Labo 1: Spring boot

Dependencies toevoegen:

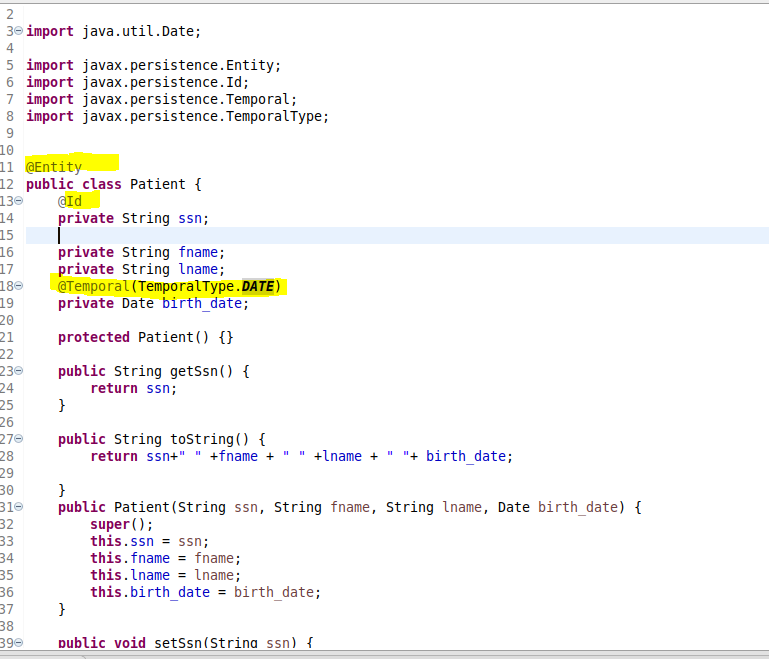




Right click project -> maven -> update project -> ok

# Entity

@GeneratedValue(strategy=GenerationType.AUTO) indien je een auto id veld wilt



Getters/setters/constructors genereren: RightClick -> source -> kies ‘Generate ...’

# Database

Standaard voorzien CRUD methodes:

<S extends T> S save(S entity);

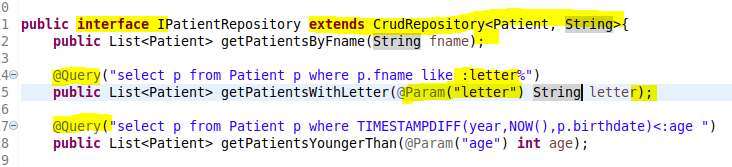
Optional<T> findById(ID primaryKey);

Iterable<T> findAll();

long count();

void delete(T entity);

boolean existsById(ID primaryKey);

zelf nog aanvullen (via interface) (moet via die functienotatie): 

Andere mogelijkheden:

Long countByX(

List<Klasse> deleteByX(

List<Klasse> findByX(  
List<Klasse> getByX(

@Nullable boven methode indien lege return value mag. Mag ook voor een optionele parameter staan.

Optional<Klasse> als return type dan geeft het Optional.empty() terug wanneer lege return value.

List<Person> find**Distinct**People**By**Lastname**Or**Firstname(String lastname, String firstname);

// Enabling ignoring case for an individual property

List<Person> findByLastnameIgnoreCase(String lastname);

// Enabling ignoring case for all suitable properties

List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

// Enabling static ORDER BY for a query

List<Person> findByLastnameOrderByFirstnameAsc(String lastname);

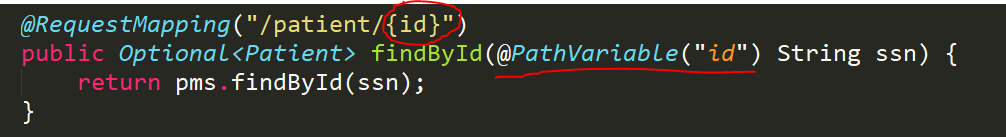
Indien er meerdere objecten van iets behoren tot een klasse  
 @ManyToOne(fetch = FetchType.LAZY, optional = false)  
 @JoinColumn(name = "ward\_id", nullable = false)  
 @OnDelete(action = OnDeleteAction.CASCADE)  
Ward ward;  
Aan de andere kant:  
 @OneToMany(cascade = CascadeType.ALL, mappedBy="ward", fetch = FetchType.EAGER)  
List<Bed> beds;

REST

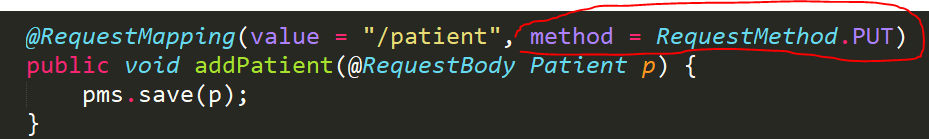
src/main/resources/application.properties



Class annotation: @restController gebruiken



PUT



# Autowired

@autowired voor een attribuut zetten voor het in te laden met dependancy injection

# MongoDB

*sudo service mongod start*

class annotation: @Document als het als document moet worden opgeslagen

public interface IFinanceRepository extends **MongoRepository<Invoice, String>**{}

*<ClassToSave, IdType>*

# DeferredResult

Voor asyncrone data

# Output

@output(STRINGVARNAME)  
MessageChannel methode();

application.properties:

spring.cloud.stream.bindings.**assign\_bed.destination**=**assign\_bed** spring.cloud.stream.bindings.assign\_bed.contentType=application/json spring.cloud.stream.bindings.**check\_in\_result.destination**=**check\_in\_completed** spring.cloud.stream.bindings.check\_in\_result.contentType=application/json

in main method class:  
@EnableBinding(ProducerChannels.class)

Gateway interface maken voor abstractie:

@MessagingGateway   
public interface MessageChannelGateway {   
 @Gateway(requestChannel = ProducerChannels.ASSIGN\_BED)   
 void assignBed(HospitalStay hospitalStay);   
  
 @Gateway(requestChannel = ProducerChannels.CHECK\_IN\_RESULT)   
 void emitHospitalCheckInCompleted(HospitalStay hospitalStay);  
 }

# Saga

@Component boven klasse

Bevat aantal void operaties die elk een stap uit de flow afhandelen. Meestal altijd bepaalde functie van het object dat wordt meegegeven uitvoeren en iets doorgeven aan de gateway

Opt einde via de RestController een bericht terug aan de client geven

# Application service

@component boven klasse

Bevat de saga. Vanuit de RestAPI wordt via deze klasse stukken van de saga opgeroepen.

Grootste nut van deze methodes is dat hier via de repository het juiste object wordt opgehaald op basis van een id en gecheckt wordt of het wel bestaat enzo. De uiteindelijke output gebeurd via de saga methodes zelf dus keert niet terug.

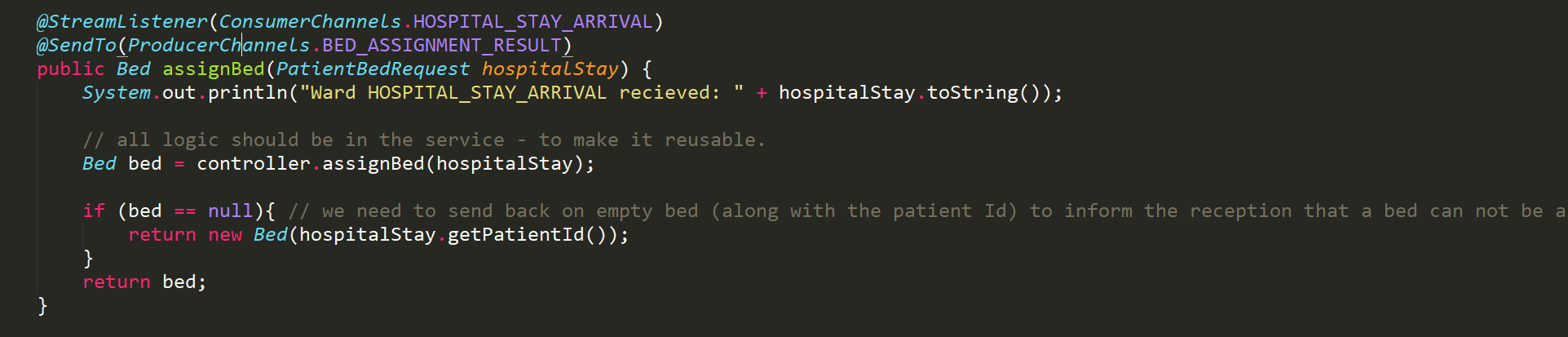
Soms wel de .save() van de repository oproepen

# Command handler

Neemt een binnenkomende message en roept nodige methods op

@streamlistener boven methode zodat het kan ontvangen

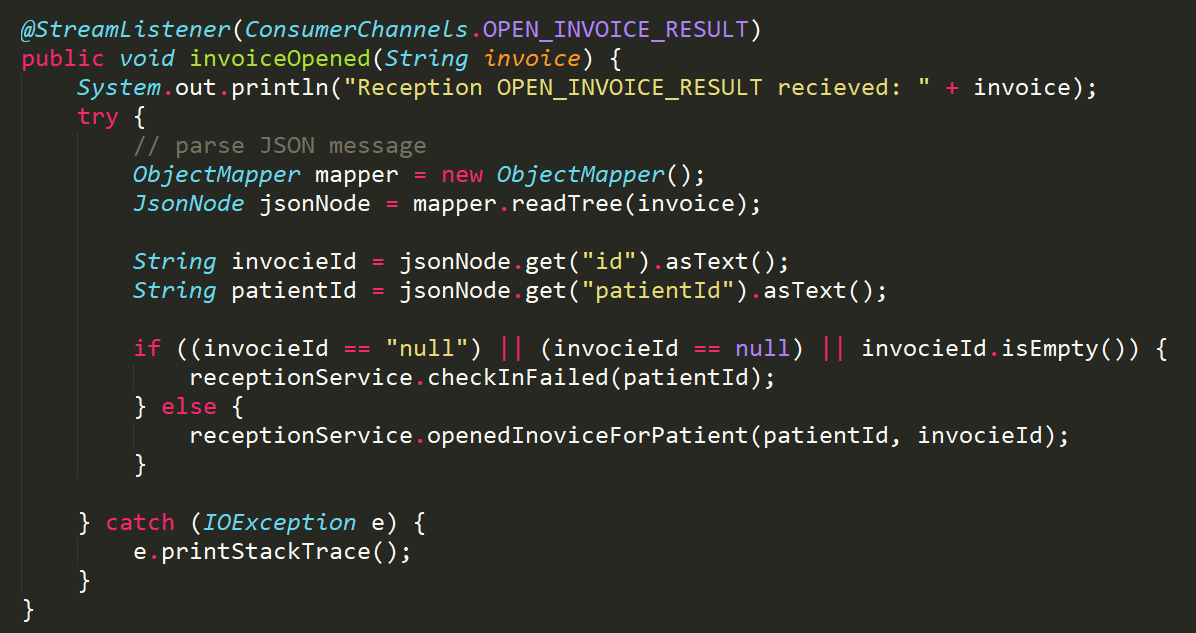
@sentTo voor het antwoord terug op de channel te zetten (zet bed id op kanaal met identificatie dat in die string staat van producerchannels. Dus ie zet wel degelijk da bed op de channel en niet enkel die string (in json))



Geen logica hier, die komt in de service. Direct doorsluizen naar application service

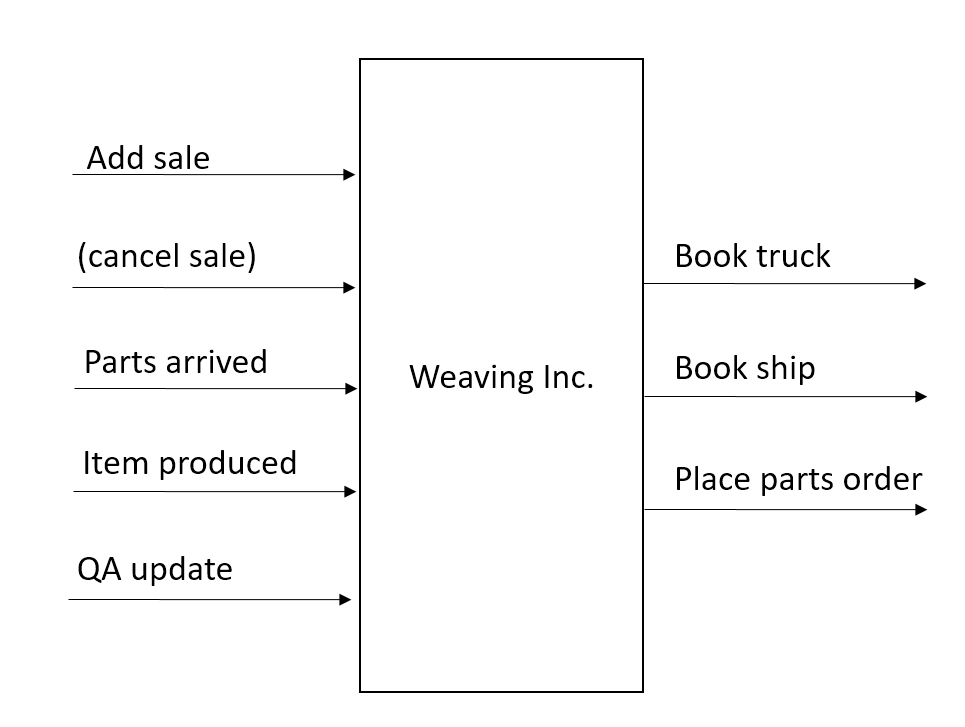
# EventConsumer

Bevat stream listeners die de json van de channel halen en dit doorsluizen naar de service

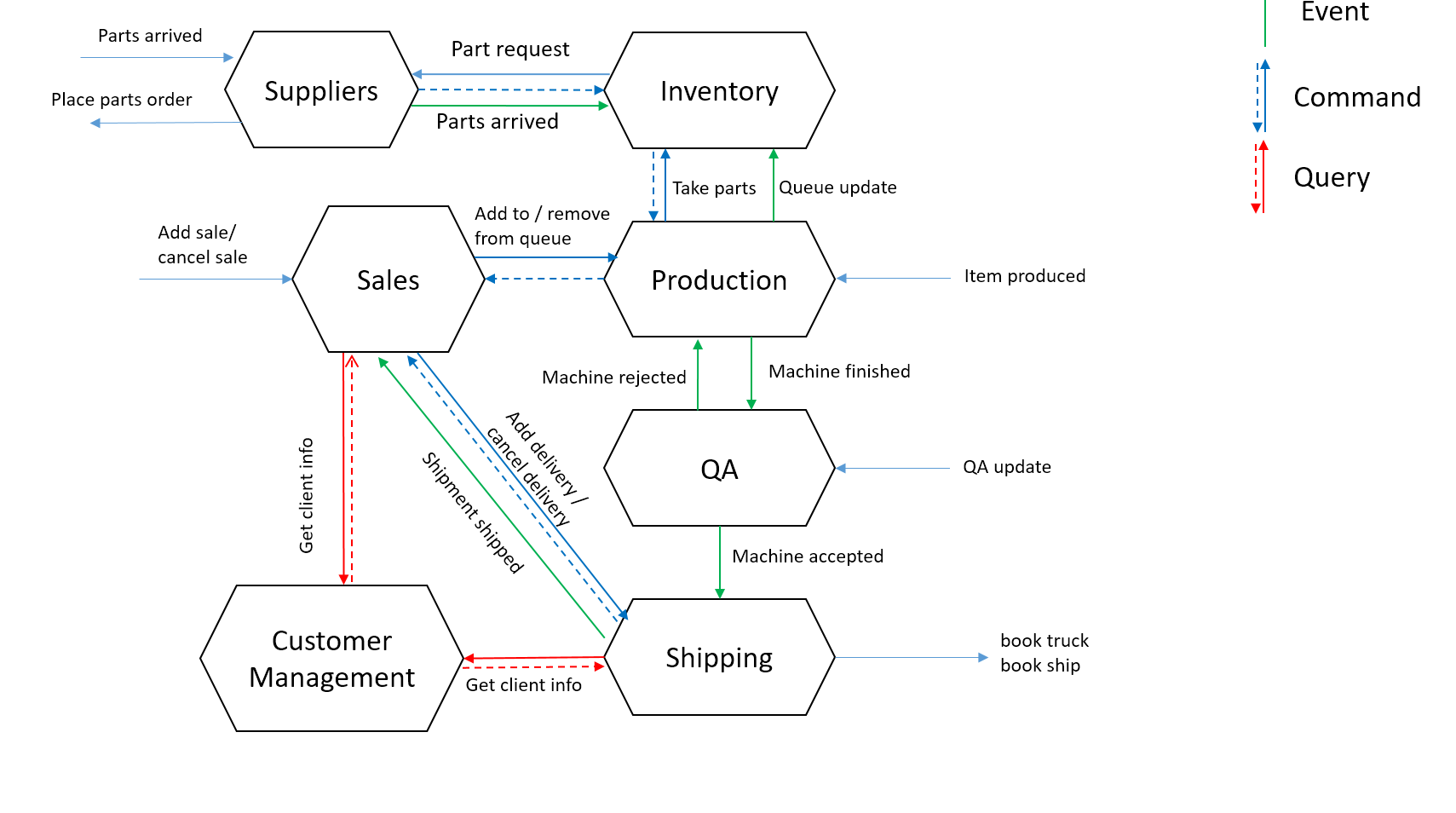


Labo 3

1. Systeem operaties bepalen. Systeem als blok zien

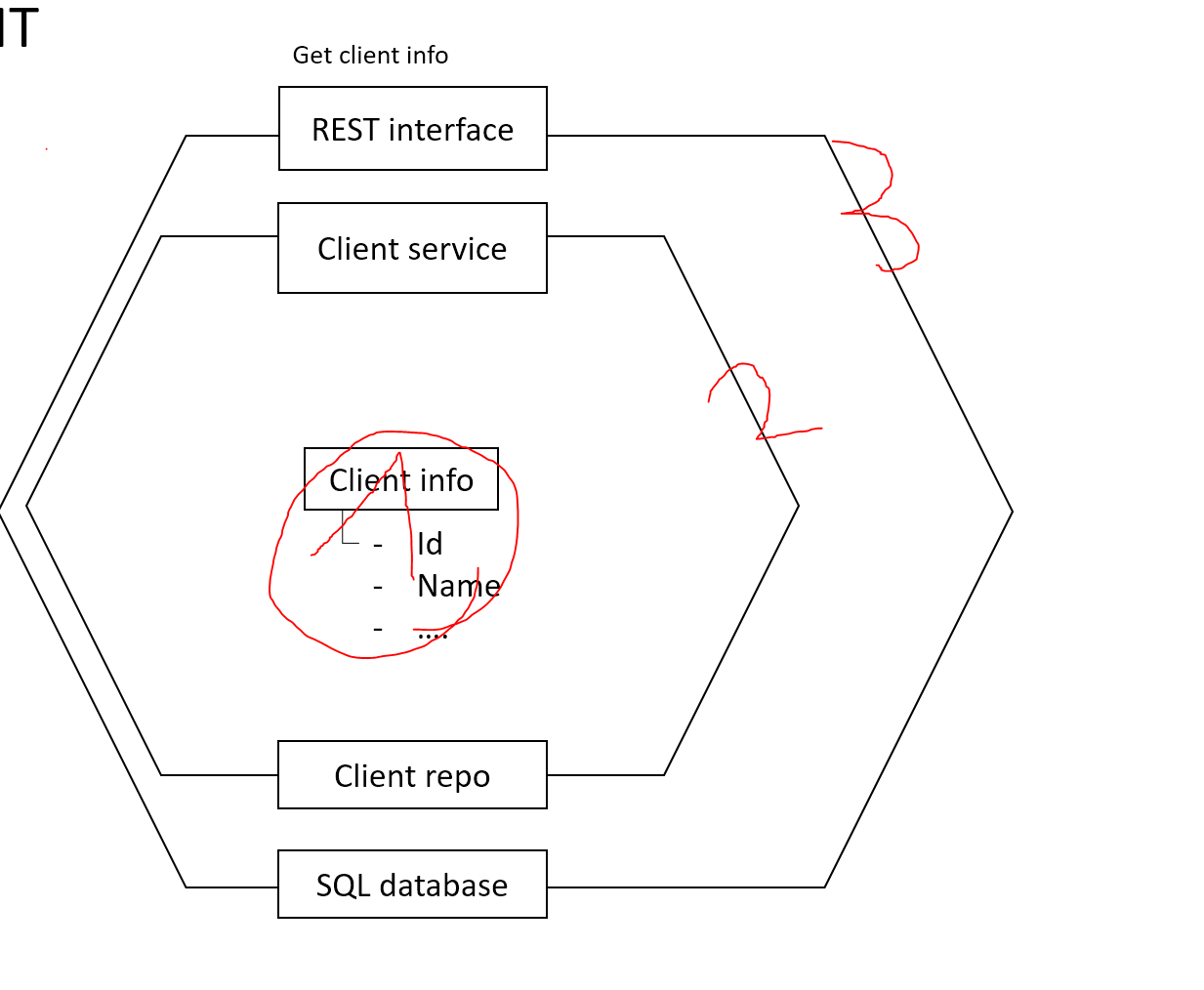


1. Services uit elkaar (in dit geval op basis fysieke departementen) halen en events/commands uitzoeken



1. Dan maak je per service een hexagonaal schema

3. contact met buitenwereld 2. hoe hij dingen intern doorsluist enzo 1. het domein model



Labo docker

4.1

Docker pull mysql dat binnenhalen

Eerst docker run as ge hem dan wilt herstarten docker start

Docker run –d --name patient\_db mysql –e MYSQL\_ROOT\_PASSWORD=’root’ –e MYSQL\_DATABASE=’patientDB’

-d voor detach (background)

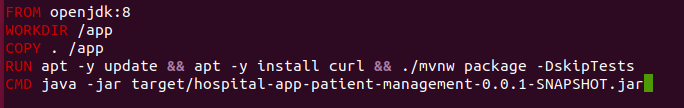
Hub.docker.com/\_/mysql voor environment variabeles beneden te vinden

Docker inspect patient\_db | grep “Env” –C 10 controle

Docker exec –i -t patient\_db /bin/bash en dan kunje linux commando’s runnen (exec /bin/bash in patient\_db op interactieve manier)

Touch Dockerfile

Nano Dockerfile

welke directory in uw container  
copy huidige directory in de app  


mag maar 1 CMD staan

en fout in opgave, tis targe/ en niet src/target

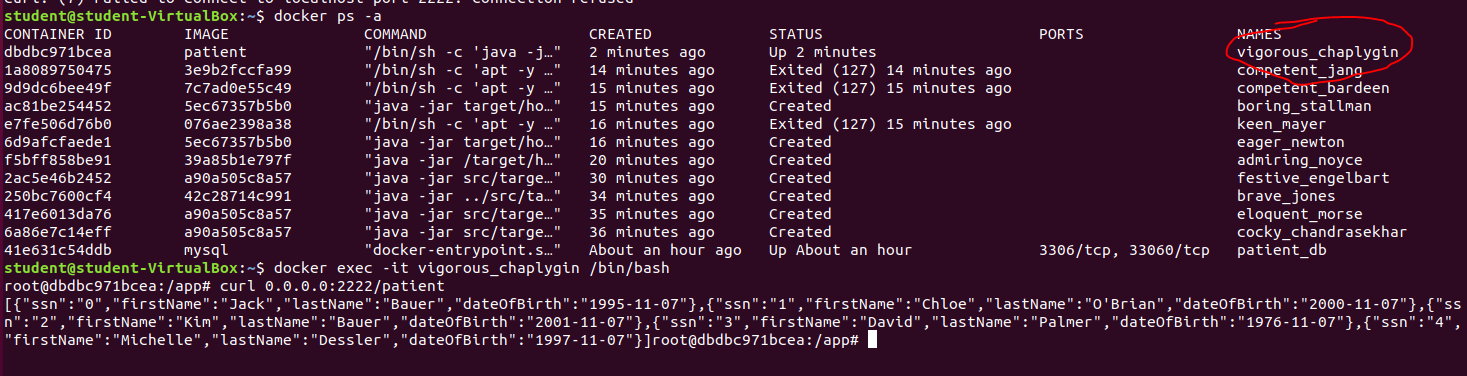
en geen cmd [“ “] doen ma zoals het er staat

mkdir docker NIET DOEN FOUT IN OPGAVE

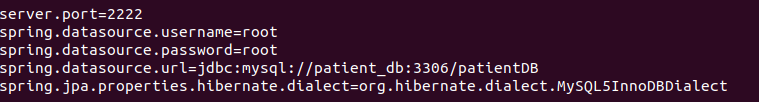
mv Dockerfile docker/Dockerfile NIET DOEN FOUT IN OPGAVE

docker build –t patient . dien image aanmaken

docker run patient dien image runnen



Application.properties voor mysql



Mysql://CONTAINERNAME::3360/DATABANKNAME

Docker builden en runnen dan terug