



# Spring Advanced

Deze cursus is eigendom van de VDAB©

## Inhoudsopgave

<b>1</b>	<b>INLEIDING.....</b>	<b>11</b>
1.1	Doelstelling.....	11
1.2	Vereiste voorkennis.....	11
1.3	Nodige software .....	11
<b>2</b>	<b>XML.....</b>	<b>12</b>
2.1	Project .....	12
2.2	Java code .....	12
2.3	XML.....	12
2.4	Gebruik van de bean .....	13
<b>3</b>	<b>@BEAN.....</b>	<b>14</b>
3.1	Project .....	14
3.2	Bean definitie .....	14
3.3	Bean gebruik.....	15
<b>4</b>	<b>MEERTALIG.....</b>	<b>16</b>
4.1	Inleiding.....	16
4.2	Project .....	16
4.3	IntelliJ .....	16
4.4	Taal van de gebruiker .....	16
4.5	messages.properties .....	16
4.6	Parameters .....	18
4.7	System locale.....	18
4.8	Taal en het land van de gebruiker .....	18
4.8.1	AcceptHeaderLocaleResolver .....	19
4.8.2	FixedLocaleResolver.....	19
4.8.3	SessionLocaleResolver .....	19
4.8.4	LocaleChangeInterceptor.....	19
4.8.5	CookieLocaleResolver .....	20

<b>5</b>	<b>SPRING SECURITY.....</b>	<b>21</b>
5.1	Security woordenschat.....	21
5.2	Database.....	21
5.3	Project .....	21
5.4	application.properties.....	22
5.5	Controllers.....	22
5.5.1	IndexController .....	22
5.5.2	OffertesController.....	22
5.5.3	WerknemersController .....	22
5.6	CSS.....	22
5.7	Thymeleaf.....	23
5.7.1	index.html.....	23
5.7.2	offertes.html.....	23
5.7.3	werknemers.html .....	23
5.8	Default security .....	23
5.9	Security zelf configureren .....	24
5.10	Fobidden pagina.....	25
5.11	Inlogpagina .....	25
5.12	Login fouten .....	26
5.13	Inloggen.....	26
5.14	Uitloggen .....	26
5.15	Thymeleaf.....	27
5.15.1	pom.xml.....	27
5.15.2	authorize.....	27
5.15.3	Naam van de gebruiker.....	28
5.16	Database.....	28
5.16.1	Password encoding .....	28
5.16.2	Default tabel structuren .....	28
5.16.3	SecurityConfig.....	29
5.16.4	Afwijkende tabel structuren .....	29
5.17	Authorization met annotations.....	30
5.17.1	@PreAuthorize .....	30
<b>6</b>	<b>OAUTH .....</b>	<b>31</b>
6.1	Project .....	31
6.2	Registreren bij GitHub.....	31

6.3	SecurityConfig .....	32
6.4	Controllers .....	32
6.4.1	IndexController .....	32
6.4.2	BeveiligdeController .....	32
6.5	Thymeleaf.....	33
6.5.1	index.html.....	33
6.5.2	beveilgd.html .....	33
<b>7</b>	<b>SPRING DATA.....</b>	<b>34</b>
7.1	Repository interfaces .....	34
7.2	Database.....	34
7.3	Project .....	34
7.4	application.properties .....	34
7.5	Entities.....	35
7.5.1	Filiaal.....	35
7.5.2	Werknemer .....	35
7.6	JpaRepository .....	35
7.7	Derived query methods.....	37
7.7.1	Filteren.....	37
7.7.2	Implementatie .....	38
7.7.3	Test .....	38
7.7.4	Filteren: tweede voorbeeld .....	38
7.7.5	Andere voorbeelden .....	38
7.7.6	Sorteren .....	39
7.7.7	Tellen .....	39
7.8	@Query .....	39
7.9	Named query's .....	40
7.10	Gerelateerde entities .....	40
7.10.1	Filteren op attributen van gerelateerde entities .....	40
7.10.2	@EntityGraph .....	41
7.11	Pagineren .....	42
<b>8</b>	<b>MAIL.....</b>	<b>43</b>
8.1	Database.....	43
8.2	Project .....	43
8.3	application.properties .....	43
8.4	Mail configuratie .....	44

8.5	Entity .....	44
8.6	Repository .....	45
8.7	Bean die de mail stuurt .....	45
8.8	Service .....	45
8.9	Controller .....	46
8.10	CSS .....	47
8.11	ValidationMessages .....	47
8.12	Thymeleaf pagina's .....	47
8.12.1	registratieform.html .....	47
8.12.2	geregistreerd.html .....	47
8.12.3	nietgeregistreerd.html .....	48
8.13	Opmaak .....	48
8.14	Hyperlink .....	49
8.14.1	Controller .....	49
8.14.2	lidinfo.html .....	49
8.14.3	DefaultLidMailing .....	49
<b>9</b>	<b>ASYNCHROON .....</b>	<b>50</b>
9.1	@Async .....	50
9.2	@EnableAsync .....	50
9.3	Voorbeeld .....	50
<b>10</b>	<b>SCHEDULING .....</b>	<b>51</b>
10.1	@Scheduled .....	51
10.2	@EnableScheduling .....	51
10.3	Voorbeeld .....	51
<b>11</b>	<b>REST .....</b>	<b>53</b>
11.1	Entities identificeren met URI's .....	53
11.2	HTTP methods .....	53
11.2.1	GET .....	53
11.2.2	POST .....	53
11.2.3	PUT .....	53
11.2.4	DELETE .....	53
11.3	Formaat van de data .....	54

11.4	Response status code.....	54
11.5	HATEOAS .....	54
<b>12</b>	<b>REST SERVICE.....</b>	<b>55</b>
12.1	Database.....	55
12.2	Project .....	55
12.3	application.properties .....	55
12.4	Entity .....	55
12.5	Repository .....	55
12.6	Exception .....	56
12.7	Service .....	56
12.7.1	FiliaalService .....	56
12.7.2	DefaultFiliaalService .....	56
12.8	Controller .....	57
12.9	GET .....	57
12.10	Test.....	57
12.11	JAXB .....	57
12.12	404 (Not Found) .....	58
12.13	DELETE .....	58
12.14	POST .....	58
12.15	Validatie.....	59
12.16	PUT .....	59
12.17	HATEOAS .....	60
12.17.1	Configuratie per controller .....	60
12.17.2	EntityLinks.....	60
12.17.3	Location .....	60
12.17.4	Link elementen in de response body .....	61
12.17.5	Response met een verzameling entities .....	61
<b>13</b>	<b>DOCUMENTATIE .....</b>	<b>63</b>
13.1	pom.xml.....	63
13.2	Resultaat.....	63
13.3	Omschrijving.....	63

13.4	Titel.....	63
<b>14</b>	<b>REST SERVICE INTEGRATION TEST .....</b>	<b>64</b>
<b>15</b>	<b>REST CLIENT.....</b>	<b>66</b>
15.1	WebClient.....	66
15.1.1	GET.....	66
15.1.2	POST.....	66
15.1.3	PUT .....	67
15.1.4	DELETE .....	67
15.2	Voorbeeld.....	67
15.2.1	Project.....	67
15.2.2	Structuur van de JSON data .....	68
15.3	RestClients layer .....	68
15.4	RestClients layer gebruiken .....	70
15.4.1	Controller:.....	70
15.4.2	user.html.....	70
<b>16</b>	<b>REST CLIENT MET JAVASCRIPT.....</b>	<b>71</b>
16.1	Frontend – backend .....	71
16.1.1	Één applicatie .....	71
16.1.2	Twee applicaties .....	71
16.2	CORS .....	71
16.3	Frontend project .....	72
16.3.1	CSS .....	72
16.3.2	HTML.....	73
16.3.3	JavaScript .....	73
16.3.4	Detail.....	74
16.3.5	Toevoegen .....	75
<b>17</b>	<b>VALIDATION ANNOTATIONS MAKEN.....</b>	<b>76</b>
17.1	Project .....	76
17.2	pom.xml.....	76
17.3	Standaard annotations.....	76
17.4	Test.....	76
17.5	Samenstelling van andere validation annotations .....	77
17.6	Eigen bean validation in detail .....	79
17.7	Object properties ten opzichte van elkaar valideren .....	80

<b>18</b>	<b>PROFILES .....</b>	<b>82</b>
18.1	Databases .....	82
18.2	Project .....	82
18.3	application.properties .....	82
18.4	Test .....	82
18.4.1	spring.properties .....	82
18.4.2	DataSourceTest .....	83
18.5	Dev profile .....	83
18.6	Profile activeren .....	83
18.6.1	Command line argument .....	83
18.6.2	Environment variable .....	83
18.7	@Profile .....	84
<b>19</b>	<b>DOCKER .....</b>	<b>85</b>
19.1	Images, daemon, cli, registry .....	85
19.1.1	Image .....	85
19.1.2	Daemon .....	85
19.1.3	CLI .....	85
19.1.4	Registry .....	85
19.1.5	Volumes .....	86
19.1.6	Networks .....	86
19.1.7	Overzicht .....	86
19.2	Docker desktop .....	86
19.3	Opdrachten .....	86
19.3.1	Opdrachten voor images .....	86
19.3.2	Opdrachten voor containers .....	87
19.3.3	Levensfasen .....	87
19.3.4	docker run .....	88
19.4	Eigen applicatie .....	88
19.4.1	Project .....	88
19.4.2	Controller .....	88
19.4.3	Image .....	88
19.4.4	Uitvoeren .....	89
19.5	Container in achtergrond uitvoeren .....	89
19.6	Volumes .....	89
19.6.1	Kopiëren .....	90
19.7	TCP/IP poorten .....	90
19.7.1	Controller .....	91
19.7.2	Index.html .....	91
19.8	MySQL .....	91



19.8.1	Container .....	91
19.8.2	Workbench .....	92
19.9	Applicatie spreekt MySQL in container aan .....	92
19.9.1	pom.xml .....	92
19.9.2	application.properties .....	92
19.9.3	Land .....	92
19.9.4	LandRepository .....	92
19.9.5	LandService .....	93
19.9.6	DefaultLandService .....	93
19.9.7	IndexController .....	93
19.9.8	index.html .....	93
19.10	Networks .....	93
19.10.1	Netwerk maken .....	93
19.10.2	Container in netwerk uitvoeren .....	93
19.10.3	Project .....	94
<b>20</b>	<b>MESSAGING .....</b>	<b>95</b>
<b>21</b>	<b>RABBITMQ .....</b>	<b>96</b>
21.1	Installeren .....	96
21.2	RabbitMQ management .....	96
21.3	Exchanges en queues .....	96
21.3.1	Direct exchange .....	96
21.3.2	Fanout Exchange .....	97
<b>22</b>	<b>BERICHTEN STUREN .....</b>	<b>98</b>
22.1	Voorbeeld .....	98
22.2	RabbitMQ .....	98
22.3	Database .....	98
22.4	Project .....	98
22.5	application.properties .....	98
22.6	Artikel entity .....	99
22.7	ArtikelRepository .....	99
22.8	ArtikelGemaakt .....	99
22.9	ArtikelService .....	99
22.10	CatalogusApplication .....	100
22.11	DefaultArtikelService .....	100

22.12	ArtikelController .....	100
22.13	Uitvoeren.....	101
<b>23</b>	<b>BERICHTEN ONTVANGEN .....</b>	<b>102</b>
23.1	Voorbeeld.....	102
23.2	Database.....	102
23.3	Project .....	102
23.4	application.properties .....	102
23.5	Artikel entity.....	102
23.6	ArtikelRepository.....	103
23.7	ArtikelService.....	103
23.8	DefaultArtikelService.....	103
23.9	ArtikelGemaakt .....	103
23.10	VorraadApplication .....	103
23.11	ArtikelListener .....	104
23.12	Uitvoeren.....	104
23.13	MicroServices .....	104
<b>24</b>	<b>OUTBOX .....</b>	<b>105</b>
24.1	Stap 1.....	105
24.1.1	Database .....	105
24.1.2	ArtikelGemaakt .....	105
24.1.3	ArtikelGemaaktRepository.....	105
24.1.4	DefaultArtikelService .....	106
24.2	Stap 2.....	106
24.2.1	@EnableScheduling .....	106
24.2.2	MessageSender.....	106
24.3	Docker .....	107
<b>25</b>	<b>TESTCONTAINERS .....</b>	<b>108</b>
25.1	Project .....	108
25.2	pom.xml.....	108
25.3	application.properties .....	108

25.4	Persoon .....	109
25.5	PersoonRepository .....	109
25.6	insertPersonen.sql.....	109
25.7	spring.properties.....	109
25.8	PersoonRepositoryTest .....	109
<b>26</b>	<b>COLOFON.....</b>	<b>111</b>

# 1 INLEIDING

## 1.1 Doelstelling

Je leert werken met geavanceerde onderdelen van het Spring framework.

## 1.2 Vereiste voorkennis

- |   |   |
|---|---|
| <ul style="list-style-type: none"><li>• Java</li><li>• JDBC</li><li>• Maven</li></ul> | <ul style="list-style-type: none"><li>• Unit testing</li><li>• Spring fundamentals</li><li>• JPA met Hibernate</li><li>• JavaScript (voor het hoofdstuk REST client met JavaScript)</li></ul> |
|---|---|

## 1.3 Nodige software

- JDK (minstens versie 11).
- MySQL (minstens versie 8.0.14)
- IntelliJ Ultimate (minstens versie 2020.1)

## 2 XML

Je maakt tot nu een Spring bean door voor een class `@Component`, `@Repository`, `@Service` of `@Controller` te typen. Voor Java annotations bevatte definieerde je beans in XML bestanden.

Je leert dit hier kennen. Je begrijpt dit dan als je een oudere applicatie onderhoudt.

### 2.1 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ `be.vdab` bij Group.
4. Typ `xml` bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web
Template Engines	Thymeleaf

7. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 2.2 Java code

Maak een package `be.vdab.xml.services`. Maak daarin een interface:

```
package be.vdab.xml.services;
public interface KwadraatService {
    int kwadraat(int getal);
}
```

Maak een class die de interface implementeert:

```
package be.vdab.xml.services;
class DefaultKwadraatService implements KwadraatService {
    @Override
    public int kwadraat(int getal) {
        return getal * getal;
    }
}
```

### 2.3 XML

Je maakt een XML bestand. Je definieert daarin een Spring bean van de class hierboven.

1. Klik in het project overzicht met de rechtermuisknop op de folder resources.
2. Kies New, File.
3. Typ `beans.xml` (de bestandsnaam is vrij te kiezen). Druk Enter.

Wijzig dit bestand:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="kwadraatService"
        class="be.vdab.xml.services.DefaultKwadraatService"/>
</beans>
```

①  
②

- (1) Je definieert een bean met een `<bean>` element. Je geeft elke bean een unieke id.
- (2) Je typt bij `class` de class (met zijn package) waarvan Spring een bean zal maken.

Je geeft aan dat Spring, bij de start van de website, dit bestand moet verwerken.

Typ volgende regel voor de class XmlApplication:

```
@ImportResource("beans.xml")
```

## 2.4 Gebruik van de bean

Injecteer de bean in een controller. Zo zie je dat de bean werkt.

```
package be.vdab.xml.controllers;
// enkele imports
@Controller
@RequestMapping("/")
class IndexController {
    private final KwadraatService kwadraatService;
    IndexController(KwadraatService kwadraatService) {
        this.kwadraatService = kwadraatService;
    }
    @GetMapping
    public ModelAndView index() {
        return new ModelAndView("index", "kwadraat", kwadraatService.kwadraat(4));
    }
}
```

IntelliJ geeft je een foutmelding bij de parameter van de constructor.

Je vermijdt die door het beans.xml kenbaar te maken aan IntelliJ:

1. Open beans.xml.
2. Kies rechts boven Configure application context.
3. Kies Create new application context.
4. Kies OK.

Maak een Thymeleaf pagina:

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Kwadraat</title>
    </head>
    <body>
        <h1> Het kwadraat van 4 is:<span th:text="${kwadraat}"></span></h1>
    </body>
</html>
```

Je kan de applicatie uitvoeren.



Commit de sources. Publiceer op je remote repository.

### 3 @BEAN

Je definieert tot nu Spring beans met de annotations `@Component`, `@Repository`, `@Service` en `@Controller` te typen. Je definieert hier beans met `@Bean`.

#### 3.1 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ `be.vdab` bij Group.
4. Typ `atbean` bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web
Template Engines	Thymeleaf

7. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

#### 3.2 Bean definitie

Maak een package `be.vdab.xml.services`. Maak daarin een interface:

```
package be.vdab.atbean.services;
public interface KwadraatService {
    int kwadraat(int getal);
}
```

Maak een class die de interface implementeert:

```
package be.vdab.atbean.services;
class DefaultKwadraatService implements KwadraatService {
    @Override
    public int kwadraat(int getal) {
        return getal * getal;
    }
}
```

Maak in dezelfde package een class `ServiceBeansConfiguration`:

```
package be.vdab.atbean.services;
// enkele imports
@Configuration
class ServiceBeansConfiguration {
    @Bean
    DefaultKwadraatService kwadraat() {
        return new DefaultKwadraatService();
    }
}
```

①  
②  
③  
④  
⑤

- (1) Je typt `@Configuration` voor een class waarin je beans definieert met `@Bean`.
- (2) De naam van de class is vrij te kiezen.
- (3) Je typt `@Bean` voor een method.  
Spring maakt dan een bean van het object dat deze method teruggeeft.
- (4) De naam van de method is vrij te kiezen.
- (5) Je geeft een `DefaultKwadraatService` object terug. Spring maakt hiervan een bean.

### 3.3 Bean gebruik

Maak een package `be.vdab.atbean.controllers`. Maak daarin een controller:

```
package be.vdab.atbean.controllers;
@RequestMapping("/")
class IndexController {
    private final KwadraatService kwadraatService;
    IndexController(KwadraatService kwadraatService) {
        this.kwadraatService = kwadraatService;
    }
    @GetMapping
    public ModelAndView index() {
        return new ModelAndView("index", "kwadraat", kwadraatService.kwadraat(4));
    }
}
```

(1) Je typt geen `@Controller` voor de class. Je zal een Spring bean maken van de class met `@Bean`.

Maak in dezelfde package een class `ControllerBeansConfiguration`:

```
package be.vdab.atbean.controllers;
// enkele imports
@Configuration
class ControllerBeansConfiguration {
    @Bean
    IndexController index(KwadraatService kwadraatService) {
        return new IndexController(kwadraatService);
    }
}
```

(1) Je maakt in deze method een `IndexController` object. De `IndexController` constructor verwacht als parameter een object dat de interface `KwadraatService` implementeert. Je geeft de `@Bean` method een parameter van het type `KwadraatService`. Spring zoekt een bean die deze interface implementeert en vult de parameter met deze bean.

Maak een Thymeleaf pagina:

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Kwadraat</title>
    </head>
    <body>
        <h1> Het kwadraat van 4 is:<span th:text="${kwadraat}"></span></h1>
    </body>
</html>
```

Je kan de applicatie uitvoeren.

Voordelen van `@Bean`:

- ➕ Je kan beans maken van classes waarvan je de sources niet hebt. Je kan dit niet met `@Component`, `@Repository`, `@Service` en `@Controller`.
- ➕ Je kan in een `@Bean` method een complex algoritme schrijven (met `if`, `while`, ...) waarin je beslist hoe je de bean maakt.



Je kan ook in de class, waarvoor `@SpringBootApplication` (hier de class `AtbeanApplication`) staat, `@Beans` methods maken. Het nadeel is dat in een grote applicatie deze class *veel* zo'n methods bevat.



Commit de sources. Publiceer op je remote repository.



## 4 MEERTALIG

### 4.1 Inleiding

Je typt de teksten van een website tot nu in *één* taal: Nederlands.

Je maakt hier een *meertalige* website: Nederlands en Engels.

### 4.2 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ `be.vdab` bij Group.
4. Typ meertalig bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web
Template Engines	Thymeleaf

7. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 4.3 IntelliJ

Je typt de teksten van je website straks in bestanden met de extensie `properties`.

IntelliJ bewaart deze bestanden standaard met `windows-1252` encoding.

De bestanden kunnen dan geen tekens met accenten (`é, ï, ñ, ...`) bevatten.

Je wijzigt de instellingen van IntelliJ. IntelliJ bewaart de bestanden dan in `UTF-8` encoding.

De bestanden kunnen dan wél tekens met accenten bevatten.

1. Kies in het menu File de opdracht Settings.
2. Open links Editor.
3. Kies links File Encodings.
4. Kies `UTF-8` bij Default encoding for properties files.
5. Kies OK.

### 4.4 Taal van de gebruiker

Spring leest de taal van de gebruiker standaard in de request header `Accept-Language`.

Die bevat een taalcode (bvb. `en`) of een combinatie van een taalcode en een landcode (bvb. `en-US`).

### 4.5 messages.properties

Je typt de teksten van je website in `messages.properties` en variaties van dit bestand.

Een resource bundle is een *verzameling* van zo'n bestanden.

Één bestand bevat teksten in één taal.

De bestandsnaam definieert de taal die bij de resource bundle hoort.

De bestandsnaam eindigt op `_` gevolgd door de taalcode. Voorbeeld: `messages_en.properties`

Men spreekt sommige talen in *meerdere* landen.

Je kan de tekst per land vertalen in een bestand specifiek voor dat land.

Je geeft dit land ook aan in de bestandsnaam. Voorbeeld: `messages_en_US.properties`

Als Spring een tekst nodig heeft, doorzoekt Spring de bestanden in deze volgorde:

1. `messages_en_US.properties` Het bestand met de gebruikerstaal- én landcode.
2. `messages_en.properties` Het bestand met enkel de taalcode.
3. `messages.properties` Het bestand zonder taalcode.



Je kan *meerdere* taal-land combinaties instellen in je browser.

De browser stuurt al deze combinaties in de request header Accept-Language.

Spring gebruikt bij het zoeken in de resource bundles al deze combinaties.

Als de header en én fr bevat, doorzoekt Spring de resource bundles in de volgorde messages\_en.properties, messages\_fr.properties, messages.properties.

Nederlands wordt de standaardtaal van je website. Je maakt ook een Engelstalige versie:

1. Klik met de rechtermuisknop op src/main/resources.
2. Kies New, Resource Bundle.
3. Typ messages bij Resource bundle name.
4. Kies + bij Locales to add.
5. Typ en.
6. Kies OK.
7. Kies OK.

IntelliJ toont de bestanden in het project overzicht op volgende manier:

```
Resource Bundle 'messages'
├── messages.properties
└── messages_en.properties
```

Typ volgende regel in messages.properties:

```
welkom=Welkom
```

❶

- (1) Je geeft elke tekst een unieke key (bvb. welkom). De tekst zelf komt na =.

Typ volgende regel in messages\_en.properties:

```
welkom=Welcome
```

❶

- (1) De keys van de teksten in deze taal moeten dezelfde zijn als in messages.properties.

Maak een package be.vdab.meertalig.controllers. Maak daarin IndexController:

```
package be.vdab.meertalig.controllers;
// enkele imports
@Controller
@RequestMapping("/")
class IndexController {
    @GetMapping
    public String index() {
        return "index";
    }
}
```

Maak index.html in de folder templates:

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head>
    <title th:text="#{welkom}"></title>
  </head>
  <body>
    <h1 th:text="#{welkom}"></h1>
  </body>
</html>
```


❶

❷

- (1) Je leest uit een resource bundle de tekst met de key welkom.  
Je doet dit met de syntax #{keyVanDeTekst}.
- (2) Je haalt dezelfde tekst nog eens op.

Je start de website. Je ziet de pagina in de standaardtaal van de website: Nederlands.

Je stelt in Firefox Engels in als je voorkeurstaal:

1. Kies rechts boven in Firefox .
2. Kies Options.
3. Kies Choose bij Choose your preferred language for displaying pages.
4. Voeg English toe met Select a language to add en Add.
5. Stel English als voorkeurstaal met Move Up.

Ververs de pagina. Je ziet de pagina in het Engels.

## 4.6 Parameters

Een tekst in een resource bundle kan parameters bevatten.

Je typt een parameter als {volgNrVanDeParameter}. De volgnummers beginnen vanaf 0.

Voeg een regel toe aan messages.properties:

```
dagVanHetJaar=Dag van het jaar: {0}. ❶
```

(1) De tekst bevat één parameter, aangegeven met {0}.

Voeg een regel toe aan messages\_en.properties:

```
dagVanHetJaar=Day of the year: {0}.
```

Wijzig in IndexController de method index:

```
@GetMapping
public ModelAndView index() {
    return new ModelAndView("index", "volgnummer", LocalDate.now().getDayOfYear());
}
```

Voeg in index.html een regel toe, na </h1>:

```
<p th:text="#{dagVanHetJaar(${volgnummer})}"></p> ❶
```

(1) Bij het oproepen van de tekst geef je waarden mee voor de parameters.

Je geeft als waarde het getal die de controller meegaf onder de naam volgnummer.

Druk Ctrl+F9. Ververs de pagina in je browser.

## 4.7 System locale

Voeg volgende regel toe aan application.properties:

```
spring.messages.fallback-to-system-locale=false ❶
```

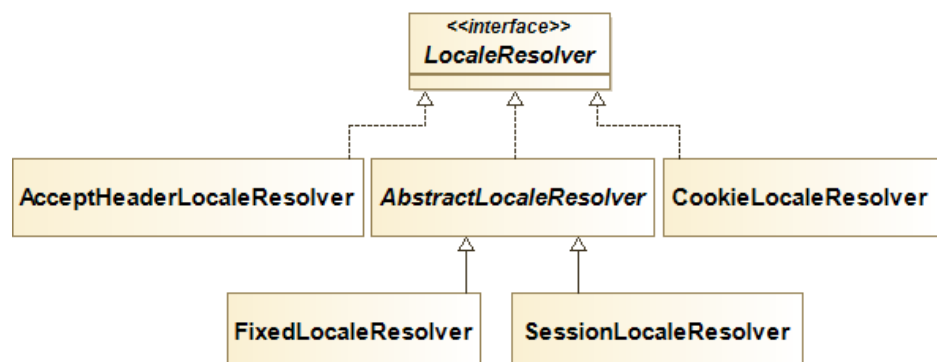
(1) Default staat deze property op true.

Als Spring een tekst niet vindt in de resource bundle met taalcode en landcode van de gebruiker en ook niet in de resource bundle met taalcode van de gebruiker, zoekt Spring de tekst in de resource bundle met de taalcode van het besturingssysteem. Dit is vervelend: je website gedraagt zich anders volgens de taal van het besturingssysteem waarop hij wordt uitgevoerd. Je plaatst de property daarom op false.

Als Spring nu een tekst niet vindt in de resource bundle met taalcode en landcode van de gebruiker en ook niet in de resource bundle met taalcode van de gebruiker, zoekt Spring de tekst in de resource bundle zonder taalcode (messages.properties).

## 4.8 Taal en het land van de gebruiker

Meerdere Spring classes bevatten een strategie om de taal en het land van de gebruiker te bepalen:



#### 4.8.1 AcceptHeaderLocaleResolver

Dit is de default strategie die je al gebruik.

Spring bepaalt de taal en het land van de gebruiker met de request header Accept-Language.

#### 4.8.2 FixedLocaleResolver

De taal en het land is voor elke gebruiker dezelfde.

Je kiest deze strategie door een bean van de class FixedLocaleResolver te maken.

Maak in de package be.vdab.meertalig.controllers een class WebConfig:

```
package be.vdab.meertalig.controllers;
// enkele imports
@Configuration
class WebConfig {
    @Bean
    LocaleResolver localeResolver() {
        return new FixedLocaleResolver(new Locale("en", "US"));
    }
}
```

❶

❷

❸

(1) Je typt @Configuration voor een class waarbinnen je Spring beans maakt met @Bean (bij ❷).

(2) Je schrijft @Bean voor een method.

Spring maakt dan een bean van de returnwaarde van die method.

(3) Locale stelt een combinatie van een land en een taal voor.

Je probeert de website. Ongeacht je browserinstelling zie je de welkompagina in het Engels.

#### 4.8.3 SessionLocaleResolver

Jij vraagt de taal (en eventueel het land) van de gebruiker. Je geeft de keuze door aan de SessionLocaleResolver bean. Die onthoudt de keuze in een HttpSession variabele.

De gebruiker behoudt zo zijn keuze zolang hij op de website blijft.

Zolang de gebruiker geen taal of land kiest, gebruikt Spring de taal en landkeuze van de browser.

Wijzig de code in de method localeResolver:

```
return new SessionLocaleResolver();
```

#### 4.8.4 LocaleChangeInterceptor

De gebruiker kiest zijn taal via hyperlinks Nederlands of English.

Voeg regels toe aan index.html, na </p>:

```
<ul>
    <li><a th:href="@{/ (locale=n1_BE)}">Nederlands</a></li>
    <li><a th:href="@{/ (locale=en_US)}">English</a></li>
</ul>
```

❶

(1) Als de gebruiker op deze hyperlink klikt, doet de browser een request naar /?locale=n1\_BE.

##### 4.8.4.1 Interceptor

Je kan bij veel websites je taal kiezen op *elke* pagina. De browser stuurt die request telkens naar een andere controller. Het zou omslachtig zijn de request in elke controller te verwerken en de taalkeuze door te geven aan de LocaleResolver bean.

Een betere oplossing is de class LocaleChangeInterceptor.

Vooraleer LocaleChangeInterceptor te gebruiken, leer je wat een interceptor is.

Dit is een class die Spring oproept telkens een browser request binnenkomt en *voor* Spring die request naar de Controller stuurt.

Een interceptor implementeert de interface HandlerInterceptor.

#### 4.8.4.2 De methods in de interface HandlerInterceptor

- `preHandle`  
Spring roept `preHandle` op *voor* de controller de request verwerkt.
- `postHandle`  
Spring roept `postHandle` op *nadat* de controller method de request verwerkt, maar voor de Thymeleaf pagina de request verwerkt.
- `afterCompletion`  
Spring roept `afterCompletion` op nadat ook de Thymeleaf pagina de request verwerkt.

#### 4.8.4.3 LocaleChangeInterceptor

`LocaleChangeInterceptor` is een interceptor van het Spring framework.

Als de request een parameter `locale` bevat, roept de interceptor de method `setLocale` van de `SessionLocaleResolver` bean op en geeft die request parameter mee.

De bean onthoudt de taal en het land dat de gebruiker gekozen heeft als HTTP session variabele.

Je moet elke interceptor registreren. Je doet dit in `WebConfig`.

De class moet daartoe de interface `WebMvcConfigurer` implementeren:

```
class WebConfig implements WebMvcConfigurer
```

Je doet de registratie zelf in de method `addInterceptors` uit in de interface `WebMvcConfigurer`:

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new LocaleChangeInterceptor());
}
```

Je kan de website proberen.

#### 4.8.5 CookieLocaleResolver

Jij vraagt de taal (en eventueel het land) aan de gebruiker. Je hebt deze code al in `index.html`.

De `CookieLocaleResolver` bean onthoudt de keuze in een cookie.

Zolang de gebruiker geen taal of land kiest, gebruikt Spring de taal en landkeuze van de browser.

Maake een constante:

```
private static final int ZEVEN_DAGEN = 604_800;
```

Wijzig de code in de method `localeResolver`:

```
var resolver = new CookieLocaleResolver();
resolver.setCookieMaxAge(ZEVEN_DAGEN);
return resolver;
```

❶

- (1) `cookieMaxAge` bepaalt de levensduur (in seconden) van de cookie.

Je hoeft niets anders te wijzigen. Je kan de website proberen.



Commit de sources. Publiceer op je remote repository.



Weekend: zie taken

## 5 SPRING SECURITY

### 5.1 Security woordenschat

- **Principal**  
De gebruiker (of applicatie) die je applicatie wil gebruiken.
- **Authentication**
  - Identificatie van de principal.  
De principal geeft zijn identiteit aan via zijn user id.
  - Controle of de principal de identiteit heeft die hij beweert te hebben.  
De principal bewijst zijn identiteit door een bijbehorend paswoord te typen.
- **Authority**  
Een rol die een principal speelt. Voorbeelden: manager, magazijnier.  
Er is een veel op veel relatie tussen authority en principal:
  - Meerdere principals kunnen eenzelfde authority hebben.  
De managers van een firma (Joe en Jack) hebben beiden de authority manager.
  - Één principal kan meerdere authorities hebben.  
In de firma heeft Averell twee authorities: helpdeskmedewerker en magazijnier.

		Principals		
		Joe	Jack	Averell
Authorities	Manager	✓	✓	
	Helpdeskmedewerker			✓
	Magazijnier			✓

- **Authorization**  
Welke authorities mogen welke use case uitvoeren ? Voorbeelden:

Use case	Authorities die de use case mogen uitvoeren
Stockopname	Magazijnier
Weddeverhoging	Manager
Personeelslijst	Manager, Helpdeskmedewerker

- **Access control**  
Een principal wil een use case uitvoeren. Jij controleer eerst of hij die use case *mag* uitvoeren.  
Vb. Averell wil de use case Stockopname uitvoeren.  
Dit mag: Averell heeft de authority Magazijnier. Die mag Stockopname uitvoeren.

### 5.2 Database

Je gebruikt verder in dit hoofdstuk de database beveiligd. Je maakt die met `beveiligd.sql`.

### 5.3 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ `be.vdab` bij Group en `beveiligd` bij Artifact.
4. Kies 11 bij Java Version. Kies Next.
5. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web
Template Engines	Thymeleaf
Security	Spring Security
SQL	MySQL Driver, JDBC API

6. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

## 5.4 application.properties

```
spring.datasource.url=jdbc:mysql://localhost/beveiligd
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED
```

## 5.5 Controllers

### 5.5.1 IndexController

```
package be.vdab.beveiligd.controllers;
// enkele imports
@Controller
@RequestMapping("/")
class IndexController {
    @GetMapping
    public String index() {
        return "index";
    }
}
```

### 5.5.2 OffertesController

```
package be.vdab.beveiligd.controllers;
// enkele imports
@Controller
@RequestMapping("/offertes")
class OffertesController {
    @GetMapping
    public String offertes() {
        return "offertes";
    }
}
```

### 5.5.3 WerknemersController

```
package be.vdab.beveiligd.controllers;
// enkele imports
@Controller
@RequestMapping("/werknemers")
class WerknemersController {
    @GetMapping
    public String werknemers() {
        return "werknemers";
    }
}
```

## 5.6 CSS

Maak in de folder static een folder css. Maak daarin security.css:

```
body {
    font-family: sans-serif;
}
input, button {
    display: block;
    margin-top: 0.4em;
}
```

## 5.7 Thymeleaf

### 5.7.1 index.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Welkom</title>
    <link rel="stylesheet" th:href="@{/css/security.css}"></link>
  </head>
  <body>
    <h1>Welkom</h1>
    <ul>
      <li><a th:href="@{/offertes}">Offertes</a></li>
      <li><a th:href="@{/werknemers}">Werknemers</a></li>
    </ul>
  </body>
</html>
```

### 5.7.2 offertes.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Offertes</title>
    <link rel="stylesheet" th:href="@{/css/security.css}"></link>
  </head>
  <body>
    <h1>Offertes</h1>
    <a th:href="@{/}">Welkom</a>
  </body>
</html>
```

### 5.7.3 werknemers.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Werknemers</title>
    <link rel="stylesheet" th:href="@{/css/security.css}"></link>
  </head>
  <body>
    <h1>Werknemers</h1>
    <a th:href="@{/}">Welkom</a>
  </body>
</html>
```

## 5.8 Default security

Spring maakt standaard één gebruiker. Hij heeft de gebruikersnaam user.

Spring maakt een bijbehorende random paswoord bij de start van de applicatie.

Je vindt dit paswoord in de boodschappen bij het starten van de applicatie in IntelliJ.

Voor het paswoord staat de tekst Using generated security password:

Je bezoekt de website met een browser. Je moet inloggen als je een eerste pagina bezoekt.

Spring Security onthoudt bij het inloggen dat je ingelogd bent als een Http session attribuut.

Als je een andere pagina van de website bezoekt moet je zo niet meer inloggen.



## 5.9 Security zelf configureren

Je configureert *zelf* de security. Je overschrijft dan de default security.

Maak een package `be.vdab.beveiligd.security` en daarin een class `SecurityConfig`:

```
package be.vdab.beveiligd.security;
// enkele imports
@EnableWebSecurity
class SecurityConfig extends WebSecurityConfigurerAdapter {
    private static final String MANAGER = "manager";
    private static final String HELPDESKMEDEWERKER = "helpdeskmedewerker";
    private static final String MAGAZIJNIER = "magazijnier";
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser("joe").password("{noop}theboss").authorities(MANAGER)
            .and()
            .withUser("averell").password("{noop}hungry")
            .authorities(HELPDESKMEDEWERKER, MAGAZIJNIER);
    }
    @Override
    public void configure(WebSecurity web) {
        web.ignoring()
            .mvcMatchers("/images/**")
            .mvcMatchers("/css/**")
            .mvcMatchers("/js/**");
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.formLogin();
        http.authorizeRequests(requests -> requests
            .mvcMatchers("/offertes").hasAuthority(MANAGER)
            .mvcMatchers("/werknemers").hasAnyAuthority(MAGAZIJNIER, HELPDESKMEDEWERKER));
    }
}
```

- (1) `@EnableWebSecurity` integreert Spring Security en Spring MVC.
- (2) Je erft van de class `WebSecurityConfigurerAdapter`.  
Je configureert de security door methods van die class te overriden.
- (3) Je overridet de method die de principals definieert.
- (4) Je geeft met `inMemoryAuthentication` aan dat je principals onthoudt in het interne geheugen. Spring maakt dan zelf geen gebruiker met de naam user meer.  
Je maakt op de volgende regel een principal met de naam joe, het paswoord theboss en de authority manager. `{noop}` betekent dat het paswoord niet versleuteld is.  
Je leert verder in de cursus werken met versleutelde paswoorden.
- (5) Je overridet de method die de web eigenschappen van de security configureert.
- (6) Spring Security moet geen beveiliging doen op URL's die passen bij `/images/**`.  
`**` betekent dat het patroon ook subfolders van `/images` bevat.
- (7) Je overridet de method die de HTTP beveiliging van Spring Security configureert.
- (8) De gebruiker authenticceert zich door zijn naam en paswoord te typen in een HTML form.
- (9) Enkel ingelogde gebruikers met de authority manager kunnen de URL `/offertes` aanspreken.
- (10) Enkel ingelogde gebruikers met de authority magazijnier of helpdeskmedewerker kunnen de URL `/werknemers` aanspreken.

Je kan de website proberen.

Als je de eerste keer Offertes of Werknemers kiest, zie je een inlogpagina.

- Als je een onbestaande userid of een verkeerd paswoord typt moet je opnieuw inloggen.
- Als je een bestaande naam en een correct paswoord typt zie je
  - de gewenste pagina als de principal de URL mag aanspreken.
  - een pagina met fout 403 (verboden) als de principal de URL niet mag aanspreken.

Op sommige websites kan een anonieme gebruiker slechts enkele pagina's zien (bvb. de welkompagina en de inlogpagina). Hij moet minstens ingelogd zijn om andere pagina's te zien.

Vervang in de SecurityConfig method configure(HttpSecurity http) ); door:

```
.mvcMatchers("/", "/login").permitAll()           ❶
.mvcMatchers("/**").authenticated();              ❷
```

- (1) Je geeft alle (ook anonieme) gebruikers toegang tot de welkompagina en de loginpagina.
- (2) Voor alle andere URL's moet de gebruiker minstens ingelogd zijn.

De volgorde waarmee je mvcMatchers oproept is belangrijk.

Je moet eerst de oproepen doen met de meest specifieke URL patronen (die zonder wildcards).

Je doet daarna de oproepen met de meer algemene URL patronen (die met wildcards).

Als Spring Security een browser request verwerkt, overloopt hij de URL patronen in de volgorde zoals je ze met mvcMatchers() toegevoegde. Hij gebruikt het eerste URL patroon dat bij de browser request past. Hij negeert de volgende URL patronen.

Als je bovenstaande code zou schrijven als

```
.mvcMatchers("/**").authenticated().
.mvcMatchers("/", "/login").permitAll();
```

zou een browser request naar /login horen bij het URL patroon /\*\*.

Dan zouden verkeerdelijk enkel reeds ingelogde gebruikers de login pagina kunnen bereiken!

Je kan de website proberen.

## 5.10 Verboden pagina

Je maakt een eigen pagina die hoort bij de fout 403 (verboden).

Maak in de folder templates een folder error. Maak daarin 403.html:

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Geen toegang</title>
    <link rel="stylesheet" th:href="@{/css/security.css}"></link>
  </head>
  <body>
    <h1>U hebt geen toegang tot de pagina.</h1>
    <a th:href="@{/}">Welkom</a>
  </body>
</html>
```

Je kan de website proberen.

## 5.11 Inlogpagina

Je vervangt de ingebouwde inlogpagina door een eigen inlogpagina. Maak login.html:

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Inloggen</title>
    <link rel="stylesheet" th:href="@{/css/security.css}"></link>
  </head>
```

```

<body>
  <h1>Inloggen</h1>
  <form th:action="@{/login}" method="post">
    <input name="username" required autofocus placeholder="Gebruikersnaam"> ❷
    <input name="password" type="password" required placeholder="Paswoord"> ❸
    <button>Inloggen</button>
  </form>
</body>
</html>

```

- (1) Je submit de form met een POST request naar dezelfde URL waarop je straks de form toont. Spring Security verwerkt de request en doet authenticatie.
- (2) De name van het invoervak voor de userid moet username zijn.
- (3) De name van het invoervak voor het paswoord moet password zijn.

Maak LoginController:

```

package be.vdab.beveiligd.controllers;
// enkele imports
@Controller
@RequestMapping("/login")
class LoginController {
    @GetMapping
    public String login() {
        return "login";
    }
}

```

Je definieert de URL van de controller als inlogpagina in de SecurityConfig method configure(HttpSecurity http).

Wijzig http.formLogin(); naar:

```
http.formLogin(login->login.loginPage("/login"));
```

Je kan de website proberen.

## 5.12 Login fouten

Als de gebruiker in de inlogpagina een verkeerde userid of paswoord typt, doet Spring Security een redirect terug naar die inlogpagina en voegt een parameter error toe aan de URL.

Je toont dan een foutmelding. Breid login.html uit, na </form>:

```
<div th:if="${param.error}" class="fout">Verkeerde gebruikersnaam of paswoord</div>
```

Je kan de website proberen.

## 5.13 Inloggen

De gebruiker ziet tot nu de inlogpagina als hij de eerste keer een beveiligde pagina opent.

Je kan hem ook een hyperlink aanbieden waarmee hij zelf beslist in te loggen.

Voeg in index.html een regel toe na <ul>:

```
<li><a th:href="@{/login}">Inloggen</a></li>
```

Je kan de website proberen.

## 5.14 Uitloggen

Spring Security houdt de identiteit van de principal bij in een HttpSession variabele.

De principal behoudt dus zijn identiteit zolang die variabele bestaat.

De principal kan zijn identiteit sneller laten vergeten, als je een uitlog knop toevoegt.

Een klik op de button verwijdert de variabele.

Je activeert het uitloggen.

Voeg een opdracht toe in de SecurityConfig method configure(HttpSecurity http):

```
http.logout();
```

Spring Security logt je uit bij een POST request naar de URL `/logout`.

Voeg in `index.html` regels toe voor `</body>`:

```
<form method="post" th:action="@{/logout}">
  <button>Uitloggen</button>
</form>
```

Je kan de website proberen.

Spring Security toont na het afmelden standaard de inlogpagina.

Je kan dit wijzigen naar de welkompagina.

Wijzig in de `SecurityConfig` `http.logout()`; naar:

```
http.logout(logout->logout.logoutSuccessUrl("/"));
```

Je kan de website proberen.

## 5.15 Thymeleaf

Thymeleaf bevat functionaliteit waarmee je security informatie leest.

### 5.15.1 pom.xml

Voeg een dependency toe:

```
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

Klik rechts op het verticale tabblad Maven. Klik daar op .

### 5.15.2 authorize

Je toont met `sec:authorize` een stuk HTML conditioneel, afhankelijk van de authorities van de huidige gebruiker. Je bouwt de conditie op, met één van volgende Spring Security functies:

- `isAnonymous()`  
geeft true terug als de gebruiker niet ingelogd is.
- `isAuthenticated()`  
geeft true terug als de gebruiker ingelogd is.
- `hasAuthority('manager')`  
geeft true terug als de gebruiker ingelogd is en de authority manager heeft.
- `hasAnyAuthority('helpdeskmedewerker', 'magazijnier')`  
geeft true terug als de gebruiker ingelogd is en de authority helpdeskmedewerker of magazijnier heeft.

Voorbeeld 1:

```
<div sec:authorize="hasAuthority('manager')">
  Enkel gebruikers met de role manager zien deze tekst
</div>
```

Voorbeeld 2:

```
<div sec:authorize="isAuthenticated()">
  Enkel ingelogde gebruikers zien deze tekst
</div>
```

Je gebruikt de tag in `index.html`:

- Voeg `xmlns:sec="http://www.thymeleaf.org/extras/spring-security"` toe aan de `<html>` tag.
- Je toont de hyperlink Aanmelden enkel aan niet-ingelogde gebruikers.  
Voeg `sec:authorize="isAnonymous()"` toe aan de `<li>` van het inloggen.
- Je toont de hyperlink Afmelden enkel aan ingelogde gebruikers.  
Voeg `sec:authorize="isAuthenticated()"` toe aan de `<form>` van het uitloggen.

Je kan de website proberen.

Je toont met `sec:authorize-url` een stuk HTML

enkel als de principal toegang heeft tot de URL die je in dit attribuut vermeldt.

- Je toont Offertes enkel als de gebruiker toegang heeft tot `/werknemers`.  
Voeg `sec:authorize-url="/offertes"` toe aan de `<li>` van Offertes.
- Je toont Werknemers enkel als de gebruiker toegang heeft tot `/werknemers`.  
Voeg `sec:authorize-url="/werknemers"` toe aan de `<li>` van Werknemers.

Je kan de website proberen.

### 5.15.3 Naam van de gebruiker

Je toont met `sec:authentication="name"` de naam van de ingelogde gebruiker.

Je gebruikt dit in `index.xml`. Wijzig `<button>Uitloggen</button>` naar:

```
<button><span sec:authentication="name"></span> uitloggen</button>
```

Je kan de website proberen.

## 5.16 Database

Je onthoudt de principals en de authorisation nu in een database.

### 5.16.1 Password encoding



Het is sterk afgeraden om paswoorden letterlijk op te nemen in een database. Als een hacker de database kan stelen, weet hij alle namen en paswoorden. Hij kan inloggen alsof hij één van de gebruikers is.

Je vermijdt dit door de paswoorden in geëncrypteerde (versleutelde) vorm te bewaren.

Er bestaan algoritmes (MD4, MD5, SHA, bcrypt ...) die one-way encryptie doen. Dit betekent dat je

- een leesbaar paswoord kan omzetten in geëncrypteerde vorm.
- een geëncrypteerd paswoord niet kan omzetten naar de leesbare vorm.

Één van de beste algoritmes is bcrypt. Dit algoritme encrypteert bijvoorbeeld het leesbaar paswoord `theboss` naar `$2a$10$3DPuiwz0.I2UYgge1Be8NuCHdd7Jblz2cu8K0ZkkguQZYnCIA4u50`

Als een hacker de database met geëncrypteerde paswoorden steelt heb je geen probleem.

Hij kan `$2a$10$3DPuiwz0.I2UYgge1Be8NuCHdd7Jblz2cu8K0ZkkguQZYnCIA4u50`

niet terugvormen naar `theboss`. De hacker kan dus niet als joe inloggen.

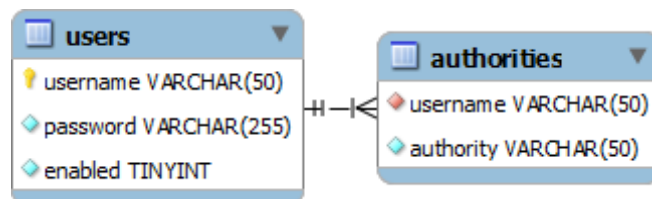
Je kan met volgende code fragment de bcrypt geëncrypteerde versie van een paswoord krijgen:

```
String geencrypteerd = new BCryptPasswordEncoder().encode(origineelPaswoord);
```

### 5.16.2 Default tabel structuren

Je onthoudt de principals en hun authorities in een database.

Spring Security verwacht de informatie standaard in twee tables met volgende structuur:



- De database beveiligd bevat de twee tables:  
users bevat records voor joe en voor averell, authorities bevat hun authorities.
- De kolom password in de table users bevat het geëncrypteerde paswoord:  
theboss voor Joe en hungry voor Averell.  
Voor dit paswoord staat de naam van het encryptie algoritme tussen accolades {bcrypt}.
- De kolom enabled in de table users bevat 1 (true) of 0 (false). Als de kolom false bevat, kan je met de bijbehorende principal niet inloggen. Je kan zo een principal tijdelijk uitschakelen.
- De database mag daarnaast ook andere tables bevatten (bvb. Klanten, Producten, ...).

### 5.16.3 SecurityConfig

Voeg een variabele toe:

```
private final DataSource datasource;
```

IntelliJ stelt voor een geparametriseerd constructor toe te voegen. Doe dit.

Vervang de code in de method `configure(AuthenticationManagerBuilder auth):`  
`auth.jdbcAuthentication().dataSource(datasource);` ❶

Je geeft met `jdbcAuthentication` aan dat je principals uit een database leest.

Je geeft de `DataSource` mee die gebaseerd is op `application.properties`.

Spring security zoekt de principals in de database die bij die `DataSource` hoort.

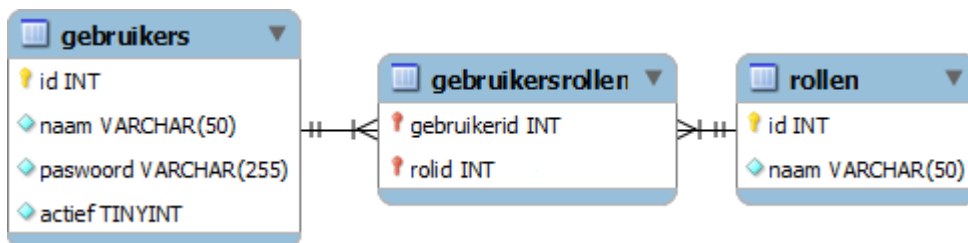
Je kan de website proberen.

### 5.16.4 Afwijkende tabel structuren

De database bevat ook de tables `gebruikers`, `rollen` en `gebruikersrollen`.

Ze bevatten ook informatie over de principals `joe` en `avere11`,

maar hebben een andere structuur dan de default structuur van Spring Security:



Spring Security kan met tables samenwerken die niet de default structuur hebben, als je Spring Security twee SQL statements aanbiedt.

Vervang in de method `configure(AuthenticationManagerBuilder auth)` de `;` door:

```
.usersByUsernameQuery(
    "select naam as username, paswoord as password, actief as enabled" +
    "from gebruikers where naam = ?")
.authoritiesByUsernameQuery(
    "select gebruikers.naam as username, rollen.naam as authorities" +
    "from gebruikers inner join gebruikersrollen" +
    "on gebruikers.id = gebruikersrollen.gebruikerid" +
    "inner join rollen" +
    "on rollen.id = gebruikersrollen.rolid" +
    "where gebruikers.naam = ?"
);
```

- (1) Dit SQL statement leest één gebruiker aan de hand van een gebruikersnaam die als parameter binnenkomt. Het statement geeft maximaal 1 rij terug met 3 kolommen:
  - a. username gebruikersnaam
  - b. password bijbehorende paswoord
  - c. enabled is de gebruiker bruikbaar of uitgeschakeld
- (2) Dit SQL statement leest de authorities van één gebruiker aan de hand van een gebruikersnaam die als parameter binnenkomt. Dit statement geeft een rij terug per authority met 2 kolommen:
  - a. username gebruikersnaam
  - b. authorities authority

Je kan de website proberen.

## 5.17 Authorization met annotations

Je doet de authorization tot nu met Java code in de class SecurityConfig.

Je kan dit ook met annotations doen.

Verwijder in de class SecurityConfig de regels met de authorization:

```
.mvcMatchers("/offertes").hasAuthority(MANAGER)
.mvcMatchers("/werknemers").hasAnyAuthority(MAGAZIJNIER, HELPDESKMEDEWERKER)
```

Je activeert de authorization met annotations. Typ voor de class SecurityConfig:

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

### 5.17.1 @PreAuthorize

Je geeft aan @PreAuthorize een expressie mee. Spring Security voert de method, waarvoor je @PreAuthorize schrijft, enkel uit als deze expressie true terug geeft.

Voorbeelden

- ```
@PreAuthorize("hasAuthority('manager')")
public void beveiligd()
```

 Enkel gebruikers met de authority manager kunnen beveiligd uitvoeren.
- ```
@PreAuthorize("hasAnyAuthority('helpdeskmedewerker', 'magazijnier')")
public void beveiligd()
```

 Enkel gebruikers met de authority helpdeskmedewerker of magazijnier kunnen beveiligd uitvoeren.

Je kan @PreAuthorize schrijven

- Voor een class @PreAuthorize geldt dan voor alle methods van die class.
- Voor een method @PreAuthorize geldt dan enkel voor die method.

Als je bij een class én bij een method @PreAuthorize typt, overschrijft @PreAuthorize bij de method @PreAuthorize bij de class.

Typ voor de class OffertesController:

```
@PreAuthorize("hasAuthority('manager')")
```

Typ voor de class WerknemersController:

```
@PreAuthorize("hasAnyAuthority('helpdeskmedewerker', 'magazijnier')")
```

Je kan de website proberen.



Commit de sources. Publiceer op je remote repository.



Firma

## 6 OAUTH

Je leerde reeds dat een gebruiker moet inloggen op je beveiligde website voor hij bepaalde pagina's kan zien.

Elke website heeft dan zijn eigen database met gebruikers.

De gebruiker moet zich op elke website registreren met een gebruikersnaam en een paswoord.

Na een tijdje is het moeilijk om al die namen en paswoorden te onthouden.

OAuth is een open standaard die het volgende toelaat.

De gebruiker moet zich niet op jouw website aanmelden.

Hij meldt zich aan op een andere website (google, github, ...).

Nadat hij daar aangemeld is, is hij ook een gekende gebruiker op jouw website.

### 6.1 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ be.vdab bij Group.
4. Typ oauth bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web
Template Engines	Thymeleaf
Security	Spring Security, OAuth2 Client

7. Kies Next, Finish.

Voeg een dependency toe aan pom.xml:

```
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

Klik rechts op het verticale tabblad Maven. Klik daar op .

Maak van het project een GIT project. Maak een bijbehorende remote repository.

Je website zal werken met gebruikers die zich aanmelden bij GitHub.

### 6.2 Registreren bij GitHub

Je moet één keer je website registreren bij GitHub.

1. Surf naar <https://github.com/settings/applications/new>.
2. Typ een vrij te kiezen naam voor je website bij Application name.
3. Typ <http://localhost:8080> bij Homepage URL.
4. Typ <http://localhost:8080/login/oauth2/code/github> bij Authorization callback URL. Het onderdeel /login/oauth2/code/github is hetzelfde bij elke website die je maakt.
5. Kies **Register application**.
6. Laat de pagina die daarna komt openstaan. Je hebt de informatie straks nodig.

Voeg in je project regels toe application.properties:

```
spring.security.oauth2.client.registration.github.client-id=XXX
spring.security.oauth2.client.registration.github.client-secret=YYY
```

❶  
❷

(1) Je vervangt XXX door de code bij Client ID die openstaat in je browser.

(2) Je vervangt YYY door de code bij Client Secret die openstaat in je browser.



## 6.3 SecurityConfig

```
package be.vdab.oauth;
// enkele imports
@EnableWebSecurity
class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.oauth2Login();
        http.authorizeRequests(requests -> requests
            .mvcMatchers("/").permitAll()
            .mvcMatchers("/beveiligd").authenticated());
    }
}
```

- (1) Je geeft aan dat je OAuth gebruikt. De gebruiker zal dus inloggen op een *andere* website.
- (2) Iedereen heeft toegang tot de beginpagina.
- (3) Enkel ingelogde gebruikers hebben toegang tot de pagina op de URL beveiligd. Als ze de pagina de eerste keer openen, moeten ze eerst inloggen op GitHub.

## 6.4 Controllers

### 6.4.1 IndexController

```
package be.vdab.oauth.controllers;
// enkele imports
@Controller
@RequestMapping("/")
class IndexController {
    @GetMapping
    public String index() {
        return "index";
    }
}
```

### 6.4.2 BeveiligdeController

```
package be.vdab.oauth.controllers;
// enkele imports
@Controller
@RequestMapping("/beveiligd")
class BeveiligdeController {
    @GetMapping
    public ModelAndView securedPage(@AuthenticationPrincipal OAuth2User user) {
        return new ModelAndView("beveiligd", "gebruikersNaam",
            user.getAttribute("login"));
    }
}
```

- (1) De controller verwerkt requests naar de beveiligde URL beveiligd.
- (2) Bij een request naar die URL moet de gebruiker eerst inloggen op GitHub. Pas als dat lukt komt de request in deze method binnen. Je geeft de method een OAuth2User parameter. Je typt voor de parameter @AuthenticationPrincipal. Spring vult de parameter met eigenschappen over de ingelogde gebruiker.
- (3) Je vraagt de eigenschap login. Dit is de gebruikersnaam van de gebruiker op GitHub. Je geeft die gebruikersnaam door aan de Thymeleaf pagina.

## 6.5 Thymeleaf

### 6.5.1 index.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Welkom</title>
  </head>
  <body>
    <a th:href="@{\beveiligd}">Beveiligde pagina</a>
  </body>
</html>
```

### 6.5.2 beveiligd.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
  <head>
    <title>Beveiligd</title>
  </head>
  <body>
    <h1 th:text="${gebruikersNaam}"></h1>
    <div sec:authorize="hasAuthority('ROLE_USER')">
      Je hebt de authority ROLE_USER.
    </div>
  </body>
</html>
```

❶

(1) Gebruiker die inloggen met OAuth hebben automatisch de authoriy ROLE\_USER.

Zorg dat je uitgelogd bent op GitHub.

Start je website. Klik op beveiligd. Je wordt omgeleid naar GitHub. Log daar in.  
Je komt daarna terug in je website, op de pagina Beveiligd.



Commit de sources. Publiceer op je remote repository.

## 7 SPRING DATA

### 7.1 Repository interfaces

Je maakt tot nu, in de repositories laag, twee onderdelen per entity:

- een interface (bijvoorbeeld `KlantRepository`)
- een class die de interface implementeert (bijvoorbeeld `JpaKlantRepository`)

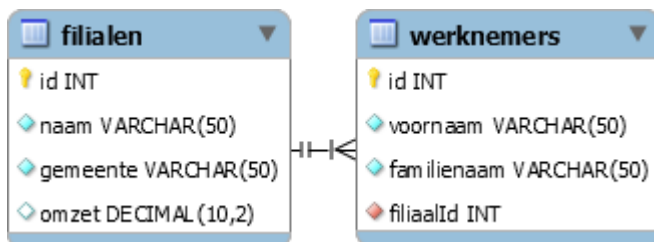
Je gebruikt hier de library Spring Data.

- Je maakt dan enkel interfaces in de repositories laag.
- Spring Data maakt voor jou classes die de interfaces implementeren, bij de start van je applicatie. Spring Data maakt van elke class een Spring bean. Jij kan deze beans injecteren in je services laag.

Je wint veel tijd omdat je de classes niet *zelf* maakt.

### 7.2 Database

Voer het script `springdata.sql` uit. Het maakt een database `springdata` met twee tables:



De tables zijn leeg. Je zal records toevoegen gedurende tests.

### 7.3 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typt `be.vdab` bij Group.
4. Typ `springdata` bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
SQL	Spring Data JPA, MySQL Driver

7. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 7.4 application.properties

```

spring.datasource.url=jdbc:mysql://localhost/springdata
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.test.database.replace=none
logging.level.org.hibernate.SQL=DEBUG
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED
  
```

## 7.5 Entities

### 7.5.1 Filiaal

```
package be.vdab.springdata.domain;
// enkele imports ...
@Entity
@Table(name = "filialen")
public class Filiaal {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String naam;
    private String gemeente;
    private BigDecimal omzet;
    // constructor met parameters: naam, gemeente, omzet
    // protected default constructor
    // getters
}
```

### 7.5.2 Werknemer

```
package be.vdab.springdata.domain;
// enkele imports ...
@Entity
@Table(name = "werknemers")
public class Werknemer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String voornaam;
    private String familienaam;
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "filiaalid")
    private Filiaal filiaal;
    // getters
}
```

## 7.6 JpaRepository

Maak de interface FiliaalRepository:

```
package be.vdab.springdata.repositories;
// enkele imports
public interface FiliaalRepository extends JpaRepository<Filiaal, Long> { ❶
}
```

- (1) Je erft je repository interface van JpaRepository. Je typt tussen < en > twee types:  
 Filiaal: het type van de entity die bij je repository hoort.  
 Long: het type van private variabele die hoort bij de primary key.

Je interface erft veel handige methods. De belangrijkste methods:

- `count()` Het aantal entiteiten tellen.
- `findById(id)` Een entity zoeken op zijn primary key waarde.
- `findAll()` Alle entiteiten lezen, zonder een bepaalde volgorde.
- `findAll(sort)` Alle entiteiten lezen, met een bepaalde volgorde.
- `findAllById(List of Set)` De entiteiten lezen waarvan de primary key voorkomt in een List of Set.
- `save(entity)` Een entity toevoegen (create), als die nieuw is, of een entity wijzigen (update) als die al bestaat.
- `deleteById(id)` Een entity verwijderen waarvan je de primary key meegeeft

Je leert deze methods kennen aan de hand van tests.

Maak in de folder test een folder resources.

Maak daarin het bestand spring.properties:

```
spring.test.constructor.autowire.mode=all
```

Maak daar ook het bestand insertFilialen.sql:

```
insert into filialen(naam, gemeente,omzet) values ('Alfa', 'Brussel', 1000);
insert into filialen(naam, gemeente,omzet) values ('Bravo', 'Brussel', 2000);
insert into filialen(naam, gemeente,omzet) values ('Charly', 'Antwerpen', 3000);
```

Maak in de folder test/java een package be.vdab.springdata.repositories.

Maak daarin de class FiliaalRepositoryTest:

```
package be.vdab.springdata.repositories;
import static org.assertj.core.api.Assertions.assertThat;
// enkele andere imports
@DataJpaTest
@Sql("/insertFilialen.sql")
class FiliaalRepositoryTest
    extends AbstractTransactionalJUnit4SpringContextTests {
    private static final String FILIALEN = "filialen";
    private final FiliaalRepository repository;
    // constructor met parameter
    private long idVanAlfa() {
        return super.jdbcTemplate.queryForObject(
            "select id from filialen where naam = 'Alfa'", Long.class);
    }
    private long idVanBravo() {
        return super.jdbcTemplate.queryForObject(
            "select id from filialen where naam = 'Bravo'", Long.class);
    }
    @Test
    void count() {
        assertThat(repository.count()).isEqualTo(super.countRowsInTable(FILIALEN)); ❶
    }
    @Test
    void findById() {
        var optionalFiliaal = repository.findById(idVanAlfa());
        assertThat(optionalFiliaal.get().getNaam()).isEqualTo("Alfa"); ❷
    }
    @Test
    void findAll() {
        var filialen = repository.findAll();
        assertThat(filialen).hasSize(super.countRowsInTable(FILIALEN)); ❸
    }
    @Test
    void findAllGesorteerdOpGemeente() {
        var filialen = repository.findAll(Sort.by("gemeente"));
        assertThat(filialen).hasSize(super.countRowsInTable(FILIALEN)); ❹
        assertThat(filialen).extracting(filiaal -> filiaal.getGemeente()).isSorted();
    }
    @Test
    void findAllById() {
        var idAlfa = idVanAlfa();
        var idBravo = idVanBravo();
        var filialen = repository.findAllById(Set.of(idVanAlfa(), idVanBravo())); ❺
        assertThat(filialen).extracting(filiaal -> filiaal.getId())
            .containsOnly(idAlfa, idBravo);
    }
}
```

```

@Test
void save() {
    var filiaal = new Filiaal("Delta", "Brugge" , BigDecimal.TEN);
    repository.save(filiaal);
    var id = filiaal.getId();
    assertThat(id).isPositive();
    assertThat(super.countRowsInTableWhere(FILIALEN, "id=" + id)).isOne();
}
@Test
void deleteById() {
    var id = idVanAlfa();
    repository.deleteById(id);
    repository.flush();
    assertThat(super.countRowsInTableWhere(FILIALEN, "id=" + id)).isZero();
}
@Test
void deleteByOnbestaandeId() {
    assertThatExceptionOfType(EmptyResultDataAccessException.class).isThrownBy(
        () -> repository.deleteById(-1L));
}
}

```

- (1) count geeft het aantal entities.
- (2) Je zoekt met findById een entity aan de hand van zijn primary key waarde.
- (3) Je zoekt met findAll alle entities.
- (4) Je kan met findAll sorteren. Sort.by bevat het Filiaal attribuut waarop je sorteert.
- (5) Je zoekt met findAllById de entities waarvan je de primary key waarden meegeeft in een List of Set.
- (6) Je bewaart met save een entity in de database.
- (7) Je verwijdert met deleteById een entity in de database.  
Je geeft als parameter de primary key waarde van de entity mee.
- (8) JPA stelt het verwijderen van entities standaard uit tot het einde van de transactie.  
Je doet de verwijderingen nu met flush.
- (9) Spring Data werpt een EmptyResultDataAccessException als het filiaal niet bestaat.

Voer de tests uit. Ze lukken.

## 7.7 Derived query methods

Je kan in je repository interface method declaraties toevoegen. Je typt de naam van zo'n method volgens een conventie. Spring Data maakt dan een implementatie van de method.

De method stuurt een JPQL query naar de database en geeft de bijbehorende entities terug.

### 7.7.1 Filteren

Je wil de filialen uit één gemeente.

Voeg volgende method declaratie toe aan FiliaalRepository:

```
List<Filiaal> 1 findByGemeente(2 String 3 gemeente); 4
```

- (1) Je gebruik List<Filiaal> als return type als de query *meerdere* filialen terug geeft.  
Je gebruikt Optional<Filiaal> als de query maximaal één filiaal teruggeeft.
- (2) Je begint de method altijd met findBy.
- (3) Je typt het filiaal attribuut waarop je een where onderdeel wil in de query: Gemeente.
- (4) Je geeft de method een parameter. Als je straks de method oproept, vermeld je hier de gemeente waarvan je filialen wil. Voorbeeld: findByGemeente("Brussel").

Naarmate je de method declaratie typt, krijg je pop-up assistentie van IntelliJ.

### 7.7.2 Implementatie

Spring Data implementeert de method. Die stuurt volgende JPQL query naar de database.

```
select f from Filiaal f where f.gemeente = ?
```

### 7.7.3 Test

Test deze method met een extra test in FiliaalRepositoryTest:

```
@Test
void findByGemeente() {
    var filialen = repository.findByGemeente("Brussel");
    assertThat(filialen).hasSize(2)
        .allSatisfy(filiaal ->
            assertThat(filiaal.getGemeente()).isEqualTo("Brussel"));
}
```

Voer de test uit. Hij lukt.

### 7.7.4 Filiteren: tweede voorbeeld

Voeg een method declaratie toe aan FiliaalRepository:

```
List<Filiaal> findByOmzetGreaterThanOrEqualTo(BigDecimal vanaf);
```

❶
❷
❸

- (1) Je typt het filiaal attribuut waarop je een where onderdeel wil in de query: Omzet.
- (2) Je typt GreaterThanOrEqualTo. Je geeft zo aan dat je de filialen wil waarvan de omzet (❶) groter of gelijk is aan een waarde.
- (3) Je geeft de method een parameter. Als je de method oproept, vermeld je hier vanaf welke omzet je de filialen wil. Voorbeeld: findByOmzetGreaterThanOrEqualTo(BigDecimal.TEN).

Test deze method met een extra test in FiliaalRepositoryTest:

```
@Test
void findByOmzetGreaterThanOrEqualTo() {
    var tweeduizend = BigDecimal.valueOf(2_000);
    var filialen = repository.findByOmzetGreaterThanOrEqualTo(tweeduizend);
    assertThat(filialen).hasSize(2)
        .allSatisfy(filiaal ->
            assertThat(filiaal.getOmzet()).isGreaterThanOrEqualTo(tweeduizend));
}
```

Voer de test uit. Hij lukt.

### 7.7.5 Andere voorbeelden

Andere voorbeelden van methods:

- findByOmzetIsNull  
Filialen waarvan de omzet niet gekend is.
- findByOmzetBetween(BigDecimal van, BigDecimal tot)  
Filialen waarvan de omzet ligt tussen van en tot.
- findByNaamStartingWith(String woord)  
Filialen waarvan de naam begint met de letters in woord.
- findByNaamContaining(String woord)  
Filialen waarvan de naam de letters in woord bevat.
- findByNaamIn(Set<String> namen)  
Filialen waarvan de naam één van de namen in de verzameling namen is.
- findByNaamAndGemeente(String naam, String gemeente)  
Filialen waarvan de naam gelijk is aan naam én de gemeente gelijk is aan gemeente.

### 7.7.6 Sorteren

Je kan in de method declaratie ook het Filiaal attribuut aangeven waarop je wil sorteren.

Wijzig de eerste method declaratie in FiliaalRepository:

```
List<Filiaal> findByGemeenteOrderByNaam(String gemeente);
```

❶

(1) Je typt OrderBy. Je typt daarna het attribuut waarop je wil sorteren.

Wijzig in FiliaalRepositoryTest de method findByGemeente:

```
@Test
void findByGemeenteOrderByNaam() {
    var filialen = repository.findByGemeenteOrderByNaam("Brussel");
    assertThat(filialen).hasSize(2)
        .allSatisfy(
            filiaal -> assertThat(filiaal.getGemeente()).isEqualTo("Brussel"))
        .extracting(filiaal -> filiaal.getNaam()).isSorted();
}
```

Voer de test uit. Hij lukt.

### 7.7.7 Tellen

Je kan een method declaratie beginnen met countBy.

Spring Data maakt dan een implementatie die een aantal filialen *telt*.

Voeg een method declaratie toe aan FiliaalRepository:

```
int countByGemeente(String gemeente);
```

Voeg een bijbehorende test toe aan FiliaalRepositoryTest:

```
@Test
void countByGemeente() {
    assertThat(repository.countByGemeente("Brussel")).isEqualTo(2);
}
```

Voer de test uit. Hij lukt.

## 7.8 @Query

Je kan niet alle query's die je nodig hebt uitdrukken met een method naam volgens de conventie.

Voorbeeld: `select avg(f.omzet) from Filiaal f`

Je typt de query dan zelf in @Query.

Voeg een method declaratie toe aan FiliaalRepository:

```
@Query("select avg(f.omzet) from Filiaal f")
BigDecimal findGemiddeldeOmzet();
```

❶

❷

(1) Je typt @Query voor de method. Je geeft je query mee als parameter.

(2) Je mag de methodnaam vrij kiezen.

Voeg een bijbehorende test toe aan FiliaalRepositoryTest:

```
@Test
void findGemiddeldeOmzet() {
    assertThat(repository.findGemiddeldeOmzet()).isEqualToByComparingTo("2000");
}
```

Voer de test uit. Hij lukt.



## 7.9 Named query's

Sommige query's zijn lang. Ze zijn onleesbaar als je ze typt in @Query.

Een oplossing is zo'n query te typen in orm.xml, zoals in de cursus JPA.

Maak in de folder java/resources een folder META-INF.

Maak daarin het bestand orm.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<entity-mappings xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm
    http://xmlns.jcp.org/xml/ns/persistence/orm_2_1.xsd" version="2.1">
  <named-query name="Filiaal.findMetHoogsteOmzet">
    <query>
      select f from Filiaal f where f.omzet =
        (select max(f.omzet) from Filiaal f)
    </query>
  </named-query>
</entity-mappings>
```

❶

- (1) Je maakt een named query met de naam Filiaal.findMetHoogsteOmzet.  
Je begint de naam met het type entity dat je zoekt: Filiaal.

Voeg een method declaratie aan FiliaalRepository:

```
List<Filiaal> findMetHoogsteOmzet();
```

❶

- (1) Je geeft de method dezelfde naam als het einde van de naam van de named query.  
Spring Data maakt een implementatie die de named query oproept.

Voeg een bijbehorende test toe aan FiliaalRepositoryTest:

```
@Test
void findMetHoogsteOmzet() {
    var filialen = repository.findMetHoogsteOmzet();
    assertThat(filialen).hasSize(1);
    assertThat(filialen.get(0).getNaam()).isEqualTo("Charly");
}
```

Voer de test uit. Hij lukt.

## 7.10 Gerelateerde entities

Je gebruikt tot nu één type entity: Filiaal.

Je maakt nu voorbeelden van Werknemer entities en hun relatie met Filiaal entities.

Maak in src/test/resources het bestand insertWerknemers.sql:

```
insert into werknemers(voornaam, familienaam, filiaalid)
  values ('Joe', 'Dalton', (select id from filialen where naam='Alfa'));
insert into werknemers(voornaam, familienaam, filiaalid)
  values ('Jack', 'Dalton', (select id from filialen where naam='Bravo'));
insert into werknemers(voornaam, familienaam, filiaalid)
  values ('Lucky', 'Luke', (select id from filialen where naam='Charly'));
```

### 7.10.1 Filteren op attributen van gerelateerde entities

Maak de interface WerknemerRepository:

```
package be.vdab.springdata.repositories;
// enkele imports
public interface WerknemerRepository extends JpaRepository<Werknemer, Long> {
    List<Werknemer> findByFiliaalGemeente(String gemeente);
}
```

❶

- (1) Na findBy staat Filiaal: het attribuut filiaal van de entity Werknemer.  
Daarna staat Gemeente: het attribuut gemeente van de entity Filiaal.  
Spring Data maakt een implementatie die de werknemers zoekt die behoren tot filialen uit een bepaalde gemeente. Je geeft die gemeente mee in de parameter gemeente.

Maak WerknemerRepositoryTest:

```
package be.vdab.springdata.repositories;
import static org.assertj.core.api.Assertions.assertThat;
// enkele andere imports
@DataJpaTest
@Sql("/insertFilialen.sql")
@Sql("/insertWerknemers.sql")
class WerknemerRepositoryTest
    extends AbstractTransactionalJUnit4SpringContextTests {
    private final WerknemerRepository repository;
    // constructor met parameter
    @Test
    void findByFiliaalGemeente() {
        var antwerpen = "Antwerpen";
        var werknemers = repository.findByFiliaalGemeente(antwerpen);
        assertThat(werknemers).hasSize(1);
        assertThat(werknemers.get(0).getFiliaal().getGemeente()).isEqualTo(antwerpen);
    }
}
```

Voer de test uit. Hij lukt.

### 7.10.2 @EntityGraph

Je leerde in de JPA query het N+1 probleem kennen.

Je leert hier hoe je het oplost in Spring Data met @EntityGraph.

Voeg een method declaratie toe aan WerknemerRepository:

```
List<Werknemer> findByVoornaamStartingWith(String woord);
```

Voeg een bijbehorende test toe aan WerknemerRepositoryTest:

```
@Test
void findByVoornaamStartingWith() {
    var werknemers = repository.findByVoornaamStartingWith("J");
    assertThat(werknemers).hasSize(2).allSatisfy(werknemer ->
        assertThat(werknemer.getVoornaam().startsWith("J")));
    assertThat(werknemers).extracting(
        werknemer -> werknemer.getFiliaal().getNaam());
}
```

- (1) Je haalt van elke werknemer de naam van het bijbehorende filiaal op.

Voer de test uit. Je kijkt naar de SQL statements die JPA naar de database stuurde.

Je ziet het N+1 probleem: JPA stuurt een SQL statement om de werknemers te lezen en stuurt daarna per filiaal een extra SQL statement naar de database om dat filiaal te lezen.

Typ volgende regel voor de class Werknemer (zie JPA cursus) :

```
@NamedEntityGraph(name = "Werknemer.metFiliaal",
    attributeNodes = @NamedAttributeNode("filiaal"))
```

Typ in WerknemerRepository een regel voor de method findByVoornaamStartingWith:

```
@EntityGraph(value = "Werknemer.metFiliaal")
```

- (1) Je verwijst naar de named entity graph Werknemer.metFiliaal.

Spring Data zal daarmee rekening houden bij het uitvoeren van de method.

Voer de test opnieuw uit. Je kijkt naar de SQL statements die JPA naar de database stuurde.

Je ziet dat het N+1 probleem is opgelost. JPA stuurt nu één SQL statement naar de database. Dit statement leest de werknemers én de bijbehorende filialen.

## 7.11 Pagineren

Als je de gebruiker *veel* data toont, is het gebruikelijk niet alle data in één keer te tonen. Voorbeeld: de gebruiker wil alle werknemers zien. Je toont hem de eerste 20 werknemers. Je toont hem ook een knop Volgende. Als hij daarop klikt ziet hij de volgende werknemers, ... Het zou niet performant zijn *alle* werknemers uit de database te lezen als je er maar 20 toont. Bij MySQL kan je het keyword limit gebruiken: `select ... from werknemers limit 20`.

Je erft in `WerknemerRepository` van `JpaRepository` een method dit gebruikt:

```
Page<Werknemer> findAll(Pageable pageable)
```

De parameter heeft als type `Pageable`. Je geeft daarin mee:

- het volgnummer van de pagina die je wil
- en hoeveel werknemers de pagina maximaal mag bevatten.

Het return type is `Page<Werknemer>`. Dit bevat

- de werknemers op de gevraagde pagina
- en extra informatie zoals: is dit de laatste pagina ?

Voeg tests toe aan `WerknemerRepositoryTest` om dit te proberen:

```
@Test
void eerstePagina() {
    var page = repository.findAll(PageRequest.of(0, 2));
    assertThat(page.getContent()).hasSize(2);
    assertThat(page.hasPrevious()).isFalse();
    assertThat(page.hasNext()).isTrue();
}
@Test
void tweedePagina() {
    var page = repository.findAll(PageRequest.of(1, 2));
    assertThat(page.getContent()).hasSize(1);
    assertThat(page.hasPrevious()).isTrue();
    assertThat(page.hasNext()).isFalse();
}
```

- (1) De parameter van `findAll` heeft als type de interface `Pageable`.  
De method `of` van `PageRequest` heeft een object terug dat die interface implementeert.  
Je geeft als eerste parameter het volgnummer van de pagina mee die je vraagt.  
Je geeft als tweede parameter het aantal werknemers mee die je per pagina vraagt.
- (2) `getContent` geeft je de verzameling werknemers op de gevraagde pagina.
- (3) `hasPrevious` geeft je `false` als er geen vorige pagina is.
- (4) `hasNext` geeft je `true` als er een volgende pagina is.
- (5) Je vraagt de tweede pagina met maximaal twee werknemers.
- (6) Er zijn drie werknemers in totaal. De tweede pagina bevat dus één werknemer.
- (7) Er is een vorige pagina.
- (8) Er is geen volgende pagina.



Als je zelf een method toevoegt aan `WerknemerRepository`, kan die ook pagineren:  
`Page<Werknemer> findByVoornaam(String voornaam, Pageable pageable);`



Spring Data ondersteunt ook het aanspreken van niet-relatieve databases:  
MongoDB, Redis, Cassandra, Neo4J, ...



Voert alle testen uit. Commit de sources. Publiceert op je remote repository.



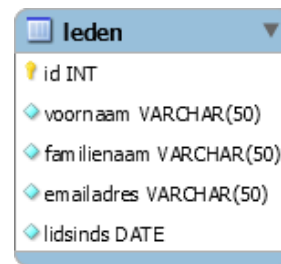
Artikels en artikelgroepen

## 8 MAIL

Je kan met Spring vanuit je code mails versturen.

### 8.1 Database

Voer het script mail.sql uit.  
Het maakt een database mail.  
De database bevat één table:



leden	
id	INT
voornaam	VARCHAR(50)
familienam	VARCHAR(50)
emailadres	VARCHAR(50)
lidsinds	DATE

### 8.2 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ be.vdab bij Group.
4. Typ mail bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web
Template Engines	Thymeleaf
SQL	Spring Data JPA, MySQL Driver
I/O	Java Mail Sender

7. Kies Next, Finish.

Je voegt de validatie dependency toe aan pom.xml:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Klik rechts op het verticale tabblad Maven. Klik daar op .

Maak van het project een GIT project. Maak een bijbehorende remote repository.

De gebruiker zal zich in je website registreren als een nieuw lid.

Je voegt dan een record toe aan de database leden.

Je stuurt ook een mail naar het nieuwe lid.

### 8.3 application.properties

```
spring.datasource.url=jdbc:mysql://localhost/mail
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.jpa.hibernate.naming.physical-strategy=\
org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED
```

## 8.4 Mail configuratie

Zoals een applicatie een database server aanspreekt om met een database te werken, spreekt een applicatie een mailserver aan om mails te versturen.

Je moet in de cursus geen mailserver installeren.

Als je een Google account hebt, kan je de mailserver van Google gebruiken.

Voeg regels, met eigenschappen van de mailserver, toe aan `application.properties`.

Let er op dat de regels op het einde geen overtollige spaties bevatten !

```
spring.mail.host=smtp.gmail.com
spring.mail.port=465
spring.mail.protocol=smtps
spring.mail.username=eenGebruikersNaam@gmail.com
spring.mail.password=paswoordVanDieGebruiker
```

❶  
❷  
❸  
❹  
❺

- (1) De netwerknnaam van de computer waarop de mailserver geïnstalleerd is.
- (2) Het TCP-IP nummer van de mailserver.
- (3) Het protocol van de mailserver.  
SMTP (Simple Mail Transfer Protocol) is het standaard protocol om mails te sturen. SMTPS beveiligt het SMTP protocol met authenticatie en versleuteling van de mails.
- (4) De gebruikersnaam onder wiens naam je mails stuurt.  
Gebruik hier niet je persoonlijke Google account, maar een nieuwe Google account. Zo raakt je account informatie niet bekend als je de project sources deelt met anderen (wat bijvoorbeeld gebeurt als je het project publiceert op je remote repository). Om vanuit code mails te versturen moet je de toegang activeren op <https://www.google.com/settings/security/lesssecureapps>.
- (5) Het paswoord van de gebruiker.

Spring maakt een `JavaMailSender` bean. Spring initialiseert de bean met de mail eigenschappen uit `application.properties`. Als je een mail wil versturen injecteer je de bean in je code. De bean helpt je een mail te versturen.

## 8.5 Entity

```
package be.vdab.mail.domain;
// enkele imports
@Entity
@Table(name = "leden")
public class Lid {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @NotBlank
    private String voornaam;
    @NotBlank
    private String familienaam;
    @NotNull
    @Email
    private String emailAdres;
    @DateTimeFormat(style = "S-")
    private LocalDate lidSinds;
    // constructor met parameters voornaam, familienaam, emailAdres
    // die lidSinds initialiseert op LocalDate.now()
    // protected default constructor
    // getters voor alle private variabelen
}
```

## 8.6 Repository

```
package be.vdab.mail.repositories;
// enkele imports
public interface LidRepository extends JpaRepository<Lid, Long> {
}
```

## 8.7 Bean die de mail stuurt

Maak een package be.vdab.beveiligd.exceptions. Maak daar een exception class:

```
package be.vdab.mail.exceptions;
public class KanMailNietZendenException extends RuntimeException {
    private static final long serialVersionUID = 1L;
    public KanMailNietZendenException(Exception oorspronkelijkeFout) {
        super(oorspronkelijkeFout);
    }
}
```

Maak een package be.vdab.mail.mailing. Maak daarin een interface LidMailing:

```
package be.vdab.mail.mailing;
import be.vdab.mail.domain.Lid;
public interface LidMailing {
    void stuurMailNaRegistratie(Lid lid, String ledenURL);
}
```

Implementeer de interface in de class DefaultLidMailing:

```
package be.vdab.mail.mailing;
// enkele imports
@Component ❶
class DefaultLidMailing implements LidMailing {
    private final JavaMailSender sender;
    DefaultLidMailing(JavaMailSender sender) { ❷
        this.sender = sender;
    }
    @Override
    public void stuurMailNaRegistratie(Lid lid, String ledenURL) { ❸
        try {
            var message = new SimpleMailMessage(); ❹
            message.setTo(lid.getEmailAdres()); ❺
            message.setSubject("Geregistreerd");
            message.setText("Je bent nu lid. Je nummer is:" + lid.getId());
            sender.send(message); ❻
        } catch (MailException ex) {
            throw new KanMailNietZendenException(ex);
        }
    }
}
```

(1) Je typt @Component voor de class. Spring maakt dan een bean van de class.

(2) Je injecteert de JavaMailSender bean. Die helpt je een mail te sturen.

(3) Je gebruikt straks de parameter ledenURL.

(4) SimpleMailMessage stelt een email zonder opmaak voor.

(5) Je vult de eigenschappen van de email in.

(6) Je verstuurt de email.

## 8.8 Service

```
package be.vdab.mail.services;
// enkele imports
public interface LidService {
    void registreer(Lid lid, String ledenURL);
    Optional<Lid> findById(long id);
}
```

Je implementeert de interface in de class DefaultLidService:

```
package be.vdab.mail.services;
// enkele imports
@Service
@Transactional
class DefaultLidService implements LidService {
    private final LidRepository lidRepository;
    private final LidMailing lidMailing;
    // constructor met parameter
    @Override
    public void registreer(Lid lid, String ledenURL) {
        lidRepository.save(lid);
        lidMailing.stuurMailNaRegistratie(lid, ledenURL);
    }
    @Override
    @Transactional(readOnly = true)
    public Optional<Lid> findById(long id) {
        return lidRepository.findById(id);
    }
}
```

## 8.9 Controller

```
package be.vdab.mail.controllers;
// enkele imports
@Controller
@RequestMapping("leden")
class LidController {
    private final Logger logger = LoggerFactory.getLogger(this.getClass());
    private final LidService lidService;
    // constructor met parameter
    @GetMapping("registratieform")
    public ModelAndView registratieForm() {
        return new ModelAndView("registratieform").addObject(new Lid("", "", ""));
    }
    @PostMapping
    public String registreer(@Valid Lid lid, Errors errors,
        RedirectAttributes redirect, HttpServletRequest request) {
        if (errors.hasErrors()) {
            return "registratieform";
        }
        try {
            lidService.registreer(lid, request.getRequestURL().toString());
            redirect.addAttribute("id", lid.getId());
            return "redirect:/leden/geregistreerd/{id}";
        } catch (KanMailNietZendenException ex) {
            logger.error("Kan mail niet zenden", ex);
            return "redirect:/leden/nietgeregistreerd";
        }
    }
    @GetMapping("geregistreerd/{id}")
    public String geregistreerd(@PathVariable long id) {
        return "geregistreerd";
    }
    @GetMapping("nietgeregistreerd")
    public String nietGeregistreerd() {
        return "nietgeregistreerd";
    }
}
```

## 8.10 CSS

Maak in de folder static een folder css. Maak daarin mail.css:

```
body {
    font-family: sans-serif;
}
input, button {
    display: block;
    margin-bottom: 1em;
    margin-top: 0.2em;
}
.fout {
    color: red;
}
```

## 8.11 ValidationMessages

Maak in de folder resources het bestand ValidationMessage.properties:

```
javax.validation.constraints.NotNull.message=Mag niet leeg zijn:
javax.validation.constraints.Email.message=Ongeldig e-mail adres:
javax.validation.constraints.NotBlank.message=Mag niet leeg zijn:
```

## 8.12 Thymeleaf pagina's

### 8.12.1 registratieform.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Registratie</title>
        <link rel="stylesheet" th:href="@{/css/mail.css}">
    </head>
    <body>
        <h1>Registratie</h1>
        <form th:object="${lid}" method="post" th:action="@{/leden}">
            <span th:errors="*{voornaam}" class="fout"></span>
            <input th:field="*{voornaam}" autofocus required placeholder="voornaam">
            <span th:errors="*{familienaam}" class="fout"></span>
            <input th:field="*{familienaam}" required placeholder="familienaam">
            <span th:errors="*{emailAdres}" class="fout"></span>
            <input th:field="*{emailAdres}" type="email" required
                placeholder="email adres">
            <button>Registreren</button>
        </form>
    </body>
</html>
```

### 8.12.2 geregistreerd.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Geregistreerd</title>
        <link rel="stylesheet" th:href="@{/css/mail.css}">
    </head>
    <body>
        <h1>Geregistreerd</h1>
        <div>Uw lidnummer is:<span th:text="${id}"></span>.</div>
        <div>We stuurden je een mail.</div>
    </body>
</html>
```



### 8.12.3 nietgeregistreerd.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Niet geregistreerd</title>
    <link rel="stylesheet" th:href="@{/css/mail.css}">
  </head>
  <body>
    <h1>Niet geregistreeerd</h1>
    <div class="fout">We konden je geen mail sturen.</div>
  </body>
</html>
```

Je kan de website proberen op <http://localhost:8080/leden/registratieform>.

### 8.13 Opmaak

De tekst in de mail heeft nu geen opmaak. Je kan opmaak geven met HTML elementen.

Wijzig in de class DefaultLidMailing de code in de method `stuurMailNaRegistratie`:

```
try {
    var message = sender.createMimeMessage();
    var helper = new MimeMessageHelper(message);
    helper.setTo(lid.getEmailAdres());
    helper.setSubject("Geregistreerd");
    helper.setText("<h1>Je bent nu lid.</h1>Je nummer is:" + lid.getId(), true);
    sender.send(message);
} catch (MailException | MessagingException ex) {
    throw new KanMailNietZendenException(ex);
}
```

- (1) `MimeMessage` stelt een email met HTML opmaak voor.
- (2) `MimeMessageHelper` helpt je de eigenschappen van de email in te stellen.
- (3) Je gebruikt HTML tags in de mail. Je moet dan de tweede parameter dan op `true` plaatsen.

Je kan dit proberen.

## 8.14 Hyperlink

Je kan een hyperlink opnemen in de mail. Die kan ook verwijzen naar je eigen website. Eenvoudig voorbeeld: als de gebruiker op de hyperlink klikt, ziet hij een pagina met zijn lid info.

### 8.14.1 Controller

Voeg een method toe aan LidController.

De method verwerkt de GET request als de gebruiker op de hyperlink in de mail klikt.

De URL van de request bij lid 8: <http://localhost:8080/leden/8>

```
@GetMapping("/{id}")
public ModelAndView info(@PathVariable long id) {
    var modelAndView = new ModelAndView("lidinfo");
    lidService.findById(id).ifPresent(lid -> modelAndView.addObject(lid));
    return modelAndView;
}
```

### 8.14.2 lidinfo.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Lid info</title>
        <link rel="stylesheet" th:href="@{/css/mail.css}">
    </head>
    <body>
        <dl th:if="${lid}" th:object="${lid}">
            <dt>Nnummer:</dt>
            <dd th:text="*{id}"></dd>
            <dt>Voornaam:</dt>
            <dd th:text="*{voornaam}"></dd>
            <dt>Familiennaam:</dt>
            <dd th:text="*{familiennaam}"></dd>
            <dt>Email adres:</dt>
            <dd th:text="*{emailAdres}"></dd>
            <dt>Lid sinds:</dt>
            <dd th:text="*{lidSinds}"></dd>
        </dl>
        <div th:if="not ${lid}" class="fout">Niet gevonden</div>
    </body>
</html>
```

### 8.14.3 DefaultLidMailing

Wijzig in de method geregistreerd de opdracht helper.setText(...);

```
var urlVanDeLidInfo = ledenURL + "/" + lid.getId();
var tekst = "<h1>Je bent nu lid.</h1>Je nummer is:" + lid.getId() + "." + ❶
    "Je ziet je info <a href='" + urlVanDeLidInfo + "'>hier</a>."; ❷
helper.setText(tekst, true);
```

- (1) ledenURI bevat de URI van de POST request op: <http://localhost:8080/leden>.  
Je plakt daarna een / en het lidnummer.  
Je bekomt de URI van de informatiepagina van het lid.
- (2) Je maakt een hyperlink. Hij verwijst naar de URL bij ❶.

Je kan dit proberen.



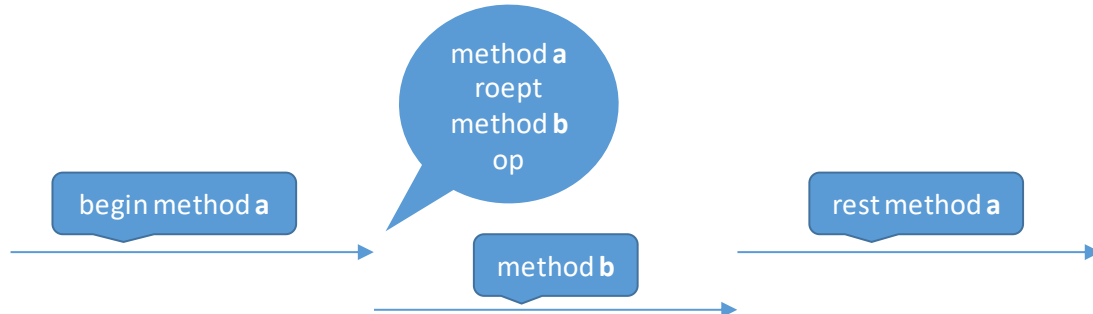
Commit de sources. Publiceer op je remote repository.

## 9 ASYNCHROON

Je werkt verder in het vorige project van de theorie.

Als je een bean method oproept, voert Java die method synchroon uit.

Dit betekent: als je in een method **a** een bean method **b** oproept, wacht Java tot de method **b** uitgevoerd is, vooraleer de code van method **a** verder uit te voeren:

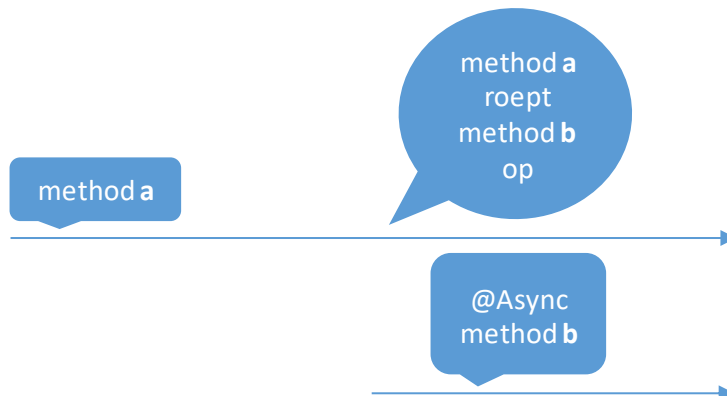


### 9.1 @Async

Als je voor de bean method **b** `@Async` typt, voert Spring de method **b** asynchroon uit.

Dit betekent: als je in een method **a** de `@Async` bean method **b** oproept, voert Spring de code van method **b** uit in een aparte thread.

Spring voert tegelijk de code van de method **a** verder uit in de thread die hoort bij **a**. Je verkrijgt dus multithreading.



Opmerking: dit werkt enkel als:

- de method **b** behoort tot een andere class dan de method **a**
- je beans maakt van die classes.

### 9.2 @EnableAsync

Typ `@EnableAsync` voor de class `MailApplication`. Enkel dan werkt `@Async`.

### 9.3 Voorbeeld

Je verstuurt de mail (trage operatie) in een aparte thread met `@Async`.

Typ `@Async` voor de `DefaultLidMailing` method `stuurMailNaRegistratie`.

Je kan dit proberen. Je merkt dat de use case sneller werkt.



Je commit de sources. Je publiceert op je remote repository.

## 10 SCHEDULING

Je werkt verder in het vorige project van de theorie.

Je applicatie voert soms taken uit die de gebruiker niet start, maar automatisch starten.

Voorbeelden:

- De applicatie verwijdt om het uur tijdelijke bestanden.
- De applicatie stuurt op de eerste dag van elke maand een mail met een rapport.

### 10.1 @Scheduled

Spring voert een bean method automatisch uit op terugkerende momenten als je voor die method `@Scheduled` typt.

`@Scheduled` heeft parameters waarmee je het terugkerend moment aangeeft:

- `@Scheduled(fixedDelay = 5000)`  
Spring voert de method uit. Nadat de method volledig uitgevoerd is, wacht Spring 5000 milliseconden. Spring voert daarna de method opnieuw uit ...
- `@Scheduled(fixedRate = 5000)`  
Spring voert de method om de 5000 milliseconden uit.
- `@Scheduled(cron = "0 0 20 1 * ?")`  
Spring voert de method uit om 20 uur, 1<sup>o</sup> dag van de maand van elke maand van elk jaar. De string is een cron expressie: een expressie die een tijdstip bepaalt.  
Zie <http://en.wikipedia.org/wiki/Cron>



De syntax van een cron expressie kan moeilijk zijn. Je vindt meer informatie op <https://www.baeldung.com/cron-expressions> (in het puntje 3).

### 10.2 @EnableScheduling

`@Scheduled` werkt enkel als je voor de class MailApplicatie `@EnableScheduling` typt.

### 10.3 Voorbeeld

Je stuurt op de eerste dag van elke maand een mail met het aantal leden naar de webmaster.

Voeg een regel toe aan `application.properties` met het email adres van de webmaster:

`emailAdresWebMaster=typHierEenEmailAdres`

Voeg een method declaratie toe aan `LidMailing`:

`void stuurMailMetAantalLeden(long aantalLeden);`

Implementeert de method in `DefaultLidMailing`.

Voeg een private variabele toe:

`private final String emailAdresWebMaster;`

IntelliJ geeft je een fout. IntelliJ stelt voor die op te lossen door de constructor uit te breiden.

Verfijn deze constructor zodat Spring de laatste parameter uit `application.properties` leest:

```
DefaultLidMailing(JavaMailSender sender,
    @Value("${emailAdresWebMaster}") String emailAdresWebMaster) {
    this.sender = sender;
    this.emailAdresWebMaster = emailAdresWebMaster;
}
```

Voeg de method uit de interface toe:

```
@Override
public void stuurMailMetAantalLeden(long aantalLeden) {
    try {
        var message = new SimpleMailMessage();
        message.setTo(emailAdresWebMaster);
        message.setSubject("Aantal Leden");
        message.setText(aantalLeden + " leden");
        sender.send(message);
    } catch (MailException ex) {
        throw new KanMailNietZendenException(ex);
    }
}
```

Voeg een method declaratie toe aan LidService:

```
void stuurMailMetAantalLeden();
```

Implementeer de method in DefaultLidService:

```
@Override
@Transactional(readOnly = true)
@Scheduled(fixedRate = 60_000)
// test = om de minuut
public void stuurMailMetAantalLeden() {
    lidMailing.stuurMailMetAantalLeden(lidRepository.count());
}
```

Je kan dit proberen.



Commit de sources. Publiceer op je remote repository.

## 11 REST

REST is een afkorting van Representational State Transfer.

Bij REST roept een applicatie via HTTP de diensten op van een andere applicatie.

- De applicatie die de diensten oproept heet de (REST) client.
- De applicatie die de diensten aanbiedt heet de (REST) service.

Je leesde in de cursus Spring fundamentals de dollar koers van de ECB website en de Fixer website. Je applicatie was REST client. De ECB website en de Fixer websites waren REST services.

REST is gebaseerd op enkel principes, die je hieronder leert kennen.

### 11.1 Entities identificeren met URI's

Elke entity én elke verzameling entities krijgt een URI.

In theorie is de URI vrij te kiezen. De URI /a1bd89 kan filiaal 1 voorstellen en /9zdb3 filiaal 2

De meeste web applicaties gebruiken echter *betekenisvolle* URI's.

Een betekenisvolle URI verduidelijkt *welke* data de URI aanbiedt.

Voorbeelden:

/filialen	Alle filialen
/werknemers	Alle werknemers
/filialen/3	Het filiaal met het nummer 3
/werknemers/6	De werknemer met het nummer 6
/filialen/3/werknemers	De werknemers van filiaal 3
/filialen?beginnaam=de	De filialen waarvan de naam begint met de
/filialen?vanafomzet=3000	De filialen met een omzet vanaf 3000
/filialen?sort=naam	Alle filialen gesorteerd op naam

### 11.2 HTTP methods

De client doet een request naar de URI die één of meerdere entities voorstelt.

De client duidt met de HTTP method (GET, POST, PUT of DELETE)

de *handeling* aan die hij wil uitvoeren op de entities

HTML gebruikt enkel GET en POST. Andere clients kunnen ook PUT en DELETE gebruiken.

#### 11.2.1 GET

De client doet een GET request om één of meerdere entities te *lezen*.

Hij krijgt een response terug met de data van die entity of entities.

Voorbeelden:

- een GET request naar /filialen om alle filialen te lezen.
- een GET request naar /filialen/3 om filiaal 3 te lezen.

#### 11.2.2 POST

De client doet een POST request om een entity *toe te voegen*.

De request body bevat de data van de toe te voegen entity.

Voorbeeld: een POST request naar /filialen om een filiaal toe te voegen.

#### 11.2.3 PUT

De client doet een PUT request om een entity te *wijzigen*.

De request body bevat de te wijzigen data van de entity.

Voorbeeld: een PUT request naar /filialen/3 om filiaal 3 te wijzigen.

#### 11.2.4 DELETE

De client doet een DELETE request om een entity te *verwijderen*.

Voorbeeld: een DELETE request naar /filialen/3 om filiaal 3 te verwijderen.

### 11.3 Formaat van de data

De data in de response kan in JSON uitgedrukt zijn, of in XML, ...

De client definieert in de request header Accept het formaat waarin hij de data in de response verwacht. Voorbeeld: een GET request naar `/filialen/3` met `Accept:application/json`. Dit betekent: geef de data van filiaal 3 aub in JSON formaat.

De response bevat het filiaal in JSON formaat:

```
{ "id": 3, "naam": "Gavdos", "gemeente": "Genk" "omzet": 3000 }
```

### 11.4 Response status code

De meest gebruikte response status codes:

Code	Betekenis	Omschrijving
200	OK	De request is correct verwerkt.
201	Created	De client deed een POST request om een entity toe te voegen. Die request is correct verwerkt.
400	Bad request	De client deed een POST request of een PUT request. De entity, die hij meestuurde in de request body, bevat verkeerde data (bvb. een werknemer met een negatieve wedde).
401	Unauthorized	De client deed een request op een beveiligde URI. De client heeft zich nog niet geauthenticeerd, of heeft een verkeerde gebruikersnaam/paswoord meegegeven.
403	Forbidden	De client deed een request op een beveiligde URI. De client heeft zich correct geauthenticeerd, maar heeft niet de rechten om de request te doen.
404	Not found	De client deed een request naar een URI die de service niet kent.
405	Method not allowed	De client deed een request op een URI met een HTTP method die de URI niet ondersteunt. Voorbeeld: een DELETE request naar <a href="#">/filialen</a>
406	Not acceptable	De service ondersteunt de waarde in request header Accept niet. Voorbeeld: de request header Accept bevat <code>application/xml</code> . De service kan echter enkel responses in JSON formaat produceren.
409	Conflict	De request probeert een entity in een verkeerde toestand te plaatsen. Voorbeeld: een DELETE request naar <a href="#">/filialen/3</a> . Filiaal 3 heeft nog werknemers. Je kan het daarom niet verwijderen.
415	Unsupported Media Type	De service ondersteunt het data formaat in de request body niet.
500	Internal server error	De service heeft interne (bvb. database) problemen om de request te verwerken.

### 11.5 HATEOAS

HATEOAS is een afkorting van Hypermedia As The Engine Of Application State.

HATEOAS betekent dat de response naast data ook links bevat naar andere URI's.

Voorbeeld: een GET request naar `/filialen/1`. De response bevat naast data over filiaal 1, ook URI's met gerelateerde data over dit filiaal:

```
"links": [
  { "werknemers": "http://www.voorbeeld.org/filialen/1/werknemers" },
  { "verkoopstatistiek": "http://www.voorbeeld.org/filialen/1/verkoopstatistiek" }
]
```

## 12 REST SERVICE

### 12.1 Database

Voer het script `rest.sql` uit.  
Het maakt een database `rest`.  
De database bevat één table:



### 12.2 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ `be.vdab` bij Group.
4. Typ `restservice` bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web, HATEOAS
SQL	Spring Data JPA, MySQL Driver

7. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 12.3 application.properties

```
spring.datasource.url=jdbc:mysql://localhost/rest
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED
```

### 12.4 Entity

```
package be.vdab.restservice.domain;
// enkele imports ...
@Entity
@Table(name = "filialen")
public class Filiaal {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String naam;
    private String gemeente;
    private BigDecimal omzet;
    // constructor met parameters: naam, gemeente, omzet
    // protected default constructor
    // getters
}
```

### 12.5 Repository

```
package be.vdab.restservice.repositories;
// enkele imports
public interface FiliaalRepository extends JpaRepository<Filiaal, Long> {
}
```



## 12.6 Exception

```
package be.vdab.restservice.exceptions;
public class FiliaalNietGevondenException extends RuntimeException {
    private static final long serialVersionUID = 1L;
}
```

## 12.7 Service

### 12.7.1 FiliaalService

```
package be.vdab.restservice.services;
// enkele imports
public interface FiliaalService {
    Optional<Filiaal> findById(long id);
    List<Filiaal> findAll();
    void create(Filiaal filiaal);
    void update(Filiaal filiaal);
    void delete(long id);
}
```

### 12.7.2 DefaultFiliaalService

```
package be.vdab.restservice.services;
// enkele imports
@Service
@Transactional
class DefaultFiliaalService implements FiliaalService {
    private final FiliaalRepository filiaalRepository;
    // constructor met parameter
    @Override
    @Transactional(readOnly = true)
    public Optional<Filiaal> findById(long id) {
        return filiaalRepository.findById(id);
    }
    @Override
    @Transactional(readOnly = true)
    public List<Filiaal> findAll() {
        return filiaalRepository.findAll();
    }
    @Override
    public void create(Filiaal filiaal) {
        filiaalRepository.save(filiaal);
    }
    @Override
    public void update(Filiaal filiaal) {
        filiaalRepository.save(filiaal);
    }
    @Override
    public void delete(long id) {
        try {
            filiaalRepository.deleteById(id);
        }
        catch (EmptyResultDataAccessException ex) {
            throw new FiliaalNietGevondenException();
        }
    }
}
```

## 12.8 Controller

```
package be.vdab.restservice.restcontrollers;
// enkele imports
@RestController
@RequestMapping("/filialen")
class FiliaalController {
    private final FiliaalService filiaalService;
    // constructor met parameter
}
```

❶

- (1) Je typt `@RestController` bij een controller die request verwerkt van clients die geen HTML als response verwachten, maar XML data of JSON data.

## 12.9 GET

Maak in de class `FiliaalController` een method Die verwerkt een GET request naar `/filialen/{filiaal}` (bvb. `/filialen/1`). De response bevat de data van het gevraagde filiaal.

```
@GetMapping("/{id}")
Filiaal get(@PathVariable long id) {
    return filiaalService.findById(id).get();
}
```



❶

- (1) De `@GetMapping` method behoort tot een `@RestController` class. Spring converteert dan de returnwaarde van de method naar XML of JSON. Spring vult hiermee de response body.
- Bij een request header `Accept` met `application/xml`, vult Spring de response met XML.
  - Bij een request header `Accept` met `application/json`, vult Spring de response met JSON. Spring doet dit ook als in de request de `Accept` header ontbreekt.

## 12.10 Test

Je start de applicatie.

Je test de applicatie door requests te sturen vanuit IntelliJ Ultimate.

1. Klik op  voor de method `get`.
2. Dubbelklik `http://localhost:8080/filialen/{id}`
3. Vervang `{id}` door `1`.
4. Klik op  in de marge. Kies `Run localhost:8080`. Je verstuurt zo de request.
5. Je ziet de response.
  - a. Je ziet `Response code: 200 ...`
  - b. Je ziet daar boven de response body: JSON data over filiaal 1.
  - c. Je ziet daar boven de response headers. Vb. `Content-type` bevat `application/json`.

Als je geen IntelliJ Ultimate hebt, gebruik je de tool Postman (<https://www.getpostman.com>).



## 12.11 JAXB

Je zal de data vragen in XML formaat. Spring zet dan een `Filiaal` object om naar XML.

Spring doet dit met de library JAXB. JAXB vereist dat je annotations typt voor de class `Filiaal`.

1. Typ `@XmlRootElement`. Je typt dit bij een class die een XML root element voorstelt. De naam van het element is de naam van de class, met de eerste letter in kleine letters: `<filiaal>`.
2. Typ `@XmlAccessorType(XmlAccessType.FIELD)`. JAXB vereist standaard getters én setters. Met deze annotation heeft JAXB geen getters of setters nodig.

Je test met  terug een GET request naar `http://localhost:8080/filialen`.

1. Geef aan dat je de response in XML formaat wil met de request header `Accept`:  
Voor je op  klikt typt je onder GET `http://localhost:8080/filialen/1`  
`Accept: application/xml`.
2. Verstuur de request met . Je ziet de response nu in XML formaat.

## 12.12 404 (Not Found)

Als de client een request doet naar een niet-bestaand filiaal, definieert HTTP dat je een response stuurt met status code 404 ( Not Found ).

Wijzig in de FiliaalController de code in de method get:

```
return filiaalService.findById(id)
    .orElseThrow(FiliaalNietGevondenException::new);
```

❶

(1) Je werpt een FiliaalNietGevondenException als je het filiaal niet vindt in de database.

Voeg een method toe:


```
@ExceptionHandler(FiliaalNietGevondenException.class)
@ResponseStatus(HttpStatus.NOT_FOUND)
void filiaalNietGevonden() {
}
```

❶  
❷

(1) Je geeft aan dat de method die op deze annotation volgt (filiaalNietGevonden) een response naar de browser stuurt als een FiliaalNietGevondenException optreedt.

(2) Je geeft de status code van die response aan.

Als een @GetMapping method een FiliaalNietGevondenException werpt, roept Spring filiaalNietGevonden op. Spring maakt zo een response met de status code 404 (Not Found).

Doe met  een request naar `http://localhost:8080/filiaalen/666`.

Je krijgt een response met een status code 404 (Not Found).


## 12.13 DELETE

Voeg een method toe aan FiliaalController:

```
@DeleteMapping("{id}")
void delete(@PathVariable long id) {
    filiaalService.delete(id);
}
```

❶

(1) Het returntype van de method is void . Spring maakt dan een response met een lege body.

Doe met  een DELETE request naar `http://localhost:8080/filiaalen/1`.

## 12.14 POST

De client stuurt een POST request naar /filialen om een nieuw filiaal toe te voegen


De request body bevat data (in XML of JSON formaat) over het toe te voegen filiaal.



Voeg een method toe aan FiliaalController:

```
@PostMapping
void post(@RequestBody Filiaal filiaal) {
    filiaalService.create(filiaal);
}
```

❶

(1) Spring vertaalt met @RequestBody de data in de request body naar een Filiaal object en geeft dit door in de parameter filiaal.

Test dit met .

1. Geef aan dat je de request body JSON data bevat met de request header Content-Type: Voor je op  klikt typ je onder POST `http://localhost:8080/filiaalen`. Content-Type: application/json.
2. Typ een lege regel (om de headers af te sluiten). Typ daar onder de request body: `{"naam": "Nieuw filiaal", "gemeente": "Brussel", "omzet": 7000}`
3. Verstuur de request met . Je krijgt een response met de status code 200 (OK).

## 12.15 Validatie

De data die de client doorgeeft kan verkeerd zijn. Voorbeeld: een negatieve omzet.  
Als de data verkeerd is, voeg je geen filiaal toe, maar stuurt je een response met fouten terug.

Je voegt de validatie dependency toe aan pom.xml:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Klik rechts op het verticale tabblad Maven. Klik daar op .

Je voegt validation annotations toe aan de class Filiaal:

- @NotBlank voor de variabelen naam en gemeente.
- @NotNull en @PositiveOrZero voor de variabele omzet.

Wijzig in de class FiliaalController de method post:

```
@PostMapping
void post(@RequestBody @Valid Filiaal filiaal) {
    filiaalService.create(filiaal);
}
```

❶

- (1) Spring vertaalt de data in de request body naar een Filiaal object.  
Spring valideert met @Valid dit Filiaal object.  
Als er validatiefouten zijn, voert Spring de method post niet uit,  
maar werpt een MethodArgumentNotValidException.

Voeg een method toe:

```
@ExceptionHandler(MethodArgumentNotValidException.class)
@ResponseStatus(HttpStatus.BAD_REQUEST)
Map<String,String> verkeerdeData(MethodArgumentNotValidException ex) {
    return ex.getBindingResult().getFieldErrors().stream()
        .collect(Collectors.toMap(FieldError::getField,
            FieldError::getDefaultMessage));
}
```

❶

❷


❸

❹

❺

❻


- (1) Je geeft aan dat de method bij ❹ een response naar de browser stuurt als een MethodArgumentNotValidException optreedt.
- (2) Je geeft de status code van die response aan.
- (3) Je baseert de response op een Map. De key van elke entry in de Map zal de naam van het foute attribuut zijn. De value van de entry is de omschrijving van de fout. Spring plaatst de fouten in de request data in de de MethodArgumentNotValidException parameter.
- (4) getBindingResult().getFieldErrors() geeft een verzameling FieldError objecten: de Filiaal attributen met validatiefouten.
- (5) Je maakt van deze verzameling een Map. De key is de naam van het foute attribuut.
- (6) De value is de omschrijving van de fout.

Doe met  een POST request naar http://localhost:8080/filialen met de request body:  
{ "naam": "Nieuw filiaal", "gemeente": "", "omzet": -7000 }

## 12.16 PUT

Voeg een method toe aan FiliaalController:

```
@PutMapping("{id}")
void put(@RequestBody @Valid Filiaal filiaal) {
    filiaalService.update(filiaal);
}
```

Doe met  met een PUT request naar http://localhost:8080/filialen/4 met de request header Content-Type: application/json en de volgende request body:  
{ "id": 4, "naam": "Nieuwe naam", "gemeente": "Nieuwe gemeente", "omzet": 4321 }

## 12.17 HATEOAS

Bij HATEOAS bevat de response, naast data, ook links naar interessante URL's.

Je implementeert HATEOAS met de library Spring HATEOAS. Die heeft een beperking.

Hij ondersteunt enkel JSON, geen XML.

### 12.17.1 Configuratie per controller

Je moet de URL aangeven die een type entity hoort (/filialen hoort bij Filiaal entities).

Je typt `@ExposesResourceFor(Filiaal.class)` voor de class `FiliaalController`.

Spring ziet in `@RequestMapping` bij de class dat de URL /filialen hoort bij Filiaal entities.

### 12.17.2 EntityLinks

Spring maakt standaard een EntityLinks bean. Je maakt met die bean hyperlinks.

Je injecteert de bean in de class `FiliaalController`.

Voeg een private variabele toe:

```
private final EntityLinks entityLinks;
```

IntelliJ stelt voor de constructor uit te breiden:

```
FiliaalController(FiliaalService filiaalService, EntityLinks entityLinks) {
    this.filiaalService = filiaalService;
    this.entityLinks = entityLinks;
}
```

IntelliJ geeft een verkeerde fout op de parameter `entityLinks`. Je mag de fout negeren.

Je maakt met EntityLinks Link objecten. Die stellen JSON data met hyperlinks voor.

- `entityLinks.linkToCollectionResource(Filiaal.class)`  
maakt een Link object met JSON data die de URL van *alle* filialen bevat:  
`"self": {"href": "http://localhost:8080/filialen"}`.
- `entityLinks.linkToItemResource(Filiaal.class, 1)`  
maakt een Link object met JSON data die de URL van filiaal **1** bevat:  
`"self": {"href": "http://localhost:8080/filialen/1"}`

### 12.17.3 Location

REST bepaalt dat je na het toevoegen van een entity een response stuurt met

- status code 201 (Created)
- een Location header met de URL van de toegevoegde entity

Wijzig de method `post`:

```
@PostMapping
@ResponseStatus(HttpStatus.CREATED)
HttpHeaders create(@RequestBody @Valid Filiaal filiaal) {
    filiaalService.create(filiaal);
    var link = entityLinks.linkToItemResource(Filiaal.class, filiaal.getId());
    var headers = new HttpHeaders();
    headers.setLocation(link.toUri());
    return headers;
}
```

- (1) Spring vult de response status code met de waarde uit `@ResponseStatus`. De constante `CREATED` bevat de waarde 201 (Created).
- (2) Je gebruikt `HttpHeaders` als return type. Je kan daarmee response headers opbouwen.
- (3) Je maakt een Link object dat verwijst naar de URI van het nieuwe filiaal.
- (4) Je vult de Location header van de response met de URI van het nieuwe filiaal.

Je test dit. Voeg een filiaal toe met een POST request.

Je kijkt dan in de response header Location.

### 12.17.4 Link elementen in de response body

Bij een GET request naar één filiaal zal de response naast de data van het filiaal

- een "self" link naar het filiaal zelf bevatten. Voorbeeld:  
`"self": {"href": "http://localhost:8080/filialen/1"}`
- een link naar de werknemers van dat filiaal bevatten. Voorbeeld:  
`"werknemers": {"href": "http://localhost:8080/filialen/1/werknemers"}`

Je stelt zo'n response voor met het type `EntityModel<Filiaal>`.

Dit plaatst de data van een filiaal in de response en laat toe ook hyperlinks toe te voegen.

Wijzig de method `get`:

```
@GetMapping("/{id}")
EntityModel<Filiaal> get(@PathVariable long id) {
    return filiaalService.findById(id).map(filiaal -> EntityModel.of(
        filiaal,
        entityLinks.linkToItemResource(Filiaal.class, filiaal.getId()),
        entityLinks.linkForItemResource(Filiaal.class, filiaal.getId())
            .slash("werknemers").withRel("werknemers")))
    .orElseThrow(FiliaalNietGevondenException::new);
}
```

- (1) Je maakt een `EntityModel`. Spring maakt hiermee de response.
- (2) Je voegt hieraan het filiaal toe. De response bevat dan ook dit filiaal.
- (3) Je voegt ook de URI van het filiaal toe (voorbeeld: `http://localhost:8080/filialen/1`)
- (4) Je voegt ook de URI van de werknemers van het filiaal toe (voorbeeld: `http://localhost:8080/filialen/1/werknemers`).  
`slash("werknemers")` maakt `/weknemers` in de URI.  
`withRel("werknemers")` maakt `"werknemers":` in de response.

Je test dit. Doe een GET request naar `http://localhost:8080/filialen/4`.

### 12.17.5 Response met een verzameling entities

Als de client een request doet naar `/filialen`, zal hij volgende response krijgen:

```
[
  {
    "id": 1, "naam": "Andros",
    "links": [{"rel": "self", "href": "http://localhost:8080/filialen/1"}]},
  {
    "id": 2, "naam": "Delos",
    "links": [{"rel": "self", "href": "http://localhost:8080/filialen/2"}]}
]
```

- (1) De response bevat per filiaal enkel de belangrijkste data: id en naam.  
 Het aantal bytes in de response is zo beperkt.  
 De communicatie tussen de service en de client is zo performant.  
 Als de client detail van dat filiaal wil, doet hij een GET request naar de URL bij ②.
- (2) De response bevat per filiaal de URI van dat filiaal.

Maak een class die enkel de id en de naam van een filiaal voorstelt. Die class hoort bij ①.

Je gebruikt deze class enkel in `FiliaalController`. Maak de class daarom als een nested class: tik de class voor de sluit accolade van `FiliaalController`:

```
private static class FiliaalIdNaam {
    private final long id;
    private final String naam;
    public FiliaalIdNaam(Filiaal filiaal) {
        id = filiaal.getId();
        naam = filiaal.getNaam();
    }
    // getters
}
```

Wijzig in FiliaalController de method findAll:

```
@GetMapping
CollectionModel<EntityModel<FiliaalIdNaam>> findAll() {
    return CollectionModel.of(
        filiaalService.findAll().stream()
            .map(filiaal ->
                EntityModel.of(new FiliaalIdNaam(filiaal),
                    entityLinks.linkToItemResource(Filiaal.class, filiaal.getId())))
            .collect(Collectors.toList()),
        entityLinks.linkToCollectionResource(Filiaal.class));
}
```

- (1) CollectionModel stelt een response voor met een *verzameling* objecten en hyperlink(s). Elk object is hier een EntityModel: data uit FiliaalIdNaam en een hyperlink.
- (2) Je maakt een CollectionModel.
- (3) Je leest alle filialen in de database.
- (4) Je maakt een EntityModel op basis van elk filiaal.
- (5) De data in dit EntityModel is een FiliaalIdNaam.
- (6) Je voegt de URL van het filiaal toe aan het EntityModel.
- (7) Je verzamelt de EntityModel objecten in een List. Je geeft die aan je CollectionModel.
- (8) Je voegt de URL van alle filialen toe aan je CollectionModel.

Je test dit. Je doet een GET request naar <http://localhost:8080/filialen>



Commit de sources. Publiceer op je remote repository.



Muziek


## 13 DOCUMENTATIE

Je maakt online documentatie van je REST service. Andere programmeurs kunnen die raadplegen.

### 13.1 pom.xml

Je voegt een dependency toe:

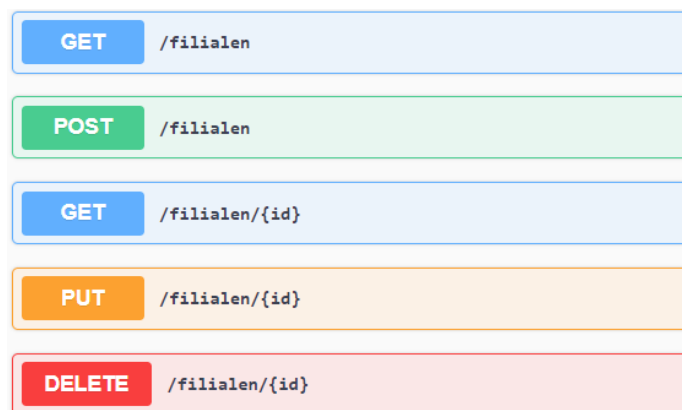
```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.4.1</version>
</dependency>
```

Klik rechts op het verticale tabblad Maven. Klik daar op .



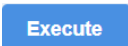
### 13.2 Resultaat

Start de applicatie. Surf naar <http://localhost:8080/swagger-ui.html>.

Je ziet een overzicht van de requests die je applicatie verwerkt:



Je kan een request testen:

1. Je kiest  `/filialen/{id}`.
2. Je kiest .
3. Je tikt een filiaal id bij id.
4. Je kiest . Je ziet daar onder de response.

### 13.3 Omschrijving

Je kan in het overzicht van de request elke request een omschrijving geven.

Je tikt in FilaalController volgende annotations voor volgende methods:

Method	Annotation
get	<code>@Operation(summary = "Een filiaal zoeken op id")</code>
post	<code>@Operation(summary = "Een filiaal toevoegen")</code>
findAll	<code>@Operation(summary = "Alle filialen zoeken")</code>
put	<code>@Operation(summary = "Een filiaal wijzigen")</code>
delete	<code>@Operation(summary = "Een filiaal verwijderen")</code>

### 13.4 Titel

Je wijzigt de titel v.h. document naar Filialen, met een extra method in RestserviceApplication:

```
@Bean
OpenAPI openAPI() {
  return new OpenAPI().info(new Info().title("Filialen"));
}
```

Je kan dit testen.



## 14 REST SERVICE INTEGRATION TEST

Je stuurt in een integration test HTTP requests naar je REST service.

Je test zo de correcte werking van je REST service.

Maak in de folder src/test/ een folder resources. Maak daarin spring.properties:

```
spring.test.constructor.autowire.mode=all
```

Maak daar ook insertFiliaal.sql:

```
insert into filialen(naam, gemeente, omzet) values ('test', 'test', 1000);
```

Maak in de folder src/test/java een package be.vdab.restservice.restcontrollers.

Maak daarin een class FiliaalControllerTest:

```
package be.vdab.restservice.restcontrollers;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
// enkele andere imports
@SpringBootTest
@AutoConfigureMockMvc
@Sql("/insertFiliaal.sql")
class FiliaalControllerTest extends AbstractTransactionalJUnit4SpringContextTests {
    private final MockMvc mvc;
    //constructor met parameter
    private long idVanTestFiliaal() {
        return super.jdbcTemplate.queryForObject(
            "select id from filialen where naam='test'", Long.class);
    }
    @Test
    void onbestaandFiliaalLezen() throws Exception {
        mvc.perform(get("/filialen/{id}", -1))
            .andExpect(status().isNotFound());
    }
    @Test
    void filiaallezen() throws Exception {
        var id = idVanTestFiliaal();
        mvc.perform(get("/filialen/{id}", id))
            .andExpect(status().isOk())
            .andExpect(jsonPath("id").value(id));
    }
}
```

- (1) @SpringBootTest maakt alle beans (Controllers, Services, Repositories, ...).
- (2) @AutoConfigureMockMvc maakt een object van de class MockMvc.  
Je stuurt met zo'n object HTTP requests vanuit je test.
- (3) Je injecteert dit MockMvc object.
- (4) Je stuurt een GET request naar de URI van een niet-bestaand filiaal.  
De method get is een static method van de class MockMvcRequestBuilders.  
Je importeerde de method boven in de source met static import.  
De eerste parameter is een URI template.  
De tweede parameter is de waarde voor de path variabele in de URI template.
- (5) Je controleert of de response status code Not Found (404) is.
- (6) Je stuurt een GET request naar de URL van het filiaal dat je aan de database toevoegde.
- (7) Je controleert of de response status code OK (200) is.
- (8) Je geeft aan jsonPath een JSONPath expressie mee.  
Je zoekt met zo'n expressie data in JSON data (zoals je met XPATH data zoekt in XML data).  
De waarde van het attribuut id in de JSON data moet gelijk zijn aan  
de id van het filiaal dat je aan de database toevoegde.

Voer de test uit.



Commit de sources. Publiceer op je remote repository.



Muziek test

## 15 REST CLIENT

### 15.1 WebClient

Je doet requests naar een REST service met de Spring class WebClient.

Je bestudeert onderstaande code. Je moet ze niet typen.

#### 15.1.1 GET

Je stuurt met de volgend code een GET request naar `http://localhost:8080/filialen/1`:

```
try {
    var filiaal = client.get()
        .uri("http://localhost:8080/filialen/1")
        .retrieve()
        .bodyToMono(Filiaal.class)
        .block();
    // hier komt code om het gelezen filiaal (in de var. filiaal) te verwerken.
} catch (WebClientResponseException.NotFound ex) {
    // hier komt code als response status code 404 (Not Found) is.
}
```

- (1) Je geeft aan dat je een GET request wil versturen.
- (2) Je geeft de URI aan waarnaar je de request wil versturen.
- (3) Je verstuurt de request.
- (4) Je geeft aan dat je de response body wil omzetten naar een Filiaal object.
- (5) Je doet de omzetting die je bij 4 aangaf. Je krijgt een Filiaal object.

#### 15.1.2 POST

Je stuurt met de volgend code een POST request naar `http://localhost:8080/filialen`:

```
try {
    var uri = client.post()
        .uri("http://localhost:8080/filialen")
        .contentType(MediaType.APPLICATION_JSON)
        .body(BodyInserters.fromValue(filiaal))
        .retrieve()
        .toBodilessEntity()
        .block()
        .getHeaders()
        .getLocation();
} catch (WebClientResponseException.NotFound ex) {
    // hier komt code als response status code 404 (Not Found) is.
}
} catch (WebClientResponseException.BadRequest ex) {
    // hier komt code als response status code 400 (Bad request) is.
    // Je krijgt zo'n response als het filiaal verkeerde data bevat.
}
```

- (1) Je geeft aan dat je een POST request wil versturen.
- (2) Je geeft de URI aan waarnaar je de request wil versturen.
- (3) Je plaatst de request header content-type op application/json.
- (4) Je plaatst het object in de variabele filiaal in de request body.
- (5) Je verstuurt de request.
- (6) Je geeft aan dat je toegang wil tot de response, maar de inhoud van de response body niet wil lezen, omdat die leeg is.
- (7) Je krijgt toegang tot de response die je bij 6 vroeg.
- (8) Je vraagt alle response headers.
- (9) Je vraagt daarvan de location header. Je krijgt die als een URI object.

### 15.1.3 PUT

Je stuurt met de volgend code een PUT request naar `http://localhost:8080/filialen/1`. De code is gelijkaardig aan de code bij een POST request:

```
try {
    var uri = client.put()
        .uri("http://localhost:8080/filialen/1")
        .contentType(MediaType.APPLICATION_JSON)
        .body(BodyInserters.fromValue(filial))
        .retrieve()
        .toBodilessEntity()
        .block()
} catch (WebClientResponseException.NotFound ex) {
    // hier komt code als response status code 404 (Not Found) is.
}
} catch (WebClientResponseException.BadRequest ex) {
    // hier komt code als response status code 400 (Bad request) is.
    // Je krijgt zo'n response als het filiaal verkeerde data bevat.
}
```

### 15.1.4 DELETE

Je stuurt met de volgend code een DELETE request naar `http://localhost:8080/filialen/1`. De code is gelijkaardig aan de vorige voorbeelden:

```
try {
    client.delete()
        .uri("http://localhost:8080/filialen/1")
        .retrieve()
        .toBodilessEntity()
        .block()
    // hier komt code als de response status code 200 (OK) is.
} catch (WebClientResponseException.NotFound ex) {
    // hier komt code als response status code 404 (Not Found) is.
}
```

## 15.2 Voorbeeld

Je zal een GET request sturen naar <https://reqres.in/api/users/1>. De response:

```
{
  "data": {
    "id": 1,
    "email": "george.bluth@reqres.in",
    "first_name": "George",
    "last_name": "Bluth",
    "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/calebogden/128.jpg"
  }
}
```

### 15.2.1 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ `be.vdab` bij Group.
4. Typ `restclient` bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web, Spring Reactive Web
Template Engines	Thymeleaf

7. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

Maak in de folder test een folder resources. Maak daarin spring.properties:

`spring.test.constructor.autowire.mode=all`

### 15.2.2 Structuur van de JSON data

Je stelt de structuur van de JSON data voor als Java classes.

Je ziet in de JSON data een object met een attribuut data.

Dit attribuut is terug een object met de attributen id, email, ...

Maak eerst de class die het binnenste object (data) voorstelt.

Je gebruikt straks een object van die class als je het buitenste object voorstelt.

```
package be.vdab.restclient.dto;
import com.fasterxml.jackson.annotation.JsonProperty;
public class Data {
    private long id;
    @JsonProperty("first_name")
    private String firstName;
    @JsonProperty("last_name")
    private String lastName;
    // getters
}
```

❶  
❷  
❸  
❹

- (1) Deze variabele stelt het attribuut id voor in het binnenste object in de JSON data.
- (2) De variabele bij ❸ stelt het attribuut first\_name voor.  
De naamgeving conventie van Java verbiedt \_ tekens in de naam van de variabele.  
@JsonProperty("first\_name") koppelt de variabele bij ❸ aan het attribuut first\_name.
- (3) Deze variabele stelt dus het attribuut first\_name voor.
- (4) Je hebt het attribuut avatar in de JSON data (zoals) in deze applicatie niet nodig.  
Je moet dan geen bijbehorende private variabele maken.

Maak de class die het buitenste object voorstelt in de JSON data:

```
package be.vdab.restclient.dto;
public class User {
    private Data data;
    // getter
}
```

❶

- (1) Deze variabele stelt het attribuut data voor.

Typ de URL van de rest service die je wil aanspreken in application.properties:

`reqres.eenUser=https://reqres.in/api/users/{id}`

❷

- (1) Je gebruikt deze URL bijvoorbeeld als je een GET request stuurt om één user op te halen.  
Je stelt het nummer van de user voor met de parameter id, tussen accolades.

## 15.3 RestClients layer

Je Rest client code is een aparte layer van je applicatie.

Je maakt voor die layer een package: `be.vdab.restclient.restclients`.

Maak daarin een interface. Die beschrijft de werking van je rest client.

```
package be.vdab.restclient.restclients;
// enkele imports
public interface ReqResClient {
    Optional<User> findById(long id);
}
```

❶  
❷

- (1) De interface beschrijft methods om de website Reqres aan te spreken.  
Een goede naam is dus ReqResClient.
- (2) Deze method haalt een user op van de website Reqres.  
De parameter is de id van de op te halen user. Je krijgt een Optional terug.  
Die is leeg als de user niet gevonden werd, of bevat de user als die gevonden werd.

Implementeer de interface in de class DefaultReqResClient:

```
package be.vdab.restclient.restclients;
// enkele imports
@Component
class DefaultReqResClient implements ReqResClient {
    private final WebClient client;
    private final String eenUserURI;
    DefaultReqResClient(WebClient.Builder builder,
        @Value("${reqres.eenUser}") String eenUserURI) {
        client = builder.build();
        this.eenUserURI = eenUserURI;
    }
    @Override
    public Optional<User> findById(long id) {
        try {
            return Optional.of(client.get()
                .uri(eenUserURI, uriBuilder->uriBuilder.build(id))
                .retrieve().bodyToMono(User.class).block());
        } catch (WebClientResponseException.NotFound ex) {
            return Optional.empty();
        }
    }
}
```

- (1) Je maakt van de class een Spring bean.
- (2) Je injecteert een WebClient.Builder. Spring maakt altijd een WebClient.Builder bean. Je maakt daarmee een WebClient bij 4. Dit is een vb. van het builder design pattern.
- (3) Je injecteert de URI uit application.properties.
- (4) Je maakt het WebClient object.
- (5) Je doet een GET request naar de reqres website.
- (6) De eerste parameter is de URI uit application.properties. De tweede parameter is een lambda. Hij heeft een parameter waarmee je URI bewerkt. Je roept de build method op. Je geeft een waarde mee voor de parameter id in de URL: de id van de huidige method.
- (7) Als de response een status code 404 (Not Found) heeft, geef je een lege Optional terug.

Controleer de juiste werking van de rest client met een test:

```
package be.vdab.restclient.restclients;
// enkele imports
@SpringBootTest
class DefaultReqRestClientTest {
    private final DefaultReqResClient client;
    // constructor met parameter
    @Test
    void findBestaandeUser() {
        assertThat(client.findById(1).get().getData().getId()).isOne();
    }
    @Test
    void findOnbestaandeUser() {
        assertThat(client.findById(-1)).isEmpty();
    }
}
```

- (1) Je haalt de user met de id 1 op. De user moet als id 1 hebben.
- (2) Je haalt een user met een onbestaande id op. Je moet een lege Optional krijgen.

Voer de test uit.

## 15.4 RestClients layer gebruiken

Je maakt een minimaal voorbeeld waarin je de RestClients layer gebruikt: je toont user 1.

### 15.4.1 Controller:

```
package be.vdab.restclient.controllers;
// enkele imports
@Controller
@RequestMapping("/users")
class UserController {
    private final ReqResClient client;
    // constructor met parameter
    @GetMapping("1")
    public ModelAndView getUser() {
        var modelAndView = new ModelAndView("user");
        client.findById(1).ifPresent(user -> modelAndView.addObject(user));
        return modelAndView;
    }
}
```

### 15.4.2 user.html

```
<!DOCTYPE html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Gebruiker</title>
    </head>
    <body>
        <p th:if="not ${user}">Niet gevonden</p>
        <dl th:if="${user}" th:object="${user.data}">
            <dt>Nummer</dt>
            <dd th:text="*{id}"></dd>
            <dt>Voornaam</dt>
            <dd th:text="*{firstName}"></dd>
            <dt>Voornaam</dt>
            <dd th:text="*{lastName}"></dd>
        </dl>
    </body>
</html>
```

Start de applicatie. Surf naar <http://localhost:8080/users/1>.



Commit de sources. Publiceer op je remote repository.



Je vindt een overzicht van interessante rest services op <http://www.programmableweb.com/apis/directory>.



Temperatuur

## 16 REST CLIENT MET JAVASCRIPT

### 16.1 Frontend – backend

#### 16.1.1 Één applicatie

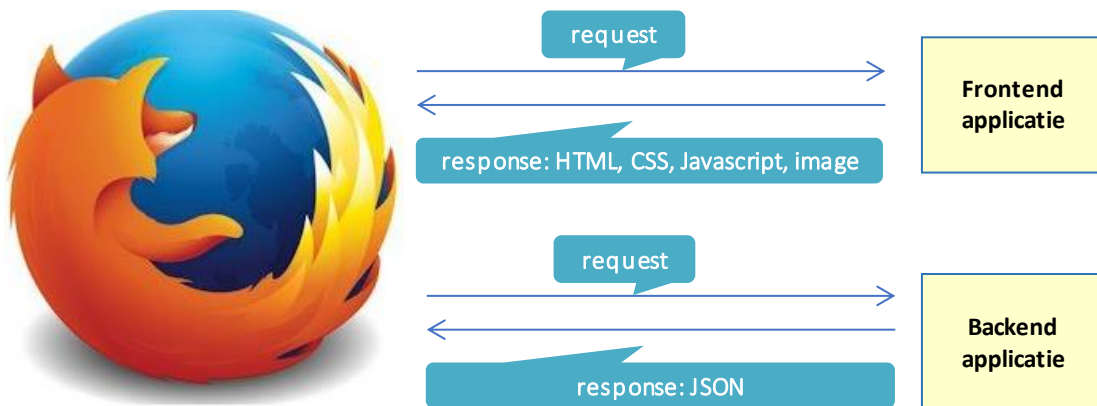
Je applicatie is tot nu *één* applicatie:



Je genereert de HTML in de response met Thymeleaf.

#### 16.1.2 Twee applicaties

Men maakt regelmatig *twee* applicaties.



De gebruiker surft naar de frontend applicatie. Dit is geen Spring applicatie, maar een statische website, met HTML bestanden, CSS bestanden, Javascript bestanden en afbeeldingen.

Je haalt in de JavaScript code JSON data op uit de backend applicatie. Je toont die data in de HTML. De backend applicatie is een Spring applicatie die de rol speelt van REST service.

Één team kan de frontend applicatie maken en een ander team de backend applicatie.

- Het frontend team is gespecialiseerd in HTML, CSS en JavaScript.
- Het backend team is gespecialiseerd in Java, JPA, Spring, database toegang, rest services, ...

Je maakt hier een frontend applicatie.

Je gebruikt het bestaande project `restservice` als backend applicatie. Open eerst dit project.

### 16.2 CORS

Standaard spreekt JavaScript een REST service aan met een “same-origin” policy.

Javascript code van website A kan enkel de REST services van de *eigen* website A aanspreken.

Javascript code van website A kan de REST services van *andere* websites B,C,... *niet* aanspreken.

De Javascript code in de frontend applicatie kan dus de backend applicatie niet kan aanspreken.

Met CORS (Cross-Origin Resource Sharing) kan Javascript code van website A wél REST services van andere websites aanspreken. Website B moet toestemming geven dat JavaScript code van andere websites zijn services aanspreken. Website B en de JavaScript code van die website onderhandelen over de toestemming met extra request headers en responses headers.



Dank zij Spring moet je de detail van deze headers niet kennen.

Het volstaat dat je in je REST controller `@CrossOrigin` gebruikt:

- Als je in `FiliaalController` voor de method `get @CrossOrigin` typt, kan Javascript code van andere websites een GET request doen naar de URL die hoort bij die method. De Javascript code kan de andere diensten van `FiliaalController` niet oproepen.
- Als je `@CrossOrigin` typt voor de class `FiliaalController`, kan JavaScript code van andere websites alle requests doen die `FiliaalController` behandelt.

Typ voor `FiliaalController`

`@CrossOrigin(exposedHeaders = "Location")` ❶

- (1) Bij een CORS request bevat de response standaard slechts een beperkt aantal headers. De Location header ontbreekt. Je hebt in het voorbeeld straks deze header nodig. Met `exposedHeaders = "Location"` plaatst Spring deze header toch in de response.

Je start de applicatie. Je laat ze in uitvoering.

## 16.3 Frontend project

Je maakt in IntelliJ het project voor de frontend applicatie:

1. Kies in het menu File de opdracht New, Project.
2. Kies links JavaScript.
3. Kies rechts JavaScript.
4. Kies Next.
5. Typ frontend bij Project name.
6. Kies Finish.
7. Kies New Window.

IntelliJ opent het project in een nieuw venster en laat het project `restservice` open staan.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 16.3.1 CSS

Je voegt `frontend.css` toe:

1. Klik in het venster Project met de rechtermuisknop op het project frontend.
2. Kies New, Stylesheet.
3. Typ frontend. Druk Enter.

Typ volgende code:

```
body {
    font-family: sans-serif;
}
dd,dt {
    margin-left: 0;
}
dt {
    margin-top: 0.5em;
}
dd {
    font-weight: bold;
}
.fout {
    color:red;
}
```

### 16.3.2 HTML

Je voegt index.html toe:

1. Klik in het venster Project met de rechtermuisknop op het project frontend.
2. Kies New, HTML File.
3. Typ index. Druk Enter.

Wijzig de code:

```
<!doctype html>
<html lang="nl">
  <head>
    <script src="index.js" defer></script>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="frontend.css">
    <title>Filialen</title>
  </head>
  <body>
    <h1>Filialen</h1>
    <ul id="filialen">
    </ul>
    <div id="technischeFout" hidden class="fout">
      Technische fout, contacteer helpdesk</div>
  </body>
</html>
```

### 16.3.3 JavaScript

Voeg index.js toe:

4. Klik in het venster Project met de rechtermuisknop op het project frontend.
5. Kies New, JavaScript File.
6. Typ index. Druk Enter.

Voeg code toe. De code doet een GET request naar `http://localhost:8080/filialen`.

De code krijgt een lijst met filialen van de backend.

De code maakt per filiaal een hyperlink. De code voegt de hyperlink toe aan de lijst (`<ul>`).

```
"use strict";
const filialenUrl = "http://localhost:8080/filialen";
LeesFilialen();
async function LeesFilialen() {
  try {
    const response = await fetch(filialenUrl);
    if (response.ok) {
      const filialen = await response.json();
      maakHyperLinksMet(filialen);
    } else {
      technischeFout();
    }
  } catch {
    technischeFout();
  }
}
function maakHyperLinksMet(filialen) {
  const ul = document.getElementById("filialen");
  for (const filiaal of filialen._embedded.filiaalIdNaamList) {
    const li = maakLiMet(filiaal.naam, filiaal._links.self.href);
    ul.appendChild(li);
  }
}
```

```
function maakLiMet(naam, url) {
  const li = document.createElement("li");
  const hyperlink = document.createElement("a");
  hyperlink.innerText = naam;
  hyperlink.href = "#";
  hyperlink.dataset.url = url;
  hyperlink.onclick = function () {
    LeesFiliaalMetUrl(this.dataset.url);
  };
  li.appendChild(hyperlink);
  return li;
}
function technischeFout() {
  document.getElementById("technischeFout").hidden = false;
}
```

Klik in het venster Project met de rechtermuisknop op index.html en kies Run 'index.html'.  
Je ziet het resultaat in de browser.

### 16.3.4 Detail

Als de gebruiker op een hyperlink klikt, toon je de detail van het filiaal.

Voeg code toe aan index.html, na </ul>:

```
<h1>Detail</h1>
<dl>
  <dt>Nummer</dt>
  <dd id="id"></dd>
  <dt>Naam</dt>
  <dd id="naam"></dd>
  <dt>Gemeente</dt>
  <dd id="gemeente"></dd>
  <dt>Omzet</dt>
  <dd id="omzet"></dd>
</dl>
```

Voeg code toe aan index.js:

```
async function LeesFiliaalMetUrl(url) {
  try {
    const response = await fetch(url);
    if (response.ok) {
      const filiaal = await response.json();
      toonDetailVan(filiaal);
    } else {
      technischeFout();
    }
  } catch {
    technischeFout();
  }
}
function toonDetailVan(filiaal) {
  document.getElementById("id").innerText = filiaal.id;
  document.getElementById("naam").innerText = filiaal.naam;
  document.getElementById("gemeente").innerText = filiaal.gemeente;
  document.getElementById("omzet").innerText = filiaal.omzet;
}
```

Klik in het venster Project met de rechtermuisknop op index.html en kies Run 'index.html'.  
Je ziet het resultaat in de browser.

### 16.3.5 Toevoegen

De gebruiker kan een filiaal toevoegen.

Voeg code toe aan index.html, na </dl>:

```
<h1>Nieuw filiaal:</h1>
<input id="nieuweNaam" required autofocus placeholder="Naam">
<span id="nieuweNaamFout" hidden class="fout">Verplicht.</span>
<input id="nieuweGemeente" required placeholder="Gemeente">
<span id="nieuweGemeenteFout" hidden class="fout">Verplicht.</span>
<input id="nieuweOmzet" required type="number" min="1" placeholder="Omzet">
<span id="nieuweOmzetFout" hidden class="fout">Verplicht.</span>
<button id="toevoegen">Toevoegen</button>
```

Voeg code toe aan index.js:

```
document.getElementById("toevoegen").onclick = toevoegen;
async function toevoegen() {
  const verkeerdeElementen = document.querySelectorAll(":invalid");
  for (const element of verkeerdeElementen) {
    document.getElementById(`${element.id}Fout`).hidden = false;
  }
  const correcteElementen = document.querySelectorAll(":valid");
  for (const element of correcteElementen) {
    document.getElementById(`${element.id}Fout`).hidden = true;
  }
  if (verkeerdeElementen.length === 0) {
    const filiaal = {
      naam: document.getElementById("nieuweNaam").value,
      gemeente: document.getElementById("nieuweGemeente").value,
      omzet: document.getElementById("nieuweOmzet").value
    };
    try {
      const response = await fetch(filialenUrl,
        {method: "POST", headers: {"content-type": "application/json"},
        body: JSON.stringify(filiaal)});
      if (response.ok) {
        const ul = document.getElementById("filialen");
        const li = maakLiMet(filiaal.naam, response.headers.get("Location"));
        ul.appendChild(li);
        document.getElementById("technischeFout").style.display = "none"
      } else {
        technischeFout();
      }
    } catch {
      technischeFout();
    }
  }
}
```

Klik in het venster Project met de rechtermuisknop op index.html en kies Run 'index.html'.  
Je ziet het resultaat in de browser.



Commit de sources. Publiceer op je remote repository.



Muziek client

## 17 VALIDATION ANNOTATIONS MAKEN

Je gebruikt regelmatig validation annotations: @NotNull, @NotBlank, ...

Je leert hier *zelf* validation annotations maken.

### 17.1 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ be.vdab bij Group.
4. Typ validation bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 17.2 pom.xml

Voeg een dependency toe:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Klik rechts op het verticale tabblad Maven. Klik daar op .

### 17.3 Standaard annotations

Voor je *zelf* een validation annotation maakt, maak je een class waarin je *standaard* validation annotations gebruikt.

```
package be.vdab.validation.domain;
// enkele imports
public class Product {
    @PositiveOrZero
    @Digits(integer = 7, fraction = 2)
    private BigDecimal aankoopPrijs;
    @PositiveOrZero
    @Digits(integer = 7, fraction = 2)
    private BigDecimal verkoopPrijs;
    // getters en setters
}
```

### 17.4 Test

Maak een test. Hij controleert of je de *juiste* validation annotations gebruikte in Product.

Als je bijvoorbeeld @Negative gebruikte in plaats van @PositiveOrZero, is Product verkeerd.

```
package be.vdab.validation.domain;
// enkele andere imports
class ProductTest {
    private Validator validator;
    private Product product;
    @BeforeEach
    void beforeEach() {
        ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
        validator = factory.getValidator();
        product = new Product();
        product.setAankoopPrijs(BigDecimal.ONE);
        product.setVerkoopPrijs(BigDecimal.TEN);
    }
}
```

```

@Test
void correctProduct() {
    assertThat(validator.validate(product)).isEmpty();
}
@ParameterizedTest
@ValueSource(strings = {"-1", "-1.234", "12345678"})
void verkeerdeAankoopPrijzen(String prijs) {
    product.setAankoopPrijzen(new BigDecimal(prijs));
    assertThat(validator.validate(product)).isNotEmpty();
}
@ParameterizedTest
@ValueSource(strings = {"-10", "10.234", "12345678"})
void verkeerdeVerkoopPrijzen(String prijs) {
    product.setVerkoopPrijzen(new BigDecimal(prijs));
    assertThat(validator.validate(product)).isNotEmpty();
}
}

```

- (1) Tot nu deed Spring de validatie voor je, als jij voor de parameter van een controller method `@Valid` type. In deze test doe je de validatie *zelf*. Je doet dit met een `Validator` object.
- (2) Je maakt een `Validator` object met een `ValidatorFactory` object. Dit is een toepassing van het factory design pattern. Je krijgt het `ValidatorFactory` object zelf van de static method `buildDefaultValidatorFactory` van de class `Validation`.
- (3) Je maakt het `Validation` object.
- (4) Je valideert het product. `validate` geeft je een verzameling met fouten in het product. De verzameling moet leeg zijn: het product is correct gemaakt in `beforeEach`.
- (5) Dit zijn verkeerde waarden voor de aankoopprijs.
- (6) JUnit roept per waarde bij ⑥ de method op en geeft de waarde mee in `prijs`.
- (7) Je vult de aankoopprijs met deze verkeerde waarde.
- (8) Je valideert het product. `validate` geeft je een verzameling met fouten. De verzameling mag niet leeg zijn: de aankoopprijs is verkeerd.

Voer de testen uit. Ze lukken.

Je type in `Product` *dezelfde* validation annotations bij aankoopprijs en bij verkoopprijs. Code herhalen is niet goed. De oplossing: je maak zelf een validation annotation `@Prijs`. Deze controleert dat het getal

- niet negatief is
- maximaal zeven cijfers voor de komma bevat
- maximaal twee decimalen bevat.

Je typt dan bij aankoopprijs en bij verkoopprijs slechts één validation annotation: `@Prijs`.

Je kan een validation annotation op twee manieren maken:

- Als een samenstelling van andere validation annotations.
- De validatie in detail uitprogrammeren

## 17.5 Samenstelling van andere validation annotations

Je maakt `@Prijs` als een samenstelling van `@PositiveOrZero` en `@Digits(integer = 7, fraction = 2)`.

Je maakt een package `be.vdab.validation.constraints` en daarin de annotation `@Prijs`:

1. Maak de package `be.vdab.validation.constraints`.
2. Klik met de rechtermuisknop op de package `be.vdab.validation.constraints`.
3. Kies New, Java Class.
4. Typ `Prijs` bij Name.
5. Kies Annotation.
6. Druk Enter.

Wijzig de source:

```
package be.vdab.validation.constraints;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
// enkele andere imports
@Target({METHOD, FIELD, ANNOTATION_TYPE})
@Retention(RUNTIME)
@Constraint(validatedBy = {})
@PositiveOrZero
@Digits(integer = 7, fraction = 2)
public @interface Prijs {
    String message() default "{be.vdab.Prijs.message}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

- (1) Je definieert voor welke source onderdelen je @Prijs kan typen.
  - a. METHOD voor een method  
(Je kan @Prijs schrijven voor een getter)
  - b. FIELD voor de declaratie van een instance variabele  
(Je kan @Prijs schrijven voor voor een instance variabele)
  - c. ANNOTATION\_TYPE voor de definitie van een andere annotation  
(Je kan @Prijs gebruiken als basis van nog een andere eigen annotation.
- (2) Je definieert hoe lang Java de annotation behoudt. RUNTIME betekent dat de annotation bij het uitvoeren van het programma nog ter beschikking is. Als je SOURCE zou kiezen in plaats van RUNTIME, neemt de compiler de annotation niet op in de bytecode.
- (3) Je typt bij een annotation @Constraint als die annotation een *validation* annotation is. validatedBy is verplicht te vermelden. Je geeft een lege array mee als je een validation annotation maakt op basis van andere validation annotations, zoals hier.
- (4) Je typt een bean validation annotation waarop je de eigen bean validation baseert.
- (5) Je typt nog een bean validation annotation waarop je de eigen bean validation baseert.
- (6) Je maakt een annotation met het keyword @interface.
- (7) Je definieert een annotation parameter. Dit lijkt op de syntax van een method declaratie. String message() betekent dat je aan @Prijs een parameter message kan meegeven. Als je de parameter message niet meegeeft krijgt hij een default waarde. Je geeft die waarde mee met het keyword default. Je bepaalt met de message parameter de key van de foutboodschap die Spring opzoekt in ValidationMessages.properties.
- (8) Een validation annotation moet ook een parameter groups hebben. Het gebruik van die parameter valt buiten de cursus.
- (9) Een validation annotation moet ook een parameter payload hebben. Het gebruik van die parameter valt buiten de cursus.

Vervang in Product twee keer @PositiveOrZero en @Digits(integer = 7, fraction = 2) door @Prijs.

Voer de testen uit. Ze lukken. @Prijs werkt dus correct.

## 17.6 Eigen bean validation in detail

Sommige bean validations bevatten complexe validaties.

Je kan die soms niet uitdrukken als een samenstelling van andere validation annotations.

Je leert hier hoe je zo'n bean validation schrijft, waarbij je de validatie zelf in Java code schrijft.

Wijzig de code van Prijs, vanaf `@Constraint(...)`:

```
@Constraint(validatedBy = PrijsValidator.class) ❶
public @interface Prijs {
    String message() default "{be.vdab.Prijs.message}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

- (1) Je definieert straks in een aparte class (PrijsValidator) de code die de prijs valideert. Je koppelt die class aan de annotation met `@Constraint`.

Maak in dezelfde package PrijsValidator:

```
package be.vdab.validation.constraints;
// enkele imports
public class PrijsValidator implements ConstraintValidator<Prijs, BigDecimal> { ❶
    @Override
    public void initialize(Prijs constraintAnnotation) { ❷
    }
    @Override
    public boolean isValid(BigDecimal value, ConstraintValidatorContext context){ ❸
        if (value == null) { ❹
            return true;
        }
        if (value.compareTo(BigDecimal.ZERO) < 0) { ❺
            return false;
        }
        var string = value.toString();
        var positieVanDePunt = string.indexOf(".");
        if (positieVanDePunt == -1) {
            return string.length() <= 7; ❻
        }
        if (positieVanDePunt > 7) { ❼
            return false;
        }
        return string.length() - positieVanDePunt <= 3; ❽
    }
}
```

- (1) Een class met de code van een bean validation annotation implementeert `ConstraintValidator`. Je typt tussen `<` en `>` eerst je bean validation annotation. Je typt daarna het type variabele waarbij de annotation mag voorkomen. `@Prijs` mag voorkomen bij een `BigDecimal` variabele.
- (2) Het gebruik van `initialize` valt buiten de cursus.
- (3) Bean validation roept `isValid` op bij de validatie van een variabele met `@Prijs`. Je geeft `true` terug als de variabele een correcte waarde bevat.
- (4) Zoals de standaard annotations (`@Min`, ...) aanzie je de waarde correct als ze `null` bevat. Je gebruikt `@NotNull` in de class `Product` als je de waarde geen `null` mag bevatten.
- (5) Je geeft `false` terug als de prijs negatief is.
- (6) Als de prijs geen decimalen bevat, geef je `true` terug als het aantal cijfers maximaal 7 is.
- (7) Als de prijs decimalen bevat, geef je `true` terug bij maximaal 7 cijfers voor de komma.
- (8) Er mogen maximaal 2 decimalen zijn.

Voer de testen uit. Ze lukken. Ook deze `@Prijs` implementatie werkt dus correct.



## 17.7 Object properties ten opzichte van elkaar valideren

Je annotation valideert één attribuut van een object: verkoopprijs of aankoopprijs.

Je kan ook een validation annotation maken waarmee je het object valideert, niet enkel één attribuut van het object. Je maakt een voorbeeld: @VerkoopPrijsAankoopPrijs valideert een Prijs object: de verkoopprijs moet groter zijn of gelijk aan de aankoopprijs.

```
package be.vdab.validation.constraints;
import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
// enkele andere imports
@Retention(RUNTIME)
@Target({TYPE, ANNOTATION_TYPE})
@Constraint(validatedBy = VerkoopPrijsAankoopPrijsValidator.class)
public @interface VerkoopPrijsAankoopPrijs {
    String message() default "{be.vdab.VerkoopPrijsAankoopPrijs.message}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

- (1) Je definieert ook hier de source onderdelen waarvoor je de annotation kan typen.
- TYPE voor een class
  - ANNOTATION\_TYPE voor de definitie van een andere annotation  
(Je kan de annotation gebruiken als basis van nog een andere eigen annotation).

Maak ook de class VerkoopPrijsAankoopPrijsValidator:

```
package be.vdab.validation.constraints;
// enkele imports
public class VerkoopPrijsAankoopPrijsValidator
    implements ConstraintValidator<VerkoopPrijsAankoopPrijs, Product> {
    @Override
    public void initialize(VerkoopPrijsAankoopPrijs constraintAnnotation) {
    }
    @Override
    public boolean isValid(Product product, ConstraintValidatorContext context) {
        if (product == null) {
            return true;
        }
        if (product.getVerkoopPrijs() == null || product.getAankoopPrijs() == null) {
            return false;
        }
        return product.getVerkoopPrijs().compareTo(product.getAankoopPrijs()) >= 0;
    }
}
```

- (1) @VerkoopPrijsAankoopPrijs mag voorkomen bij de class Product.
- (2) Je produceert geen foutmelding als het te valideren product gelijk zijn aan null, zoals de ingebakken annotations (@Min, @Max, ...) dat ook doen.
- (3) De validatie is correct als de verkoopprijs groter is of gelijk aan de aankoopprijs.

Typ @VerkoopPrijsAankoopPrijs voor de class Product.

Voeg een test toe aan ProductTest:

```
@Test
void aankoopPrijs10EnVerkoopPrijs1IsVerkeerd () {
    product.setAankoopPrijs(BigDecimal.TEN);
    product.setVerkoopPrijs(BigDecimal.ONE);
    assertThat(validator.validate(product)).isNotEmpty();
}
```

Voer de testen uit. Ze lukken. @VerkoopPrijsAankoopPrijs werkt dus correct.



Commit de sources. Publiceer op je remote repository.



Ondernemingsnummer

## 18 PROFILES

Je kan je applicatie in verschillende omgevingen uitvoeren. Voorbeelden:

- Ontwikkelomgeving
- Testomgeving
- Productieomgeving

Je applicatie kan in elke omgeving andere behoeften hebben. Voorbeeld:

- Je applicatie gebruikt in de ontwikkelomgeving en de testomgeving een test database. Daarbij is niet erg als je applicatie per ongeluk verkeerde records schrapt.
- Je applicatie gebruikt in de productieomgeving de productie database. Dit is de database met de gegevens van de eindgebruikers. Hier mag je applicatie niet per ongeluk verkeerde records schrappen.

Je applicatie moet dus per omgeving andere instellingen gebruiken (`application.properties`). Spring biedt daarvoor een oplossing met profiles. Een profile is een synoniem van een omgeving.

Je geeft elk profile een zelf gekozen naam.

We gebruiken in de cursus de profile met de naam *dev* om de ontwikkelomgeving aan te duiden.

We geven de profile, die de productieomgeving voorstelt, geen naam.

We gebruiken voor de productieomgeving de *default* profile. Spring gebruikt de *default* profile als geen enkel andere profile actief is. Je leert straks hoe je een profile activeert.

### 18.1 Databases

- Voer het script `productiedatabase.sql` uit. Het maakt een database `productiedatabase`. Je gebruikt deze database in de productieomgeving.
- Voer ook het script `ontwikkeldatabase.sql` uit. Het maakt een database `ontwikkeldatabase`. Je gebruikt deze database in de ontwikkelomgeving.

### 18.2 Project

Je maakt het project in IntelliJ:

8. Kies in het menu File de opdracht New, Project.
9. Kies links Spring Initializr. Kies Next.
10. Typ `be.vdab` bij Group.
11. Typ `profiles` bij Artifact.
12. Kies 11 bij Java Version. Kies Next.
13. Voeg dependencies toe:

Categorie	Dependency
SQL	Spring Data JPA, MySQL Driver

14. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 18.3 application.properties

Spring gebruikt de instellingen in `application.properties` als de *default* profile actief is.

```
spring.datasource.url=jdbc:mysql://localhost/productiedatabase
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.test.database.replace=none
```

### 18.4 Test

Je bewijst met een test dat de *default* profile actief is.

#### 18.4.1 spring.properties

Maak in de folder `test` een folder `resources`.

Maak daarin het bestand `spring.properties`:

```
spring.test.constructor.autowire.mode=all
```

### 18.4.2 DataSourceTest

```
package be.vdab.profiles.repositories;
// enkele imports
@DataJpaTest
class DataSourceTest {
    private final DataSource dataSource;
    // constructor met parameter
    @Test
    void databaseNaam() throws SQLException {
        try (var connection = dataSource.getConnection()) {
            System.out.println(connection.getCatalog());
        }
    }
}
```

- (1) `getCatalog()` geeft je de naam van de database waarmee je verbonden bent.

Voer de test uit. Je ziet ergens in de diagnostische boodschappen `productiedatabase`.

## 18.5 Dev profile

Maak in `src/main/resources` het bestand `application-dev.properties`. Dit bestand bevat specifieke instellingen als de `dev` profile actief is. De naam van het bestand is dus `application-`, gevolgd door de naam van de profile, gevolgd door `.properties`. Je typt in dit bestand:

```
spring.datasource.url=jdbc:mysql://localhost/ontwikkeldatabase
```

- (1) Als de profile `dev` actief is, gebruikt de applicatie de database `ontwikkeldatabase`.  
 Je activeert straks de profile `dev` Als Spring dan een instelling nodig heeft, zoek Spring die eerst in `application-dev.properties`.  
 Als Spring de instelling daar vindt, gebruikt Spring die instelling.  
 Als Spring de instelling niet vindt, zoekt Spring de instelling in `application.properties`.

## 18.6 Profile activeren

Je activeert nu de profile `dev`. Dit kan op veel manieren. Je leert er twee:

1. Met een command line argument
2. Met een environment variabele

### 18.6.1 Command line argument

Een command-line argument is een waarde die je meegeeft bij het starten van je applicatie.

In het volgend voorbeeld geeft je de waarden `joe` en `jack` mee bij het starten van een applicatie:

```
java -jar MijnApplicatie.jar joe jack
```

Je activeert het profile `dev` met het command-line argument `-Dspring.profiles.active=dev`:

```
java -jar groenetenen.jar -Dspring.profiles.active=dev
```

Tijdens het uitvoeren van een test start je de applicatie niet zoals hier boven, maar vanuit IntelliJ.

Je geeft dan met volgende stappen het command-line argument mee:

1. Kies in het menu Run de opdracht `Edit Configurations`.
2. Kies links JUniten daarbinnen `DataSource.databaseNaam`.
3. Open rechts het onderdeel `Environment`.
4. Typ een spatie en `-Dspring.profiles.active=dev` naast `-ea` bij VM options.
5. Kies OK.

Voer de test uit. Je ziet nu in de diagnostische boodschappen `ontwikkeldatabase`.

### 18.6.2 Environment variable

Verwijder eerst in de configuratie het command-line argument.

Een environment variabele is een variabele die je maakt in het besturingssysteem.

Je kan de inhoud van zo'n variabele lezen in elke applicatie.

Als je een environment variabele maakt met de inhoud `spring_profiles_active` en de inhoud `dev`, activeer je het profile `dev`.


Je doet dit in Windows in een command-line venster als volgt:

```
set spring_profiles_active=dev
```

Je doet dit in Linux/Apple in een terminal venster als volgt:

```
export spring_profiles_active=dev
```

Je simuleert dit nu vanuit IntelliJ. De environment variabele die je zo maakt, geldt niet voor alle applicaties, maar enkel voor de applicatie die je nu ontwikkelt in IntelliJ:

1. Kies in het menu Run de opdracht `Edit Configurations`.
2. Kies links JUniten daarbinnen `DataSource.databaseNaam`.
3. Open rechts het onderdeel `Environment`.
4. Kies  bij `Environment variables`.
5. Kies het `+` teken.
6. Typ `spring_profiles_active` bij `Name`.
7. Typ `dev` bij `Value`.
8. Kies `OK`.
9. Kies `OK`.

Voer de test uit. Je ziet in de diagnostische boodschappen terug `ontwikkeldatabase`.

## 18.7 @Profile

Afhankelijk van het actief zijn van een profile, kan je een bean wel of niet maken.

Als je voor een class `@Profile("dev")` typt,

maakt Spring van die class enkel een bean als het *dev* profile actief is.



Commit de sources. Publiceer op je remote repository.

## 19 DOCKER

Als je applicatie af is, voer je ze uit op productiemachines.

Dit zijn andere computers dan de computer waarop je de applicatie ontwikkelt.

De ontwikkelaar beheert zelden de productiemachines. Andere medewerkers doen dit.

Je kan hierbij problemen hebben:

- ⊖ De productiemachine bevat geen Java runtime of een verouderde Java runtime.
- ⊖ Je applicatie gebruikt een TCP/IP poort die een andere applicatie al gebruikt.
- ⊖ Je applicatie gebruikt MySQL. De productiemachine bevat geen (of een verouderde) MySQL.
- ⊖ Je applicatie spreekt een directory aan die een andere applicatie verwijdt.
- ⊖ Je applicatie leest/schrijft naar een bestand dat een andere applicatie leest/schrijft/verwijdt
- ⊖ ...

Je lost deze problemen op: je voert je applicatie uit in een container.

Een container lijkt op een “computer in je computer”.

Elke container heeft zijn eigen TCP/IP poorten, zijn eigen bestandssysteem, ...

Elke container is zo “geïsoleerd” van de andere containers.

Je voert elke applicatie uit in zijn eigen container.

Je doet dit zowel voor applicaties die jij maakt als applicaties die je nodig hebt (bvb. MySQL).

Docker is een populaire open source technologie om met containers te werken.

### 19.1 Images, daemon, cli, registry

#### 19.1.1 Image

Een docker image is een bestand met de software van een container.

Men zegt ook: een container is een image “in uitvoering”.

De image bevat

- software voor het bestandsbeheer en netwerkbeheer in een container.  
Dit is standaard een minimale versie van Linux.
- je applicatie.
- software die je applicatie nodig heeft , zoals een Java runtime.

De ontwikkelaar maakt de image voor zijn applicatie. Hij kan dus een Java runtime (met de juiste versie) aan de image toevoegen. Zo werkt zijn applicatie zeker op de productiemachine.

#### 19.1.2 Daemon

De docker daemon is een applicatie die je computer in de achtergrond uitvoert.

De daemon beheert containers, images, ...

Je kan als mens niet *rechtstreeks* opdrachten geven aan de docker daemon.

#### 19.1.3 CLI

De docker cli is is het programma docker dat je uitvoert aan de command line prompt.

De cli geeft elke opdracht die je typt door aan de docker daemon. Die voert de opdracht uit.

Je zal de cli regelmatig gebruiken.

#### 19.1.4 Registry

Een docker registry is een website met docker images.

Docker hub is een *publieke* docker registry op <https://hub.docker.com>

Iedereen heeft toegang tot Docker hub.

Docker hub bevat images voor veel populaire applicaties (MySQL, Tomcat, ....).

Je kan in het zoekveld bovenaan de startpagina bijvoorbeeld MySQL typen en Enter drukken.

Je ziet dan een lijst met images met een MySQL server of een aanverwant product.

### 19.1.5 Volumes

Nadat je een container verwijderd gaat het bestandssysteem in die container verloren. Je lost dit op met een volume. Dit is een bestandssysteem dat Docker los van je container beheert. Je verbindt je container met een zo'n volume. Alle data die het programma in het bestandssysteem van de container schrijft, schrijft docker dan in het bijbehorende volume. Nadat je de container verwijdert, bevat het volume deze data nog.

### 19.1.6 Networks

Een applicatie in een container (bijvoorbeeld een website) communiceert soms over TCP/IP met een applicatie in een andere container (bijvoorbeeld MySQL). Dit kan als je in Docker een netwerk definieert en beide containers tot dit netwerk behoren.

### 19.1.7 Overzicht



## 19.2 Docker desktop

Docker desktop is een eenvoudig te gebruiken versie van docker op Windows en Apple.

Je download Docker desktop op

<https://hub.docker.com/editions/community/docker-ce-desktop-windows>

Voer de installatie uit. Verwijder het vinkje bij Enable WSL 2 Windows Features.

Plaats *geen* vinkje bij Use Windows containers instead of Linux containers.

- Linux containers kunnen perfect Java applicaties uitvoeren.
- Docker hub bevat meer Linux images dan Windows images.

Herstart je computer. Vanaf nu start docker bij de start van je computer. Je ziet dit rechts onder in de Windows status bar. Het icoon met de walvis stelt docker voor. Zolang de walvis beweegt is docker nog aan het starten. Je wacht dan nog even voor je docker gebruikt.

## 19.3 Opdrachten

### 19.3.1 Opdrachten voor images

Open een command line venster. Tik volgende opdracht:

**docker pull hello-world**

①                      ②

- (1) De pull opdracht downloadt een image van de docker hub naar je computer.
- (2) De naam van de image. hello-world is een eenvoudige image.  
Je voert die straks uit in een container.  
Het programma in de container zal een boodschap Hello tonen.

Waar en hoe Docker images bewaart op je computer is interne huishouding van Docker.

**docker images**

①

- (1) De images opdracht toont een lijst van de images op je computer.

**docker image rm hello-world**

① ②

- (1) De image rm opdracht verwijdert een image van je computer.  
 (2) De naam van de image.

Voer de opdracht docker images uit. Je ziet een lege lijst.

### 19.3.2 Opdrachten voor containers

**docker create --name hallo hello-world**

① ② ③

- (1) De create opdracht maakt een container. De container is dan nog niet gestart.  
 (2) Een vrij te kiezen naam voor de container.  
 Je verwijst met die naam in volgende opdrachten naar de container.  
 (3) De naam van de image waarop je de container baseert.  
 Als deze image niet op je computer bestaat, downloadt Docker die van docker hub.

**docker start hallo**

① ②

- (1) De start opdracht start een container.  
 (2) De naam van de container.

Je ziet hallo. Dit is enkel de naam van de gestarte container.

Het is niet wat het programma in de container op het scherm toont.

**docker logs hallo**

① ②

- (1) De logs opdracht toont wat een programma in een container op het scherm toont.  
 (2) De naam van de container.

Het programma toont veel tekst, onder andere Hello from Docker!

**docker ps -a**

① ② ③

- (1) De ps opdracht toont een lijst met de containers op je computer.  
 (2) De ps opdracht toont standaard enkel containers die nog in uitvoering zijn.  
 De container hallo is gestopt na het tonen van wat tekst op het scherm.  
 Met -a toont docker ook containers die gestopt zijn.

Je kan het console venster wat breder maken om alle kolommen *naast elkaar* te zien.

**docker rm hallo**

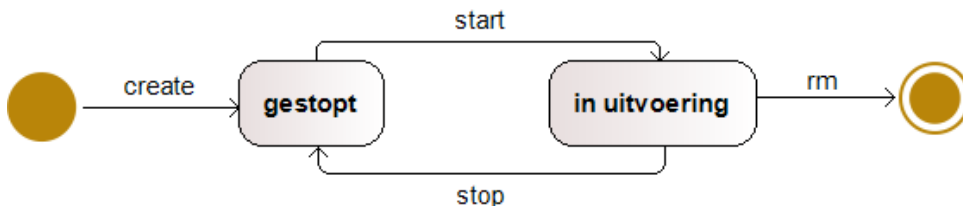
① ②

- (1) De rm opdracht verwijdert een container van je computer.  
 (2) De naam van de container.

Voer de opdracht docker container ps -a uit. Je ziet een lege lijst.

### 19.3.3 Levensfasen

Je ziet de levensfasen van een container. De pijlen zijn docker opdrachten.





### 19.3.4 docker run

docker run --rm --name hallo hello-world

①      ②                      ③                      ④

- (1) De run opdracht doet meerdere handelingen die je hiervoor met aparte opdrachten deed.
  - a. Hij maakt een container op basis van een image.  
Als je computer deze image nog niet bevat, download Docker die van docker hub.
  - b. Hij start de container.
  - c. Hij toont wat het programma in de container op het scherm toont.
- (2) Met `--rm` verwijdert docker de container nadat de container stopt.
- (3) De naam van de container.
- (4) De naam van de image waarop je de container baseert.

## 19.4 Eigen applicatie

Je maakt een Spring Boot website. Je zal die uitvoeren in een container.

### 19.4.1 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ `be.vdab` bij Group.
4. Typ landen bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web
Template Engines	Thymeleaf

7. Kies Next, Finish.

Je verwijdert de test `LandenApplicationTests`.

### 19.4.2 Controller

```
package be.vdab.landén.controllers;
// enkele imports
@Controller
@RequestMapping("/")
class IndexController {
    IndexController()
        System.out.println("Besturingssysteem:" + System.getProperty("os.name"));
        System.out.println("Gebruikersnaam:" + System.getProperty("user.name"));
        System.out.println("Home directory van de gebruiker:" +
            System.getProperty("user.home"));
    }
}
```

### 19.4.3 Image

Spring Boot heeft voorzieningen om van je applicatie een Docker image te maken. Je definieert de naam van image in `pom.xml`, na `<artifactId>` `spring-boot-maven-plugin` `</artifactId>`:

```
<configuration>
  <image>
    <name>landen</name>
  </image>
</configuration>
```

①

- (1) Je definieert de naam van de image als landen.

Je maakt de image:

1. Open rechts het verticale tabblad Maven.
2. Open het onderdeel Plugins.
3. Open daarin het onderdeel spring-boot.
4. Dubbelklik `spring-boot:build-image`. Dit duurt de eerste keer zeer lang.

#### 19.4.4 Uitvoeren

Open een command line venster.

Toon een lijst met de images:

**docker images**

Je ziet in de lijst landen (de image van jouw applicatie).

Voer je programma uit in een container:

**docker run --rm --name landen landen**

Je ziet in de output ook de informatie die je in de `IndexController` constructor afbeeldde.

### 19.5 Container in achtergrond uitvoeren

Je ziet in de console de prompt niet meer. Je kan de container dus niet stoppen.

De reden: je website stopt nooit (in tegenstelling tot het programma `hello-docker`).

Je website blijft voortdurend in uitvoering om browsers requests te beantwoorden.

Start even een nieuwe command line venster, waar je de container stopt:

**docker stop landen**

Je vermijdt deze problemen door `-d` mee te geven bij de run opdracht:

**docker run -d --rm --name landen landen**

❶

- (1) Met `-d` start docker de container als een achtergrondproces. Docker toont een unieke identifier (die je niet nodig hebt) van je container. Je ziet terug de prompt.

Je ziet de opstartinformatie van je website met **docker logs landen**.

Stop de container: **docker stop landen**

### 19.6 Volumes

Elke container heeft zijn eigen bestandssysteem dat het programma in de container kan gebruiken.

Wanneer een container stopt, gaat de data in het bestandssysteem van die container verloren.

Wijzig de constructor van `IndexController` om dit te zien:

```
IndexController() throws IOException {
    var homeDirectory = Path.of(System.getProperty("user.home"));
    var bestand = homeDirectory.resolve("organisatie.txt");
    if (!Files.exists(bestand)) {
        Files.writeString(bestand, "VDAB");
        System.out.println(bestand + " gemaakt");
    }
}
```

❶

❷

- (1) Als de home directory geen bestand `organisatie.txt` bevat,
- (2) maak je dit bestand

Maak de Docker image.

Voer je programma uit:

**docker run -d --rm --name landen landen**

Je ziet met **docker logs landen** dat je applicatie het bestand maakt.

Stop de container. Voer het programma nog eens uit.

Je ziet met **docker logs landen** dat je applicatie het bestand opnieuw maakt.

Het ging verloren na de vorige uitvoering van de container.

Je lost dit probleem op met een Docker volume. Dit is een bestandssysteem dat Docker beheert.

Maak een volume:

**docker volume create volume1**

①

- (1) De (vrij te kiezen) naam van het volume.

Het volume behoudt zijn data tot je het volume ooit verwijdert met `docker volume rm volume1`.

Je doet dit nu nog niet.

Voer je programma uit en koppel het volume aan de directory `/home/cnb` in je container.

Alle handelingen die je programma doet in `/home/cnb` gebeuren in het volume.

**docker run -d --rm --name landen --mount source=volume1,target=/home/cnb landen**

①

②

③

- (1) Je koppelt met `--mount` een volume aan een directory in je container.
- (2) De naam van het volume. Als je dit volume niet vooraf maakte met `docker volume create`, maakt Docker dit volume nu.
- (3) De directory in de container.

Je ziet in de logs dat je applicatie hebt het bestand maakt.

Voer je programma nog eens. Je ziet dat je applicatie het bestand niet meer maakt.

Het bestand bleef dus behouden in het volume na de vorige uitvoering van de container.

### 19.6.1 Kopiëren

Waar en hoe Docker volumes bewaart op je computer is interne huishouding van Docker.

Je kan wel gemakkelijk bestanden kopiëren tussen je container en je computer, zolang de container niet verwijderd is.

Je kopieert het bestand `naam.txt` in de container naar een bestand `naam1.txt` op je computer:

**docker cp landen:/home/cnb/organisatie.txt organisatie1.txt**

①

②

③

④

- (1) De `cp` opdracht kopieert bestanden.
- (2) De naam van de container waaruit je een bestand kopieert.
- (3) Het pad naar en de naam van het bestand dat je kopieert vanuit de container.
- (4) De naam van het bestand waarnaar je kopieert op je computer

Toon de inhoud van `naam1.txt` op je computer:

**type organisatie1.txt**

Je kan ook een bestand op je computer kopiëren naar een container

**docker cp organisatie1.txt landen:/home/cnb/organisatie2.txt**

①

②

③

- (1) De naam van het bestand dat je kopieert vanuit je computer.
- (2) De naam van de container waarnaar je het bestand kopieert.
- (3) Het pad naar en de naam van het bestand in de container.

Stop de container.

### 19.7 TCP/IP poorten

Alle TCP/IP poorten in een container behoren tot die container. Normaal voer je maar één programma in een container uit. Dat programma heeft dus alle TCP/IP poorten ter beschikking.

Een website in een container kan requests verwerken op TCP/IP poort 8080.

Het gaat hier om poort 8080 *in* de container. Je kan op je computer niet surfen naar de website. Je leert hier hoe je dit oplost.

### 19.7.1 Controller

```
package be.vdab.landen.controllers;
// enkele imports
@Controller
@RequestMapping("/")
class IndexController {
    @GetMapping
    public String index() {
        return "index";
    }
}
```

### 19.7.2 Index.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
    <head><title>Welkom</title></head>
    <body>
        <h1>Welkom</h1>
    </body>
</html>
```

Compileer het project. Maak de docker image.

Start een container gebaseerd op de image:

```
docker run -d -p 8081:8080 --rm --name landen landen
```

❶

- (1) Je website gebruikt poort 8080 in de container.  
Je koppelt poort 8081 op je computer met poort 8080 in de container.  
Docker leidt requests naar poort 8081 op je computer om naar poort 8080 in de container.

Surf naar <http://localhost:8081>. Je ziet de welkompagina van je website. Stop de container.

## 19.8 MySQL

### 19.8.1 Container

Start MySQL in een container:

```
docker run -d -p 3307:3306 -e MYSQL_ROOT_PASSWORD=vdab --rm --name landendb ❶  
--mount source=landendb,target=/var/lib/mysql mysql
```

❷

❸

❹

❺

❻

- (1) We kunnen de volledige opdracht niet op één regel tonen. Jij typt hem *wel* aaneengesloten.
- (2) MySQL gebruikt TCP/IP poort 3306 in de container.  
Je koppelt poort 3307 op je computer met poort 3306 in de container.  
Je kan zo met de MySQL WorkBench op je computer de MySQL in de container aanspreken.
- (3) Met `-e` maak je een environment variabele in de container. Je maakt hier een variabele met de naam `MYSQL_ROOT_PASSWORD` en de inhoud `vdab`. De MySQL in de container gebruikt de inhoud van deze variabele als het paswoord van de MySQL gebruiker `root`.
- (4) De naam van de container.
- (5) Je koppelt een volume `landendb` met de directory `/var/lib/mysql` in de container.  
Docker maakt dit volume de eerste keer omdat het dan nog niet bestaat.  
MySQL bewaart databases in de directory `/var/lib/mysql`, dus in het volume.  
Je databases gaan zo niet verloren nadat je de container verwijdt.
- (6) De naam van de image op docker hub.

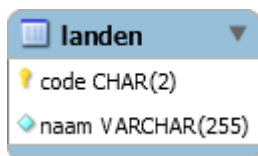
### 19.8.2 Workbench

Start de MySQL Workbench. Je maakt verbinding met de MySQL in de container:

1. Kies in het menu Database de opdracht Connect to Database.
2. Wijzig Port naar 3307.
3. Kies bij Password de knop Store in Vault, typ vdab. Kies OK.
4. Kies OK.

Je bent verbonden met de MySQL in de container. Je ziet links in het tabblad Schemas dat deze geen enkele database bevat, behalve de database sys voor het interne gebruik door MySQL.

Voer het script landen.sql uit. Het maakt een database landen met één table landen:



Je hebt de container nog nodig. Laat hem in uitvoering.

## 19.9 Applicatie spreekt MySQL in container aan

Je breidt het project landen uit zodat het data leest uit de database in de container landendb.

Je voert de website eerst uit op je computer.

Je zorgt er later ook voor dat je de website in een container kan uitvoeren.

### 19.9.1 pom.xml

Voeg de dependencies voor Spring Data en voor MySQL toe:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

Klik rechts op het verticale tabblad Maven. Klik daar op .

### 19.9.2 application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3307/landen
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED
```

(1) Je vermeldt poort 3307: de poort die je verbond met de poort 3306 in de container landendb.

### 19.9.3 Land

```
package be.vdab.landendb.domain;
// enkele imports
@Entity
@Table(name = "landen")
public class Land {
  @Id
  private String code;
  private String naam;
}
```

### 19.9.4 LandRepository

```
package be.vdab.landendb.repositories;
// enkele imports
public interface LandRepository extends JpaRepository<Land, String> {
}
```

### 19.9.5 LandService

```
package be.vdab.landen.services;
public interface LandService {
    long findAantal();
}
```

### 19.9.6 DefaultLandService

```
package be.vdab.landen.services;
// enkele imports
@Service
@Transactional(readOnly = true)
class DefaultLandService implements LandService {
    private final LandRepository repository;
    // constructor met parameters
    @Override
    public long findAantal() {
        return repository.count();
    }
}
```

### 19.9.7 IndexController

```
package be.vdab.landen.controllers;
// enkele imports
@Controller
@RequestMapping("/")
class IndexController {
    private final LandService landService;
    // constructor met parameters
    @GetMapping
    public ModelAndView index() {
        return new ModelAndView("index", "aantalLanden", landService.findAantal());
    }
}
```

### 19.9.8 index.html

Gewijzigde h1 element:

```
<h1>Aantal landen: <span th:text='${aantalLanden}'></span></h1>
```

Je kan de applicatie uitproberen.

## 19.10 Networks

Elke container is geïsoleerd van de andere containers. Ze behoren standaard niet tot hetzelfde netwerk. Ze kunnen dus niet via TCP/IP met elkaar communiceren.

Als je de applicatie landen in een container uitvoert, kan die dus standaard niet de MySQL aanspreken in container landendb.

De oplossing is een Docker netwerk te maken en beide containers in dit netwerk te plaatsen. Een Docker netwerk is geen “hardware” netwerk, maar een “virtueel” netwerk tussen containers.

### 19.10.1 Netwerk maken

Maak een netwerk:

```
docker network create landennet
```

❶

(1) De (vrij te kiezen) naam van het netwerk.

### 19.10.2 Container in netwerk uitvoeren

Stop de container met de database:

```
docker stop landendb
```

Start de container opnieuw, in het netwerk landennet:

```
docker run -d -p 3307:3306 -e MYSQL_ROOT_PASSWORD=vdab --rm --name landendb
--mount source=landendb,target=/var/lib/mysql --network landennet mysql
```

(1) De naam van het netwerk waartoe de container behoort.

De netwerknnaam van de container in dit netwerk is de naam van de container: landendb.

### 19.10.3 Project

Wijzig in het project landen in application.properties de eerste regel:

```
spring.datasource.url=jdbc:mysql://landendb/landen
```

(1) Je verwijst naar de container met de netwerknnaam landendb.

Compileer het project.

Maak een image van het project.

Voer het programma uit in een container die ook behoort tot net netwerk landennet:

```
docker run -d -p 8081:8080 --rm --name landen --network landennet landen
```

Surf naar <http://localhost:8081/>. Je ziet de welkompagina van je programma.

Het is jammer dat je de 1<sup>e</sup> regel in application.properties moest wijzigen:

- ⊖ Als je de applicatie *buiten* de container wil uitvoeren, moet je deze regel opnieuw wijzigen.
- ⊖ De naam van het docker netwerk is hier hard gecodeerd.  
Als de netwerknnaam wijzigt, moet je de binnenkant van je applicatie wijzigen.

Wijzig de regel terug, zodat je de applicatie *buiten* een container kan uitvoeren:

```
spring.datasource.url=jdbc:mysql://localhost:3307/landen
```

Compileert het project. Maak een image.

Stop de container met de applicatie:

```
docker stop landen
```

Start de applicatie in een container, maar je geeft nu een environment variabele mee:

```
docker run -d -p 8081:8080 --rm --name landen --network landennet
-e spring.datasource.url=jdbc:mysql://landendb/landen landen
```

- (1) Je geeft een environment variabele mee. De naam is gelijk aan de key van de eerste regel uit application.properties. Als Spring zo'n environment variabele ziet, leest hij de JDBC URL niet uit application.properties, maar uit de environment variabele.  
Je verwijst dus op de computer met de netwerknnaam landendb naar de database landen.

Surf naar <http://localhost:8081/>. Je ziet de welkompagina van je programma.

Opmerking: je kan op dezelfde manier environment variabelen spring.datasource.username en spring.datasource.password meegeven. Je definieert zo de MySQL gebruiker waarmee je applicatie inlogt op de MySQL in de container.



Grote firma's hebben honderden containers op tientallen servers.  
Je kan zoveel containers niet meer handmatig starten, stoppen ...  
Kubernetes is een tool om het beheer van veel containers te automatiseren.



Commit de sources. Publiceer op je remote repository.



Docker

## 20 MESSAGING

Je gebruikt tot nu REST om twee applicaties te laten samenwerken.

Je leert hier een alternatief: messaging.

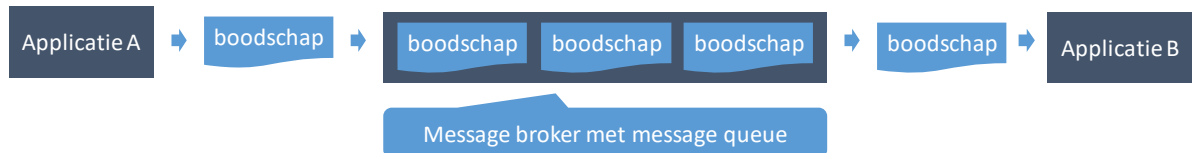
Een applicatie A wil de diensten oproepen van een applicatie B.

A stuurt daartoe een bericht naar een wachtrij: een message queue.

Aan de andere kant van de message queue ontvangt B de berichten en verwerkt ze.

Nadat B een bericht correct verwerkt, verdwijnt het bericht uit de queue.

De message queue behoort tot een message broker: een applicatie die de message queue beheert.



Voordelen van messaging:

- De communicatie verloopt asynchroon.
  - A gaat, na het afleveren van het bericht aan de message broker, verder met zijn werk.
  - A wacht niet tot B het bericht ontvangen en verwerkt heeft.
  - Bij REST verloopt de communicatie synchroon.: A stuurt een request naar B en pauzeert zijn werk tot B de request ontvangt, verwerkt en een response naar A terugstuurt.
- B moet niet noodzakelijk online zijn opdat A zou kunnen werken.
  - Terwijl B offline is, kan A berichten blijven sturen naar de message broker.
  - De message broker onthoudt de berichten in zijn queue.
  - Zodra B terug online stuurt de message broker de berichten naar B.
- A kan tijdelijk sneller berichten versturen dan B ze kan verwerken.
  - De message broker houdt de berichten bij in de queue
  - en stuurt ze naar B met de snelheid waarmee B de berichten verwerkt.
- De message broker kan een bericht naar één of *meerdere* applicaties sturen.
  - De message broker kan een bericht van A sturen naar B én naar C.

A en B zijn niet altijd in dezelfde programmeertaal geprogrammeerd.

Ze moeten elkaars berichten toch begrijpen. Je plaatst daarom de data in de berichten in een dataformaat dat alle programmeertalen kennen: JSON of XML.

Er bestaan verschillende message brokers: RabbitMQ, ActiveMQ, , IBM MQ, ...

Je gebruikt in deze cursus RabbitMQ.



## 21 RABBITMQ

### 21.1 Installeren

Dankzij Docker moet je RabbitMQ niet op je computer installeren.

Voer RabbitMQ uit in een container met volgende opdracht:

```
docker run --name rabbitmq -p 5672:5672 -p 15672:15672 -d --rm
--mount source=rabbitmq,target=/var/lib/rabbitmq --hostname rabbit
rabbitmq:3-management-alpine
```

- (1) Je associeert TCP/IP poort 5672 in de container met poort 5672 op je computer. Programma's gebruiken deze poort om berichten uit te wisselen met RabbitMQ. Ze gebruiken daarbij AMQP (Advanced Message Queuing Protocol). AMQP is een standaard protocol voor messaging. AMQP gebruikt zelf het TCP/IP protocol. Programma's die berichten sturen, RabbitMQ én programma's die berichten ontvangen kunnen zich dus op verschillende computers bevinden.
- (2) RabbitMQ bevat een ingebakken webserver. Je surft er straks naar om RabbitMQ te beheren. De webserver gebruikt poort 15672. Je associeert daarom TCP/IP poort 15672 in de container met poort 15672 op je computer.
- (3) Je koppelt een volume rabbitmq met de directory /var/lib/rabbitmq in de container. Docker maakt dit volume de eerste keer omdat het dan nog niet bestaat. RabbitMQ bewaart messages in de directory /var/lib/rabbitmq, dus in het volume. Je messages gaan zo niet verloren nadat je de container verwijdert.
- (4) RabbitMQ maakt in de directory /var/lib/rabbitmq bestanden met de berichten. De naam van die bestanden wordt bepaald door de hostname van de container. Je plaatst hier de hostname van de container op (een vrij te kiezen naam) rabbit. Als je de hostname niet invult, kiest Docker een random hostname. Als je de image meerdere keren uitvoert krijgt elke container dan een andere hostname. De messages van de vorige uitvoering van de image zouden dan verloren gaan.
- (5) De naam van de RabbitMQ image.

RabbitMQ starten duurt even. Typ enkele keren de opdracht `docker logs rabbitmq`, tot je de tekst `Server startup complete` ziet.

### 21.2 RabbitMQ management

Surf naar <http://localhost:15672>. Log in in met de gebruiker `guest` en het paswoord `guest`.

### 21.3 Exchanges en queues

RabbitMQ bevat twee belangrijke onderdelen: exchanges en queues.

- Je *stuurt* een bericht naar een exchange. Die stuurt het bericht verder naar één of meerdere queues.
- Je *ontvangt* berichten uit een queue.

Er bestaan meerdere soorten exchanges. De belangrijkste zijn direct en fanout.

#### 21.3.1 Direct exchange

Een direct exchange is verbonden met één queue.

De exchange stuurt elk bericht dat hij ontvangt door naar die ene queue.



Je maakt in de RabbitMQ management een direct exchange met de naam exchange1:

1. Kies het tabblad Exchanges.
2. Typ exchange1 bij Name (onder Add a new exchange).
3. Kies Add exchange. Je ziet je exchange1 in de lijst (boven Add a new exchange).

Je maakt een queue met de naam queue1:

1. Kies het tabblad Queues.
2. Typ queue1 bij Name (onder Add a new queue).
3. Kies Add queue. Je ziet je queue in de lijst (boven Add a new exchange).

Je verbindt queue1 met exchange1:

1. Kies queue1 in de lijst.
2. Typ exchange1 bij From exchange (onder Add binding to this queue).
3. Kies Bind. Je ziet de verbinding naar exchange1 boven Add binding to this queue.

Je stuurt een testbericht naar exchange1:

1. Kies het tabblad Exchanges.
2. Kies exchange1 in de lijst.
3. Kies Publish message.
4. Typ test1 bij Payload (payload is de inhoud van het bericht)
5. Kies Publish message. Je verstuurt zo het bericht.

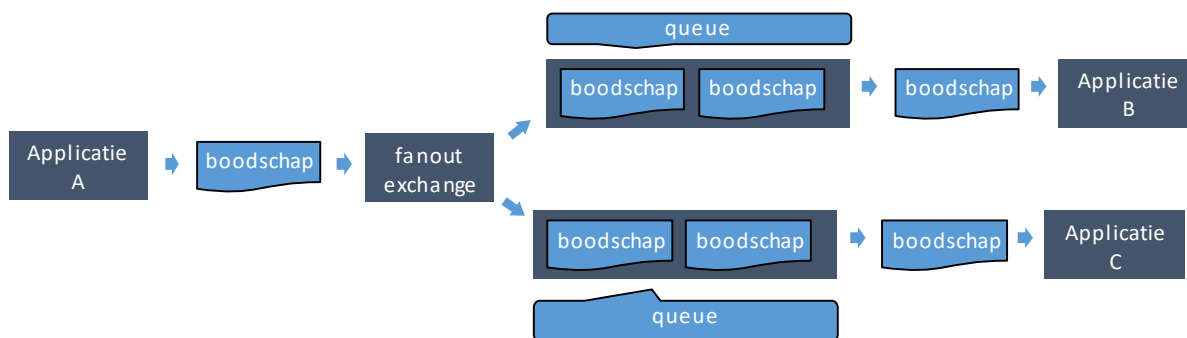
Je controleert dat exchange1 het bericht doorstuurt naar queue1:

1. Kies het tabblad Queues.
2. Kies queue1 in de lijst.
3. Kies Get Message(s).
4. Je ziet dat er een bericht is aangekomen met als Payload test1.

### 21.3.2 Fanout Exchange

Een fanout exchange is verbonden met één of meerdere queues.

De exchange stuurt elk bericht dat hij ontvangt door naar die queue(s).



Je maakt een fanout exchange met de naam exchange2:

1. Kies het tabblad Exchanges.
2. Typ exchange2 bij Name (onder Add a new exchange).
3. Kies fanout bij Type.
4. Kies Add exchange. Je ziet je exchange1 in de lijst (boven Add a new exchange).

Maak een queue met de naam queue2.

Verbind die met exchange2.

Je doet dit op dezelfde manier als bij een direct exchange.

Maak een queue met de naam queue3.

Verbind die met exchange2.

Stuur een testbericht met als payload test2 naar exchange2.

Controleer dat exchange2 het bericht doorstuurt naar queue2.

Controleer dat exchange2 het bericht ook doorstuurt naar queue3.

## 22 BERICHTEN STUREN

### 22.1 Voorbeeld

Je maakt een applicatie met de naam `catalogus`. De applicatie is een REST service. Ze krijgt POST requests binnen. Zo'n request bevat de naam van een nieuw artikel. De applicatie voegt met die naam in zijn database een record toe aan de table `artikels`. De applicatie stuurt ook een bericht met informatie over het artikel naar RabbitMQ.

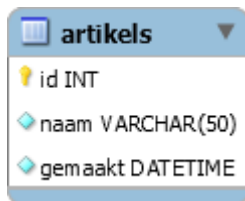
Je maakt in het volgende hoofdstuk een andere applicatie. Die ontvangt de berichten. De applicatie werkt zijn eigen database bij met de data in de berichten.

### 22.2 RabbitMQ

1. Maak een fanout exchange met de naam `catalogus`.
2. Maak een queue met de naam `voorraad`.
3. Verbind de queue met de exchange.

### 22.3 Database

Voer het script `catalogs.sql` uit. Het maakt een database `catalogs`. Die bevat één table:



De table is leeg. Je zal records toevoegen met je applicatie.

### 22.4 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ `be.vdab` bij Group.
4. Typ `catalogus` bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web
SQL	Spring Data JPA , MySQL Driver
Messaging	Spring for RabbitMQ

7. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 22.5 application.properties

```
spring.datasource.url=jdbc:mysql://localhost/catalogus
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED
```

## 22.6 Artikel entity

```
package be.vdab.catalogus.domain;
// enkele imports
@Entity
@Table(name = "artikels")
public class Artikel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String naam;
    private LocalDateTime gemaakt;
    public Artikel(String naam) {
        this.naam = naam;
        gemaakt = LocalDateTime.now();
    }
    // protected default constructor
    // getters
}
```

## 22.7 ArtikelRepository

```
package be.vdab.catalogus.repositories;
// enkele imports
public interface ArtikelRepository extends JpaRepository<Artikel, Long> {
}
```

## 22.8 ArtikelGemaakt

Maak de class die het bericht voorstelt dat je stuurt naar RabbitMQ.  
Zo'n bericht moet niet *alle* data over het nieuwe artikel bevatten,  
enkel data die interessant is voor *andere* applicaties.

```
package be.vdab.catalogus.events; ❶
import be.vdab.catalogus.domain.Artikel;
public class ArtikelGemaakt {
    private final long id;
    private final String naam;
    public ArtikelGemaakt(Artikel artikel) { ❷
        id = artikel.getId();
        naam = artikel.getNaam();
    }
    // getters
}
```

- (1) De class ArtikelGemaakt stelt een event (gebeurtenis) voor die optreedt in je applicatie. Andere applicaties kunnen geïnteresseerd zijn in dit event om zelf iets te doen, zoals hun eigen database bijwerken. Men spreekt over een event driven architecture. Een applicatie waarin een event optreedt, stuurt messages naar andere applicaties met informatie over dit event.
- (2) Deze constructor vult de private variabelen met de overeenstemmende informatie uit een Artikel object dat als parameter binnenkomt.

## 22.9 ArtikelService

```
package be.vdab.catalogus.services;
import be.vdab.catalogus.domain.Artikel;
public interface ArtikelService {
    void create(Artikel artikel); ❶
}
```

- (1) Je zal deze method oproepen in een RestController.

## 22.10 CatalogusApplication

Voeg code toe:

```
@Bean
Jackson2JsonMessageConverter converter() {
    return new Jackson2JsonMessageConverter();
}
```

❶

- (1) Als je een bericht stuurt naar RabbitMQ, stel je het bericht eerst voor als een Java object. Spring converteert dit object standaard naar een bericht met serialization. Dit heeft nadelen:
  - a. Je kan de inhoud van het bericht niet lezen in de RabbitMQ management console.
  - b. Een applicatie die niet in Java geschreven is, kan het bericht niet lezen.
 Je lost dit op door een bean te maken van het type Jackson2JsonMessageConverter. Als zo'n bean beschikbaar is, converteert Spring met deze bean het Java object naar JSON. Het bericht bevat dus JSON data. Iedereen kan de inhoud van het bericht lezen.

## 22.11 DefaultArtikelService

```
package be.vdab.catalogus.services;
// enkele imports
@Service
@Transactional
class DefaultArtikelService implements ArtikelService {
    private final ArtikelRepository repository;
    private final AmqpTemplate template;
    //constructor met parameters
    @Override
    public void create(Artikel artikel) {
        repository.save(artikel);
        template.convertAndSend("catalogus", null, new ArtikelGemaakt(artikel));
    }
}
```

❶  
❷  
❸  
❹

- (1) Je stuurt met een AmqpTemplate berichten naar RabbitMQ.
- (2) Spring heeft een AmqpTemplate bean ter beschikking als je de Spring for RabbitMQ dependency toevoegt aan je project. Je injecteert de bean hier.
- (3) Je voegt het artikel toe aan de database.
- (4) Je verstuurt een bericht met convertAndSend.
 

De eerste parameter is de naam van de exchange waar je het bericht naar stuurt.

De tweede parameter is voor geavanceerd gebruik.

De derde parameter is het object dat het bericht voorstelt.

Spring converteert dit object naar JSON en vult het bericht met deze JSON.

## 22.12 ArtikelController

```
package be.vdab.catalogus.restcontrollers;
// enkele imports
@RestController
@RequestMapping("/artikels")
class ArtikelController {
    private final ArtikelService artikelService;
    // constructor met parameters
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    void post(@RequestBody String naam) {
        artikelService.create(new Artikel(naam));
    }
}
```

❶  
❷

- (1) De method verwerkt een POST request. Die heeft een body bevat de naam van een artikel.

- (2) De method maakt met deze naam een `Artikel` object.  
De method geeft dit object door aan de `create` method van de service layer.  
Deze method voegt het artikel toe aan de database en stuurt een bericht naar RabbitMQ.

### 22.13 Uitvoeren

Start de applicatie.

Doe vanuit IntelliJ een POST request naar `http://localhost:8080/artikels`.

Plaats de request header `Content-type` op `application/json`.

Typ in de body de naam van een artikel, bijvoorbeeld `appel`.

Verstuur de request.

In de database bevat de table `artikels` een nieuw record.

Kijk in RabbitMQ in de queue `voorraad`. Die bevat een bericht:

```
{"id":1,"naam":"appel"}
```



Commit de sources. Publiceer op je remote repository.



Sportwinkel

## 23 BERICHTEN ONTVANGEN

### 23.1 Voorbeeld

Je maakt een applicatie met de naam voorraad.

De applicatie ontvangt berichten uit de queue voorraad.

De applicatie voegt in zijn eigen database met zo'n bericht een record toe aan de table artikels.

### 23.2 Database

Voer het script voorraad.sql uit.

Het maakt een database voorraad.

Die bevat één table.

De table is leeg. Je zal records toevoegen met je applicatie.



### 23.3 Project

Je maakt het project in IntelliJ:

1. Kies in het menu File de opdracht New, Project.
2. Kies links Spring Initializr. Kies Next.
3. Typ be.vdab bij Group.
4. Typ voorraad bij Artifact.
5. Kies 11 bij Java Version. Kies Next.
6. Voeg dependencies toe:

Categorie	Dependency
Developer Tools	Spring Boot DevTools
Web	Spring Web
SQL	Spring Data JPA , MySQL Driver
Messaging	Spring for RabbitMQ

7. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 23.4 application.properties

```
spring.datasource.url=jdbc:mysql://localhost/voorraad
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED
server.port=8081
```

- (1) Deze applicatie gebruikt TCP/IP poort 8081.

Je kan de applicatie kan tegelijk uitvoeren met de applicatie catalogus:

de applicatie catalogus gebruikt een andere TCP/IP poort: 8080.

### 23.5 Artikel entity

```
package be.vdab.voorraad.domain;
// enkele imports
@Entity
@Table(name = "artikels")
public class Artikel {
    @Id
    private long id;
    private int voorraad;
    // protected default constructor
    // public constructor met parameter id
}
```

## 23.6 ArtikelRepository

```
package be.vdab.voorraad.repositories;
// enkele imports
public interface ArtikelRepository extends JpaRepository<Artikel, Long> {
}
```

## 23.7 ArtikelService

```
package be.vdab.voorraad.services;
import be.vdab.voorraad.domain.Artikel;
public interface ArtikelService {
    void create(Artikel artikel);
}
```

## 23.8 DefaultArtikelService

```
package be.vdab.voorraad.services;
// enkele imports
@Service
@Transactional
class DefaultArtikelService implements ArtikelService {
    private final ArtikelRepository repository;
    // constructor met parameter
    @Override
    public void create(Artikel artikel) {
        repository.save(artikel);
    }
}
```

## 23.9 ArtikelGemaakt

Deze class stelt een bericht voor dat je ontvangt uit de queue voorraad.

```
package be.vdab.voorraad.events;
import be.vdab.voorraad.domain.Artikel;
public class ArtikelGemaakt {
    private long id;
    private String naam;
    public Artikel naarArtikel() {
        return new Artikel(id);
    }
    // setters
}
```

## 23.10 VoorraadApplication

Voeg code toe:

```
@Bean
Jackson2JsonMessageConverter converter() {
    return new Jackson2JsonMessageConverter();
}
```

❶

- (1) Het bericht dat je ontvangt bevat JSON data.  
Als je applicatie een Jackson2JsonMessageConverter bean bevat, converteert Spring met deze bean de JSON data naar een Java object. Spring biedt je dit object aan (zie hieronder), zodat je het bericht kan lezen.



### 23.11 ArtikelListener

```
package be.vdab.voorraad.messaging;
// enkele imports
@Component
class ArtikelListener {
    private final ArtikelService service;
    // constructor met parameters
    @RabbitListener(queues = "voorraad")
    void verwerkBericht(ArtikelGemaakt gemaakt) {
        service.create(gemaakt.naarArtikel());
    }
}
```

❶  
❷  
❸  
❹

- (1) Je ontvangt met deze class berichten van RabbitMQ.
- (2) Je typt `@RabbitListener` voor de method bij ❸. Je vermeldt de queue voorraad. Spring stuurt dan de berichten uit die queue één per één naar de method bij ❹.
- (3) Je geeft de method een parameter. Die stelt een bericht voor dat je ontvangt. Spring converteert de JSON data in het bericht naar een `ArtikelGemaakt` object en vult hiermee de parameter.
- (4) Je maakt met het bericht een `Artikel` object. Je voegt dit object toe aan de database.

### 23.12 Uitvoeren

Start de applicatie.

De table `artikels` bevat een nieuw record.

Kijk in RabbitMQ in de queue voorraad. Die bevat geen berichten.

Start de applicaties `catalogus` en `voorraad` tegelijk. Stuur POST requests naar `catalogus`.

Per request bevat de table `artikels` in de databases `catalogus` en `voorraad` een nieuw record.

### 23.13 MicroServices

Je hebt de verwerking van de artikels verdeeld over twee applicaties: `Catalogus` en `Voorraad`. Elke applicatie is klein, heeft zijn eigen database en communiceert met andere applicaties via REST en messaging. Zo'n applicatie heet een microservice.

Als je de verwerking van de artikels in één applicatie plaats, heet zo'n applicatie een monolith.

Een microservice architectuur is dus een andere keuze dan een monolith architectuur.



Commit de sources. Publiceer op je remote repository.



Sportmailing

## 24 OUTBOX

Open het project catalogus.

Je doet in de class `DefaultArtikelService` in de method `create` twee handelingen:

- Een record toevoegen.
- Een bericht versturen.

Er is een kans dat je het bericht verstuurt terwijl je geen record aan de database toevoegde:

1. Je voegt het record toe.
2. Je verstuurt het bericht.
3. Je programma valt juist daarna uit.

Spring heeft geen commit kunnen doen op de transactie waarin de method `create` loopt.

De database doet een rollback. Het record is niet toegevoegd.

Ondertussen is het bericht wel verstuurd.

De applicatie voorraad ontvangt het bericht en voegt een record toe in zijn database.

Je hebt een fout: de database catalogus bevat artikel niet, de database voorraad wel.

Hiervoor bestaat een oplossing: het outbox pattern. Je past het hier toe. Het bevat twee stappen.

### 24.1 Stap 1

Je stuurt in de class `DefaultArtikelService` in de method `create` geen bericht, maar je voegt een record toe aan een nieuwe table `artikelsgemaakt`.

Records in deze table stellen berichten voor die je *later* stuurt naar RabbitMQ.

#### 24.1.1 Database

Voer het script `artikelsgemaakt.sql` uit.

Dit maakt in de database catalogs een table `artikelsgemaakt`:



#### 24.1.2 ArtikelGemaakt

```
package be.vdab.catalogus.events;
// enkele imports
@Entity
@Table(name = "artikelsgemaakt")
public class ArtikelGemaakt {
    @Id
    private long id;
    private String naam;
    public ArtikelGemaakt(Artikel artikel) {
        id = artikel.getId();
        naam = artikel.getNaam();
    }
    // protected default constructor
    // getters
}
```

#### 24.1.3 ArtikelGemaaktRepository

```
package be.vdab.catalogus.repositories;
// enkele imports
public interface ArtikelGemaaktRepository
    extends JpaRepository<ArtikelGemaakt, Long> {
}
```

#### 24.1.4 DefaultArtikelService

```
package be.vdab.catalogus.services;
// enkele imports
@Service
@Transactional
class DefaultArtikelService implements ArtikelService {
    private final ArtikelRepository artikelRepository;
    private final ArtikelGemaaktRepository artikelGemaaktRepository;
    // constructor met parameters
    @Override
    public void create(Artikel artikel) {
        artikelRepository.save(artikel);
        artikelGemaaktRepository.save(new ArtikelGemaakt(artikel));
    }
}
```

①  
②

(1) Je voegt een record toe aan de table artikels

(2) Je voegt een record toe aan de table artikelsgemaakt.

Spring zal ofwel beide records toevoegen ofwel geen enkel record toevoegen, want beide bewerkingen behoren tot dezelfde transactie.

Start de applicatie. Stuurt POST requests naar catalogus.

Per request bevatten de tables artikels en artikelsgemaakt een nieuw record.

### 24.2 Stap 2

Je leest op regelmatige tijdstippen de records in de table artikelsgemaakt.

Je verstuurt per record een bericht naar RabbitMQ.

Je verwijdert daarna de gelezen records.

#### 24.2.1 @EnableScheduling

Voeg een regel toe voor de class CatalogusApplication:

```
@EnableScheduling
```

①

Met deze regel kan je in je applicatie op regelmatige tijdstippen code uitvoeren.

#### 24.2.2 MessageSender

```
package be.vdab.catalogus.messaging;
// enkele imports
@Component
class MessageSender {
    private final ArtikelGemaaktRepository repository;
    private final AmqpTemplate template;
    // constructor met parameters
    @Scheduled(fixedDelay = 5_000)
    @Transactional
    void verstuurMessages() {
        var artikelsGemaakt = repository.findAll();
        for (ArtikelGemaakt gemaakt : artikelsGemaakt) {
            template.convertAndSend("catalogus", null, gemaakt);
        }
        repository.deleteAll(artikelsGemaakt);
    }
}
```

①  
②  
③  
④  
⑤

(1) Je voert de method bij ② uit, Je wacht 5000 milliseconden, je voert de method opnieuw uit, ...

(2) Dit is de method die je regelmatig uitvoert.

(3) Je leest de records in de table artikelsgemaakt.

(4) Je verstuurt per record een bericht naar de exchange artikels in RabbitMQ.

(5) Je verwijdert de gelezen records.

Start de applicatie.

Je ziet in de management van RabbitMQ nieuwe berichten in de queue voorraad.

De table artikelsaangemaakt is leeg.

Deze code kan een bericht wel *meerdere* keren versturen:

Spring voert de code in de method `verstuurMessages` uit.

Je verstuurt daar de berichten.

Juist voor je de records verwijdert, valt je applicatie uit.

Je herstart de applicatie.



Spring voert de code in de method `verstuurMessages` terug uit.

Je leest de records die je de vorige keer niet schrapte. Je verstuurt dezelfde berichten.

De voorraad applicatie moet er dus op voorzien zijn hetzelfde bericht *meerdere keren* te ontvangen. Dit is zo. Je roept daar in `DefaultArtikelService` de method `save` van `ArtikelRepository` op. De method `save` (die je erft van `JpaRepository`) zoekt eerst of het artikel al bestaat in de database.

Enkel als dit niet het geval is, voegt de method een record toe.

## 24.3 Docker

Je kan in de productie omgeving volgende containers hebben:

- Een container met de applicatie catalogus.
- Een container met MySQL met de database catalogus.
- Een container met RabbitMQ.
- Een container met de applicatie voorraad.
- Een container met MySQL met de database voorraad.



Commit de sources. Publiceer op je remote repository.



Outbox:

## 25 TESTCONTAINERS

Als je de repository layer test, kan de database die je aanspreekt al records bevatten of kunnen andere applicaties tijdens je test records toevoegen, wijzigen of verwijderen.

Je test kan zo verkeerde informatie lezen uit de database.

Een oplossing is de library Testcontainers. Die maakt voor je test een Docker container en voert daarin de database uit. Je test heeft dus de database exclusief voor zich.

Testcontainers verwijdert de container na je test.

### 25.1 Project

Je maakt het project in IntelliJ:

8. Kiest in het menu File de opdracht New, Project.
9. Kies links Spring Initializr. Kies Next.
10. Typ be.vdab bij Group.
11. Typ testcontainers bij Artifact.
12. Kies 11 bij Java Version. Kies Next.
13. Voeg dependencies toe:

Categorie	Dependency
SQL	Spring Data JPA , MySQL Driver

14. Kies Next, Finish.

Maak van het project een GIT project. Maak een bijbehorende remote repository.

### 25.2 pom.xml

Voeg de Testcontainers dependencies toe:

```
<dependency>
  <groupId>org.testcontainers</groupId>
  <artifactId>mysql</artifactId>
  <version>1.14.3</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.testcontainers</groupId>
  <artifactId>junit-jupiter</artifactId>
  <version>1.14.3</version>
  <scope>test</scope>
</dependency>
```

Klik rechts op het verticale tabblad Maven. Klik daar op .

### 25.3 application.properties

```
spring.datasource.url=jdbc:mysql://localhost/eendatabase
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.test.database.replace=none
spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED
spring.jpa.generate-ddl=true
```

- (1) Dit is de (fictieve) JDBC URL van de productiedatabase. Je gebruikt die niet in je tests.
- (2) Testcontainers maakt de database in een container. De database bevat geen tables. Je tests verwijzen echter naar tables. Je lost hier op. JPA maakt nu de juiste tables in de database. JPA maakt bijvoorbeeld een table personen omdat hij voor de entity class `Persoon` `@Table(name="personen")` ziet. JPA maakt in die table een kolom `id` omdat de class `Persoon` een private variabele `id` bevat. JPA maakt een primary key gebaseerd op de kolom `id`, omdat voor de private variabele `id` `@Id` staat, ...

## 25.4 Persoon

```
package be.vdab.testcontainers.domain;
// enkele imports
@Entity
@Table(name = "personen")
public class Persoon {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String voornaam;
    private String familienaam;
    // getters
}
```

## 25.5 PersoonRepository

```
package be.vdab.testcontainers.repositories;
// enkele imports
public interface PersoonRepository extends JpaRepository<Persoon, Long> {
}
```

## 25.6 insertPersonen.sql

```
insert into personen(voornaam, familienaam) values ('Joe', 'Dalton');
```

## 25.7 spring.properties

```
spring.test.constructor.autowire.mode=all
```

## 25.8 PersoonRepositoryTest

```
package be.vdab.testcontainers;
// enkele imports
@Testcontainers
@DataJpaTest
@Sql("/insertPersonen.sql")
class PersoonRepositoryTest
    extends AbstractTransactionalJUnit4SpringContextTests {
    private final PersoonRepository repository;
    @Container
    private static final MySQLContainer<?> mySQL =
        new MySQLContainer<>("mysql:latest")
            .withDatabaseName("testcontainers")
            .withUsername("cursist")
            .withPassword("cursist");
    // constructor met parameter
    @DynamicPropertySource
    private static void source(DynamicPropertyRegistry registry) {
        registry.add("spring.datasource.url", mySQL::getJdbcUrl);
    }
    @Test
    void erIs1Persoon() {
        assertThat(repository.count()).isOne();
    }
    @Test
    void dePersoonIsJoeDalton() {
        var eerstePersoon = repository.findAll().get(0);
        assertThat(eerstePersoon.getVoornaam()).isEqualTo("Joe");
        assertThat(eerstePersoon.getFamilienaam()).isEqualTo("Dalton");
    }
}
```

- (1) Je typt `@Testcontainers` voor een test waarin je `Testcontainers` gebruikt.
- (2) Je typt `@Container` voor een variabele die een container van `Testcontainers` voorstelt.

- (3) MySQLContainer stelt het MySQL programma in een container voor.  
Zodra je zo'n variabele maakt, maakt Testcontainer een container met daarin MySQL.  
Zo'n container maken vraagt wat tijd. Je maakt daarom de variabele static.  
Testcontainers maakt de container dan maar één keer voor alle tests in deze class.
- (4) Standaard gebruikt Testcontainers een image met een oudere versie van MySQL.  
`mysql:latest` duidt aan dat je een image met de recentste MySQL gebruikt.  
Als je PC deze image niet bevat, downloadt Testcontainer die vanaf docker hub.
- (5) Je definieert de naam van de database die Testcontainers in de MySQL moet maken.
- (6) Je definieert de naam van de MySQL gebruiker die Testcontainers moet maken.
- (7) Je definieert het bijbehorende paswoord.
- (8) Je overschrijft met de method bij ⑧ instellingen uit `application.properties`.
- (9) De method moet static void zijn  
en moet een `DynamicPropertyRegistry` parameter hebben.
- (10) Spring leest de JDBC URL nu niet meer uit de instelling `spring.datasource.url`  
in `application.properties` maar gebruikt de JDBC URL van de database  
die Testcontainers in zijn container maakte.

Voer de test uit. Dit zal lang duren. De test lukt.



Commit de sources. Publiceer op je remote repository.

## **26 COLOFON**

<b>Domeinexpertisemanager:</b>	Jean Smits
<b>Moduleverantwoordelijke:</b>	Hans Desmet
<b>Medewerkers:</b>	Hans Desmet
<b>Versie:</b>	2/7/2020
<b>Nummer dotatielijst:</b>	