ulm university universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Entwicklung eines Importmoduls für medizinische und psychotherapeutische Anwendungen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Vincenzo Francesco Brancaccio
vincenzo.brancaccio@uni-ulm.de
519883

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Michael Stach

2019

Fassung 3. Juli 2019

Satz: PDF-LaTeX 2$_\varepsilon$

Abstract

Die Zahl der Gesundheits-Apps wächst stetig. Ihr Potential für Vorsorge oder Therapie wird von Ärzten anerkannt, allerdings mangelt es an wissenschaftlicher Evaluation. Für Nutzer stehen nur subjektive Bewertungen durch andere Nutzer, die Laien sind, zur Verfügung. Ziel der mobile Health App Database (mHAD) ist es in Zusammenarbeit mit Experten der jeweiligen medizinischen und psychotherapeutischen Fachrichtungen diesem Umstand Abhilfe zu schaffen. Die Experten bewerten Gesundheits-Apps nach festgelegten Kriterien. Diese Ratings werden an die mHAD geschickt, die sie speichert und Nutzern der mHAD anzeigt. In der vorliegenden Arbeit wurde der Import der Ratings in die mHAD neu implementiert. Der Import umfasst Daten aus zwei verschiedenen Datenformaten und lässt sich jetzt leicht um zusätzliche Formate erweitern. Die Daten werden validiert, damit nur formal korrekte und anzeigbare Daten in die Datenbank übernommen werden. Außerdem wurden graphische Benutzeroberflächen für den Upload von Dateien und anschließendes Anzeigen der Daten in tabellarischer Form entwickelt. Mit diesem System können Experten ihre Ratings vor dem Import in die Datenbank validieren lassen und korrigieren.

Inhaltsverzeichnis

1 Einleitung

Zuerst wird dargelegt, warum die *mobile Health App Database* (mHAD) für Nutzer von Gesundheits-Apps entwickelt wird. Anschließend wird die Problemstellung, die dieser Arbeit zugrunde liegt, erläutert und eine Übersicht über den Aufbau der Arbeit gegeben.

1.1 Motivation

Die Zahl der Gesundheits-Apps steigt kontinuierlich an. Ihre Zahl lag 2017 bei 325.000, 78.000 mehr als 2016 [25]. Das liegt an den technischen Fortschritten im Bereich der Endgeräte und dem Optimismus der Anbieter, weshalb sowohl kommerzielle, als auch private Entwickler auf den Markt drängen [9].
Das *Deutsche Ärzteblatt* erkennt das große Potential der Gesundheits-Apps, warnt aber auch vor unkritischem Einsatz, da die klinisch-wissenschaftliche Evaluation bei den meisten noch ausstehe [11]. Für den Verbraucher steht meist nur die Bewertung der App durch andere Benutzer zur Verfügung. Für einen sensiblen Bereich wie Gesundheit ist das ungenügend.
Für Nutzer wäre eine Datenbank von Gesundheits-Apps mit Bewertungen durch Experten deshalb von großem Nutzen. Diese Datenbank zu werden, ist das Ziel der mHAD. Experten bewerten Gesundheits-Apps nach verschiedenen Kriterien und laden diese Informationen in die Datenbank hoch. Nutzer der mHAD können die Gesundheits-Apps dann besser einschätzen.

1.2 Problemstellung

Die Daten, die die Experten generieren, werden in die Datenbank übernommen, damit sie den Nutzern der mHAD angezeigt werden können. Dazu bedarf es Mechanismen um die Daten

1. hochzuladen und zu validieren
2. zu importieren, falls sie korrekt sind

Es muss bestimmt werden, welche Daten erforderlich sind und inwieweit leere Felder oder inkorrekte Daten toleriert werden. Diese Vorgaben erlauben den Experten formal korrekte Dateien zu erstellen. Sie ermöglichen auch die automatische Validierung der Daten in die mHAD zu implementieren.
Die Daten können in unterschiedlichen Formaten vorliegen, sowohl was Datenstrukturen, als auch was Dateiformate betrifft. Der Datenimport ist schwer erweiterbar. Der neue Datenimport muss flexibel und leicht erweiterbar sein, so dass er an neue Anforderungen angepasst werden kann. Eine Validierung der Daten, bevor sie in die Datenbank importiert werden, findet nicht statt. Nach Erstellung der Vorgaben, kann die Validierung implementiert werden.
Es existiert keine Benutzeroberfläche, die das Hochladen von Dateien erlaubt und die Daten anzeigt. Es kann nur eine korrekte Datei importiert werden. Wenn der Import verweigert wird, erhalten Experten keine Rückmeldung. Eine Benutzeroberfläche, die hochgeladene Daten anzeigt und deren Modifikation erlaubt, erhöht die Benutzerfreundlichkeit.

1.3 Struktur der Arbeit

Die Arbeit gliedert sich in zwei Meilensteine: Datenimport und Dateiupload. Dies schlägt sich in der Struktur der Arbeit nieder. Die folgenden Kapitel enthalten jeweils Unterkapitel zu den beiden Projekten.
In Kapitel 2 werden erst die allgemeinen Anwendungsfälle dargestellt, dann folgt je ein Unterkapitel zu den Funktionalen und zu den nicht-Funktionalen Anforderungen. In diesen Unterkapiteln werden die Anforderungen an Datenimport und Dateiupload getrennt behandelt.

Die theoretischen Vorüberlegungen in Kapitel 3 betreffen hauptsächlich den Datenimport, für den die Definition der Eingabeparameter entscheidend ist. Auch der Dateiupload ist davon betroffen, weil die Eingaben ebenfalls validiert werden müssen, bevor sie in die Datenbank übertragen werden können.
Kapitel 4 beschreibt die Architektur der Projekte auf einer höheren Abstraktionsebene. Jedes Projekt hat ein eigenes Unterkapitel, in dem beschrieben wird, was warum und wie implementiert ist. Im Fall des Datenimports wird zusätzlich auf die vorgefundene Implementierung eingegangen, die die Ausgangsbasis für den neuen Datenimport bildet. Der Dateiupload ist eine neue Funktionalität und baut auf keinem vorhergehenden Code auf.
In Kapitel 5 werden ausgewählte Aspekte der Implementierung näher erläutert. Zuerst erfolgt eine Übersicht und kurze Beschreibung verwendeter Technologien, die für das Verständnis der Arbeit wichtig sind. Anschließend werden in Unterkapiteln wichtige, oder herausfordernde Methoden aus beiden Projekten mit Codebeispielen vorgestellt.
Die in Kapitel 2 eingeführten Anforderungen, werden in Kapitel 6 auf ihre Erfüllung im Rahmen dieser Arbeit hin untersucht. Das siebte Kapitel enthält eine Zusammenfassung der Arbeit und einen Ausblick auf zukünftige Erweiterungen des Datenimports und Dateiuploads.

2 Anforderungsanalyse

Die vorliegende Arbeit basiert auf den Anforderungen, die sich aus der Arbeit der Experten, die die Apps bewerten, und den Nutzern der mHAD ergeben. Zuerst werden die Anwendungsfälle beschrieben und dann die, sich daraus ableitenden, Funktionalen und nicht-Funktionalen Anforderungen.

2.1 Anwendungsfälle

In Abbildung 2.1 sind die Anwendungsfälle für Experten und Nutzer graphisch dargestellt.

Bewertung hochladen
Eine Bewertung wird auf Vollständigkeit und formale Voraussetzungen der einzugebenden Parameter *Topic* und Datei geprüft. Falls die Voraussetzungen erfüllt sind, wird die Bewertung in das System hochgeladen. Sind die Voraussetzungen nicht erfüllt, wird der Vorgang abgebrochen und die Fehler angezeigt.

Bewertung verbessern
Die Inhalte der Datei werden auf die Erfüllung formaler Voraussetzungen geprüft und dem Experten angezeigt. Unter der Bedingung, dass die Inhalte der Datei nicht formal korrekt sind, können die Inhalte der Datei verbessert werden.

Bewertung importieren
Die Bewertung und die Inhalte der Datei werden auf formale Korrektheit geprüft. Unter der Bedingung, dass die Bewertung formal korrekt ist, werden die Daten in die Datenbank importiert. Falls die Prüfung nicht bestanden wird, bricht der Importvorgang ab.

Bewertung lesen
Eine Anfrage an die Datenbank wird daraufhin geprüft, ob Daten in der Datenbank

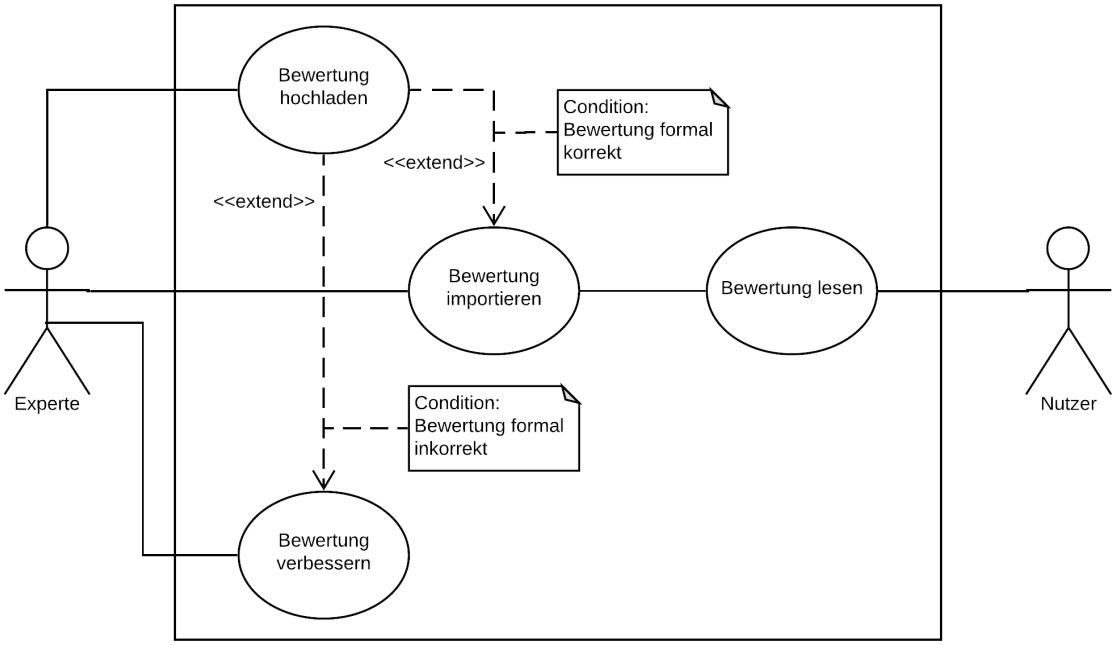

Abbildung 2.1: UML-Anwendungsfalldiagramm zeigt die, für diese Arbeit relevanten, Anwendungsfälle für die Experten, die ihre Bewertungen abgeben, und die Nutzer, die diese einsehen wollen.

sind. Falls Daten in der Datenbank sind, werden sie angezeigt. Enthält die Datenbank keine Daten, wird angezeigt, dass keine Daten gefunden wurden.

2.2 Funktionale Anforderungen

Die Funktionalen Anforderungen an den Datenimport sind in Tabelle 2.1, an das Hochladen von Dateien in Tabelle 2.2 aufgeführt. Neben einer Identifikationsnummer (ID) und einem Stichwort, zeigt eine Zahl zwischen 1 und 5 die Priorität an, wobei 1 für die niedrigste und 5 für die höchste Priorität steht. Die ID besteht aus einem Buchstaben für den Datenimport (I) oder Dateiupload (U), der Abkürzung für Funktionale Anforderung (FA) und einer fortlaufenden Nummer.

I_FA1
Dem Experten steht ein Konsolenbefehl für das Einlesen einer CSV-Datei zur Ver-

ID	Stichwort	Priorität
I_FA1	Konsolenbefehl für CSV	5
I_FA2	Konsolenbefehl für JSON	5
I_FA3	CSV-Datei in korrektes Format für Import übertragen	5
I_FA4	JSON-Datei in korrektes Format für Import übertragen	5
I_FA5	Validierung vor Import	5
I_FA6	In Dezimalzahlen Kommata zu Punkten konvertiert	4
I_FA7	Kein Import bei fehlenden Feldern oder Fehlern in verpflichtenden Werten	5
I_FA8	Import bei korrekten Werten	5

Tabelle 2.1: Funktionale Anforderungen an den Datenimport.

fügung. Eine CSV-Datei mit Bewertungen von Gesundheits-Apps muss dafür dem vorgegebenen Format entsprechen. Das Einlesen in die Datenbank beginnt, falls der Name, der Separator und das *Topic* korrekt angegeben sind.

I_FA2
Dem Experten steht ein Konsolenbefehl für das Einlesen einer JSON-Datei zur Verfügung. Eine JSON-Datei mit Bewertungen von Gesundheits-Apps muss dafür dem vorgegebenen Format entsprechen. Das Einlesen in die Datenbank beginnt, falls der Name und das *Topic* korrekt angegeben sind.

I_FA3
Ein Experte versucht mittels Konsolenbefehl, eine CSV-Datei einzulesen. Es gibt ein Format, über das Daten in die Datenbank importiert werden. Die CSV-Datei wird korrekt in dieses Format übertragen, falls sie den formalen Vorgaben entspricht, ansonsten wird der Importvorgang abgebrochen.

I_FA4
Ein Experte versucht mittels Konsolenbefehl, eine JSON-Datei einzulesen. Es gibt ein Format, über das Daten in die Datenbank importiert werden. Die JSON-Datei wird korrekt in dieses Format, übertragen, falls sie den formalen Vorgaben entspricht und valides JSON ist, ansonsten wird der Importvorgang abgebrochen.

I_FA5
Bevor Daten, die in dem Format für den Import in die Datenbank, vorliegen, in die Datenbank übernommen werden, werden sie vom System validiert. Falls benötigte Informationen fehlen oder fehlerhaft sind, werden *Exceptions* geworfen, die einen

Hinweis auf den aufgetretenen Fehler liefern und den Datenimport abbrechen, ansonsten erfolgt der Datenimport. Falls optionale Daten fehlerhaft sind, werden sie zu *Null* konvertiert, das dem Nutzer der mHAD das Fehlen der Informationen anzeigt, ansonsten verläuft der Datenimport ohne Veränderungen.

I_FA6
Falls in den Daten, die in dem Format für den Import in die Datenbank vorliegen, Komma-getrennte Dezimalzahlen enthalten sind, konvertiert das System die Kommata als Dezimalzahl-Trennzeichen automatisch in Punkte, ansonsten folgt keine Konvertierung von Kommata in Punkte.

I_FA7
Zu importierende Daten werden nicht in die Datenbank geschrieben, falls ein oder mehrere Felder fehlen, oder falls verpflichtende Werte fehlen oder nicht den vorher definierten Vorgaben entsprechen, ansonsten werden die Daten in die Datenbank übertragen.

I_FA8
Falls die Datei, die ein Experte importiert, und alle ihre Daten, den formalen Anforderungen bezüglich des Dateiformats und der benötigten Felder und Werte genügen, werden die Daten erfolgreich in die Datenbank importiert und dem Nutzer der mHAD angezeigt, ansonsten wird der Import abgebrochen.

ID	Stichwort	Priorität
U_FA1	GUI	5
U_FA2	Validierung vor Hochladen der Datei	5
U_FA3	Daten in Tabelle angezeigt	5
U_FA4	Hochgeladene Daten validiert	4
U_FA5	Modifikation der Daten durch Experte	4
U_FA6	Hochgeladene Daten importieren	5

Tabelle 2.2: Funktionale Anforderungen an den Dateiupload.

U_FA1
Dem Experten steht eine graphische Benutzeroberfläche (GUI) zur Verfügung, die die Angabe eines *Topics* und das Hochladen einer CSV-Datei in die mHAD ermöglicht.

U_FA2
Die Eingaben des Experten werden nach dem Absenden vom System validiert

und falls sie korrekt sind, werden sie von dem System weiterverarbeitet, ansonsten stoppt die Verarbeitung und der Experte erhält eine aussagekräftige Fehlermeldung.

U_FA3
Die Daten, die von dem Experten hochgeladen wurden, werden tabellarisch in einer graphischen Benutzeroberfläche angezeigt.

U_FA4
Angezeigte, hochgeladene Daten werden vom System validiert und dem Experten Probleme graphisch angezeigt.

U_FA5
Der Experte hat die Möglichkeit, angezeigte Daten zu modifizieren, wobei eine Validierung durch das System Verbesserungen oder Fehler darstellt.

U_FA6
Hochgeladene, angezeigte Daten können von dem System, sofern sie die formalen Vorgaben erfüllen, in die Datenbank importiert werden.

2.3 Nicht-Funktionale Anforderungen

Die nicht-Funktionalen Anforderungen an den Datenimport sind in Tabelle 2.3, an das Hochladen von Dateien in Tabelle 2.4 aufgeführt. Neben einer ID und einem Stichwort, zeigt eine Zahl zwischen 1 und 5 die Priorität an, wobei 1 für die niedrigste und 5 für die höchste Priorität steht. Die ID besteht aus einem Buchstaben für den Datenimport (I) oder Dateiupload (U), der Abkürzung für Qualitätsansforderung (QA), die Art der nicht-Funktionalen Anforderung, und einer fortlaufenden Nummer.

ID	Stichwort	Priorität
I_QA1	Datenimport erweiterbar	5
I_QA2	Minimum an Codeduplikation	4
I_QA3	Wartbarkeit	4
I_QA4	Unit- und System-Tests	5

Tabelle 2.3: Qualitätsanforderungen an den Datenimport.

I_QA1
Der Datenimport muss um neue Datei- und Datenformate erweiterbar sein, so dass pro neuem Datei- oder Datenformat nur bis maximal 500 neue *Lines of Code* (LoC) hinzukommen.

I_QA2
Der Datenimport muss ein Minimum an Codeduplikation aufweisen, das maximal 200 LoC beträgt.

I_QA3
Der Datenimport muss leicht wartbar sein, indem verwendete Daten an klar ausgewiesenen Orten gebündelt sind, Klassen-, Methoden- und Variablennamen aussagekräftig sind und jede Methode kommentiert ist.

I_QA4
Die Methoden, die der Datenimport verwendet, sollen mittels Unit- und System-Tests erfolgreich getestet sein.

ID	Stichwort	Priorität
U_QA1	GUI einfach und intuitiv	4
U_QA2	Darstellung von Fehlern	4

Tabelle 2.4: Qualitätsanforderungen an den Dateiupload.

U_QA1
Die graphische Benutzeroberfläche für das Hochladen von Dateien, muss für einen Laien einfach und intuitiv bedienbar sein.

U_QA2
Die graphische Benutzeroberfläche, die die Daten tabellarisch anzeigt, muss für Laien einfach bedienbar und die Darstellung von Fehlern farblich eindeutig und barrierefrei sein.

3 Theoretische Vorüberlegungen

Zuerst wird überlegt, welche Daten in die Datenbank aufgenommen werden. Daraus bestimmt sich, welche Felder zu Spaltennamen werden. Die verwendeten Spaltennamen basieren auf der Mobile Anwendungen Rating Skala (MARS) [27] .
Es wird zwischen verpflichtenden und optionalen Feldern unterschieden. Verpflichtende Felder müssen Werte enthalten, die dargestellt werden können. Sie sind in Tabelle 3.1 dargestellt. Optionale Felder dürfen leer sein oder nicht darstellbare Werte, die als leer interpretiert werden, enthalten. Sie sind in Tabelle 3.2 zu sehen.

Name	Beschreibung
`plattform`	Plattform (iOS, Android, Web) auf der App verfügbar ist.
`name`	Name der App.
`developer`	Name des App-Entwicklers.
`version`	Versionsnummer der App.
`samean`	Dezimalzahl $\in$ [1, 5]. MARS Sektion A “Engagement”.
`sbmean`	Dezimalzahl $\in$ [1, 5]. MARS Sektion B “Funktionalität”.
`scmean`	Dezimalzahl $\in$ [1, 5]. MARS Sektion C “Ästhetik”.
`sdmean`	Dezimalzahl $\in$ [1, 5]. MARS Sektion D “Information”.
`overallmean`	Dezimalzahl $\in$ [1, 5]. Arithmetisches Mittel aus den Werten `samean`, `sbmean`, `scmean` und `sdmean`.

Tabelle 3.1: Die verpflichtenden Spaltennamen mit Angabe des Typs und einer kurzen Beschreibung.

Welche Felder benötigt werden, wurde an zwei Kriterien festgemacht:

1. Die evaluierte App muss eindeutig identifizierbar sein. Daher sind Name, Version, Plattform und Entwickler (auch aus rechtlichen Gesichtspunkten) verpflichtend.
2. Die Bewertung durch Experten, die vollständig sein muss. Sie bildet den Kern der mHAD. Deswegen sind die Bewertungen (`samean`, `sbmean`, `scmean`, `sdmean` und `overallmean`) verpflichtend.

Name	Beschreibung
searchstring	Suchbegriff, der zu App führt.
searchdate	Datum der Suche nach App.
userrating	Dezimalzahl $\in$ [0, 5]. Die Bewertung durch Nutzer auf der Plattform.
numberratings	Anzahl der abgegebenen Nutzerbewertungen auf der Plattform.
category	Kategorie zur Klassifikation der App.
price	Positive Dezimalzahl. Preis in Euro.
link	Link zu der App (URL).
inc_ger	Boolescher Wert. Zeigt an, ob App auf Deutsch erhältlich ist.
inc_en	Boolescher Wert. Zeigt an, ob App auf Englisch erhältlich ist.
focus_app_targets	Ziele aus vorgegebener Liste.
focus_app_targets_other	Ziele aus Freitextfeld.
theoretical_background	Theoretische Hintergründe bei Interventionen aus vorgegebener Liste.
theoretical_background_other	Theoretische Hintergründe aus Freitextfeld.
affiliation	Angliederung der App, z. B. an Organisation oder Universität.
age_group	Altersgruppen für die die App entwickelt wurde oder angemessen ist.
technical_aspects	Technische Aspekte, die zusätzlich angeboten oder für den Gebrauch benötigt werden.

Tabelle 3.2: Die optionalen Spaltennamen mit Angabe des Typs und einer kurzen Beschreibung.

Für die Felder *focus_app_targets*, *theoretical_background*, *affiliation*, *age_group* und *technical_aspects* sind nach MARS Werte vorgegeben. Es wurden dafür kurze und eindeutige Schlüssel erstellt, die in die entsprechenden Tabellenspalten einzufügen sind. Sie sparen Platz und sind sprachunabhängig. Die dazugehörigen Tabellen sind als Anhang A beigefügt.

4 Architektur

Die Implementierung der beiden Meilensteine, Datenimport und Dateiupload, wird erläutert. Das umfasst auch die bereits vorgefundene Implementierung, sofern vorhanden.

4.1 Datenimport

Dateien werden eingelesen und die Daten, nach vorhergehender Validierung, direkt in die Datenbank übertragen. Den Nutzern stehen dafür Konsolenbefehle zur Verfügung.

4.1.1 Ausgangslage

Es gibt eine abstrakte Klasse *AbstractImportService*, die Methoden für den Import bereitstellt. Dabei wird ein *pending import* erstellt, ein vorläufiger Import, der erfolgreich durchgeführt wird, wenn die Daten den Vorgaben in der Datenbank entsprechen. Für jeden Datenimport erweitert der jeweilige *ImportService* diese Klasse und implementiert den kompletten Import für das jeweilige Format. Der Konsolenbefehl wird in der *ImportCommand*-Klasse definiert und ruft den *ImportService* auf. Daten aus einer Datei, die sich in dem *imports*-Ordner befindet, können so in die Datenbank übertragen werden. Der Ablauf ist in Abbildung 4.1 dargestellt, einem UML-Klassendiagramm zu dem vorgefundenen CSV-Import.
Es gibt keine Validierung. Der Import wird nur dann zurückgewiesen, wenn die Vorgaben der Datenbank nicht eingehalten werden. Dies geschieht ohne Fehlermeldung.

Wenn der Import um ein weiteres Datenformat erweitert werden soll, muss der komplette Import dafür erstellt werden, mit Ausnahme der durch *AbstractImportService* geerbten Methoden. Müssen Daten gesondert modifiziert werden, müssen unter Umständen auch die geerbten Methoden stark angepasst werden.

4.1.2 Neuimplementierung

In Abbildung 4.2 ist die Neuimplementierung des Datenimports dargestellt. Sie zeigt die Hauptkomponenten und ihre Abhängigkeiten. Dateien werden über einen Konsolenbefehl eingelesen und dann in *Jobs* konvertiert, die von der Datenimport-Komponente weiterverarbeitet werden können. Bei erfolgreicher Validierung werden die Daten in die Datenbank übernommen.

Für jedes neu zu importierende Datei- oder Datenformat, werden nur eine neue *Sender*-Klasse und ein Konsolenbefehl benötigt. Die neue Implementierung ist in Abbildung 4.3 als UML-Klassendiagramm dargestellt. Darin ist, neben dem CSV-Import, auch der JSON-Import zu sehen.

Die *Sender*-Klasse erbt einige grundlegende Methoden von *AbstractFileSender* und hat drei Aufgaben:

1. Einlesen der Datei
2. Umwandlung der Daten in ein assoziatives Array
3. Versenden des Datenpakets an eine Warteschlange

Das Datenpaket *ImportAssocArray* gehört zu der *Job*-Klasse. Instanzen dieser Art Klasse können in Warteschlangen eingespeist werden [21].
Aus der Warteschlange wird das Datenpaket an den *AssocArrayImportService* weitergegeben. In dieser Klasse werden erst verschiedene Methoden zur Datenumwandlung und -validierung aufgerufen. Falls die Daten den Vorgaben entsprechen, werden sie in die Datenbank importiert. Falls nicht werden individualisierte Ausnahmen (*Exceptions*) geworfen und der Import abgebrochen. In Abbildung 4.4 ist dargestellt, welche Datenkonverter und -validatoren dabei aufgerufen werden.

Methoden zur Datenumwandlung ersetzen in Dezimalzahlen als Dezimaltrennzeichen Kommata durch Punkte, denn PHP erkennt nur Ganze oder Punkt-getrennte

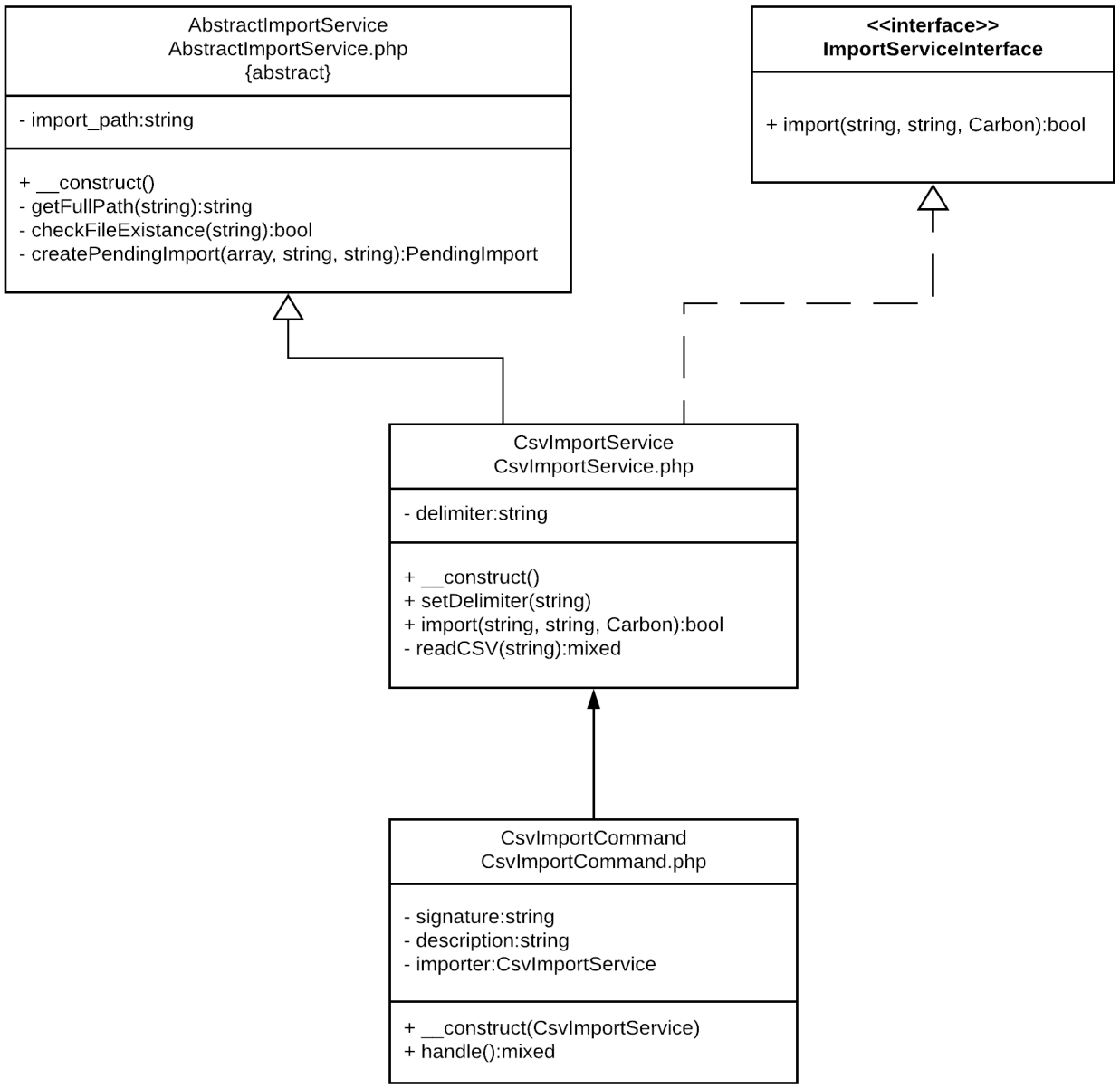

Abbildung 4.1: UML-Klassendiagramm des ursprünglichen Dateiimports. Sobald der Konsolenbefehl eingegeben wird, ruft CsvImportCommand den CsvImportService auf, der AbstractImportService erweitert und das ImportServiceInterface implementiert.

Zahlen. Optionale Werte, die nicht den Anforderungen genügen, werden zu *Null* modifiziert. In Abbildung 4.5 ist zu sehen, welche Klassen der *ImportedDataCon-*

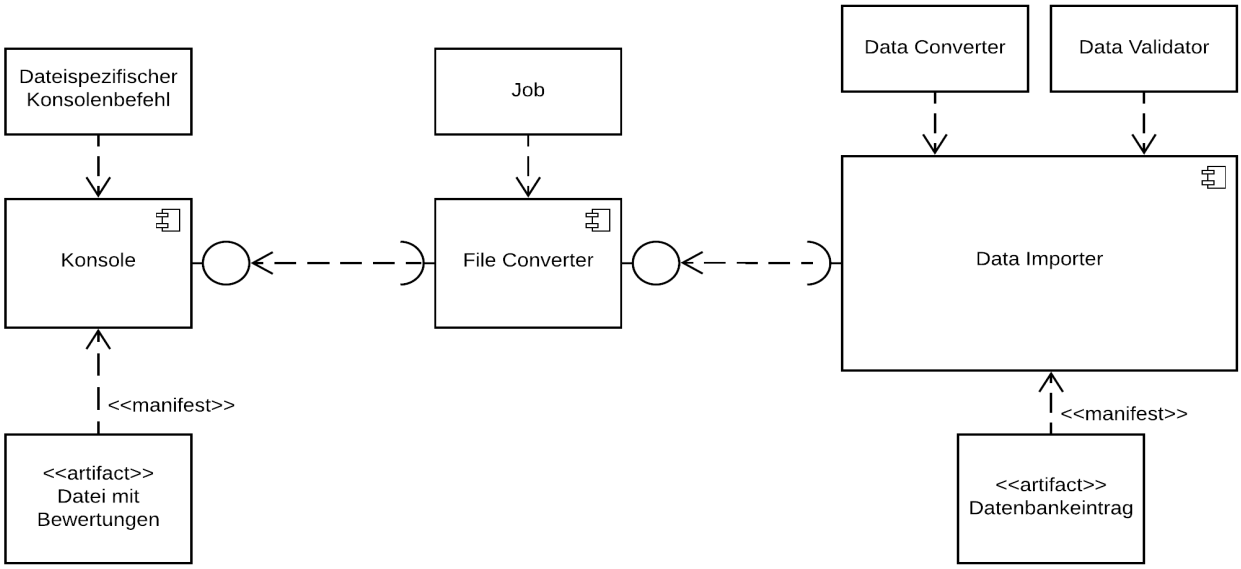

Abbildung 4.2: UML-Komponentendiagramm, das die Neuimplementierung des Datenimports anschaulich darstellt. Eine Datei wird über die Konsole eingelesen und zur Verarbeitung an die Datenimportkomponente weitergeschickt, wo Daten validiert und modifiziert werden. Sind die Daten korrekt, werden sie in die Datenbank eingelesen.

verter aufruft. Die Klasse *ImportedDataConverter* überprüft die optionalen Werte auf ihre formale Richtigkeit hin. Sind die Daten nicht interpretierbar, werden sie zu *Null*. Für die Validierung werden Klassen aus der Datei *GenericDataValidators.php* genutzt, wie *NumberValidator*, *DateValidator* und *UrlValidator*. Diese enthalten kleine, auf den Laravel-eigenen Validierungsmethoden beruhende Methoden, um Zahlen, Datumsangaben und die formale Korrektheit übermittelter URLs zu überprüfen [22].

Validatoren prüfen, ob benötigte Felder vorhanden und korrekt ausgefüllt sind und ob optionale Felder vorhanden sind. Falls nicht, werden spezifische Ausnahmen geworfen:

- *OptionalFieldNotPresentException*: Optionale Felder können leer sein oder falsche Werte enthalten, müssen aber vorhanden sein.
- *RequiredValueInvalidException*: Verpflichtende Felder müssen vorhanden sein und die enthaltenen Werte den Vorgaben entsprechen.

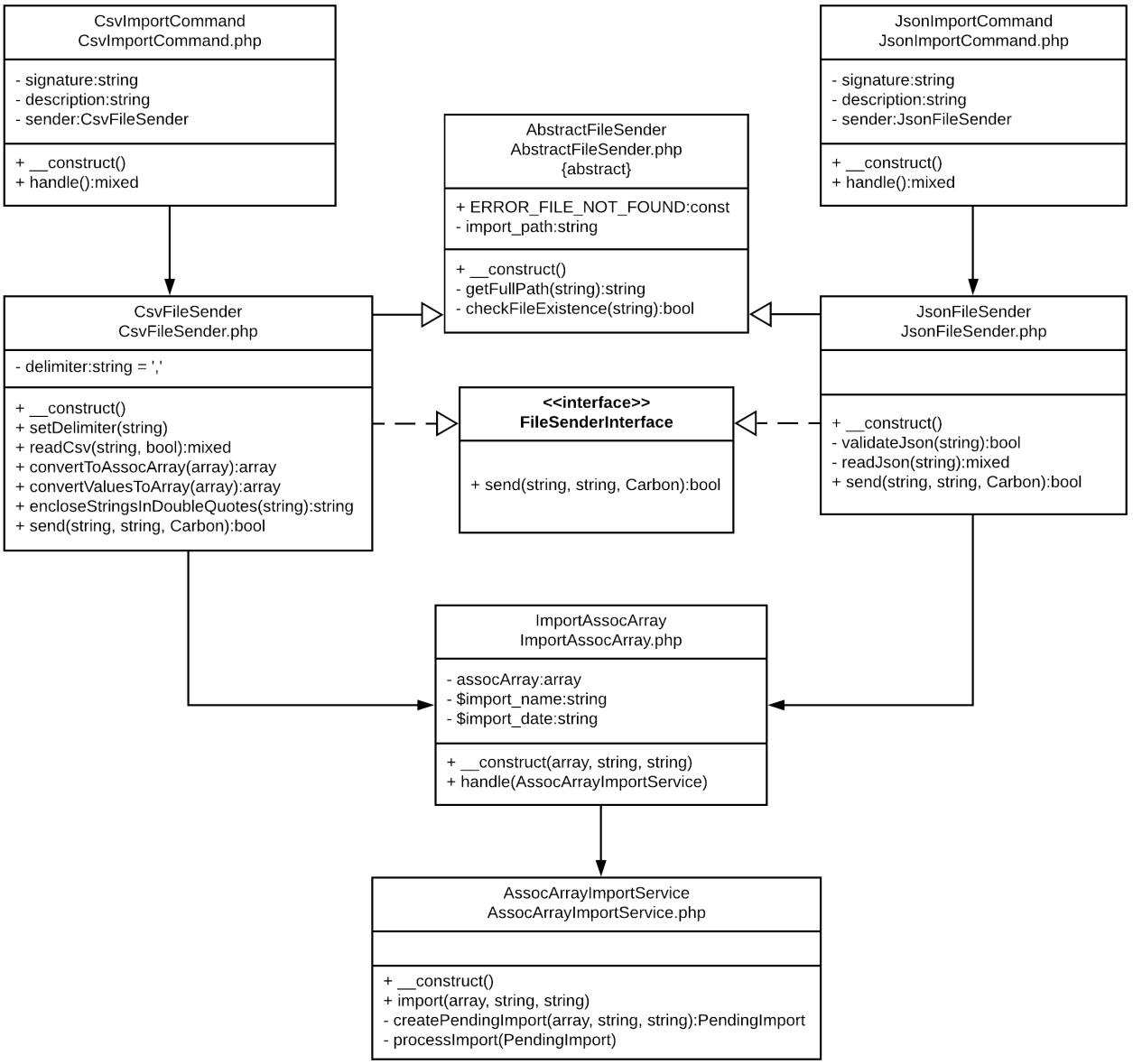

Abbildung 4.3: UML-Klassendiagramm des neuen Dateiimports. Wenn der Konsolenbefehl eingegeben wird, ruft ImportCommand den FileSender auf, der AbstractFileSender erweitert und das FileSenderInterface implementiert. Der FileSender verschickt den Job ImportAssocArray an den AssocArrayImportService.

- *RequiredValueIsNullException*: Verpflichtende Felder dürfen nicht leer sein.

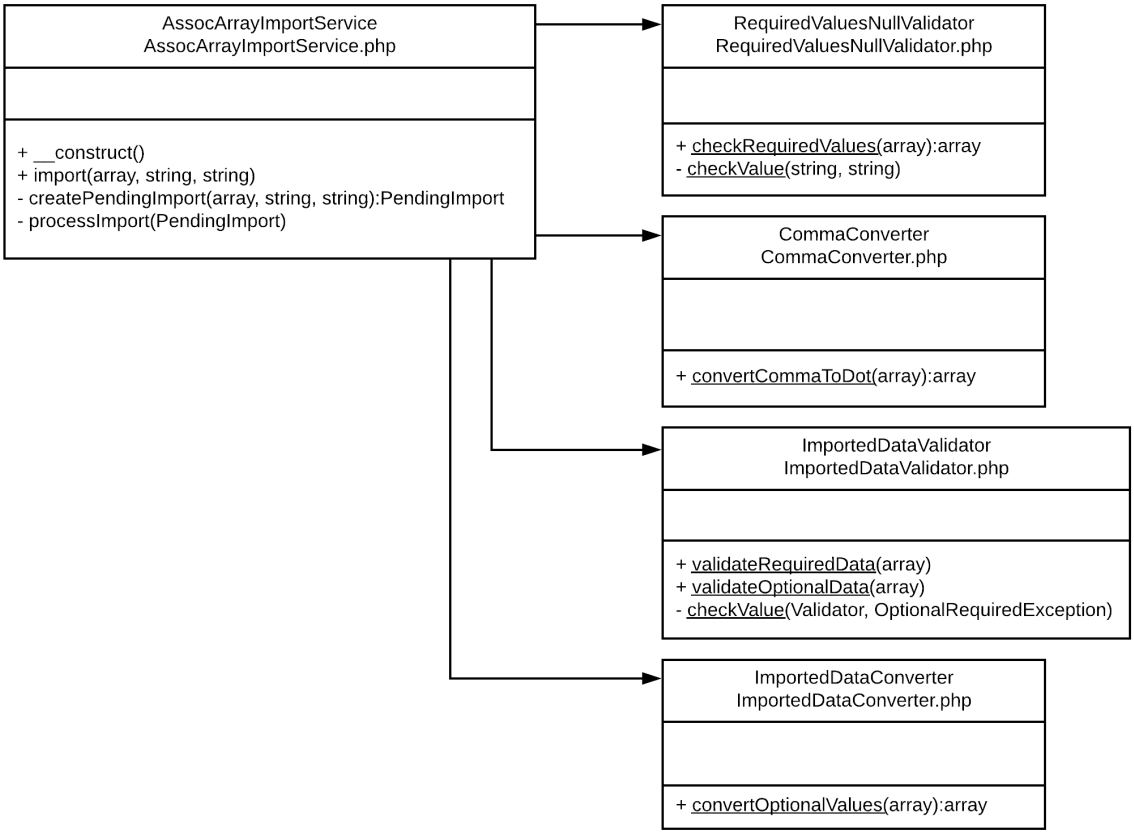

Abbildung 4.4: UML-Klassendiagramm, das den Aufruf von Datenkonvertern und -validatoren vor dem Import neuer Daten in die Datenbank zeigt.

Diese Ausnahmen erweitern die Ausnahme *OptionalRequiredException*, damit der Typ leicht geprüft werden kann.
Im Rahmen dieser Arbeit wurden der Datenimport aus CSV- und JSON-Dateien implementiert. Die Klassen heißen *CsvFileSender* und *JsonFileSender*.
Als Datenstruktur für den Import wurde das *Assoziative Array* bestimmt [1]. Alle Daten, ob z. B. JSON oder CSV, werden darin überführt und danach in die Datenbank importiert. Dieses Array hat die, in Listing 4.1 zu sehende Form.

Listing 4.1: Assoziatives Array in PHP

```
1 Array {
2     Key => Value
3 }
```

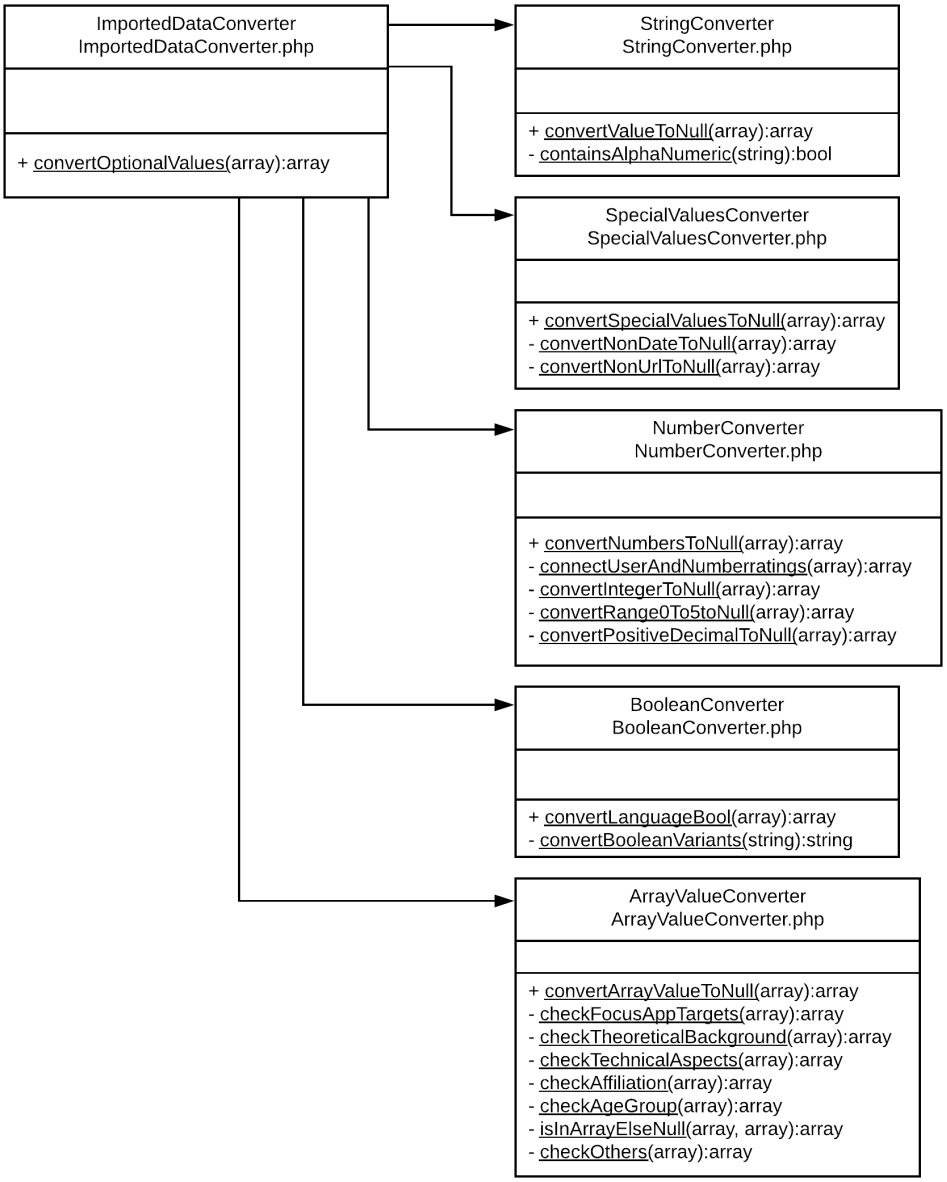

Abbildung 4.5: UML-Klassendiagramm, das den Aufruf von Datenkonvertern durch den ImportedDataConverter zeigt.

Der Schlüssel kann frei gewählt werden, ansonsten beginnt er bei Wert 0. Er dient dazu, auf die Werte zu verweisen. Vorteile sind, dass es eine PHP-eigene Datenstruktur ist, für die viele Methoden existieren. Alle Daten lassen sich einfach darin übertragen, denn alle Werte werden akzeptiert. Man muss die Daten entsprechend

umwandeln, eine weitere Modifikation ist nicht nötig. Nähme man z. B. JSON, so müssten Daten unter Umständen vorher aufbereitet werden, denn JSON akzeptiert nur Ganze oder Punkt-getrennte Zahlen. Deutsche CSV-Dateien enthalten oft Komma-getrennte Dezimalzahlen. Mit dem *Assoziativen Array* kann man die Datenmodifikation dagegen an einem Ort bündeln.
Die Feldnamen (siehe Tabellen 3.1 und 3.2) werden zu den Schlüsseln und ihre jeweiligen Werte zu den Werten des *Assoziativen Arrays*. Für jede App wird ein eigenes *Assoziatives Array* erstellt und diese(s) wiederum in ein übergeordnetes Array eingebettet. Jede Dateneingabe ergibt ein Array aus *Assoziativen Arrays*.

4.2 Dateiupload

Die Möglichkeit, Dateien hochzuladen und anzuzeigen bestand in früheren Versionen der mHAD nicht. Diese Funktionalität wurde neu implementiert.
In Abbildung 4.6 ist der zeitliche Ablauf der Interaktion zwischen Experte und mHAD während des Dateiuploads dargestellt. Zuerst wird die Eingabe des Experten geprüft und nur falls sie korrekt ist, werden die Daten angezeigt. In der Anzeige werden die Daten selbst validiert und eventuelle Fehler angezeigt. Die Fehler in den Daten kann der Experte verbessern.

Auf der Seite *insert* der mHAD, befindet sich eine Eingabemaske für den Nutzer. Sie enthält zwei Felder

- *Topic*: Klassifikation der Gesundheits-Apps.
- *File*: Pfad der hochzuladenden Datei auf der Festplatte. Erwartet wird eine CSV-Datei.

Dazu gibt es einen Knopf, um die Daten abzusenden. Die Eingabemaske ist in Abbildung 4.7 zu sehen.

Bei fehlerhaften Eingaben erfolgt kein Hochladen der Datei, sondern die Fehler werden dem Nutzer auf der Seite angezeigt. Das Scheitern des Hochladevorgangs ist in Abbildung 4.8 dargestellt.

Das *Topic* muss den vorgegebenen Werten entsprechen. Diese dienen später der Klassifikation der Gesundheits-Apps, um die Suchfunktionalität zu verbessern. Der

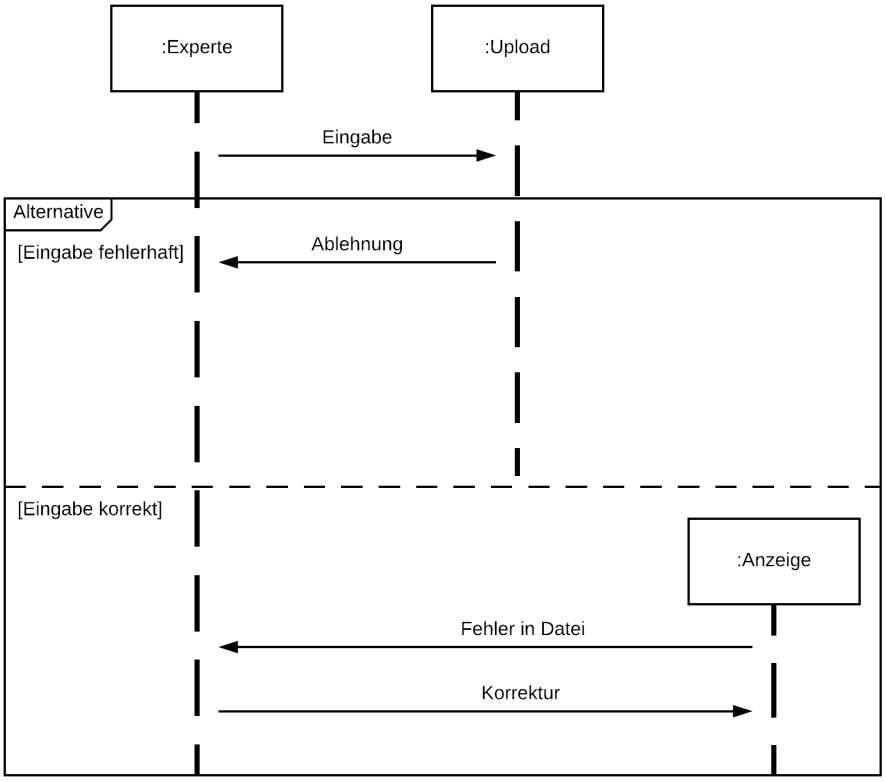

Abbildung 4.6: Ablauf des Hochlade- und Korrekturvorgangs des Dateiuploads.

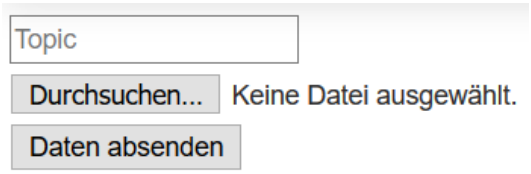

Abbildung 4.7: Eingabemaske für den Dateiupload. Ein Textfeld für das *Topic* mit Platzhalter und die Möglichkeit eine Datei auszuwählen. Wenn der Experte fertig ist, können die Daten abgesendet werden.

Nutzer kann die Gesundheits-Apps so filtern, dass z. B. nur die zum Thema “Angst” angezeigt werden.
Die abgesendeten Daten werden als *Request* weitergeleitet [23]. Das wurde zu einem Typ der eigens erstellten Klasse *FileToUpload* angepasst. In dieser erfolgt die Validierung der Eingabedaten, vor Weiterverarbeitung der Nutzereingaben. Sowohl das *Topic*, als auch eine Datei werden benötigt. Das *Topic* muss den vorgegebenen

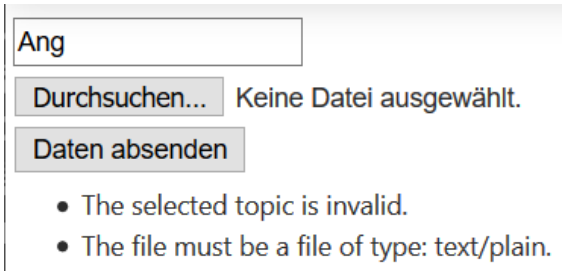

Abbildung 4.8: Bei dem Dateiupload sind zwei Fehler aufgetreten: Das *Topic* war nicht korrekt und die angegebene Datei (nicht mehr angezeigt), war keine Textdatei. Die Fehlermeldungen unter dem Absendeknopf teilen dies dem Experten mit.

Werten entsprechen. Die Datei muss eine nicht leere Text-Datei sein. Intern wird die CSV-Datei als Text-Datei interpretiert, weshalb eine genauere Überprüfung des Typs nicht möglich ist. Durch die Auslagerung der Validierung in die neue Klasse ist der Code übersichtlicher und einfacher zu warten.
Die Weiterverarbeitung erfolgt in der Klasse *FileUploadController*. Bei Aufruf ihrer Methode *index()*, wird die Seite *insert* angezeigt. Bei Dateneingabe durch den Nutzer, wird dagegen die Methode *store(FileToUpload)* aufgerufen. Noch vor ihrem Aufruf erfolgt die Validierung der Eingaben. Falls die Eingaben korrekt sind, wird das *Topic* in einer Variablen gespeichert und die Datei in den Ordner *uploads* geladen. Auch der Pfad zu der hochgeladenen Datei wird in einer Variablen gespeichert.
Für die Weiterverarbeitung einer CSV-Datei muss der verwendete Separator bekannt sein. Damit die mHAD auch für Laien einfach zu benutzen ist, wird dieser automatisch erkannt. Das erfolgt in der Klasse *DelimiterDetector*. Die erste Spalte einer validen CSV-Datei muss die Spaltennamen enthalten, die selbst durch den verwendeten Separator getrennt sind. Die Spaltennamen, siehe Tabellen 3.1 und 3.2, sind so gewählt, dass sie keine der üblicherweise verwendeten Separatoren enthalten. Die erste Zeile der CSV-Datei wird eingelesen und als *String* zwischengespeichert. Es wird durch eine Liste typischer Separatoren (Komma, Semikolon, Doppelpunkt, Tabulatorzeichen, senkrechter Strich und Zirkumflex) gegangen und der *String* nach diesen getrennt. Die so getrennten *Substrings* werden in einem Array abgespeichert. Das größte Array zeigt an, welches Zeichen als Separator verwendet wird. Der Algorithmus folgt dabei der Bestimmung des Maximums.
Nach Bestimmung des Separators wird mittels der Klasse *UploadedFileValidator* zusätzlich überprüft, ob die erhaltenen *Substrings* den vorgegebenen Spaltenna-

men entsprechen. Wenn das der Fall ist, erfolgt die Umwandlung der Daten in JSON, damit sie einfach in JavaScript-Code verwendet werden können.
Anschließend wird die Seite *show* aufgerufen, der das *Topic* und die JSON-Variable als Parameter mitgegeben werden. Die Daten werden dem Nutzer tabellarisch dargestellt. Die Felder können validiert und von Nutzern modifiziert werden. Über den Knopf *import* wird eine JSON-Datei in die Datenbank importiert. Dabei wird mittels *Asynchronous JavaScript and XML* (Ajax) die Methode *post(Request)* aus der Klasse *ImportUploadedFileController* aufgerufen, in der die JSON-Datei aus dem *uploads*- in das *imports*-Verzeichnis verschoben und dann der JSON-Importbefehl aufgerufen wird.

5 Ausgewählte Implementierungsaspekte

Zuerst werden die verwendeten Technologien vorgestellt. Anschließend ausgewählte Aspekte der Implementierung beider Meilensteine.

5.1 Technologien

Apache HTTP Server
Ein Webserver stellt Daten bereit, die ein Client über eine URL anfragen kann. Die Antwort auf eine Anfrage enthält einen Statuscode, der angibt, ob die Anfrage erfolgreich war. Falls sie erfolgreich war, enthält die Antwort auch einen *body* mit den entsprechenden Informationen [15].

CSV-Dateiformat
Das CSV-Dateiformat erlaubt den Austausch tabellarischer Daten [26] [18]. Jede Zeile einer Textdatei entspricht einer Zeile der Tabelle. Die Einträge der ersten Zeile können dabei den Spaltennamen entsprechen. Die Spalten werden durch Separatoren voneinander unterschieden, meist dem namensgebenden Komma. Falls kein Komma verwendet wird, bezeichnet man die Dateien auch als DSV. CSV-Dateien können aus Microsoft Excel Arbeitsmappen generiert werden. In der deutschen Version ist das Semikolon der Separator.

Laravel
Laravel ist ein PHP-Framework, das auf dem Model-View-Controller (MVC) Design Pattern basiert [10]. *Models* sind darin Ressourcen, wie Datenbankeinträge [10]. Als *Controller* fungieren *Routes* und *Controller*, die Anfragen empfangen und passende Antworten zurückliefern [10]. Die Antworten werden in den *Views* angezeigt

[10].

MariaDB
MariaDB ist eine relationale Datenbank, die Datenzugriff über SQL erlaubt. Sie ist eine *open source* Abspaltung von MySQL [14].

PHP
PHP ist eine serverseitige Skriptsprache, die in HTML eingebettet werden kann [2]. Sie ist schwach typisiert [3] und seit Version 5 objektorientiert [4].

TOAST UI
TOAST UI Grid [13] ist eine *library* der NHN Entertainment Corporation [12]. Sie steht unter der MIT Lizenz [28]. Die *library* ist in JavaScript geschrieben und stellt Daten tabellarisch dar.

Virtuelle Maschine
Als Virtuelle Maschine dient Laravel Homestead [24], das auf Vagrant [17] basiert, und VirtualBox [20].

Visual Studio Code
Der verwendete Editor ist Visual Studio Code für PHP von Microsoft [19].

5.2 Konvertierung eines falschen optionalen Werts

Optionale Felder müssen vorhanden sein, dürfen aber falsche oder keine Werte enthalten. Nutzern der mHAD können entweder korrekte Werte oder *Null* angezeigt werden.
Als Beispiel für eine Methode zur Konvertierung falscher optionaler Wert dient *convertIntegerToNull(array)* aus der Klasse *NumberConverter*. Sie ist in Listing 5.1 dargestellt (Anmerkung: Der Code wurde gegenüber der Implementierung stilistisch, aber nicht funktional, verändert, damit die Formatierung erhalten bleibt.).

Listing 5.1: Methode convertIntegerToNull(array) in der Klasse NumberConverter

```
1  protected static function convertIntegerToNull($data)
2  {
3      $integerValues = ColumnNameArrays::$integerColumns;
4      foreach ($integerValues as $key => $val) {
5          $isPositiveInteger =
6          NumberValidator::isPositiveInteger($data[$val]);
7          if (!($isPositiveInteger)) {
8              $data[$val] = null;
9          }
10     }
11
12     return $data;
13 }
```

Die Funktion erhält einen kompletten Gesundheits-App-Array als Parameter, aber hier wurde die Typisierung (`convertIntegerToNull(array $data)`) entfernt, da die Zeile sonst zu lang ist (Listing 5.1, Zeile 1). Die Einträge des Arrays mit den Namen aller Felder, deren Werte Ganze Zahlen sind, werden in die Variable `$integerValues` übernommen (Listing 5.1, Zeile 3).
In Listing 5.1, Zeile 4 beginnt eine foreach-Schleife [5], die durch `$integerValues` geht und ein Validator prüft, ob der jeweilige Wert aus der Gesundheits-App-Bewertung an dieser Stelle eine positive Ganze Zahl ist (Listing 5.1, Zeilen 5 und 6). Falls das nicht der Fall ist, wird der Wert auf *Null* gesetzt (Listing 5.1, Zeilen 7 bis 9).
Das gesamte, eventuell modifizierte, Gesundheits-App-Array ist der Rückgabewert (Listing 5.1, Zeile 12).

5.3 Validierung von Werten

Laravel bietet Validatoren und eine Vielzahl an Validierungsmethoden an [22]. Diese Methoden werden genutzt, um Daten zu validieren, weil sie mächtig und gut getestet sind. Das wird in Listing 5.2 am Beispiel der Methode *isPositiveInteger(value)* gezeigt.

Listing 5.2: Methode isPositiveInteger(value) in der Klasse NumberValidator

```
1  public static function isPositiveInteger($value)
2  {
3      $array['integer'] = $value;
4      $validator = Validator::make($array, [
5          'integer' => ['integer', 'min:0'],
6      ]);
7      if ($validator->fails()) {
8          return false;
9      } else {
10         return true;
11     }
12 }
```

Ein *Validator*-Objekt bietet die Validierungsmethoden an, die in einem assoziativen Array die Werte zu einem Schlüssel validieren.
Der Methode wird ein nicht-typisierter Wert übergeben (Listing 5.2, Zeile 1). Weil übergebene Werte nicht als assoziative Arrays vorliegen, wird der Wert zu dem Wert eines Schlüssels in einem assoziativen Array (Listing 5.2, Zeile 3). In Listing 5.2, Zeilen 4 bis 6 wird einem neuen *Validator*-Objekt die Validierungsregel für den Schlüssel gegeben. Diese Methode wird beispielsweise für Felder wie *numberratings* (die Anzahl der, von Nutzern abgegebenen, Bewertungen) verwendet, weshalb es eine Ganze Zahl größer Null sein muss.
In einer if-Abfrage in Listing 5.2, Zeile 7 bis 10, wird überprüft, ob der *Validator* wahr oder falsch zurückgibt, und das ist der Rückgabewert der Methode.

5.4 Umwandlung von CSV in ein Assoziatives Array

In der Klasse *CsvFileSender* wird eine CSV-Datei eingelesen, in ein assoziatives Array umgewandelt und dann als *Job* an die Datenimportklasse *AssocArrayImportService* versendet.
In einer CSV-Datei liegen die Daten nicht als Array vor. Die erste Zeile besteht aus den Spaltennamen, die zu den Schlüsseln des assoziativen Arrays werden. Jede weitere Zeile enthält die Werte jeweils einer Gesundheits-App. Bestimmte Wer-

te (*focus_app_targets*, *focus_app_targets_other*, *theoretical_background*, *theoretical_background_other*, *affiliation*, *technical_aspects* und *age_group*) sind selbst wiederum Arrays mit Werten. Jeder Array für eine Gesundheits-App ist ein 2D-Array. Alle Arrays für Gesundheits-Apps sind selbst wiederum in ein Array eingebettet. Eine CSV-Datei muss daher in ein 3D-Array umgewandelt werden, damit die Daten in die Datenbank importiert werden können.
Die Methode *convertToAssocArray(array)* in *CsvFileSender* wandelt Daten aus einer CSV-Datei in ein assoziatives Array um. Sie ist in Listing 5.3 dargestellt.

Listing 5.3: Methode convertToAssocArray(array) in der Klasse CsvFileSender

```
1  public function convertToAssocArray($array)
2  {
3      $delimiter = $this->delimiter;
4
5      $appAssocArray = array();
6
7      foreach ($array as $key => $value) {
8          $key =
9              $this->encloseStringsInDoubleQuotes($key);
10         $value =
11             $this->encloseStringsInDoubleQuotes($value);
12         $columns = str_getcsv($key, $delimiter, '"');
13         $row = str_getcsv($value, $delimiter, '"');
14         foreach ($columns as $column => $content) {
15             $appAssocArray[$content] = $row[$column];
16         }
17     }
18
19     $appAssocArray =
20         $this->convertValuesToArray($appAssocArray);
21
22     return $appAssocArray;
23 }
```

Als Parameter werden die Daten aus dem eingelesenen CSV übergeben (Listing 5.3, Zeile 1). Als Separator wird der Separator des *CsvFileSender* Objekts festgelegt (Listing 5.3, Zeile 3). In Listing 5.3, Zeile 5 wird ein neues Array deklariert, das

die Daten einer Gesundheits-App umfasst. Anschließend werden die, als Parameter übergebenen, Werte in eine foreach-Schleife [5] als Schlüssel und Wert eingelesen (Listing 5.3, Zeile 7).
In Listing 5.3, Zeile 8 bis 11, wird eine Hilfsmethode *encloseStringsInDoubleQuotes(string)* aufgerufen, die die Schlüssel und Werte in Anführungszeichen setzt. Die Anführungszeichen sind in den übergebenen Daten nicht mehr enthalten. In Listing 5.3, Zeile 12 wird eine Variable mit den Spaltennamen gefüllt, die zu den Schlüsseln des neuen assoziativen Arrays werden. In Listing 5.3, Zeile 13 werden die Werte der Gesundheits-Apps eingelesen. Beide Male wird die Methode *str_getcsv* [8] aufgerufen, die die Werte nach dem Separator auftrennt. Wenn *encloseStrings-InDoubleQuotes(string)* nicht aufgerufen wird, somit die Anführungszeichen fehlen und die Werte in sich wiederum den Separator enthalten (z. B. Komma ist der Separator, ein Wert ist eine Komma-getrennte Dezimalzahl oder eine Komma-getrennte Aufzählung von *Strings*), werden die Werte in sich wieder aufgespalten.
In Listing 5.3, Zeile 14 wird eine erneute foreach-Schleife gestartet, die in dem Array die Werte zu den Werten der Schlüssel macht. Die Variable `$row[$column]` enthält den Wert (`$row`) zu dem jeweiligen Schlüssel (`$column`) und füllt den, noch leeren, Eintrag des assoziativen Arrays. Dabei wird der Feldname der Schlüssel zu dem Wert.
In Listing 5.3, Zeilen 19 und 20 wird eine Hilfsmethode *convertValuesToArray(array)* auf das Array aufgerufen, die Werte, die selber wiederum Arrays sind, in Arrays umwandelt. So entsteht ein 2D-Array mit den Daten einer Gesundheits-App. Dieses Array ist der Rückgabewert (Listing 5.3, Zeile 22).
In der Methode, die *convertToAssocArray(array)* aufruft, wird dann ein 3D-Array aus all den einzelnen 2D-Arrays aufgebaut und als *Job* versandt.

5.5 Automatische Erkennung des Separators

Der Separator soll automatisch erkannt werden, damit der Dateneintrag auch für Laien einfach ist. Für CSV-Dateien wird die Struktur vorgegeben, dass die erste Zeile die korrekten Spaltennamen enthält. Diese Feldnamen selbst enthalten keine Sonderzeichen, außer den Unterstrich. Das wird für die automatische Erkennung des Separators in der Klasse *DelimiterDetector* ausgenutzt. Nur die erste Zeile wird untersucht, wodurch vermieden wird, dass Separatoren innerhalb von Werten das

Ergebnis verfälschen.
Die Methode *calculateDelimiter(string)* ermittelt den Separator. Sie ist in Listing 5.4 dargestellt.

Listing 5.4: Methode calculateDelimiter(string) in der Klasse DelimiterDetector

```
1  public static function calculateDelimiter($row)
2  {
3      $delimiters = UploadImportArrays::$commonDelimiters;
4
5      $delimiter = $delimiters['comma'];
6      foreach ($delimiters as $del) {
7          if (!(sizeof(explode($delimiter, $row)) >=
8              sizeof(explode($del, $row)))) {
9          $delimiter = $del;
10         }
11     }
12
13     return $delimiter;
14 }
```

Als Parameter wird ein *String* übergeben (Listing 5.4, Zeile 1), die erste Zeile der CSV-Datei. In Listing 5.4, Zeile 3 wird ein Array mit häufig verwendeten Separatoren befüllt. Diese sind in einer separaten Datei an einem zentralen Ort gespeichert, damit sie einfach modifiziert werden können. Eine Variable wird mit dem Komma als Standard-Separator initialisiert (Listing 5.4, Zeile 5).
Ab Listing 5.4, Zeile 6 geht eine foreach-Schleife durch die Separatoren und wendet eine leichte Modifikation des Algorithmus zum Auffinden eines Maximums in einem Array an [16]. In Listing 5.4, Zeilen 7 und 8 wird die Methode *explode* [6] verwendet, die einen übergebenen *String* mittels eines Separators aufspaltet und in ein Array überführt. Als Separator wird einmal der Standard-Separator `$delimiter` und einmal der jeweilige Separator aus dem Array (`$del`) verwendet. Unterschiedliche Separatoren führen zu unterschiedlich großen Arrays. Der tatsächlich verwendete Separator zu dem größten Array. Zur Bestimmung der Arraygröße wird *sizeof* [7] verwendet, weil der Name besser beschreibt, was in der if-Abfrage miteinander verglichen wird. Wenn `$delimiter` nicht größer oder gleich dem Separator aus der Liste (`$del`), wird er auf den neuen Separator gesetzt (Listing 5.4, Zeile 9).

Der Rückgabewert der Methode ist `$delimiter` (Listing 5.4, Zeile 13).

5.6 Dateiupload

Der Dateiupload über die mHAD-Seite funktioniert vollständig. Die Methode *store(FileToUpload)* aus der Klasse *FileUploadController* ist in Listing 5.5 dargestellt (Anmerkung: Der Code wurde gegenüber der Implementierung stilistisch, aber nicht funktional, verändert, damit die Formatierung erhalten bleibt.). Er kann sich in Zukunft allerdings noch ändern.

Listing 5.5: Methode store(FileToUpload) in der Klasse FileUploadController

```
1  public function store(FileToUpload $request)
2  {
3      $topic = $request->input('topic');
4      $file = $request->file('file');
5      $path = $file->store('/', 'uploads');
6      $fP = storage_path("uploads/") . $path;
7
8      $d = DelimiterDetector::detectDelimiter($fP);
9      $FirstRowIsCorrect =
10         UploadedFileValidator::checkCsvFirstRow($fP, $d);
11     if ($FirstRowIsCorrect) {
12         $aRD = CsvConverter::convertCsvToJson($fP, $d);
13     }
14
15     Storage::disk('uploads')
16         ->put('appData.json', $aRD);
17
18     return view('home.show', [
19         'topic' => $topic,
20         'appArray' => $aRD
21     ]);
22 }
```

Daten, die in ein Webformular eingetragen wurden, werden als Typ *Request* wei-

tergegeben [23]. Diese Daten können dann in der *store(Request)*-Methode validiert werden. In diesem Fall wird die Validierung in ein eigens erstelltes Objekt *FileToUpload*, das die Validierungsregeln enthält, ausgelagert. Das ist der übergebene Parameter (Listing 5.5, Zeile 1).
In Listing 5.5, Zeilen 3 und 4 werden die übergebenen Daten, das *Topic* und die hochgeladene Datei, in Variablen gespeichert. In Listing 5.5, Zeile 5 folgt das Speichern der Datei in den vorgegebenen Ordner *uploads*. Die Methode *store* [23] gibt den Pfad als Wert zurück. Der komplette Pfad zu der Datei wird in Listing 5.5, Zeile 6 erstellt.
In Listing 5.5, Zeilen 8 und 9 wird der Separator ermittelt, mit Hilfe dessen die Datei in Listing 5.5, Zeile 10 zusätzlich überprüft wird. Die Methode *checkCsvFirstRow(string, string)* liest die erste Zeile einer Datei ein und separiert die Werte mittels des Separators. Sie überprüft, ob die erhaltenen Werte mit den vorgegebenen Feldnamen übereinstimmen. Wenn das der Fall ist, wird die CSV-Datei in Listing 5.5, Zeilen 11 und 12 zu JSON konvertiert und in einer Variablen gespeichert.
Es sind zwei Arten implementiert, um den erhaltenen JSON-*String* weiterzuverarbeiten:

1. Abspeichern in einer eigenen Datei, auf die dann zugegriffen werden kann (Listing 5.5, Zeilen 15 bis 16).
2. Übergeben der Variablen an die *View*, deren JavaScript-Code JSON interpretieren kann (Listing 5.5, Zeilen 18 bis 21). Die *View* stellt die Daten dem Experten tabellarisch dar.

6 Abgleich der Anforderungen

In diesem Kapitel werden die, in Unterkapiteln 3.2 und 3.3 vorgestellten, Anforderungen aufgelistet und danach bewertet, ob diese Anforderungen erfüllt sind oder nicht. Die Anforderungen sind über ihre ID und ein Stichwort identifizierbar. Der Status (Erfüllt, Ausstehend) zeigt an, ob die Anforderung erfüllt ist.

6.1 Funktionale Anforderungen

In Tabelle 6.1 sind die Funktionalen Anforderungen für den Datenimport, in Tabelle 6.2 für den Dateiupload nach ihrem Status bewertet. Letztere konnten noch nicht vollständig erfüllt werden.

ID	Stichwort	Priorität	Status
I_FA1	Konsolenbefehl für CSV-Datei	5	Erfüllt
I_FA2	Konsolenbefehl für JSON-Datei	5	Erfüllt
I_FA3	CSV-Datei in korrektes Format für Import übertragen	5	Erfüllt
I_FA4	JSON-Datei in korrektes Format für Import übertragen	5	Erfüllt
I_FA5	Validierung vor Import	5	Erfüllt
I_FA6	In Dezimalzahlen Kommata zu Punkten konvertiert	4	Erfüllt
I_FA7	Kein Import bei fehlenden Feldern oder Fehlern in verpflichtenden Werten	5	Erfüllt
I_FA8	Import bei korrekten Werten	5	Erfüllt

Tabelle 6.1: Grad der Erfüllung Funktionaler Anforderungen an den Datenimport.

ID	Stichwort	Priorität	Status
U_FA1	GUI	5	Erfüllt
U_FA2	Validierung vor Hochladen der Datei	5	Erfüllt
U_FA3	Daten in Tabelle angezeigt	5	Erfüllt
U_FA4	Hochgeladene Daten validiert	4	Ausstehend
U_FA5	Modifikation der Daten durch Experte	4	Ausstehend
U_FA6	Hochgeladene Daten importieren	5	Erfüllt

Tabelle 6.2: Grad der Erfüllung Funktionaler Anforderungen an den Dateiupload.

6.2 Nicht-Funktionale Anforderungen

In Tabelle 6.3 sind die nicht-Funktionalen Anforderungen für den Datenimport, in Tabelle 6.4 für den Dateiupload nach ihrem Status bewertet. Letztere konnten noch nicht vollständig erfüllt werden.

ID	Stichwort	Priorität	Status
I_QA1	Datenimport erweiterbar	5	Erfüllt
I_QA2	Minimum an Codeduplikation	4	Erfüllt
I_QA3	Wartbarkeit	4	Erfüllt
I_QA4	Unit- und System-Tests	5	Erfüllt

Tabelle 6.3: Grad der Erfüllung von Qualitätsanforderungen an den Datenimport.

ID	Stichwort	Priorität	Status
U_QA1	GUI einfach und intuitiv	4	Erfüllt
U_QA2	Darstellung von Fehlern	4	Ausstehend

Tabelle 6.4: Grad der Erfüllung von Qualitätsanforderungen an den Dateiupload.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Der Datenimport wurde überarbeitet. Die Datenstruktur, in der die Daten vorliegen müssen, ist das PHP-eigene assoziative Array. Um den Datenimport um neue Datenformate zu erweitern, müssen die Daten nur in ein assoziatives Array überführt werden. Das erfordert einen neuen Konsolenbefehl und eine Klasse für die Umwandlung, während die eigentlichen Datenimportklassen ohne Veränderung weiter genutzt werden können. Dadurch wird die Erweiterung des Datenimports um neue Formate einfach und Codeduplikation minimiert. Zusätzlich wurde der Import von Daten in JSON neu hinzugefügt.
Vor dem Import von Daten in die Datenbank, werden sie mittels neuer Methoden validiert und, unter Umständen, modifiziert. Dadurch wird sichergestellt, dass keine benötigten Informationen fehlen und alle Daten in der Datenbank in einer Form sind, die dem Nutzer angezeigt werden kann. Treten Fehler auf, werden spezifische *Exceptions* geworfen, die auf die aufgetretenen Fehler hinweisen.
Die Methoden wurden in Unit-Tests getestet. Es wurden auch Tests erstellt, die den kompletten Import simulieren. Alle Tests werden erfolgreich bestanden.

Der mHAD wurde eine graphische Benutzeroberfläche hinzugefügt, die das Hochladen von CSV-Dateien erlaubt. Bevor die Dateien hochgeladen werden, werden sie validiert und falls sie nicht dem Format entsprechen, erscheint eine aussagekräftige Fehlermeldung. Wenn sie dem Format entsprechen, werden die Daten in JSON übertragen und tabellarisch angezeigt. Die Daten, die in JSON vorliegen, können in die Datenbank importiert werden.

7.2 Ausblick

Die Implementierung des Datenimport ist größtenteils abgeschlossen. Er kann einfach angepasst werden, sobald der Import neuer Datei- oder Datenformate benötigt wird. In Abhängigkeit von der Implementierung der mHAD, kann noch die Datenbank für die Warteschlange der *Jobs* angepasst werden. Dadurch können mehrere Anfragen effizienter verarbeitet werden, weil die *Jobs* in einer Warteschlange zwischengespeichert werden.

Der Dateiupload bietet bereits die grundlegenden Funktionen, wird in Zukunft aber erweitert. Experten sollen im Mehrbenutzerbetrieb die Daten modifizieren können, bis sie valide sind und dann als JSON in die Datenbank importieren können.
Eine zusätzliche Funktion soll den direkten Eintrag von Daten in die Tabelle, ohne Zwischenschritt über eine CSV-Datei, ermöglichen. Der Experte wählt auf der *insert*-Seite aus, ob eine Datei hochgeladen oder Daten direkt eingegeben werden sollen. In letzterem Fall wird eine leere Tabelle angezeigt, die befüllt werden kann und in Echtzeit auf Fehler geprüft wird. Ist die Validierung erfolgreich, können die Daten direkt in die Datenbank importiert werden.

Literatur

[1] M. Achour, F. Betz, A. Dovgal u. a. *PHP.net Manual. Array*. Hrsg. von P. Cowburn. 2019. URL: https://www.php.net/manual/de/language.types.array.php. (Abgerufen am 02.07.2019).

[2] M. Achour, F. Betz, A. Dovgal u. a. *PHP.net Manual. Was ist PHP?* Hrsg. von P. Cowburn. 2019. URL: https://www.php.net/manual/de/intro-whatis.php. (Abgerufen am 02.07.2019).

[3] M. Achour, F. Betz, A. Dovgal u. a. *PHP.net Manual. Funktionsparameter*. Hrsg. von P. Cowburn. 2019. URL: https://www.php.net/manual/de/functions.arguments.php. (Abgerufen am 02.07.2019).

[4] M. Achour, F. Betz, A. Dovgal u. a. *PHP.net Manual. Klassen und Objekte - Einführung*. Hrsg. von P. Cowburn. 2019. URL: https://www.php.net/manual/de/oop5.intro.php. (Abgerufen am 02.07.2019).

[5] M. Achour, F. Betz, A. Dovgal u. a. *PHP.net Manual. foreach*. Hrsg. von P. Cowburn. 2019. URL: https://www.php.net/manual/de/control-structures.foreach.php. (Abgerufen am 02.07.2019).

[6] M. Achour, F. Betz, A. Dovgal u. a. *PHP.net Manual. explode*. Hrsg. von P. Cowburn. 2019. URL: https://www.php.net/manual/de/function.explode.php. (Abgerufen am 02.07.2019).

[7] M. Achour, F. Betz, A. Dovgal u. a. *PHP.net Manual. sizeof*. Hrsg. von P. Cowburn. 2019. URL: https://www.php.net/manual/de/function.sizeof.php. (Abgerufen am 02.07.2019).

[8] M. Achour, F. Betz, A. Dovgal u. a. *str_getcsv*. Hrsg. von P. Cowburn. 2019. URL: https://www.php.net/manual/de/function.str-getcsv.php. (Abgerufen am 02.07.2019).

[9] U.-V. Albrecht, M. Höhn und U. von Jan. „Kapitel 2. Gesundheits-Apps und Markt.“ In: *Chancen und Risiken von Gesundheits-Apps (CHARISMHA)*. Hrsg. von U.-V. Albrecht. 2016, S. 78. urn:nbn:de:gbv:084-160408112.

[10] M. Bean. *Laravel 5 Essentials*. Birmingham: Packt Publishing Ltd., 2015, S. 8–9. ISBN: 978-1-78528-301-7.

[11] U. Bork, J. Weitz und V. Penter. „Apps und Mobile Health: Viele Potenziale noch nicht ausgeschöpft“. In: *Deutsches Ärzteblatt* 115 (3), S. 62–66 A.

[12] NHN Corp. *NHN*. 2019. URL: https://www.nhn.com/ko/index.nhn. (Abgerufen am 02.07.2019).

[13] NHN Corp. *TOAST UI Grid*. 2019. URL: https://ui.toast.com/tui-grid/. (Abgerufen am 02.07.2019).

[14] MariaDB Foundation. *About MariaDB*. 2019. URL: https://mariadb.org/about/. (Abgerufen am 02.07.2019).

[15] The Apache Software Foundation. *Getting Started*. 2019. URL: https://httpd.apache.org/docs/2.4/getting-started.html. (Abgerufen am 02.07.2019).

[16] M. T. Goodrich, R. Tamassia und D. M. Mount. *Data Structures and Algorithms in C++*. 2. Aufl. John Wiley & Sons, Inc., 2011. Kap. 4.2. Analysis of Algorithms, S. 177. ISBN: 978-0-470-38327-8.

[17] HashiCorp. *Vagrant*. 2018. URL: https://www.vagrantup.com/. (Abgerufen am 02.07.2019).

[18] M. Hausenblas, E. Wilde und J. Tennison. *RFC 7111. URI Fragment Identifiers for the text/csv Media Type*. 2014. URL: https://tools.ietf.org/html/rfc7111. (Abgerufen am 02.07.2019).

[19] Microsoft. *Visual Studio Code*. 2019. URL: https://code.visualstudio.com/. (Abgerufen am 02.07.2019).

[20] Oracle. *VirtualBox*. 2018. URL: https://www.virtualbox.org/. (Abgerufen am 02.07.2019).

[21] T. Otwell. *Laravel. Queues*. Version 5.6. 2018. URL: https://laravel.com/docs/5.6/queues. (Abgerufen am 02.07.2019).

[22] T. Otwell. *Laravel. Validation*. Version 5.6. 2018. URL: https://laravel.com/docs/5.6/validation. (Abgerufen am 02.07.2019).

[23] T. Otwell. *Laravel. HTTP Requests*. Version 5.6. 2018. URL: https://laravel.com/docs/5.6/requests. (Abgerufen am 02.07.2019).

[24] T. Otwell. *Laravel. Laravel Homestead*. Version 5.6. 2018. URL: https://laravel.com/docs/5.6/homestead. (Abgerufen am 02.07.2019).

[25] Research2Guidance. *mHealth App Economics 2017/2018*. 2018, S. 9.

[26] Y. Shafranovich. *RFC 4180. Common Format and MIME Type for Comma-Separated Values (CSV) Files*. 2005. URL: https://tools.ietf.org/html/rfc4180. (Abgerufen am 02.07.2019).

[27] S. R. Stoyanov, L. Hides, D. J. Kavanagh, O. Zelenko, D. Tjondronegoro und Mani M. „Mobile App Rating Scale: A New Tool for Assessing the Quality of Health Mobile Apps“. In: *JMIR mHealth and uHealth* 3.1, e27 (2015). DOI: 10.2196/mhealth.3422.

[28] Massachusetts Institute of Technology. *MIT License*. 1988. URL: https://opensource.org/licenses/MIT. (Abgerufen am 02.07.2019).

A Anhang

Die Schlüssel für die Felder *focus_app_targets* sind in Tabelle A.1 zu sehen, für *theoretical_background* in Tabelle A.2, für *affiliation* in Tabelle A.3, für *age_group* in Tabelle A.4 und für *technical_aspects* in Tabelle A.5.

Schlüssel	Bedeutung
fat_alcohol	Alcohol / Substance Use
fat_anger	Anger
fat_anxiety	Anxiety / Stress
fat_behaviour	Behaviour Change
fat_depression	Depression
fat_entertainment	Entertainment
fat_goal	Goal Setting
fat_happiness	Increase Happiness / Well-being
fat_mindfulness	Mindfulness / Meditation / Relaxation
fat_health	Physical health
fat_negative_emotions	Reduce negative emotions
fat_relationships	Relationships
fat_other	Other

Tabelle A.1: Schlüssel und ihre Bedeutung nach den Vorgaben für Focus App Targets.

Schlüssel	Bedeutung
tb_act	ACT - Acceptance commitment therapy
tb_advice	Advice / Tips / Strategies / Skill training
tb_assessment	Assessment
tb_behavioural	CBT-Behavioural (positive events)
tb_cognitive	CBT-Cognitive (thought challenging)
tb_feedback	Feedback
tb_goal	Goal Setting
tb_gratitude	Gratitude
tb_information	Information / Psychoeducation
tb_mindfulness	Mindfulness / Mediation
tb_monitoring	Monitoring / Tracking
tb_relaxation	Relaxation
tb_strengths	Strenghts based
tb_other	Other

Tabelle A.2: Schlüssel und ihre Bedeutung nach den Vorgaben für Theoretical Background.

Schlüssel	Bedeutung
a_commercial	Commercial
a_government	Government
a_ngo	Non-governmental organization (NGO)
a_university	University
a_unknown	Unknown

Tabelle A.3: Schlüssel und ihre Bedeutung nach den Vorgaben für Affiliation.

Schlüssel	Bedeutung
ag_adolescents	Adolescents (13-17)
ag_adults	Adults
ag_children	Children (< 12)
ag_general	General
ag_young_adults	Young Adults (18-25)

Tabelle A.4: Schlüssel und ihre Bedeutung nach den Vorgaben für Age Group.

Schlüssel	Bedeutung
ta_password	Allows password-protection
ta_sharing	Allows sharing (Facebook, Twitter, etc.)
ta_community	Has an app community
ta_web_access	Needs web access to function
ta_login	Requires login
ta_reminders	Sends reminders

Tabelle A.5: Schlüssel und ihre Bedeutung nach den Vorgaben für Technical Aspects.

B Abkürzungsverzeichnis

Die in dieser Arbeit verwendeten Abkürzungen sind in Tabelle B.1 aufgelistet.

Ajax	Asynchronous JavaScript and XML
CSV	Comma-Separated Values
DSV	Delimiter-Separated Values
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifikationsnummer
JSON	JavaScript Object Notation
MARS	Mobile Anwendungen Rating Skala
MHAD	Mobile Health App Database
MVC	Model-View-Controller
PHP	PHP: Hypertext Preprocessor
SQL	Structured Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator

Tabelle B.1: Die verwendeten Abkürzungen.

Name: Vincenzo Francesco Brancaccio Matrikelnummer: 519883

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Vincenzo Francesco Brancaccio