



Spring Fundamentals

Deze cursus is eigendom van de VDAB©

Inhoudsopgave

1	INLEIDING.....	12
1.1	Doelstelling.....	12
1.2	Vereiste voorkennis.....	12
1.3	Nodige software	12
1.4	Opmerkingen.....	12
2	BROWSER, WEBSERVER, WEBSITE, URL, WEB FARM	13
2.1	HTTP – TCP/IP	13
2.2	Port number	13
2.3	URL	13
2.3.1	Betekenisvolle URI	14
2.4	Web farm.....	14
2.5	Webservers	14
3	TOMCAT	15
3.1	Download	15
3.2	Installatie	15
3.2.1	Unzippen.....	15
3.2.2	JAVA_HOME.....	15
3.3	Start	15
3.4	Stop	15
3.5	Installatie website	16
3.5.1	WAR bestand	16
3.5.2	Gebruikers	16
3.5.3	Installatie via browser.....	16
3.5.4	Installatie via bestandsbeheer	16
4	REQUEST - RESPONSE.....	17
4.1	Method.....	17
4.2	GET request	17
4.3	POST request.....	18

4.4	Verkeerd gebruik.....	18
4.5	Samenvatting.....	18
5	CONTAINER – EMBEDDED	19
5.1	Webserver als container	19
5.2	Embedded webserver	19
5.3	War - Jar	19
6	SPRING BOOT	20
6.1	IntelliJ Ultimate	20
6.2	start.spring.io	20
6.3	Project	21
6.3.1	Package.....	21
6.4	JUnit en AssertJ	21
7	STATIC CONTENT	22
7.1	Welkom pagina.....	22
7.2	CSS.....	22
7.3	Afbeelding	22
8	UITVOEREN.....	23
8.1	Starten	23
8.2	DevTools.....	23
8.3	Stoppen	23
9	BEAN	24
9.1	Algemeen	24
9.2	Annotations	24
9.3	Singleton.....	24
9.4	Voorbeeld.....	24
10	CONTROLLER	25
10.1	IndexController.....	25

10.2	Overzicht	25
10.3	Uitvoeren.....	25
10.4	Meerdere controllers	25
10.5	Nadelen	26
10.6	Overzicht	26
11	THYMELEAF	27
11.1	Voorbeeld.....	27
11.2	Controller	28
11.3	Overzicht	28
11.4	Data doorgeven.....	28
12	<code>\${VARIABLE EXPRESSION}</code>	29
12.1	Basis voorbeeld	29
12.2	Rekenen.....	29
12.3	Vergelijken.....	29
12.4	not, and, or.....	29
12.5	Verkorte if met <code>?:</code>	29
12.6	Vaste tekst en data combineren	29
12.7	Object	29
13	GETTER	30
13.1	Getters.....	30
13.2	Constructor.....	30
13.3	Immutable	30
13.4	Final	30
13.5	Genest object	31
13.6	Getter gebaseerd op een berekening	31
13.7	Controller	31
13.8	Thymeleaf.....	31

13.9	Selection expression.....	32
13.10	Genest attribuut.....	32
14	TH:EACH	33
14.1	Controller	33
14.2	Thymeleaf.....	33
14.3	Verzameling objecten.....	34
14.3.1	Pizza	34
14.3.2	Controller	34
14.3.3	pizzas.html	34
14.4	List, Set	34
14.5	Stream	35
14.6	Map	35
15	TH:IF.....	36
16	TH:BLOCK	37
16.1	Voorbeeld 1	37
16.2	Voorbeeld 2	37
17	@{URL EXPRESSION}.....	38
17.1	URL expression	38
17.2	Nut van @{...}	39
17.3	Thymeleaf expressie.....	39
18	FRAGMENT	40
18.1	Maken.....	40
18.2	Gebruiken	40
18.3	Parameter.....	40
18.3.1	Maken	41
18.3.2	Gebruiken	41
18.4	pizzaTabel	41
19	PATH VARIABLE	42
19.1	URI template	42

19.2	Controller	42
19.3	pizza.html	43
19.4	Meerdere path variabelen	43
20	@REQUESTMAPPING	44
21	NAAM WAARONDER JE DATA DOORGEeft AAN THYMELEAF	45
22	MODEL-VIEW-CONTROLLER	46
23	REQUEST HEADER	47
23.1	Controller	47
23.2	os.html.....	47
24	COOKIE	48
24.1	Voorbeeld.....	48
24.1.1	KleurController	48
24.1.2	kleuren.html	49
25	MULTITHREADING	50
25.1	Thread safe.....	50
25.2	Voorbeeld.....	50
26	STATELESS	51
26.1	PizzaController	51
26.2	pizzasperprijs.html	51
26.3	PizzaController	51
26.4	pizzasperprijs.html	52
26.5	Proberen.....	52
26.6	Oplossing	52
26.7	Korte code	52
26.8	Voordelen van stateless	52
27	ENTERPRISE APPLICATION.....	53
27.1	Lagen (layers)	53

27.2	Repository	53
27.3	Rest client.....	53
27.4	Service	54
27.4.1	Voorbeeld	54
27.5	Rest service.....	55
27.6	Samenwerking.....	55
27.6.1	Ander overzicht.....	55
27.6.2	Voorbeeld	55
28	DEPENDENCY INJECTION.....	56
28.1	Algemeen	56
28.2	FixerKoersClient	56
28.2.1	KoersClientException	56
28.2.2	FixerKoersClient	57
28.2.3	FixerKoersClientTest	57
28.3	EuroService.....	58
28.4	EuroServiceTest.....	59
28.5	PizzaController	60
28.6	pizza.html	61
28.7	Bean.....	61
28.8	Samenvatting.....	61
29	MEERDERE IMPLEMENTATIES	62
29.1	ECBKoersClient	62
29.2	ECBKoersClientTest	63
29.3	Dependency injection via een interface.....	63
29.3.1	Stap 1: FixerKoersClient implementeer een interface KoersClient	63
29.3.2	Stap 2: ECBKoersClient implementeer ook de interface KoersClient.....	64
29.4	@Primary.....	64
29.5	@Qualifier	64
29.5.1	@Qualifier bij de bean classes	64
29.5.2	@Qualifier bij constructor injection	64
29.6	Alle dependencies injecteren.....	65
29.6.1	EuroService	65
29.6.2	EuroServiceTest	65
29.7	Package visibility	65

29.8	Samenvatting.....	66
29.8.1	Je hebt van een dependency meerdere implementaties.	66
29.8.2	Je hebt van een dependency slechts één implementatie	66
29.9	IOC – IOC container	66
29.10	Als een dependency ontbreekt	66
29.11	@Controller, @Service, @Component, @Repository	66
30	APPLICATION.PROPERTIES	67
30.1	application.properties	67
30.2	FixerKoersClient	67
30.3	ECBKoersClient	67
31	BEAN TESTEN.....	68
31.1	spring.properties	68
31.2	FixerKoersClientTest.....	68
31.3	ECBKoersClientTest	68
32	LOGGING	69
32.1	Gelogde data	69
32.2	Naar een bestand loggen	69
33	DATABASE	70
33.1	MySQL Workbench.....	70
33.2	IntelliJ	70
33.2.1	Verbinding maken.....	70
33.2.2	Database structuur	70
33.2.3	SQL statement uitvoeren	70
33.2.4	Andere database merken	70
34	DATASOURCE.....	71
34.1	pom.xml.....	71
34.2	application.properties	71
34.3	DataSource	71
34.4	Test	72
34.4.1	application.properties	72
34.4.2	DataSourceTest.....	72

35	EXCEPTION BIJ DATABASE TOEGANG	73
35.1	Belangrijkste Spring verhelpbare database exceptions	73
35.2	Belangrijkste Spring fatale database exceptions.....	73
35.3	PizzaNietGevondenException	73
36	REPOSITORY	74
36.1	JdbcTemplate	74
36.2	PizzaRepository	74
36.3	Scalar value.....	75
36.4	Update of delete SQL statement.....	75
36.5	Meerdere parameters	75
36.6	Record toevoegen	76
36.7	RowMapper	76
36.8	Meerdere records lezen	77
36.9	Één record lezen.....	77
36.10	Resterende methods	78
36.11	Algemeen	79
37	REPOSITORY TEST	80
37.1	Transactie	80
37.2	Test records.....	80
37.3	PizzaRepositoryTest	81
38	SQL STATEMENTS LOGGEN.....	84
39	SERVICE EN TRANSACTIE	85
39.1	PizzaService	85
39.2	Test.....	86
39.3	Transactie	86
39.3.1	Isolation level.....	86
39.3.2	Read-only	87
39.3.3	Timeout.....	87
39.4	@Transactional	87

39.5	Propagation	88
39.6	Controller	90
39.7	Samenwerking	90
39.8	Database bewerkingen die één geheel vormen in een transactie	91
39.8.1	Voorbeeld 1	91
39.8.2	Voorbeeld 2	92
40	DTO	93
40.1	Repository	93
40.2	Service	94
40.3	Controller	94
40.4	Thymeleaf	94
41	FORM	95
41.1	Form class of record	95
41.2	Form tonen	96
41.2.1	PizzaController	96
41.2.2	vantotprijs.html	96
41.3	Leeg invoervak	96
41.4	Query string	97
41.5	Gesubmitte form verwerken	97
41.6	Validatie	97
41.6.1	vantotprijs.html	98
41.6.2	messages.properties	98
42	BEAN VALIDATION	100
42.1	Annotation	100
42.2	Andere validation annotations	100
42.3	@Valid	101
42.4	messages.properties	101
42.5	Form object	102
42.6	@Valid in de controller	102
43	CLIENT SIDED VALIDATIE	103

43.1	vantotprijs.html	103
43.2	Server sided validatie testen	103
44	POST REQUEST.....	104
44.1	PizzaController	104
44.2	toevoegen.html	104
44.3	Proberen.....	105
44.4	PizzaController	105
44.5	POST-REDIRECT-GET	105
44.5.1	Chrome	105
44.5.2	Oplossing	106
44.5.3	PizzaController.....	106
44.5.4	Proberen	106
44.6	RedirectAttributes.....	107
44.6.1	PizzaController.....	107
44.6.2	pizzas.html.....	107
44.7	Dubbele submit vermijden.....	107
44.7.1	Proberen	107
44.7.2	JavaScript.....	108
44.8	Overzicht	108
45	SESSION.....	109
45.1	Session.....	109
45.2	Serializable	109
45.3	Session persistence	109
45.4	Session replication.....	109
45.5	Kleine data.....	110
45.6	Session identificatie.....	110
45.7	Webserver verwijdert een session.....	110
45.8	Voorbeeld 1	111
45.8.1	Identificatie.....	111
45.8.2	IdentificatieController.....	111
45.8.3	identificatie.html	112
45.9	Voorbeeld 2	112
45.9.1	pizza.html.....	112
45.9.2	Mandje.....	112
45.9.3	MandjeTest	112

45.9.4	MandjeController	113
45.9.5	mandje.html	113
45.10	Session fixation.....	114
46	@CONTROLLERADVICE	115
47	OPMAAK	116
47.1	Getal	116
47.1.1	@NumberFormat.....	116
47.1.2	Thymeleaf	116
47.2	Datum en tijd.....	117
48	CUSTOM ERROR PAGE	118
48.1	404.....	118
48.2	500.....	118
49	PRODUCTIE.....	119
49.1	Website met een embedded Tomcat.....	119
49.2	Website op aparte webserver	120
49.2.1	SpringBootServletInitializer	120
49.2.2	WAR	120
49.2.3	Embedded Tomcat weglaten uit WAR	120
49.2.4	WAR maken	120
49.2.5	WAR installeren op de aparte webserver.	120
50	STAPPENPLAN	121
51	HERHALINGSTAKEN	123
52	COLOFON.....	124

1 INLEIDING

1.1 Doelstelling

Je leert werken met Spring:

een open source Java framework waarmee je enterprise applicaties maakt.

Spring bevat veel libraries. Enkele voorbeelden:

- Spring MVC om een website te maken.
- Spring JDBC om de database aan te spreken.

Bij een website waarin je een database aanspreekt, gebruik je Spring MVC en Spring JDBC.

Spring bevat nog veel libraries. Je leert sommige in de cursus Spring advanced.

De ontwikkelaars van Spring maken niet alle libraries *zelf*. Voorbeeld: de library Thymeleaf.

Je stuurt met Thymeleaf HTML naar de browser. Thymeleaf is niet gemaakt door de ontwikkelaars van Spring. Ze zorgden er *wel* voor dat je Thymeleaf gemakkelijk kan gebruiken in je Spring website.

1.2 Vereiste voorkennis

- Java
- Maven
- JDBC
- JUnit

1.3 Nodige software

- JDK (minstens versie 17).
- MySQL (minstens versie 8.0.14)
- IntelliJ Ultimate
- Chrome browser

1.4 Opmerkingen



Je ziet in de cursus regelmatig *Je publiceert op je remote repository*. We bedoelen daarmee je repository op GitHub (of Bitbucket, ...).



We zullen in de cursus regelmatig een waarde die gemaakt wordt, en wat verder (eventueel in een andere source) gebruikt wordt op de twee plaatsen dezelfde **achtergrondkleur** geven, om te verduidelijken dat het over dezelfde waarde gaat.

2 BROWSER, WEBSERVER, WEBSITE, URL, WEB FARM

De computers met browsers en de computers met webserverns zijn via het internet of intranet (bedrijfsnetwerk) met elkaar verbonden:



2.1 HTTP – TCP/IP

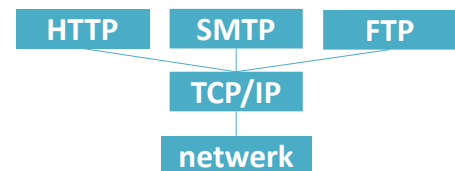
Browsers en webserverns wisselen data uit met het protocol HTTP (HyperText Transfer Protocol).

HTTP spreekt zelf het netwerk aan met software met de naam TCP/IP:

Transmission Control Protocol / Internet Protocol.

Andere protocollen gebruiken ook TCP/IP:

- SMTP Simple Mail Transfer Protocol (om mails te versturen).
- FTP File Transfer Protocol (om bestanden uit te wisselen).
- ...



2.2 Port number

Meerdere programma's kunnen op één computer TCP/IP gebruiken. Elk programma krijgt daarbij een uniek identificatie getal: het port number.

Meest gebruikte port numbers:

- 80 Webserver
- 21 FTP (File Transfer Protocol) server
- 25 Mail server

2.3 URL

Een webserver bevat één of meerdere websites. Een website bevat pagina's.

Elke pagina heeft een unieke identificatie: de URL (Uniform Resource Locator).

Een URL heeft volgende opbouw:

<http://pizzaluigi.be/pizzas>



- Naast HTTP bestaat ook HTTPS. Bij HTTPS versleutelen de browser en webserver de data die ze uitwisselen. Zo kan een hacker deze data niet lezen.
- Alle pagina's van een website delen dezelfde domeinnaam.
- Een domeinnaam is niet hoofdlettergevoelig, een pad wel.
- Je kan surfen naar een URL zonder pad (voorbeeld: pizzaluigi.be).
Je ziet dan de 'welkompagina' van die website.
- Als een webserver afwijkt van het standaard TCP/IP port number (80), vermeld je het port number. Voorbeeld: pizzaluigi.be:8080/pizzas.



Een URI (Universal Resource Identifier) is een identifier voor een stukje data.

Elke URL is een URI. Niet elke URI is een URL.

Een URI is een URL als de URI een protocol gebruikt (zoals http).

Voorbeeld: een ISBN (uniek boeknummer) is een URI maar geen URL.

2.3.1 Betekenisvolle URI

Een URL moet voor browsers en webservern geen menselijke betekenis hebben.

De URL <http://pizzaluigi.be/X12y37> kan verwijzen naar een pagina met de pizza's van Luigi.

URL's hebben meestal wel een menselijke betekenis. Voorbeelden:

- <http://pizzaluigi.be/pizzas> Overzicht van alle pizza's van Luigi.
- <http://pizzaluigi.be/pizzas/3> Detail van pizza 3.
- <http://pizzaluigi.be/pizzas/inpromotie> Overzicht van de pizza's die in promotie zijn.

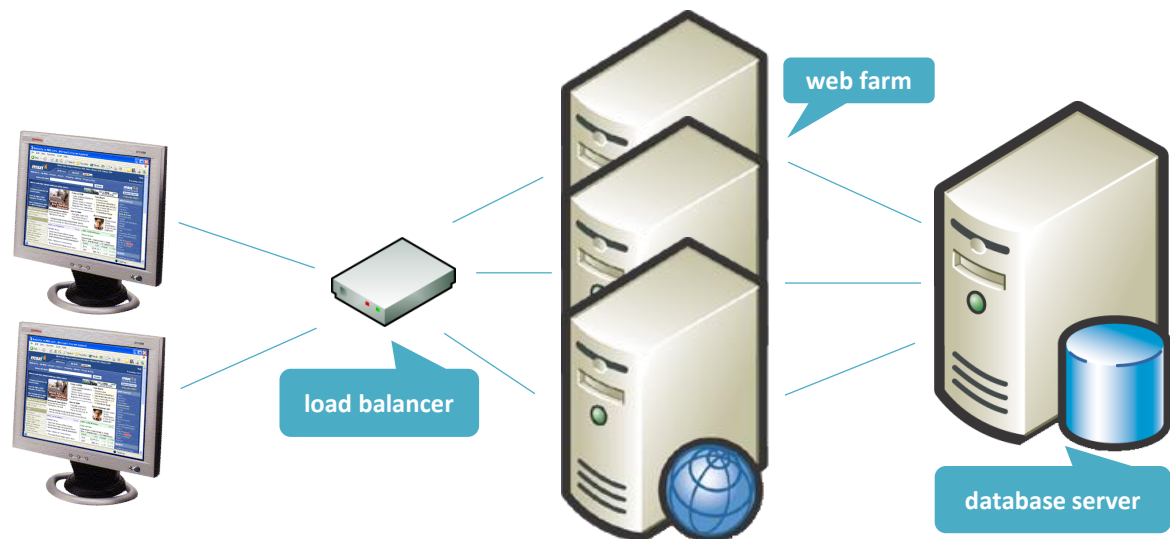
2.4 Web farm

Een website draait meestal op *meerdere* webservern.

Zo'n groep webservern heet een web farm of een cluster.

Een load balancer is een apparaat (of software op een server) in een web farm.

Hij verdeelt het werk (de pagina's leveren die de browsers vragen) over die webservern.



Voordelen:

- Als een webserver uitvalt, verdeelt de load balancer het werk over de andere webservern. De website blijft zo ter beschikking van de browsers.
- De load balancer verdeelt de browser vragen gelijkmatig over de webservern. Zo raakt één webserver niet overbelast. De website blijft performant.

2.5 Webservern

Veel webservern ondersteunen Java:

Webserver	Van de firma
Tomcat	Apache
Jetty	Eclipse
WildFly	JBoss
WebLogic	Oracle
WebSphere	IBM

Je gebruikt in de cursus de populairste Java webserver: Tomcat.

3 TOMCAT

3.1 Download

Download Tomcat.

1. Surf naar <http://tomcat.apache.org>
2. Klik links in het onderdeel Download op de hyperlink Tomcat 9.
3. Kies bij Binary Distributions voor Core.
4. Kies daar de hyperlink zip (...).



3.2 Installatie

3.2.1 Unzippen

Pak het ZIP bestand uit in een directory op je computer.

Kies geen subdirectory van \Program Files of \Program Files (x86).

Je kan dan het configuratiebestand van Tomcat niet aanpassen.

3.2.2 JAVA_HOME

Maak een environment variabele met de naam JAVA_HOME.

Die bevat het absolute pad van de directory waarin Java geïnstalleerd is.

Doe dit op Windows als volgt:

1. Open het Control Panel.
2. Kies System.
3. Kies Advanced system settings.
4. Kies het tabblad Advanced.
5. Kies Environment Variables.
6. Kies New onder User variables for ...
7. Typ JAVA_HOME bij Variable name.
8. Typ het absolute pad naar de directory waarin Java geïnstalleerd is (bijvoorbeeld C:\Program Files\AdoptOpenJDK\jdk-11.0.6.10-hotspot) bij Variable value.
9. Kies drie keer OK.

3.3 Start

Dubbelklik startup.bat in de subdirectory bin van de Tomcat Directory. Je ziet

- een Command-Prompt venster waarin Tomcat opstart.
- enkele diagnostische (informatieve) meldingen tijdens het starten.
- als laatste melding INFO: Server startup in x ms.

Tomcat is gestart. Laat dit venster openstaan. Anders stop je Tomcat.

Tomcat gebruikt TCP poort 8080.

localhost is in een netwerk het synoniem voor je eigen computer.

Typ in de browser adresbalk dus localhost:8080. Je ziet de Tomcat welkompagina.

3.4 Stop

1. Druk Ctrl+C in het Command-prompt venster waarin Tomcat draait of.
2. Dubbelklik shutdown.bat in de bin directory van Tomcat.
3. Het Command-Prompt venster met Tomcat verdwijnt na enkele seconden.

3.5 Installatie website

3.5.1 WAR bestand

Een WAR (web archive) is een bestand met de extensie war

Het heeft de structuur van een ZIP bestand.

Het bevat alle onderdelen van een Java website (code, afbeeldingen, CSS, ...).

3.5.2 Gebruikers

Enkel geregistreerde Tomcat gebruikers kunnen via de browser een website installeren.

Een gebruiker heeft één of meerdere rollen (roles).

Enkel gebruikers met de role manager-gui kunnen via de browser een website installeren.

Het bestand tomcat-users.xml in de Tomcat subdirectory conf bevat de gebruikers.

Maak in dit bestand een gebruiker met de role manager-gui:

1. Open dit bestand met IntelliJ: menu File, Open.
2. Maak een blanco regel juist onder de regel `<tomcat-users ...>`. Typ daarin `<user username="cursist" password="cursist" roles="manager-gui,manager-script"/>`
3. Sluit het tabblad waarin dit bestand geopend is. IntelliJ bewaart het bestand.
4. Herstart Tomcat.

3.5.3 Installatie via browser

1. Surf met een browser naar localhost:8080.
2. Kies de knop Manager App.
3. Typ als gebruikersnaam én als paswoord cursist.Log in.
4. Kies de knop naast Select WAR file to upload.
5. Duid sterrenbeelden.war (bestand bij cursus) aan op je harde schijf.
6. Kies de knop Deploy (to deploy = installeren).

Je ziet na enkele seconden /sterrenbeelden in de lijst van websites (Applications):

/sterrenbeelden	None specified		true	0	Start Stop Reload Undeploy	Expire sessions with idle ≥ 30 minutes
---------------------------------	----------------	--	------	---	----------------------------	--

7. Klik op de hyperlink /sterrenbeelden. Je ziet je website. Je kan die testen.
8. Stop de website met **Stop**. Probeer daarna naar de website te surfen.
Je ziet een pagina met status code 404 (Not found): de website is niet actief.
9. Start de website terug met **Start**.
10. Herstart de website met **Reload**.
11. Verwijder de website van Tomcat met **Undeploy**.
12. Een website kan data (zoals een winkelmandje) per gebruiker bijhouden in het RAM geheugen.
Je schrapt de data, als die 30 minuten niet gelezen of gewijzigd werd, met **Expire sessions**.

Verwijder de website: je installeert hem straks opnieuw via het bestandsbeheer.

3.5.4 Installatie via bestandsbeheer

Kopieer sterrenbeelden.war naar de Tomcat subdirectory webapps.

Tomcat installeert elke WAR in die directory automatisch als een website.

Surf naar de (hoofdlettergevoelige) URL van de website: localhost:8080/sterrenbeelden.

Verwijder de website door sterrenbeelden.war te verwijderen uit de directory webapps.



Laat Tomcat draaien. Je hebt hem nog nodig.

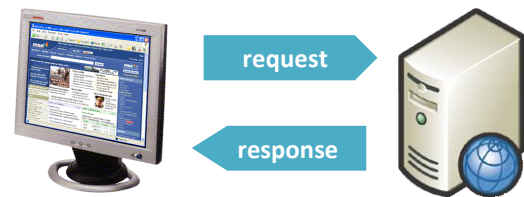


Seizoenen

4 REQUEST - RESPONSE

Telkens je

- een URL typt in de browser adresbalk
- een URL kiest in de browser favorieten
- een hyperlink aanklikt
- een knop aanklikt in een formulier



stuurt de browser een request (vraag) naar een webserver.

De browser krijgt een response (antwoord). Die bevat HTML, CSS, JavaScript en/of afbeelding(en).

4.1 Method

De request method definieert het *soort* request met één woord: GET of POST. HTTP schrijft voor:

GET	<p>Gebruik GET bij elke request die <i>enkel data vraagt</i>. Voorbeeld: een request naar <code>luigi.be/producten</code> vraagt producten. Een request heeft de method GET als de gebruiker</p> <ul style="list-style-type: none"> • een URL typt in de adresbalk van de browser en Enter drukt. • een hyperlink aanklikt. • een formulier submit waarvan de method op get staat.
POST	<p>Gebruik POST bij elke request die meer doet dan enkel data vragen. Voorbeeld: een request naar <code>luigi.be/producten/toevoegen</code> voegt een product toe. Een request heeft enkel de method POST als de gebruiker</p> <ul style="list-style-type: none"> • een formulier verstuurt waarvan de method op post staat.

4.2 GET request

Je ziet voorbeelden met de developer tools in Chrome.

1. Druk F12.
2. Kies het tabblad Network. Je ziet daar vanaf nu data over requests en responses.
3. Surf naar <http://localhost:8080/sterrenbeelden/>

Je ziet onder in het venster een request naar de welkompagina.

Je ziet daarna een request naar `default.css`, waar de welkompagina naar verwijst:

Name	Status
<code>sterrenbeelden/</code>	200
<code>default.css</code>	200

Klik met de rechtermuisknop in de kolomhoofding Name. Kies Method.

Surf nog eens naar de pagina.

Je ziet in een extra kolom dat het requests zijn met de method GET.

Dit is correct: de requests vragen enkel data.

Name	Method	Status
<code>sterrenbeelden/</code>	GET	200
<code>default.css</code>	GET	200

We noemen een request met de method GET vanaf nu een GET request.

De browser

- maakt bij elke request verbinding met de webserver.
- sluit die verbinding nadat hij de response van de webserver krijgt.
- maakt bij de volgende request een nieuwe verbinding met de webserver ...

4.3 POST request

Voorbeeld: Typ een naam en een bericht. Kies de knop Toevoegen.

Dit veroorzaakt een request met de method POST.

De request doet meer dan data vragen: hij voegt een item toe aan het gastenboek.

Name	Method	Status
<input type="checkbox"/> sterrenbeelden/	POST	302

Daarna volgen enkele GET requests die de pagina terug opvragen.

We noemen een request met de method POST vanaf nu een POST request.

4.4 Verkeerd gebruik

Je krijgt problemen als je GET en POST anders gebruikt dan HTTP het voorschrijft.

Fictief voorbeeld: een pagina toont een product.

De pagina bevat een hyperlink Verwijderen.

Als de je hierop klikt, verwijdert de website het product uit de database.

Naast mensen bezoeken ook zoekrobots de pagina.

Ze volgen elke hyperlink, hopen dat die hen naar interessante pagina's zal leiden.

Ze mogen dit ook: een hyperlink volgen is een GET request.

HTTP schrijft voor dat een GET request enkel data leest.

De hyperlink Verwijderen. (en de bijbehorende GET request) volgt HTTP niet.

Gevolg: de zoekrobot verwijdert het product bij het volgen van de hyperlink.

Als je HTTP volgt, vervang je de hyperlink Verwijderen door een knop Verwijderen.

Die is een onderdeel van een form met het attribuut method gelijk aan post.

Zoekrobots volgen geen POST requests: ze weten dat ze zo je website kunnen beschadigen.

4.5 Samenvatting

Telkens je een hyperlink of button van een website kiest, doet de browser het volgende:

1. Een verbinding naar de website maken.
2. Een request naar de website sturen (via die verbinding).
3. Wachten op de response.
4. Met de response het beeld aanpassen.
5. De verbinding sluiten.



Stop Tomcat. Zo is er geen conflict met een andere Tomcat die je straks gebruikt.



GET of POST

5 CONTAINER – EMBEDDED

5.1 Webserver als container

Een webserver kan meerdere websites bevatten.

Als een website slecht werkt, kan dit de andere websites op de webserver ook benadelen.

Om dit te verhinderen installeert men meestal één website op een webserver.

De webserver speelt de rol van container: hij bevat de website:



Je hebt nu volgende situatie: één Tomcat webserver met twee websites:



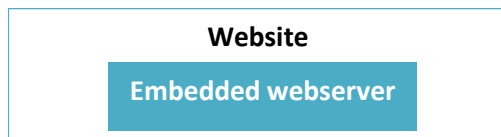
5.2 Embedded webserver

Men gaat in moderne omgevingen verder.

De webserver is een onderdeel van de website. Men spreekt dan over een embedded webserver.

Je hoeft dan niet een webserver én de website te installeren.

Het volstaat de website te installeren: hij bevat zijn eigen webserver in zich:



Je gebruikt verder in de cursus een embedded webserver.

5.3 War - Jar

Je kan een website verpakken in een WAR bestand of in een JAR bestand.

- Je kiest een WAR bestand als je de website wil installeren in een webserver die draait als een container (bovenste afbeelding).
- Anders verpak je de website in een JAR bestand (onderste afbeelding).

Je kan met enkele stappen een website, die oorspronkelijk verpakt was in een JAR bestand, verpakken in een WAR bestand. Je leert dit verder in de cursus.

6 SPRING BOOT

Voor Spring Boot bestond, had je veel werk als je een Spring project maakte:

- Je moest een Maven project maken.
- Je moest dependencies typen in `pom.xml`.
- Je moest veel initialisatiecode typen voor een website,
Die code was voor elke website dezelfde.

Spring Boot doet die stappen voor je. Je wint zo tijd.

6.1 IntelliJ Ultimate

Een Spring Boot project maken is ingebakken in IntelliJ Ultimate.

1. Kies het menu File, New, Project.
2. Kies links Spring Initializr.
3. Typ luigi bij Name.
Wijzig eventueel de Location.
Typ `be.vdab` bij Group.
Kies 17 bij Java.
Kies Next.
4. Je ziet een venster waar je de Maven dependencies kiest.
Ze zijn verdeeld in categorieën (Developer Tools, Web, ...).
Kies een dependency door een vinkje te plaatsen bij de dependency.

Categorie	Dependency	Waarom je de dependency nodig hebt
Developer Tools	Spring Boot DevTools	Sneller websites maken. Zonder DevTools moet je bij elke wijziging je website stoppen en volledig herstarten om het resultaat van de wijziging te zien. DevTools herstart snel enkel het gewijzigde deel.
Web	Spring Web	Een embedded Tomcat in je applicatie en de Spring library waarmee je websites maakt.
Template Engines	Thymeleaf	HTML naar de browser sturen.
I/O	Validation	Data in instance variabelen valideren.

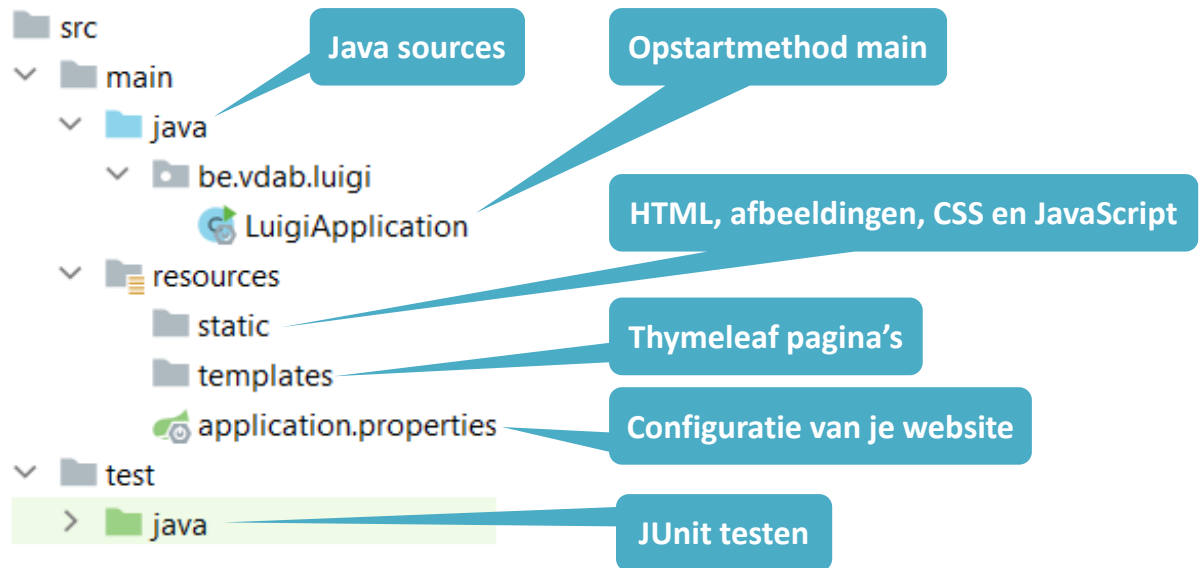
5. Kies Finish.

6.2 start.spring.io

Als je een andere IDE gebruikt dan IntelliJ Ultimate, maak je een Spring Boot project als volgt:

1. Surf naar <https://start.spring.io>.
2. Typ `be.vdab` bij Group.
3. Typ luigi bij Artifact.
4. Kies ADD DEPENDENCIES. Je definieert de Maven dependencies.
Typ één per één de naam van de dependencies. Druk na elke naam `Ctrl + Enter`.
`devtools`
`web`
`thymeleaf`
`validation`
5. Druk Enter.
6. Kies GENERATE.
Je krijgt een bestand `luigi.zip`.
Dit bestand bevat je Maven project.
Open dit in je IDE.

6.3 Project



Voor de class `LuigiApplication` staat `@SpringBootApplication`. Spring initialiseert zichzelf hiermee bij de start van je applicatie en start ook de embedded Tomcat.

6.3.1 Package

Je programma werkt enkel correct als je classes straks maakt in subpackages van de package (`be.vdab.luigi`) van de class `LuigiApplication`.

- Een correcte subpackage: `be.vdab.luigi.domain`.
- Een verkeerde subpackage: `org.myorg.domain`.

6.4 JUnit en AssertJ

Spring voegde aan je project ook een dependency toe met de naam `spring-boot-starter-test`. Die heeft zelf een transitive dependency naar JUnit en AssertJ.

Je project bevat zo de nodige libraries om testen te programmeren.

Je project bevat ook al het onderdeel `src/test/java` waarin je test classes programmeert.

Je zal dit regelmatig doen.



- Maak van het project een Git project.
- Commit de sources.
- Publiceer op je remote repository.



Project

7 STATIC CONTENT

Static content zijn statische HTML pagina's, CSS bestanden, JavaScript bestanden en afbeeldingen.

7.1 Welkom pagina

index.html is de welkom pagina van de website.

Maak de pagina:

1. Klik met de rechtermuisknop op static.
2. Kies New, HTML File.
3. Typ index. Druk Enter.

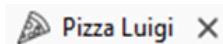


Wijzig de pagina:

```
<!doctype html>
<html lang="nl">
  <head>
    <title>Pizza Luigi</title>
    <link rel="icon" href="images/luigi.ico" type="image/x-icon">
    <link rel="stylesheet" href="css/luigi.css">
  </head>
  <body>
    <h1>Pizza Luigi</h1>
    
  </body>
</html>
```

1

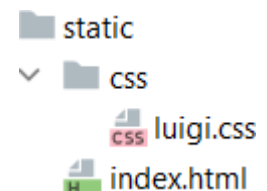
(1) Je toont de afbeelding luigi.ico in het tabblad boven in de browser:



7.2 CSS

Plaats de CSS bestanden in een aparte folder. Je project blijft zo overzichtelijk.

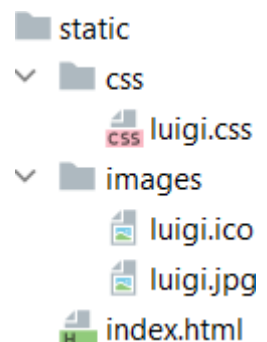
1. Klik met de rechtermuisknop op static.
2. Kies New, Directory.
3. Typ css. Druk Enter.
4. Klik in de Windows File Explorer met de rechtermuisknop op luigi.css (materiaal bij de cursus). Kies Copy.
5. Klik in IntelliJ met de rechtermuisknop op de folder css. Kies Paste, OK.



7.3 Afbeelding

Plaats de afbeeldingen in een aparte folder.

1. Klik met de rechtermuisknop op static.
2. Kies New, Directory.
3. Typ images. Druk Enter.
4. Kopieer in die folder: luigi.ico en luigi.jpg



Commit de sources. Publiceer op je remote repository.

8 UITVOEREN

8.1 Starten


1. Klik in het project overzicht met de rechtermuisknop op LuigiApplication.
2. Kies Run 'LuigiApplication'.
Je website start.
Je ziet in het venster Console enkele informatieve, diagnostische boodschappen.
Één boodschap bevat de tekst Tomcat started on port(s): 8080 (http)...
Je applicatie heeft dus een embedded Tomcat gestart. Die gebruikt TCP/IP poort 8080.
3. Surf in een browser naar je website: localhost:8080.
Je ziet de welkom pagina.

8.2 DevTools

Je hebt bij de aanmaak van je project de library DevTools toegevoegd.
Dankzij DevTools moet je je website niet herstarten om een wijziging in de browser te zien.
Probeer dit:

1. Wijzig in index.html Luigi naar Luiga.
2. Druk Ctrl+F9. Je build (compileert) zo het project.
3. Ververs de pagina in de browser.

8.3 Stoppen

Je maakt in de taken een andere website. Die zal ook TCP/IP poort 8080 gebruiken.
Stop daarom deze website: kies  (links naast het venster Console).



Welkom pagina

9 BEAN

9.1 Algemeen

Een bean is een Java object dat Spring maakt.

Omdat Spring zo'n object maakt en beheert, kan Spring handelingen op dit object doen, waarvoor jij weinig code moet typen in je applicatie.

Voorbeelden van handelingen die Spring op een bean kan doen:

- Dependency injection
- Database transactie starten, committen en rollbacken
- requests vanuit browsers verwerken en responses terugsturen.
- ...

Je leert dit verder in de cursus in detail kennen.

9.2 Annotations

Spring maakt een bean van een class als voor die class één van volgende annotations staat: `@Repository`, `@Service`, `@RestController`, `@Controller`, `@Component`, ...

Je leert de betekenis van deze annotations kennen in deze cursus.

9.3 Singleton

Beans zijn standaard *singleton* beans.

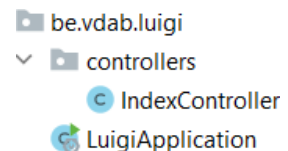
Spring overloopt bij de start van je applicatie je classes. Als voor een class één van de annotations (hier boven) staat, maakt Spring één (bean) object van die class. Spring houdt dit object in het RAM geheugen zolang de applicatie in uitvoering is. Dit lijkt op het singleton design pattern.

9.4 Voorbeeld

Je maakt een eerste eenvoudige class waarvan Spring een bean maakt.


De class bevat weinig code. Je maakt ze hier om de basiswerking van Spring te tonen bij het maken van een bean. De class krijgt meer code in de volgende hoofdstukken.

Maak een package `be.vdab.luigi.controllers`.
Maak daarin de class `IndexController`:



```
package be.vdab.luigi.controllers;
// enkele imports
@RestController
class IndexController {
    IndexController() {
        System.out.println("IndexController constructor");
    }
}
```

①
②
③
④

- (1) Je typt `@RestController` voor de class.
Spring maakt een bean (object) van die class wanneer de applicatie start.
IntelliJ toont dit met  in de marge.
- (2) De class moet niet public zijn. De class heeft hier package visibility. Dit betekent: de class is enkel zichtbaar in zijn eigen package (`be.vdab.luigi.controllers`). Het is een "best practice" om classes, die je niet vanuit andere packages moet aanspreken, package visibility te geven.
- (3) Dit is de constructor. Wanneer Spring de bean maakt zal Spring de constructor oproepen.
Ook de constructor heeft package visibility.
- (4) Je toont een informatieve boodschap wanneer de constructor wordt opgeroepen.

Start je applicatie. Je ziet in het venster Console nu ook `IndexController constructor`, op het moment dat Spring de bean maakt.

10 CONTROLLER

Verwijder `index.html`. Je vervangt dit door een controller.

Een controller is een Java object. Het krijgt requests binnen en stuurt responses.

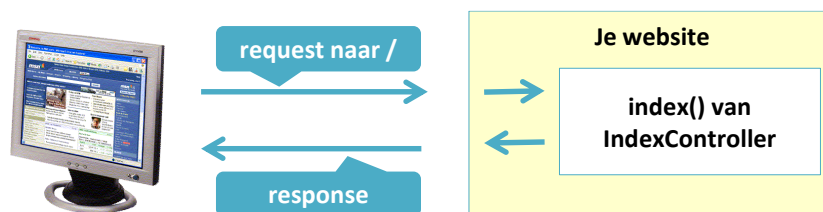
10.1 IndexController

Wijzig de class. Je mag daarbij de constructor verwijderen.

```
package be.vdab.luigi.controllers;
// enkele imports.
@RestController
class IndexController {
    @GetMapping("/")
    public String index() {
        var morgenOfMiddag = LocalTime.now().getHour() < 12 ? "morgen" : "middag";
        return "<!doctype html><html><title>Hallo</title><body>Goede"
            + morgenOfMiddag +
            "</body></html>";
    }
}
```

- (1) `@GetMapping("/")` geeft aan dat de method bij ② GET requests verwerkt naar de URL `/`. `/` staat voor de welkom pagina.
Telkens een browser een GET request doet naar die URL, roept Spring de method bij ② op.
- (2) Je mag de naam van de method vrij kiezen. De method geeft een String terug.
Spring stuurt die als response naar de browser.
- (3) De String bevat HTML met daarin de tekst morgen of middag.

10.2 Overzicht



10.3 Uitvoeren

1. Spring overloopt bij de start van je website je classes. Dit heet component scanning.
2. Als Spring een class vindt waarvoor `@RestController` staat, maakt Spring van die class een object (een Spring bean).
3. Spring configureert de embedded Tomcat: als een request binnenkomt naar de URL `/`, stuurt Tomcat die naar de method `index` van `IndexController`.
Spring baseert dit op `@GetMapping` in `IndexController`.

Bekijk het resultaat in je browser.

10.4 Meerdere controllers

Een website bevat meestal *meerdere* controllers. Je zal ook in deze cursus controllers toevoegen.

De vuistregel is: een controller per belangrijk datatype uit de te automatiseren werkelijkheid.

Voorbeeld: de VDAB

Gegevenstype	Bijbehorende controller	Voorbeeld bijbehorende <code>@GetMapping</code>
werkzoekende	WerkzoekendeController	<code>@GetMapping("/werkzoekenden")</code>
vacature	VacatureController	<code>@GetMapping("/vacatures")</code>
opleiding	OpleidingController	<code>@GetMapping("/opleidingen")</code>

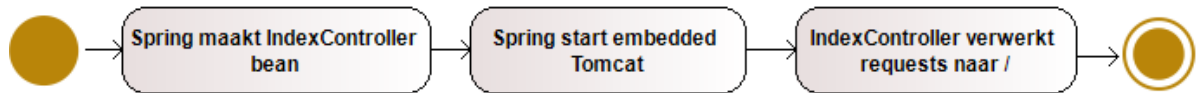
10.5 Nadelen

Je stuurt *enkel als kennismaking* op deze manier responses naar de browser. Dit heeft nadelen:

- ❌ IntelliJ valideert geen HTML tussen " en ".
- ❌ Als een web designer de HTML verfijnt die de programmeur typt, is de kans groot dat hij per ongeluk fouten aanbrengt in de Java code.
- ❌ Je mengt voortdurend Java code en HTML. Beide zijn zo moeilijk leesbaar.

Je lost de nadelen op in het volgende hoofdstuk.

10.6 Overzicht



Commit de sources. Publiceer op je remote repository.



Controller

11 THYMELEAF

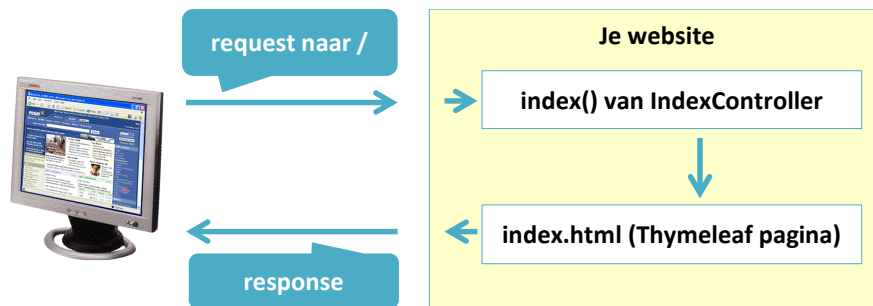
Een Thymeleaf pagina is een HTML pagina waarin je ook eenvoudige code (if, for, ...) typt. Het is niet de browser die de code uitvoert (zoals bij JavaScript code).

De webserver voert de code uit, voor hij de HTML naar de browser stuurt.

Je plaatst Thymeleaf pagina's in de folder templates of in een subfolder van templates.

Een controller en een Thymeleaf pagina werken samen bij het verwerken van een request:

1. Een method van een controller verwerkt de request.
2. De method verwijst naar een Thymeleaf pagina.
3. Die stuurt een response met HTML naar de browser:



11.1 Voorbeeld

De method index van IndexController zal samenwerken met de Thymeleaf pagina index.html

Maak index.html:

1. Klik met de rechtermuisknop op templates.
2. Kies New, HTML File.
3. Typ index. Druk Enter.



Wijzig index.html:

```

<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Pizza Luigi</title>
    <link rel="icon" href="images/luigi.ico" type="image/x-icon">
    <link rel="stylesheet" href="css/luigi.css">
  </head>
  <body>
    <h1>Pizza Luigi</h1>
    
    <h2>Goede dag</h2>
  </body>
</html>
  
```

①

②

- (1) Je associeert de prefix th met de URI van de Thymeleaf library: <http://www.thymeleaf.org>. Je deed dit soort associatie al in XML bestanden. Je neemt later met die prefix code (if, for, ...) op in de pagina.
- (2) Goede dag wordt straks Goede morgen of Goede dag, naargelang het moment van de dag.

11.2 Controller

Wijzig IndexController:

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

①

②

(1) Je vervangt `@RestController` door `@Controller`.

Je doet dit bij een controller die samenwerkt met een Thymeleaf pagina.

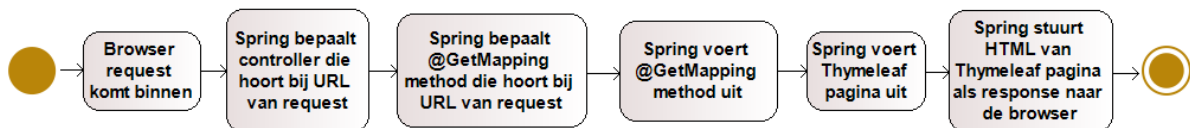
(2) Je geeft de naam van de te gebruiken Thymeleaf pagina terug. Je typt de extensie (`.html`) niet.

Kies  voor de method `index`. Kies `Navigate to related views`.

IntelliJ opent de bijbehorende Thymeleaf pagina.

Druk `Ctrl+F9`. Je kan de website proberen.

11.3 Overzicht



11.4 Data doorgeven

De Thymeleaf pagina toont nu `Goede dag`. Je wijzigt dit naar `Goede morgen` of `Goede middag`:

1. De controller maakt een `String` met `morgen` of `middag`.
2. De controller geeft de `String` door aan de Thymeleaf pagina.
3. De Thymeleaf pagina toont die `String`.

Wijzig de code in `IndexController`:

```
@GetMapping("/")
public ModelAndView index() {
    var morgenOfMiddag = LocalTime.now().getHour() < 12 ? "morgen" : "middag";
    return new ModelAndView("index", "moment", morgenOfMiddag);
}
```

①

②

(1) Een method die data doorgeeft aan de Thymeleaf pagina heeft als returntype `ModelAndView`.
`Model` staat voor de data, `View` staat voor de Thymeleaf pagina.

(2) De 1° parameter van de `ModelAndView` constructor is de naam van de Thymeleaf pagina.

De 2° parameter is de *naam* waaronder je een stukje data doorgeeft: `moment`.

De 3° parameter is de *data* zelf: de inhoud van de variabele `morgenOfMiddag`.

Wijzig in `index.html` de regel `<h2>...</h2>`:

```
<h2>Goede <span th:text="${moment}"></span></h2>
```

①

(1) `th:text` vervangt de tekst *tussen* `` en `` door de data die je van de controller doorkreeg onder de naam `moment`. Je omringt die naam met `${}` en `}`.

Doe een `Ctrl+muisklik` op `${moment}`.

IntelliJ brengt je naar de code in `IndexController` die deze data aanbiedt.

Druk `Ctrl+F9`. Je kan de website proberen.



Commit de sources. Publiceer op je remote repository.



Thymeleaf

12 \${VARIABLE EXPRESSION}

index.html bevat `${moment}`. Een expressie tussen `${}` is een variable expression. Je leert hier meer over variable expressions.

De controller geeft in elk voorbeeld het getal 8 door onder de naam kinderen:

```
return new ModelAndView("index", "kinderen", 8);
```

12.1 Basis voorbeeld

Thymeleaf: Luigi heeft `` kinderen.

In de browser: Luigi heeft 8 kinderen.

12.2 Rekenen

Thymeleaf bevat de bewerkingen `+` `-` `*` `/` en `%`

Thymeleaf: Deling: ``

In de browser: Deling: 4

Opmerking: je doet complexe berekeningen beter in Java code dan in Thymeleaf.

- ⊕ De berekeningen zijn dan herbruikbaar
(als je ze in een method opneemt die het resultaat van de berekening teruggeeft).
- ⊕ Je kan een JUnit test maken voor de berekeningen.

12.3 Vergelijken

Thymeleaf bevat de vergelijkingen `==` `!=` `>` `<` `>=` `<=`

Thymeleaf: ``

In de browser: true

12.4 not, and, or

Je kan not and or gebruiken

Thymeleaf: ` 7 and ${kinderen} < 10">`

In de browser: true

12.5 Verkorte if met ? :

Syntax: voorwaarde ? waardeAlsVoorwaardeTrue : waardeAlsVoorwaardeFalse

Thymeleaf: ``

In de browser: ongeluk

12.6 Vaste tekst en data combineren

Je omsluit een combinatie van vaste tekst en data met `|` en `|` :

Thymeleaf: ``

In de browser: Luigi heeft 8 kinderen.

12.7 Object

De controller geeft een object door, hier een BigDecimal:

```
return new ModelAndView("index", "koers", BigDecimal.valueOf(40.3399));
```

Thymeleaf: De koers van euro is: ``

Thymeleaf voert op het object de method toString uit en neemt die String op in de HTML.

In de browser: De koers van de euro is: 40.3399

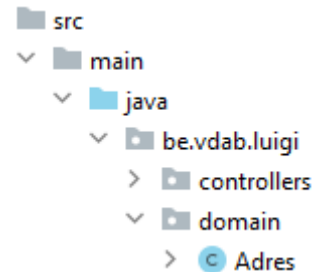
13 GETTER

Je leert hoe Thymeleaf attributen van objecten leest met getters.

13.1 Getters

Maak een package `be.vdab.luigi.domain`. Maak daarin een class `Adres`:

```
package be.vdab.luigi.domain;
public class Adres {
    private String straat;
    private String huisNr;
    private int postcode;
    private String gemeente;
}
```



Maak getters:

1. Klik in de source van de class `Adres` voor de sluit `}` van de class.
2. Kies het menu `Code, Generate, Getter`.
3. Selecteert alle variabelen. Kies `OK`.

13.2 Constructor

Maak een constructor met parameters:

1. Klik in de source van de class `Adres` voor de sluit `}` van de class.
2. Kies het menu `Code, Generate, Constructor`.
3. Selecteer alle variabelen. Kies `OK`.

13.3 Immutable

Een class kan naast getters ook setters bevatten.

Moderne Java programmeurs maken weinig setters. Ze maken zo hun objecten immutable.

Een immutable object is een object waarvan je de private variabelen in de constructor initialiseert.

Je wijzigt daarna die variabelen *niet meer*. Dit heeft veel voordelen:

- Minder code. Dit bevordert de leesbaarheid. Je moet ook minder code testen.
- Hoe minder een object wijzigt, hoe minder het verkeerde waarden kan bevatten.
- Thread safe: threads kunnen tegelijk het object gebruiken zonder dat er iets fout loopt: ze kunnen het object enkel lezen, niet wijzigen.

Soms wijzigt een private variabele wel. Een setter is zelden de mooie voorstelling van de wijziging.

Voorbeeld: je maakt in een class `Rekening` (uit een bank) geen method `setSaldo`.

Je kan in de werkelijkheid het saldo niet *zomaar* wijzigen (wat `setSaldo` wel doet lijken).

Het saldo van een rekening wijzigt enkel door te storten, af te halen of over te schrijven.

Je maakt dus geen method `setSaldo(BigDecimal saldo)`, maar volgende methods:

- `stort(BigDecimal bedrag)`
- `haalAf(BigDecimal bedrag)`
- `schrijfOver(BigDecimal bedrag, Rekening naarRekening)`

De standaard Java libraries bevatten ook immutable classes: `String`, `BigDecimal`, `LocalDate`, ...

13.4 Final

Je kan in een immutable class per ongeluk een private variabele wijzigen *na* zijn initialisatie.

Je voorkomt dit met het keyword `final`. De compiler controleert dan dat de class immutable is.

Je krijgt een fout als je een private variabele wijzigt in andere methods dan de constructor.

```
private final String straat;
private final String huisNr;
private final int postcode;
private final String gemeente;
```

13.5 Genest object

Een attribuut van een class kan een object zijn dat *zelf* ook attributen bevat. Voorbeeld: Maak de class Persoon. Die heeft onder andere attribuut adres van het type Adres:

```
package be.vdab.luigi.domain;
import java.time.LocalDate;
public class Persoon {
    private final String voornaam;
    private final String familienaaam;
    private final int aantalKinderen;
    private final boolean gehuwd;
    private final LocalDate geboorte;
    private final Adres adres;
    // constructor met parameters, getters
}
```



13.6 Getter gebaseerd op een berekening

Sommige getters zijn gebaseerd op een berekening (optelling, concatenatie, ...).

Maak zo'n getter in de class Persoon:

```
public String getNaam() {
    return voornaam + ' ' + familienaaam;
}
```

13.7 Controller

Wijzig in IndexController de method index:

```
@GetMapping("/")
public ModelAndView index() {
    var morgenOfMiddag = LocalTime.now().getHour() < 12 ? "morgen" : "middag";
    var modelAndView = new ModelAndView("index", "moment", morgenOfMiddag);
    modelAndView.addObject("zaakvoerder",
        new Persoon("Luigi", "Peperone", 7, true, LocalDate.of(1966, 1, 31),
            new Adres("Grote markt", "3", 9700, "Oudenaarde")));
    return modelAndView;
}
```

- (1) Je geeft met addObject *extra* data door onder de naam zaakvoerder.
- (2) De data is een Persoon object.

13.8 Thymeleaf

Typ in index.html voor </body>:

```
<h2>De zaakvoerder</h2>
<dl>
    <dt>Naam</dt>
    <dd th:text="${zaakvoerder.naam}"></dd>
    <dt>Aantal kinderen</dt>
    <dd th:text="${zaakvoerder.aantalKinderen}"></dd>
    <dt>Gehuwd</dt>
    <dd th:text="${zaakvoerder.gehuwd} ? 'Ja' : 'Nee'"></dd>
</dl>
```

- (1) Thymeleaf vertaalt als volgt .naam naar de method getNaam van het Persoon object:
 - Thymeleaf vervangt in naam de eerste letter door zijn hoofdletter vorm: Naam.
 - Thymeleaf plaatst hier get voor: getNaam.
 Thymeleaf roept de method getNaam op en neemt de return waarde op in de HTML.

Je kan de website proberen.

13.9 Selection expression

Je herhaalt in elk <dd> element dat je een eigenschap van zaakvoerder aanspreekt. Dit kan korter:

```
<dl th:object="${zaakvoerder}">
  <dt>Naam</dt>
  <dd th:text="*{naam}"></dd>
  <dt>Aantal kinderen</dt>
  <dd th:text="*{aantalKinderen}"></dd>
  <dt>Gehuwd</dt>
  <dd th:text="*{gehuwd} ? 'Ja' : 'Nee'"></dd>
</dl>
```

❶

❷

- (1) Je geeft met th:object aan dat je van zaakvoerder attributen zal lezen.
Je kan die lezen in het huidige element <dl> en in al zijn child elementen.
- (2) Je leest het attribuut naam met een selection expression: een expression tussen *{ en }.

Je kan de website proberen.

13.10 Genest attribuut

Typ in index.html voor </dl>:

```
<dt>Adres</dt>
<dd th:object="${zaakvoerder.adres}"
    th:text="|*{straat} *{huisNr} *{postcode} *{gemeente}|"></dd>
```

❶

❷

- (1) Je geeft aan dat je attributen zal lezen van zaakvoerder.adres.
- (2) Je combineert data en vaste tekst (hier en daar een spatie) tussen | en |.

Je kan de website proberen.



Commit de sources. Publiceer op je remote repository.



Adres

14 TH:EACH

14.1 Controller

Maak `PizzaController`. Hij verwerkt alle requests die met pizza's te maken hebben:

```
package be.vdab.luigi.controllers;
// enkele imports
@Controller
class PizzaController {
    private final String[] allePizzas = {"Prosciutto", "Margherita", "Calzone"};
    @GetMapping("/pizzas")
    public ModelAndView findAll() {
        return new ModelAndView("pizzas", "allePizzas", allePizzas);
    }
}
```

- (1) De method op de volgende regel (`findAll`) verwerkt GET request naar de URL `/pizzas`
- (2) `return new ModelAndView("pizzas", "allePizzas", allePizzas);`
 1. de naam van de Thymeleaf pagina (`pizzas.html`)
 2. de *naam* waaronder je data doorgeeft aan de Thymeleaf pagina.
 3. de *data* die je doorgeeft aan de Thymeleaf pagina: een array van pizza's

14.2 Thymeleaf

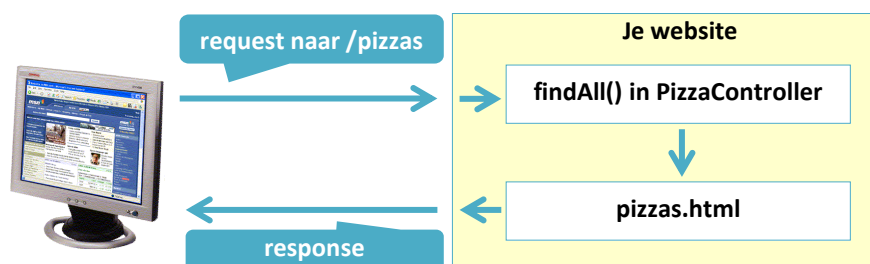
Maak `pizzas.html`:

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Pizza's</title>
        <link rel="icon" href="images/luigi.ico" type="image/x-icon">
        <link rel="stylesheet" href="css/luigi.css">
    </head>
    <body>
        <h1>Pizza's</h1>
        <ul>
            <li th:each="pizza : ${allePizzas}" th:text="${pizza}"></li>
        </ul>
    </body>
</html>
```

- (1) `allePizzas` bevat de array met pizza's. `th:each` itereert over de pizza's in die array. De variabele `pizza` wijst bij elke iteratie naar een volgende pizza in de array. `th:each` maakt per pizza een `<li ...>` element. `th:text` vervangt de tekst tussen `` en `` door de inhoud van de variabele `pizza`.

Probeer dit op <http://localhost:8080/pizzas>.

Overzicht:



14.3 Verzameling objecten

14.3.1 Pizza

Maak een class Pizza:

```
package be.vdab.luigi.domain;
import java.math.BigDecimal;
public class Pizza {
    private final long id;
    private final String naam;
    private final BigDecimal prijs;
    private final boolean pikant;
    // constructor met parameters, getters
}
```

14.3.2 Controller

Vervang in PizzaController de variabele allePizzas:

```
private final Pizza[] allePizzas = {
    new Pizza(1, "Prosciutto", BigDecimal.valueOf(4), true),
    new Pizza(2, "Margherita", BigDecimal.valueOf(5), false),
    new Pizza(3, "Calzone", BigDecimal.valueOf(4), false)};
```

14.3.3 pizzas.html

Vervang tot en met :

```
<table id="pizzatabel">
    <thead>
        <tr>
            <th>Naam</th>
            <th>Prijs</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="pizza : ${allePizzas}" th:object="${pizza}">
            <td th:text="*{naam}"></td>
            <td th:text="*{prijs}" class="getal"></td>
        </tr>
    </tbody>
</table>
```

❶

❷

❸

(1) Je geeft de table de id pizzatabel.

Je verwijst naar die id in luigi.css om de tabel opmaak te geven.

(2) Je geeft met th:object aan dat je attributen zal lezen van pizza.

(3) Je toont het attribuut naam van de pizza.

Je kan dit proberen.

14.4 List, Set

Je kan met th:each ook itereren over de items in een List of een Set.

Als een controller een List zou doorgeven:

```
modelAndView.addObject("getallen", List.of(3, 7));
```

itereer je met volgende Thymeleaf code over de item in die List:

```
<ul th:each="getal : ${getallen}">
    <li th:text="${getal}"></li>
</ul>
```

14.5 Stream

Je kan met `th:each` ook itereren over de items in een Stream als je die Stream doorgeeft onder de gedaante van een Iterator. Je converteert een Stream naar een Iterator met de method `iterator()`.

Als een controller een List zou doorgeven:

```
modelAndView.addObject("getallen", Stream.of(3, 7).iterator());
```

itereer je met volgende Thymeleaf code over de item in die Stream:

```
<ul th:each="getal : ${getallen}">
  <li th:text="${getal}"></li>
</ul>
```

14.6 Map

Als een controller een Map zou doorgeven:

```
modelAndView.addObject("landen", Map.of("B", "België", "NL", "Nederland"));
```

itereer je met volgende Thymeleaf code over de entries in die Map:

```
<ul th:each="entry : ${landen}">
  <li th:text="|${entry.key} ${entry.value}|"></li>
</ul>
```



Kies het menu View, Tool Windows, Endpoints. Je ziet een lijst met de URL's waarnaar je applicatie requests verwerkt. Dubbelklik zo'n URL. IntelliJ opent de source van de method die een request naar die URL verwerkt.

15 TH:IF

th:if neemt een stuk HTML enkel op in de response als een voorwaarde true is.

Voorbeeld: Je voegt een kolom toe aan de tabel in pizzas.html.

Je toont in die kolom de tekst pikant bij een pikante pizza.

Voeg na de laatste </th> een kolomhoofding toe:

```
<th>Pikant</th>
```

Voeg na de laatste </td> een kolom toe:

```
<td><span th:if="{pikant}">pikant</span></td>
```

(1) Thymeleaf neemt de span tag enkel op in de response als de eigenschap pikant true is.

Je kan dit proberen.



Je kan een Thymeleaf pagina niet debuggen of unit testen.

Je programmeert complexe code daarom niet in Thymeleaf, maar in Java.



Je hebt nu twee controllers:

- IndexController: verwerkt requests naar /.
- PizzaController: verwerkt requests naar /pizzas.



Commit de sources. Publiceer op je remote repository.



Sauzen

16 TH:BLOCK

Je maakte in Thymeleaf pagina's soms elementen, met als enige reden er Thymeleaf attributen (`th:text`, `th:if`) op toe te passen. Die elementen hebben geen nut voor de browser.

Je gebruikt dan beter het Thymeleaf element `th:block`.

Thymeleaf verwerkt de attributen van dit element, maar maakt geen bijbehorend HTML element.

16.1 Voorbeeld 1

`index.html` bevat

```
<h2>Goede <span th:text="${moment}"></span></h2>.
```

Je gebruikt het `span` element enkel om de inhoud van de variabele `moment` te tonen.

Je stuurt volgende HTML naar de browser als de variabele de string `morgen` bevat:

```
<h2>Goede <span>morgen</span></h2>
```

Het `span` element heeft in de browser geen meerwaarde.

De gebruiker zou hetzelfde zien als je volgende HTML naar de browser stuurt:

```
<h2>Goede morgen</h2>
```

Je doet dit met het element `th:block`:

```
<h2>Goede <th:block th:text="${moment}"></th:block></h2>
```

Thymeleaf verwerkt het `th:text` attribuut, maar maakt geen HTML element voor `th:block`.

Je stuurt nu volgende HTML naar de browser:

```
<h2>Goede morgen</h2>
```

16.2 Voorbeeld 2

`pizzas.html` bevat

```
<td><span th:if="*{pikant}">pikant</span></td>.
```

Je gebruikt het `span` element enkel om de tekst `pikant` te tonen als de pizza pikant is.

Je stuurt dan volgende HTML naar de browser:

```
<td><span>pikant</span></td>
```

Het `span` element heeft in de browser geen meerwaarde.

De gebruiker zou hetzelfde zien als je volgende HTML naar de browser stuurt:

```
<td>pikant</td>
```

Je doet dit met het element `th:block`:

```
<td><th:block th:if="*{pikant}">pikant</th:block></td>
```

Thymeleaf verwerkt het `th:text` attribuut, maar maakt geen HTML element voor `th:block`.

Je stuurt nu volgende HTML naar de browser:

```
<td>pikant</td>
```

17 @{URL EXPRESSION}

pizzas.html bevat verwijzingen naar URL's binnen je website:

- `<link rel="icon" href="images/luigi.ico" type="image/x-icon">`
- `<link rel="stylesheet" href="css/luigi.css">`

De browser verwerkt de URL `css/luigi.css` als volgt:

1. Hij neemt de URL van de request die naar de pagina leidde: `http://localhost:8080/pizzas`
2. Hij verwijdert alles na de laatste slash: `http://localhost:8080/`
3. Hij voegt `css/luigi.css` toe: `http://localhost:8080/css/luigi.css`

Dit soort URL's kunnen problemen veroorzaken.

Je wijzigt de URL in `PizzaController` om dit te zien: `@GetMapping("/pizzas/all")`

Je bekijkt het resultaat in je browser op <http://localhost:8080/pizzas/all>.

De pagina is niet meer opgesteld, omdat de browser `luigi.css` niet meer vindt:

1. Hij neemt de URL van de request: `http://localhost:8080/pizzas/all`
 2. Hij verwijdert alles na de laatste slash: `http://localhost:8080/pizzas/`
 3. Hij voegt `css/luigi.css` toe: `http://localhost:8080/pizzas/css/luigi.css`
- Het CSS bestand bevindt zich niet op die URL: `/pizzas` is overbodig in die URL.

Je ziet in het tabblad Network in Chrome dat de browser `luigi.css` en `luigi.ico` niet meer vindt.

De requests geven responses met status code 404 (Not Found).

17.1 URL expression

Je lost dit probleem op met een onderdeel van Thymeleaf: URL expressions.

Gebruik een URL expression in de 5° regel in `pizzas.html`:

```
<link rel="icon" th:href="@{/images/luigi.ico}" type="image/x-icon">
```

❶

- (1) Je vervangt `href` door `th:href`. Je omsluit de URL met `@{` en `}`. Je begint de URL met `/`. `/` staat voor de "root URL" van je website: de URL die hoort bij je welkom pagina.

Wijzig ook de 6° regel:

```
<link rel="stylesheet" th:href="@{/css/luigi.css}">
```

Je kan dit proberen. De pagina werkt terug correct.

Wijzig de URL van `PizzaController` terug naar een kortere vorm:

```
@GetMapping("/pizzas")
```

Je kan dit proberen. De pagina werkt nog altijd correct.

Gebruik ook URL expressions in `index.html`:

```
regel 5: <link rel="icon" th:href="@{/images/luigi.ico}" type="image/x-icon">
```

```
regel 6: <link rel="stylesheet" th:href="@{/css/luigi.css}">
```

```
regel 10: 
```

17.2 Nut van @{...}

Je bekijkt de welkom pagina in je browser. Je bekijkt de bron van de pagina.

Je ziet de HTML die je website naar de browser stuurde. Je ziet onder andere volgende regel:

```

```

Het lijkt dat het omsluiten van de URL met @{ en } niets doet en overbodig is.

@{ en } gebruiken is echter wél belangrijk.

Je maakt op het einde van de cursus een WAR bestand van je website: luigi.war.

Je installeert met dit bestand je website op je aparte Tomcat.

De URL van je website is dan localhost:8080/**luigi**.

Tomcat leidt het onderdeel luigi in die URL af uit de naam van je WAR bestand.

De afbeelding luigi.jpg bevindt zich dan op de URL /**luigi**/images/luigi.jpg.

Gelukkig houdt @{ } hiermee rekening. De regel

```

```

stuurt dan volgende HTML naar de browser

```

```

17.3 Thymeleaf expressie

Je leerde drie soorten Thymeleaf expressies:

- `${...}` Data lezen die je kreeg van de controller.
- `*{...}` Attribuut lezen van een object dat je aangaf met `th:object`.
- `@{...}` URL maken.



Commit de sources. Publiceer op je remote repository.



URL

18 FRAGMENT

Sommige stukken HTML komen *elke* pagina terug. Voorbeeld: Elke pagina bevat een menu:

```
<nav>
  <ul>
    <li><a th:href="@{/}">Welkom</a></li>
    <li><a th:href="@{/pizzas}">Pizza's</a></li>
  </ul>
</nav>
```

Als je dit menu in elke pagina typt is de website moeilijk te onderhouden.

Als er een pagina bijkomt, moet je in elke pagina een hyperlink toevoegen aan het menu.

De oplossing is een Thymeleaf fragment: HTML code die je één keer typt in een HTML bestand.

Je verwijst daarna vanuit andere HTML bestanden naar dit fragment.

Thymeleaf neemt dan de HTML code uit het fragment op in die andere bestanden.

18.1 Maken

Maak `fragments.html` (de bestandsnaam is vrij te kiezen) in de folder `templates`:

```
<div xmlns:th="http://www.thymeleaf.org">
  <nav th:fragment="menu">
    <ul>
      <li><a th:href="@{/}">Welkom</a></li>
      <li><a th:href="@{/pizzas}">Pizza's</a></li>
    </ul>
  </nav>
</div>
```

❶

(1) Je geeft elk fragment een unieke naam met `th:fragment`. Dit fragment heet `menu`.

18.2 Gebruiken

Typ volgende regel in `index.html` en in `pizzas.html`, onder `<body>`:

```
<nav th:replace="fragments::menu"></nav>
```

❶

(1) Thymeleaf vervangt het huidig `<nav>` element door het fragment met de naam `menu` in het bestand `fragments(.html)`.

Je kan dit proberen.

18.3 Parameter

`index.html` en `pizzas.html` bevatten *nog* gelijkaardige code:

```
<head>
  <title>Pizza Luigi</title>
  <link rel="icon" th:href="@{/images/luigi.ico}" type="image/x-icon">
  <link rel="stylesheet" th:href="@{/css/luigi.css}">
</head>
```

❶

(1) Er is één verschil: `index.html` bevat hier `Pizza Luigi`. `pizzas.html` bevat hier `Pizza's`.

18.3.1 Maken

Je kan ook deze code één keer typen in een fragment. Je stelt het verschil voor met een parameter.

Voeg dit fragment toe aan `fragments.html`, na `</nav>`:

```
<head th:fragment="head(title)"> ❶
  <link rel="icon" th:href="@{/images/luigi.ico}" type="image/x-icon">
  <title th:text="${title}"></title> ❷
  <link rel="stylesheet" th:href="@{/css/luigi.css}">
</head>
```

- (1) Het fragment heet `head`. Het heeft één parameter. Hij heet `title`.
Een fragment kan meerdere parameters hebben. Je scheidt ze met een komma.
- (2) Je gebruikt de inhoud van de parameter `title`.

18.3.2 Gebruiken

Vervang in `index.html` de regels `<head>...</head>`:

```
<head th:replace="fragments::head(title='Pizza Luigi')"></head> ❶
```

- (1) Thymeleaf vervangt het huidige `<head>` element door het fragment met de naam `head`.
Je geeft `Pizza Luigi` mee als waarde voor de parameter `title` van dit fragment.

Vervang in `pizzas.html` de regels `<head>...</head>`:

```
<head th:replace="fragments::head(title='Pizza\'s')"></head> ❶
```

- (1) Je typt in een String die zelf omsloten is met `'` tekens `'` als `\'`.

Je kan dit proberen.

18.4 pizzaTabel

`pizzas.html` bevat code waarmee je pizza's toont in een tabel.

Je zal ook in andere pagina's zo'n tabel nodig hebben.

Je maakt daarom een fragment van de code:

1. Kopieer `<table>` tot en met `</table>` op het klembord.
2. Plak dit in `fragments.html`, na `</head>`.
3. Wijzig daar `<table>`:

```
<table th:fragment="pizzaTabel(pizzas)" xmlns:th="http://www.thymeleaf.org"
      id="pizzatabel">
```

 Je maakt zo een fragment `pizzaTabel` met een parameter `pizzas`.
4. Wijzig `allePizzas` naar `pizzas`.
5. Vervang in `pizzas.html` `<table> ... </table>` door de oproep van het fragment.
 Je geeft de inhoud van de variabele `allePizzas` door aan de parameter `pizzas`:

```
<table th:replace="fragments::pizzaTabel(pizzas=${allePizzas})"></table>
```



Commit de sources. Publiceer op je remote repository.



Fragments

19 PATH VARIABLE

Een path variabele stelt een *variabel* onderdeel in een URL voor.

Voorbeeld: je maakt in de pagina met pizza's van elke pizza naam een hyperlink.

Als de gebruiker er op klikt, ziet hij een nieuwe pagina met de detail van die pizza.

Je moet in de nieuwe pagina weten over welke *pizza* je details toont. Je lost dit als volgt op.

Elke hyperlink in pizzas.html bevat een URL die naar de nieuwe pagina verwijst.

Je neemt de id van de pizza op in de URL. De hyperlinks zullen er als volgt uitzien:

- `detail`
- `detail`
- `detail`

De URL's zijn gelijkaardig, behalve de pizza id. Dit is de path variabele (variabel onderdeel).

19.1 URI template

Een URI template is een URL waarin je de path variabele aangeeft met een naam tussen accolades.

Je stelt de URL's in ons voorbeeld voor met de URI template: `/pizzas/{id}`

Vervang in fragments.html de eerste `<td>...</td>`:

```
<td><a th:text="{naam}" th:href="@{/pizzas/{id}(id={id})}"></a></td> ❶
```

- (1) Je vermeldt de URI template `/pizzas/{id}`. Thymeleaf maakt een URL op basis van de URI template. Je typt tussen ronde haakjes dat Thymeleaf de path variabele `id` moet invullen met de id van de huidige pizza. Je vindt dit nummer in `{id}`.

Je kan dit proberen. Laat de muisaanwijzer rusten boven de eerste pizza naam. Je ziet onder in de browser dat hij verwijst naar de URL `localhost:8080/pizzas/1`. Klik op de hyperlink.

Je krijgt een fout. Er is nog geen controller method die requests naar `/pizzas/1` verwerkt.

19.2 Controller

Maak in `PizzaController` methods:

```
private Optional<Pizza> findByIdHelper(long id) { ❶
    return Arrays.stream(allePizzas).filter(pizza->pizza.getId()==id).findFirst();
}
@GetMapping("/pizzas/{id}") ❷
public ModelAndView findById(@PathVariable long id) { ❸
    var modelAndView = new ModelAndView("pizza"); ❹
    findByIdHelper(id).ifPresent(gevondenPizza -> ❺
        modelAndView.addObject("pizza", gevondenPizza));
    return modelAndView;
}
```

- (1) Deze method zoek de pizza waarvan je de id meegeeft als parameter. De method geeft een `Optional` terug. Die bevat de gevonden pizza of is leeg als de pizza niet bestaat.
- (2) De method bij ❸ verwerkt GET requests naar URL's die passen bij de URL template `pizzas/{id}`.
- (3) Voor de method parameter `id` staat `@PathVariable`. Spring vult dan die parameter `id` met de waarde van de path variabele met dezelfde naam (`id`). Als de URL `pizzas/1` is, vult Spring `id` met 1.
- (4) Je zal samenwerken met de pagina `pizza.html`.
- (5) Je roept de method bij ❶ op en je gebruikt `id` als parameter. Als je de pizza vindt, geef je die door aan de Thymeleaf pagina onder de naam `pizza`.

19.3 pizza.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head th:replace="fragments::head(title = ${pizza} ? ${pizza.naam} :
    'Pizza niet gevonden')"></head>
  <body>
    <nav th:replace="fragments::menu"></nav>
    <div th:if="not ${pizza}" class="fout">Pizza niet gevonden:
      <th:block th:text="${id}"></th:block>
    </div>
    <th:block th:if="${pizza}" th:object="${pizza}">
      <h1 th:text="*{naam}"></h1>
      <dl>
        <dt>Nummer</dt>
        <dd th:text="*{id}"></dd>
        <dt>Prijs</dt>
        <dd th:text="*{prijs}"></dd>
        <dt>Pikant</dt>
        <dd th:text="*{pikant} ? 'Ja' : 'Nee'"></dd>
      </dl>
    </th:block>
  </body>
</html>
```

①

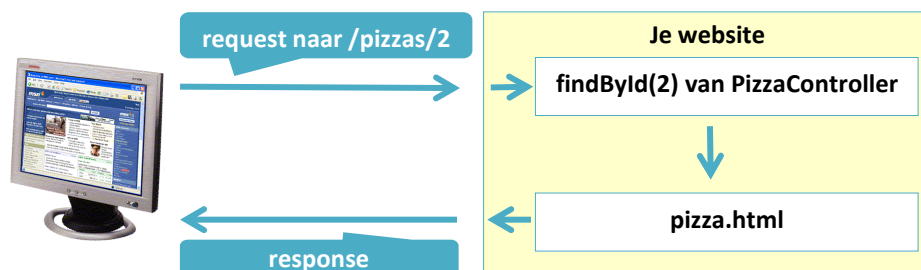
②

- (1) Als de controller een pizza doorgaf, verschilt de variabele pizza van null. Je toont dan de naam van de pizza. Je toont anders de tekst Pizza niet gevonden.
- (2) De pagina hoort bij de controller method pizza. Die verwerkt een path variabele id. Spring geeft de path variabele *zelf* door aan de Thymeleaf pagina onder de naam id.

Je kan dit proberen.

Surf naar <http://localhost:8080/pizzas/666> om de tekst Pizza niet gevonden ... te zien.

Overzicht:



19.4 Meerdere path variabelen

Een URI template kan *meerdere* path variabelen bevatten.

De fictieve URL pizzas/verkoopstatistieken/2015/10 geeft de verkoopstatistieken van oktober 2015.

De bijbehorende URI template: verkoopstatistieken/{jaar}/{maand}

De bijbehorende controller method:

```
@GetMapping("/pizzas/verkoopstatistieken/{jaar}/{maand}")
ModelAndView statistieken(@PathVariable int jaar, @PathVariable int maand) {
    ...
}
```



Commit de sources. Publiceer op je remote repository.



URL's en methods, Path variabele

20 @REQUESTMAPPING

De class `PizzaController` bevat meerdere `@GetMapping` annotations, met telkens dezelfde begin van de URL: `/pizzas`:

- `@GetMapping("/pizzas")`
- `@GetMapping("/pizzas/{id}")`

Meerdere keer hetzelfde typen is niet goed:

- ❌ Je kan typefouten maken.
- ❌ Als dit URL beginstuk wijzigt, moet je het op meerdere plaatsen wijzigen.

Je lost dit probleem op met `@RequestMapping`:

1. Typ `@RequestMapping("pizzas")` voor de class.
Spring voegt hierin voor `pizzas` zelf / toe.
Dit bevat dus het URL beginstuk: `/pizzas`.
2. Vervang `@GetMapping("/pizzas")` door `@GetMapping`.
Deze `@GetMapping` bevat nu geen URL meer.
Spring gebruikt nu bij deze `@GetMapping` de URL van `@RequestMapping`: `/pizzas`.
3. Vervang `@GetMapping("/pizzas/{id}")` door `@GetMapping("{id}")`
Spring maakt een URL template. Die bestaat uit de URL van `@RequestMapping`: `/pizzas`, daarna / en daarna het URL onderdeel bij `@GetMapping`: `{id}`.
De URL template bij deze `@GetMapping` wordt dus: `/pizzas/{id}`.



Commit de sources. Publiceer op je remote repository.



`@RequestMapping`

21 NAAM WAARONDER JE DATA DOORGEEFT AAN THYMELEAF

In de class `PizzaController` bevat de method `findById` volgend stukje code:

```
modelAndView.addObject("pizza", gevondenPizza)
```

Je kan dit verkorten:

```
modelAndView.addObject(gevondenPizza);
```




1

- (1) Spring kijkt naar het type van `gevondenPizza`. Dit is de class `Pizza`.
 - Spring wijzigt de 1^e letter daarvan naar kleine letter: `pizza`.
 - Spring geeft de data onder die naam (`pizza`) aan de Thymeleaf pagina.








22 MODEL-VIEW-CONTROLLER

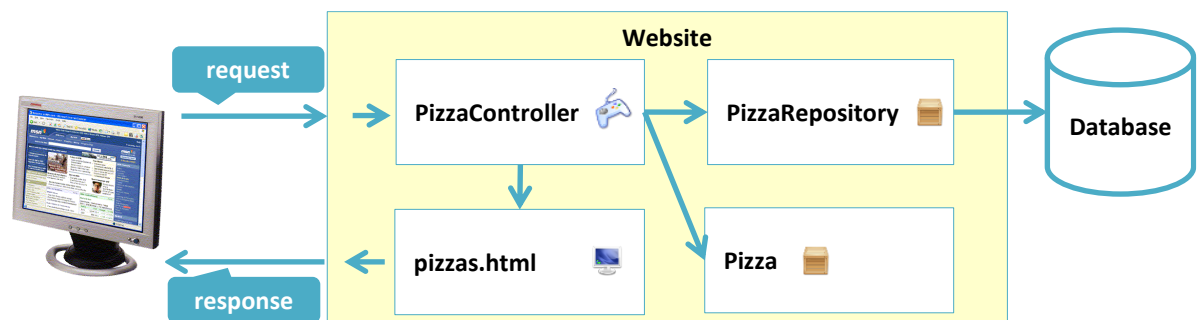
Het MVC (Model-View-Controller) design pattern bevat drie soorten onderdelen.

Elk onderdeel heeft zijn eigen taak.

	Onderdeel	Taak
	Model	De werkelijkheid voorstellen en database bewerkingen doen. <ul style="list-style-type: none">• Domain classes stellen de werkelijkheid voor.• Repository classes doen database bewerkingen.
	View	Data tonen aan de gebruiker. Dit onderdeel is een Thymeleaf pagina.
	Controller	Requests binnenkrijgen. Dit onderdeel is een controller bean.

Model, view en controller werken samen bij het verwerken van een request. Voorbeeld:

1. PizzaController  verwerkt een request om pizza's te zien.
2. PizzaController  roept PizzaRepository  op.
Die leest pizza's uit de database en geeft ze terug als een `List<Pizza`  `>`.
3. PizzaController  geeft de `List<Pizza`  door aan `pizzas.html` .
Die maakt HTML en stuurt die als response naar de browser.



23 REQUEST HEADER

Een request bevat headers. Ze bevatten informatie over de browser.

De gebruiker ziet de headers niet. Je ziet ze in de Chrome developer tools:

1. Druk F12.
2. Surf naar <http://localhost:8080>
3. Kies in het tabblad Network de GET request naar /.
Scrol rechts naar beneden tot Request Headers.
Een header heeft een naam en een waarde. Voorbeelden:
De header Accept-Language header bevat de voorkeur taal en land van de gebruiker.
De header User-Agent bevat het type browser en het besturingssysteem.

23.1 Controller

Je leest de header User-Agent. Je bepaalt daarmee het operating systeem van de gebruiker.

Maak OSController:

```
package be.vdab.luigi.controllers;
// enkele imports
@Controller
@RequestMapping("os")
class OSController {
    private static final String[] OSS = {"Windows", "Macintosh", "Android", "Linux"};
    @GetMapping
    public ModelAndView os(@RequestHeader("User-Agent") String userAgent) {
        var modelAndView = new ModelAndView("os");
        Arrays.stream(OSS)
            .filter(os -> userAgent.contains(os))
            .findFirst()
            .ifPresent(gevondenOS -> modelAndView.addObject("os", gevondenOS));
        return modelAndView;
    }
}
```

- (1) De controller verwerkt requests naar de URL os.
- (2) Spring vult de method parameter userAgent met de inhoud van de request header User-Agent vermeld in @RequestHeader.
- (3) Je werkt samen met os.html.
- (4) Als in de header User-Agent bvb. de tekst Windows voorkomt, draait de browser op Windows.
- (5) Je geeft de naam van het besturingssysteem onder de naam os door aan de Thymeleaf pagina.

23.2 os.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head th:replace="fragments::head(title='OS')"></head>
  <body>
    <nav th:replace="fragments::menu"></nav>
    <h1>Je besturingssysteem is
      <th:block th:text="${os} ? ${os} : 'onbekend'"></th:block>
    </h1>
  </body>
</html>
```

Voeg een menu-punt voor de nieuwe pagina toe in het fragment menu in fragments.html:

```
<li><a th:href="@{/os}">OS</a></li>
```

Je kan dit proberen.



Commit de sources. Publiceer op je remote repository.



Taal

24 COOKIE

Een cookie bevat data. De browser onthoudt die ten dienste van een website.

Voorbeelden: gebruikersnaam, favoriete achtergrondkleur.

- ⊖ Bewaar in cookies geen confidentiële data (paswoorden, betaalkaart nummers, ...)!
Je kan in browsers gemakkelijk de inhoud van cookies zien.
Je ziet bij Chrome de cookies via F12, tabblad Application, links Storage ► Cookies.
- ⊖ De gebruiker kan in de browser cookies uitschakelen.
Als je website cookies gebruikt, werkt je website bij die gebruiker slecht.

24.1 Voorbeeld

De gebruiker kiest zijn favoriete kleur: roze of blauw. Je onthoudt die in een permanente cookie. Je gebruikt de kleur vanaf dan als achtergrondkleur van de pagina.

24.1.1 KleurController

```
package be.vdab.luigi.controllers;
// enkele imports
@Controller @RequestMapping("kleuren")
class KleurController {
    private static final int EEN_JAAR_IN_SECONDEN = 31_536_000;
    @GetMapping
    public ModelAndView toonPagina(@CookieValue Optional<String> kleur) {           ❶
        var modelAndView = new ModelAndView("kleuren");
        kleur.ifPresent(hetKleur -> modelAndView.addObject("kleur", hetKleur));    ❷
        return modelAndView;
    }
    @GetMapping("{kleur}")
    public String kiesKleur(@PathVariable String kleur,                           ❸
        HttpServletResponse response) {                                           ❹
        var cookie = new Cookie("kleur", kleur); // javax.servlet.http.Cookie    ❺
        cookie.setMaxAge(EEN_JAAR_IN_SECONDEN);                                  ❻
        cookie.setPath("/");                                                       ❼
        response.addCookie(cookie);                                                ❽
        return "kleuren";                                                         ❾
    }
}
```

- (1) Spring vult de parameter `kleur` met de inhoud van de cookie met dezelfde naam (`kleur`). Als de cookie ontbreekt, is de `Optional` leeg.
- (2) Als de cookie er is, geeft je de inhoud aan de Thymeleaf pagina als de variabele `kleur`.
- (3) De Thymeleaf pagina zal twee hyperlinks bevatten.
Bij een klik op de 1^o hyperlink stuurt de browser een GET request naar `/kleuren/roze`.
Bij een klik op de 2^o hyperlink stuurt de browser een GET request naar `/kleuren/blauw`.
`roze` of `blauw` komen binnen als path variabele `kleur`.
- (4) Je stuurt een cookie naar de browser met een `HttpServletResponse` object.
- (5) Je maakt een cookie. De naam is `kleur`. De inhoud is de kleur die de gebruiker selecteerde.
- (6) Je vult `maxAge` in. Je maak zo een permanente cookie. De browser onthoudt die op de harde schijf. De browser verwijder de cookie na `maxAge` seconden.
Als je `maxAge` niet invult, maak je een tijdelijke cookie.
De browser onthoud die in het RAM geheugen, tot je de browser sluit.
- (7) `path` bevat het begin van een URL in je website.
Enkel requests naar URL's die hiermee beginnen, kunnen de cookie lezen. Voorbeelden:
 - a. `path` bevat `/`. Alle requests van je website kunnen de cookie lezen.
 - b. `path` bevat `/mandje`. Enkel requests naar URL's die beginnen met `/mandje` kunnen de cookie lezen.
- (8) Je voegt de cookie toe aan de response.
- (9) Je werkt samen met `kleuren.html`. Je moet de `kleur` niet zelf doorgeven aan de Thymeleaf pagina. Spring doet dit zelf met elke path variabele.

24.1.2 kleuren.html

```
<!doctype html>
<html lang="nl" xmlns:th='http://www.thymeleaf.org'>
  <head th:replace="fragments::head(title='Kleur')"></head>
  <body th:class="${kleur}">
    <nav th:replace="fragments::menu"></nav>
    <h1>Kleur</h1>
    <ul>
      <li><a th:href="@{/kleuren/roze}">Roze</a></li>
      <li><a th:href="@{/kleuren/blauw}">Blauw</a></li>
    </ul>
  </body>
</html>
```

❶

(1) Je verwijst naar de CSS class `roze` of `blauw` die je van de controller kreeg onder de naam `kleur`.

Als er geen variabele `kleur` is, vereenvoudigt Thymeleaf deze regel tot `<body>`.

Voeg een menu-punt voor de nieuwe pagina toe in het fragment `menu` in `fragments.html`:

```
<li><a th:href="@{/kleuren}">Kleur</a></li>
```

Probeer dit.

1. Kies een kleur.
2. Bezoek andere pagina's van de website.
Keer terug naar de kleur pagina.
3. Die heeft nog de laatst gekozen achtergrondkleur.



Als je in een `@Get` method van een andere controller de cookie `kleur` wil lezen, voeg je aan de method volgende parameter toe:

```
@CookieValue Optional<String> kleur
```

Je verwijdert een permanente cookie door `maxAge` op `0` te plaatsen.

Het volgend codefragment zou de cookie `kleur` verwijderen:



```
var cookie = new Cookie("kleur", "");
cookie.setMaxAge(0);
cookie.setPath("/");
response.addCookie(cookie);
```



Commit de sources. Publiceer op je remote repository.



Bezocht

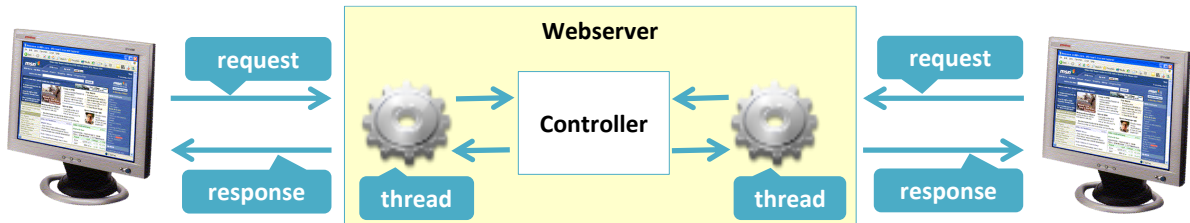
25 MULTITHREADING

Spring maakt bij de start van je website van elke controller class één bean (één object).

Men spreekt ook over een singleton bean (naar het singleton design pattern).

Spring houdt de bean bij in het RAM geheugen zolang de website draait.

Je website verwerkt elke request met een thread. 10 gelijktijdige requests zijn dus 10 threads



25.1 Thread safe

Als je, bij het verwerken van een request, in een controller een *private* variabele wijzigt, moet die thread safe zijn. Dit betekent: de variabele wijzigen door gelijktijdige threads verloopt correct.

- Primitieve variabelen (int, long, double, ...) zijn *niet* thread safe.

Volgende thread safe classes zijn dan alternatieven:

- AtomicInteger een thread safe int
- AtomicLong een thread safe long
- AtomicBoolean een thread safe boolean

- Veel classes uit de standaard Java library zijn niet thread safe:

- JDBC objecten (Connection, Statement, ResultSet, ...).
- Collection objecten (ArrayList, HashSet, HashMap, ...).

De package `java.util.concurrent` bevat thread safe alternatieven:

- CopyOnWriteArrayList een thread safe List
- CopyOnWriteArraySet een thread safe Set
- ConcurrentHashMap een thread safe Map

Lokale variabelen (die je in een method declareert), moeten niet thread-safe zijn.

25.2 Voorbeeld

Maak in `IndexController` een teller. Je houdt daarin bij hoeveel requests de controller kreeg:

```
private final AtomicInteger aantalBezoeken = new AtomicInteger(); ❶
```

- (1) De constructor van `AtomicInteger` initialiseert het getal in de `AtomicInteger` op 0.

Voeg in de method `index` code toe voor de return opdracht:

```
modelAndView.addObject("aantalBezoeken", aantalBezoeken.incrementAndGet()); ❶
```

- (1) `incrementAndGet` verhoogt de teller in de `AtomicInteger` op een thread-safe manier. De method geeft daarna de verhoogde teller terug.

Typ in `index.html` voor `</body>`

```
<div>Aantal bezoeken: <th:block th:text='${aantalBezoeken}'></th:block>.</div>
```

Je kan dit proberen.

Je simuleert meerdere gebruikers door de website in verschillende browsers te openen.



De `AtomicInteger` variabele `aantalBezoeken` gaat verloren als de website stopt. Als je op lange termijn het aantal bezoekers van een pagina wil onthouden, doe je dit beter in een database.



Commit de sources. Publiceer op je remote repository.

26 STATELESS

HTTP is een stateless protocol: de webserver vergeet, na het verwerken van een request, de data die hij in het RAM geheugen maakte tijdens het verwerken van de request.

Je moet dus bij elke request *alle* data maken die bij de request hoort.

Een beginnende ontwikkelaar ziet dit over het hoofd. Je leert dit met een voorbeeld.

Je maakt een pagina. Je toont daarin de unieke pizza prijzen als hyperlinks.

Als de gebruiker zo'n hyperlink kiest, toon je de pizza's met de gekozen prijs.

26.1 PizzaController

Maak methods:

```
private Stream<BigDecimal> findPrijzenHelper() { ❶
    return Arrays.stream(allePizzas).map(Pizza::getPrijs).distinct().sorted();
}
@GetMapping("prijzen") public ModelAndView prijzen() { ❷
    return new ModelAndView("pizzasperprijs", ❸
        "prijzen", findPrijzenHelper().iterator()); ❹
}
```

- (1) De method geeft een verzameling met de unieke pizza prijzen.
- (2) Deze method verwerkt GET requests naar de URL bij @RequestMapping ("pizzas"), gevolgd door /, gevolgd door de string hier vermeld ("prijzen"): pizzas/prijzen.
- (3) Je werkt samen met de Thymeleaf pagina pizzasperprijs.html.
- (4) Je roept de method bij ❶ op. Je geeft de return waarde door onder de naam prijzen.

26.2 pizzasperprijs.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head th:replace="fragments::head(title='Prijzen')"></head>
  <body>
    <nav th:replace="fragments::menu"></nav>
    <h1>Prijzen</h1>
    <ul>
      <li th:each="prijs:${prijzen}">
        <a th:href="@{/pizzas/prijzen/{prijs}(prijs=${prijs})}"
          th:text="${prijs}"></a>
      </li>
    </ul>
  </body>
</html>
```

- (1) Je maakt een hyperlink per prijs. Bij de prijs 4 is de hyperlink /pizzas/prijzen/4.

Voeg een menu-punt naar de pagina toe in fragments.html in het fragment menu:

```
<li><a th:href="@{/pizzas/prijzen}">Prijzen</a></li>
```

Probeer dit. Klik op een hyperlink. Je krijgt een fout.

Je hebt nog geen controller method die de request verwerkt als je op zo'n hyperlink klikt.

26.3 PizzaController

Maak methods:

```
private Stream<Pizza> findByPrijsHelper(BigDecimal prijs) {
    return Arrays.stream(allePizzas)
        .filter(pizza -> pizza.getPrijs().compareTo(prijs) == 0);
}
@GetMapping("prijzen/{prijs}")
public ModelAndView findByPrijs(@PathVariable BigDecimal prijs) {
    return new ModelAndView("pizzasperprijs",
        "pizzas", findByPrijsHelper(prijs).iterator()); ❶
}
```

- (1) Je denkt als beginnend ontwikkelaar dat je enkel de pizza's moet zoeken met de gekozen prijs. Je denkt dat je de unieke prijzen niet moet zoeken, gezien ze "al op de pagina staan".

26.4 pizzasperprijs.html

Voeg regels toe na ``:

```
<th:block th:if="${prijs}">
  <h2>Pizzas van <th:block th:text="${prijs}"></th:block> €:</h2>
  <table th:replace="fragments:pizzaTabel(pizzas=${pizzas})"></table>
</th:block>
```

26.5 Proberen

Probeer dit. Klik op een hyperlink. Je ziet de pizza's met die prijs.

Je ziet echter de hyperlinks met de prijzen niet meer!

De browser hertekent bij elke request de pagina vanaf nul met de HTML uit de response.

- ➖ De method `findByPrijs` geeft de prijzen niet door aan de Thymeleaf pagina.
- ➖ Die maakt dus ook geen HTML met prijzen.
- ➖ De browser toont dus geen prijzen.

26.6 Oplossing

Wijzig de code in method `findByPrijs`:

```
var modelAndView = new ModelAndView("pizzasperprijs", "pizzas",
    findByPrijsHelper(prijs).iterator());
modelAndView.addObject("prijzen", findPrijzenHelper().iterator());
return modelAndView;
```

❶

- (1) Je geeft naast de pizza's ook de prijzen door aan de Thymeleaf pagina.

Je kan dit proberen. Alles werkt.

26.7 Korte code

Je kan de code in de method `findByPrijs` verkorten:

```
return new
    ModelAndView("pizzasperprijs", "pizzas", findByPrijsHelper(prijs).iterator())
    .addObject("prijzen", findPrijzenHelper().iterator());
```

❶

- (1) `addObject` geeft hetzelfde `ModelAndView` object terug waarop je `addObject` opriep.

26.8 Voordelen van stateless

- ➕ Je website houdt tussen de requests geen data bij in het RAM geheugen. Veel gebruikers kunnen de website bezoeken zonder dat de webserver geheugen te kort komt.
- ➕ In een web farm mogen *verschillende* webserver requests van *eenzelfde* gebruiker verwerken. Voorbeeld: Ann kiest Prijzen in het menu. Server 1 verwerkt de request. Ann kiest een prijs. Server 2 verwerkt die request. Ann ziet een correcte pagina: Server 2 leest de prijzen én de pizza's uit de database. Hij probeert de prijzen niet uit het RAM geheugen te halen in de hoop dat hij ze daar bij een vorige request bewaarde. Dit zou niet lukken: de vorige request werd verwerkt door een Server 1 (een andere server).
- ➕ De gebruiker ziet altijd correcte prijzen. Voorbeeld: Ann ziet de prijzen. Jos voegt een pizza met een nieuwe prijs toe. Ann "refresht" haar pagina. Je website leest bij die request ook de prijs van de nieuwe pizza. Ann ziet dus ook de nieuwe prijs.



Commit de sources. Publiceer op je remote repository.



Alfabet

27 ENTERPRISE APPLICATION

Je spreekt in een grote ("enterprise") applicatie de database aan en/of je haalt je data op uit andere websites.

Als je al die code typt in de controller

- ⊖ zijn de methods lang en onoverzichtelijk: ze voeren te veel uiteenlopende taken uit.
- ⊖ herhaal je *dezelfde* code in verschillende controllers. Vb.: producten uit de database lezen

Oplossing: je typt de verschillende verantwoordelijkheden in verschillende classes.


27.1 Lagen (layers)

Men noemt soorten classes ook software lagen (layers). Je project bevat nu twee lagen:

- Presentation (controllers en Thymeleaf pagina's)
 - gegevens tonen aan de gebruiker
 - gegevens vragen aan de gebruiker
- Domain
 - de werkelijkheid voorstellen

De andere lagen:

- Repositories
- Services
- Rest clients
- Rest services.

De classes in deze andere lagen zullen (zoals controllers) Spring beans  worden.

27.2 Repository

Repositories spreken de database aan.

Ze bevatten geen transactie code. Die zit in een andere laag: services (zie verder).

Er is één repository class per domain class:

Domain class	Bijbehorende repository class	De database doet bewerkingen in verband met
Pizza	PizzaRepository	pizza's
Klant	KlantRepository	klanten

Een repository class bevat CRUD methods. CRUD staat voor Create, Read, Update en Delete.

	Handeling	Voorbeeld(en)
Create	Record toevoegen	<code>void create(Pizza pizza)</code>
Read	Record(s) lezen	<code>Optional<Pizza> findById(long id)</code> <code>List<Pizza> findByPrijsTot(BigDecimal tot)</code> <code>Optional<Pizza> findByIdAndLock(long id)</code>
Update	Record(s) wijzigen	<code>void update(Pizza pizza)</code> <code>void updatePrijs(long id, BigDecimal prijs)</code> <code>void verhoogAllePrijzen(BigDecimal percentage)</code>
Delete	Record(s) verwijderen	<code>void deleteById(long id)</code>

27.3 Rest client

Rest clients communiceren over het internet of intranet met andere applicaties.

De Fixer website (<https://fixer.io>) biedt de koers van de dollar ten opzichte van de euro aan.

Je rest client class `FixerKoersClient` leest die koers en geeft ze als een `BigDecimal` aan de je andere lagen.

FixerKoersClient
<code>+ getDollarKoers(): BigDecimal</code>

27.4 Service

Een method van een service class stelt één use case (programma onderdeel) voor, behalve de user-interface van de use case. Die zit in de presentation layer.

Een service class method roept meestal method(s) op van repository class(es). De service class method zorgt er voor dat de bijbehorende database bewerkingen tot één transactie behoren.

Er is één service class per domain class:

Domain class	Bijbehorende service class	Methods stellen use cases voor over
Pizza	PizzaService	pizza's
Klant	KlantService	klanten

27.4.1 Voorbeeld

In een bank applicatie bevat RekeningService volgende method:

```
void schrijfOver(String vanNummer, String naarNummer, BigDecimal bedrag)
```

De method schrijft geld over van de rekening met vanNummer naar de rekening met naarNummer. De method verlaagt het saldo van de “van” rekening en verhoogt het saldo van de “naar” rekening. Deze wijzigingen moeten behoren tot één transactie. De database wijzigt ofwel *beide* rekeningen of doet de wijzigingen ongedaan (bv. bij een stroompanne midden in de wijzigingen).

```
@Transactional ❶
void schrijfOver(String vanNummer, String naarNummer, BigDecimal bedrag) {
    var optionalVanRekening = rekeningRepository.findAndLockByNummer(vanNummer); ❷
    if ( ! optionalVanRekening.isPresent()) {
        throw new RekeningNietGevondenException(vanNummer);
    }
    var optionalNaarRekening = rekeningRepository.findAndLockByNummer(naarNummer);
    if ( ! optionalNaarRekening.isPresent()) {
        throw new RekeningNietGevondenException(naarNummer);
    }
    optionalVanRekening.get().schrijfOver(bedrag, optionalNaarRekening.get()); ❸
    rekeningRepository.update(vanRekening); ❹
    rekeningRepository.update(naarRekening); ❺
} ❻
```

- (1) Sprint start een transactie bij de oproep van een een method waarbij @Transactional staat. Alle database bewerkingen in die method (via oproepen van repository methods) behoren tot die transactie. Je leert hier meer over verder in de cursus.
- (2) findAndLockByNummer zoek in de database de rekening met het nummer in vanNummer. Als findAndLockByNummer de rekening niet vindt, geeft ze een lege Optional. Anders geeft ze een Rekening object met de data van de rekening in de database. Ze lockt het record tot het einde van de transactie. Je verhindert zo dat een andere gebruiker het saldo vermindert terwijl jij dat saldo controleert en vermindert. Je verhindert zo ook dat een andere gebruiker het record verwijdert tijdens je transactie.
- (3) schrijfOver verlaagt het saldo in het Rekening object waarop je de method uitvoert. De method verhoogt het saldo in het naarRekening object. Beide wijzigingen gebeuren in het RAM geheugen.

```
public void schrijfOver(BigDecimal bedrag, Rekening naarRekening) {
    if (saldo.compareTo(bedrag) < 0) {
        throw new OnvoldoendeSaldoException();
    }
    saldo = saldo.subtract(bedrag);
    naarRekening.saldo = naarRekening.saldo.add(bedrag);
}
```
- (4) update wijzigt in de database het record dat hoort bij de gewijzigde vanRekening.
- (5) Je wijzigt ook het record dat hoort bij het gewijzigde naarRekening object.
- (6) Spring commit de transactie als Spring de method kon uitvoeren zonder een exception. Bij een exception doet Spring een rollback.

27.5 Rest service

Rest services bieden je data aan andere applicaties aan, over het internet of over het intranet. Ze bieden bijvoorbeeld de pizza's in JSON formaat aan.

Je leert meer over rest services in de cursus "Spring advanced".

27.6 Samenwerking

De layers werken samen bij het uitvoeren van een use case:

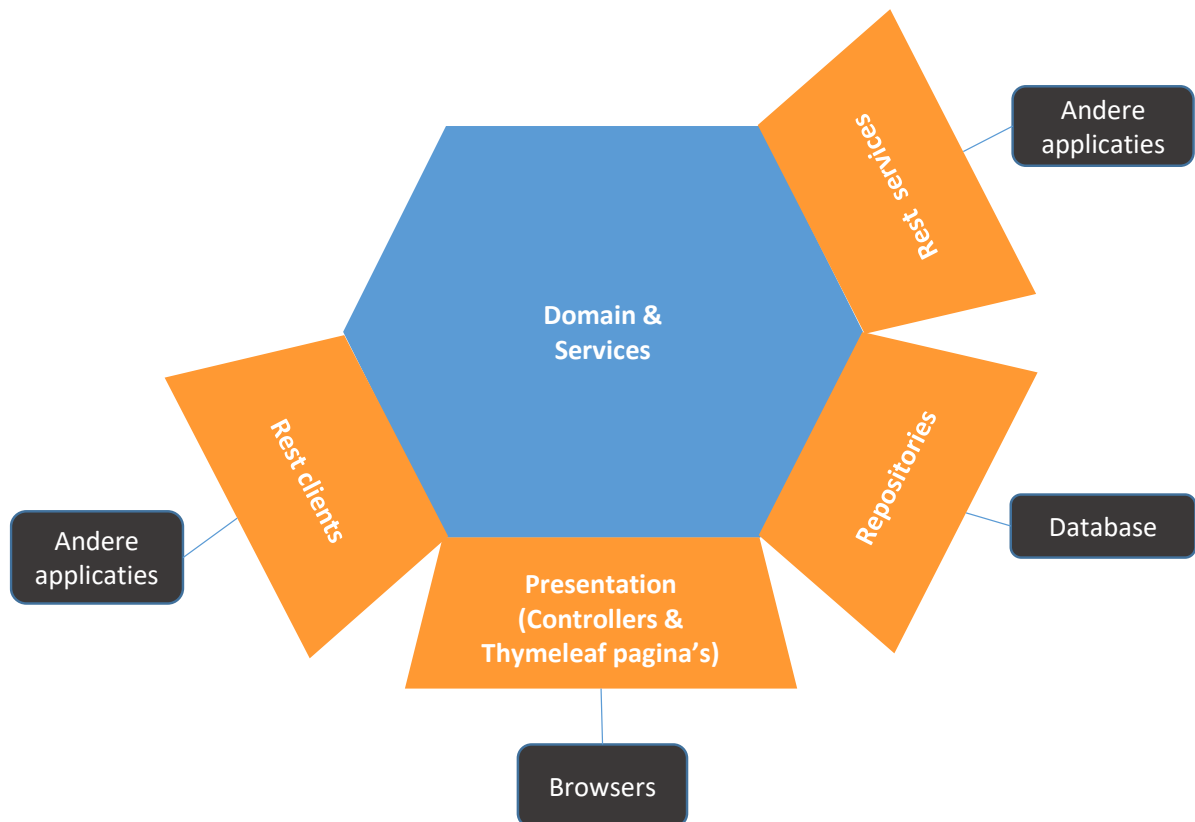
- De presentation layer roept de services layer op.
- Die roept op zijn beurt de repositories layer en/of de rest clients layer op



De pijlen zijn dependencies (afhankelijkheden).

De layer waaruit de pijl vertrekt gebruikt (is afhankelijk van) de layer waarin de pijl aankomt.

27.6.1 Ander overzicht



- De domain classes en services classes zijn de **kern layer** van je applicatie.
- Browsers communiceren via de presentation layer met de kern layer.
- De kern layer communiceert via de repositories layer met de database.
- De kern layer communiceert via de rest clients layer met andere applicaties.
- Andere applicaties communiceren via de rest services layer met de kern layer.

27.6.2 Voorbeeld

De use case "De gebruiker wil het assortiment pizza's zien":

1. De browser request van de gebruiker komt binnen in PizzaController 🌐.
2. Die roept PizzaService 🌐 op.
3. Die roept PizzaRepository 🌐 op.

28 DEPENDENCY INJECTION

28.1 Algemeen




Je leerde de basis van dependency injection in de cursus JUnit.

Je leert hier hoe je dependency injection doet met Spring.

Je gebruikt hierbij volgend voorbeeld. Je toont de data van een pizza nu al in de browser.

Je toont daarbij de prijs in euro. Je breidt dit uit: je toont de prijs ook in dollar.

Drie beans zullen daartoe samenwerken:

-  Een bean in de rest clients layer.
Je zoekt met die bean de huidige dollar koers op het internet.
-  Een bean in de services layer.
Je converteert met die bean een bedrag in euro naar dat bedrag in dollar.
De bean heeft daarbij de dollar koers nodig. De bean roept daartoe de vorige bean op.
De bean heeft dus een dependency op de vorige bean.
-  De bestaande PizzaController bean in de presentation layer.
De bean toont de prijs van de pizza in dollar. De bean roept daartoe de vorige bean op.
Deze bean heeft dus een dependency op de vorige bean.

28.2 FixerKoersClient

Je maakt een bean in de rest clients layer.

De bean zoekt de dollar koers op het internet, op de website <https://fixer.io>.

Vraag op <https://fixer.io> een free API key.

Je geeft die telkens mee als je een request doet naar de fixer website om de dollar koers te lezen.

Je zal de dollar koers lezen met een request naar de URL

http://data.fixer.io/api/latest?symbols=USD&access_key=TypHierJeFreeAPIKey

Probeer dit in de browser. Je krijgt een response in JSON formaat:

```
{ "success": true, "timestamp": 1526453227, "base": "EUR",  
  "date": "2018-05-16", "rates": { "USD": 1.183715 } }
```

❶

(1) 1.183715 is in dit voorbeeld de dollar koers.

28.2.1 KoersClientException

Je zal deze exception werpen als je de koers niet kan lezen (bvb. wegens geen internetverbinding).

Maak een package `be.vdab.luigi.exceptions`.

Maak daarin de class `KoersClientException`:

```
package be.vdab.luigi.exceptions;  
public class KoersClientException extends RuntimeException {  
    private static final long serialVersionUID = 1L;  
    public KoersClientException(String message) {  
        super(message);  
    }  
    public KoersClientException(String message, Exception oorzaak) {  
        super(message, oorzaak);  
    }  
}
```

❶

❷


- (1) Je maakt straks met deze constructor een `KoersClientException` waarbij je enkel een `String` met een omschrijving van de fout meegeeft.
- (2) Je maakt met deze constructor een `KoersClientException` waarbij je 2 dingen meegeeft:
 - a. een `String` met een omschrijving van de fout
 - b. de originele fout die de `KoersClientException` veroorzaakte.

28.2.2 FixerKoersClient

Maak een package `be.vdab.luigi.restclients`.

Maak daarin de class `FixerKoersClient`:

```
package be.vdab.luigi.restclients;
// enkele imports
@Component
public class FixerKoersClient {
    private static final Pattern PATTERN =
        Pattern.compile("^.*\\\"USD\\\": *(\\d+\\.?\\d*).*$");
    private final URL url; // uit de package java.net
    public FixerKoersClient() {
        try {
            url = new URL(
                "http://data.fixer.io/api/latest?symbols=USD&access_key=TyPHierJeFreeKey");
        } catch (MalformedURLException ex) {
            throw new KoersClientException("Fixer URL is verkeerd.");
        }
    }
    public BigDecimal getDollarKoers() {
        try (var stream = url.openStream()) {
            var matcher = PATTERN.matcher(new String(stream.readAllBytes()));
            if (!matcher.matches()) {
                throw new KoersClientException("Fixer data ongeldig");
            }
            return new BigDecimal(matcher.group(1));
        } catch (IOException ex) {
            throw new KoersClientException("Kan koers niet lezen via Fixer.", ex);
        }
    }
}
```

- (1) Je typt `@Component` voor de class. Spring maakt dan bij de start van de website een singleton bean van die class. IntelliJ toont dit met het icoon  in de marge.
- (2) De regular expression stelt de tekst `"USD":` voor, gevolgd door een getal (met de koers)
- (3) Je stuurt een request naar de Fixer website.
- (4) Je leest de volledige response.
- (5) Je maakt een `BigDecimal` op basis van het getal met de koers na `"USD":` (Het deel in de regular expression tussen ronde haken).

28.2.3 FixerKoersClientTest

Je test met JUnit de class `FixerKoersClient`.

Maak in het project onderdeel `src/test/java` een package `be.vdab.luigi.restclients`.

Maak daarin de class `FixerKoersClientTest`:

```
package be.vdab.luigi.restclients;
import static org.assertj.core.api.Assertions.assertThat;
// enkele andere imports
class FixerKoersClientTest {
    private FixerKoersClient client;
    @BeforeEach
    void beforeEach() {
        client = new FixerKoersClient();
    }
    @Test
    void deKoersIsPositief() {
        assertThat(client.getDollarKoers()).isPositive();
    }
}
```

- (1) `FixerKoersClient` werkt correct als de gelezen dollar koers een positief getal is.

Voer de test uit. Hij lukt.

28.3 EuroService

Je maakt een bean in de services layer.

De bean converteert een bedrag in euro naar dat bedrag in dollar.

De bean heeft daartoe de dollar koers nodig en heeft dus een FixerKoersClient object nodig.


De bean heeft dus een dependency op de FixerKoersClient bean.

Je maakt de class EuroService (die bij de bean hoort) eerst zonder dependency injection.

Maak een package `be.vdab.luigi.services`.

Maak daarin de class EuroService:

```
package be.vdab.luigi.services;
// enkele imports
@Service
public class EuroService {
    private final FixerKoersClient koersClient = new FixerKoersClient();
    public BigDecimal naarDollar(BigDecimal euro) {
        return euro.multiply(koersClient.getDollarKoers())
            .setScale(2, RoundingMode.HALF_UP);
    }
}
```

- (1) Je typt `@Service` voor de class. Spring maakt dan bij de start van de website een singleton bean van die class. IntelliJ toont dit met het icoon  in de marge.
- (2) Deze method krijgt een bedrag in euro als parameter binnen en geeft het geconverteerde bedrag in dollar terug als return waarde.
- (3) Je maakt een object van je class waarop je een dependency hebt: `FixerKoersClient`.
- (4) Je roept daarop de method `getDollarKoers` op tijdens je conversie van euro naar dollar.

Een JUnit test voor EuroService moet het volgende testen:

- gebruikt EuroService de juiste wiskundige bewerking: vermenigvuldigen ?
- roept EuroService de method `getDollarKoers` op van de class `FixerKoersClient` ?
- rondt EuroService het resultaat af op twee cijfers na de komma ?

Je kan die JUnit test niet goed maken: de dollar koers wijzigt voortdurend. De method `getDollarKoers` van `FixerKoersClient` geeft dus bij elke oproep een andere koers terug.

Je kan de JUnit de test dus bijvoorbeeld niet beschrijven als:

een bedrag van 2 euro wordt geconverteerd naar 2.13 dollar.


Je lost dit probleem op met dependency injection.

Vervang in EuroService de private variabele `koersClient` door:

```
private final FixerKoersClient koersClient;
public EuroService(FixerKoersClient koersClient) {
    this.koersClient = koersClient;
}
```

- (1) Je maakt een `FixerKoersClient` reference variabele, maar je initialiseert ze nog niet.
- (2) Je maakt een constructor. Die krijgt een `FixerKoersClient` object als parameter binnen.
- (3) Je onthoudt dit object in de variabele bij ❶.

De constructor (❷) zal op twee manieren worden opgeroepen:

- in de JUnit test van EuroService. Je zal de constructor oproepen en een mock (dummy) object als parameter meegeven. Die mock zal geen instance zijn van `FixerKoersClient`. Je zal de mock laten maken door Mockito (zie JUnit cursus). Deze mock zal als dollar koers een vast getal teruggeven. Je lost zo het probleem op van de steeds veranderende koers.
- bij het uitvoeren van het programma. Spring maakt een `FixerKoersClient` bean (`@Component` staat voor de class). Spring maakt ook een `EuroService` bean (`@Service` staat voor de class). Spring roept hierbij de constructor (❷) op en geeft de `FixerKoersClient` bean mee als parameter. IntelliJ toont dit met  in de marge bij ❷.

Dit is dependency injection: EuroService moet niet zelf een FixerKoersClient object maken, maar krijgt dit object (als constructor parameter) aangereikt (geïnjecteerd).

28.4 EuroServiceTest

Maak in het project onderdeel src/test/java een package be.vdab.luigi.services.

Maak daarin de class EuroServiceTest:

```
package be.vdab.luigi.services;
import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;
// enkele andere imports
@ExtendWith(MockitoExtension.class) ❶
class EuroServiceTest {
    private EuroService euroService; ❷
    @Mock
    private FixerKoersClient koersClient; ❸
    @BeforeEach
    void beforeEach() {
        System.out.println(koersClient.getClass()); ❹
        System.out.println(koersClient.getDollarKoers()); ❺
        euroService = new EuroService(koersClient); ❻
    }
    @Test
    void naarDollar() { ❼
    }
}
```

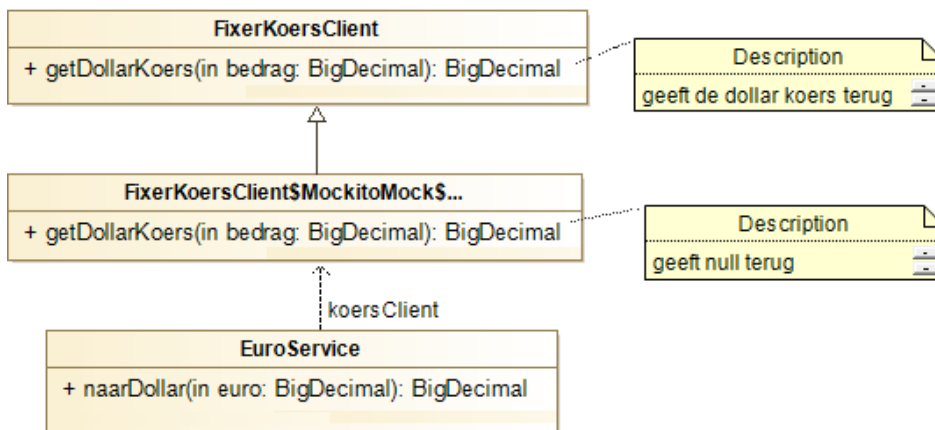
- (1) Je laat JUnit samenwerken met Mockito.
- (2) Je maakt een reference variabele voor de class die je wil testen: EuroService.
- (3) Voor de variabele koersClient staat @Mock. Mockito maakt dan een class. Die erft van de class bij die variabele: FixerKoersClient . Mockito override't in die class de method getDollarKoers: de method geeft null terug. Je lost dit straks op. Mockito vult de variabele koersClient met dit mock object.
- (4) Je voegt deze opdracht even toe om te zien dat de variabele koersClient niet verwijst naar een object van de class FixerKoersClient, maar naar een mock object van een class gemaakt door Mockito.
- (5) Je voegt deze object even toe om te zien dat de method getDollarKoers, in de class gemaakt door Mockito, null teruggeeft.
- (6) Je geeft het mock object mee aan de constructor van EuroService.
- (7) Je typt voorlopig geen code in de test method.

Voer de test uit. Je ziet:

```
class be.vdab.luigi.restclients.FixerKoersClient$MockitoMock$... ❶
null ❷
```

- (1) De variabele koersClient verwijst naar mock object van een class die Mockito maakte.
- (2) De method getDollarKoers van die class geeft null terug.

De test gebruikt dus een EuroService object, waarin je een mock object injecteert. De class van dit mock object wordt door Mockito gemaakt en erft van FixerKoersClient:



Nu je dit gezien hebt, verwijder je de `println` opdrachten uit de method `beforeEach`:

```

@BeforeEach
void beforeEach() {
    euroService = new EuroService(koersClient);
}
  
```

en voeg je code toe aan de method `naarDollar`:

```

@Test
void naarDollar() {
    when(koersClient.getDollarKoers()).thenReturn(BigDecimal.valueOf(1.1565)); ❶
    assertThat(euroService.naarDollar(BigDecimal.valueOf(3)))
        .isEqualToByComparingTo("3.47"); ❷
    verify(koersClient).getDollarKoers(); ❸
}
  
```

- (1) Je traint het mock object: de method `getDollarKoers` geeft geen `null`, maar `1.1565` terug.
- (2) Je test de `EuroService` method `naarDollar`. Die moet een bedrag van 3 euro converteren naar een bedrag van 3.47 dollar, bij een koers van 1.1565 (die de mock teruggeeft).
- (3) De code bij ❷ moet de method `getDollarKoers` van de mock hebben opgeroepen.

Voer de test uit. Hij lukt.

28.5 PizzaController

De class `PizzaController` heeft een dependency op de class `EuroService`, om een pizza prijs in euro ook te tonen als een pizza prijs in dollar.

Je doet dit ook via dependency injection:


Voeg aan de class `PizzaController` een variabele toe:

```
private final EuroService euroService;
```

IntelliJ toont een fout op deze regel. Kies daarin als oplossing `Add constructor parameter`. IntelliJ maakt dan een constructor:

```

PizzaController(EuroService euroService) {
    this.euroService = euroService;
}
  
```

- (1) Bij de start van de website roept maakt Spring een bean van de class `PizzaController`. Spring roept daarbij de constructor op. Spring geeft de `EuroService` bean mee als constructor parameter. IntelliJ toont dit met  in de marge. Als je daarop klikt, opent IntelliJ de class van deze bean: `EuroService`.

Wijzig in PizzaController de method pizza:

```
@GetMapping("{id}")
public ModelAndView findById(@PathVariable long id) {
    var modelAndView = new ModelAndView("pizza");
    findByIdHelper(id)
        .ifPresent(gevondenPizza -> {
            modelAndView.addObject(gevondenPizza);
            try {
                modelAndView.addObject(
                    "inDollar", euroService.naarDollar(gevondenPizza.getPrijs()));
            } catch (KoersClientException ex) {
                // Hier komt later code die de exception verwerkt.
            }
        });
    return modelAndView;
}
```

- (1) Je roept de method `naarDollar` van de class `EuroService` op.
 Je geeft de pizza prijs in euro mee als parameter.
 Je krijgt de prijs in dollar terug.
 Je geeft die door aan de Thymeleaf pagina onder de naam `inDollar`.

28.6 pizza.html

Voeg code toe, voor de laatste `<dt>`:

```
<dt th:if="${inDollar}">In dollar</dt>
<dd th:if="${inDollar}" th:text="${inDollar}"></dd>
```

Je kan de website proberen. In de detailpagina van een pizza zie je de prijs ook in dollar.

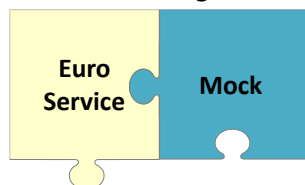
28.7 Bean

De beans van je applicatie (🌐 = bean, → = dependency):

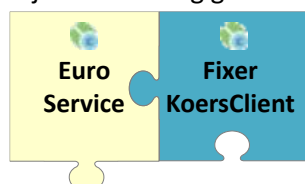
🌐 IndexController
 🌐 OSController
 🌐 KleurController
 🌐 PizzaController → 🌐 EuroService → 🌐 FixerKoersClient

28.8 Samenvatting

In de JUnit test gebruikt `EuroService` een mock (gemaakt door Mockito):



Bij de uitvoering gebruikt `EuroService` een `FixerKoersClient` object. Beiden zijn 🌐 beans:



Voer alle testen uit. Commit de sources. Publiceer op je remote repository.



Sausen.csv

29 MEERDERE IMPLEMENTATIES

Een dependency heeft soms *meerdere* implementaties. Je leert hier hoe je daarmee omgaat.

29.1 ECBKoersClient

Je maakt een tweede class die de dollar koers op het internet leest: ECBKoersClient.


Twee classes implementeren het lezen van de dollar koers: FixerKoersClient en ECBKoersClient.

ECBKoersClient leest de dollar koers op de website van de ECB (European Central Bank).

De ECB biedt op de URL <https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml> de koersen aan in XML formaat. Één van de regels bevat de koers van de dollar:

```
<Cube currency="USD" rate="1.3594"/>
```

```
package be.vdab.luigi.restclients;
// enkele imports
@Component ❶
public class ECBKoersClient {
    private final URL url; // uit package java.net
    private final XMLInputFactory factory = XMLInputFactory.newInstance(); ❷
    ECBKoersClient() {
        try {
            url = new URL(
                "https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml");
        } catch (MalformedURLException ex) {
            throw new KoersClientException("ECB URL is verkeerd.", ex);
        }
    }
    public BigDecimal getDollarKoers() {
        try (var stream = url.openStream()) {
            for (var reader = factory.createXMLStreamReader(stream); ❸
                reader.hasNext(); ) { ❹
                reader.next(); ❺
                if (reader.isStartElement()) { ❻
                    if ("USD".equals(reader.getAttributeValue(0))) { ❼
                        return new BigDecimal(reader.getAttributeValue(1)); ❽
                    }
                }
            }
            throw new KoersClientException("XML van ECB bevat geen USD."); ❾
        } catch (IOException | NumberFormatException | XMLStreamException ex) {
            throw new KoersClientException("Kan koers niet lezen via ECB.", ex);
        }
    }
}
```

- (1) Je typt `@Component` voor de class. Spring maakt dan bij de start van de website een singleton bean van de class. IntelliJ toont dit met het icoon  in de marge.
- (2) Je leest de XML bij ❷ met een `XMLStreamReader`. Je maakt die met een `XMLInputFactory`. Dit is een toepassing van het factory design pattern. Je maakt de `XMLInputFactory` met de static method `newInstance`.
- (3) Je maakt de `XMLStreamReader`. Die leest één per één de onderdelen van XML data: begintags, eindtags, commentaar, ...
- (4) `hasNext` geeft `true` zolang je onderdelen kan lezen. `hasNext` geeft `false` bij het einde van de data.
- (5) `next` leest het volgend onderdeel.
- (6) `isStartElement` geeft `true` als dit onderdeel een begintag is.
- (7) Je controleert of de inhoud van het eerste attribuut USD is.
- (8) Je leest de dollar koers in het tweede attribuut.
- (9) Je hebt alle onderdelen gelezen, maar je vond geen dollar koers.

29.2 ECBKoersClientTest

Maak een JUnit test voor de class ECBKoersClient:

```
package be.vdab.luigi.restclients;
import static org.assertj.core.api.Assertions.assertThat;
// enkele imports
class ECBKoersClientTest {
    private ECBKoersClient client;
    @BeforeEach void beforeEach() {
        client = new ECBKoersClient();
    }
    @Test void deKoersIsPositief() {
        assertThat(client.getDollarKoers()).isPositive();
    }
}
```

Voer de test uit. Hij slaagt.

29.3 Dependency injection via een interface

De bedoeling is dat je in de class EuroService de dollar koers kan lezen via ECBKoersClient in plaats van FixerKoersClient, omdat de fixer website bijvoorbeeld enkele dagen niet werkt.

Die overschakeling vereist momenteel dat je code zou wijzigen in de class EuroService: je gebruikt in die class op meerdere plaatsen het type FixerKoersClient.

Je zou ook code moeten wijzigen in de class EuroServiceTest.

Je lost dit probleem (veel code wijzigen) in stappen op:

29.3.1 Stap 1: FixerKoersClient implementeer een interface KoersClient

FixerKoersClient implementeert een nieuwe interface: KoersClient.

Die bevat de declaratie van de method getDollarKoers.

Je zou deze interface zelf kunnen maken en de class aanpassen, maar IntelliJ kan je helpen:

1. Klik in het project overzicht met de rechtermuisknop op FixerKoersClient.
2. Kies Refactor, Extract Interface.
3. Typ KoersClient bij Interface name.
4. Plaats onder in het venster een vinkje bij de method getDollarKoers():BigDecimal.
5. Kies Refactor.
6. IntelliJ stelt voor in je project variabelen, die momenteel het type FixerKoersClient hebben het type KoersClient (de nieuwe interface) te geven.
Verwijder het vinkje bij Preview usages to be changed (om een bug in IntelliJ te ontwijken) en kies Yes.

IntelliJ heeft het volgende gedaan:

- Een nieuwe interface KoersClient (in de package be.vdab.restclients).
- De class FixerKoersClient implementeer deze interface.
- In de class FixerKoersClientTest is het type van de variabele client nu KoersClient. Je verwijst in de method beforeEach die variabele naar een FixerKoersClient object. Je roept in de method deKoersIsPositief op die variabele de method getDollarKoers op. Dit is de method uit de class van het object bij de reference variabele: FixerKoersClient.
- In de class EuroServiceTest is het type van de variabele koersClient nu KoersClient. Voor de variabele staat @Mock. Mockito maakt een class die de interface KoersClient implementeert. De method getDollarKoers in die class geeft null terug. Je traint die method in EuroServiceTest echter, zodat ze een andere waarde teruggeeft.
- In de class EuroService is het type van de variabele koersClient en van de constructor parameter nu KoersClient. Spring maakt een bean van de class EuroService. Spring roept daarbij de constructor op en merkt dat die een parameter van het type KoersClient heeft. Spring biedt de bean aan die KoersClient implementeert: FixerKoersClient.

Je test of alles nog werkt:

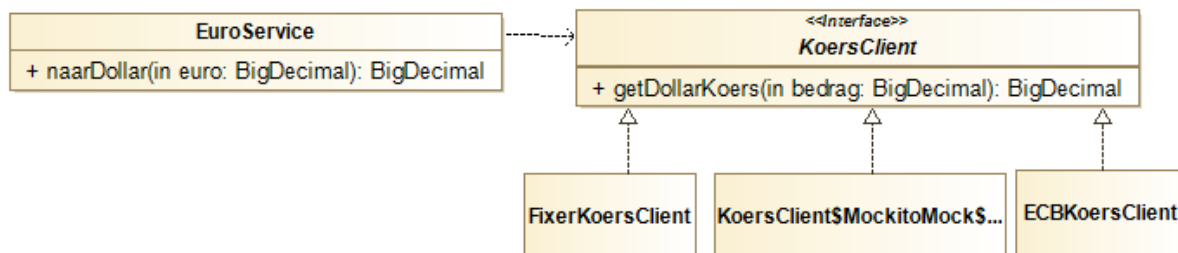
- Je voert FixerKoersClientTest uit. Die lukt.
- Je voert EuroServiceTest uit. Die lukt
- Je voert de website uit. Alles werkt.

29.3.2 Stap 2: ECBKoersClient implementeer ook de interface KoersClient

IntelliJ kan je daarbij niet helpen. Je doet zelf volgende handelingen:

1. Typ implements KoersClient na public class ECBKoersClient.
2. Typ @Override voor de method getDollarKoers.

Je moet in de testen geen code wijzigen.



29.4 @Primary

Je ziet in de class EuroService in de constructor een fout bij de parameter koersClient: Could not autowire. There is more than one bean of 'KoersClient' type.

Dit betekent: Spring kan niet kiezen welke bean hij injecteert in EuroService: fixerKoersClient of ECBKoersClient, want beide implementeren KoersClient.

Jij moet die keuze maken. Je lost dit probleem op met @Primary of met @Qualifier.

Je typt bij één van de implementaties @Primary. Spring injecteert dan die implementatie. Typ @Primary voor de class ECBKoersClient. De fout in de class EuroService verdwijnt.

29.5 @Qualifier

@Qualifier is een alternatief voor @Primary. Verwijder daarom @Primary in ECBKoersClient.

@Qualifier lost het probleem in 2 stappen op:

29.5.1 @Qualifier bij de bean classes

Je typt @Qualifier voor elke class, die dezelfde interface implementeert.

Je typt bij @Qualifier een unieke string.

- Typ @Qualifier("Fixer") voor de class FixerKoersClient.
- Typ @Qualifier("ECB") voor de class ECBKoersClient.

29.5.2 @Qualifier bij constructor injection

Je typt @Qualifier ook bij een constructor waarin je één van die beans injecteert.

Je typt bij @Qualifier een string. Spring injecteert de bean met een @Qualifier met die string.

Doe dit in de EuroService constructor:

```
public EuroService(@Qualifier("ECB") KoersClient koersClient) {
    this.koersClient = koersClient;
}
```

(1) Je typt @Qualifier vóór de te injecteren parameter. Je geeft "ECB" mee.

Spring injecteert de bean die de interface KoersClient implementeert én voorzien is van @Qualifier("ECB"). Dit is de bean van de class ECBKoersClient.

Je kan dit proberen.

Kies  voor de EuroService constructor.

IntelliJ opent de source van de bean die Spring in de constructor injecteert: ECBKoersClient.

29.6 Alle dependencies injecteren

Je injecteert tot nu in EuroService één dependency.

Een andere strategie is *alle* dependencies injecteren.

Je injecteert de dependencies FixerKoersClient én ECBKoersClient.

Je probeert de dollar koers te lezen via de eerste dependency.

Als dit mislukt (omdat de website die de dollar koers geeft uitgevallen is), probeer je de dollar koers te lezen via de tweede dependency.

29.6.1 EuroService

Wijzig de private variabele en de constructor:

```
private final KoersClient[] koersClients;           ❶
public EuroService(KoersClient[] koersClients) {    ❷
    this.koersClients = koersClients;
}
```

- (1) Deze array zal alle dependencies, die KoersClient implementeren, bevatten.
- (2) Spring injecteert alle dependencies die KoersClient implementeren als een array. Die bevat een FixerKoersClient bean en een ECBKoersClient bean.

Gebruik de dependencies in de method naarDollar:

```
@Override
public BigDecimal naarDollar(BigDecimal euro) {
    Exception laatste = null;
    for (var client : koersClients) {                ❶
        try {
            return euro.multiply(client.getDollarKoers())
                .setScale(2, RoundingMode.HALF_UP);    ❷
        } catch (KoersClientException ex) {
            laatste = ex;
        }
    }
    throw new KoersClientException("Kan dollar koers nergens lezen.", laatste); ❸
}
```

- (1) Je itereert over de dependencies.
- (2) Je probeert getDollarKoers op te roepen. Als dit lukt, gebruik je de gelezen waarde. Gezien je hier een return doet, stopt ook de iteratie bij ❶.
- (3) Alle dependencies hebben een fout geworpen. Je werpt zelf een fout. Je geeft de exception, die de laatst geprobeerde dependency gaf, mee als parameter.

29.6.2 EuroServiceTest

Je hebt een compiler fout: je roept in de method beforeEach de EuroService constructor op.

Je geeft één KoersClient object mee.

De constructor verwacht nu echter een *array* met KoersClient objecten.

Wijzig de laatste opdracht in de method beforeEach:

```
euroService = new EuroService(new KoersClient[] {koersClient});
```

Voer de test uit. Hij lukt.

29.7 Package visibility

Je roept de classes FixerKoersClient en ECBKoersClient niet meer op buiten hun eigen package. Je geeft ze package visibility, zodat je ze ook niet *per ongeluk* buiten hun package oproept: verwijder het keyword public voor beide classes en voor hun constructor.

29.8 Samenvatting

29.8.1 Je hebt van een dependency meerdere implementaties.

Je maakt een interface (KoersClient) waarin je de methods van die dependency declareert.

Alle implementaties (FixerKoersClient, ECBKoersClient) implementeren die interface.

Je gebruikt een variabele (en constructor parameter) van die interface als je de dependency injecteert in een andere class (EuroService)

29.8.2 Je hebt van een dependency slechts één implementatie

Je moet je geen interface maken, enkel een implementatie class (EuroService).

Je gebruikt een variabele (en constructor parameter) van deze implementatie class als je de dependency injecteert in een andere class (PizzaController).



Sommige programmeurs maken ook een interface als een dependency slechts één implementatie heeft. Ze bekomen zo 'low coupling': als een class A via een interface B een dependency heeft op een class C, komt de class C nergens voor in de code van class A. A en C kunnen zo optimaal los van mekaar aangepast en uitgebreid worden. Dit is bvb. interessant als een programmeur A maakt en een andere programmeur C.

29.9 IOC – IOC container

Dependency injection is een toepassing van IOC: inversion of control.

Hierbij doe je in een class een stukje werk *niet in die class zelf*.

Je draait de controle om (you invert the control) : *je laat dit werk door Spring doen*.

- ➖ PizzaController maakt het EuroService object *niet zelf*:
`this.euroService = new EuroService(...);`
- ➖ PizzaController maakt dit ook *niet* via een factory (design pattern):
`this.euroService = EuroServiceFactory.INSTANCE.getEuroService();`
- ➕ IOC: PizzaController laat dit object maken door Spring. Die maakt een EuroService bean en geeft het aan PizzaController als parameter van de PizzaController constructor.

De IOC container is de verzameling beans van je applicatie.

29.10 Als een dependency ontbreekt

Als je vergeet @Service bij EuroService te typen, maakt Spring geen bean van die class.

Spring heeft dan ook geen bean om te injecteren in PizzaController.

Spring werpt dan een exception bij de start van de website.

Probeer dit uit. Plaats @Service in commentaar in EuroService.

Start de website. Je krijgt een exception met als omschrijving:

Parameter 0 of constructor in be.vdab.luigi.web.PizzaController

required a bean of type 'be.vdab.services.EuroService' that could not be found.

Corrigeer: haal @Service terug uit commentaar in EuroService.

29.11 @Controller, @Service, @Component, @Repository

Overzicht:

- @Controller Een controller class. Methods van die class verwerken browser requests.
- @Service Een service class. Methods van die class stellen use-cases voor.
- @Repository Een repository class. Methods van die class spreken de database aan.
- @Component Een class die geen controller, service of repository is, maar waarvan je wel wil dat Spring er een bean van maakt. Je kan deze bean dan injecteren in andere beans.



Commit de sources. Publiceer op je remote repository.



Sauzen.properties

30 APPLICATION.PROPERTIES

De URL's van de Fixer website en van de ECB website zijn nu hard gecodeerd in je sources.

Dit is onhandig: zo'n URL kan wijzigen nadat je website af is.

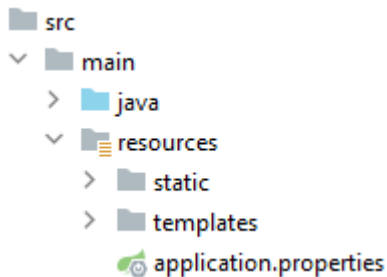
De website beheerder moet dan de Java source openen, wijzigen en compileren.

Dit is moeilijk als die persoon geen Java kent, of geen Java compiler bij de hand heeft.

Je typt de URL's beter in application.properties (het configuratiebestand van je applicatie).

Je moet je applicatie niet hercompileren als je daarin een instelling wijzigt.

Je vindt application.properties in src/main/resources:



30.1 application.properties

Voeg regels toe:

```
ecbKoersURL=https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml
fixerKoersURL=http://data.fixer.io/api/latest?symbols=USD&access_key=TypHierJeFreeKey
```

- (1) Één regel bevat één configuratie instelling.
Elke regel bevat de naam van de instelling (ecbKoersURL), een = teken
en een waarde voor de instelling (https://www.ecb.europa.eu/...).

30.2 FixerKoersClient

Je leest in application.properties de instelling met de key fixerKoersURL.

Wijzig de constructor:

```
FixerKoersClient(@Value("${fixerKoersURL}") URL url) {
    this.url = url;
}
```

- (1) De constructor heeft een parameter. Daarvoor staat @Value. Daarbij staat fixerKoersURL: de naam van de te lezen instelling in application.properties.
Spring injecteert dan de waarde van die instelling in die parameter.

De constructor is eenvoudiger dan vroeger.

- ⊕ Het is niet meer jij, maar Spring die een URL maakt op basis van een stukje tekst uit application.properties).
- ⊕ Het is niet meer jij, maar Spring die de MalformedURLException opvangt als Spring de tekst niet kan omzetten naar een URL. Je applicatie start dan niet.

30.3 ECBKoersClient

Wijzig de constructor:

```
ECBKoersClient(@Value("${ecbKoersURL}") URL url) {
    this.url = url;
}
```

FixerKoersClientTest en ECBKoersClientTest bevatten nu fouten.

Je lost dit op in het volgende hoofdstuk.

Je kan de website door die fouten nog niet proberen.

31 BEAN TESTEN

FixerKoersClientTest bevat een fout bij de oproep van de FixerKoersClient constructor. De constructor verwacht een parameter met de Fixer URL. Je geeft geen waarde mee.

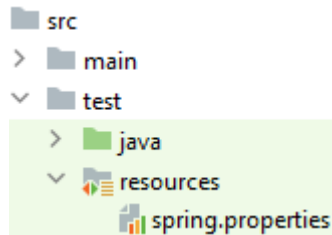
Het is jammer de URL te typen in de unit test: hij zit al in `application.properties`.

Een betere oplossing is Spring te laten samenwerken met JUnit.

31.1 spring.properties

Maak in `src/test/` een directory `resources`

Maak daarin een bestand `spring.properties`:



Typ in dit bestand:

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

❶

(1) Hiermee kan je, in de constructor van een test, dependency injection van beans doen.

31.2 FixerKoersClientTest

Typ regels voor de class:

```
@ExtendWith(SpringExtension.class)
@PropertySource("application.properties")
@Import(FixerKoersClient.class)
```

❶

❸

❷

(1) JUnit en Spring werken nu samen. Spring maakt een lege IOC container ten dienste van de test.

(2) Je geeft aan dat Spring instellingen moet lezen in `application.properties`.

(3) Spring maakt een `FixerKoersClient` bean in de IOC container. Dit is de te testen bean.

Verwijder de method `beforeEach`.

Maak de variabele `client` `final`. Laat IntelliJ een constructor maken.

Spring zal die gebruiken om de `FixerKoersClient` bean te injecteren in de test.

Voer de test uit. Hij lukt.

31.3 ECBKoersClientTest

Typ regels voor de class:

```
@ExtendWith(SpringExtension.class)
@PropertySource("application.properties")
@Import(ECBKoersClient.class)
```

Verwijder de method `beforeEach`.

Maak de variabele `client` `final`. Laat IntelliJ een constructor maken.

Spring zal die gebruiken om de `ECBKoersClient` bean te injecteren in de test.

Voer de test uit. Hij lukt.

Je kan de website proberen.



Voer alle tests uit. Commit de sources. Publiceer op je remote repository.



`application.properties`

32 LOGGING

In de class `PizzaController` vangt de method `pizza` een `KoersClientException` op. De method doet daarmee niets. De applicatie beheerder weet dus niet dat die is opgetreden. Hij kan dan de situatie dus ook niet verbeteren.

Je lost dit op. Je logt de exception. Je toont daarbij informatie over de exception in de console.

Voeg een private variabele toe:

```
private final Logger logger = LoggerFactory.getLogger(this.getClass()); ❶
// importeer Logger en LoggerFactory uit org.slf4j
```

- (1) Je logt met een `Logger`. Je krijgt die van een `LoggerFactory` (factory design pattern). `getLogger` geeft je een `Logger`. Je geeft de huidige class als parameter mee.

Typ volgende code in het catch block in de method `findById`:

```
logger.error("Kan dollar koers niet lezen", ex); ❶
```

- (1) De method `error` logt een exception.
De eerste parameter is een beschrijving van de fout.
De tweede parameter is de exception.

`FixerKoersClient` én `ECBKoersClient` moet even een exception werpen om dit te proberen.

Je maakt daartoe fouten in `application.properties`:

1. Wijzig bij de instelling `ecbKoersClient` in de URL `ecb` naar `ecp`.
2. Wijzig bij de instelling `fixerKoersClient` in de URL `fixer` naar `vixer`.

32.1 Gelogde data

Start de applicatie. Open de detailpagina van een pizza.

Je ziet de gelogde data in het console venster (in IntelliJ):

1. Je ziet als laatste een `UnknownHostException: data.vixer.io`
2. Je scrolt naar boven. Je ziet dat de exception leidde tot een `KoersClientException: Kan koers niet lezen via Fixer`.
3. Je scrolt verder. Je ziet dat de exception leidde tot een `KoersClientException: Kan koers nergens lezen`.

32.2 Naar een bestand loggen

Als de applicatie stopt, verlies je de logging data die je toonde in de console.

Er is een oplossing. Voeg volgende regel toe aan `application.properties`:

```
logging.file.name=/logging/log.txt ❶
```

- (1) Spring schrijft de logging data ook naar het bestand `log.txt` in de folder `logging` in de root van je hard disk. Dit is interessant. De applicatie beheerder ziet in dit bestand de historiek van de fouten die gebeurden in de applicatie, ook als de applicatie stopt.

Je kan dit proberen.

Corrigeer daarna de fouten die je maakte in `application.properties`.

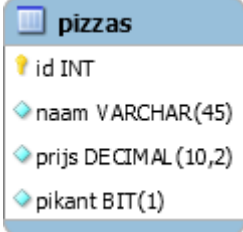
33 DATABASE

33.1 MySQL Workbench

Je voert in de MySQL Workbench het script `luigi.sql` uit.
Dit maakt een database `luigi`. Die bevat een table `pizzas`.

- `id` is een auto increment kolom.
- `bit` is een synoniem voor boolean.

De gebruiker `cursist` heeft in de table de rechten `select` en `insert`.



pizzas	
id	INT
naam	VARCHAR(45)
prijs	DECIMAL(10,2)
pikant	BIT(1)

33.2 IntelliJ

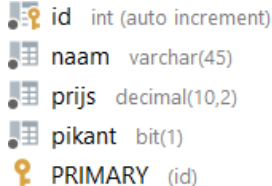
Je kan met IntelliJ een database inzien en SQL statements testen.

33.2.1 Verbinding maken

- (1) Kies **+** links boven in het venster Database.
- (2) Kies **Data Source, MySQL**
- (3) Typ `luigi` bij **Name**.
- (4) Typ `root` bij **User**.
- (5) Typ het paswoord van de gebruiker `root` bij **Password**.
- (6) Typ `luigi` bij **Database**.
- (7) Kies **Test Connection**.
- (8) Kies eventueel **Download Driver Files**.
- (9) Kies **OK**.

33.2.2 Database structuur

- (1) Kies **>** voor `luigi`.
- (2) Kies **>** voor `schemas`.
- (3) Kies **>** voor `luigi`.
- (4) Kies **>** voor `pizzas`.



id	int (auto increment)
naam	varchar(45)
prijs	decimal(10,2)
pikant	bit(1)
PRIMARY	(id)

Je ziet de kolommen en de primary key.

33.2.3 SQL statement uitvoeren

Typ een SQL statement in het middelste venster in IntelliJ:


```
select * from pizzas
```

Kies . IntelliJ voert het SQL statement uit en toont het resultaat.

Je kan in dit resultaat zelfs een waarde wijzigen en vastleggen met .

Typ een SQL statement met een parameter:

```
select * from pizzas where id = ?
```

Kies . Je ziet een venster. Typ een waarde voor de parameter. Druk **Enter**. Kies **Execute**.

33.2.4 Andere database merken

Je kan met IntelliJ ook databases van andere merken openen: PostgreSQL, Oracle, ...



Database toegang

34 DATASOURCE


34.1 pom.xml

Voeg de dependency toe voor de MySQL JDBC driver:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

Voeg de dependency toe voor de Spring JDBC library. Je spreekt er JDBC mee aan met weinig code.

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

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

Opmerking: je kan deze dependencies al toevoegen als je een nieuw project maakt in IntelliJ. Je kiest dan links de categorie SQL en rechts de dependencies MYSQL Driver en JDBC API.

34.2 application.properties

Voeg regels toe:

```
spring.datasource.url=jdbc:mysql://localhost/luigi
spring.datasource.username=cursist
spring.datasource.password=cursist
```

①
②
③


- (1) De JDBC URL die verwijst naar de database die je applicatie gebruikt.
- (2) De database gebruiker waarmee je applicatie de database opent.
- (3) Het paswoord van de gebruiker bij ②.

34.3 DataSource

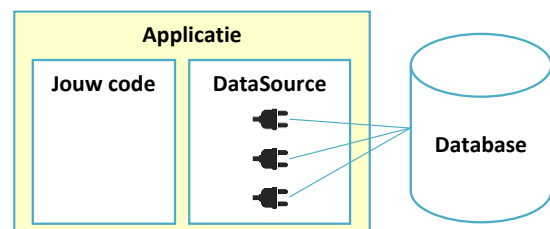
Je hebt een JDBC Connection nodig om een database bewerking uit te voeren.

Bij eenvoudige SQL statements vraagt het openen van een Connection meer tijd dan het uitvoeren van het SQL statement zelf. Je wint dus tijd door bij een SQL statement geen nieuwe Connection te openen, maar een vroeger geopende Connection te hergebruiken.

JDBC bevat hiervoor het type DataSource (synoniem: connection pool).

Die houdt een verzameling Connections () continu open.

Als je een Connection nodig hebt, vraag je de DataSource één van zijn Connections.

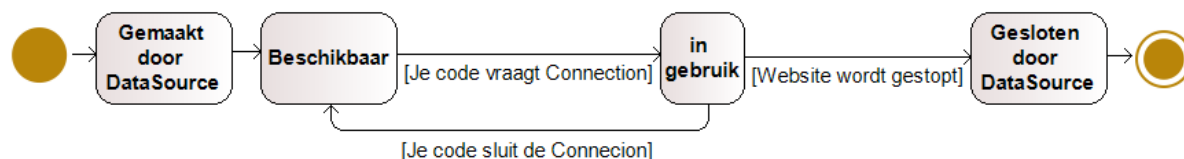


1. Spring maakt de DataSource bij de start van de website.
De DataSource opent enkele Connections naar de database en laat ze open staan.
2. Een browser request komt binnen. Je hebt de database nodig om de request te verwerken.
Je vraagt een Connection aan de DataSource. Je krijgt de Connection zeer snel: ze was al open. De DataSource onthoudt dat de Connection in gebruik is.
3. Terwijl de webserver de request verwerkt, komt een 2° request binnen.
De webserver verwerkt die request met een andere thread dan de 1° request.
Je vraagt voor die 2° request een Connection aan de DataSource.
Die zoekt een Connection die niet in gebruik is en geeft je die Connection.
Hij onthoudt dat die Connection ook in gebruik is.
4. Je sluit de Connection waarmee je de 1° request verwerkt. De DataSource onderschept dit sluiten en houdt de Connection open. Hij onthoudt dat de Connection niet meer in gebruik is.
De Connection is daarna beschikbaar als je code de database later terug nodig heeft.

Spring maakt een DataSource bean zodra je de dependency spring-boot-starter-jdbc hebt. Je voegde die toe in het begin van dit hoofdstuk.

De DataSource maakt verbindingen naar de database beschreven in de regels die je in dit hoofdstuk toevoegde aan application.properties.

Levensfasen van een Connection in de DataSource:



34.4 Test

Je test of de DataSource database connecties kan maken.

34.4.1 application.properties

Voeg een regel toe:

`spring.test.database.replace=none` ❶

- (1) Spring maakt in een test een DataSource die verwijst naar een database van het merk H2, niet naar de MySQL database gedefinieerd in application.properties. H2 houdt de data bij in het RAM geheugen, niet op de harde schijf. H2 reageert bij het uitvoeren van een SQL statement soms anders dan MySQL. Je vraagt daarom de MySQL database *niet* te vervangen door een H2 database.

34.4.2 DataSourceTest

Maak in src/test/java een package be.vdab.luigi.repositories. Maak daarin de test:

```

package be.vdab.luigi.repositories;
import static org.assertj.core.api.Assertions.assertThat;
// enkele andere imports
@JdbcTest
class DataSourceTest {
    private final DataSource dataSource;
    DataSourceTest(DataSource dataSource) {
        this.dataSource = dataSource;
    }
    @Test
    void getConnection() throws SQLException {
        try (var connection = dataSource.getConnection()) {
            assertThat(connection.getCatalog()).isEqualTo("luigi");
        }
    }
}
  
```

❶
❷
❸
❹

- (1) @JdbcTest maakt een IOC container met een DataSource bean ten dienste van de test.
- (2) Je injecteert de DataSource bean.
- (3) Je vraagt een Connection aan de DataSource.
- (4) getCatalog geeft de naam van de database waarmee de Connection verbonden is.

Probeer dit. Typ even een fout in application.properties: vervang jdbc door jbc.

Voer de test uit. Hij mislukt.

Corrigeer application.properties: vervang jbc door jdbc. Voer de test uit. Hij lukt.



Commit de sources. Publiceer op je remote repository.



DataSource

35 EXCEPTION BIJ DATABASE TOEGANG

Bij het aanspreken van de database kunnen twee soorten exceptions optreden:

- **Exceptions die je kan verhelpen zonder de applicatie te stoppen.**
Voorbeeld: je probeert een record te wijzigen, maar het record is gelockt.
Je wacht een aantal milliseconden. Je probeert het record opnieuw te wijzigen.
Waarschijnlijk is het record ondertussen niet meer gelockt.
Je kan dit “opnieuw proberen” zelfs een aantal keer herhalen.
- **Fatale exceptions.**
Voorbeeld: je SQL statement bevat een syntaxfout.

JDBC maakt jammer genoeg geen verschil tussen beide soorten: alle exceptions zijn SQLException.

SQLException bevat een property errorCode. Die geeft met een getal de reden van de fout aan.
Jammer genoeg verschillen de getallen per merk databases.

Spring vertaalt intern, aan de hand van de errorCode én het merk van de database, SQLExceptions naar specifiekere exceptions en werpt die naar jouw code.
Jij kan zo onderscheid maken tussen beide soorten exceptions.

35.1 Belangrijkste Spring verhelpbare database exceptions

- DuplicateKeyException
Een insert of een update ging verkeerd:
je probeerde een dubbele waarde te maken op een unieke index.
- DataIntegrityViolationException
Een insert of een update veroorzaakte een fout op een database constraint.
Voorbeeld: je liet een verplicht in te vullen kolom leeg.
- CannotAcquireLockException
Een record is gelockt door een andere applicatie.

35.2 Belangrijkste Spring fatale database exceptions

- BadSqlGrammarException
Je SQL statement bevat een syntaxfout.
- InvalidResultSetAccessException
Je probeert in een ResultSet een kolom te lezen die niet bestaat.
- TypeMismatchDataException
Een Java type komt niet overeen met een database type.
Je vult bijvoorbeeld een date kolom met een double.
- PermissionDeniedDataAccessException
Je hebt in de database onvoldoende rechten om de handeling uit te voeren.

35.3 PizzaNietGevondenException

Je werpt deze exception straks als de gebruiker een pizza vraagt die je niet vindt in de database.

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

36 REPOSITORY

Een repository heeft in je applicatie maar één implementatie.
Je moet daarom geen interface maken voor een repository.

36.1 JdbcTemplate

Je zou in een repository class de database kunnen aanspreken met JDBC.

Je moet dan in elke repository method soortgelijk werk doen:

- Een Connection openen.
- Een PreparedStatement maken.
- Eventueel parameters van dit PreparedStatement invullen.
- Het PreparedStatement uitvoeren.
- Eventueel itereren over de ResultSet (als je data leest).

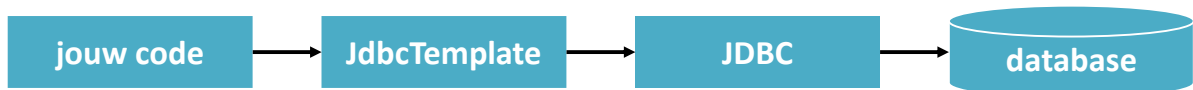
Na een tijdje wordt het saai in elke repository method dit soortgelijk werk te herhalen.

Spring bevat een class JdbcTemplate die dit probleem oplost.

Jij zal aan een method van JdbcTemplate een String met een SQL statement meegeven.

De method doet het bovenstaande werk voor jou. Je leert dit hier in detail kennen.

JdbcTemplate laat dus toe de database aan te spreken met minder code dan met JDBC.



Spring maakt bij de start van je website een JdbcTemplate bean.

Spring doet dit zodra pom.xml de JDBC dependency bevat.

JdbcTemplate heeft een interne dependency op DataSource:

wanneer JdbcTemplate een database connectie nodig heeft, vraag hij die aan DataSource.

🟢 JdbcTemplate → 🟢 DataSource

36.2 PizzaRepository

Maak een package be.vdab.luigi.repositories.

Maak daarin een class PizzaRepository.

Gezien je van een repository maar één implementatie hebt, maak je geen bijbehorende interface.

`package` be.vdab.luigi.repositories;

// enkele imports

`@Repository`


```

public class PizzaRepository {
    private final JdbcTemplate template;
    public PizzaRepository(JdbcTemplate template) {
        this.template = template;
    }
}
  
```

❶

❷

❸

(1) Je typt `@Repository` voor de class. Spring maakt bij de start van de website een singleton bean van die class. IntelliJ toont dit met het icoon  in de marge.

(2) Je injecteert de JdbcTemplate bean.

(3) Je onthoudt die bean in een private variabele.

Je kan zo straks in methods van PizzaRepository deze bean oproepen.

36.3 Scalar value

Een scalar value is het resultaat van een select statement met 1 rij en 1 kolom. Voorbeelden:

- `select count(*) from pizzas`
- `select max(prijs) from pizzas`
- `select prijs from pizzas where id = 1`

Je voegt aan `PizzaRepository` een method toe die het aantal pizza's teruggeeft.

Je leest dit aantal als een scalar value met de `JdbcTemplate` method `queryForObject`:

```
public long findAantal() {
    var sql = """
        select count(*)
        from pizzas
        """;
    return template.queryForObject(sql, Long.class);
}
```

❶

- (1) De 1° parameter is het select statement die een scalar value teruggeeft.
- (2) De 2° parameter is het type dat je wenst als return waarde van `queryForObject`.

Je kan het SQL statement dat je typte bij ❶ testen:

1. Klik met de rechtermuisknop in het SQL statement.
2. Kies Execute.
3. Kies New Session. Je ziet het resultaat van het SQL statement onder in IntelliJ.

36.4 Update of delete SQL statement

Je voegt aan `PizzaRepository` een method toe om een pizza te verwijderen.

Je voert een update of delete SQL statement uit met de `JdbcTemplate` method `update`:

```
public void delete(long id) {
    var sql = """
        delete from pizzas
        where id = ?
        """;
    template.update(sql, id);
}
```

❶

- (1) De 1° parameter is het SQL statement.
De 2° parameter is de waarde voor het ? in het SQL statement.

36.5 Meerdere parameters

Je voegt aan `PizzaRepository` een method toe om een pizza te wijzigen.

```
public void update(Pizza pizza) {
    var sql = """
        update pizzas
        set naam = ?, prijs = ?, pikant = ?
        where id = ?
        """;
    if (template.update(sql,
        pizza.getNaam(), pizza.getPrijs(), pizza.isPikant(), pizza.getId())
        == 0) {
        throw new PizzaNietGevondenException();
    }
}
```

❶

- (1) `update` geeft het aantal aangepaste records.
Als dit aantal 0 is, heb je de te wijzigen pizza niet gevonden.

36.6 Record toevoegen

Je voegt een record toe met de class SimpleJdbcInsert.

Voeg een SimpleJdbcInsert variabele toe:

```
private final SimpleJdbcInsert insert;
```

Voeg code toe aan de constructor:

```
insert = new SimpleJdbcInsert(template)           ❶
    .withTableName("pizzas")                     ❷
    .usingGeneratedKeyColumns("id");              ❸
```

- (1) Je geeft de JdbcTemplate mee aan de SimpleJdbcInsert constructor.
De JdbcTemplate is geïnjecteerd met een DataSource.
SimpleJdbcInsert zal een record toevoegen met een connectie uit die DataSource.
- (2) Je definieert de naam van de table waarin je records wil toevoegen.
- (3) Als de table een automatisch gegenereerde primary key kolom bevat, vermeld je de naam van die kolom.

Maak de method create:

```
public long create(Pizza pizza) {
    return insert.executeAndReturnKey(           ❶
        Map.of("naam", pizza.getNaam(),         ❷
               "prijs", pizza.getPrijs(),
               "pikant", pizza.isPikant()))
    .longValue();                               ❸
}
```

- (1) executeAndReturnKey voegt een record toe.
- (2) De parameter is een Map.
Die bevat een entry per kolom van het toe te voegen record.
De key is een kolom naam.
De value is de waarde die je invult in die kolom.
De method maakt zelf een SQL insert statement en voert dit uit.
- (3) De method geeft de automatisch gegenereerde primary key waarde als een Number.
Jij neemt daarvan de long waarde.

36.7 RowMapper

Een RowMapper implementeert de interface RowMapper. Die bevat één method: mapRow.

Die maakt een (bij ons Pizza) object per rij in een ResultSet.

JdbcTemplate gebruikt een RowMapper bij het lezen van records: JdbcTemplate maakt op basis van elke record een Pizza object en verzamelt die objecten in een List.

Je kan RowMapper implementeren met een lambda:

```
private final RowMapper<Pizza> pizzaMapper =    ❶
    (result, rowNum) ->                        ❷
    new Pizza(result.getLong("id"), result.getString("naam"), ❸
               result.getBigDecimal("prijs"), result.getBoolean("pikant"));
```

- (1) Je typt tussen <> het type object dat RowMapper moet maken op basis van een ResultSet rij.
- (2) Je lambda heeft de parameters in de method mapRow.
 - a. De 1° parameter is de ResultSet.
 - b. De 2° parameter is het volgnummer van de huidige rij in de ResultSet.
- (3) De return waarde is de Pizza die je maakt op basis van de huidige rij in de ResultSet.

36.8 Meerdere records lezen

Je voegt aan `PizzaRepository` een method toe die alle pizza's leest:

```
public List<Pizza> findAll() {
    var sql = """
        select id, naam, prijs, pikant
        from pizzas
        order by id
        """;
    return template.query(sql, pizzaMapper);
}
```

❶

- (1) query voert volgende stappen uit:
- een lege `List<Pizza>` maken.
 - een `Connection` vragen aan de `DataSource` bean.
 - een `Statement` met het select statement maken en uitvoeren.
 - itereren over de `ResultSet` van het resultaat van het `Statement`.
 - per rij je lambda uitvoeren.
 - de `Pizza`, die je lambda teruggeeft, toevoegen aan de `List`.
 - de `ResultSet`, het `Statement` en de `Connection` sluiten.
 - de `List<Pizza>` teruggeven.

De `query` method heeft ook een versie waarbij je een SQL statement met parameters uitvoert.

Je voegt aan `PizzaRepository` een method toe die de pizza's leest met een prijs tussen 2 grenzen:

```
public List<Pizza> findByPrijsBetween(BigDecimal van, BigDecimal tot) {
    var sql = """
        select id, naam, prijs, pikant
        from pizzas
        where prijs between ? and ?
        order by prijs
        """;
    return template.query(sql, pizzaMapper, van, tot);
}
```

❶

- (1) De 1° parameter is een select statement. De 2° parameter is een `RowMapper`. De volgende parameters zijn waarden voor de parameters in het select statement (aangegeven met `?`).

36.9 Één record lezen

Je leest met de `JdbcTemplate` method `queryForObject` één record als van een Java object.

De method werpt een `IncorrectResultSizeDataAccessException` als het select statement geen record vindt, of meer dan één record.

Je voegt aan `PizzaRepository` een method toe die één pizza leest waarvan je de id kent:

```
public Optional<Pizza> findById(long id) {
    try {
        var sql = """
            select id, naam, prijs, pikant
            from pizzas
            where id = ?
            """;
        return Optional.of(template.queryForObject(sql, pizzaMapper, id));
    } catch (IncorrectResultSizeDataAccessException ex) {
        return Optional.empty();
    }
}
```

❶

- (1) `queryForObject` vond geen record. Je geeft dan een "lege" `Optional` terug.

36.10 Resterende methods

```
private final RowMapper<BigDecimal> prijsMapper =
    (result, rowNum) -> result.getBigDecimal("prijs");
```

Een method die de unieke prijzen leest:

```
public List<BigDecimal> findUniekePrijzen() {
    var sql = """
        select distinct prijs
        from pizzas
        order by prijs
        """;
    return template.query(sql, prijsMapper);
}
```

Een method die de pizza's met een bepaalde prijs leest:

```
public List<Pizza> findByPrijs(BigDecimal prijs) {
    var sql = """
        select id, naam, prijs, pikant
        from pizzas
        where prijs = ?
        order by naam
        """;
    return template.query(sql, pizzaMapper, prijs);
}
```

Een method die de pizza's met bepaalde id's leest:

```
public List<Pizza> findByIds(Set<Long> ids) {
    if (ids.isEmpty()) {
        return List.of();
    }
    var sql = """
        select id, naam, prijs, pikant
        from pizzas
        where id in (
            """+
            + "?, ".repeat(ids.size() - 1)
            + "?)" + " order by id";
    return template.query(sql, pizzaMapper, ids.toArray());
}
```

①
②

③
④
⑤
⑥

- (1) Als de verzameling id's leeg is
- (2) spreek je de database niet aan. Je geeft een lege verzameling pizza's terug.
- (3) Dit is het *begin* van een SQL statement dat pizza's zoekt waarvan enkele id's gekend zijn.
- (4) Je voegt aan het SQL statement een ? en een , toe per op te zoeken id, behalve de laatste.
- (5) Je voegt een ? toe voor de laatste te zoeken id en je voegt de rest van het SQL statement toe.
- (6) De 1° parameter is een SQL statement met meerdere parameters.
De 2° parameter is een RowMapper
De 3° parameter is een array met waarden die bij de parameters horen.

Als ids één getal bevat, zal het SQL statement zijn:

```
select id, naam, prijs, pikant from pizzas where id in (?)
```


Als ids 3 getallen bevat, zal het SQL statement zijn:

```
select id, naam, prijs, pikant from pizzas where id in (?, ?, ?)
```



Je kan dit SQL statement niet testen in IntelliJ omdat het bestaat uit een concatenatie van strings (③ en ⑤) en code (④).

36.11 Algemeen

- Een repository method stuurt één SQL statement naar de database.
Als je in een use-case meerdere SQL statements naar de database moet sturen, stuur je elk SQL statement in één repository method.
Je roept na mekaar op in een method in de service layer.
- Een repository method bevat geen business logica.
Je plaatst business logica in domain classes en/of in de service layer.
Spring maakt het gemakkelijk om de database aan te spreken.
 Jij blijft echter verantwoordelijk om de database performant aan te spreken (zie cursus JDBC) en correct om te gaan met meerdere gebruikers die tegelijk dezelfde data lezen én wijzigen (`select ... for update`) (zie cursus JDBC).



Repository

37 REPOSITORY TEST

Een *unit* test van een repository bean is zinloos: je kan de correcte werking van zo'n bean niet testen als hij geen verbinding maakt naar de database en er SQL statements naar stuurt.

Je doet een *integration* test: je test de repository bean *met* zijn samenwerking met de database.

37.1 Transactie

Een integration test van een repository bean heeft een probleem.

De database onthoudt toevoegingen, wijzigingen en verwijderingen die je in de test doet.

Als je de test een 2° keer uitvoert, beïnvloedt de 1° uitvoering de 2° uitvoering negatief. Voorbeeld:

- Je voegt een pizza toe aan de database. Je controleert daarna of de pizza 1 keer voorkomt.
- De test slaagt wanneer je hem de 1° keer uitvoert.
- De 2° keer faalt de test, omdat de pizza al bestaat in de database (van de 1° uitvoering).

De oplossing:

1. Je start een transactie voor elke test.
2. Je voert de test uit.
3. Je rollbackt de transactie na de test. Dit doet de bewerkingen van je test ongedaan.

Je kan zo je test meerdere keren uitvoeren.

Je erft hiertoe je test class van `AbstractTransactionalJUnit4SpringContextTests`.

Spring voert dan elke test uit in een transactie. Spring doet na de test een rollback.

37.2 Test records

⊖ Als je in een repository test bestaande records gebruikt, is de test fragiel. Voorbeeld:

De table pizzas bevat een record met id 1 en naam Prosciutto.

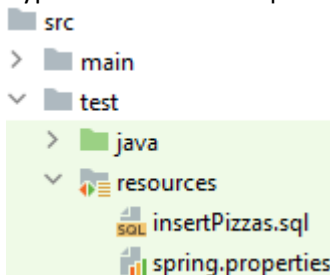
Je maakt een test voor de method `findById`: je roept `findById(1)` op. De method moet een `Pizza` object teruggeven met als naam Prosciutto (Je ziet straks de test code).

Deze test zal lukken. Zodra je echter (bvb. via de MySQL Workbench) dit record verwijdert, of in dit record de naam wijzigt, zal de test mislukken. De test is zo op lange termijn fragiel.

⊕ Je lost dit probleem op. Je maakt de test robuust en zelfstandig. De test voegt zelf een record toe en test `findById` met dit record. Gezien de test op het einde zijn transactie rollbackt, is dit record na de test verdwenen in de database.

Je doet hiertoe voorbereidend werk:

1. Klik in het project overzicht met de rechtermuisknop op `resources` binnen `test`.
2. Typ `insertPizzas.sql`. Druk Enter.



3. Kies Change dialect to rechts boven in het nieuwe venster.
4. Kies MySQL bij Global SQL Dialect. Kies OK.
5. Typ volgend SQL statement (JUnit zal dit uitvoeren bij de start van een repository test):

```
insert into pizzas(naam,prijs,pikant) values
('test', 10, false),
('test2', 20, true);
```



Je kan meerdere statements in zo'n bestand typen.
Je sluit elk statement met een ;

37.3 PizzaRepositoryTest

```

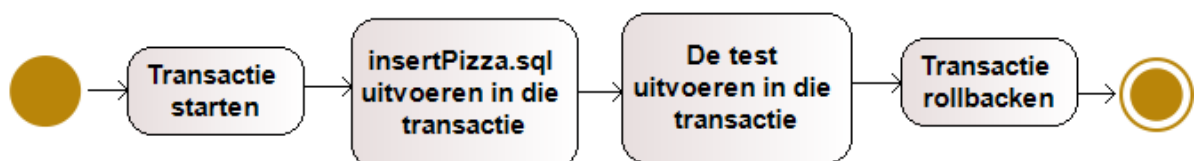
package be.vdab.luigi.repositories;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatExceptionOfType;
// enkele andere imports
@JdbcTest
@Import(PizzaRepository.class)
@Sql("/insertPizzas.sql")
class PizzaRepositoryTest
    extends AbstractTransactionalJUnit4SpringContextTests {
    private static final String PIZZAS = "pizzas";
    private final PizzaRepository repository;
    // constructor met parameter
    @Test
    void findAantal() {
        assertThat(repository.findAantal())
            .isEqualTo(countRowsInTable(PIZZAS));
    }
    @Test
    void findAllGeeftAllePizzasGesorteerdOpId() {
        assertThat(repository.findAll())
            .hasSize(countRowsInTable(PIZZAS))
            .extracting(Pizza::getId)
            .isSorted();
    }
}

```

- (1) Je verwijst naar `insertPizzas.sql`. Spring voert de SQL statements daarin uit voor elke test. Spring doet dit in dezelfde transactie waarin Spring ook de test zelf uitvoert. Spring rollbackt die transactie na de test. Spring doet dus ook het SQL statement in `insertPizzas.sql` ongedaan. Je kan bij `@Sql(...)` verwijzen naar meerdere `.sql` bestanden: `@Sql({"a.sql", "b.sql"})`. Spring voert voor elke test de SQL statements in al die bestanden uit.
- (2) Je test erft van `AbstractTransactionalJUnit4SpringContextTests`. Die zorg er voor dat elke test een transactie is die Spring na de test rollbackt. De class ook bevat methods om gemakkelijk een repository te testen.
- (3) Deze constante bevat de naam van de table (in de database) die de pizza's bevat. Je gebruikt de constante bij 6 en 7.
- (4) Je maakt een reference variabele met als type de bean die je wil testen.
- (5) Spring injecteert in de constructor de bean waarmee je de variabele bij 3 vult.
- (6) Je base class bevat een method `countRowsInTable`. Je geeft de naam van een table mee. Je krijgt het aantal records in de table terug. `findAantal` moet hetzelfde aantal geven.
- (7) `findAll` moet evenveel pizza's geven als het aantal records in de table `pizzas`.
- (8) Je controleert of de pizza's gesorteerd zijn op hun id. Je doet hier de eerste stap: je maakt een verzameling pizza id's op basis van de verzameling pizza's.
- (9) Je doet hier de tweede stap: je controleert of die verzameling gesorteerd is.

Voer de tests uit. Ze lukken.

Spring doet volgende handelingen bij het uitvoeren van elke test:



Voeg tests toe:

```
@Test void create() {
    var id = repository.create(new Pizza(0, "test2", BigDecimal.TEN, false));
    assertThat(id).isPositive();
    assertThat(countRowsInTableWhere(PIZZAS, "id = " + id)).isOne();
}
private long idVanTestPizza() {
    return jdbcTemplate.queryForObject(
        "select id from pizzas where naam = 'test'", Long.class);
}
private long idVanTest2Pizza() {
    return jdbcTemplate.queryForObject(
        "select id from pizzas where naam = 'test2'", Long.class);
}
@Test void delete() {
    var id = idVanTestPizza();
    repository.delete(id);
    assertThat(countRowsInTableWhere(PIZZAS, "id = " + id)).isZero();
}
```

- (1) Als de autonummering van de database werkt, heeft je toegevoegde pizza een positieve id.
- (2) Je base class bevat een method `countRowsInTableWhere`. Je geeft een table naam mee. Je geeft ook een voorwaarde mee waaraan records moeten voldoen. Je krijgt het aantal records dat aan de voorwaarde voldoet als return waarde. Je test hiermee of de table `pizzas` een record bevat met de id van de toegevoegde pizza.
- (3) Je gebruikt deze method verder in andere test methods.
- (4) Je base class bevat een variabele `jdbcTemplate`, van het type `JdbcTemplate`. Je voert daarmee SQL statements uit in je test. Je zoekt hier de id van de eerste pizza die je maakte in `insertPizzas.sql`.
- (5) Na het verwijderen van een pizza mag de database geen record meer bevatten met het id van de verwijderde pizza.

Voeg tests toe:

```
@Test void findById() {
    assertThat(repository.findById(idVanTestPizza()))
        .hasValueSatisfying(pizza->assertThat(pizza.getNaam()).isEqualTo("test"));
}
@Test void findByIdOnbestaandeIdVindtGeenPizza() {
    assertThat(repository.findById(-1)).isEmpty();
}
@Test void update() {
    var id = idVanTestPizza();
    var pizza = new Pizza(id, "test", BigDecimal.TEN, false);
    repository.update(pizza);
    assertThat(countRowsInTableWhere(PIZZAS, "prijs=10 and id=" + id)).isOne();
}
@Test void updateOnbestaandePizzaGeeftEenFout() {
    assertThatExceptionOfType(PizzaNietGevondenException.class).isThrownBy(
        () -> repository.update(new Pizza(-1, "test", BigDecimal.TEN, false)));
}
```

- (1) Je controleert met `hasValueSatisfying` de waarde in een `Optional`. De parameter `pizza` van de lambda bevat die waarde. Je test met `assertThat` of die pizza aan een voorwaarde voldoet.
- (2) Je wijzigde de prijs van de eerst toegevoegde pizza naar 10. Het aantal records met de id van de toegevoegde pizza en de prijs 10 moet 1 zijn.

Voeg tests toe:

```

@Test
void findByPrijsBetween() {
    var van = BigDecimal.ONE;
    var tot = BigDecimal.TEN;
    assertThat(repository.findByPrijsBetween(van, tot))
        .hasSize(super.countRowsInTableWhere(PIZZAS, "prijs between 1 and 10"))
        .extracting(Pizza::getPrijs)
        .allSatisfy(prijs -> assertThat(prijs).isBetween(van, tot)) ❶
        .isSorted();
}

@Test
void findUniekePrijzenGeeftPrijzenOplopend() {
    assertThat(repository.findUniekePrijzen())
        .hasSize(jdbcTemplate.queryForObject(
            "select count(distinct prijs) from pizzas", Integer.class))
        .doesNotHaveDuplicates() ❷
        .isSorted();
}

@Test
void findByPrijs() {
    assertThat(repository.findByPrijs(BigDecimal.TEN))
        .hasSize(super.countRowsInTableWhere(PIZZAS, "prijs = 10"))
        .allSatisfy(pizza ->
            assertThat(pizza.getPrijs()).isEqualByComparingTo(BigDecimal.TEN))
        .extracting(Pizza::getNaam)
        .isSortedAccordingTo(String::compareToIgnoreCase); ❸
}

@Test
void findByIds() {
    long id1 = idVanTestPizza();
    long id2 = idVanTest2Pizza();
    assertThat(repository.findByIds(Set.of(id1, id2)))
        .extracting(Pizza::getId)
        .containsOnly(id1, id2)
        .isSorted();
}

@Test
void findByIdsGeeftLegeVerzamelingPizzasBijLegeVerzamelingIds() {
    assertThat(repository.findByIds(Set.of())).isEmpty();
}

@Test
void findByIdsGeeftLegeVerzamelingPizzasBijOnbestaandeIds() {
    assertThat(repository.findByIds(Set.of(-1L))).isEmpty();
}

```

- (1) allSatisfy test of alle objecten in een verzameling aan een voorwaarde voldoen.
Je geeft een lambda mee. allSatisfy roept die op per object in de verzameling.
De parameter van de lambda is dit object.
De code van de lambda is de test (met assertThat) die je op het object uitvoert.
- (2) De unieke prijzen mogen geen dubbele waarden bevatten.
- (3) SQL sorteert hoofdletter onafhankelijk.
Je test, of de pizza's goed gesorteerd zijn, moet dus ook hoofdletter onafhankelijk zijn.

Voer de tests uit. Ze lukken.

38 SQL STATEMENTS LOGGEN

Het is interessant de SQL statements te zien die Spring naar de database stuurt.

- ⊕ Het helpt je fouten te vinden als een statement een exception veroorzaakt.
- ⊕ Je kan nazien of je de database performant benadert.
- ⊕ Je kan de waarden zien die je invult in de ? tekens in de statements.

Voeg daartoe regels toe aan application.properties:

```
logging.level.org.springframework.jdbc.core.JdbcTemplate=DEBUG ❶
logging.level.org.springframework.jdbc.core.simple.SimpleJdbcInsert=DEBUG ❷
logging.level.org.springframework.jdbc.core.StatementCreatorUtils=TRACE ❸
```

- (1) JdbcTemplate toont hiermee de SQL statements.
- (2) SimpleJdbcInsert toont hiermee de SQL statements.
- (3) Spring toont zo de waarden die hij invult in de ? tekens in SQL statements met parameters.

Voer de test findByPrijsBetween uit.

Je ziet in het venster onder in IntelliJ onder andere volgende informatie:

```
Executing prepared SQL statement [select id,naam,prijs,pikant ❶
from pizzas
where prijs between ? and ?
order by prijs]
Setting SQL statement parameter value: column index 1, parameter value [1],
value class [java.math.BigDecimal], SQL type unknown ❷
Setting SQL statement parameter value: column index 2, parameter value [10],
value class [java.math.BigDecimal], SQL type unknown ❸
```

- (1) Spring voert volgend SQL statement uit:
select id,naam,prijs,pikant from pizzas where prijs between ? and ? order by prijs
- (2) Spring vult het eerste ? in dat statement in met de waarde 1.
- (3) Spring vult het tweede ? in dat statement in met de waarde 10.



Voer alle testen uit. Commit de sources . Publiceer op je remote repository.



Repository test

39 SERVICE EN TRANSACTIE

39.1 PizzaService

Maak een class PizzaService.

Gezien je van deze service maar één implementatie hebt, maak je geen bijbehorende interface.

```
package be.vdab.luigi.services;
```

```
// enkele imports
```

```
@Service
```

❶

```
public class PizzaService {
    private final PizzaRepository pizzaRepository;
    public PizzaService(PizzaRepository pizzaRepository) {
        this.pizzaRepository = pizzaRepository;
    }
}
```

❷

(1) Je typt @Service voor de class.

Spring maakt bij de start van de website een singleton bean van de class.

IntelliJ toont dit met het icoon  in de marge.

(2) Je injecteert de PizzaRepository bean.

Voeg methods toe aan deze class:

```
public long create(Pizza pizza) {
    return pizzaRepository.create(pizza);
}
public void update(Pizza pizza) {
    pizzaRepository.update(pizza);
}
public void delete(long id) {
    pizzaRepository.delete(id);
}
public List<Pizza> findAll() {
    return pizzaRepository.findAll();
}
public Optional<Pizza> findById(long id) {
    return pizzaRepository.findById(id);
}
public List<Pizza> findByPrijsBetween(BigDecimal van, BigDecimal tot){
    return pizzaRepository.findByPrijsBetween(van, tot);
}
public long findAantal() {
    return pizzaRepository.findAantal();
}
public List<BigDecimal> findUniekePrijzen() {
    return pizzaRepository.findUniekePrijzen();
}
public List<Pizza> findByPrijs(BigDecimal prijs) {
    return pizzaRepository.findByPrijs(prijs);
}
public List<Pizza> findByIds(Set<Long> ids) {
    return pizzaRepository.findByIds(ids);
}
```

39.2 Test

De methods in `PizzaService` bevatten elk maar één opdracht.
Een unit test van zo'n method heeft daarom weinig meerwaarde.

We tonen hier toch zo'n test, zodat je de opbouw ziet. Je kan die kennis later gebruiken als je een complexe service method wil testen. We injecteren in de test een mock als repository in de service.

```
package be.vdab.luigi.services;
// enkele imports
@ExtendWith(MockitoExtension.class) ❶
class PizzaServiceTest {
    @Mock
    private PizzaRepository repository; ❷
    private TPizzaService service;
    @BeforeEach
    void beforeEach() {
        service = new PizzaService(repository); ❸
    }
    @Test
    void create() {
        var pizza = new Pizza(0, "test", BigDecimal.TEN, false); ❹
        when(repository.create(pizza)).thenReturn(1L); ❺
        assertThat(service.create(pizza)).isEqualTo(1L); ❻
        verify(repository).create(pizza); ❼
    }
}
```

- (1) Je laat JUnit samenwerken met Mockito.
- (2) Je laat (met `@Mock` op de vorige regel) Mockito een mock object maken waarvan de class erft van `PizzaRepository`.
- (3) Je maakt een object van de te testen class (`PizzaService`).
Je injecteert het mock object van bij ❷ in de constructor.
- (4) Je maakt een `Pizza` object.
- (5) Je traint de mock: als je de method `create` oproept, met het `pizza` van bij ❹, moet de mock het getal 1 teruggeven (als "fake" nummer die de pizza kreeg na het opslaan).
- (6) Je roept de method op die je wil testen: `create`.
Als die correct werkt, geeft die ook het nummer 1 terug (die de pizza kreeg na het opslaan).
- (7) Je test of de service method `create` de repository method `create` heeft opgeroepen.

39.3 Transactie

Je beheert transacties in de services laag. Je leert transactie eigenschappen kennen.
De belangrijkste zijn isolation level, read-only, timeout, en propagation.

39.3.1 Isolation level

Dit bepaalt hoe gelijktijdige transacties (van bvb. andere gebruikers) jouw transactie beïnvloeden.
Volgende problemen kunnen optreden bij gelijktijdige transacties T1 en T2:

- ❖ **Dirty read**
T1 leest data die T2 schreef, maar nog niet committe.
Als T2 rollbackt, heeft T1 verkeerde data gelezen.
- ❖ **Nonrepeatable read**
T1 leest dezelfde data meerdere keren en krijgt per lees opdracht andere data.
Oorzaak: T2 die, tussen de lees opdrachten van T1, data wijzigt die T1 leest.
T1 krijgt geen stabiel beeld van de gelezen data.
- ❖ **Phantom read**
T1 leest dezelfde data meerdere keren en krijgt per lees opdracht meer records.
Oorzaak: T2 die, tussen de lees opdrachten van T1, records toevoegt.
T1 krijgt geen stabiel beeld van de gelezen data.

Je verhindert één of meerdere problemen door het transaction isolation level in te stellen

↓ Isolation level ↓ van snel naar traag	Dirty read mogelijk	Nonrepeatable read mogelijk	Phantom read mogelijk
Read uncommitted	Ja	Ja	Ja
Read committed	Nee	Ja	Ja
Repeatable read	Nee	Nee	Ja
Serializable	Nee	Nee	Nee

Serializable lost alle problemen op. Het is wel het *traagste* isolation level.

Je analyseert dus per use case welke problemen (dirty read, ...) die use case benadelen. Je kiest daarna het isolation level dat voldoende is om die problemen op te lossen. Je gebruikt zelden read uncommitted: het lost geen enkel probleem op. Je kan bijna altijd read committed gebruiken.

Als je het isolation level niet instelt, krijgt de transactie het default isolation level van de database. Dit verschilt per merk database. Bij MySQL is het bijvoorbeeld Repeatable read.

Voeg een regel toe aan application.properties:

`spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED` ❶

(1) Je definieert het standaard isolation level dat Spring gebruikt bij je transacties.

Als je voor een use-case een ander isolation level nodig hebt, definieer je dat in die use-case.

39.3.2 Read-only

Als een transactie enkel records leest (niet toevoegt, wijzigt of verwijdert) maak je de transactie read-only. Spring voert dan op de JDBC connectie de method `setReadOnly(true)` uit.

Sommige merken database optimaliseren dan de uitvoeringssnelheid van de transactie.

Als je in een read-only transactie toch records toevoegt, wijzigt of verwijdert, krijg je een exception.

Als je read-only niet instelt, is een transactie niet read-only.

39.3.3 Timeout

De database vergrendelt records tijdens het uitvoeren van bepaalde transacties. Als andere transacties dezelfde records aanspreken, wachten ze tot de 1^e transactie de records ontgrendelen. Je stelt met de timeout eigenschap in hoelang je transactie maximaal mag lopen.

Als je transactie langer loopt (als je records aanspreekt die andere transacties lang vergrendelen), rollbackt Spring je transactie.

Als je de timeout niet instelt, krijg je de timeout van de achterliggende database.

Dit verschilt per merk database.

39.4 @Transactional

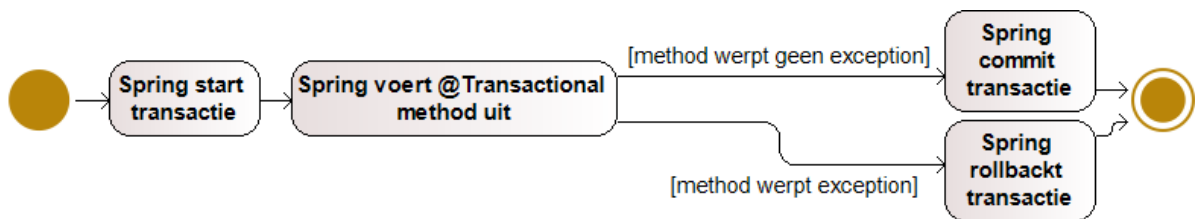
Je moet zelf geen code typen waarmee je een transactie start, commit of rollbackt.

Je typt voor een method `@Transactional`. Spring verzamelt dan alle database bewerkingen, die je in die method uitvoert, in één transactie.

- Als je zo'n method oproept, start Spring eerst een transactie.
- Spring commit die transactie op het einde van de method:

```
@Transactional
void eenMethod() { // Spring start een transactie
...
} // Spring doet een commit op de transactie
```
- Spring doet een rollback als de method een exception werpt (die de method zelf niet opvangt):

```
@Transactional
void eenMethod() { // Spring start een transactie
...
...throw new EenException(); // Spring doet een rollback
...
}
```

Je kan `@Transactional` typen bij een class en/of bij methods van een class.

- Bij een class: elke method in die class is één transactie.
- Bij een method: die method is één transactie.
- Bij één class én bij een method van die class,
De transactie eigenschappen, beschreven bij de method,
overschrijven de transactie eigenschappen beschreven bij de class.

Je kan bij `@Transactional` de transactie eigenschappen instellen:

- `isolation` `READ_UNCOMMITTED`, `READ_COMMITTED`, `REPEATABLE_READ`, `SERIALIZABLE`
- `readOnly` `true` of `false`
- `timeout` een aantal seconden

Wijzig `PizzaService`:

- Typ `@Transactional(readOnly = true)` voor de class.
- Typ `@Transactional` voor de methods `create`, `update` en `delete`,
om af te wijken van `readOnly = true` dat je voor de class type.

39.5 Propagation

Propagation is een transactie eigenschap. Je kan die ook meegeven aan `@Transactional`.

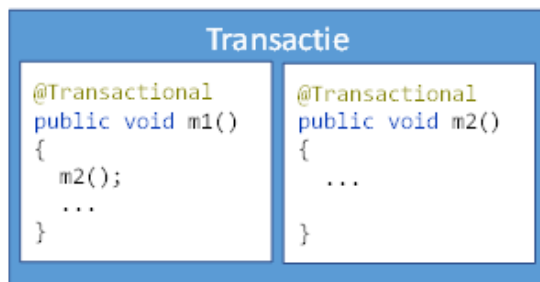
Je bepaalt met de propagation wat Spring doet met een `@Transactional` method `m2`, als je die oproept vanuit een `@Transactional` method `m1`. De meest gebruikte propagations:

REQUIRED (de standaard propagation)

Spring voert de database bewerkingen in `m2` uit in de transactie die al loopt in `m1`.

De bewerkingen van `m1` en `m2` behoren dus tot dezelfde transactie.

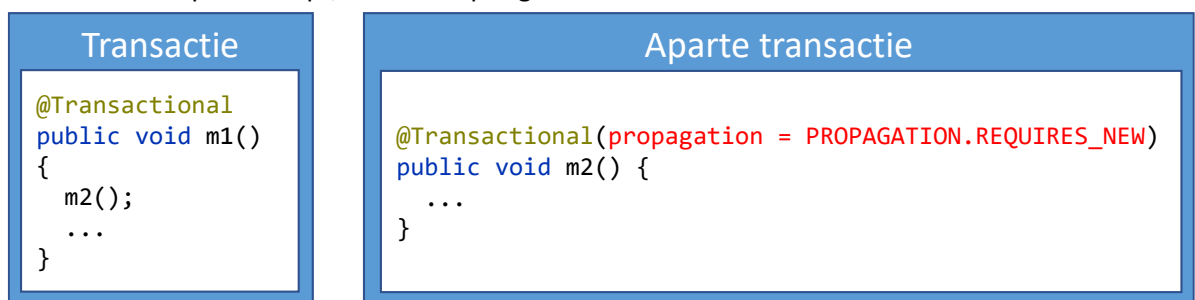
Als `m1` of `m2` een exception werpen, rollbackt Spring de transactie.



REQUIRES_NEW

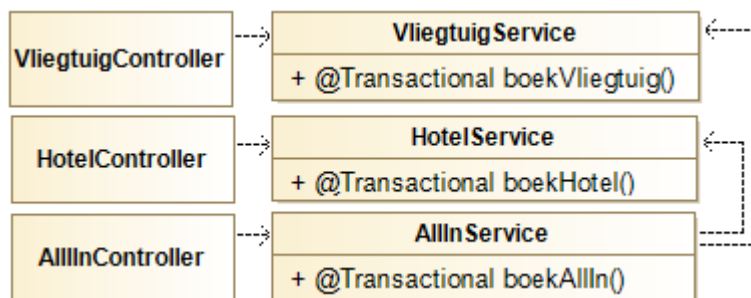
Spring voert `m2` uit in een nieuwe transactie. Spring pauzeert daarbij de transactie van `m1`.

Als `m2` een exception werpt, rollbackt Spring enkel de transactie van `m2`.



Onderstaand voorbeeld (een reisbureau) toont dat REQUIRED handig is om een method

- apart te kunnen oproepen, (boekVliegtuig, boekHotel)
- maar ook te kunnen combineren met *andere* methods tot één transactie (boekAllIn)



- **Gebruiker boekt vliegtuigreis**

VliegtuigController roept boekVliegtuig op in VliegtuigService.

```

@Transactional
public void boekVliegtuig(VliegtuigBoeking boeking) {
    // alle repository oproepen behoren tot een transactie
    if ( ! vliegtuigRepository.heeftPlaatsenVrijVoor(boeking)) {
        throw new OnvoldoendeVliegtuigPlaatsenException();
    }
    vliegtuigRepository.create(boeking);
}
  
```

- **Gebruiker boekt hotelverblijf**

HotelController roept boekHotel op in HotelService.

```

@Transactional
public void boekHotel(HotelBoeking boeking) {
    // alle repository oproepen behoren tot een transactie
    if ( ! hotelRepository.heeftBeddenVrijVoor(boeking)) {
        throw new OnvoldoendeHotelBeddenException();
    }
    hotelRepository.create(boeking);
}
  
```

- **Gebruiker boekt all-in**

AllInController roept boekAllIn op in AllInService.

```

@Transactional
public void boekAllIn(AllInBoeking boeking) {
    // methods boekVliegtuig en boekHotel combineren tot één transactie:
    vliegtuigService.boekVliegtuig(boeking.getVliegtuigBoeking());
    hotelService.boekHotel(boeking.getHotelBoeking());
}
  
```

- (1) Spring start een transactie.
- (2) Spring start geen nieuwe transactie, maar voert de methods in boekVliegtuig uit in de transactie die al startte bij ❶.
- (3) Spring start ook hier geen nieuwe transactie, maar voert de methods in boekHotel uit in de transactie die al startte bij ❶.

Als boekHotel een exception werpt, rollbackt Spring de transactie.

Spring doet dan ook de handelingen die boekVliegtuig uitvoerde ongedaan.

Dit is OK: als je het hotel van een all-in niet kan boeken, mag ook het vliegtuig niet geboekt zijn.

Je zag een voorbeeld van een service die andere service(s) oproept. Je hebt dit zelden nodig, enkel als de service de intelligentie(if statements, berekeningen...) van die service(s) nodig heeft.

39.6 Controller

Je injecteert de PizzaService. Je kan daarna methods oproepen van de PizzaService. Bij een unit test van de controller kan je een mock injecteren.

```
package be.vdab.luigi.controllers;
// enkele imports
@Controller @RequestMapping("pizzas")
class PizzaController {
    private final Logger logger = LoggerFactory.getLogger(this.getClass());
    private final EuroService euroService;
    private final PizzaService pizzaService;
    // constructor met parameters die beide services injecteert.
    // een controller kan dus MEERDERE dependencies hebben.
    @GetMapping public ModelAndView findAll() {
        return new ModelAndView("pizzas", "allePizzas", pizzaService.findAll());
    }
    @GetMapping("{id}") public ModelAndView findById(@PathVariable long id) {
        var modelAndView = new ModelAndView("pizza");
        pizzaService.findById(id).ifPresent(pizza -> { // service oproepen
            modelAndView.addObject("pizza", pizza);
            try {
                modelAndView.addObject(
                    "inDollar", euroService.naarDollar(pizza.getPrijs()));
            } catch (KoersClientException ex) {
                logger.error("Kan dollar koers niet lezen", ex);
            }
        });
        return modelAndView;
    }
    @GetMapping("prijzen") public ModelAndView findPrijzen() {
        return new ModelAndView("prijzen",
            "prijzen", pizzaService.findUniekePrijzen()); // service oproepen
    }
    @GetMapping("prijzen/{prijs}")
    public ModelAndView findByPrijs(@PathVariable BigDecimal prijs) {
        return new ModelAndView("prijzen",
            "pizzas", pizzaService.findByPrijs(prijs)) // service oproepen
            .addObject("prijzen", pizzaService.findUniekePrijzen()); // ook hier
    }
}
```

Je kan de website proberen.

39.7 Samenwerking

PizzaController gebruikt volgende dependencies:

```

PizzaController → EuroService → FixerKoersClient
                → PizzaService → PizzaRepository → DataSource

```

Volgende bean methods werken samen bij een request naar /pizzas:

```

PizzaController → PizzaService → PizzaRepository
  findAll()      findAll()      findAll()

```

Volgende bean methods werken samen bij een request naar /pizzas/1:

```

PizzaController → PizzaService → PizzaRepository
  findById(1)    findById(1)      findById(1)
                → EuroService → FixerKoersClient
                  naarDollar(...) getDollarKoers()

```

39.8 Database bewerkingen die één geheel vormen in een transactie

Meerdere database bewerkingen vormen soms één geheel. Het is belangrijk dat deze bewerkingen te doen binnen één transactie, zodat de database deze bewerkingen:

- ofwel *allemaal* uitvoert (commit)
- ofwel *allemaal* ongedaan maakt als halverwege een fout optreedt (rollback)

Je doet dit door repositories op te roepen in één method van een `@Transactional` service.

39.8.1 Voorbeeld 1

Geld overschrijven van één bankrekening naar een andere bankrekening.

 Goede oplossing: beide bewerkingen uitvoeren in één en dezelfde transactie in de service:

```
@Service
@Transactional
public class RekeningService {
    private final RekeningRepository rekeningRepository;


    ...
    void schrijfOver(...) {
        var vanRekening = rekeningRepository.findAndLockByNummer(vanNummer);
        ...
        var naarRekening = rekeningRepository.findAndLockByNummer(naarNummer);
        ...
        rekeningRepository.update(vanRekening);
        rekeningRepository.update(naarRekening);
    }
}
```

- (1) De bewerkingen in de method `schrijfOver` vormen met `@Transactional` één transactie.
- (2) Je leest de van-rekening en lockt het record tot het einde van je transactie.
Je verhindert zo dat andere gebruikers in die tijd het record wijzigen of verwijderen.
- (3) Je leest de naar-rekening en lockt het record tot het einde van je transactie.
Je verhindert zo dat andere gebruikers in die tijd het record wijzigen of verwijderen.
- (4) Deze bewerking (het saldo van de van-rekening verlagen) behoort tot die transactie.
- (5) Deze bewerking (het saldo van de naar-rekening verhogen) behoort tot *dezelfde* transactie.

Als er een fout optreedt tussen ④ en ⑤ rollbackt Spring de transactie.

De database maakt de bewerking bij ④ ongedaan

Het saldo van de van-rekening is dus gelukkig niet verminderd.

 Verkeerde oplossing: de 2 bewerkingen oproepen in een Controller:
de database bewerkingen behoren dan niet tot *dezelfde* transactie !

```
@Controller
class RekeningController {
    @PostMapping
    String schrijfOver(...) {
        ...
        rekeningService.update(vanRekening);
        rekeningService.update(naarRekening);
        ...
    }
}
```

- (1) De bewerkingen die je doet binnen deze update method behoren tot een transactie.
- (2) De bewerkingen die je doet binnen deze update method behoren tot een *andere* transactie.

Als er een fout optreedt tussen ① en ② doet Spring *geen* rollback op de transactie van ①.

Het saldo van de van rekening is verlaagd terwijl het saldo van de naar rekening niet verhoogd is.

Het over te schrijven bedrag is “in het niets” verdwenen!


39.8.2 Voorbeeld 2

Een artikel toevoegen aan een levering en de voorraad van dat artikel verminderen.

Je moet daarbij twee handelingen doen in de database:

- een record toevoegen aan de table leveringen
- én een record wijzigen in de tabel artikels.

De database moet ofwel beide handelingen uitvoeren, ofwel een rollback doen.

 Goede oplossing: beide bewerkingen uitvoeren in één en dezelfde transactie in de service:

```
@Service
@Transactional
public class LeveringService {
    private final LeveringRepository leveringRepository;           ❶
    private final ArtikelRepository artikelRepository;             ❷
    ...
    void voegArtikelToeAanLevering(...) {                          ❸
        ...
        leveringRepository.create(levering);                       ❹
        artikelRepository.update(artikel);                          ❺
        ...
    }
}
```

(1) Dit is een voorbeeld van een service met twee verschillende dependencies: één bij ❶.

(2) De andere bij ❷. Complexere services hebben *meerdere* dependencies.

(3) Met `@Transactional` vormen de bewerkingen in deze method één transactie.

(4) Deze bewerking (record toevoegen aan leveringen) behoort tot die transactie.

(5) Deze bewerking (voorraad verminderen van artikel) behoort tot *dezelfde* transactie.

 Verkeerde oplossing: beide bewerkingen zijn aparte transacties in de controller:

```
@Controller
class LeveringController {
    @PostMapping
    String voegArtikelToeAanLevering(...) {
        ...
        leveringService.update(levering);                           ❶
        artikelService.update(artikel);                             ❷
        ...
    }
}
```

(1) Als je programma juist hierna uitvalt, wordt ❷ niet uitgevoerd en staat de voorraad verkeerd !



Voer alle testen uit. Commit de sources. Publiceer op je remote repository.



Services

40 DTO

Je kan het resultaat van een SQL select statement soms niet voorstellen met je domain classes.

Voorbeeld use case: je wil het aantal pizza's per prijs tonen.

Het SQL statement:

```
select prijs, count(*) as aantal
from pizzas
group by prijs
order by prijs
```

Het resultaat van deze query bevat twee kolommen: prijs en aantal.

Je stelt één rij van dit resultaat in Java gemakkelijk voor met een record.

Je hebt geen domain class die dit voorstelt.

Je maakt dan een DTO (Data Transfer Object) die dit voorstelt, zoals in de cursus JDBC:

```
package be.vdab.luigi.dto;
import java.math.BigDecimal;
public record AantalPizzasPerPrijs(BigDecimal prijs, int aantal) {
}
```

40.1 Repository

Voeg een method toe aan PizzaRepository:

```
public List<AantalPizzasPerPrijs> findAantalPizzasPerPrijs() {
    var sql = """
        select prijs, count(*) as aantal
        from pizzas
        group by prijs
        order by prijs
        """;
    RowMapper<AantalPizzasPerPrijs> mapper = (result, rowNum) ->
        new AantalPizzasPerPrijs(
            result.getBigDecimal("prijs"), result.getInt("aantal"));
    return template.query(sql, mapper);
}
```

Test dit met een extra method in PizzaRepositoryTest:

```
@Test
void aantalPizzasPerPrijs() {
    var aantalPizzasPerPrijs = repository.findAantalPizzasPerPrijs();
    assertThat(aantalPizzasPerPrijs).hasSize(super.jdbcTemplate.queryForObject(
        "select count(distinct prijs) from pizzas", Integer.class)); ❶
    var rij1 = aantalPizzasPerPrijs.get(0); ❷
    assertThat(rij1.aantal()) ❸
        .isEqualTo(super.countRowsInTableWhere(PIZZAS, "prijs =" + rij1.prijs()));
}
```

(1) Het aantal rijen in het resultaat moet gelijk zijn aan het aantal unieke prijzen.

(2) Je haalt de eerste rij op.

(3) Het aantal in die rij moet gelijk zijn aan het aantal pizza's met de prijs in die rij.

Voer de test uit. Hij lukt.

40.2 Service

Voeg een method toe aan PizzaService:

```
@Transactional(readOnly = true)
public List<AantalPizzasPerPrijs> findAantalPizzasPerPrijs() {
    return pizzaRepository.findAantalPizzasPerPrijs();
}
```

40.3 Controller

Voeg een method toe aan PizzaController:

```
@GetMapping("aantalpizzasperprijs")
public ModelAndView findAantalPizzasPerPrijs() {
    return new ModelAndView("aantalpizzasperprijs",
        "aantalPizzasPerPrijs", pizzaService.findAantalPizzasPerPrijs());
}
```

40.4 Thymeleaf

Maak de pagina `aantalpizzasperprijs.html`:

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
<head th:replace="fragments::head(title='Aantal pizza's per prijs')"></head>
<body>
    <nav th:replace="fragments::menu"></nav>
    <h1>Aantal pizza's per prijs</h1>
    <table id="aantalpizzasperprijs">
        <thead>
            <tr>
                <th>Prijs</th>
                <th>Aantal pizza's</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="prijsEnAantal:${aantalPizzasPerPrijs}"
                th:object="${prijsEnAantal}">
                <td th:text="*{prijs}"></td>
                <td th:text="*{aantal}"></td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

Voeg een menu-punt naar de pagina toe in `fragments.html` in het fragment menu:

```
<li><a th:href="@{/pizzas/aantalpizzasperprijs}">Aantal pizza's per
prijs</a></li>
```

Je kan dit proberen.



Voer alle testen uit. Commit de sources. Publiceer op je remote repository.



Dagverkopen

41 FORM

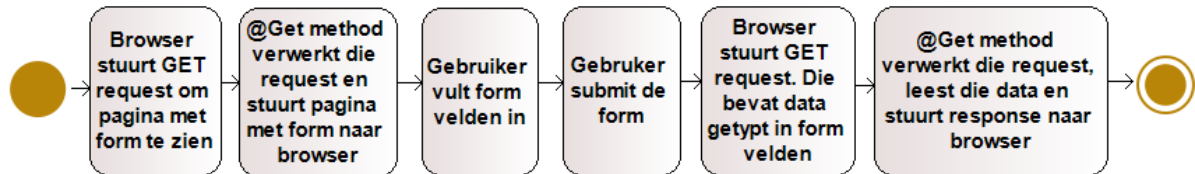
Je verwerkt hier data die de gebruiker typt in een HTML form.

Wanneer de gebruiker de form submit, stuurt de browser

- een GET request, bij `<form method="get" ...>` of
- een POST request, bij `<form method="post" ...>`.

Je gebruikt hier een GET request, later in de cursus een POST request.

Volgorde van handelingen:



41.1 Form class of record

Je stelt een HTML form voor met een class of record

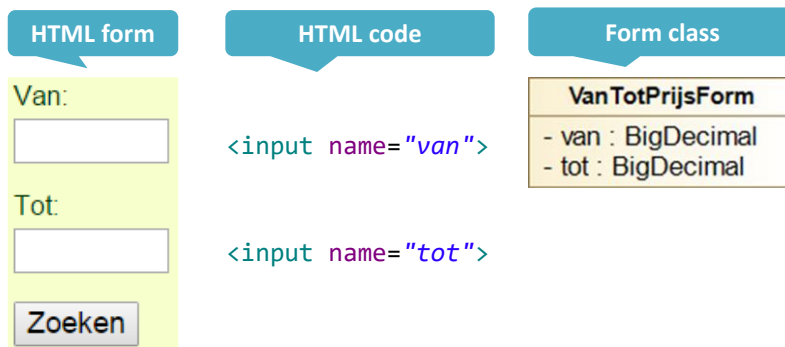
Dit bevat een private variabele per invoervak van de form.

Je gebruikt de class/record

- Eerst om de form te tonen.
- Later om te lezen wat de gebruiker in de form typte, na de submit van de form.

Voorbeeld: de gebruiker typt in volgende form een “van” en een “tot” prijs.

Als hij de form submit, toon jij de pizza's met een prijs tussen die van en tot:



Maak een package `be.vdab.luigi.forms`. Maak daarin `VanTotPrijsForm`:

```

package be.vdab.luigi.forms;
import java.math.BigDecimal;
public record VanTotPrijsForm(BigDecimal van, BigDecimal tot) {
}
  
```

❶

- (1) Een form object bevat enkel data, geen logica. Je maakt dit snel met een record.

41.2 Form tonen

41.2.1 PizzaController

Je toont de form aan de gebruiker bij een GET request naar de URL `/pizzas/vantotprijs/form`.
Je verwerkt die request in een nieuwe method in `PizzaController`:

```
@GetMapping("vantotprijs/form")
public ModelAndView vanTotPrijsForm() {
    return new ModelAndView("vantotprijs")
        .addObject(new VanTotPrijsForm(BigDecimal.ONE, BigDecimal.TEN)); ❶
}
```

- (1) Je maakt een form object. Je vult daarin van met 1 en tot met 10.
Je geeft dit object door aan de Thymeleaf pagina onder de naam `vanTotPrijsForm`.
Spring maakt die naam op basis van het type van dit object: het record `VanTotPrijsForm`.

41.2.2 vantotprijs.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head th:replace="fragments::head(title='Van tot prijs')"></head>
  <body>
    <nav th:replace="fragments::menu"></nav>
    <h1>Van tot prijs</h1>
    <form th:object="${vanTotPrijsForm}" ❶
      method="get" th:action="@{/pizzas/vantotprijs}"> ❷
      <label>Van:<input th:field="*{van}" autofocus></label> ❸
      <label>Tot:<input th:field="*{tot}"></label>
      <button>Zoeken</button>
    </form>
  </body>
</html>
```

- (1) Je geeft aan dat je attributen zal lezen van het form object `vanTotPrijsForm`.
- (2) De browser stuurt, bij de form submit, een GET request naar `/pizzas/vantotprijs`.
Die request bevat de inhoud van de invoervakken van de form. Je ziet dit straks.
- (3) Je verwijst met `th:field` naar het “van” attribuut van het form object.
Thymeleaf stuurt HTML met volgend `<input...>` element naar de browser:
`<input id="van" name="van" value="1" autofocus>`
`value` is ingevuld met 1: de waarde van het “van” attribuut van het form object.
1 wordt zo de inhoud van het invoervak.

Voeg een menu-punt naar de pagina toe in `fragments.html` in het fragment `menu`:

```
<li><a th:href="@{/pizzas/vantotprijs/form}">Van tot prijs</a></li>
```

Je kan dit proberen.

Je krijgt een fout als je op de knop Zoeken klikt: je hebt nog geen method in `PizzaController` die de request verwerkt die bij de submit van de form hoort. Je doet dit straks.

41.3 Leeg invoervak

Als je lege invoervakken wil tonen, vul je de attributen van het form object, die horen bij de invoervakken, met `null`.

Wijzig in `PizzaController` de code in de method `vanTotPrijs`:

```
return new ModelAndView("vantotprijs").addObject(new VanTotPrijsForm(null, null));
```

Je kan dit proberen.

41.4 Query string

Open de pagina in je browser. Typ 4 in het 1° vak. Typ 5 in het 2° vak. Kies Zoeken.

De browser stuurt een GET request bij de submit van de form.

Je krijgt een fout. Je lost die straks op.

Kijk in de adres balk van je browser. Je ziet de URL van de GET request:

<http://localhost:8080/pizzas/vantotprijs?van=4&tot=5>

- (1) Dit is wat je typte in het action attribuut van de form in `vantotprijs.html`:

```
<form ... th:action="@{/pizzas/vantotprijs}">
```

 De rest van de URL is de query string. Die bevat parameters.
 Elke parameter bevat een waarde die de gebruiker typte in een invoervak.
- (2) Voor de eerste parameter staat ?.
- (3) De 1° parameter: naam van het invoervak (van), =, inhoud van het invoervak (4).
- (4) Tussen elke parameter staat &.
- (5) De 2° parameter: naam van het invoervak (tot), =, inhoud van het invoervak (5).

41.5 Gesubmitte form verwerken

Maak in `PizzaController` een `@GetMapping` method. Die verwerkt de request van de submit:

```
@GetMapping("vantotprijs")
public ModelAndView findByPrijsBetween(VanTotPrijsForm form) {           ❶
    return new ModelAndView("vantotprijs", "pizzas",
        pizzaService.findByPrijsBetween(form.van(), form.tot()));      ❷
}
```

- (1) De method heeft een `VanTotPrijsForm` parameter.
 Spring ziet dit en maakt een `VanTotPrijsForm` object met de constructor.
 Spring vult de `van` parameter van de constructor met de `van` parameter in de query string.
 Spring vult de `tot` parameter van de constructor met de `tot` parameter in de query string.
- (2) Je zoekt de pizza's met de `PizzaService` method `findByPrijsBetween`.
 Je geeft de pizza's onder de naam `pizzas` door aan de Thymeleaf pagina.
 Je leest zeker niet *alle* pizza's uit de database. Dit zou de website vertragen.

Toon de pizza's in `vantotprijs.html` onder `</form>`:

```
<th:block th:if="${pizzas}">
    <table th:replace="fragments::pizzaTabel(pizzas=${pizzas})"></table>
</th:block>
```

Je kan dit proberen.

41.6 Validatie

De gebruiker typt soms verkeerde waarden in de invoervakken. Je valideert daarom die waarden.

Je doet dit in de method die de request bij de submit van de form verwerkt: `vantotPrijs`.

Je doet hier eenvoudige validaties. Je doet in het volgende hoofdstuk extra validaties, zoals:

- De invoervakken zijn verplicht in te vullen.
- De invoervakken mogen geen negatieve waarden bevatten.

De gebruiker moet in de invoervakken `van` en `tot` een getal typen.

Als hij `bla` typt veroorzaakt dit een exception in je website.

Probeer dit. Typ `bla` in het 1° vak. Typ `ai` in het 2° vak. Kies Zoeken.

Je krijgt een fout. Je lost die straks op. Kijk in de adres balk van je browser. Je ziet:

<http://localhost:8080/pizzas/vantotprijs?van=bla&tot=oei>

De browser stuurde bij de submit van de form een GET request. Je verwerkt die in de controller method `vanTotPrijs`. Spring probeert daarbij de `VanTotPrijsForm` constructor op te roepen. Spring werpt een exception: hij kan de inhoud van de parameter "van" in de query string (`bla`) niet omzetten naar constructor parameter `van`: die is van het type `BigDecimal`.

Je zorgt er voor dat de gebruiker een vriendelijke boodschap ziet in de plaats van de exception. Wijzig de method `findByPrijsBetween`:

```
@GetMapping("vantotprijs")
public ModelAndView findByPrijsBetween(VanTotPrijsForm form, Errors errors) { ❶
    var modelAndView = new ModelAndView("vantotprijs");
    if (errors.hasErrors()) { ❷
        return modelAndView; ❸
    }
    return modelAndView.addObject("pizzas",
        pizzaService.findByPrijsBetween(form.van(), form.tot()));
}
```

- (1) Je detecteert validatiefouten met een `Errors` parameter. Die moet komen na de parameter met het form object.
- (2) `hasErrors` geeft `true` als de gebruiker verkeerde data typte.
- (3) Je zoekt dan geen pizza's. Je toont wel de pagina met de form (met de verkeerde data) opnieuw, zodat de gebruiker die data kan corrigeren. Spring geeft het form object in de parameter bij ❶ zelf door aan de Thymeleaf pagina, onder de naam `vanTotPrijsForm`. Jij moet het form object dus niet doorgeven.

41.6.1 `vantotprijs.html`

Je toont foutboodschappen als de gebruiker verkeerde waarden typte.

Typ na Van:

```
<span th:errors="*{van}"></span>
```

❶

- (1) Thymeleaf toont deze `` als er een fout was in het invoervak van. Thymeleaf plaats een foutboodschap tussen `` en ``. Deze `` krijgt in `luigi.css` een mooie opmaak.

Typ na Tot:

```
<span th:errors="*{tot}"></span>
```

Probeer dit. Typ bla in het vak van. Submit de form. Je ziet de foutboodschap:

```
Failed to convert property value of type java.lang.String
to required type java.math.BigDecimal for property van; ...
```

41.6.2 `messages.properties`

Je overschrijft die foutboodschap in een bestand `messages.properties`.

Een `properties` bestand kan standaard geen geaccentueerde tekens (ë, ...) bevatten.

Je wijzigt dit:

1. Kies het menu File, Settings.
2. Kies links Editor.
3. Kies daarbinnen File Encodings.
4. Kies UTF-8 bij Default encoding for properties files.
5. Kies OK.

Je maakt `messages.properties`:

1. Klik in het project overzicht met de rechtermuisknop op de folder `resources`.
2. Kies New File.
3. Typ `messages.properties`. Druk Enter.

Voeg een regel toe:

```
typeMismatch.java.math.BigDecimal=typ een bedrag
```

Het bestand bevat, zoals elk properties bestand, regels met een key, een = teken en een waarde. Als Spring een foutboodschap wil tonen, zoekt Spring volgende keys, tot hij er één vindt. Spring gebruikt de gevonden waarde als foutboodschap.

1. `typeMismatch.naamFormObject.naamVak` (`typeMismatch.vanTotPrijsForm.van`)
2. `typeMismatch.naamVak` (`typeMismatch.van`).
3. `typeMismatch.typeVariabeleDieBijVakHoort` (`typeMismatch.java.math.BigDecimal`)
4. `typeMismatch`

Stop de website. Start die opnieuw. Je doet dit enkel als je `messages.properties` toevoegde.

Je kan dit proberen.

Spring gebruikt de geparametriseerde constructor van de form class. Dit lukt enkel als de class één constructor heeft én dit een geparametriseerde constructor is.



Misschien wil je een class, die hieraan niet voldoet, gebruiken als form class.

Je kan de class toch gebruiken als form class als de class setters heeft.

Spring gebruikt dan de default constructor (constructor zonder parameters) en de setters om de waarden die de gebruiker typte over te brengen naar het form object.



Voer alle testen uit. Commit de sources. Publiceer op je remote repository.



Form

42 BEAN VALIDATION

Bean validation is een Java specificatie waarmee je objecten valideert.

42.1 Annotation

Je beschrijft de validatie met een annotation voor de te valideren private variabele. Voorbeeld:

```
public class Gemeente {
    @Min(1000)
    @Max(9999)
    private short postcode;
}
```

①
②

- (1) Je beschrijft een validatie voor de variabele postcode: de inhoud is ≥ 1000 .
- (2) Je beschrijft een tweede validatie voor de variabele postcode: de inhoud is ≤ 9999 .

42.2 Andere validation annotations

Annotation	Betekenis
@DecimalMin(<i>minimum</i>)	Zoals @Min, maar je geeft minimum als een String. Je gebruikt @DecimalMin als minimum cijfers na de komma bevat.
@DecimalMax(<i>maximum</i>)	Zoals @Max, maar je geeft maximum als een String.
@Digits(integer= <i>voorKomma</i> , fraction= <i>naKomma</i>)	De variabele heeft max. <i>voorKomma</i> cijfers voor de komma en max. <i>naKomma</i> cijfers na de komma.
@Email	De variabele moet de structuur van een e-mailadres hebben.
@NotBlank	De String variabele mag niet null zijn en moet meer dan enkel spaties bevatten.
@NotEmpty	De String variabele mag niet null zijn en mag niet leeg zijn. De variabele (array, List, Set, Map) mag niet null zijn en mag niet leeg zijn.
@Negative	De variabele moet een negatief getal zijn.
@NegativeOrZero	De variabele moet een negatief getal of 0 zijn.
@Positive	De variabele moet een positief getal zijn.
@PositiveOrZero	De variabele moet een positief getal of 0 zijn.
@NotNull	De variabele mag niet null bevatten.
@Null	De variabele moet null bevatten.
@Future	De variabele moet in de toekomst liggen.
@FutureOrPresent	De variabele moet vandaag zijn of in de toekomst liggen.
@Past	De variabele moet in het verleden liggen.
@PastOrPresent	De variabele moet vandaag zijn of in het verleden liggen.
@Pattern(regexp= <i>regularExpression</i>)	De String variabele moet passen bij regularExpression.
@Size(min= <i>min</i> , max= <i>max</i>)	<ul style="list-style-type: none"> Het aantal tekens in de String variabele moet liggen tussen <i>min</i> en <i>max</i> of. Het aantal elementen in de variabele (array, List, Set, Map) moet liggen tussen <i>min</i> en <i>max</i>.

De annotations (behalve @NotBlank en @NotEmpty) valideren een variabele enkel als die \neq null.

Er bestaan meerdere implementaties van de Bean validation specificatie. De Hibernate implementatie bevat naast de standaard annotations extra annotations. De interessantste:

Annotation	Betekenis
@CreditCardNumber	De String variabele moet de structuur en het controlegetal hebben van een betaalkaart nummer.
@EAN	De String variabele moet de structuur en het controlegetal hebben van een European Article Number (zoals je ziet op een barcode).
@Length(min=min, max=max)	De String variabele bevat minstens min en maximaal max tekens.
@Range(min=min, max=max)	De getal variabele heeft een waarde tussen min en max.

42.3 @Valid

Je typt @Valid bij een variabele die verwijst naar een object dat zelf ook properties met validation annotations bevat. Bean validation valideert dan ook die geneste validation annotations.

```
public class Persoon {
    @Range(min = 0, max = 69)
    private int aantalKinderen;
    @Valid
    private Adres adres;
}

public class Adres {
    @Range(min = 1000, max = 9999)
    private int postcode;
}
```

①

- (1) Bean validation valideert bij de validatie van een Persoon object ook zijn Adres object en controleert dus of postcode ligt tussen 1000 en 9999.

Je kan @Valid ook toepassen op een verzameling objecten.

Bean validation valideert dan alle objecten in die verzameling. Voorbeeld:

```
@Valid private Set<Adres> adressen; // valideer alle Adres objecten in de Set
```

42.4 messages.properties

Voeg aan message.properties foutboodschappen toe die horen bij Bean validation:

```
javax.validation.constraints.Null.message=moet leeg zijn
javax.validation.constraints.NotNull.message=mag niet leeg zijn
javax.validation.constraints.Min.message=minstens {value}
javax.validation.constraints.DecimalMin.message=minstens {value}
javax.validation.constraints.Email.message=ongeldig e-mail adres
javax.validation.constraints.Max.message=maximaal {value}
javax.validation.constraints.DecimalMax.message=maximaal {value}
javax.validation.constraints.Size.message=tussen {min} en {max}
javax.validation.constraints.Digits.message=max. {integer} voor en {fraction} na komma
javax.validation.constraints.Past.message=moet in verleden
javax.validation.constraints.PastOrPresent.message=moet vandaag zijn of in verleden
javax.validation.constraints.Future.message=moet in toekomst
javax.validation.constraints.FutureOrPresent.message=moet vandaag zijn of in toekomst
javax.validation.constraints.Pattern.message=moet voldoen aan patroon {regexp}
javax.validation.constraints.NotBlank.message=moet meer dan enkel spaties bevatten
javax.validation.constraints.NotEmpty.message=mag niet leeg zijn
javax.validation.constraints.Negative.message=moet negatief zijn
javax.validation.constraints.NegativeOrZero.message=moet negatief of nul zijn
javax.validation.constraints.Positive.message=moet positief zijn
javax.validation.constraints.PositiveOrZero.message=moet positief zijn of nul zijn
org.hibernate.validator.constraints.CreditCardNumber.message=ongeldig kredietkaartnummer
org.hibernate.validator.constraints.EAN.message=ongeldig artikelnummer
org.hibernate.validator.constraints.Length.message=minstens {min} / maximaal {max} tekens
org.hibernate.validator.constraints.Range.message=moet liggen tussen {min} en {max}
```

Bean validation vervangt de woorden {value}, {min}, {max}, ... in de teksten door de bijbehorende parameters die je bij de validation annotation typt.

42.5 Form object

Gebruik bean validation in het record VanTotPrijsForm.

Je typt bij een record de validation annotations voor de parameters van de constructor:

```
package be.vdab.luigi.forms;
import java.math.BigDecimal;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.PositiveOrZero;
public record VanTotPrijsForm(@NotNull @PositiveOrZero BigDecimal van,
    @NotNull @PositiveOrZero BigDecimal tot) {
}
```

42.6 @Valid in de controller

De validation annotations *beschrijven* de validaties, maar *doen* de validaties *niet*.

Je kan bijvoorbeeld een VanTotPrijsForm object maken en van op -7 plaatsen.

Je valideert een form object in de method vanTotPrijs van PizzaController.

Typ @Valid voor de parameter VanTotPrijsform form:

```
@GetMapping("vantotprijs")
public ModelAndView findByPrijsBetween(@Valid VanTotPrijsForm form, Errors errors) {
    ...
}
```

Spring valideert als volgt:

1. De gebruiker vult de vakken van de HTML form in. Hij submit de form.
2. Spring verwerkt de bijbehorende request. Spring roept daarbij de method vanTotPrijs op.
3. Spring probeert een VanTotPrijsForm object te maken met de constructor.
Spring valideert hierbij elke parameter met wat de gebruiker typte.,
op basis van de validation annotations.
Bij validatiefouten geeft de method hasErrors van de Errors parameter true.

Je kan dit proberen.



Voer alle testen uit. Commit de sources. Publiceer op je remote repository.



Bean validation

43 CLIENT SIDED VALIDATIE

Je valideert tot nu aan de kant van de server (server sided).

Deze validaties zijn essentieel: een hacker kan ze niet wijzigen of uitschakelen.

Je kan daarnaast ook client sided valideren: validatie op de browser.

Die heeft als voordeel dat ze zeer interactief is: de browser valideert voor de submit van de form.

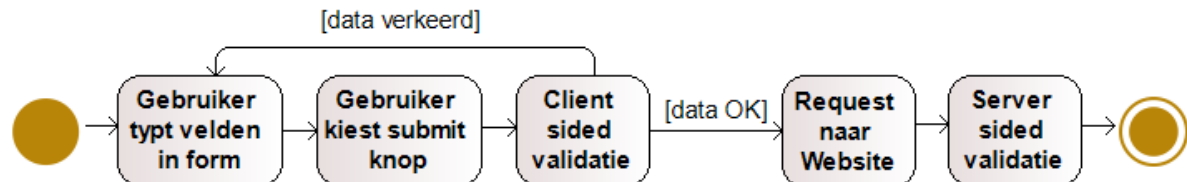
De browser moet voor de validaties dus niets naar de website doorsturen.



Een hacker kan gemakkelijk client sided validatie verwijderen!

Je doet daarom *zowel* server sided validatie *als* client sided validatie.

Volgorde van handelingen:



43.1 vantotprijs.html

Je voegt client sided validation toe:

Breid het invoervak "van" uit:

```
<input th:field="*{van}" autofocus type="number" required min="0">
```

❶

(1) Het invoervak kan enkel een getal bevatten.

Het is verplicht in te vullen.

Het getal moet minstens 0 zijn.

Breid ook het invoervak "tot" uit:

```
<input th:field="*{tot}" type="number" required min="0">
```

Je kan dit proberen.

Je typt een verkeerde waarde. De browser toont een foutmelding.

43.2 Server sided validatie testen

Omdat je client sided validatie hebt toegevoegd, is het moeilijk de server sided validatie te testen.

Je kan dit toch, door tijdelijk (voor de huidige request) de client sided validatie te verwijderen:

Je ziet zo ook hoe gemakkelijk een hacker client sided validatie kan manipuleren !

1. Open je website in Chrome.
2. Klik met de rechtermuisknop in het invoervak van.
3. Kies Inspect Element.
4. Dubbelklik `required=""` (onder in het scherm). Verwijder dit onderdeel.
5. Doe hetzelfde bij `min="0"`
6. Vul een getal in bij "tot".
7. Submit de form.

Je kan de client sided foutmelding aanpassen met JavaScript.



Voorbeeld: de foutmelding bij het invoervak van wijzigen naar Verplicht:

```
document.getElementsByName("van")[0].oninvalid = function() {
    this.setCustomValidity("Verplicht");
};
```



Voer alle testen uit. Commit de sources. Publiceer op je remote repository.



Client sided validatie

44 POST REQUEST

Als de gebruiker een form met method='post' submit, stuurt de browser een POST request.

Je maakt een form met method="post" als je met de form

- data toevoegt, wijzigt of verwijdt.
- een paswoord opvraagt.
- een bestand uploadt.

Je maakt als voorbeeld een pagina waarmee de gebruiker een pizza toevoegt.

Je gebruikt een object van de class Pizza als form object.

Je voegt validation annotations toe aan de class Pizza:

- @NotBlank voor de private variabele naam.
- @NotNull en @PositiveOrZero voor de private variabele prijs.

44.1 PizzaController

Je toont de toevoeg pagina als de gebruiker een GET request stuurt naar /pizzas/toevoegen/form

```
@GetMapping("toevoegen/form")
public ModelAndView toevoegenForm() {
    return new ModelAndView("toevoegen")
        .addObject(new Pizza(0, "", null, false));
}
```

❶

- (1) Je geeft een Pizza object als form object door aan de Thymeleaf pagina, onder de naam pizza. Deze naam is gebaseerd op de class van het object (Pizza), met de 1° letter in kleine letter.

44.2 toevoegen.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head th:replace="fragments::head(title='Toevoegen')"></head>
  <body>
    <nav th:replace="fragments::menu"></nav>
    <h1>Toevoegen</h1>
    <form th:object="${pizza}" method="post" th:action="@{/pizzas}">
      <label>Naam:<span th:errors="*{naam}"></span>
        <input th:field="*{naam}" autofocus required></label>
      <label>Prijs:<span th:errors="*{prijs}"></span>
        <input th:field="*{prijs}" type="number" required min="0"></label>
      <label>Pikant:<input type="checkbox" th:field="*{pikant}"></label>
      <input type="hidden" name="id" value="0">
      <button>Toevoegen</button>
    </form>
  </body>
</html>
```

❶

❷

- (1) Als de gebruiker de form submit, stuurt de browser een POST request naar /pizzas. De browser verstuurt in die request de inhoud van de invoervakken van de form.
- (2) Je gebruikt de class Pizza als form object. De Pizza constructor heeft onder andere een parameter id. Bij de form submit zal Spring die constructor oproepen om een Pizza te maken met de getypte data. Er moet dus ook een waarde zijn voor de parameter id. Je voegt daartoe een verborgen veld toe. Dat heeft de naam id, zoals de parameter van de Pizza constructor. De waarde is een dummy waarde: 0. Nadat je de pizza toevoegt aan de database, krijgt hij via de autonummering van de database een definitieve id.

Voeg een menu-punt naar de pagina toe in fragments.html in het fragment menu:

```
<li><a th:href="@{/pizzas/toevoegen/form}">Toevoegen</a></li>
```

44.3 Proberen

1. Open je website in Chrome.
2. Druk F12.
3. Kies het tabblad Network.
4. Typ correcte waarden bij naam en prijs. Kies Toevoegen.
5. Je ziet onderaan dat de browser een POST request stuurt. De response is een fout: PizzaController bevat nog geen method die de request verwerkt. Je doet dit straks. Kijk naar de URL in de adresbalk van de browser. De browser geeft de waarden van de invoervakken bij een POST request *niet* door in de query string achteraan in de URL. De browser geeft bij een POST request de waarden door in de request *body*. Je ziet die als je onderaan de request kiest en rechts Parameters kiest.

44.4 PizzaController

Maak een method die de POST request verwerkt:

```
@PostMapping
public ModelAndView toevoegen(@Valid Pizza pizza, Errors errors) {
    if (errors.hasErrors()) {
        return new ModelAndView("toevoegen");
    }
    pizzaService.create(pizza);
    return new ModelAndView("pizzas", "pizzas", pizzaService.findAll());
}
```

- (1) Je geeft aan dat de method, die na deze regel komt, POST requests verwerkt. Het zijn POST requests naar de URL bij @RequestMapping (pizzas).
- (2) Bij validatiefouten toon je de pagina met de form opnieuw. Spring geeft het Pizza object in de method parameter pizza zelf door aan de Thymeleaf pagina, onder de naam pizza. Jij moet dit object dus niet doorgeven.
- (3) Je voegt de pizza toe aan de database.
- (4) Je toont de pagina met alle pizza's.

Je kan dit proberen.

44.5 POST-REDIRECT-GET

Een POST request heeft een probleem als de gebruiker de pagina “refresht”. Je simuleert dit:

1. Voeg met Chrome een pizza toe. Je ziet daarna een lijst met alle pizza's.
2. Ververs de pagina in de browser (bvb. met de toets F5).
3. De browser herhaalt de laatste request. De website voegt daarbij de pizza nog eens toe: de laatste request was de POST request waarmee je een pizza toevoegde. Dit had de gebruiker niet verwacht: hij zag een pagina met alle pizza's en dacht bij een ‘refresh’ de pizza's opnieuw op te vragen.

44.5.1 Chrome

Je kan de volgorde van requests zien:

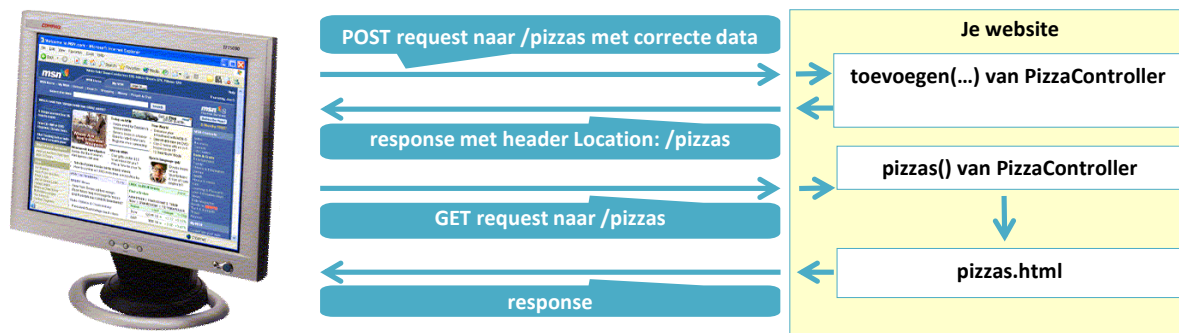
1. Surf naar de pagina Toevoegen.
2. Druk F12.
3. Kies het tabblad Network.
4. Vul de naam en de prijs van een pizza in. Kies Toevoegen.
5. Je ziet onder in het venster dat je een POST request verstuurd.

pizzas	POST	200
--------	------	-----
6. Druk F5 om de pagina te verversen.
7. Je ziet dat je dezelfde POST request verstuurd (en zo de pizza nog eens toevoegt).

pizzas	POST	200
--------	------	-----

44.5.2 Oplossing

1. Als je de request correct kan verwerken, stuur je een redirect response. Die bevat de status code 302 (Found), een lege body en een Location header met een URL. Dit is bij ons de URL met alle pizza's: /pizzas.
2. De browser ontvangt die response. Hij stuurt direct een GET request naar de URL in de response header Location: /pizzas. Je verwerkt die request in de method pizzas. Die leest toont alle pizza's. Als de gebruiker de pagina 'refresht', stuurt de browser die GET request opnieuw. De gebruiker ziet de pizza's opnieuw.



44.5.3 PizzaController

Wijzig de method toevoegen:

@PostMapping

```

public String toevoegen(@Valid Pizza pizza, Errors errors) {
    if (errors.hasErrors()) {
        return "toevoegen";
    }
    pizzaService.create(pizza);
    return "redirect:/pizzas";
}
  
```

①

②

- (5) De gebruiker typte verkeerde data. Je toont de pagina met de form (met de verkeerde data) opnieuw, zodat de gebruiker die data kan corrigeren. Spring geeft het Pizza object in de method parameter pizza zelf door aan de Thymeleaf pagina, onder de naam pizza. Jij moet dit object dus niet doorgeven. Het is belangrijk hier geen redirect te doen. Anders moet de gebruiker *alle* data opnieuw typen, ook de data waarin hij geen fouten maakte !
- (6) Je typt redirect: voor een URL. Spring maakt dan een redirect response met die URL.

44.5.4 Proberen

1. Surf naar de pagina Toevoegen.
2. Druk F12.
3. Vul de naam en de prijs van een pizza in. Kies Toevoegen.
4. Je ziet onder in het venster dat je met die keuze een POST request verstuurd:

pizzas POST 302

 De response heeft de status code 302 en een response header Location. Je ziet de response headers als je de regel pizzas POST 302 aanklikt:

Location: http://localhost:8080/pizzas

 De browser stuurt, bij de ontvangst van die response, direct een GET request naar de URL in die response header Location:

pizzas GET 200

 De response is de lijst met pizza's.
5. Ververs de pagina.
6. Die keuze stuurt terug een GET request naar /pizzas en voegt geen pizza toe.

pizzas GET 200

44.6 RedirectAttributes

Je wil soms data doorgeven aan de pagina waar naar je redirect.

Die data worden parameters in de URL waarnaar je redirect.

Je redirect in het voorbeeld naar de URL /pizzas.

Je voegt aan die URL een parameter toe met de naam idNieuwePizza.

De waarde is de id van de nieuwe pizza: 777.

De URL wordt:

/pizzas?idNieuwePizza=777.

44.6.1 PizzaController

Voeg een parameter toe aan de method toevoegen:

```
..., RedirectAttributes redirect) {
```

❶

(1) Je voegt met RedirectAttributes parameters toe aan de redirect URL.

Wijzig de opdracht pizzaService.create(pizza);:

```
redirect.addAttribute("idNieuwePizza", pizzaService.create(pizza));
```

❶

(1) Je voegt aan de URL een parameter toe met de naam toegevoegd.

De inhoud is de id van de toegevoegde pizza.

44.6.2 pizzas.html

Voeg regels toe voor <h1>:

```
<div th:if="${param.idNieuwePizza}" class="boodschap">Pizza is toegevoegd.  
Zijn nummer:<th:block th:text="${param.idNieuwePizza}"></th:block></div>
```

❶

(1) param.idNieuwePizza verwijst naar de request parameter idNieuwePizza

Je kan dit proberen.



Je kan met RedirectAttributes ook een path variabele invullen in de redirect URL.

Fictief voorbeeld:

```
redirect.addAttribute("id", id);
```

```
return "redirect:/pizzas/toegevoegd/{id}";
```

zal een redirect doen naar /pizzas/toegevoegd/17 als de variabele id 17 bevat.

44.7 Dubbele submit vermijden

De gebruiker kan een trage internet verbinding hebben.

Als hij Toevoegen kiest, duurt het dan lang voor de request verstuurd is en de response aankomt.

Als de gebruiker ongeduldig is, kiest hij in de tussentijd nog eens Toevoegen kiezen.

Hij verstuurt dan nog een keer dezelfde request en voegt de pizza nog een keer toe.

44.7.1 Proberen

Je simuleert dit met de Chrome browser:

1. Open de pagina om een pizza toe te voegen.
2. Druk F12.
3. Kies onder aan het tabblad Network.
4. Kies Online in de rij onder de tabs. Wijzig dit naar Slow 3G.
5. Typ de naam en de prijs van een pizza.
6. Kies Toevoegen. Het verwerken van de request gaat traag.
7. Kies nog eens Toevoegen.
8. Je ziet uiteindelijk de pagina met pizza's. Je pizza is 2 keer toegevoegd.

44.7.2 JavaScript

Je vermijdt dit. Je disable met JavaScript de submit knop bij het submitten van de form.

Maak in de folder static een folder js. Maak daarin preventDoubleSubmit.js:

```
document.querySelector("form").onsubmit = function() {
  this.querySelector("button").disabled = true;
};
```

❶
❷

- (1) Je zoekt het <form> element. Je koppelt een functie aan het onsubmit event.
De browser voert die functie uit als de gebruiker de form submit.
- (2) De expressie this geeft de gesubmitte form.
Je zoekt in die form de submit knop. Je disable die knop.

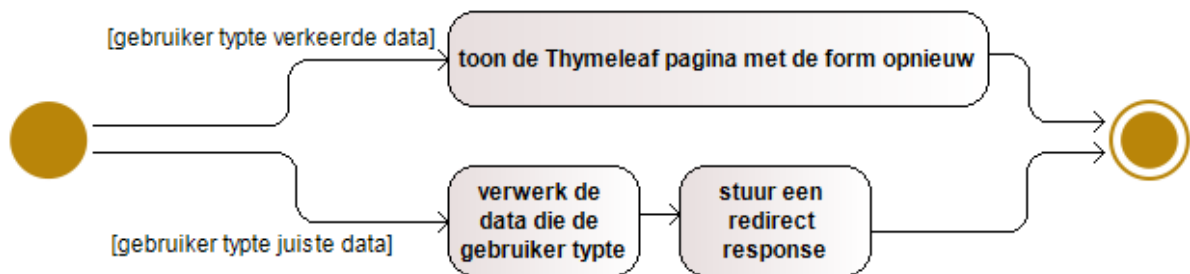
Typ in toevoegen.html na <body>:

```
<script th:src="@{/js/preventDoubleSubmit.js}" defer></script>
```

Je kan dit proberen.

44.8 Overzicht

Een POST request verwerken in de method van een controller:



Doe geen redirect als de gebruiker verkeerde data typte.
De gebruiker zou dan alle velden van de form opnieuw moeten typen, ook de velden die hij correct invulde. Dit zou zeer gebruikersonvriendelijk zijn!



Voer alle testen uit. Commit de sources. Publiceer op je remote repository.



Snack wijzigen

45 SESSION

HTTP is een stateless protocol is: elke request is een zelfstandige handeling.
 Zijn verwerking hangt niet af van de verwerking van vorige requests.
 Variabelen die je maakt tijdens het verwerken van een request,
 verdwijnen uit het geheugen nadat de request verwerkt is.
 Ze zijn niet meer beschikbaar bij een volgende request.

45.1 Session

Je moet sommige data *toch* onthouden over requests heen, zoals gebruikersnaam of winkelmandje.
 Je gebruikt voor die data een session: een stuk webserver geheugen met data voor één gebruiker.
 Één session kan meerdere data bevatten. Elk data heet een session attribuut.
 Een session kan bijvoorbeeld 2 attributen bevatten: gebruikersnaam en winkelmandje.



Één controller object verwerkt de requests van *alle* gebruikers.
 Je onthoudt een winkelmandje niet zomaar in een private variabele van die controller.
 Alle gebruikers zouden *hetzelfde* winkelmandje delen!

45.2 Serializable

De objecten in een session attribuut moeten Serializable implementeren. De redenen:

- Session persistence
- Session replication

45.3 Session persistence

De webserver onthoud sessions in zijn RAM geheugen.

De webserver doet volgende stappen om de sessions niet te verliezen bij een herstart:

1. Hij schrijft de sessions via serialization naar een tijdelijk bestand.
Dit werkt enkel als de objecten Serializable zijn.
2. Hij herstart.
3. Hij leest de sessions terug uit het tijdelijk bestand via deserialization.

45.4 Session replication

Een webserver stuurt in een web farm elke session wijziging naar de andere webserver in de farm.
 Dit heet session replication.

Alle webserver moeten dezelfde sessions bevatten:

- Mieke legt appels in haar mandje.
- Server 1 verwerkt die request. Hij onthoudt het mandje als een session attribuut.
- Server 1 stuurt die session naar server 2.
- Server 2 onthoudt die session.
- Mieke vraagt haar mandje te zien.
- Server 2 verwerkt die request: hij haalt het mandje uit de session van Mieke.

Je kan objecten enkel over het netwerk versturen als ze Serializable zijn.

45.5 Kleine data

Men raadt aan de data in session zo klein mogelijk te houden:

- Bij veel gelijktijdige gebruikers bevat het geheugen van de webserver veel sessions. Als elk van die sessions veel data bevat, kom je geheugen te kort.
- De webserver van een web farm wisselen de sessions onderling uit. Als de sessions weinig data bevat, gebeurt dit uitwisselen performant.

Voorbeelden van kleine data in session:

- Als de gebruiker per artikel één stuk kan bestellen, houd je in het winkelmandje artikelnummers bij, niet volledige artikel objecten.
- Als de gebruiker per artikel meerdere stuks kan bestellen, houd je in het in het winkelmandje enkel artikelnummers en -aantallen bij.

45.6 Session identificatie

Wanneer een gebruiker een eerste request stuurt naar een website, maakt de webserver voor hem een session met een unieke identificatie: de session ID.

De session van Jan krijgt bijvoorbeeld een session ID 0AAB9C8DE666.

De session van Mieke krijgt bijvoorbeeld een session ID DD12EE78A4B5.

De webserver stuurt die sessionID in de response naar de browser. Dit kan op 2 manieren:

- **Optie 1: tijdelijke cookies zijn ingeschakeld in de browser.**
De webserver stuurt de session ID als een tijdelijke cookie. Die heeft de naam JSESSIONID. De webserver leest de cookie bij elke volgende request. De webserver bepaalt daarmee de session die bij de huidige gebruiker hoort. Bij Jan heeft de cookie de waarde 0AAB9C8DE666. De webserver haalt de session met die ID uit zijn RAM geheugen. De webserver biedt je code die session aan zodat je hem kan lezen of wijzigen. Bij Mieke heeft de cookie de waarde DD12EE78A4B5. De webserver haalt de session met die ID uit zijn RAM geheugen en biedt je die aan.
- **Optie 2: tijdelijke cookies zijn uitgeschakeld in de browser.**
De webserver voegt de session ID toe aan elke URL in de response. Dit heet URL rewriting. De URL /pizzas wordt bij Jan /pizzas;jsessionid=0AAB9C8DE666. Zo'n URL kan een onderdeel zijn van een hyperlink tag of van een form tag. Als de gebruiker een hyperlink aanklikt of een form submit, verwerkt de webserver de bijbehorende request. De webserver leest de session ID in de request URL. Als de request URL /pizzas;jsessionid=0AAB9C8DE666 is, weet de webserver dat de session ID van de gebruiker 0AAB9C8DE666 is en haalt de session van Jan op uit zijn RAM geheugen. De webserver biedt je code die session aan.

De webserver weet bij de eerste request niet of tijdelijke cookies zijn ingeschakeld.

Hij gebruikt daarom in de eerste response beide strategieën: een tijdelijke cookie én URL rewriting.

Als de volgende request een cookie JSESSIONID bevat, zijn cookies ingeschakeld.

De webserver gebruikt dan geen URL rewriting meer.



De webserver stuurt enkel de session ID naar de browser, nooit de session data. Session data mag dus confidentiële data bevatten.

45.7 Webserver verwijdert een session

De webserver verwijdert een session, die gedurende een aantal minuten niet aangesproken werd, uit zijn geheugen. Men zegt dat de session 'vervalt'.

Dit gebeurt bijvoorbeeld omdat de gebruiker de browser sluit of naar een andere website gaat.

Elk webserver merk heeft een eigen waarde voor het aantal minuten waarna een session vervalt.

Bij Tomcat is dit 30 minuten.

45.8 Voorbeeld 1

De gebruiker typt zijn email adres. Je onthoudt dit in de session van de gebruiker. Zolang de gebruiker op de website blijft, kan je zijn email adres terug lezen uit de session.

45.8.1 Identificatie

Je beschrijft in een class de data die je wil onthouden in de session van de gebruiker.

```
package be.vdab.luigi.sessions;
// enkele imports
@Component
@SessionScope
public class Identificatie implements Serializable {
    private static final long serialVersionUID = 1L;
    @Email private String emailAdres;
    // je maakt een getter en een setter voor emailAdres
}
```

- (1) De data die je onthoudt in de session moet een Spring bean zijn.
- (2) @Component maakt standaard een *singleton* bean van een class. Alle gebruikers delen die bean. Bij session moet elke gebruiker echter zijn eigen bean hebben. Je doet dit met @SessionScope.
- (3) De class moet Serializable implementeren.
- (4) De class moet een default constructor, getters en setters hebben.

45.8.2 IdentificatieController

```
package be.vdab.luigi.controllers;
// enkele imports
@Controller @RequestMapping("identificatie")
class IdentificatieController {
    private final Identificatie identificatie;
    IdentificatieController(Identificatie identificatie) {
        this.identificatie = identificatie;
    }
    @GetMapping public ModelAndView identificatie() {
        return new ModelAndView("identificatie").addObject(identificatie);
    }
    @PostMapping
    public String identificatie(@Valid Identificatie identificatie, Errors errors){
        if (errors.hasErrors()) {
            return "identificatie";
        }
        this.identificatie.setEmailAdres(identificatie.getEmailAdres());
        return "redirect:/";
    }
}
```

- (1) Je injecteert de identificatie van de huidige gebruiker (uit zijn session).
- (2) Je geeft die identificatie door aan de Thymeleaf pagina. Je gebruikt het daar als form object. Er zijn twee mogelijkheden. Mogelijkheid 1: de gebruiker surft voor het eerst naar de pagina. Spring maakte dan voor die gebruiker een nieuwe session, met een nieuwe Identificatie. Mogelijkheid 2: de gebruiker surft voor een volgende keer naar de pagina. Spring haalde dan de session van die gebruiker uit het geheugen. Die session bevat het Identificatie object van de vorige requests van die gebruiker. Je kan uit dit object het email adres lezen dat de gebruiker bij vorige requests typte.
- (3) De method bij 4 verwerkt de submit van de form. Hierbij *wijzigt* data in de session. Je submit daarom de form met method="post", niet met method="get".
- (4) De method verwerkt de request als de gebruiker de form submit. Spring maakte daarbij een *nieuwe* Identificatie object. Spring vulde daarin het email adres met de setter. Je valideert dit object.
- (5) Je brengt het getypte adres over naar het Identificatie object in de session van de gebruiker. Het is zo is ter beschikking bij volgende requests van de gebruiker.

45.8.3 identificatie.html

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head th:replace="fragments::head(title='Identificatie')"></head>
  <body>
    <script th:src="@{/js/preventDoubleSubmit.js}" defer></script>
    <nav th:replace="fragments::menu"></nav>
    <h1>Identificatie</h1>
    <form th:object="${identificatie}" method="post" th:action="@{/identificatie}">
      <label>Email adres:<span th:errors="*{emailAdres}"></span>
        <input th:field="*{emailAdres}" autofocus required type="email"></label>
      <button>OK</button>
    </form>
  </body>
</html>
```

Voeg een menu-punt naar de pagina toe in fragments.html in het fragment menu:

```
<li><a th:href="@{/identificatie}">Identificatie</a></li>
```

Je kan dit proberen. Je identificeert je. Je bezoekt andere pagina's van de website. Je gaat terug naar de identificatiepagina. Je ziet dat je email adres bewaard is (in de gebruiker session).

Je kan meerdere gebruikers van de website simuleren.

Je opent de website in browsers met verschillende merken (Chrome, Edge, Firefox, ...).

Je identificeert je in elke browser met een ander email adres.

45.9 Voorbeeld 2

De gebruiker kan pizza's in een mandje leggen. Je onthoudt dit mandje in de gebruiker session.

45.9.1 pizza.html

Je voegt na </dl> een form met een knop toe. De gebruiker legt daarmee de pizza in zijn mandje:

```
<form method="post" th:action="@{/mandje/{id}(id={id})}">
  <button>In mandje</button>
</form>
```

45.9.2 Mandje

Je beschrijft in een class de data die je wil onthouden in de session van de gebruiker.

```
package be.vdab.luigi.sessions;
// enkele imports
@Component @SessionScope
public class Mandje implements Serializable {
  private static final long serialVersionUID = 1L;
  private final Set<Long> ids = new LinkedHashSet<>();
  public void voegToe(long id) {
    ids.add(id);
  }
  // getter voor ids
}
```

①

(1) Je minimaliseert de data in de session. Je onthoudt geen Pizza objecten, enkel pizza id's.

45.9.3 MandjeTest

Je test met JUnit de werking van de class Mandje:

```
package be.vdab.luigi.sessions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class MandjeTest {
  private Mandje mandje;
  @BeforeEach void beforeEach() {
    mandje = new Mandje();
  }
}
```

```

@Test
void eenNieuwMandjeIsLeeg() {
    assertThat(mandje.getIds()).isEmpty();
}
@Test
void nadatJeEenItemInHetMandjeLegtBevatDitMandjeEnkelDitItem() {
    mandje.voegToe(10L);
    assertThat(mandje.getIds()).containsOnly(10L);
}
@Test
void jeKanEenItemMaarÉénKeerToevoegenAanHetMandje() {
    mandje.voegToe(10L);
    mandje.voegToe(10L);
    assertThat(mandje.getIds()).containsOnly(10L);
}
@Test
void nadatJeTweeItemsInHetMandjeLegtBevatDitMandjeEnkelDieItems() {
    mandje.voegToe(10L);
    mandje.voegToe(20L);
    assertThat(mandje.getIds()).containsOnly(10L, 20L);
}
}

```

45.9.4 MandjeController

```

package be.vdab.luigi.controllers;
// enkele imports
@Controller @RequestMapping("mandje")
class MandjeController {
    private final Mandje mandje;
    private final PizzaService pizzaService;
    // constructor met parameters
    @PostMapping("/{id}") public String voegToe(@PathVariable long id) {
        mandje.voegToe(id);
        return "redirect:/mandje";
    }
    @GetMapping public ModelAndView toonMandje() {
        return new ModelAndView("mandje",
            "pizzas", pizzaService.findByIds(mandje.getIds()));
    }
}

```

(1) Je leest in de database *enkel* de pizza's die in het mandje liggen. Dit is een performante aanpak.

45.9.5 mandje.html

```

<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
<head th:replace="fragments::head(title='Mandje')"></head>
<body>
    <nav th:replace="fragments::menu"></nav>
    <h1>Mandje</h1>
    <th:block th:if="not ${pizzas.isEmpty()}">
        <table th:replace="fragments::pizzaTabel(pizzas=${pizzas})"></table>
    </th:block>
    <div th:if="${pizzas.isEmpty()}" class="boodschap">Je mandje is leeg.</div>
</body>
</html>

```

Voeg een menu-punt naar de pagina toe in fragments.html in het fragment menu:

```
<li><a th:href="@{/mandje}">Mandje</a></li>
```

Je kan dit proberen.

45.10 Session fixation



Session fixation is een techniek van een hacker. Hij ziet een session id in een URL. Hij stuurt een request naar de website. Hij ziet zijn session id in de response URL. Hij stuurt een mail naar een gebruiker. De mail bevat een hyperlink naar de website. De hyperlink bevat de session id van de hacker. De gebruiker volgt de hyperlink. Hij voert daarbij een handeling (voorbeeld geld overschrijven) uit op de website in naam van de hacker.

Je verhindert session fixation met een extra regel in application.properties:

```
server.servlet.session.tracking-modes=cookie
```



(1) Spring stuurt de session id naar de browser altijd als een tijdelijke cookie, nooit in de URL.



Voer alle testen uit. Commit de sources. Publiceer op je remote repository.



Zoek de friet en Raad de saus

46 @CONTROLLERADVICE

Je hebt bepaalde session data in *elke* controller nodig. Het zou omslachtig zijn om in *elke* controller

1. een private variabele toe te voegen die de session data voorstelt.
2. een parameter aan de constructor toe te voegen waarmee je de session data injecteert.
3. de session data in elke @GetMapping en elke @PostMapping method aan de Thymeleaf pagina door te geven.

Je kan dit beter centraliseren in één class:

```
package be.vdab.luigi.controllers;
// enkele imports
@ControllerAdvice                                ❶
class MyControllerAdvice {
    private final Identificatie identificatie;      ❷
    MyControllerAdvice(Identificatie identificatie) { ❸
        this.identificatie = identificatie;
    }
    @ModelAttribute                                ❹
    void extraDataToevoegenAanModel(Model model) {    ❺
        model.addAttribute(identificatie);           ❻
    }
}
```

- (1) Je typt @ControllerAdvice voor een class die bij elke request extra data doorgeeft aan Thymeleaf. @ControllerAdvice maakt van de class ook een Spring bean. Je kan dus dependency injection toepassen in de class.
- (2) Deze private variabele is de session data die je aan de Thymeleaf pagina wil doorgeven.
- (3) Je injecteert de bean met die session data.
- (4) Je typt @ModelAttribute voor een method waarin je data doorgeeft aan de Thymeleaf pagina.
- (5) Je voorziet de method van een Model parameter. Dit is de "Model" uit ModelAndView.
- (6) Je geeft met de addAttribute method van het Model data door aan de Thymeleaf pagina.

Vanaf nu roept Spring automatisch bij elke request de method extraDataToevoegenAanModel op. Je geeft dus bij elke request de session data door aan de Thymeleaf pagina.

Voeg in index.html volgende regel toe voor </body>:

```
<div th:text="{identificatie.emailAdres}"></div>
```

Je kan dit proberen.

47 OPMAAK




47.1 Getal

Je baseert de opmaak van getallen op het land van de gebruiker. Voorbeelden:

- Je toont getallen aan een Belgische gebruiker met een , tussen eenheden en decimalen, en een . tussen duizendtallen (1.000,23).
- Je toont getallen aan een gebruiker uit de USA met een . tussen eenheden en decimalen, en een , tussen duizendtallen (1,000.23).

Spring helpt je bij de opmaak. Spring baseert het land van de gebruiker op de request header `Accept-Language`. Die bevat de taal en het land van de gebruiker. Als dit bijvoorbeeld `n1-be` is, betekent dit dat de gebruiker Nederlands spreekt en in België woont. De header kan *meerdere* items bevatten. De geprefereerde combinatie staat vooraan. Voorbeeld: `n1-be`, `en-us`.

De gebruiker bepaalt de inhoud van de header in de browser instellingen. Bij Chrome:

1. Kies rechts boven .
2. Kies Settings.
3. Kies links Advanced en daarbinnen Languages.
4. Kies in het midden  naast Language.
5. Je kan een taal of een taal-land combinatie toevoegen met Add Languages.
6. Je kiest de volgorde van de talen met  naast een taal.

47.1.1 @NumberFormat

Je definieert de opmaak van een `getal` variabele (`int`, `long`, `BigDecimal`, ...) met `@NumberFormat` voor die variabele. Je geeft een parameter `style` of een parameter `pattern` mee.

`style` heeft als type de enum `Style`, met volgende mogelijkheden:

Style	Betekenis	Getal	Opgemaakt (Belgisch formaat)
NUMBER	puntjes tussen 1000 tallen	2100,12	2.100,12
CURRENCY	zoals NUMBER, mét muntcode	2100,12	2.100,12 €
PERCENT	In percentage vorm	0.37	37%

Voorbeeld: `@NumberFormat(style = Style.NUMBER)`

`style` is soms beperkt. Je kan bvb. het aantal cijfers na de komma niet definiëren.

Je kan bijvoorbeeld `50` niet tonen als `50.00`, `style` toont `50` altijd als `50`.

Als je dit wil, vervang je `style` door `pattern`. Je geeft dan een opmaak patroon mee.

Je gebruikt in dit patroon volgende tekens:

- 0 cijfer altijd tonen
- # cijfer enkel tonen als dat nodig is
- , scheidingsteken tussen 1000 tallen
- . scheidingsteken tussen eenheden en decimalen

Voorbeelden:

pattern	Getal	Opgemaakt (Belgisch formaat)
<code>#,##0.00</code>	1000	1.000,00
<code>#,##0.00</code>	1000000	1.000.000,00
<code>#,##0.00</code>	50	50,00

Typ in de class `Pizza` vóór `private BigDecimal prijs`

```
@NumberFormat(pattern = "0.00")
```

47.1.2 Thymeleaf

Thymeleaf past de opmaak enkel toe als je de expressie omringt met dubbele accolades.

Je wijzigt `fragments.html` en `pizza.html`. Je vervangt `*{prijs}` door `*{{prijs}}`.

Je kan dit proberen.

Je toont in `pizza.html` ook de prijs in \$. Je hebt daarbij een uitdaging: de prijs is geen private variabele (van een class) waarvoor je `@NumberFormat` kan typen.

Je laat in die uitzonderlijke situatie de opmaak helemaal door Thymeleaf doen.

Vervang `${inDollar}` door `${#numbers.formatDecimal(inDollar,1,2)}` ❶

(1) Je roept een object op dat ingebakken is in Thymeleaf. Dit object heeft de naam `numbers`.

Als je een ingebakken object oproept, typ je voor dit object #.

Dit object heeft een method `formatDecimal`, waarmee je een getal opmaakt.

De 1° parameter is het op te maken getal.

De 2° parameter is het minimum aantal cijfers voor de komma in het opgemaakte getal.

De 3° parameter is het aantal cijfers na de komma in het opgemaakte getal.

Je kan dit proberen.

47.2 Datum en tijd

Je baseert de opmaak van datums op het land van de gebruiker. Voorbeelden:

- Je toont datums aan Belgische gebruikers als dag/maand/jaar (31/1/2019).
- Je toont datums aan Amerikaanse gebruikers als maand/dag/jaar (1/31/2017)

Je definieert de opmaak van een `Date`, `LocalDate`, `LocalTime` of `LocalDateTime` variabele met `@DateTimeFormat` voor die variabele. `@DateTimeFormat` heeft een parameter `style`.

Hij definieert de opmaak van het datum deel en/of het tijd deel.

Het 1° teken in `style` definieert de opmaak van het datum deel:

Teken	Betekenis	Toegepast op 1/4/2010 (Europees formaat)
S	Short style	1/04/10
M	Medium style	1-apr-2010
L	Long style	1 april 2010
F	Full style	donderdag 1 april 2010
-	Datum deel niet tonen	

Het 2° teken in `style` definieert de opmaak van het tijd deel:

Teken	Betekenis	Toegepast op 14:28:56 (Europees formaat)
S	Short style	14:28
M	Medium style	14:28:56
L	Long style	14:28:56 CET (CET betekent Central European Time)
F	Full style	14:28 u. CET
-	Tijd deel niet tonen	

Voorbeeld: je toont het datum deel in short style. Je toont het tijd deel niet.

```
@DateTimeFormat(style = "S-")
```

Als je geen parameter meegeeft aan `@DateTimeFormat`, is de default: `style = "SS"`

Voorbeeld: typ in de class `Persoon` voor de variabele `geboorte`:

```
@DateTimeFormat(style = "S-")
```

Je toont de datum opgemaakt in `index.html`. Voeg regels toe voor `</dl>`:

```
<dt>Geboorte</dt>
<dd th:text="*{{geboorte}}"></dd>
```

❶

(1) Thymeleaf past de opmaak toe als je *dubbele* accolades gebruikt.

Je kan dit proberen.



Voer alle testen uit. Commit de sources. Publiceer op je remote repository.

48 CUSTOM ERROR PAGE

48.1 404

Een browser request kan een fout veroorzaken.

Het bekendste voorbeeld is een request naar een onbestaande URL.

De webserver stuurt dan een response met een status code 404 (Not Found)

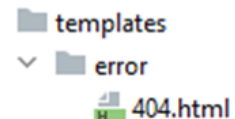
en een body (HTML) die Spring zelf aanmaakt.

Surf naar <http://localhost:8080/xxx>. Je ziet de “lelijke” response van Spring.

Je kunt een eigen mooiere foutpagina associëren met een status code.

Als een fout met die status code optreedt, stuurt de webserver een response met jouw pagina.

Maak in de map templates een map error. Die naam ligt vast.



Maak daarin 404.html. De naam van de pagina is de status code waarmee je de pagina associeert.

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head th:replace="fragments::head(title='Pagina niet gevonden')"></head>
  <body>
    <nav th:replace="fragments::menu"></nav>
    <h1>We vinden de gevraagde pagina niet.</h1>
  </body>
</html>
```

Je kan dit proberen.

48.2 500

Als in je applicatie een exception optreedt die je niet opvangt, stuurt de webserver een response met status code 500 (Internal Server Error) en een body met informatie over de exception.

Je kan dit zien: pauzeer MySQL:

1. Kies in de MYSQL Workbench links het tabblad Administration.
2. Kies daar Startup / Shutdown.
3. Kies in het midden Bring Offline.

Probeer in de browser de pagina met de pizza's te openen.

Je krijgt na een tijdje een pagina met technische informatie over de exception.

Dit geeft problemen:

- ⊖ Een gewone gebruiker is overdonderd door de technische informatie.
- ⊖ Een hacker leert de binnenkant van de website kennen. Hij weet dat je MySQL gebruikt. Hij kan zijn hacker pogingen op MySQL concentreren.

Om dit op te lossen maak je in de map error ook 500.html:

```
<!doctype html>
<html lang="nl" xmlns:th="http://www.thymeleaf.org">
  <head th:replace="fragments::head(title='Technische storing')"></head>
  <body>
    <nav th:replace="fragments::menu"></nav>
    <h1>Technische storing</h1>
    <div>We kunnen dit niet uitvoeren. Gelieve de helpdesk te contacteren.</div>
  </body>
</html>
```

Je kan dit proberen. Je ziet de fout informatie wel nog in de Tomcat log (in IntelliJ).



Commit de sources. Publiceer op je remote repository.

49 PRODUCTIE

Als je website klaar is bezorg je hem aan de netwerkverantwoordelijke.
Hij installeert je website op de productieserver.

Er zijn 2 mogelijkheden:

1. Je mag de website in zijn huidige vorm (met embedded Tomcat) bezorgen.
2. De website moet draaien op een aparte webserver.
Je website moet dan geen embedded Tomcat bevatten.

49.1 Website met een embedded Tomcat

Je bezorgt je website aan de netwerkverantwoordelijke als een fat JAR.
Dit bestand bevat naast jouw code ook libraries die je gebruikt: Spring, Thymeleaf, ...

Maak de fat JAR:

1. Stop de website.
2. Klik rechts op de verticale tab Maven.
3. Open het onderdeel Lifecycle.
4. Dubbelklik install. Maven maakt de fat JAR. Dit duurt enkele seconden.

Je vindt de fat JAR met de Windows File Explorer in de target folder van je project als het bestand `luigi-0.0.1-SNAPSHOT.jar`.

Simuleer dat jij de verantwoordelijke bent die de fat JAR installeert.

1. Maak een folder met de naam `websites` in de root van de harde schijf.
2. Kopieer de fat JAR in die folder.
3. Hernoem de fat JAR daar voor het gemak naar `luigi.jar`.
4. Start een Command Line venster.
5. Plaats je in de folder `websites` met de opdracht `cd \websites`.
6. Start de website met de opdracht `java -jar luigi.jar`.
Je ziet de diagnostische boodschappen bij het starten van je website.

De website draait. Je kan er naar surfen op de URL <http://localhost:8080>.

Sluit het Command line venster. Je stopt zo je website.

49.2 Website op aparte webserver

49.2.1 SpringBootServletInitializer

Je eeft de class LuigiApplication van SpringBootServletInitializer.

Je kan dan geavanceerd configureren hoe je website op een aparte webserver moet draaien.

Je hebt deze geavanceerde configuratie zelden nodig.

Wijzig de regel `public class LuigiApplication {` naar `public class LuigiApplication extends SpringBootServletInitializer {`

Voeg een method toe, waarin je de geavanceerde configuratie zou kunnen doen:

```
@Override
protected SpringApplicationBuilder configure(
    SpringApplicationBuilder application) {
    return application.sources(LuigiApplication.class);
}
```

49.2.2 WAR

Definieer in pom.xml dat het eindproduct van je project een WAR bestand moet zijn.

Voeg een regel toe na `</version>`: `<packaging>war</packaging>`

49.2.3 Embedded Tomcat weglaten uit WAR

Je website heeft zijn embedded Tomcat webserver niet nodig. Geef dit aan in pom.xml:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
```

❶

(1) Je definieert met provided dat de Tomcat software er al is tijdens het uitvoeren.

Maven neemt deze software dan niet op in je WAR.

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

49.2.4 WAR maken

1. Klik rechts op de verticale tab Maven, tenzij het Maven venster al open staat.
2. Open het onderdeel Lifecycle.
3. Dubbelklik install. Maven maakt de WAR. Dit duurt enkele seconden.

Je vindt de WAR met de Windows File Explorer in de target folder van je project als het bestand `luigi-0.0.1-SNAPSHOT.war`. Je hernoemt dit bestand naar `luigi.war`.

49.2.5 WAR installeren op de aparte webserver.

Je volgt de stappen in het hoofdstuk “Tomcat” in het begin van de cursus om de aparte Tomcat te starten en daarop je WAR te installeren.

50 STAPPENPLAN

Je kan dit stappenplan volgen om een website te maken:

1. Maak regels in application.properties:


```
spring.datasource.url=...
spring.datasource.username=...
spring.datasource.password=...
logging.level.org.springframework.jdbc.core.JdbcTemplate=DEBUG
logging.level.org.springframework.jdbc.core.simple.SimpleJdbcInsert=DEBUG
logging.level.org.springframework.jdbc.core.StatementCreatorUtils=TRACE
server.servlet.session.tracking-modes=cookie
spring.datasource.hikari.transaction-isolation=TRANSACTION_READ_COMMITTED
```
2. Maak spring.properties in src/test/resources:


```
spring.test.constructor.autowire.mode=all
```
3. Maak een DataSourceTest.
4. Doe volgende stappen voor de eerste pagina van de website:
 - a. Voeg domain classes (die personen en/of dingen uit de werkelijkheid voorstellen) toe die je nodig hebt in die pagina.

- i. Plaats een berekening die met de class te maken heeft in de class zelf, niet in controllers, services, repositories of Thymeleaf pagina's:

```
...
public class Artikel {
    ...
    @NumberFormat(pattern = "0.00")
    private final BigDecimal prijsExclusiefBTW;
    @NumberFormat(pattern = "0.00")
    private final BigDecimal btwPercentage;
    @NumberFormat(pattern = "0.00")
    public BigDecimal getPrijsInclusiefBTW() {
        return prijsExclusiefBTW.multiply(BigDecimal.ONE
            .add(btwPercentage.divide(BigDecimal.valueOf(100), 2,
                RoundingMode.HALF_UP)));
    }
}
```

Voordeel 1: je kan de berekening overal oproepen.

Voordeel 2: als de berekening wijzigt, doe je die wijziging één keer

Test de correcte werking van zo'n method met een unit test.

- ii. Plaats ook "intelligentie" die met de class te maken heeft in de class zelf, niet in controllers, services, repositories of Thymeleaf pagina's.

Voorbeeld: opslag voor een werknemer:

Slechte aanpak: alle intelligentie in de service class:

```
werknemer.setWedde(werknemer.getWedde().add(bedragOpslag));
```

De Werknemer class is in dit voorbeeld "dom": ze bevat enkel getters en setters

Goede aanpak: Werknemer bevat een method (die de service oproept):

```
public void opslag(BigDecimal bedrag) {
    if (bedrag.compareTo(BigDecimal.ZERO) <= 0) {
        throw new IllegalArgumentException();
    }
    wedde = wedde.add(bedrag);
}
```

Test de correcte werking van zo'n method met een unit test.

- iii. Definieer de opmaak van BigDecimal en Date private variabelen met @NumberFormat en @DateTimeFormat.
 - iv. Voeg validation annotations toe als de gebruiker objecten van de class wijzigt.

- b. Voeg repository classes toe.
 - i. Injecteer de `JdbcTemplate` in de constructor.
 - ii. Elke method stuurt één SQL statement naar de database.
 - iii. Maak niet meer methods dan echt nodig.
Voorbeeld: maak niet in *elke* repository een method `findAll`.
Doe dit enkel als je *alle* records moet tonen.
 - iv. Als je niet *alle* records uit de database moet tonen, gebruik je geen `findAll` !
Die is te traag. Je gebruik methods zoals `findByNaam(String naam)`,
`findByIds(Set<Long> ids)`, `findById(long id)`,
 - v. Als je enkel records *telt*, is een method met daarin een SQL statement
`select count(... de kortste en performantste oplossing.`
 - vi. Maak een DTO class als je het resultaat van een SQL select query
niet kan voorstellen met een domain class.
 - vii. Maak tests voor de repositories.
 - c. Voeg service classes toe.
 - i. Typ `@Service` en `@Transactional` bij de classes.
 - ii. Injecteert de nodige repositories in de constructor.
 - iii. Een service method is dé plaats om database handelingen in één transactie
te plaatsen. De database doet ofwel al deze handelingen (`commit`),
of doet (bij een exception) al deze handelingen ongedaan (`rollback`).
 - iv. Als je in een service method een record leest en daarna wijzigt,
lees en lock je het record met `select ... for update`, zodat andere gebruikers
niet tegelijk het record kunnen wijzigen. Je plaatst deze `select ... for update`
in een repository method (bvb. `findAndLockById(long id)`)
die je oproept in de service method.
 - d. Voeg de nodige form classes toe.
 - e. Voeg de nodige classes toe om data te onthouden in session.
 - i. Deze classes implementeren `Serializable`.
 - ii. Typ `@Component` en `@SessionScope` voor deze classes.
 - iii. Onthoud zoveel mogelijk id's van objecten in plaats van objecten.
 - f. Voeg de nodige controller class en/of methods toe.
 - i. Typ `@Controller` en `@RequestMapping` bij de class.
 - ii. Injecteer de nodige services en session beans in de constructor.
 - iii. Typ `@GetMapping` voor methods die GET requests verwerken.
 - iv. Typ `@PostMapping` voor methods die POST requests verwerken.
Pas de "redirect" techniek toe bij POST requests.
 - g. Voeg een Thymeleaf pagina toe.
 - h. Test de pagina in de browser.
5. Herhaal de stappen uit punt 4 voor de volgende pagina ...

51 HERHALINGSTAKEN



Gastenboek en Gastenboekbeheer



Star Trek

52 COLOFON

Domeinexpertisemanager:	Jean Smits
Moduleverantwoordelijke:	Hans Desmet
Medewerkers:	Hans Desmet
Versie:	7/3/2022
Nummer dotatielijst:	