De student is bekend met het concept van de Application Programming Interface, stateful en stateless communicatieprotocollen en kan op grond daarvan een client-server applicatie ontwerpen en realiseren. (*Canvas - O-S-OOS3-O-S31*)

API en Communicatieprotocollen

Spring Boot Microservice

Om een beter idee te krijgen bij microservice applicaties en API's heb ik een youtube cursus gevolgd van JavaBrains. In deze cursus word uitgelegd hoe je verschillende Spring Boot applicaties met elkaar laat communiceren door middel van een eureka server. Zie ook de reposetory van JavaBrains op gitHub.

Uitleg Spring Boot microservice

Bij het starten van de springboot applicatie liep ik tegen de onderstaande foutmelding aan. Deze is op te lossen door Java versie 15 te gebruiken ipv versie 16.

Fix foutmelding: Could not initialize class org.jetbrains.jps.builders.JpsBuildBundle

Links naar de aangemaakte service applicaties:
Catalog Info Service
Movie Info Service
Raitings Data Service
Eureka Server

Handige links:
Spring initializer

Stateful communication protocol

#Todo:

- [ ] osi model

In eerdere research heb ik de verschillen bekeken tussen stateful en stateless communication. De informatie die je hier over vind ik het internet is makkelijk te verwarren met de communicatie protocollen van stateful en stateless.

in het hoofdstuk Stateful vs. Stateless omschrijf ik de verschillende fuctionaliteiten binnen de applicaties. Dit is wat anders dan het communicatie protocol wat je kan gebruiken voor een web api.

Wat is Stateful communication?

Stateful communication houdt in dat wanneer er een communicatie word opgezet dat deze constant intact word gehouden. Je hebt dus een langere sessie waarin je verschillende commands kan doen zonder iedere keer opnieuw verbinding te moeten maken.
Voorbeeld:
Een wifi-verbinding is een stateful communication. Deze is constant actief.

Wat is Stateless communication?

Stateless communication is precies het tegenover gestelde. Voor iedere request die er word gedaan word er een nieuwe verbindng opgezet informatie over en weer gestuurd en vervolgens word de verbinding verbroken.

Voorbeeld:
Een website opvragen via een browser is een stateless communication. Dit word opgehaald en daarna de verbinding met de server verbroken.

Todo: Statefull communication protocol (websockets)

Stateful vs. Stateless

"STATEFULL_VS_STATELESS.JPG" is not created yet. Click to create.

Statefull:

- Slaat de state/sessie op
- Door de state op te slaan zijn er minder calls nodig naar de Database
- Wanneer er een call word gemaakt naar een andere server dan moer de state opnieuw gevuld worden
- minder makkelijk op te schalen

Stateless:

- Word geen state opgeslagen
- communicatie 2 kanten op
- server communiceert bij iedere call met de DB voor verificatie
- makkelijk op te schalen

Wat is REST (REpresentational State Transfer).

- concepts:
    - Seperation of Client and Server
    - Server Requests ase Stateless
    - Cacheable Requests
    - Uniform Interface
- Problems
    - Too difficult to be qualified as "REST"
    - Dogma of REST vs. Pragmatism
        - Structured architectural style
        - The need to be productive

Hoe word REST gebruikt om een API te maken

Ontwerpen van URI Endpoints

Gebruik maken van Verbs en nouns

Waarom zijn "association APIs" belangrijk

Wat voor rol heeft een "operational API"

Waarom is versiebeheer belangrijk voor een API

Waarom beveiligen van je API

HTTP

Hoe werkt HTTP

Request - response
Stateless: er word een "request" gestuurd, de server ontvangt deze en stuurt een "response" terug. Hier na word de verbinding direct verbroken. Door deze methode kan je makkelijk veel requests managen. De connectie met de server is erg kort.

Verb	Get	Post	Put	Patch	Delete	En meer
Headers		Content type	Content length	Authorisatie	Acceptatie	Cookies
content	HTML, CSS, Javascript, XML, JSON	Niet alles is met een verb te combineren				

Status code

status code	
100-199	Information
200-299	Succes
300-399	Redirection
400-499	Client Errors
500-599	Server Errors

Designing RESTful API

Design:

- Can't fix an API after publishing
- Too easy to add ad-hoc endpoints
- Helps understand the requirements
- Well designed API can mature

VERB	URI (Query String)
	Headers
	request body

Status code
Headers
Response body

- APIs moeten gebaseerd zijn op simpele en begrijpelijke URI's
- Verbs moeten doen waarvoor zij bedoeld zijn (GET haalt iets op, PUT doet een update)
    - Let hier bij op dat het afhangt van de URI, bijv: de DELETE kan bijvoorbeeld niet een hele lijst verwijderen (http://api.nl/customers), maar wel een onderdeel van een lijst. (http://api.nl/customers/002)
- Designing an API betekend het volledige design, dus niet alleen de URIs.

Benodigde applicaties

- Postman (www.getpostman.com)

course over microservices
https://www.youtube.com/user/koushks

Netflix artikel:
https://medium.com/refraction-tech-everything/how-netflix-works-the-hugely-simplified-complex-stuff-that-happens-every-time-you-hit-play-3a40c9be254b

Distributie systeem:
https://www.techopedia.com/definition/18909/distributed-system
https://www.youtube.com/watch?v=ajjOEltiZm4

Voor het aantonen van een stateless communicatie heb ik een 3-tal API's gemaakt. Een eureka server, een show service en een get service.

De get service haalt data op bij een ESP8266 module. Hieraan zit een MLX90614 sensor aangesloten. Deze is zo gepgrammeerd dat je een POST call kan doen /getTemperatures en je krijgt een object terug met de omgevings- en objecttemperatuur terug.

De show service toont een invoerveld waarin de ID van de sensor ingegeven kan worden. Wanneer er op de knop geklikt wordt doet deze een API post call naar de get-temperature service die op zijn buurt de data ophaalt bij de sensor.
Deze geeft het weer terug met ID aan de show service die het vervolgens de temperatuur op het scherm weergeeft.

De code is te vinden hier