
Anleitung

Für die Digital Health InformMe Case – Anwendung

Gruppe D

Inhaltsverzeichnis

Abbildungsverzeichnis	3
Tabellenverzeichnis.....	4
Ausführen und Erkunden der Anwendung.....	5
A.1 Voraussetzungen für die Ausführung.....	5
A.2 Ausführen der Anwendung.....	6
A.3 Frontend im Browser.....	8
A.4 REST-Service.....	8
A.4.1 HL7-Mock: get-random-message	9
A.4.2 Praxis: read.....	11
A.4.3 Praxis: create-patient.....	12
A.4.4 Praxis: Datenänderung seitens Patientin.....	15
A.4.5 Praxis: Duplikate in der Datenbank	17
A.4.6 Übersicht der Endpoints	18
A.4.7 Problembehebung beim Ausführen der Anwendung	20
A.5 Kontakt.....	20

Abbildungsverzeichnis

Abbildung A 1: Dateispeicherort öffnen.....	6
Abbildung A 2: Konsolenausgabe.....	7
Abbildung A 3: GET-Request "/hl7-mock/get-random-message".....	10
Abbildung A 4: POST-Request "/patient/read".....	11
Abbildung A 5: GET-Request "/hl7-mock/123456".....	12
Abbildung A 6: POST-Request "/patient/create-patient".....	13
Abbildung A 7: POST-Request "/patient/read" (Jane Doe).....	14
Abbildung A 8: GET-Request "/hl7-mock/654321" (Jane Doe).....	15
Abbildung A 9: POST-Request "/patient/read" (Jane Doe, neue Adresse).....	16
Abbildung A 10: POST-Request "/patient/read?autoUpdate=false" (Jane Santos).....	17
Abbildung A 11: POST-Request "/patient/create-patient" (Duplikat Jane Doe).....	18
Abbildung A 12: POST-Request "/patient/read" (Duplikat Jane Doe).....	18

Tabellenverzeichnis

Tabelle A 1: REST-Endpoints der Anwendung	19
---	----

Ausführen und Erkunden der Anwendung

Im folgenden wird hier von einer „Praxis“ die Rede sein, da der Code unter anderem das Backend (und Teile des Frontends) einer von uns ausgedachten Praxis „Digital Health Praxis“ implementiert. Außer der Praxis gibt es außerdem noch eine von uns codeseitig simulierte Instanz, welche HL7-Nachrichten sendet und empfängt.

Innerhalb der Anwendungen, welche wir mit den Sprachen und Tools Java, HTML und SpringBoot umgesetzt haben, können Sie folgende Funktionalitäten ausführen:

- Anlegen von neuen Patient:innen in der Praxis
- Lesen und Verarbeiten von HL7-Nachrichten
- Einsehen von Patient:innen der Praxis
- Aktualisieren von Daten der Patient:innen

Auf HL7-Mock-Ebene gibt es folgende Funktionen:

- Eine zufällige HL7-Nachricht erhalten (Simulation des Prozesses, dass die HL7-Nachricht einer eGK von einer:m beliebigen Patient:in über das Kartenlesegerät per Terminal an unser Praxis-Backend gesendet wird)
- Bereits existierende Entitäten in der HL7-Instanz anhand ihrer HL7-ID einsehen (in Form ihrer HL7-Nachricht)

Da der Programmcode hier zu viel Platz einnehmen würde, haben wir für Sie ein öffentliches GitHub-Repository eingerichtet. Dort können Sie gerne den Code in `src > main > java` (Backend) und `src > main > resources` (Frontend) erkunden ¹:

<https://github.com/wensonsh/informme-case-d-public>

A.1 Voraussetzungen für die Ausführung

Bevor die Anwendung ausgeführt werden kann, sollten die folgenden Rahmenbedingungen erfüllt sein:

1. Java ist auf dem Rechner installiert.

Im Code der Anwendung wird Java 17 genutzt. Wir können nicht garantieren, dass die Anwendung auch mit älteren Java-Versionen ausgeführt werden kann. Falls Sie Probleme bei der Ausführung haben, versuchen Sie es bitte mit der Java-Version 17.

2. Für das Testen und erkunden aller Funktionen der Anwendung wird ein Tool benötigt, welches mit RESTful APIs kommunizieren kann (um REST-Anfragen zu senden). Während

¹ Falls Sie Zugriff auf unseren dev-Code haben möchten (private repository), wenden Sie sich dafür gerne an Wendi Shu (sh. Kontakt). Da der dev-Code sensible Daten wie Datenbankpasswörter beinhaltet, haben wir ein public Repository ohne diese Daten erstellt.

unserer Implementationstests haben wir teamintern *Postman* benutzt. Mit Postman können lokale REST-Anfragen, wie in unserem Fall, nur mit der heruntergeladenen (Desktop) Anwendung genutzt werden (kostenlos).

A.2 Ausführen der Anwendung

1. Bitte laden Sie die Datei „informme-case.jar“ aus dem [git-Repository](#) herunter und speichern Sie diese Datei an einem beliebigen Speicherort auf ihrem PC. Diese ist die Anwendungsdatei, welche die Code- und Konfigurationsressourcen der Anwendung beinhaltet.

A. Für Windows-PCs fahren Sie nun hier fort. Für PCs mit MacOS, springen Sie [hier](#) hin:
MacOS

B. Windows

- a. Öffnen Sie nun die **Eingabeaufforderung** ihres PCs.
- b. Die Anwendung kann nun mit dem folgenden **Befehl** ausgeführt werden:

```
„PFAD_ZUR_java.exe-DATEI\java.exe“ -jar  
PFAD_ZUR_JAR_DATEI\informMe-case.jar
```

Der *PFAD_ZUR_java.exe-DATEI* sowie der *PFAD_ZUR_JAR_DATEI* müssen mit dem tatsächlichen Pfad ersetzt werden:

PFAD_ZUR_java.exe-DATEI finden:

- i. Windows-Taste drücken
- ii. „Java“ eingeben
- iii. „Dateispeicherort öffnen“ auswählen (sh. Abbildung A 1)

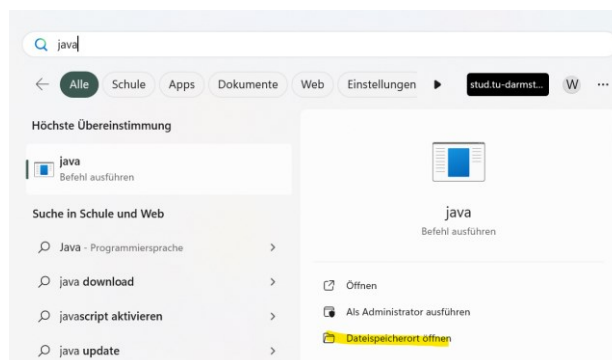


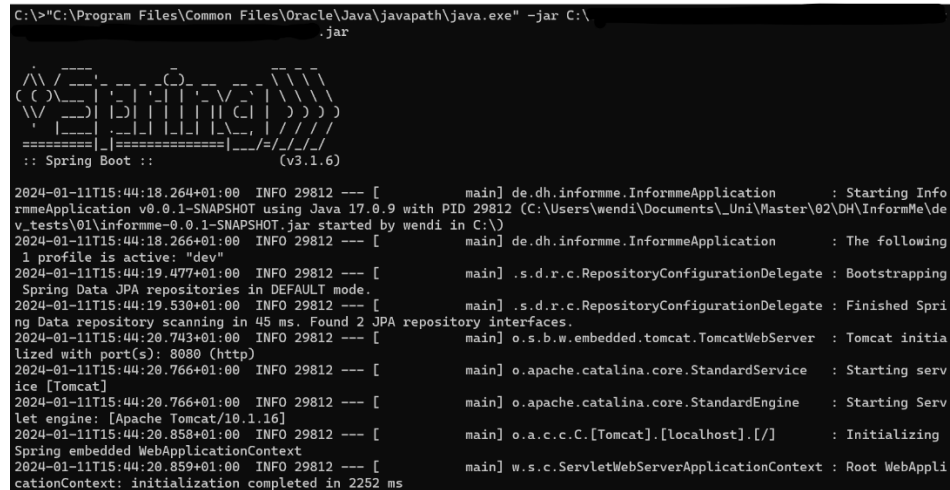
Abbildung A 1: Dateispeicherort öffnen

- iv. Den Dateipfad des Ordners kopieren, der sich geöffnet hat. Dies ist der *PFAD_ZUR_java.exe-DATEI*

Der PFAD_ZUR_JAR_DATEI

ist der Dateipfad zu dem Ordner, in welchem Sie die Anwendungsdatei „informme-case.jar“ gespeichert haben.

- c. Drücken Sie nun **Enter**. Ihre Konsole sollte nun anfangen, Zeilen zu generieren, die ungefähr so aussehen wie in Abbildung A 2.



```
C:\>"C:\Program Files\Common Files\Oracle\Java\javapath\java.exe" -jar C:\v_tests\01\informme-0.0.1-SNAPSHOT.jar

:: Spring Boot :: (v3.1.6)

2024-01-11T15:44:18.264+01:00 INFO 29812 --- [main] de.dh.informme.InformmeApplication : Starting InformmeApplication v0.0.1-SNAPSHOT using Java 17.0.9 with PID 29812 (C:\Users\wendi\Documents\Uni\Master\02\DH\InformMe\dev_tests\01\informme-0.0.1-SNAPSHOT.jar started by wendi in C:\)
2024-01-11T15:44:18.266+01:00 INFO 29812 --- [main] de.dh.informme.InformmeApplication : The following 1 profile is active: "dev"
2024-01-11T15:44:19.477+01:00 INFO 29812 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-01-11T15:44:19.530+01:00 INFO 29812 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 45 ms. Found 2 JPA repository interfaces.
2024-01-11T15:44:20.743+01:00 INFO 29812 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2024-01-11T15:44:20.766+01:00 INFO 29812 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-01-11T15:44:20.766+01:00 INFO 29812 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.16]
2024-01-11T15:44:20.858+01:00 INFO 29812 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-01-11T15:44:20.859+01:00 INFO 29812 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2252 ms
```

Abbildung A 2: Konsolenausgabe

C. MacOS

- a. Öffnen Sie das Terminal Ihres Macs.
- b. Die Anwendung kann nun mit dem folgenden Befehl geöffnet werden:

```
java -jar PFAD_ZUR_JAR_DATEI\informme-case.jar
```

Der PFAD_ZUR_JAR_DATEI muss mit dem tatsächlichen Dateipfad ersetzt werden:

PFAD_ZUR_JAR_DATEI finden:

- i. Über den Finder zur Datei „informMe-Case.jar“ navigieren
 - ii. Die „informMe-Case.jar“-Datei auswählen und die Tastenkombination „Cmd + i“ drücken
 - iii. Im Info-Fenster „Ort“ ausfindig machen und mit dem Sekundärklick anklicken und die Option „Als Pfadname kopieren“ auswählen
- c. Drücken Sie nun Enter. Ihre Konsole sollte nun anfangen, Zeilen zu generieren, die ungefähr so aussehen wie in Abbildung A 2.

2. Wenn schließlich ganz unten als letzte Zeile dieser Text erscheint: „Started InformmeApplication in xx.xx seconds“, ist die Anwendung erfolgreich im Hintergrund gestartet und Sie können ihre Funktionalitäten nun selbst erkunden.

3. Probieren Sie gerne aus, ob die Anwendung erfolgreich gestartet werden konnte, indem Sie in Ihrer Adresszeile eines beliebigen Browsers „localhost:8080“ eingeben. Nun sollte die folgende Seite in Ihrem Browser geladen werden:



4. **Anwendung beenden:** Um die Anwendung zu beenden, gehen Sie wieder in die Eingabeaufforderung bzw. in das Terminal. Drücken Sie die Tastenkombination „Strg + C“ (für Windows) bzw. „control + C“ (für MacOS). Daraufhin erscheint die Zeile „[...] Shutdown completed“. Drücken Sie nun die Enter-Taste, um die Anwendungsausführung endgültig zu beenden und ihre Konsole wieder nutzen zu können.

A.3 Frontend im Browser

Das Frontend der implementierten Anwendung ist aufgrund des hohen Aufwands, der für den High-Fidelity-Prototypen eingesetzt wurde, auf ein Minimum begrenzt. Nichtsdestotrotz können in der von uns implementierten Anwendung Patient:innen-Daten in einem stark vereinfachten User Interface angezeigt werden. Bei den Patient:innen-Daten, die hier angezeigt werden, handelt es sich stets um existierende Patient:innen in der von uns gehosteten Datenbank. Dies soll den Vorgang simulieren, dass ein:e beliebige Patient:in die Praxis betritt und für die Anmeldung ihre eGK in das entsprechende Kartenfach einsteckt.

Wenn Sie die Anwendung erfolgreich gestartet haben, rufen Sie in Ihrem Browser die Adresse „localhost:8080“ auf. Daraufhin sehen Sie den implementierten Frontend-Teil unserer Anwendung. Hier können Sie sich die Daten einer:s zufällige:n Patient:in aus unserer Praxisdatenbank anzeigen lassen.

A.4 REST-Service

Die weiter oben beschriebenen Funktionalitäten unserer Anwendung laufen, bis auf das in Anhang-Kapitel vorgestellte Feature, über eine RESTful API. Um die folgenden Features ausführen und testen zu können, benötigen Sie nun ein Tool, mit welchem Sie REST-Anfragen senden können. In den folgenden Screenshots handelt es sich um Screenshots, die innerhalb des Tools „Postman“ aufgenommen wurden.

Die Funktionalitäten unserer Anwendung basieren auf einem von uns definierten Prozess, der durchlaufen wird, wenn ein:e Patient:in sich für einen Termin anmelden möchte. Dabei ist zu

beachten, dass wir das Lesen einer eGK durch unseren eigenen Service simulieren, da wir keinen Zugriff auf die benötigte Hardware und Schnittstelle für das tatsächliche Auslesen von Gesundheitskarten haben.

Im Folgenden werden wir Sie Schritt für Schritt durch den von uns vorgesehenen Prozess, den die Anwendung implementiert, leiten.

A.4.1 HL7-Mock: get-random-message

Der Prozess zur Anmeldung von Patient:innen beginnt damit, dass die Patient:innen ihre eGK in das Kartenlesegerät stecken. Damit wird das Abrufen der HL7-Nachricht aus der eGK in Gang gesetzt. Auf Anwendungsebene wird dieser Schritt dadurch simuliert, dass eine bereits existierende HL7-Nachricht aus der HL7-Mock-Datenbank geholt wird. Dabei kann entweder ein zufälliger Eintrag geholt werden, oder, falls später die Identifikationsnummer aus dem MSH-Segment bekannt ist, kann die Nachricht direkt über die Angabe der ID geholt werden. Da Ihnen zu Beginn keine der HL7-Nachrichten bekannt vorkommen wird, empfehlen wir Ihnen, beim ersten Aufruf des Endpunktes folgende GET-Anfrage zu senden:

`localhost:8080/hl7-mock/get-random-message`

Abbildung A 3 zeigt, wie die Anfrage in Postman aussieht: Beginnend mit der Adresse *localhost:8080* (lokale Instanz der Anwendung), wird mit dem URL-Teil „/hl7-mock“ angedeutet, dass wir nun mit der HL7-Mock-Instanz kommunizieren möchten. Mit „/get-random-message“ am Ende der URL weiß das Backend nun, dass ein zufälliger Eintrag aus der HL7-Mock-Datenbank zurückgegeben werden soll. Diese wird schließlich in dem Antwort-Bereich (Response-Body) angezeigt. Dieser Schritt soll den Prozess simulieren, dass die HL7-Nachricht aus der Gesundheitskarte ausgelesen wird.

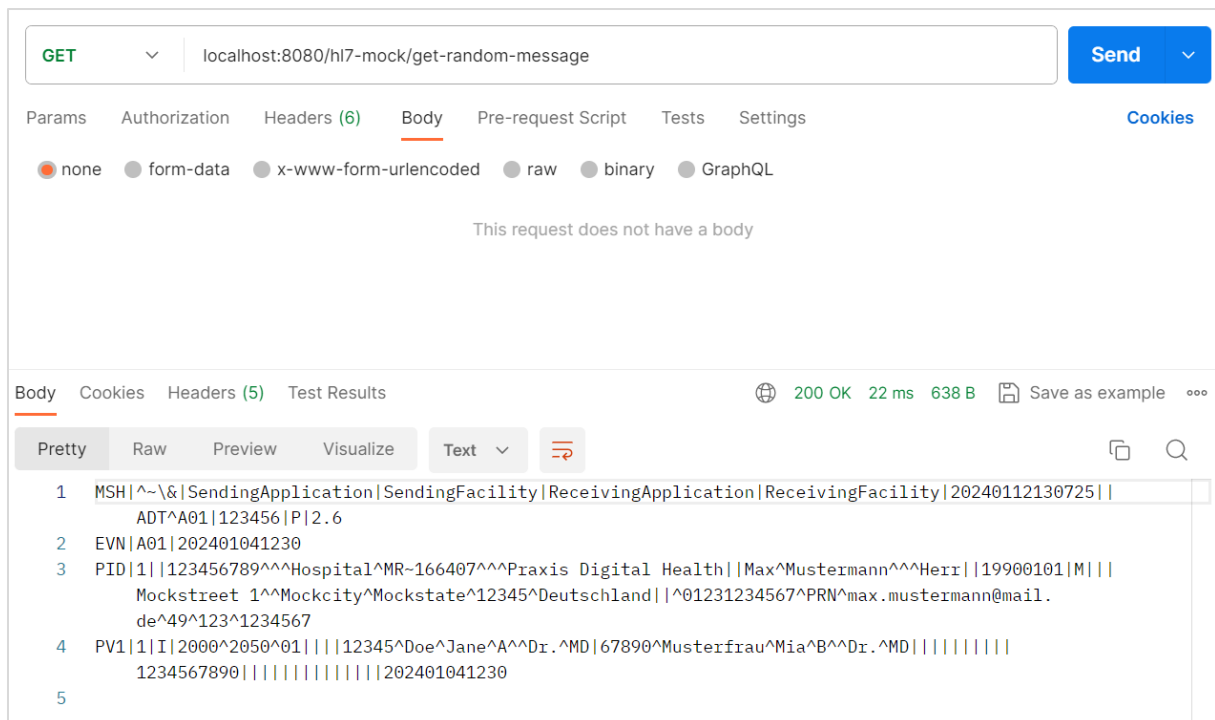


Abbildung A 3: GET-Request "/hl7-mock/get-random-message"

Damit nicht alles eine Blackbox ist, und somit in dem gesamten Prozess besser erkannt werden kann, welche Nachrichten und Objekte in der Anwendung kommuniziert werden können, teilen wir die Schritte so auf, dass „Zwischenergebnisse“ im Response-Body angezeigt werden.

In diesem Fall verarbeiten wir also nicht sofort die empfangene Nachricht, die wir „von der Gesundheitskarte bekommen“, sondern zeigen sie erst einmal an. Diese kann nun im nächsten Schritt auf Praxisebene verarbeitet werden:

Kopieren Sie dafür den Text aus dem Body (= die erhaltene HL7-Nachricht). Die Nachricht sieht zum Beispiel so aus:

```
MSH|^~\&|SendingApplication|SendingFacility|ReceivingApplication|ReceivingFacility|202401041230||ADT^A01|123456|P|2.6
```

```
EVN|A01|202401041230||
```

```
PID|1||123456789^^^Hospital^MR||Max^Mustermann^^^Herr||19900101|M|||Mockstreet
1^^Mockcity^Mockstate^12345^Deutschland||^01231234567^PRN^max.mustermann@mail.
de^49^123^1234567^^^^^^|
```

```
PV1|1|I|2000^2050^01||||12345^Doe^Jane^A^^Dr.^MD|67890^Musterfrau^Mia^B^^Dr.^MD|
||||||1234567890|||||||202401041230||
```

Folgende Felder werden wir in unseren kommenden Beispielen etwas genauer betrachten:

- MSH-Segment: Identifikationsnummer und
- PID-Segment.

A.4.2 Praxis: read

Als nächstes wird mit dieser Nachricht der Prozessschritt in der Praxis initialisiert, die übermittelte Nachricht auszulesen und zu verarbeiten. Dafür wird eine POST-Anfrage an folgende Adresse gesendet:

localhost:8080/patient/read

Die kopierte HL7-Nachricht ist dabei als Request-Body (Text) anzugeben (sh. Abbildung A 4).

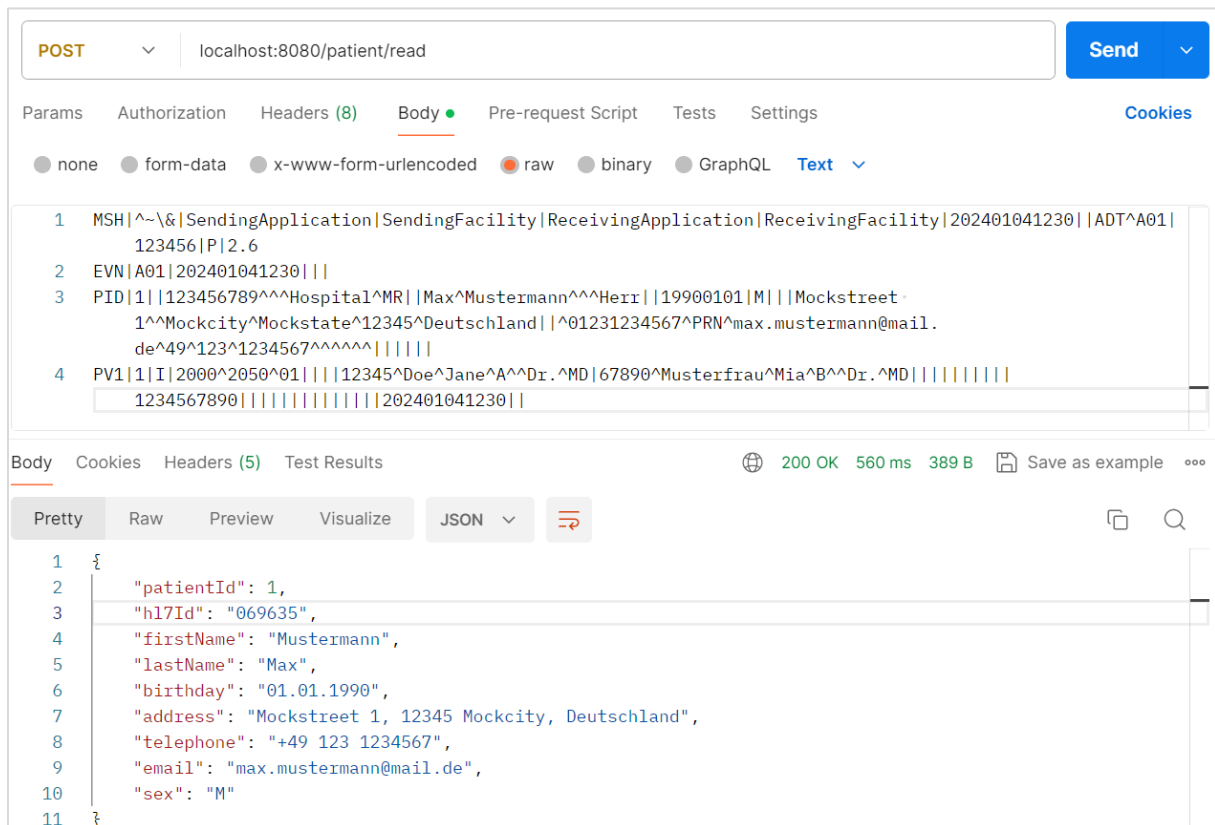


Abbildung A 4: POST-Request `/patient/read`

Das Ergebnis ist das aus der Gesundheitskarte erstellte Patient-Objekt mit den aus der Nachricht extrahierten Daten. Außerdem wurde eine hl7Id (069635) für diesen neuen Patienten Max Mustermann generiert und entsprechend abgespeichert. Die Methode für das Extrahieren der Adresse sieht zum Beispiel wie folgt aus:

```
/**
 * get the patient's address as String
 *
 * @param pid PID segment of the HL7 message
 * @return patient's address as String
 */
public String getPatientAddressAsString(PID pid) {
    XAD[] addresses = pid.getPatientAddress();
    String addressAsString = "Invalid";
    if (addresses == null || addresses.length == 0) {
        return addressAsString;
    }
    XAD address = addresses[0];
```

Dabei werden aus dem XAD-Array (Extended Address) die jeweiligen Felder herausgefiltert und als String zusammengefügt. Innerhalb der Methode werden andere Methoden unseres Codes aufgerufen, um die jeweiligen Adress-Komponenten herauszusuchen. Falls Sie sich diesen Code-Abschnitt im Repository genauer anschauen wollen, finden Sie unter *src/main/java/de/dh/informme/doctorsOffice/hl7* in der Klasse *HL7Parser.java*

Max Mustermann hat nun für unsere Digital Health Praxis auch eine Identifikationsnummer. Anhand dieser Nummer erkennen wir beim nächsten Besuch von Herrn Mustermann, dass wir kein neues Patient-Objekt mehr anlegen müssen, weil Herr Mustermann schonmal bei uns war. Wir werden lediglich prüfen, ob sich seine Daten geändert haben.



Nun gehen wir von dem Fall aus, dass eine neue Patientin Jane Doe einen Termin über Doctolib bei uns vereinbart hat. Sie hat auf der Plattform Name, Geburtsdatum, Geschlecht, Adresse,

Telefonnummer und E-Mail angegeben. In unserer Praxis wird mit Hilfe der folgenden POST-Anfrage ihr entsprechendes Objekt in der Praxisdatenbank erstellt:

localhost:8080/patient/create-patient

Der Request-Body ist dabei ein JSON-String, welcher ihre entsprechenden Informationen beinhaltet. Dabei ist darauf zu achten, dass der Geburtstag das Format „dd.MM.yyyy“ hat, während das Geschlecht eines der folgenden Angaben ist: „A“, „F“, „M“, „N“, „O“ oder „U“.

Wie aus Abbildung A 6 zu entnehmen ist, sehen wir unten im Response-Body, dass ein neuer Eintrag in unserer Datenbank gemacht wurde. Jane Doe ist die zweite Patientin unserer Praxis, also ist ihre praxisinterne Identifikationsnummer 2. Da wir ihre Gesundheitskarte noch nicht gescannt haben, hat ihre hl7Id noch keinen Wert.

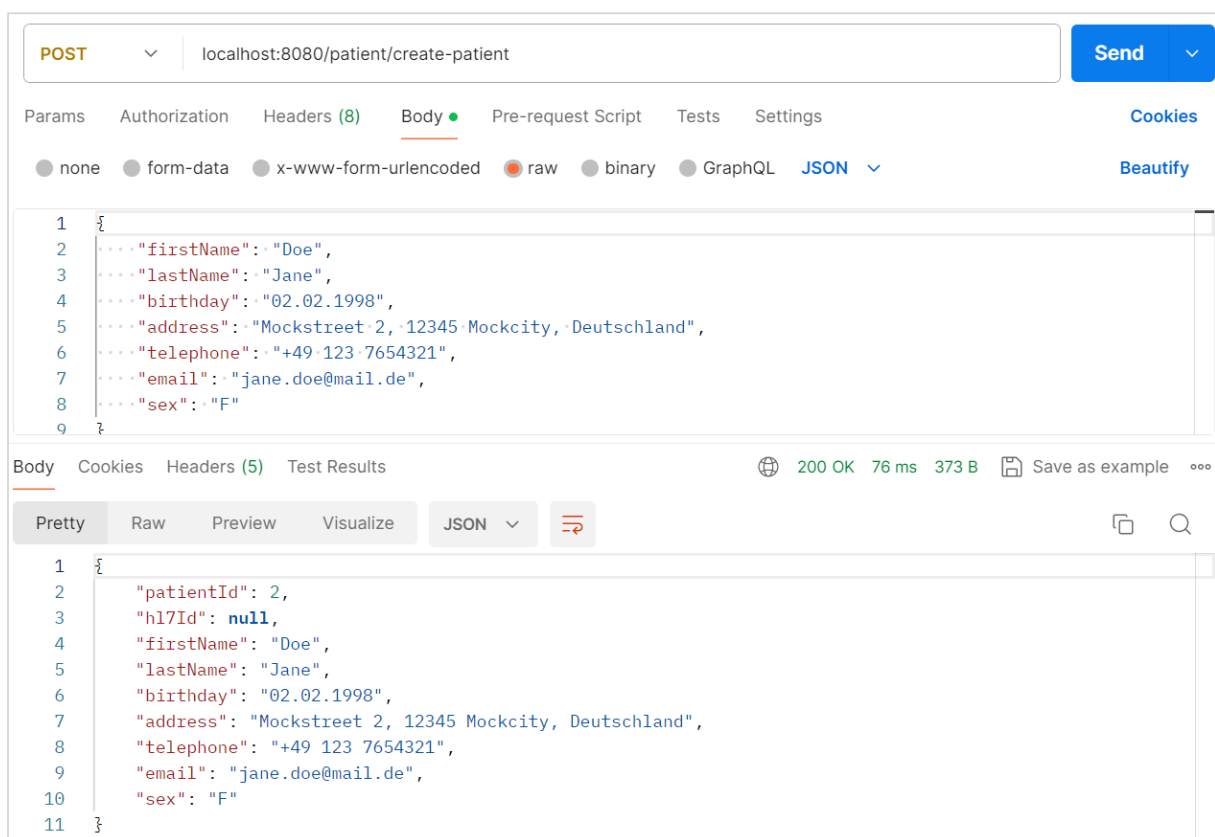


Abbildung A 6: POST-Request "/patient/create-patient"

Frau Doe besucht uns nun zum ersten Mal in der Praxis, nachdem sie sich über Doctolib angemeldet hat. Ihre Gesundheitskarte beinhaltet die folgende HL7-Nachricht:

MSH|^~\&|SendingApplication|SendingFacility|ReceivingApplication|ReceivingFacility|20240112182256||ADT^A01|654321|P|2.6

EVN|A01|202401041230

PID|1||||Jane^Doe^^^Frau||19980202|F|||Mockstreet
2^^Mockcity^Mockstate^12345^Deutschland||^01237654321^PRN^jane.doe@mail.de^49^12
3^7654321

```
PV1|1|I|2000^2050^01||||12345^Doe^Jane^A^Dr.^MD|67890^Musterfrau^Mia^B^Dr.^MD|
||||||1234567890|||||||202401041230
```

Aus dem MSH-Segment können wir ablesen, dass wir sie in der HL7-Mock-Datenbank später mit der ID 654321 finden können. Aus dem PID-Segment geht hervor, dass noch von keiner Instanz aus dem Gesundheitswesen eine eindeutige Kennung in der *Patient Identifier List* hinterlegt wurde – vielleicht war sie bis heute noch nie beim Arzt?

Beim Auslesen der Nachricht wird nun geprüft, ob bereits jemand mit ihren Daten existiert. Dabei wird geschaut, ob jemand mit dem gleichen Namen + Geburtsdatum existiert. Anhand der ID können wir schließlich keine Duplikatsprüfung durchführen, weil sie noch keine hl7Id bei uns hat.

The screenshot shows a REST client interface with a POST request to `localhost:8080/patient/read`. The request body is an HL7 MSH message. The response is a JSON object representing a patient.

Request Body (HL7 MSH):

```
1 MSH|^~\&|SendingApplication|SendingFacility|ReceivingApplication|ReceivingFacility|20240112130725||
2 ADT^A01|654321|P|2.6
3 EVN|A01|202401041230
4 PID|1||Jane^Doe^^^Frau||19980202|F||Mockstreet 2^Mockcity^Mockstate^12345^Deutschland||
5 ^01237654321^PRN^jane.doe@mail.de^49^123^7654321
6 PV1|1|I|2000^2050^01||||12345^Doe^Jane^A^Dr.^MD|67890^Musterfrau^Mia^B^Dr.^MD|||||||
7 1234567890|||||||202401041230
```

Response Body (JSON):

```
{
  "patientId": 2,
  "hl7Id": "061116",
  "firstName": "Doe",
  "lastName": "Jane",
  "birthday": "02.02.1998",
  "address": "Mockstreet 2, 12345 Mockcity, Deutschland",
  "telephone": "+49 123 7654321",
  "email": "jane.doe@mail.de",
  "sex": "F"
}
```

Abbildung A 7: POST-Request "/patient/read" (Jane Doe)

Unsere Antwort beim Auslesen ist das Patient-Objekt als JSON-String, wie in Abbildung A 7 zu sehen. Es wurden also keine Duplikate gefunden. Wir sehen außerdem, dass sie nun eine hl7Id von uns zugewiesen bekommen hat. Schauen wir uns ihre HL7-Nachricht nun über den HL7-Mock-Service an, sehen wir, dass unsere hl7Id der Nachricht hinzugefügt wurde (sh. Abbildung A 8). Beim nächsten Mal können wir sie also über diese ID finden.



Abbildung A 8: GET-Request "/hl7-mock/654321" (Jane Doe)

A.4.4 Praxis: Datenänderung seitens Patientin

Jane Doe zieht nun um und meldet ihre neue Adresse bei ihrer Krankenkasse. Diese ändert die Adresse umgehend, damit aktualisiert sich auch das PID-Segment aus ihrer HL7-Nachricht:

```
MSH|^~\&|SendingApplication|SendingFacility|ReceivingApplication|ReceivingFacility|20240112182256||ADT^A01|654321|P|2.6
```

```
EVN|A01|202401041230
```

```
PID|1||061116^^^Praxis Digital Health||Jane^Doe^^^Frau||19980202|F||Teststr.
20^^Teststadt^Testbundesland^54321^Testland||^01237654321^PRN^jane.doe@mail.de
^49^123^7654321
```

```
PV1|1|I|2000^2050^01||||12345^Doe^Jane^A^^Dr.^MD|67890^Musterfrau^Mia^B^^Dr.^MD|||||||1234567890||
|||||||202401041230
```

Sie besucht nun noch einmal unsere Digital Health Praxis, wir lesen also ihre aktualisierte Nachricht aus. Wie in dem Response-Body in Abbildung A 9 zu sehen, übernehmen wir automatisch die geänderten Daten und aktualisieren diese in der Datenbank.

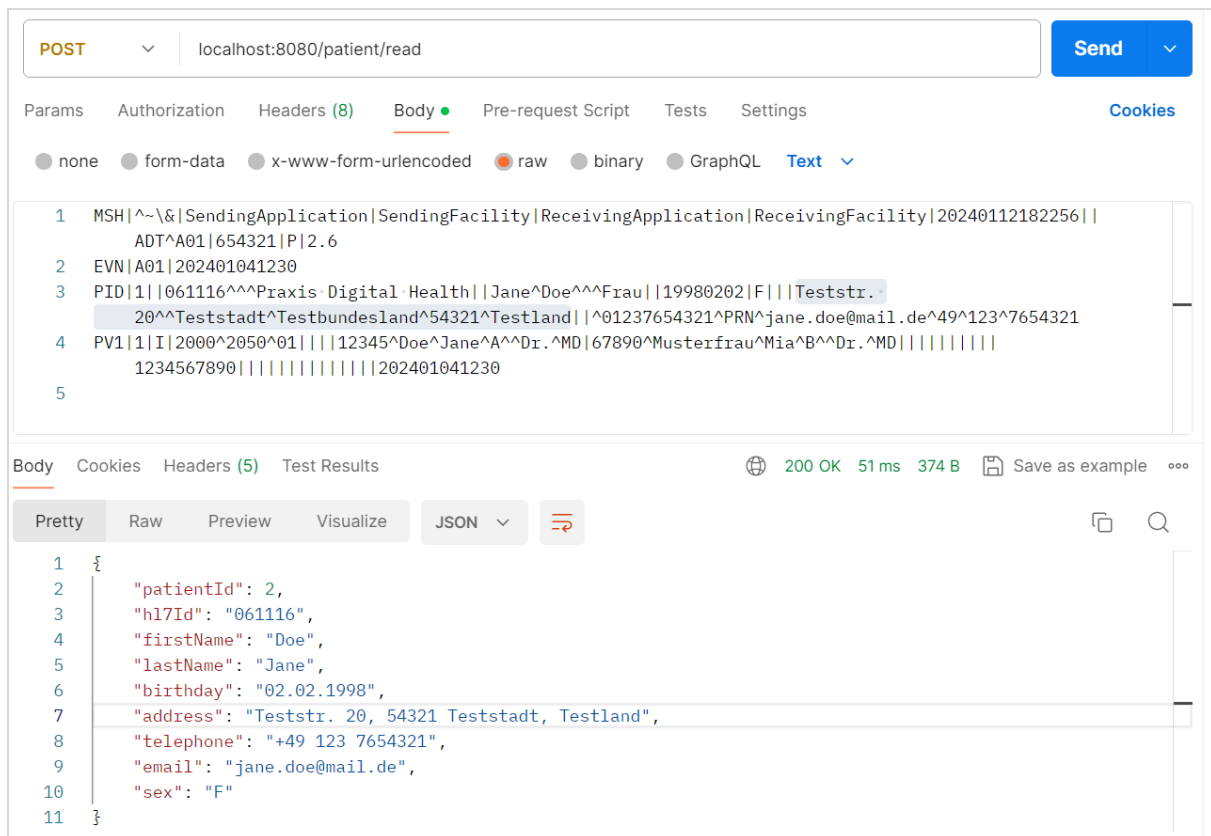


Abbildung A 9: POST-Request "/patient/read" (Jane Doe, neue Adresse)

Diese Funktion des automatischen Updates kann auch abgeschaltet werden, indem bei der POST-Anfrage ein zusätzlicher Request-Parameter mitgegeben wird: `autoUpdate=false`.

Gehen wir, um diesen Fall zu veranschaulichen, davon aus, dass Jane Doe einen neuen Nachnamen hat: Santos. Sie meldet diese Namensänderung wie gewohnt bei ihrer Krankenkasse. Das PID-Segment in ihrer Nachricht ändert sich also wieder. Dieses Mal, wollen wir als Praxis beim Auslesen ihrer Nachricht nicht, dass die Daten automatisch überschrieben werden (gemäß dem Fall aus Gawlitza, 2023). Dafür fügen wir also unserer POST-Anfrage einen Request Parameter hinzu:

`localhost:8080/patient/read?autoUpdate=false`

Wie in Abbildung A 10 zu sehen, wurde nun im Backend erkannt, dass es bereits eine Patientin mit der hl7Id „061116“ gibt, die übermittelten Daten der eGK jedoch nicht mit denen unserer Datenbank übereinstimmen. Da wir angegeben haben, dass nicht übereinstimmende Daten nicht automatisch überschrieben werden sollen, wird also die folgende Nachricht zurückgegeben: „Zu Ihren Daten wurden Uneinstimmigkeiten gefunden. Bitte gehen Sie an die Rezeption und sprechen Sie unser Personal auf das Problem an“. Jane Santos wird also an das Personal weitergeleitet. Dieses kann die Daten dann bei Bedarf manuell anpassen.

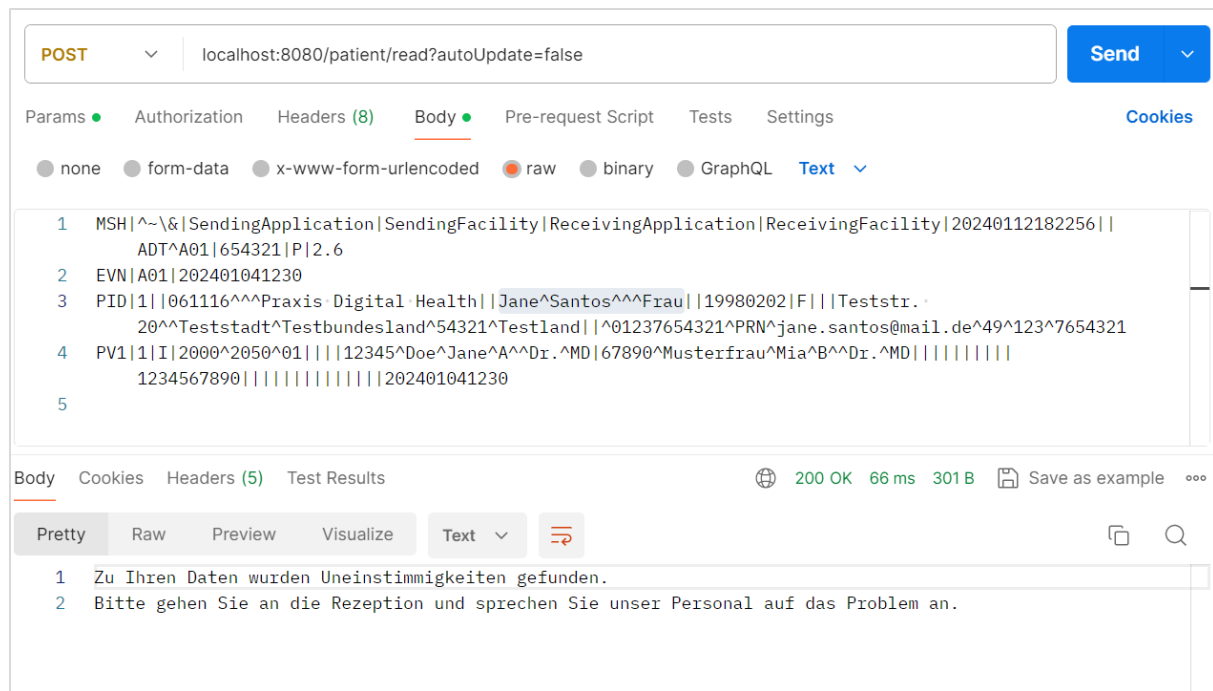


Abbildung A 10: POST-Request "/patient/read?autoUpdate=false" (Jane Santos)

A.4.5 Praxis: Duplikate in der Datenbank

Sollte der Fall auftreten, dass sich eine andere Person Namens Jane Doe mit dem selben Geburtsdatum über Doctolib für einen Termin in der Digital Health Praxis anmeldet (sh. Abbildung A 11), wird dies bei ihrem Praxisbesuch bei Einstecken ihrer Karte erkannt. Aufgrund von Mehrfacheinträgen in der Datenbank mit dem Namen Jane Doe und dem Geburtsdatum 02.02.1998, wird eine entsprechende Nachricht in der Antwort der Anfrage angezeigt, wie in Abbildung A 12 zu sehen.

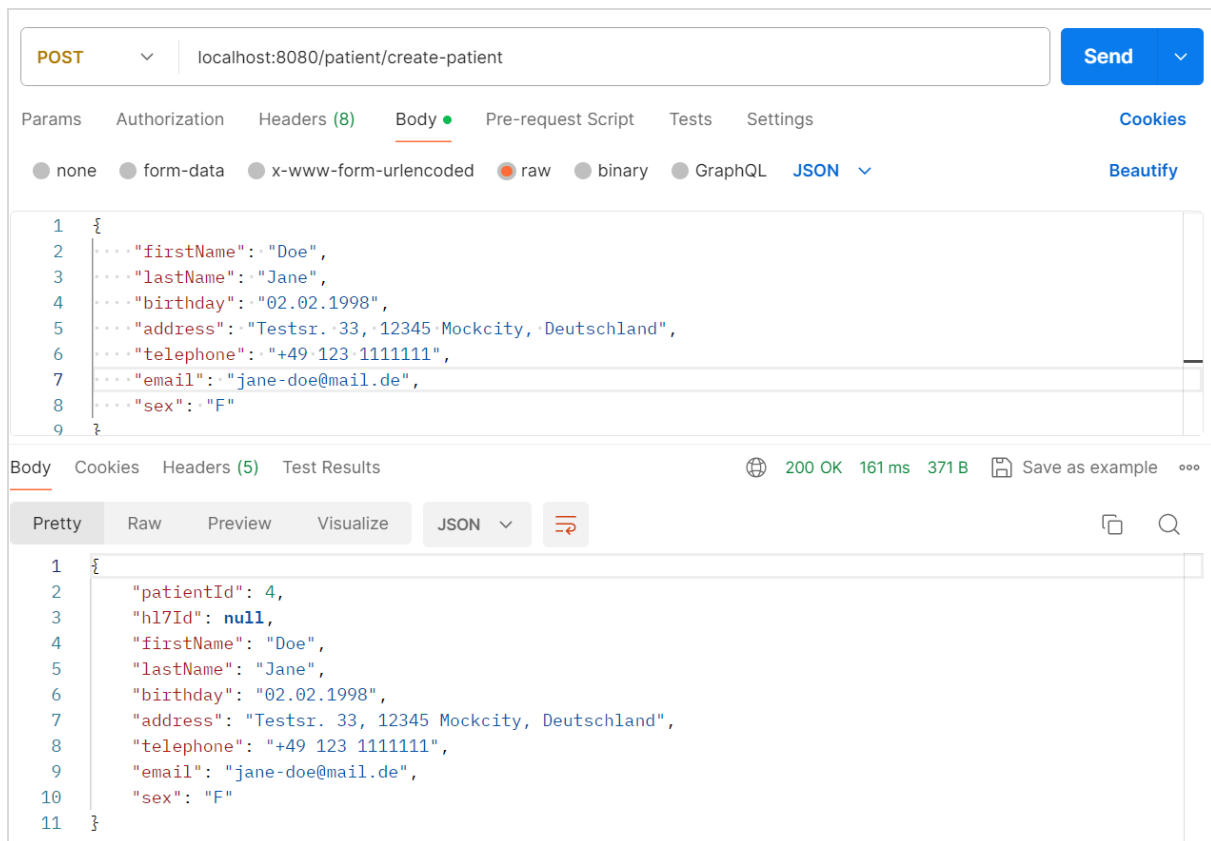


Abbildung A 11: POST-Request "/patient/create-patient" (Duplikat Jane Doe)

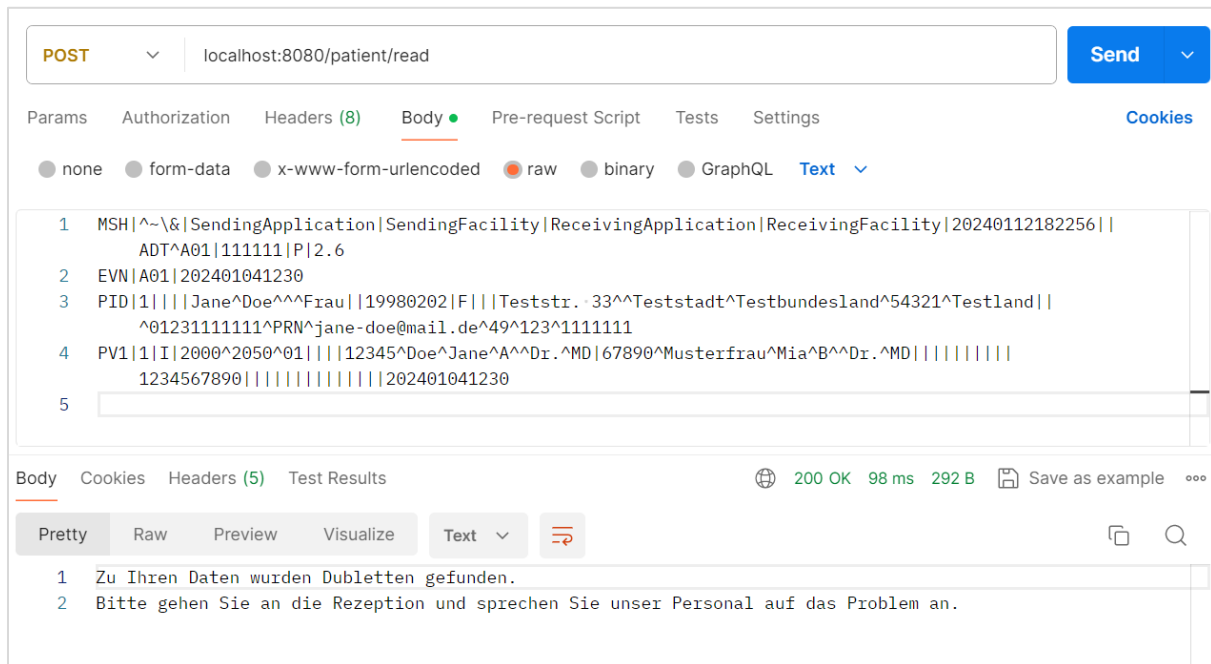


Abbildung A 12: POST-Request "/patient/read" (Duplikat Jane Doe)

A.4.6 Übersicht der Endpoints

Probieren Sie gerne selbst ein paar eigene Anfragen in der Anwendung aus. In Tabelle Tabelle A 1: REST-Endpoints der Anwendung finden Sie eine Übersicht der möglichen REST-Anfragen.

Service-Instanz	Typ	Adresse: localhost:8080/...	Anfrage-Inhalt (Request Body)	Antwort (Response Body)
HL7-Mock	GET	hl7-mock/get-random-message	-	Zufällige HL7-Nachricht aus der HL7-Mock-Datenbank
HL7-Mock	GET	hl7-mock/{id} z.B. localhost:8080/hl7-mock/123456	Bekannte MSH-ID einer HL7-Nachricht aus der HL7-Mock-Datenbank („{id}“ mit der MSH-ID ersetzen)	Nachricht zu der angegeben ID
Praxis	POST	patient/read optional: ...?autoUpdate=false (ohne diese Angabe ist autoUpdate defaultmäßig true) z.B. localhost:8080/patient/read?autoUpdate=false	HL7-Nachricht als raw Text	Patient:in als JSON-String
Praxis	GET	patient/get/{id} z.B. localhost:8080/patient/get/1	-	Patient:in mit der angegeben praxisinternen patientId (!= hl7Id)
Praxis	POST	patient/create-patient	JSON-String mit den Patient:innen-Daten	Erstelltes und in der Datenbank gespeichertes Patient:innen - Objekt
Praxis	POST	patient/{id}/update	JSON-String mit den aktualisierten Patient:innen-Daten	Aktualisiertes Patient:innen-Objekt mit den neuen Daten

Tabelle A 1: REST-Endpoints der Anwendung

A.4.7 Problembehebung beim Ausführen der Anwendung

Falls es zu Problemen beim Ausführen oder Erkunden der Anwendung kommen sollte, beachten Sie bitte die folgenden Punkte:

- Sie haben die jar-Datei ausgeführt; in der Konsole steht „Started InformmeApplication in xx.xxx seconds“
- Korrekte Angabe des Anfragetyps: GET oder POST
- Die HL7-Nachrichten haben das richtige Format. Orientieren Sie sich dafür gerne an unseren Beispielen und ersetzen Sie die Werte wie Name, Geburtsdatum, etc.. Alternativ können Sie gerne zufällige HL7-Mock-Nachrichten, mit denen wir die Anwendung erfolgreich getestet haben, verwenden (sh. Kapitel A.4.1 HL7-Mock: get-random-message
-).
- Beim Erstellen von Patient:innen: Die jeweiligen Felder haben, falls das Erstellen nicht funktioniert, eventuell den falschen Datentypen. Sehen sie sich gerne hier das Beispiel für einen JSON-String an, welcher korrekt verarbeitet wird:

```
{
  "firstName": "Doe",
  "lastName": "Jane",
  "birthday": "02.02.1998",
  "address": "Mockstreet 2, 12345 Mockcity, Deutschland",
  "telephone": "+49 123 7654321",
  "email": "jane.doe@mail.de",
  "sex": "F"
}
```

Hier ist ein JSON-String (achtung: kann so nicht vom Backend verarbeitet werden) etwas allgemeiner formuliert mit den einzugebenden Datentypen:

```
{
  "firstName": "String",
  "lastName": "String",
  "birthday": "dd.MM.yyyy",
  "address": "Straße Hausnummer, Postleitzahl Ort, Land",
  "telephone": "String",
  "email": "String",
  "sex": "A|F|M|U"
}
```

- Die Eingabe im Request-Body ist vom Typ „raw – Text“

Falls dennoch Probleme beim Erkunden und Testen der Anwendung auftreten sollten, stehen wir Ihnen gerne zur Verfügung (sh. A.5 Kontakt).

A.5 Kontakt

Falls Sie weitere Fragen zu der Anwendung haben, auf Probleme bei der Ausführung stoßen, oder unerwartete Fehlermeldungen während der Ausführung erscheinen, kontaktieren Sie gerne zu jeder Zeit unser Backend-Team:

-
- wendi.shu@stud.tu-darmstadt.de
 - jimmy.lin@stud.tu-darmstadt.de
 - clemens.hennemann@stud.tu-darmstadt.de