Spring FHIR

Aufgabe

Implementiere die Entitäten Patient und Practitioner auf eine Weise dass sie mit dem FHIR-Standard kompatibel sind. Diese FHIR-Ressoucen haben auch eine vielzahl an Unterressoucen welche ebenfalls implementiert werden müssen.

Lombok

Tipp: Um nicht immer Getter, Setter etc. in einer Klasse implementieren zu müssen, kann man mit `lombok` (bereits in dem Projekt eingebunden) sich diese im Java-Bytecode automatisch generieren lassen.

Beispiel-Klasse

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@ToString
class Something {
    int someProperty = 0;
}
```

Tipps für die Implementierung

Da sowohl `Patient` als auch `Pracitioner` Personen sind wäre es möglich eine abstrakte Basisiklasse Person zu erstellen in der allgemeine Properties bereits enthalten sind.

Da beide Controller ein sehr ähnliches Verhalten haben kann man dieses unabhängig vom Typ mittels *generics* implementieren. Für je Patient als auch Pracitioner kann dann eine Ableitung davon anderen Mappings zugeordnet sein.

```
public class PersonController<T, ...>{

    public List<T> getAllEntities(){
        ...
    }

    ...
}

@RequestMapping("/patient")
public class PatientController {

    PersonController<Patient> baseController;
```

```
    @GetMapping("/")
    public List<Patient> getAllEntities() {
        return baseController.getAllEntities();
    }

    ...
}
```

FHIR

FHIR steht für `Fast Healthcare Interoperability Resources`. Und ist ein von HL7 v eröffentlichter Standard um Daten im Gesundheitsbereich Programmübergreifend austauschen zu können. HL7 ist eine Organisation welche für die ANSI in den USA Standards im Gesundheitswesen entwickelt. Durch diesen Standard können alle möglichen Gesundheitsprogrammen Daten untereinander austauschen.

Tests

Implementiere deinen eigenen Test der die Daten auf FHIR-Kompatibilität prüft.

Du kannst prüfen, ob die von dir generierten JSON-Daten für einen Patienten passen, indem du sie mit dem Beispiel von hier vergleichst.

Erstelle dazu JSON-Daten anhand der Dokumentation und vergleiche mittels Soll- und Ist-Wert (Expected and Actual Value).

Beispiel-Test

```
@Test
public void testCompareReturnedPatientJSONtoFHIRCompliantJSON() throws Exception {
    mockMvc
        .perform(get("/patient/1")) // get patient with id 1
        .andExpect(status().isOk()) // expect 200 HTTP status code
        .andExpect(content().json('{"your_patient": "test_data"}')); // returned
data should be of type json and
        // contain the same parameters as the test data
}
```

Die initialen Testdaten sollen in der Ressourcendatei `import.sql` direkt als SQL-Statements eingefügt werden. Beim Start deines Programmes werden alle SQL-Befehle in dieser Datei ausgeführt.

Generierung von SQL-Daten

Um zu testen ob deine SQL-Statements funktionieren kannst du sie bevor du sie in `import.sql` einfügst auch manuell ausführen. Starte dazu den Webserfer und gehe dann auf die Weboberfläche der H2-Datenbank. Diese erreichst du unter: http://localhost:8080/h2-console

Wenn dein Befehl ohne Fehler ausgeführt wird, kannst du ihn bedenkenlos einfügen. Beachte aber, dass im Ausgangszustand dieses Projekts noch keine Entitäten angelegt wurden. Es ist also noch kein `CREATE TABLE`-Statement ausgeführt worden.

API-Designrichtlinien

Bitte beachte bei den HTTP-Requestmethoden den dazugehörigen RFC 7231 Sektion 4.3

Kurz zusammengefasst, wann man was nimmt:

GET	Wenn man eine Ressource holt
POST	Wenn man eine nicht identifizierte Ressource hochlädt oder ändert
PUT	Wenn man eine identifizierte Ressource hochlädt
PATCH	Wenn man eine identifizierte Ressource teilweise ändert. Hat einen extra RFC, siehe: RFC 5789
DELETE	Wenn man eine Ressource löscht

Auch gibt es einen schönen Blogartikel von Stackoverflow in dem gutes API-Design erklärt wird. Blogartikel: Best practises for REST API Design

Native Queries

Um komplexere Abfragen jeglicher Art durchführen kann man direkt SQL einsetzen. In Spring gibt es die Möglichkeit mit der @Query Annotation Abfragen direkt zu definieren. Prüfe vor der Query-Verwendung in der H2-Console ob die Abfragen auch richtig sind.

Gib jeder Funktion in einem Repository diese Annotation mit der entsprechenden Abfrage. Eine Anleitung wie man diese Aufbaut findest du hier.