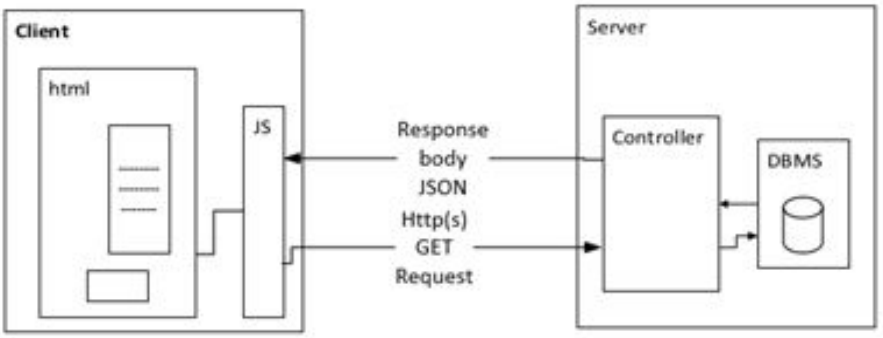
Fragensammlung zur Matura (extramural)

1. *Beschreibe die Architektur unseres APIS?*



- Server Client Architektur
  - Bei unserem Beispiel wird die REST-Schnitte miteinbezogen. (REST steht für *Representational State Transfe*r)
- Beschreibung:

  Die Server-Client Architektur besteht aus zwei wesentlichen Komponenten: Dem Client (beispielsweise ein Laptop öffnet unsere Webseite mittels einem Internet-Explorer), sowie dem Server, welcher für die Datenverarbeitung zuständig ist.

  Server

  Der Server besteht in diesem Fall aus einem Controller, sowie einem sogenannten Datenbankmanagementsystem (kurz *DBMS*). Diese arbeiten sozusagen “Hand in Hand”. Der Controller ist für die “präsentation” oder anders gesagt für die Übergabe und Aufbereitung der Datensätze zuständ. Das DBMS kümmert sich um das bereitstellen der Datensätze.

  Client

  Der Client bzw. die Client-Ansicht wird in diesem Fall mittels einem Internet-Browser möglich gemacht. Dieser verarbeitet HTML- sowie JavaScript-Code in Kombination. Der Javascript-Code ist für die Aktionen bzw. Funktionen im Browser zuständig, nachdem HTML-Code nur die Präsentation der Inhalte durchführt. Die Inhalte des HTML-Codes werden mittels JSON-Dateien übergeben, nachdem das regelmäßige Senden des gesamten Codes zu aufwendig wäre. Rückmeldungen bzw. Rückgabewerte des Clients

werden mittels HTTP(s)-GET-Requests übergeben. Diese werden mittels dem JavaScript-Code generiert. Die HTML-Dateien, welche im Client präsentiert werden, werden mittels TypeScript bzw. Angular generiert.

- Wir haben einen Server und einen Client (Angular)
- Server:
  - Module, Repository, Ressourcen, TOM Cat, SpringBoot, JPA, Datenbank, Tests, Controller
- Client:
  - Angular, Routing, Services (ts), Components (html, ts), Typescript wird zu Java umgewandelt, HTML, CSS

Genauere Erklärung des DOM:

Document Object Model (DOM, engl. für Dokumenten-Objekt-Modell)

- ist eine Spezifikation einer Programmierschnittstelle, welche HTML- oder XML-Dokumente als eine Baumstruktur darstellt, in der jeder Knoten ein Objekt ist, welches einen Teil des Dokumentes repräsentiert, z. B. einen Absatz, eine Überschrift, ein Video oder etwa eine Tabellenzelle. Die Schnittstelle ist plattform- und programmiersprachenunabhängig und erlaubt damit standardisiert, die Struktur und das Layout eines Dokumentes zu verändern.
- Im Webbrowser bildet dies einen wichtigen Baustein für dynamische Webseiten.
- In unserem Fall manipulieren wir mittels TypeScript den DOM, damit wir Inhalte im HTML-Code präsentieren können. Ein wesentlicher Anwendungsfall ist das Erstellen der Listen mittels TypeScript. Dadurch wird in den bereits geschriebenen HTML-Code eingegriffen, um die einzelnen Einträge zu erstellen.

HTTP Protokolle

Im HTTP(s)-Protokoll ((sicheres) Hypertext Transfer Protocol) gibt es verschiedene Anfragemethoden (englisch: request methods), die es dem Browser ermöglichen, Informationen, Formulare oder Dateien an den Server zu senden.

Wahl der Anfragemethode

Die Wahl der Übertragungsmethode hängt, will man es richtig machen, nicht etwa von der Präferenz des Programmierers ab, sondern folgt eigentlich ganz einfachen Regeln:

7. Werden durch den Request lediglich andere Daten als Antwort empfangen, so ist die GET-Methode die richtige Wahl.
8. Werden durch den Request Daten auf dem Server verändert, ist die [POST]-Methode die richtige Wahl.
9. Werden Daten für Logins, insbesondere Passwörter übermittelt, dann ist **nur** POST die einzig richtige Wahl.

GET

Mit der **GET**-Methode können Sie eine Ressource (zum Beispiel eine Datei) vom Server anfordern. Dabei wird ein Parameter (z. B. übertragene Formulardaten), getrennt durch ein Fragezeichen, zum URI hinzugefügt.

POST

Mit der **POST** -Methode können Sie große Datenmengen (wie Bilder oder HTML-Formular-Daten) zur weiteren Verarbeitung zum Server senden.

---

2. *Liste die Technologien auf, die am Server zum Einsatz kommen. Beschreibe jeweils aus welchen Komponenten diese jeweils aufgebaut sind.* (Mihai/Leon)

- **Tomcat**: ist ein Open Source *Webserver* auf dem in der Sprache Java geschriebene Web-Anwendungen ausgeführt werden.
- **POM Datei**: Das ist eine *Konfigurationsdatei für Maven Projekte* oder auch “Bauanleitung für die Applikation”. Speichert die Informationen eines Softwareprojekts werden, in XML-Format. Es bietet eine einfache Möglichkeit, um weitere Libraries in die eigene Applikation einzubinden. Datei: pom.xml
- **JPA**: JPA ist ein *Standard*, steht für Java Persistance API und ermöglicht das Speichern von Java Objekten in die Datenbank. Ohne JPA müssten wir etliche SQL Statements schreiben, um unser Objektmodell (Java) in das relationale Modell (Datenbank) abbilden zu können. JPA ist dabei eine Open Source Spezifikation und wird z.B. von Hibernate, Eclipselink, Toplink, Spring Data JPA implementiert. Wir benutzen es als Annotations (@). Starten wir die

Applikation werden abhängig von den Einstellungen in application.properties (create) die Tabellen in der Datenbank neu erstellt.

- **Hibernate**: Ist ein Open-Source *ORM-Framework für Java*. Hibernates Hauptaufgabe ist die objektrelationale Abbildung. Dies ermöglicht es, gewöhnliche Objekte mit Attributen und Methoden in relationalen Datenbanken zu speichern und aus entsprechenden Datensätzen wiederum Objekte zu erzeugen. Beziehungen zwischen Objekten werden auf entsprechende Datenbank-Relationen abgebildet. → implementiert JPA → Datenbankanbindung ohne "richtigen" SQL Code.
- **Spring Boot:** Spring Boot ist ein *Framework für Java-Plattformen* und wird genutzt um RESTful Webservices zu erstellen. Damit können wir ebenfalls mit mobilen Clients auf das Service zugreifen oder von einer Webseite, um dynamische Inhalte zu laden. Es ermöglicht uns eigenständige Spring Applikationen zu erstellen → .jar Datei, die direkt ausgeführt werden kann. Der Webserver muss nicht extra gestartet und die Applikation darauf deployed werden, Spring Tomcat eingebettet hat und beim Starten der Applikation gleich ausführt. Es besitzt eine Voreingestellte POM Datei, um die Maven Konfiguration zu erleichtern.

- **Datenbank**: Ist der Speicherort für alle Daten. Der Server kommuniziert über JPA mit der Datenbank. Die Datenbank sollte sich in einem separaten abgesicherten Netzwerk befinden, damit die Daten nicht gestohlen/manipuliert/gelöscht werden können. Des weiteren sollten Brute Force Attacken mit einem Sicheren Passwort gesichert werden. Das Passwort befinden sich in unserem Projekt unter application.properties. Wie bereits erwähnt wird auf die Datenbank nicht direkt vom Client zugegriffen, dazwischen befindet sich die Business- oder Application Logic, welche die benötigten Daten liefert. → Client schickt Request → Business Logic (Controller) verarbeitet es → greift bei bedarf (meistens der Fall) auf die Datenbank zu → verarbeitet Daten → Daten an den Client mittels Service als JSON.

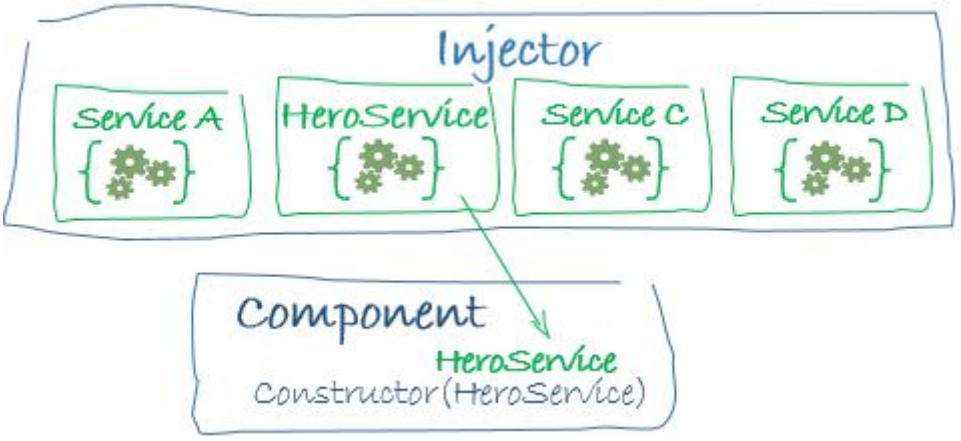
- ***Modellklasse***: (*Persistenzschicht*) Aus den Modellklassen wird mit hilfe der JPA-Annotationen ein passendes Datenbankschema erstellt (muss es aber nicht, falls eine alte Datenbank existiert, welche nicht dem FHIR-Standard entspricht). Die Model Klassen ermöglichen es uns, das Laden und Speichern von den Objekten in die Datenbank, leicht umzusetzen. Man spricht von "Objekten persistieren". Ohne JPA müssten wir die Java Klassen und die

Datenbank erstellen. Dabei ist es sehr wichtig auf die Annotationen der Relationen und der Vererbungshierarchie zu achten.

- ***Repository***: (*Persistenzschicht*) Diese Komponente greift auf die Datenbank zu. Wir benötigen für jede Modellklasse, mit denen der Client interagiert (z.B. Patient), ein Repository. Es erstellt automatisch die CRUD (Create, Read, Update, Delete) Befehle für die Datenbank. Die Repositorys ermöglichen uns ohne SQL-Skripts zu codieren und verhindert somit SQL-Injections.
- ***Test***: (*Test der Persistenzschicht*) Um die korrekte Implementierung unsere Persistenzschicht (Repositorys & Modellklassen) zu testen verwenden wir automatisierte Tests. Es wird ein Objekt erstellt und über die Repositories in der Datenbank gespeichert. Der Gespeicherte Inhalt wird geladen und mit dem Originalobjekt verglichen. Wenn beide gleich sind passt alles.
- ***Converter***: Konvertiert die von der Datenbank erhaltenen Daten, dass sie sie dem FHIR Standard entsprechen. Beispiel Set<String>: um nicht extra eine Tabelle mit Strings auf der Datenbank zu erstellen, werden diese in einem “string” Feld gespeichert. Beim Aufrufen von der Datenbank wird der String in ein Set<String> konvertiert.
- ***Controller***: Dies ist die *Schnittstelle* zur Außenwelt (Clients) unserer Applikation. Diest ist unsere REST Schnittstelle zu unserem Client. Er verarbeitet die Requests des Clients. Spring Boot bietet hierfür Controller (MVC, Model View Controller) an. Ein Controller verarbeitet HTTP Anfragen wie Get, Post, Put, Delete. Über das Repository, greift es auf die Datenbank zu, verarbeitet die Daten und schickt die verarbeiteten Daten als JSON an den Client.

3. *Liste die Technologien auf, die am Client zum Einsatz kommen. Beschreibe jeweils aus welchen Komponenten diese jeweils aufgebaut sind.* (Sara/Hami)
    - Client: Angular Framework arbeitet mit Typescript, das zu JavaScript umgewandelt wird, HTML (Dokument Inhalt strukturiert) , CSS (Formatierung) wird aufbereitet. fehlt etwas?
    - Komponenten:
        - Services: durch Dependency Injection wird den Komponenten der Service “injiziert”. Der große Vorteil ist dabei eine geordnete Struktur des Codes zu erreichen ohne “doppelten Code”. Ein einziger Service kann von mehreren Komponenten

übernommen werden.



Beispiel: Dataservice enthaltet Methoden zur Datenmanipulation (createPatient,updatePatient), welche durch eine Instanz dieses “Services” im Konstruktor übergeben wird.

=> Bitte verständlich formulieren und Daseinsberechtigung von Services erklären

- Routing: Es ist üblich das Routing durch eine separate Module Klasse zu erzeugen. *app-routing.module.ts.* Wenn etwas angeklickt wird oder eine URL eingegeben wird, kommt es beim APIS über Routing an die lokale Komponente. Der GET Request wird dabei zuerst lokal abgefangen und bearbeitet. Der Server wird erst angesprochen, falls lokal kein Pfad vorhanden ist. Die Seite kann aktualisiert werden, ohne den Server zu kontaktieren.

  Eine typische Route hat zwei Eigenschaften:

  1. path: Eine Zeichenfolge, die mit der URL in der Adressleiste des Browsers übereinstimmt.
  2. component: Die Komponente, die der Router beim Navigieren zu dieser Route erstellen soll.

  Routen teilen dem Router mit, welche Ansicht angezeigt werden soll, wenn ein Benutzer auf einen Link klickt oder eine URL in die Adressleiste des Browsers einfügt. Durch Importieren der Komponenten gibt man einen Ort an auf die der Router zugreifen soll.

1. Eine URL wird in die Adressbar eingegeben => Passende Seite wird aufgerufen
2. Links werden angeklickt => Navigation zur Seite
3. Back & Forward Buttons werden unterstützt

Die Datenübertragung aus dem Server und lokal kommt durch HTTP Anforderungen zu stande.

- Model: FHIR-Standard (JSON) wird gespiegelt. Der Client bekommt nur die REST Schnittstelle vom Server mit. Die Datenbankstruktur ist nicht relevant für den Client.

  Beispiel:

```
export class PatientModel {
    constructor(
        public name : [ HumanName ], // A name
associated with the patient
    ) {}
}

export class HumanName{
  [x: string]: any;
    public id: string = ''
        public use: string = ''
        public text: string = ''
        public family: string = ''
    constructor(
        id: string = '',
        use: string = '',
        text: string = '',
        family: string = ''
    ){this.id=id;this.use=use;this.family=family,
this.text=text}
}
```

  Der Patient hat ein Attribut “name” mit dem Datentypen HumanName, welcher gleich aufgebaut ist wie am Server.

- NgModules: Auch als *Funktionmodule* bekannt.

  NgModule-Metadaten: => haben wir das wirklich besprochen?

  1. Gibt an, welche Komponenten, Anweisungen und Pipes zum Modul gehören.
  2. Macht einige dieser Komponenten, Anweisungen und Pipes öffentlich, damit die Komponenten Template anderer Module sie verwenden können.

3. Importiert andere Module mit den Komponenten, Anweisungen und Pipes, die Komponenten im aktuellen Modul benötigen.
4. Bietet Dienste, die die anderen Anwendungskomponenten verwenden können.

Jede Angular-App verfügt über mindestens ein Modul, das Root-Modul. Sie booten dieses Modul, um die Anwendung zu starten.

Das Root-Modul ist alles, was Sie in einer einfachen Anwendung mit wenigen Komponenten benötigen. Wenn die App wächst, überarbeiten Sie das Root-Modul in Feature-Module , die Sammlungen verwandter Funktionen darstellen. Anschließend importieren Sie diese Module in das Root-Modul.

- Direktives: verbindet Applikationsdaten mit dem DOM. Wir verwenden die NgFor-Direktive, um ein Array von Elementen zu durchlaufen und mehrere Elemente dynamisch zu erstellen aus einem template Element.

Das template ist das Element, an dem die Direktive angehängt ist.

Wir können mehrere NgFor-Direktivs zusammenschachteln.

Wir können den Index des Elements abrufen, über das wir eine Schleife durchführen, indem wir einer Variablen im NgFor einen Index zuweisen.

=> Zusammenhang?

- Pipes: Ausgabe der Outputs im speziellen Format. Beispiel: Datumsformat. JavaSrcipt Daten werden passend formatiert für die HTML

  => Details |

4. *Zähle auf und beschreibe, welche Protokolle verwendet werden, um Daten zwischen dem Server und dem Client zu übertragen.* (Aida/Samra)
   - TCP/IP (Transmission Control Protocol/Internet Protocol)
   - HTTPS/HTTP

HTTP Method	CRUD	Entire Collection (e.g. /users)	Specific Item (e.g. /users/123)
POST	Create	201 (Created), ‘Location’ header with link to /users/{id} containing new ID.	Avoid using POST on single resource
GET	Read	200 (OK), list of users. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single user. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method not allowed), unless you want to update every resource in the entire collection of resource.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.

PATCH	Partial Update/Modify	405 (Method not allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method not allowed), unless you want to delete the whole collection — use with caution.	200 (OK). 404 (Not Found), if ID not found or invalid.

RESTful HTTP(s)-Verben aus https://restfulapi.net/http-methods/

Die Übertragung verläuft mittels CRUD (CREATE, READ, UPDATE, DELETE) Befehlen => HTTP mit Verben

- CRUD ist ein Akronym für die vier fundamentalen Operationen des Datenmanagement:
  i. Create (Insert): Neue Daten erstellen.
  ii. Read (Select, Retrieve. Search): Bestehende Daten selektieren und zur weiteren Verarbeitung bereit stellen.
  iii. Update (Modify): Bestehende Daten aktualisieren.
  iv. Delete' (Destroy): Veraltete Daten löschen.
- Oftmals handelt es sich bei der grafischen Benutzeroberfläche eines CRUD-Frameworks um ein simples HTML-Interface. Typischerweise berücksichtigt das CRUD-Framework einzelne Transaktionsschritte. Dies hat zur Folge, dass Daten nur gespeichert werden, wenn innerhalb der HTML-Oberfläche der Speichern- bzw. Weiter-Button gedrückt wurde. Ist dies der Fall, so wird letztlich die Update-Operation ausgeführt.

- Das CRUD-Framework weist selbstverständlich ein äquivalentes Verhalten für die verbleibenden CRUD-Operationen auf. Es handelt sich bei einer CRUD-Operation folglich um einen atomaren Vorgang.
- Atomare Operationen sind in diesem Zusammenhang von Interesse, da moderne Software-Anwendungen oftmals als Mehrbenutzersystem realisiert werden. Ein CRUD-Framework erlaubt Lesen und Schreiben eines Datensatzes auch dann, wenn beide Operationen zeitlich stark versetzt erfolgen. Trotzdem ist es anderen Personen gestattet, während dieser Zeit denselben Datensatz auszulesen. Folglich wurde der Datensatz nicht gesperrt.

5. *Beschreibe den Ablauf der Übertragung der Daten zwischen Client und Server*. (Alfons/Elia)
    - Kurze Antwort: Server: Controller, Client: DataService

    - Lange Antwort:
    - Am Client wird durch eine Aktion (z.B.: Klick von User) eine Request an den Server geschickt. http Request = im Header http-Verben mit Parameter.
    - Server hat in unserm Fall Controller
    - Parameter in http-header bestimmen welcher Controller die Request bearbeitet (z.B.: ein Put für Patient mit id 4 würde der PatientenController bearbeiten)
    - Controller „ließt“ Request und „übersetzt für Repoitory“.
    - Repository greift auf datenbank zu und führt den Befehl von Controller aus
    - Repository liefert Controller daten aus datenbank zurück
    - Controller nimmt Daten und formt http-Response
    - http-Response von Controller (mit Json im Body) wird an Client geschickt
    - Dataservice am Client empfängt response
    - Je nachdem was der Befehl von Client wollte passiert mit diesen Daten

    - *Request geht nach einem Klick auf ein Objekt in den Server ein. Controller verarbeitet den Request.*
    - *Repository holt die Daten aus DB und formuliert HTTP Response(JSON im Body)*
    - *HTTP Response geht beim Client in den Data Service ein und die Daten werden angezeigt (Je nachdem was der Benutzer geklickt hat.)*

6. *Welche Komponenten sind am Server und am Client für den Datenaustausch zuständig? Beschreibe jeweils die Funktionalität der Komponente.*(Aynur/Tamara)

    - Client: DataService

        - Im DataService wird die URL der jeweiligen Ressource angegeben, wo dann ein http Request mit http - Verb an den Server geschickt werden kann.
    - Server: Controller
        - Der Controller empfängt/bearbeitet die Requests vom Client und schickt sie ggf. über das Repository an die Datenbank weiter.
        - Im Controller wird ein JSON erstellt welches über ein http-Response an den Client (Java Script) geschickt wird. Die Kommunikation zwischen Client und Server verläuft asynchron. Dies bedeutet, dass ohne Blockieren des Prozesses durch bspw. Warten auf die Antwort des Empfängers (wie bei synchroner Kommunikation der Fall) der Austausch stattfindet.

7. *Welche Datenformate werden zwischen Client und Server ausgetauscht? Beschreibe examplarisch den Aufbau eines der Dateneinheiten.*(Basti/Vinz)
    - Kommunikation läuft über HTTP(s)
        i. Client sendet den Request (Create, Read, Update oder Delete)
        ii. Server antwortet mit http Response mit JSON, XML oder nur HTML im body
    - Aufbau Request
        i. Header: CRUD operation, Absender, Format etc., http version
        ii. Body: Zusatzinformationen optional z.B. bei einem Post Befehl die abzudatenden Daten (im JSON oder XML Format)
    - Aufbau Response
        i. Header: HTTP Status, http version ...
        ii. Body: JSON, XML Daten im Body

    Link: https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

    => Links sind keine gute Idee

8. *Welche http Endpoints gibt es am Server? Liste diese mit den notwendigen Kriterien tabellarisch auf.* (Daniel/Joseph)

    Was sind Endpoints:
    Ein Endpunkt ist das Ende eines Kommunikationskanals. Wenn eine API mit einem anderen System interagiert, gelten die Berührungspunkte dieser Kommunikation als Endpunkte. Oft erfolgt der Zugriff über eine URL, an welche HTTP-Anforderungen gesendet werden und eine Antwort erwartet

wird. Endpunkte geben an, wo Ressourcen liegen, auf die andere Systeme zugreifen können.

Bei unserem Server:

URL	GET	POST
/api/<Resource>/	Alle Datensätze der Resource auslesen oder mit Angabe der ID der Resource: /id -> Einen bestimmten Datensatz der Resource auslesen	Einen Datensatz einer Ressource hinzufügen

PUT	DELETE
Einen Datensatz einer Ressource aktualisieren/updaten (mit Angabe der ID -> /id)	Einen Datensatz einer Ressource löschen (mit Angabe der ID -> /id)

- /api/patient
- /api/bodystructure
- /api/encounter
- /api/observation
- /api/practitioner

9. *Beschreibe den Zusammenhang zwischen dem MVC Konzept und den verwendeten Komponenten am Server. Was würde dem M, was dem V und was dem C entsprechen?* (Eldar/Hannes)
   - MVC (Model View Controller) ist ein Designkonzept für Applikationen
     - M ... Models (Datenbank)
     - V ... View (Client)
     - C ... Controller (Controller Server, Business Logik)

10. *Beschreibe die Aufgabe von FHIR im medizinischen Umfeld.*(Felicity/Sema)

- Das Gesundheitswesen basiert auf Kommunikation und Datenaustausch zwischen den verschiedenen Gesundheitsdienste-Anbietern. Damit diese Kommunikation funktioniert und keine Komplikationen aufweist, braucht man Kommunikationsstandards.
- FHIR ist ein Standard von HL7 und definiert eine Datenstruktur + Datenaustausch => auch beschreiben siehe Frage 7, die bei der Implementierung von Software eingehalten werden muss. Dieser Standard ermöglicht es, medizinische Daten und Abläufe, trotz unterschiedlicher Systeme einheitlich zu übertragen und auszutauschen.
- Die Daten werden als sogenannte Ressourcen gespeichert und ausgetauscht und die einzelnen Ressourcen haben Attribute. Die Ressourcen können miteinander verknüpft sein.
- Diese Systeme sind meist von verschiedenen Entwicklern und Anbietern, damit die verschiedenen Softwaresysteme trotzdem miteinander kommunizieren können und Daten austauschen können, müssen sie kompatibel aufgebaut sein. Deshalb braucht man Standards, um sicherzustellen, dass trotz unterschiedlicher Systeme die Daten ausgetauscht werden können.

11. *Welche Protokolle gibt es, die ähnlich wie FHIR sind? Argumentiere, warum FHIR statt diesen verwendet wird.*(Kristina/Nesrin)

    FHIR Alternativen

    - HL7 v2
        - Datenaustausch syntaktisch & teils semantisch definiert
        - Dient vorwiegend dem Datenaustausch innerhalb eines Krankenhauses zwischen KIS, LIS, RIS
        - V2 Nachricht besteht aus mehreren Segmenten
        - Segmente bestehen aus Feldern mit Datentyp & Bedeutung (Semantik)
    - HL7 v3
        - XML basiert
        - Neuer Ansatz: RIM Reference Information Model
            - Davon leiten sich nach Regeln Domäneninformationsmodelle ab.
            - Beispiele für Domänen Informationsmodelle: Patientenaufnahme, Terminvergabe, Verschreibung von Medikamenten, etc.

        - Daraus ergeben Strukturen für Dokumente und Nachrichten, die ausgetauscht werden können
        - Nachrichten und Dokumente können auf Gültigkeit geprüft werden
            - XML-Schema
            - Schematron
        - XML-Schema und Schematron werden ebenfalls aus den Modellen generiert und von HL7 veröffentlicht
        - V3 löst derzeit v2 für Kommunikation in KH nicht ab
- HL7 v3 - CDA (Clinical Document Architecture)
    - Basiert auf RIM
    - Standard für Datenaustausch zwischen Krankenhäusern, dem niedergelassenen Bereich, etc.
    - Besteht aus
        - Header mit Identifikationsdaten zu Patienten, Behandelndem, Verwaltungsinformationen
        - und Body mit den tatsächlichen Daten

Warum FHIR? (Siehe Frage 13)

- FHIR eignet sich für den Datenaustausch besser => Was ist “besser”
    - HL7 v2 → nur vorwiegend für den Datenaustausch zwischen KIS, LIS und RIS ist, in einem Krankenhaus aber es mehr IS gibt.
    - HL7 v3 CDA → ist für die Kommunikation zwischen KH und externen IS zuständig
    - HL7 v3 → verwendet für den Datenaustausch nur XML, was sehr unpraktisch ist (schwer zu lesen) JSON eignet sich da besser.
- Des Weiteren auch besser für den Anwendungsbereich:
    - Innerhalb einer Organisationseinheit (z.B. APIS)
    - zwischen Organisationseinheiten (APIS, KIS, LIS, RIS)
    - zwischen mobiler Anwendung und Organisationseinheit
    - Social Web (Patienten Interaktion)

→ FHIR beinhaltet alles in einem, was die Performance von Informationssystemen steigert.

xcv=> Generelle Behauptungen, ohne Argumente. Vergleichen von Vor- und Nachteilen, Auf FHIR Webseite nachlesen

12. *Argumentiere, welche Elemente unserer Applikation dem FHIR Standard entsprechen müssen. Warum genau diese und welche nicht? (Z.B. Datenbank, Übertragene Daten zwischen Client, Server, Oberfläche, Modelklassen, ...)* (Lukas/Friedrich)
    - Die Schnittstelle am Server muss dem FHIR-Standard entsprechen und wird durch den Controller umgesetzt. Die Modellklassen definieren einerseits die DB-Struktur, andererseits die JSON Struktur. Die JSON Struktur muss genau dem FHIR Standard entsprechen, das DB Schema kann davon abweichen. Auch bei den Wertebereichen der Daten gibt FHIR Vorgaben, die z.B. bei der import.sql beachtet werden müssen. Bei der Oberfläche ist die Implementierung des Standards nur implizit durch die dahinterliegende Struktur des FHIR Standards gegeben, aber hier kann auch variiert werden.
      Die http(s) Kommunikation ist vorgegeben, die Struktur kann JSON oder XML sein.

13. *Argumentiere warum es sinnvoll ist, den FHIR Standard einzusetzen.* (Konrad/Katharina)
    - Der FHIR Standard ermöglicht die Kompatibilität zwischen Systemen im medizinischen Bereich. Im Gesundheitssystem sollte es den Datenaustausch einfach ermöglichen.
    - Einfache Implementierung durch standardisierte Webtechnologien
    - Zukunftsrelevanter Standard, weil neu
    - Einfache Umsetzung mobiler Clients Patienten Apps

14. *Beschreibe, welche Anwendungen durch FHIR für Endanwender (Patienten) ermöglicht werden?*(Noah/Boo)
    - FHIR gibt eine Alternative zu dokumenten zentrierten Ansätzen, in dem es den direkten Zugriff auf einzelne Informationsfelder als Service zulässt. Ein wesentliches Ziel von FHIR ist es, Gesundheitsdaten auch auf mobilen Endgeräten wie Tablet und Smartphone verarbeiten zu können und diese auf einfache Art und Weise in existierende Systeme einzubinden.
    - Der FHIR Standard wird verwendet, um dem Patienten die Möglichkeit zu geben auf deren Patientendaten zugreifen zu können
    - Patientendaten aufzeichnen und dem Arzt zur Verfügung stellen

- Der Patient kann mit einem Internetfähigen Device auf seine Gesundheitsdaten zugreifen, spricht der Patient kann jeder Zeit seine Daten aufrufen oder teilen.

15. *Zähle zwei Beispiele für einen sinnvollen Einsatz von FHIR auf. Wo wird dabei FHIR eingesetzt?*(Pati/Merna)
    - Einrichtungs-interne Interoperabilität => Beschreibung überall
    - Einrichtungsübergreifende Kommunikation
    - Abrechnungsrelevante Daten für die Krankenkasse
    - Regionale und nationale Netzwerke z.B.: ELGA, KIS, RIS?
    - Mobile Applikationen
    - Social Web (Patienten-Interaktion)

16. *Analysiere, wie unsere Applikation serverseitig abgesichert ist?* (Niki/Jakub)
    - Die REST Schnittstelle ist ungesichert
    - DB und Server im gleichen Netz - nicht sicher
    - DB nur mit Passwort gesichert
    - Gegen SQL Injections geschützt - weil keine SQL-Statements - Repository

17. *Welche Angriffsmöglichkeiten gibt es und welche Schwachstellen haben wir am Server?*
    - Schwachstellen:
        - kein HTTPS zwischen Client und Server (Daten sind nicht verschlüsselt → jeder kann mitlesen) → Sicherheitslücke (DB Verbindung ist verschlüsselt)
        - kein Login
        - Daten manipulierbar über REST Schnittstelle
    - Angriffsmöglichkeiten:
        - Datenmanipulation über REST Schnittstelle
        - Zugriff auf DB über geratenes PW (Bruteforce)
        - SQL-Injection → wir haben keine SQL-Injection weil wir keine SQL-Statements schreiben → das ist gut

18. *Welche Maßnahmen wären sinnvoll, um unsere Applikation abzusichern?* (Katja/Bernhard)
    - HTTPs - Zertifikat zw. Client und Server
    - Zertifikate zur DB
    - Authentifizierung und Autorisierung implementieren
    - Sicheres PW für die DB
    - Verschiedene Netzwerke zw. DB und Server
    - Netzwerksicherheit z.B. reverse Proxy siehe Unterricht Hoheiser
    - Hibernate wird bereits verwendet
      Bei der Verwendung von Hibernate hat man die Möglichkeit Platzhalter für die Eingabewerte zu verwenden. Diese werden dann beim Ausführen oder Vorbereiten der Abfrage mit den Werten befüllt und blockieren mögliche Manipulationen am SQL-Code.
    - Gegen physischen Zugriff schützen
    - … Inhalte aus MIS int