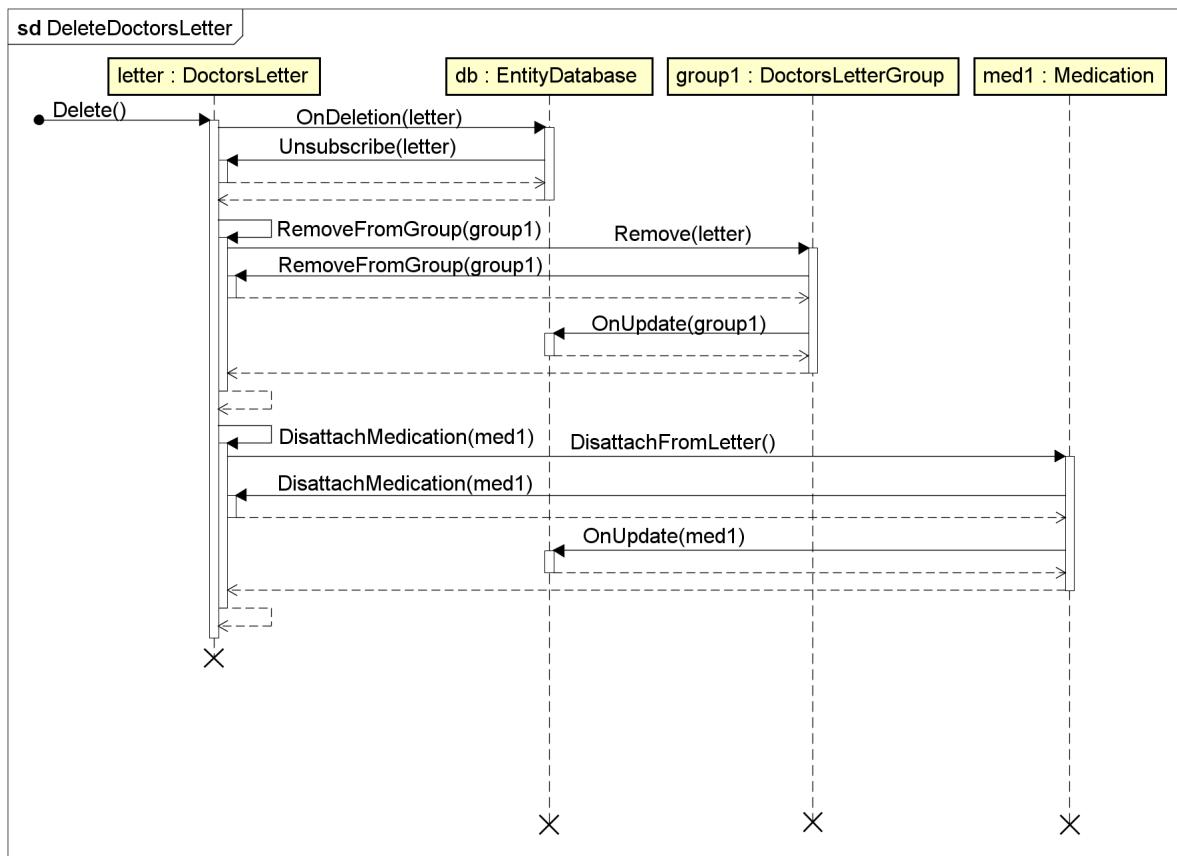


Abbildung 6.2: Löschen eines Arztbriefs

Über dessen Schnittstelle **IDoctorsLetter** wird das Löschen eines Arztbriefs *letter* veranlasst (*Delete*).

Dann wird die **EntityDatabase** *db* über ihre Schnittstelle **IEntityObserver** benachrichtigt, die wiederum *letter* aus der Datenbank löscht und sich von ihm als Beobachter abmeldet.

Nun müssen noch die Assoziationen von *letter* aufgelöst werden (*DisattachMedication*, *RemoveFromGroup*), sodass keine Referenzen mehr zu einer gelöschten Entität übrig bleiben und *letter* von der Garbage Collection eingesammelt werden kann.

Dafür werden die mit *letter* assoziierten Medikationen (*DisattachFromLetter*) und Gruppen (*Remove*) aufgerufen, die daraufhin *db* über diese Änderung an sich benachrichtigen.

Da das Auflösen der Assoziationen rekursiv funktioniert, rufen diese Entitäten daraufhin

erneut *letter* auf, der dann die Rekursion abbricht.

In diesem Beispiel wurde ein Arztbrief verwendet, das Löschen anderer Entitäten funktioniert jedoch nach dem selben Prinzip.

Parsen eines Arztbriefs

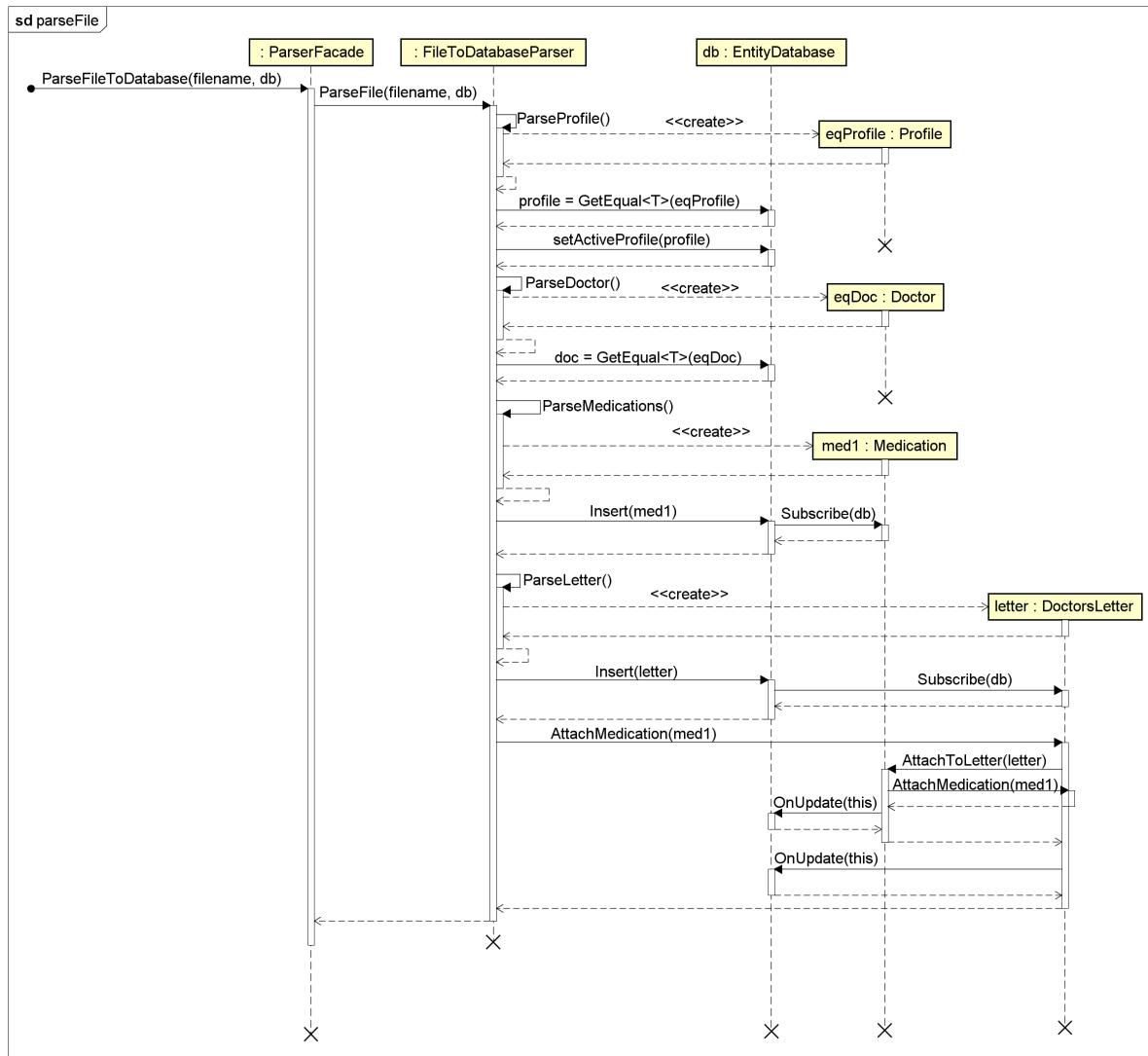


Abbildung 6.3: Parsen eines Arztbriefs

Über die **IParserFacade** Schnittstelle wird das Parsen einer Datei *filepath* in eine **EntityDatabase** *db* veranlasst.

Basierend auf dem in *filepath* angegebenen Dateiformat wird dann der passende **FileToDatabaseParser** ausgewählt und an diesen delegiert (*ParserFile*). Dieser ruft dann nacheinander seine Einschubmethoden auf:

Es werden ein **Profile** (*ParseProfile*) und **Doctor** (*ParseDoctor*) erstellt und dazu passende Objekte in der Datenbank *db* gesucht (*GetEqual*).

Dann werden die verschriebenen Medikationen erstellt (*ParseMedications*) und in *db* eingefügt (*Insert*).

Letztlich wird der Arztbrief selbst mit Assoziationen zu den restlichen geparsten Objekten erstellt (*ParseLetter*) und in *db* eingefügt (*Insert*).

Erstellen einer Medikation

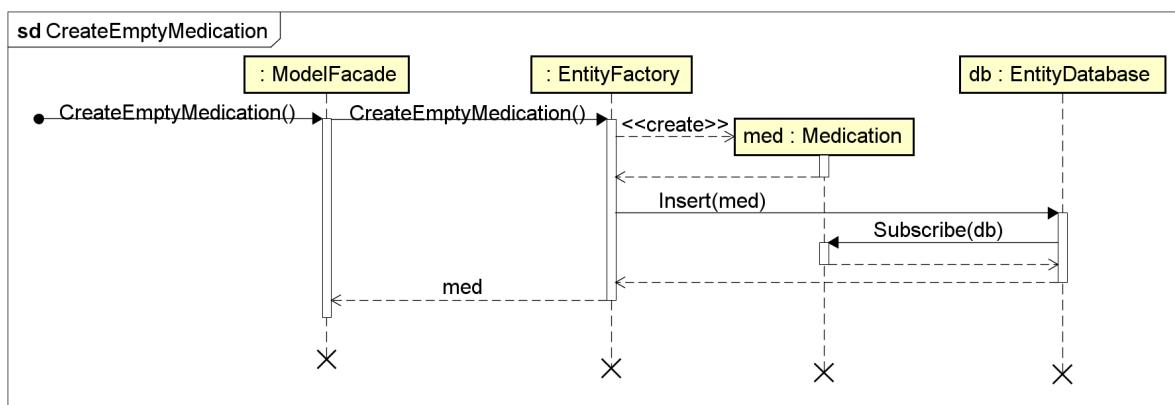


Abbildung 6.4: Erstellen einer Medikation

Über die **IModelFacade** Schnittstelle wird die Erstellung einer Medikation veranlasst (*CreateEmptyMedication*).

Die **ModelFacade** delegiert dann an die **EntityFactory** über ihre Schnittstelle (*CreateEmptyMedication*).

Dort wird dann tatsächlich die **Medication** *med* erstellt. Außerdem wird *med* in die **EntityDatabase** *db* über deren **IEntityDatabase** Schnittstelle eingefügt (*Insert*). Hier meldet sich auch *db* als Beobachter bei *med* an.

Letztlich wird dann *med* an **ModelFacade** und von dort aus an den ursprünglichen Aufrufer zurückgegeben.

6.2 UI-Navigation

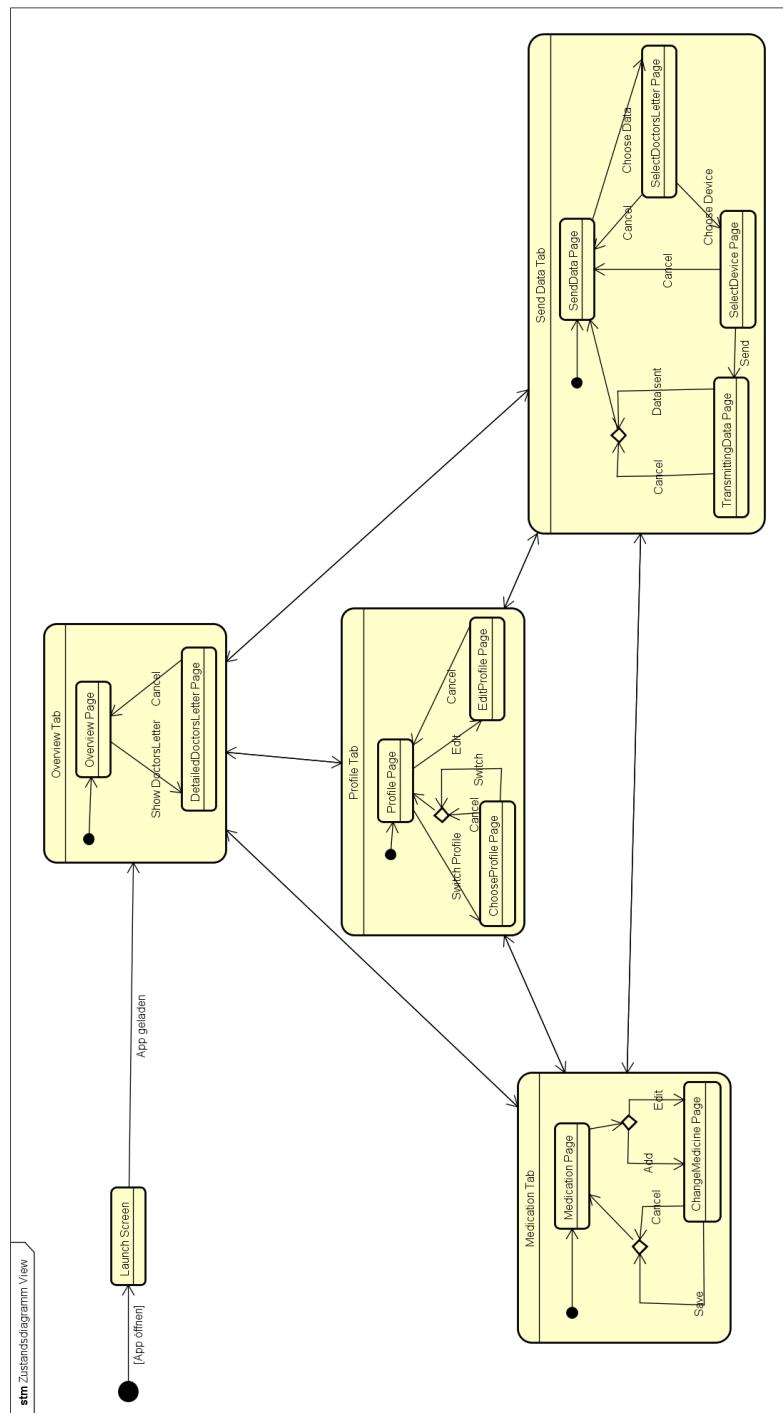


Abbildung 6.5: Zustandsdiagramm der View

Die hauptsächliche Navigation in der Anwendung erfolgt über eine Tab Bar, die es dem Nutzer erlaubt, von jedem der vier Tabs in einen beliebigen anderen wechseln zu können.

Startet der Nutzer die Anwendung zum ersten Mal, so gelangt er nach einer Begrüßung durch den Launch Screen zunächst in den Overview Tab. Hier bietet sich ihm nun die Möglichkeit, durch das Auswählen eines beliebigen Arztbriefes in eine detaillierte Ansicht des Briefes zu gelangen. Von hieraus gelangt der Nutzer nur durch einen Klick auf den Schließen Knopf zurück zur Overview Page.

Wechselt der Nutzer nun in den Medication Tab, gelangt er zunächst in die Medication Page. Diese stellt eine Übersicht über alle eingetragenen Medikationen dar. Über einen Hinzufügen Knopf gelangt man von dort aus in die ChangeMedicine Page, die das Anlegen einer neuen Medikation ermöglicht. Mit einem Klicken auf Speichern oder Abbrechen gelangt man wieder zurück zur Medication Page. Eine zweite Möglichkeit, die ChangeMedicine Page zu erreichen, stellt das Tippen auf einen beliebigen Eintrag der MedicationPage dar: Dadurch bietet sich die Möglichkeit, die vorhandenen Werte der Medikation anzupassen.

Der SendData Tab präsentiert dem Nutzer zunächst die SendData Page, von wo aus ein Tippen auf einen Knopf weiterleitet zur SelectDoctorsLetter Page, die es ermöglicht, die zu übertragenden Arztbriefe auszuwählen. Sind alle Dateien ausgewählt, gelangt der Nutzer über einen Knopf zur SelectDevice Page, wo er einen Empfänger auswählen kann. Hat er auch das erledigt, leitet ihn ein letzter Knopf zur TransmittingData Page weiter, die ihm einen Überblick über den Übertragungszustand bietet. Nach erfolgreicher Datenübertragung gelangt er über einen Knopf zurück zur SendData Page. Alle der drei genannten Pages Verfügung zudem über einen Cancel Button, der den Nutzer direkt zurück zur SendData Page leitet, und beispielsweise den Datenübertragungsvorgang abbricht.

Der letzte der vier Tabs, der Profile Tab, stellt dem Nutzer zunächst das aktuelle Profil in der Profil Page dar. Hier hat der Benutzer nun zwei Möglichkeiten: Entweder er entschließt sich zum Wechsel des aktiven Profils durch einen Knopfdruck Switch Profile, oder er möchte das aktuelle Profil editieren. Ersteres leitet ihn weiter zur ChooseProfile Page, wo er aus den gespeicherten Profilen zu einem beliebigen wechseln kann. Ein Speichern Knopf sichert die Änderung und leitet zurück zur Profile Page. Möchte der Nutzer allerdings ein Profil editieren, erreicht er dies über einen Edit Knopf, der ihn zur EditProfile Page weiterleitet. Hier können entweder die eingetragenen Werte geändert und durch einen Speichern Knopf gesichert werden, oder durch man bricht den Vorgang durch einen Cancel Knopf ab. Beides

leitet den Nutzer zurück zur ProfilePage.

Die Anwendung lässt sich zu jedem Zeitpunkt beenden. Aktive Datenübertragungsvorgänge werden dadurch jedoch beendet. Zudem gehen nichtgespeicherte Änderungen jeglicher Art verloren.

6.3 UI-Seiten

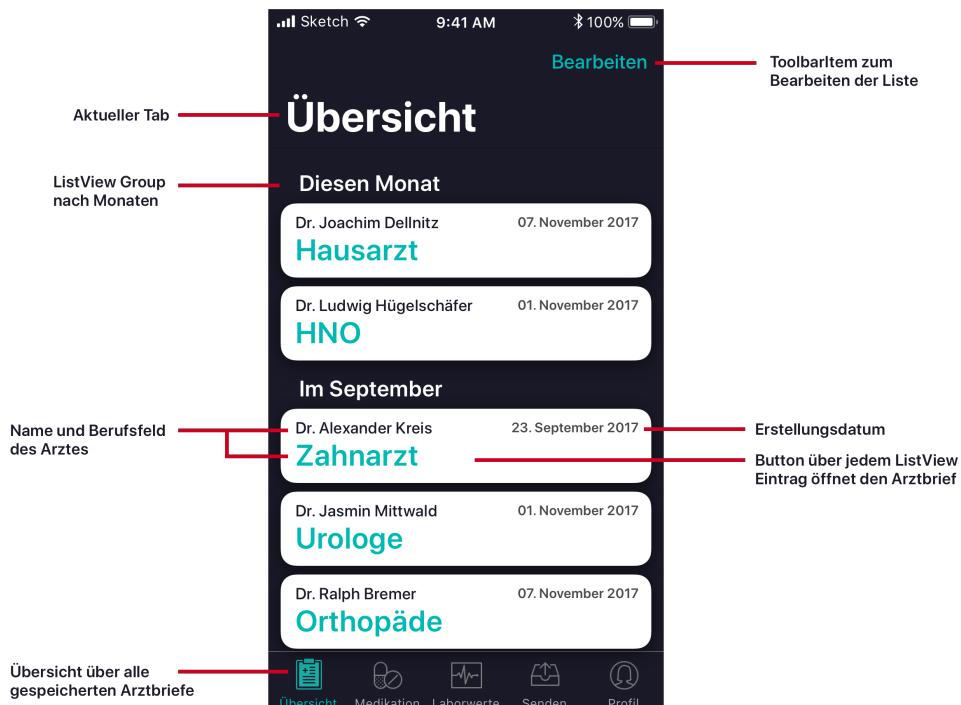


Abbildung 6.6: Beschreibung des Tabs "Übersicht"

Wird die Anwendung geöffnet, wird dem Nutzer zunächst eine Übersicht all seiner gespeicherten Arztbriefe angezeigt (Overview Page). Die Arztbriefe sind hierbei chronologisch absteigend in einer ListView angeordnet. Tippt der Nutzer einen Listeneintrag an, öffnet sich eine ausführliche Ansicht des Arztbriefes.

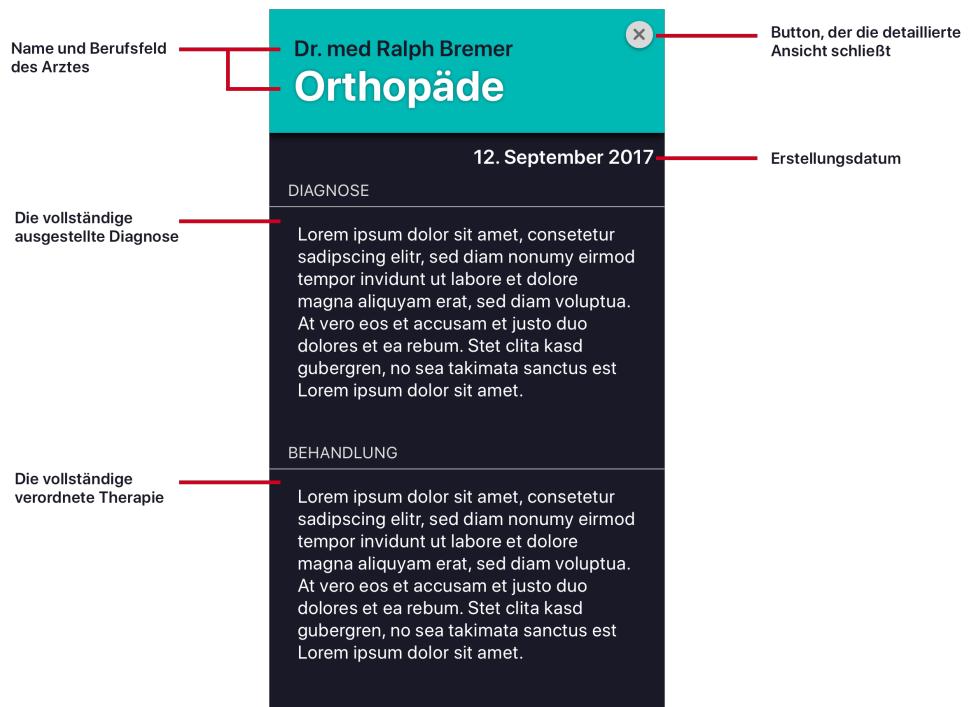


Abbildung 6.7: Beschreibung der detaillierten Ansicht eines Arztbriefes

Tippt der Nutzer einen Arztbrief in der Übersicht an, wird er auf eine detaillierte Darstellung des Arztbriefes weitergeleitet. Hier werden, neben dem Datum, dem Namen und Fachrichtung des Arztes, die hinterlegten Daten der Diagnose und der Behandlung vollständig angezeigt. Zum Schließen einer Detailansicht des Arztbriefes, kann der Nutzer auf den "Schließen" Knopf drücken und wird dann zur Übersicht geleitet.

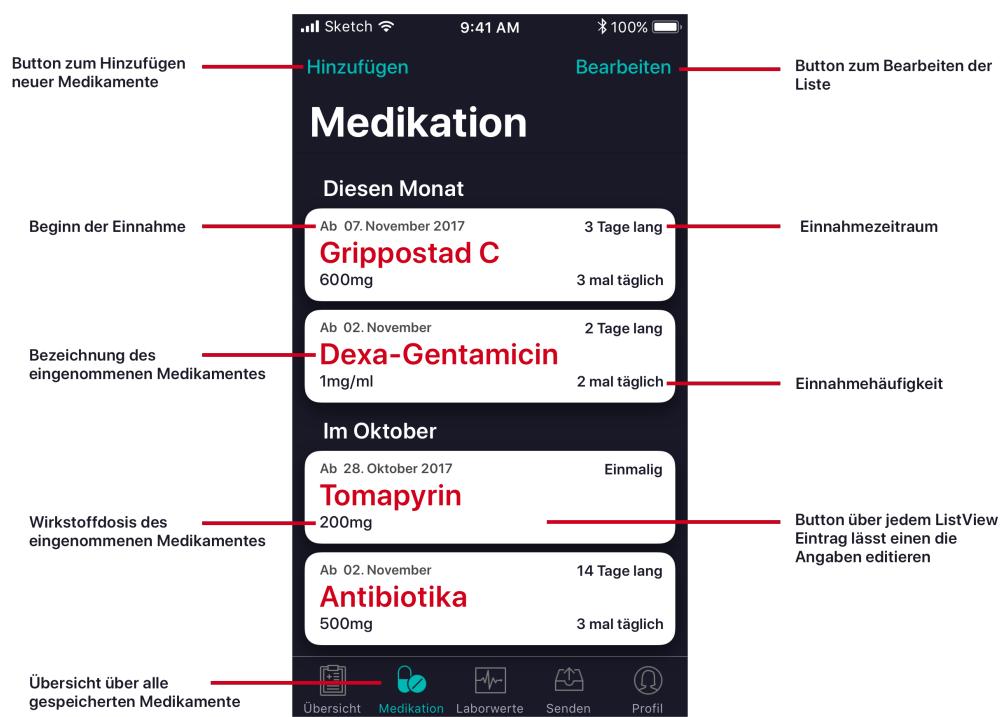


Abbildung 6.8: Beschreibung des Tabs "Medikation"

Im Tab "Medikation" kann der Nutzer eine Liste von eingenommenen/einzunehmenden Medikamenten verwalten. Dabei wird jeder Eintrag (Medikament) mit allen hinterlegten Daten chronologisch absteigend angezeigt. Um einen Eintrag zu editieren, kann der Nutzer einen Eintrag antippen.

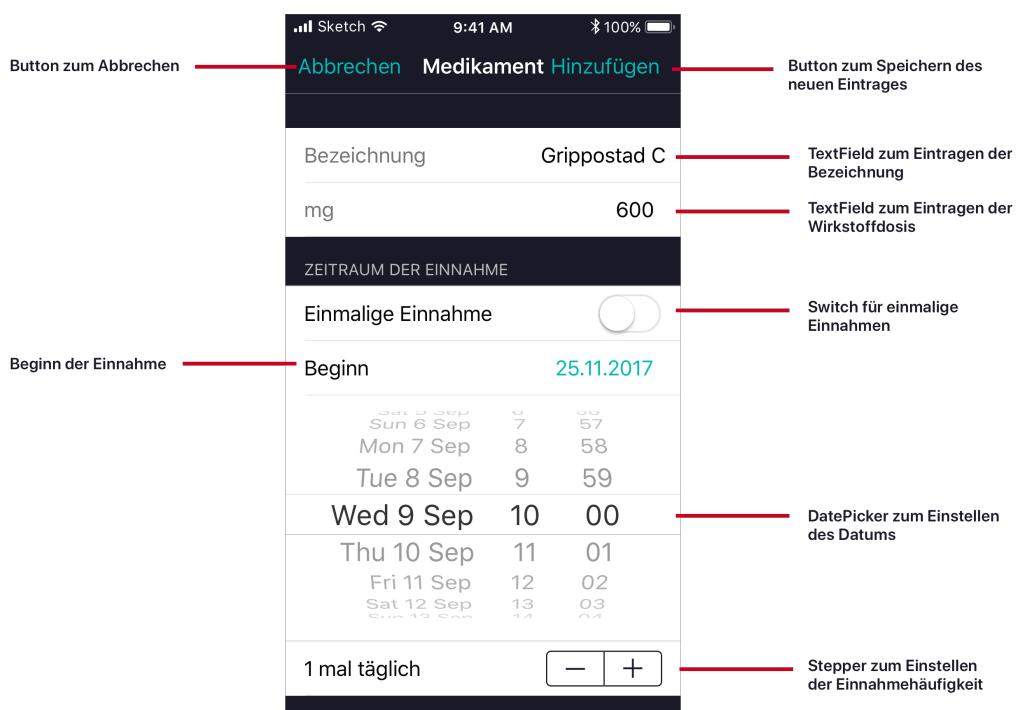


Abbildung 6.9: Beschreibung der Ansicht zum Editieren einer Medikation

In dieser Ansicht lassen sich alle Daten zu einem Medikament editieren. Der Nutzer gelangt in diese Ansicht, indem er entweder ein neues Medikament hinzufügt oder ein bereits bestehendes Medikament editiert. Sobald der Nutzer alle Werte eingetragen hat, kann er auf den "Hinzufügen" Button tippen, um seine Eingabe/Änderung zu speichern. Falls der Nutzer seine Eingabe jedoch nicht speichern möchte, kann er auf den "Abbrechen" Button tippen, um ohne eine Datenspeicherung wieder zurück in den "Medikation" Tab zu gelangen.

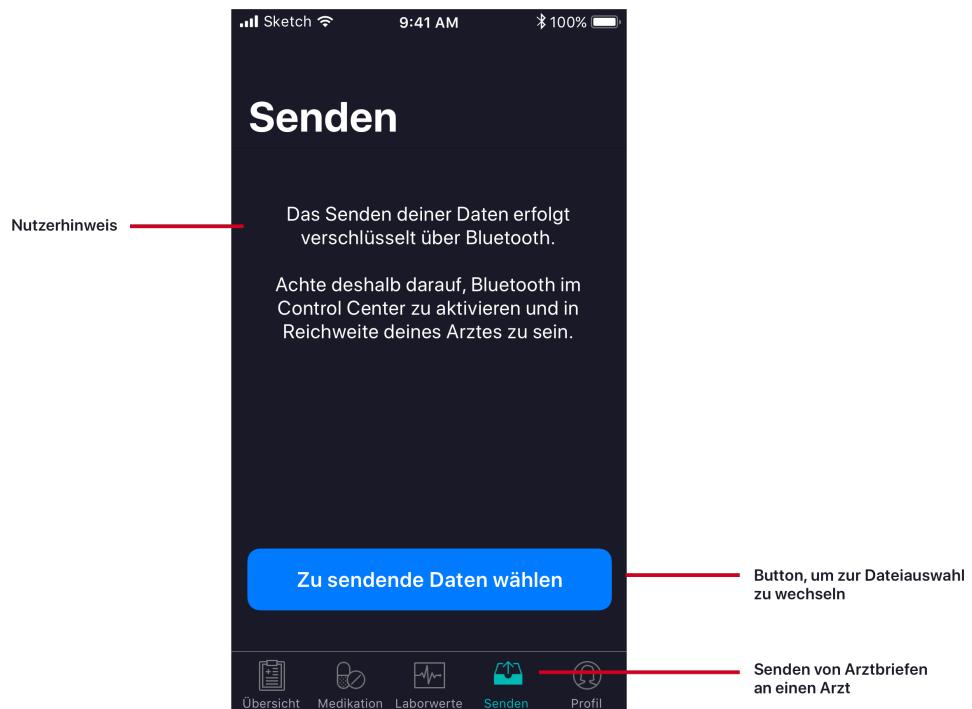


Abbildung 6.10: Beschreibung des Tabs "Senden"

Tippt der Nutzer auf den "Senden" Tab wird ihm ein Nutzerhinweis zur Datenübertragung und ein Button, der zur Dateiauswahl für einen Sendeprozess weiterleitet, angezeigt. Der Sendeprozess wiederum besteht aus einer Abfolge von drei Navigationsseiten: Zunächst wird der Nutzer aufgefordert, aus seinem Bestand an gespeicherten Arztbriefen diejenigen durch Tippen auszuwählen, die er gerne versenden würde. Hat er dies getan, führt in ein Klick auf einen *Weiter-Knopf* zu einer Ansicht, welche ihm alle in der Nähe erreichbaren Geräte, also potentielle Empfänger, auflistet. Aus dieser Liste kann er nun genau eines auswählen. Hat er den gewünschten Empfänger gefunden und ausgewählt, wird er zur dritten und letzten Ansicht weitergeleitet, die ihm den Fortschritt der nun laufenden Dateiübertragung anzeigt. Ist diese abgeschlossen, wird der Nutzer automatisch zurück zur obigen Startseite des Tabs *Senden* zurück geleitet.

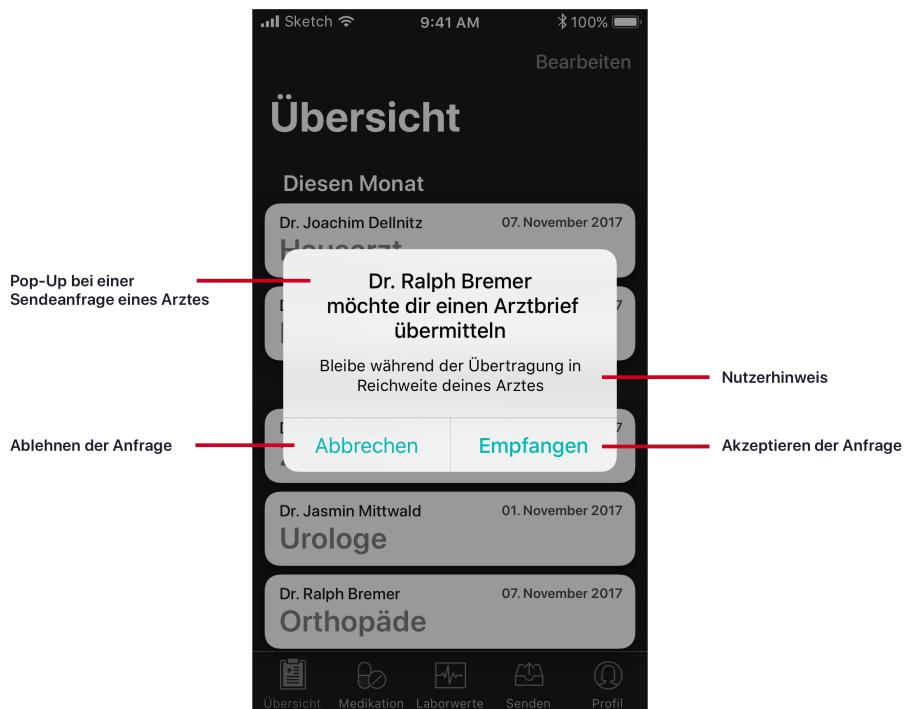


Abbildung 6.11: Beschreibung der Ansicht einer Sendeanfrage

Die Sendeanfrage wird durch ein Pop-Up Fenster realisiert. Das Pop-Up kann in jeder Ansicht erscheinen. In dem Pop-Up wird der Name des Senders, ein Nutzerhinweis und zwei Buttons angezeigt. Es gibt einen "Abbrechen" Button und einen "Empfangen" Button. Tippt der Nutzer den "Abbrechen" Button an, wird die Sendeanfrage abgelehnt und es findet keine Datenübertragung statt. Tippt der Nutzer jedoch den "Empfangen" Button an werden die Daten vom Sender an den Nutzer übertragen.

7 Anwendung für Ärzte

Die Version der Anwendung die nicht von Patienten, sondern von Ärzten verwendet werden soll um Patienten deren Arztbriefe zu senden, wird in diesem Dokument nicht genauer beschrieben, da sie nur eine reduzierte Version der hier beschriebenen mobilen Anwendung ist.

Die Anwendung für Ärzte soll nur Dateien mit mobilen Anwendungen von Patienten austauschen können, also sind dort die Pakete für die Modellierung (**DataModel**, **EntityFactory**) und Speicherung (**DatabaseModel**, **EntityObserver**) der Daten überflüssig.

Somit wird auch die Logik zur Konvertierung zwischen den Speicherungsformaten (**ParserModel**) nicht mehr benötigt.

Weiterhin werden die gesendeten/empfangenen Dateien nicht im Dateisystem der Anwendung, (was die Funktion des **FileHelper** war), sondern an einem beliebigen Dateipfad gespeichert.

Die einzige für diese Anwendung nötige Funktionalität des Models ist somit komplett in **TransmissionModel** enthalten.

Was die View betrifft, so reicht für die Anwendung für Ärzte eine leicht modifizierte Variante des Pakets **SendDataTabPages** aus, um die notwendigen Funktionen darzustellen. Die Modifikationen betreffen lediglich die Darstellungsweise, die sich auf einem Windows-Computer von Mobilgeräten unterscheidet.

Durch den Wegfall großer Teile der View reduziert sich auch der Umfang des ViewModels gewaltig. Übrig bleiben lediglich noch die Klassen des **SendDataViewModels**, die in leicht abgeänderter Version die Änderungen der View widerspiegeln.

8 Klassendiagramm

Die folgende Grafik zeigt nun abschließend das myMD zugrunde liegende System in vollem Umfang mit allen Klassen, Attributen, Methoden und Assoziationen. Es setzt sich zusammen aus den Paketen des Models, der View und des ViewModels, die gemäß der Model-View-ViewModel Idee ein kohärentes System darstellen.

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 8.1: QR Code zur Online-Variante des myMD Klassendiagrammes

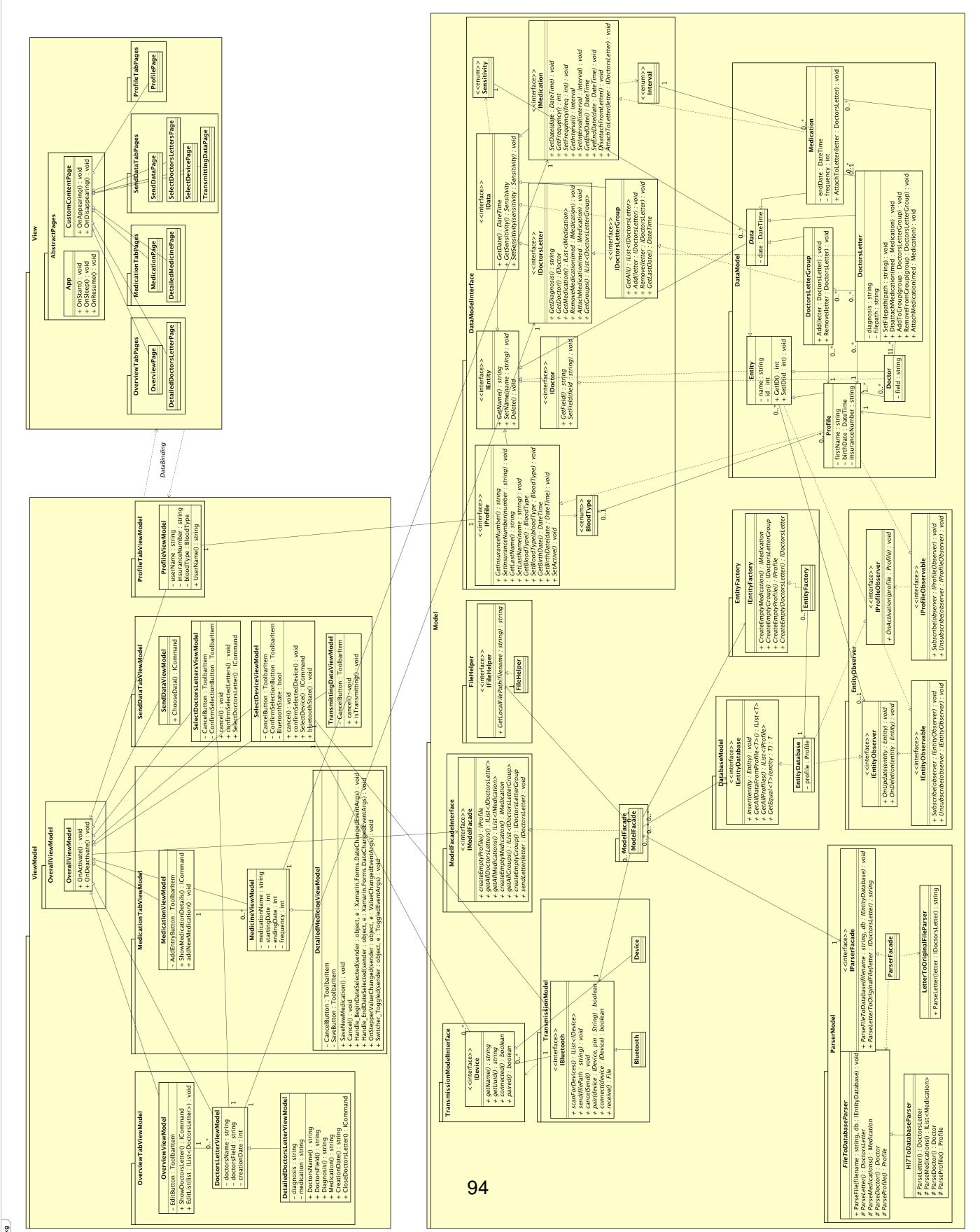


Abbildung 8.2: myMD Klassendiagramm

Glossar

Anamnese Die Anamnese (von altgriechisch ἀνάμνησις, deutsch ‚Erinnerung‘) ist die professionelle Erfragung von potenziell medizinisch relevanten Informationen durch Fachpersonal (z. B. einen Arzt). 93

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. 93

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. 93

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. 93

Cloud Die Cloud ist keine physische Größe, sondern ein riesiges Netzwerk aus Remote-servers, die über die ganzen Welt verteilt aber miteinander verbunden sind, damit sie als ein einziges großes Ökosystem funktionieren können. 93

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“. 93

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 . 93

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. 93

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. 93

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. 93

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. 93

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. 93

Tab Eine Registerkarte, auch Reiter oder Tab genannt, ist ein Steuerelement einer grafischen Benutzeroberfläche, das einem Registerblatt aus Aktenschränken nachempfunden wurde . 93

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. 93

Abbildungsverzeichnis

2.1 Pakete des Subsystems View	5
2.2 Pakete des Subsystems ViewModel	6
2.3 Pakete des Subsystems Model	7
2.4 TransmissionModel Schnittstellen	8
2.5 DataModel Schnittstellen	9
2.6 IEntity Schnittstelle	9
2.7 IData Schnittstelle	10
2.8 IDoctorsLetter Schnittstelle	11
2.9 IDoctorsLetterGroup Schnittstelle	12
2.10 IMedication Schnittstelle	13
2.11 IProfile Schnittstelle	14
2.12 IDoctor Schnittstelle	15
2.13 ModelFacade Schnittstelle	16
3.1 Pakete der View	18
3.2 Pakete des ViewModels	20
3.3 Pakete des Models	22
3.4 Benutztrelation des Systems	26
4.1 Klassen des AbstractPages Paket	27
4.2 CustomContentPage Klasse	28
4.3 App Klasse	28
4.4 Klassen des OverviewTabPages Paket	29
4.5 OverviewPage Klasse	30
4.6 DetailedDoctorsLetterPage Klasse	30
4.7 Klassen des MedicationTabPages Paket	31
4.8 MedicationPage Klasse	32
4.9 DetailedMedicinePage Klasse	32
4.10 Klassen des SendDataTabPages Paket	33

4.11 SendDataPage Klasse	34
4.12 SelectDoctorsLettersPage Klasse	34
4.13 SelectDevicePage Klasse	35
4.14 TransmittingDataPage Klasse	36
4.15 ProfilePage Klasse	37
4.16 OverallViewModel Klasse	38
4.17 Klassen des OverviewTabViewModel Paket	39
4.18 OverviewViewModel Klasse	39
4.19 DoctorsLetterViewModel Klasse	40
4.20 DetailedDoctorsLetterViewModel Klasse	41
4.21 Klassen des MedicationTabViewModel Paket	42
4.22 MedicationViewModel Klasse	43
4.23 MedicineViewModel Klasse	44
4.24 DetailedMedicineViewModel Klasse	45
4.25 Klassen des SendDataTabViewModel Paket	47
4.26 SendDataViewModel Klasse	48
4.27 SelectDoctorsLettersViewModel Klasse	48
4.28 SelectDeviceViewModel Klasse	50
4.29 TransmittingDataViewModel Klasse	51
4.30 ProfileViewModel Klasse	52
4.31 ModelFacade Klasse	53
4.32 Klassen des TransmissionModel Paket	54
4.33 IBluetooth Schnittstelle	55
4.34 Bluetooth Klasse	56
4.35 Device Klasse	56
4.36 Klassen des DatabaseModel Paket	57
4.37 IEntityDatabase Schnittstelle	58
4.38 EntityDatabase Klasse	59
4.39 Klassen des DataModel Paket	60
4.40 Entity Klasse	60
4.41 Profile Klasse	61
4.42 Doctor Klasse	62
4.43 Data Klasse	63
4.44 DoctorsLetter Klasse	64
4.45 DoctorsLetterGroup Klasse	65
4.46 Medication Klasse	65

4.47 Klassen des EntityFactory Paket	66
4.48 IEntityFactory Schnittstelle	67
4.49 EntityFactory Klasse	67
4.50 Klassen des EntityObserver Paket	68
4.51 EntityObservable Schnittstelle	68
4.52 IEntityObserver Schnittstelle	69
4.53 IProfileObservable Schnittstelle	70
4.54 IProfileObserver Schnittstelle	70
4.55 Klassen des ParserModel Paket	71
4.56 IParseFacade Schnittstelle	71
4.57 ParseFacade Klasse	72
4.58 FileToDatabaseParser Klasse	72
4.59 HI7ToDatabaseParser Klasse	73
4.60 LetterToOriginalFileParser Klasse	74
4.61 Klassen des FileHelper Paket	74
4.62 IFileHelper Schnittstelle	75
4.63 FileHelper Klasse	75
6.1 Ändern einer Entität	80
6.2 Löschen eines Arztbriefs	81
6.3 Parsen eines Arztbriefs	82
6.4 Erstellen einer Medikation	83
6.5 Zustandsdiagramm der View	84
6.6 Beschreibung des Tabs "Übersicht"	86
6.7 Beschreibung der detaillierten Ansicht eines Arztbriefes	87
6.8 Beschreibung des Tabs Medikation"	88
6.9 Beschreibung der Ansicht zum Editieren einer Medikation	89
6.10 Beschreibung des Tabs SSenden"	90
6.11 Beschreibung der Ansicht einer Sendeanfrage	91
8.1 QR Code zur Online-Variante des myMD Klassendiagrammes	93
8.2 myMD Klassendiagramm	94