Spring boot lab dag 1

Dit lab heeft een aantal opdrachten die qua beschrijving hoog over zijn. Dat betekent dat niet elke stap helemaal uitgekauwd is maar dat je zelf gaat uitzoeken hoe het allemaal werkt. Natuurlijk staan hier wel een paar hints.

# Simpele Spring Boot applicatie schrijven

Schrijf een Spring Boot applicatie die op endpoint /echo de tekst “Hello, world” terugstuurt.

## Hints

* Maven project creeren
* Spring-boot-starter-parent als parent, en spring-boot-starter-web als dependency opnemen in je pom (*zie presentatie*).
* Root package maken en main class schrijven om Spring Boot te bootstrappen *(zie presentatie*)
* EchoController (@RestController) schrijven in dezelfde package (of subpackage) die op endpoint /echo de tekst “Hello, world!” terugstuurt (niet anders dan een reguliere Spring applicatie)
* Run de applicatie vanuit je IDE (main class runnen)
* Run de applicatie vanuit de command line: java -jar target\myapplication.jar (hangt natuurlijk af van je artifact naam). Gebruik hiervoor de spring-boot maven plugin om de applicatie goed te packagen, zie <http://docs.spring.io/spring-boot/docs/current/reference/html/build-tool-plugins-maven-plugin.html>

# Property resolving

Pas de echo service zo aan dat de “Hello” dynamisch wordt.

## Hints

* Constructor maken in de EchoController met @Value(“${greeting}”) String parameter, en de value gebruiken in het opbouwen van de string. Dus “Hello, world” vervangen door myGreeting + “, world!”
* application.properties plaatsen in src/main/resources met “greeting” key, kies een value, bijvoorbeeld “greetings”.
* Applicatie herstart, /echo endpoint testen
* Gebruik nu in plaats van een @Value een @ConfigurationProperties class die hetzelfde doet.

# REST service

We maken een recepten service. Een recept heeft de volgende structuur:

* id (Long, nodig voor straks)
* titel (string)
* ingredienten (List<String>)
* omschrijving (het eigenlijke recept) (string)

Schrijf een controller /recepten waarmee je met een GET zonder parameters alle recepten in JSON formaat op kan halen. In eerste instantie gewoon 1 recept in-memory schrijven.

## Hints

* Data class schrijven die een model van een recept is
* ReceptenController schrijven die op endpoint /recepten met een GET het recept terugstuurt
* ReceptenService schrijven die middels getAllRecepten een List<Recept> teruggeeft (met dus 1 recept) en deze wiren in je controller. Of je doet alles in je REST controller, dat maakt het echter later wellicht weer moeilijker testbaar.

# JPA

We willen de recepten uit een jpa datasource halen. Tevens willen we een recept kunnen invoegen middels bijvoorbeeld Postman (<https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop>)

## Hints

* Neem spring-boot-starter-jpa en com.h2database als maven dependency op.
* Recept class uitbreiden met jpa annotaties: @Entity voor de class en @Id, @GeneratedValue op het id veld. Ingredientenlijst, indien List<String> annoteren met @ElementCollection.
* @Repository spring bean maken die met een @PersistenceContext entitymanager de recepten uit de database haalt.
* We willen voor development graag alvast 1 recept in de database plaatsen. Maak een data.sql file in resources, deze wordt automatisch door SpringBoot opgepakt. Hier kan je alvast een recept plaatsen. Voorbeeld sql:

**insert into** recept(id, titel, omschrijving) **values** (1, 'vlaflip', 'mengen en klaar')

**insert into** recept\_ingredienten(recept\_id, ingredienten) **values**(1, 'gele vla')

**insert into** recept\_ingredienten(recept\_id, ingredienten) **values**(1, 'bruine vla')

Doe nu een GET op /recepten en je zou het eerste recept moeten zien!

*Schrijf nu ook een POST rest method waarmee je een nieuw recept kan invoegen in de database. De POST kan je doen middels bv Chrome Postman.*

# Logging

* logback.xml in resources maken, voor voorbeeld zie https://logback.qos.ch/manual/configuration.html

# Caching

Zorg ervoor dat het ophalen van de lijst van recepten cacheable is.

* spring-boot-starter-cache toevoegen als maven dependency
* @EnableCaching op je configuration / application class
* @Cacheable(“recepten”) op getAllRecepten op je service
* @CacheEvict op addRecept, nu moet immers de cache ververst worden

# Server side rendering

We gaan een simpele webpage uitserveren middels freemarker. Dit om aan te tonen dat traditionele server-side rendering ook gemakkelijk te bootstrappen is in een executable jar.

## Hints

* spring-boot-starter-freemarker toevoegen als maven dependency.
* In resources/templates een freemarker template welcome.ftl maken met daarin een property, bv <html><body>${greeting}, world!</body></html>
* Nieuwe Controller class schrijven met een @Requestmapping(“/freemarker”) method die een ModelAndView als returntype heeft. De naam van het ModelAndView is “welcome” (matchend aan de template dus) en deze bevat een “greeting” attribuut met bijvoorbeeld een value “Hello” (of je hergebruikt de property in je application.properties)
* Applicatie herstart, localhost:8080/freemarker zou nu een pagina op moeten leveren.

# Spring WebMvcTest

[http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-testing-spring-boot-applications-testing-autoconfigured-mvc-teSchrijf een WebMvcTest die alle recepten ophaalt maar enkel de rest laag doortest. Een voorbeeld staat hier:](http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-testing-spring-boot-applications-testing-autoconfigured-mvc-tests)

[sts](http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-testing-spring-boot-applications-testing-autoconfigured-mvc-tests)

# Spring Boot IT test

Schrijf een SpringBootTest die alle recepten ophaalt. Een voorbeeld staat hier:

<http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-testing-spring-boot-applications-mocking-beans>

Je zou nu ook nog de repository kunnen mocken middels een @MockBean, omdat je wellicht geen database hebt tijdens development time.

# Actuator

Voeg de spring-boot-starter-actuator toe als maven dependency, en kijk of je wat metrics endpoints kan benaderen, bv /health, /metrics.

# Voor wie zich hierna nog verveelt….

* Schrijf eens een custom metric, bv hoe vaak de rest service wordt aangeroepen
* Deploy je app naar de cloud (bv ElasticBeanStalk)
* Dockerize je applicatie (<https://spring.io/guides/gs/spring-boot-docker/>)