--- TO DO ----

vrijdag 28 februari 2025 8:01

- !!! (Spring) AOP !!! --> [Mastering @Aspect Annotation & Spring AOP | Aspect-Oriented Programming in Spring]

- Appendix section: **Functional programming** in 30min
- Predicates nakijken…
- Mockito

00. ZELF GEVONDEN

donderdag 6 maart 2025 10:25

[JSP] Taglib creeëren voor localdate:

Nieuw bestand in WEB-INF/tags: localDate.tag

```
<%@ tag body-content="empty" pageEncoding="UTF-8"
trimDirectiveWhitespaces="true"%>

<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<%@ attribute name="date" required="true" type="java.time.LocalDate"%>
<%@ attribute name="pattern" required="false" type="java.lang.String"%>

<c:if test="${empty pattern}">
	<c:set var="pattern" value="dd/MM/yyyy" />
</c:if>

<fmt:parseDate value="${date}" pattern="yyyy-MM-dd" var="parsedDate"
type="date" />
<fmt:formatDate value="${parsedDate}" type="date" pattern="${pattern}" />
```

Dan in .jsp:

```
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>
```

En om te gebruiken op veld:

```
<tags:localDate date="${todo.targetDate}" />
```

HOW TO Overzicht

woensdag 12 maart 2025 14:10

JPA

1) **Een @Entity**

```
@Entity(name = "user_details")
public class User {
```

2) **Met @Id:**

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
Integer id;
```

3) **Een Interface die "extends JpaRepository<Entity,Type ID>**

```
public interface UserRepository extends JpaRepository<User, Integer> {
```

OVERZICHT ANNOTATIONS

vrijdag 14 februari 2025 9:17

1 - 4

@Configuration	Indicates that a class declares one or more @bean methods and may be processed by the Spring container to generate bean definitions
@ComponentScan	Define specific packages to scan for components. If specific packages are not defined, scanning wil occur from the package of the class that declares this annotation.
@Bean	Indicates that a method produces a bean to be managed by the Spring container
@Component	Indicates that an annotated class is a "component"
@Service	Specialization of @Component indicating that an annotated class has business logic
@Controller	Specialization of @Component indicating that an annotated class is a "Controller" (e.g. a web controller). Used to define controllers in your web applications and REST API.
@Repository	Specialization of @Component indicating that an annotated class is used to retrieve and/or manipulate data in a database.
@Primary	Indicates that a bean should be given preference when multiple candidates are qualified to autowire a single-valued dependency.
@Qualifier	Used on a field or parameter as a qualifier for candidated beans when autowiring.
@Lazy	Indiciates that a bean has to be lazily initialized. Absence of @Lazy annotation will lead to eager initialization.
@Scope (value = ConfigurableBeanFactory. SCOPE_PROTOTYPE)	Defines a bean to be a prototype - a new instance will be created every time you refer to the bean. Default scope is singleton - one instance per IOC Container.
@PostConstruct	Identifies the method that will be executed after dependency injection is done to perform any initialization.
@PreDestroy	Identifies the method that will receive the calback notification to signal that the instance is in the process of being removed by the container. Typically used to release resources that it has been holding.
@Named	Jakarta CDI annotation similar to Component
@Inject	Jakarta CDI annotation similar to Autowired
API	
@RequestBody	Als parameter van REST-API, om body te ontvangen
@PathVariable	Als parameter van REST-API, om parameter uit url te ontvangen

OVERZICHT CONCEPTS

vrijdag 14 februari 2025 9:39

1-4

Dependency Injection	Identify beans, their dependencies and wire them together (provides IOC - Inversion of Control)
Constructor Injection	Dependencies are set by creating the Bean using it Constructor
Setter Injection	Dependencies are set by calling setter methods on your beans
Field injection	No setter or constructor. Dependency is injected using reflection.
IOC Container	Spring IOC Context that manages Spring beans & their lifecycle
Bean Factory	Basic Spring IOC Container
Application Context	Advanced Spring IOC Container with enterprise-specific features - Easy to use in web applications with internationalization features and good integration with Spring AOP.
Spring Beans	Objects managed by Spring
Auto-wiring	Process of wiring in dependencies for a Spring Bean

OVERZICHT application.properties

donderdag 20 februari 2025 9:08

Server poort:
```
server.port=8081
```

Logging level
```
logging.level.org.springframework=info
```

MVC Url's
```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

Datum zetten:
```
spring.mvc.format.date=dd/MM/yyyy
```

DB URL instellen:
```
spring.datasource.url=jdbc:h2:mem:testdb
```

Er voor zorgen dat uitvoeren data.sql gebeurt na aanmaak tabellen:
```
spring.jpa.defer-datasource-initialization=true;
```

Show SQL:
```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

MySQL properties
```
spring.datasource.url=jdbc:mysql://localhost:3306/todos
spring.datasource.username=todos-user
spring.datasource.password=dummytodos
#automatisch aanmaken tabellen (indien ze nog niet bestaan):
spring.jpa.hibernate.ddl-auto=update
```

Security (spring-boot-starter-security)
```
spring.security.user.name=username
spring.security.user.password=test
```

Database instellingen voor AWS:
```
spring.datasource.url=jdbc:mysql://${RDS_HOSTNAME:localhost}:${RDS_PORT:3306}/
${RDS_DB_NAME:social-media-database}
spring.datasource.username=${RDS_USERNAME:social-media-user}
spring.datasource.password=${RDS_PASSWORD:dummypassword}
```

Deze environment variables zijn automatisch aangemaakt op de AWS server wanneer je database toevoegt aan je applicatie in Elastic Beanstalk:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html

Wat na de : komt, is de default value. Bvb ${RDS_HOSTNAME:localhost} -> default value is 'localhost'

DOCKER

donderdag 6 maart 2025 11:16

MySQL:

```
docker run --detach
--env MYSQL_ROOT_PASSWORD=dummypassword
--env MYSQL_USER=todos-user
--env MYSQL_PASSWORD=dummytodos
--env MYSQL_DATABASE=todos
--name mysql
--publish 3306:3306 mysql:8-oracle
```

OVERZICHT REACT

donderdag 20 maart 2025 11:21

Hooks

useParam()

createContext()

useState()

useNavigate()

useEffect()

Je kan zelf Hook creeëren voor (security-)Context:

```
export const useAuth = () => useContext(AuthContext);
```

Oproepen methodes vanop button

Oproepen van methode met parameter op button.
Gebruik van Arrow-function:

```
onClick={(id) => deleteTodo(todo.id)}
```

Anders wordt dit verschillende keren opgeroepen

Oproepen zonder parameter gaat wel simpelweg met:

```
onClick={deleteTodo}
```

De standaard parameter is dan het "event"

Nieuw component maken

1) ComponentNaam.jsx aanmaken met daarin
   export default function ComponentNaam() { return (…) }
2) Route path toevoegen:
   <Route path='/componentNaam/:eventueleVariabele' element = {<ComponentNaam>}/>
3) Naar navigeren:
   navigate(`todos/${eventueleVariabele}`)
   --> navigate is const van useNavigate()

2. Getting started

donderdag 19 december 2024 8:41

```
Var context = new
AnnotationConfigApplicationContext(HelloWorldConfiguration.class);
```
It can take **classes annotated with *@Configuration*, *@Component***, and JSR-330 metadata as input.

@Configuration
Indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

@Bean
A bean is an object that the Spring container instantiates, assembles, and manages.

context.getBean("nameOfBean") / context.getBean(classOfBean.class)

```
context.getBeanDefinitionNames()
Returns all beans
```

RESOLVING MATCHING BEANS
Matching beans = beans with same object class.
@Primary
Makes a bean primary.
If exists multiple beans with same object, this is the primary returned.

Use `@Qualifier("address3qualifier")`
`--> public Person person5Qualifier(String name, int age,`
`@Qualifier("address3qualifier") Address address) {`

Try with ressources (otherwise warning: Resource leak: 'context' is never closed**)**

```
try (var context = new
AnnotationConfigApplicationContext(HelloWorldConfiguration.class)) {
                    //if somethings goes wrong within this try
                    //context is automatically closed.
```

3. Using Spring Framework to Create and Manage Your Java Objects

donderdag 26 december 2024 13:23

`@Component`
Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.
---> Gebruik in combinatie met @Primary en/of @Qualifier("...")

`@ComponentScan("com.in28minutes.learn_spring_framework.game")`
Gecombineerd met @Configuration.
Zegt waar er moet gescand worden voor @Component classes.
Indien enkel @ComponentScan --> automatisch scan op huidig package

@Primary vs @Qualifier

```
@Component @Primary
class QuickSort implement SortingAlgorithm {}

@Component
class BubbleSort implement SortingAlgorithm {}

@Component @Qualifier("RadixSortQualifier")
class RadixSort implement SortingAlgorithm {}

@Component
class ComplexAlgorithm
    @Autowired
    private SortingAlgorithm algorithm;

@Component
class AnotherComplexAlgorithm
    @Autowired @Qualifier("RadixSortQualifier")
    private SortingAlgorithm iWantToUseRadixSortOnly;
```

@Primary - a bean should be given preference when multiple candidates are qualified
@Qualifier - a specific bean should be auto-wired (name of the bean can be used as qualifier)
ALWAYS think from the perspective of the class using the SortingAlgorithm:

1) Just @Autowired: give me (preferred) SortingAlgorithm
2) @Autowired + @Qualifier: I only want to use specific SortingAlgorithm - RadixSort

(REMEMBER) @Qualifier has higher priority then @Primary

4. Exploring Spring Framework Advanced Features

dinsdag 11 februari 2025 9:05

@Lazy

Kan gebruikt worden op elke class met @Component of @Bean.
Kan ook gebruikt worden op @Configuration. Dan is elke component/bean in die configuration Lazy.

```java
@Component
class ClassA {

}

@Component
@Lazy
class ClassB {
    private ClassA classA;

    public ClassB(ClassA classA) {
        //Logic
        System.out.println("Some init logic...");
        this.classA = classA;

    }

    public void doSomething() {
        System.out.println("Do Something...");
    }

}

@Configuration
@ComponentScan
public class LazyInitializationLauncherApplication {

    public static void main(String[] args) {
        try(var context = new AnnotationConfigApplicationContext(LazyInitializationLauncherApplication.class)) {

            System.out.println("Initialization of context is completed");
            context.getBean(ClassB.class).doSomething();
        }

    }

}
```

Zonder de @Lazy-tag, de system out "some init logic" zou sowieso uitgevoerd worden bij het initialiseren van de context.
Dus dan zou volgorde van sysouts zijn:

```
Some init logic...
Initialization of context is completed
Do Something...
```

Mét lazy gebeurt dat pas bij het gebruik van ClassB, dan is volgorde sysouts:

```
Initialization of context is completed
Some init logic...
Do Something…
```

@Scope

@Scope(value=ConfigurableBeanFactory.SCOPE_PROTOTYPE)

Zonder scope wordt steeds dezelfde instance van een component teruggestuurd bij context.getBean(…).
Met scope, bij elke referentie naar bean wordt er instantie gemaakt.

@PostConstruct

Is used on a method that needs to be called after dependency injection is done.

@PreDestroy

Is used on a methot as a callback notification to signal that the instance is in the rpocess of being removed by the containe r.
Typical used to release ressources it has been holding.

```java
@Component
class SomeClass {
    private SomeDependency sd;

    public SomeClass(SomeDependency sd) {
        this.sd = sd;
        System.out.println("All dependencies are ready");
    }

    @PostConstruct
    public void init() {
        sd.getReady();
    }

    @PreDestroy
    public void cleanUp() {
        System.out.println("Clean up...");
    }
}

@Component
class SomeDependency {
    public void getReady() {
        System.out.println("Getting ready...");
    }
}
```

XML Configuration
--> important to understand and know it exists

Voorbeeld configuration via xml:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean definitions here -->

    <bean id="name" class="java.lang.String">
        <constructor-arg value="Gert"/>
    </bean>

    <bean id="age" class="java.lang.Integer">
        <constructor-arg value="35"/>
    </bean>

<!--    <context:component-scan base-package="com.in28minutes.learn_spring_framework.game" /> -->

    <bean id="instancePacman"
        class="com.in28minutes.learn_spring_framework.game.PacmanGame" />

    <bean id="instanceGameRunner"
        class="com.in28minutes.learn_spring_framework.game.GameRunner">
        <constructor-arg ref="instancePacman"/>
    </bean>
</beans>
```

Te gebruiken door:

```java
try (var context = new ClassPathXmlApplicationContext("contextConfiguration.xml")) {
```

Spring Stereotype Annotations

@Component - Generic annotation applicable for any class

- Base for all Spring Stereotype Annotations
- Specializations of @Component:
  - **@Service** - Indicates that an annotated clss has business logic
  - **@Controller** - Indicates that an annotated class is a "Controller" (e.g. a web controller)
    Used to define controllers in your web applications and REST API
  - **@Repository** - Indicates that an annotated class is used to retrieve and/or manipulate data in a database

USE THE MOST SPECIFIC ANNOTATION POSSIBLE
Why?

- You give more info to the framework about your intentions.
- You can use AOP at a later point to add additional behaviour
  - Example: for @Repository, Spring automatically wires in JDBC Exception translation features

Theorie

dinsdag 11 februari 2025 9:21

Lazy VS Eager

Default initialization for Spring Beans is: **Eager.**
Eager init is recommended: Errors in the configuration are discovered immediately at application startup.
When using @Lazy, errors in configuration become runtime errors.

Comparing Lazy Initialization vs Eager Initialization

Heading	Lazy Initialization	Eager Initialization
Initialization time	Bean initialized when it is first made use of in the application	Bean initialized at startup of the application
Default	NOT Default	Default
Code Snippet	@Lazy OR @Lazy(value=true)	@Lazy(value=false) OR (Absence of @Lazy)
What happens if there are errors in initializing?	Errors will result in runtime exceptions	Errors will prevent application from starting up
Usage	Rarely used	Very frequently used
Memory Consumption	Less (until bean is initialized)	All beans are initialized at startup
Recommended Scenario	Beans very rarely used in your app	Most of your beans

Spring Bean Scopes

- **Singleton:** One object instance per Spring IoC container --> **DEFAULT**
- **Prototype:** Possibly many objects instances per Spring IoC container

Scopes ONLY for web:
- **Request**: One object instance per single HTTP request
- **Session**: One object instance per user HTTP Session
- **Application**: One object instance per web application runtime
- **Websocket**: one object instance per Websocket Instance

Java Singleton (GOF) vs Spring Singleton
- Spring singleton: one object instance per Spring IoC container
- Java singleton: One object instance per JVM

Prototype VS Singleton Bean Scope

Prototype vs Singleton Bean Scope

Heading	Prototype	Singleton
Instances	Possibly Many per Spring IOC Container	One per Spring IOC Container
Beans	New bean instance created every time the bean is referred to	Same bean instance reused
Default	NOT Default	Default
Code Snippet	@Scope(value= ConfigurableBeanFactory.SCOPE_PROTOTYPE)	@Scope(value= ConfigurableBeanFactory.SCOPE_SINGLETON) OR Default
Usage	Rarely used	Very frequently used
Recommended Scenario	Stateful beans	Stateless beans

Jakarta Contexts & Dependency Injection (CDI)
Before javax.*, now jakarta.*

CDI is a specificiation interface.
Spring Framework implements CDI

Important Inject API Annotations (Jakarta):
- Inject (= Autowired in Spring)

- Named (=Component in Spring)
- Qualifier
- Scope
- Singleton

Annotations vs XML Configuration

Let's Compare: Annotations vs XML Configuration

Heading	Annotations	XML Configuration
Ease of use	Very Easy (defined close to source - class, method and/or variable)	Cumbersome
Short and concise	Yes	No
Clean POJOs	No. POJOs are polluted with Spring Annotations	Yes. No change in Java code.
Easy to Maintain	Yes	No
Usage Frequency	Almost all recent projects	Rarely
Recommendation	Either of them is fine BUT be consistent	Do NOT mix both
Debugging difficulty	Hard	Medium

5. Spring Boot

vrijdag 14 februari 2025 10:12

https://github.com/in28minutes/master-spring-and-spring-boot/tree/main/02-springboot

Spring Initializr
https://start.spring.io/

Dependency toe te voegen:
REST-API: Spring Web

@RestController

@RequestMapping("courses")
Maakt een methode een REST-api.
In dit geval te benaderen via url/courses

Boot Starter Projects
Staat in pom.xml dependencies
Bvb spring-boot-starter-web, spring-boot-starter-test
Elk starter project bestaat uit een set van dependencies.
Bvb web bevat tomcat, MVC voor rest-api

Web application & REST API: Spring Boot Starter Web (spring-webvmc, spring-web, spring-boot-starter-tomcat, spring-boot-starter-json)
Unit Tests: Spring boot starter test
Talk to db using JPA: Spring Boot Starter Data JPA
Secure your web application or REST API: Spring Boot starter security

Autoconfiguration
Automated configuration for your app:
Decided based on:
- Which frameworks are in the Class path?
- What is the existing configuration (Annotations etc…)

Terug te vinden in Maven Dependencies:
Spring-boot-autoconfigure

Spring Boot DevTools
Zorgt er bvb voor dat server automatisch herstart bij wijziging code.

Spring Boot PRODUCTION-READY

PROFILES
Verschillende omgevingen (DEV, QA, PROD, …)
In application.properties.
Vb:
- application-dev.properties
- application-prod.properties

Application.properties blijft de master.
spring.profiles.active zet de huidige environment/profile

Voorbeeld:

application.properties

```
spring.application.name=learn-spring-boot
server.port=8081
#logging level
logging.level.org.springframework=debug
#profile
spring.profiles.active=dev
```

DEV:

application-dev.properties

```
logging.level.org.springframework=trace
```

PROD:

application-prod.properties

```
logging.level.org.springframework=info
```

Debug level zetten:
`logging.level.org.springframework=info`
In deze volgorde:
- trace
- debug
- info
- warning
- error
- off

(bvb info toont INFO + WARNING + ERROR)

`@ConfigurationProperties(prefix=" … ")`
f.e.: prefix = "currency-service"
En dan in application.properties:

```
currency-service.url=http://default.in28minutes.com
currency-service.username=defaultusername
currency-service.key=defaultkey
```

```
@ConfigurationProperties(prefix = "currency-service")
@Component
public class CurrencyServiceConfiguration {
    private String url;
    private String username;
    private String key;

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUsername() {
```

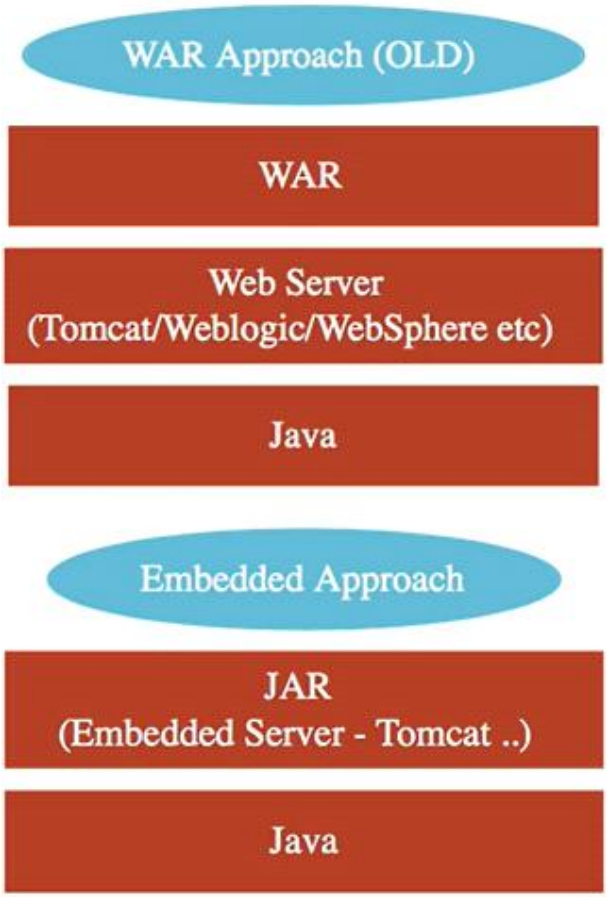
+getters & setters

Dan makkelijk te gebruiken in controllers:

```java
@RestController
public class CurrencyConfigurationController {

    @Autowired
    private CurrencyServiceConfiguration cs;

    @RequestMapping("currency-configuration")
    public CurrencyServiceConfiguration retrieveCurrencyConfig() {
        return cs;
    }

}

```

Embedded Servers

Old approach: WAR (aparte server)
NU: mvn clean install genereert een .JAR die je gewoon kan runnen…
Enige vereiste: java geïnstalleerd hebben
Default embedded server is Tomcat maar andere mogelijk (spring-boot-starter-jetty, spring-boot-starter-undertow)



Rechtsklik op project --> Run as --> Maven Build --> Goals: clean install

Creeërt JAR

```
[INFO] Installing C:\Users\gertv\OneDrive - Groupe Astek\Documents\Documentatie - Cursussen\Master Spring and Spring Boot with Java\learn-spring-boot\target\learn-spring-boot-0.0.1-SNAPSHOT.jar to C:\Users\gertv\.m2\repository\com\in28minutes\springboot\learn-spring-boot\0.0.1-SNAPSHOT\learn-spring-boot-0.0.1-SNAPSHOT.jar
```

In CMD:

```
Java -jar learn-spring-boot-0.0.1-SNAPSHOT.jar
```

Spring Boot Actuator
= Monitoring

Provides endpoints:
- Beans
- Health
- Metrics
- mappings

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

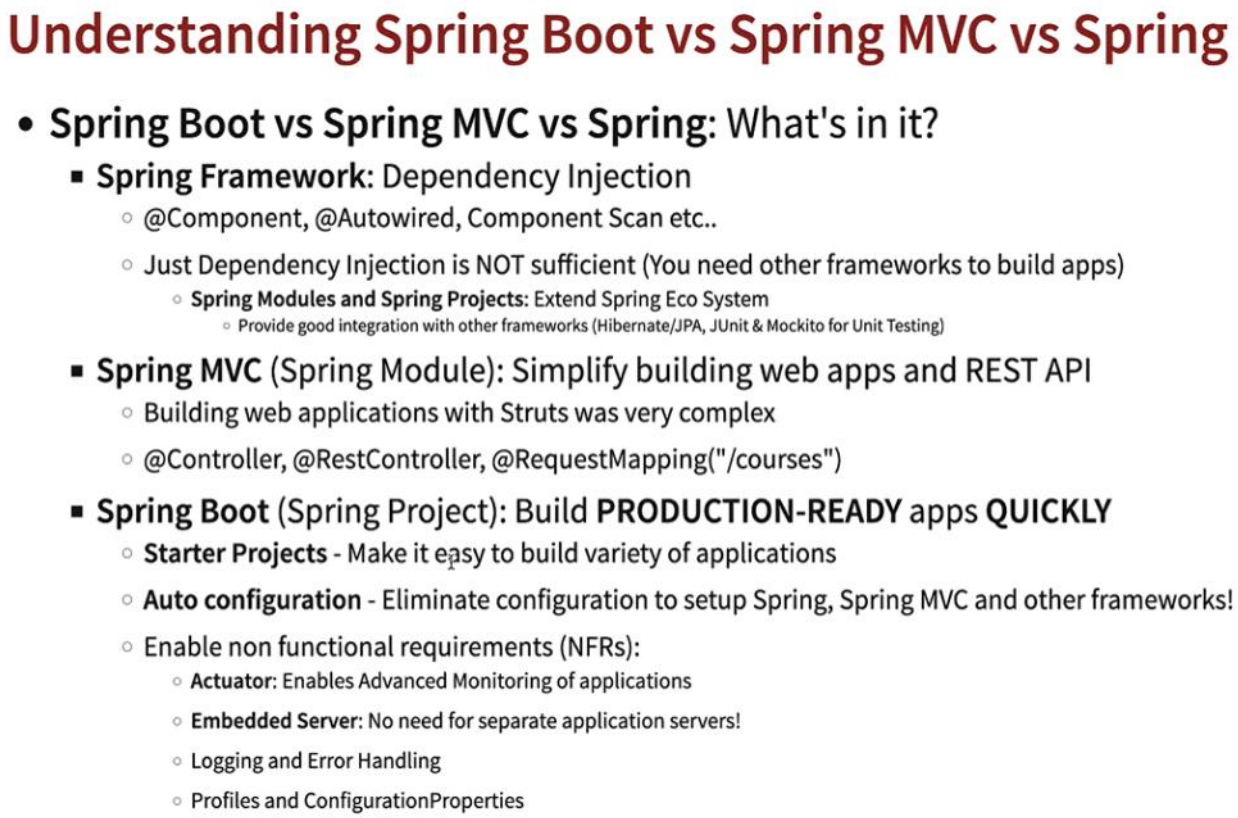
Dan via localhost:
http://http://localhost:8081**/actuator**
Default: enkel "health"

Extra endpoints toevoegen:
In application.properties:
`management.endpoints.web.exposure.include=*`
Soms staan zaken onzichtbaar (****), om te tonen:
(bvb in dit geval voor "env"):
`management.endpoint.env.show-values=ALWAYS`
(kan zijn: ALWAYS/NEVER/WHEN_AUTHORIZED)

http://localhost:8081/actuator/metrics
Geeft een lijst met metrics.
Om te openen voeg toe aan de url, bvb:
http://localhost:8081/actuator/metrics/http.server.requests

!! Hoe meer endpoints hoe zwaarder applicaties

Spring Boot vs Spring MVC vs Spring



6. JPA and Hibernate with Spring and Spring Boot

dinsdag 18 februari 2025 13:29

https://github.com/in28minutes/master-spring-and-spring-boot/tree/main/03-jpa-getting-started

Project met:
- Spring Web
- Spring Data JDBC
- Spring Data JPA
- H2 Database

Configuratie test-db

Bij het opstarten van project staat jdbc url in console:
`HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:511088f2-5d59-4faa-9f43-f79cb2a84514 user=SA`
---> dynamic URL

URL vastleggen (in application.properties):
`spring.datasource.url=jdbc:h2:mem:testdb`

Console activeren (in application.properties):
`spring.h2.console.enabled=true`
http://localhost:8081/h2-console

Create tables
Schema.sql in src/main/resources (zelfde niveau als application.properties)
Dit wordt automatisch uitgevoerd bij opstarten server

JDBC to Spring JDBC

Spring JDBC is minder java code dan JDBC

@Repository
Annotation om aan te tonen dat het gaat om een Class met JDBC acties

JdbcTemplate
Object type foor jdbc acties

JdbcTemplate.update()
Deze methode kan zowel insert,delete als update queries bevatten die dan uitgevoerd worden

Implements CommandLineRunner
Is een class die uitgevoerd wordt telkens de spring applicatie gerunned wordt.

```
8  @Repository
9  public class CourseJdbcRepository {
10
11     private static final String INSERT_QUERY = """
12                 insert into course (id,name,author) values (?,?,?)
13             """;
14     private JdbcTemplate springJdbcTemplate;
15
16     public CourseJdbcRepository(JdbcTemplate springJdbcTemplate) {
17         super();
18         this.springJdbcTemplate = springJdbcTemplate;
19     }
20
21     public void insert(Course course) {
22         springJdbcTemplate.update(INSERT_QUERY, course.getId(), course.getName(), course.getAuthor());
23     }
24
25 }
```

```java
@Component
public class CourseJdbcCommandLineRunner implements CommandLineRunner {

    @Autowired
    CourseJdbcRepository repos;

    @Override
    public void run(String... args) throws Exception {
        repos.insert(new Course(1, "learn Aws", "in28minutes"));
    }

}
```

Query'ing data using Spring JDBC

```java
return springJdbcTemplate.queryForObject(SELECT_QUERY, new BeanPropertyRowMapper<>
(Course.class), courseId);
```

--> queryForObject: via select query ophalen van 1 object
--> new BeanPropertyRowMapper<>(Course.class) : nodig voor definiëren om welk type object het gaat.

JPA and Entity mapping

Entity opbouwen:
@Entity: (jakarta.persistence:): evt met (name="...")
@Id
@Column: evt met (name="...")

@PersistenceContext
Dependency aanduiden van EntityManager

@Transactional
Nodig op repository-class.

EntityManager methodes:

- merge(object)
- find(objectType,primary key)
- remove(object)

```java
@Repository
@Transactional
public class CourseJpaRepository {

    @PersistenceContext
    private EntityManager entityManager;

    public void insert(Course course) {
        entityManager.merge(course);
    }

    public Course getCourseById(long id) {
        return entityManager.find(Course.class, id);
    }

    public void deleteById(long id) {
        Course course = entityManager.find(Course.class, id);
        entityManager.remove(course);
    }

}
```

Good to know: verschil merge() en persist():
Als je persist() doet, en binnen dezelfde transactie bvb een update op dat object.
Dan zal er een update-query uitgevoerd worden.

```
31      public void testPersist(Course course) {
32          entityManager.persist(course);
33          course.setAuthor("wazaaaaaaaaa");
34          // De setAuthor zal een update uitvoeren op DB
35      }
```

Show sql (in application.properties):
`spring.jpa.show-sql=true`

Spring Data JPA

Repository class wordt een interface die extends van JpaRepository:

```
public interface CourseSpringDataJpaRepository extends JpaRepository<Course, Long>
{
```

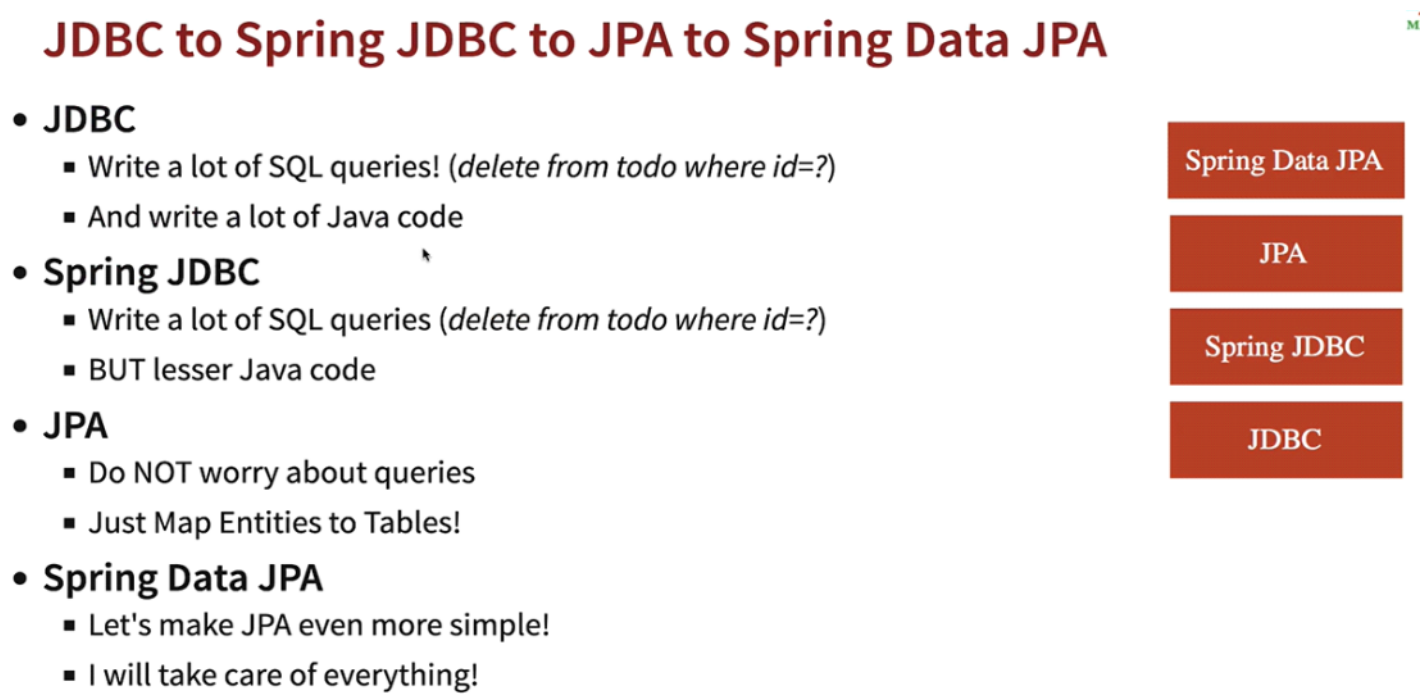
Methodes op repository:

- save
- deleteById
- findById
- count
- …

Makkelijke mapping!!
Bvb in repository toevoegen:

```
List<Course> findByAuthor(String author);
```

Volstaat om query uit te voeren die zoekt op author…



Hibernate vs JPA

JPA is de API.
Hibernate is de implementatie waarvan JPA gebruik maakt.
Maar er bestaan nog andere JPA implementaties ook.

Hibernate vs JPA

- **JPA** defines the specification. It is an API.
  - How do you define entities?
  - How do you map attributes?
  - Who manages the entities?
- Hibernate is one of the popular implementations of JPA
- Using Hibernate directly would result in a lock in to Hibernate
  - There are other JPA implementations (Toplink, for example)

7. Build webapplication with JSP,JPA etc..

woensdag 19 februari 2025 15:18

👉 **Bookmark the GitHub folder for this section**
https://github.com/in28minutes/master-spring-and-spring-boot/tree/main/11-web-application

👉 **Help for Debugging Problems**

- Here's the code backup at the end of step 05: https://github.com/in28minutes/master-spring-and-spring-boot/blob/main/11-web-application/Step05.md
- Step by Step changes are detailed here: https://github.com/in28minutes/master-spring-and-spring-boot/blob/main/11-web-application/99-step-by-step-changes.md

@ResponseBody
In combinatie met @RequestMapping("...").
Om aan te duiden dat een methode een body terug geeft.

```
@Controller
public class SayHelloController {

    @RequestMapping("say-hello")
    @ResponseBody // Wanneer je @RestController gebruikt, is @ResponseBody niet nodig
    public String sayHello() {
        return "Hello! What are you learning today?";
    }

}
```

JSP

Locatie:
Src/main/resources/META-INF/resources/WEB-INF/jsp/sayHello.jsp

Set URL's (in application.properties) (= view resolver gebruikt dit)

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

DEPENDENCY NODIG!

```
<dependency>
     <groupId>org.apache.tomcat.embed</groupId>
     <artifactId>tomcat-embed-jasper</artifactId>
     <scope>provider</scope>
</dependency>
```

```
    // location JSP: src/main/resources/META-INF/resources/WEB-INF/jsp/sayHello.jsp
    @RequestMapping("say-hello-jsp")
    public String sayHelloJsp() {
        return "sayHello";
    }

}
```

Capturing QueryParams

@RequestParam Als prefix van parameter in RequestMapping methode.
ModelMap: Type object als parameter methode om zaken door te geven aan de JSP.

Om param te gebruiken in jsp: **${**parameterNaam**}**

Voorbeeld:

```
    @RequestMapping("login")
    public String goToLogin(@RequestParam String name, ModelMap model) {
        model.put("name", name);
        System.out.println("GVV Test: " + name);
        return "login";
    }
```

Logging

Logging beperken tot bepaald package:
Logging.level.<package>=debug

```
logging.level.com.in28minutes.springboot.myfirstwebapp=debug
```

Logger class uit **org.slff4j**

```
private Logger logger = LoggerFactory.getLogger(LoginController.class);
```

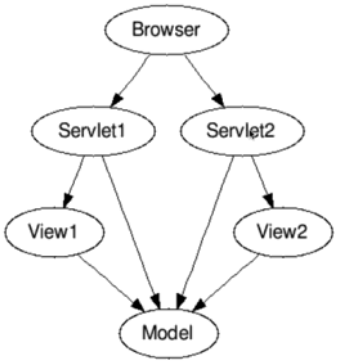
Gebruik:

```
logger.debug("Request param is {}",name); // (.info, .error, …)
```

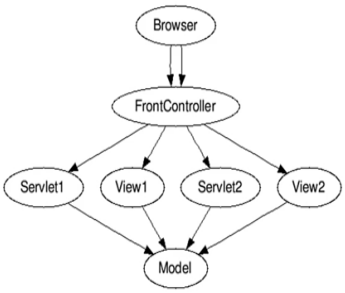
Included in spring-boot-starter-web

Understanding DispatcherServlet, Model 1, Model 2 and Front Controller

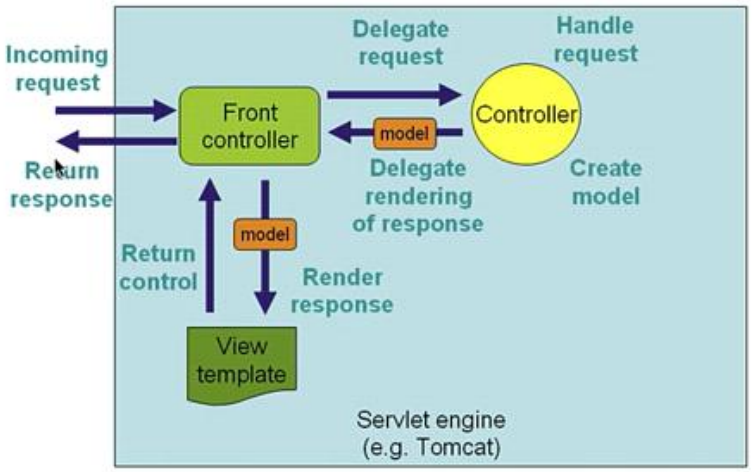
Model 1 architecture: all code in views (JSP,…) - complicated
Model 2 architecture: model (data to generate view) - view (show information to user) - controller (controls the flow)
--> simpler to maintain



Model 2 architecture - Front controller: 1 Frontcontroller for different servlets/views/…



Spring MVC Front Controller - Dispatcher Servlet



Steps (voorbeeld: surfen naar /login)

A: Receives HTTP Request

B1: Identifies correct controller method
==> /login => LoginController.gotoLoginPage

B2: Executes Controller method
=> Puts data into model
==> Returns view name => login

B3: Identifies correct View
=> uses application.properties prefix/suffix
==> /WEB-INF/jsp/login.jsp

B4: Executes View

C: Returns HTTP Response

GET/POST

Via parameter van @RequestMapping: **method = RequestMethod.GET** / RequestMethod.POST / …

```
12    @RequestMapping(value = "login", method = RequestMethod.GET)
13    public String goToLogin() {
14        return "login";
15    }
16
17    @RequestMapping(value = "login", method = RequestMethod.POST)
18    public String toToWelcomePage(@RequestParam String name, @RequestParam String password, ModelMap model) {
19        model.put("name", name);
20        model.put("password", password);
21        return "welcome";
22    }
```

Request vs Model vs Session

Request: variabelen meegestuurd via POST - steeds zichtbaar (dus niet secuur). Eens verstuurd worden attributen verwijderd van geheugen.
Model: variabelen in ModelMap gestoken - werkt enkel binnen zelfde pagina
Session: variabelen in ModelMap + gebruik van SessionAttribute. Gebruikt wel geheugen aangezien opgeslagen op server!

@SessionAttributes("…") of @SessionAttributes({ ".." , ".." })
Gebruiken als annotation op de controllers die deze session-variabele moeten kunnen gebruiken.
Komt overeen met waarde in de ModelMap.

```java
@Controller
@SessionAttributes({ "name", "password" })
public class LoginController {

//  @Autowired
    private AuthenticationService authServ;

    public LoginController(AuthenticationService authServ) {
        super();
        this.authServ = authServ;
    }

    @RequestMapping(value = "login", method = RequestMethod.GET)
    public String goToLogin() {
        return "login";
    }

    @RequestMapping(value = "login", method = RequestMethod.POST)
    public String toToWelcomePage(@RequestParam String name, @RequestParam String password, ModelMap model) {
        if (authServ.authenticate(name, password)) {
            model.put("name", name);
            model.put("password", password);
            return "welcome";
        } else {
            model.put("errorMessage", "Invalid name & password");
            return "login";
        }

    }

}
```

JSTL-Tags

Dependencies:

```
<dependency>
     <groupId>jakarta.servlet.jsp.jstl</groupId>
     <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
</dependency>
<dependency>
     <groupId>org.glassfish.web</groupId>
     <artifactId>jakarta.servlet.jsp.jstl</artifactId>
</dependency>
```

Toevoegen aan .jsp (vanboven):

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

Foreach

```
<c:forEach items="${todos}" var="todo">
```

Bootstrap

Dependencies (bootstrap + jquery)

```
<dependency>
<groupId>org.webjars</groupId>
     <artifactId>bootstrap</artifactId>
     <version>5.1.3</version>
</dependency>
<dependency>
     <groupId>org.webjars</groupId>
     <artifactId>jquery</artifactId>
     <version>3.6.0</version>
</dependency>
```

Te vinden onder Maven dependencies:

/META-INF/resources/webjars/bootstrap/5.1.3/css/bootstrap.min.css
/META-INF/resources/webjars/bootstrap/5.1.3/js/bootstrap.min.js

/META-INF/resources/webjars/jquery/3.6.0/jquery.min.js

Toevoegen aan pagina:
CSS - in head:

```
<link href="webjars/bootstrap/5.1.3/css/bootstrap.min.css" rel="stylesheet"/>
```

JS - vlak voor </body>:

```
<script src="webjars/bootstrap/5.1.3/js/bootstrap.min.js" type="text/javascript">
</script>
<script src="webjars/jquery/3.6.0/jquery.min.js" type="text/javascript"></script>
```

Redirect vanuit controller:

```
return "redirect:list-todos";
```

Let op! Gebruik url, niet de naam van de JSP.

Zorgt er in dit voorbeeld voor dat heel de methode listAllTodos uitgevoerd wordt (dus inclusief opvullen).
Moest je enkel doen return "listToDos" zou hij pagina tonen maar zonder inhoud.

```
17    public ToDoController(TodoService todoService) {
18        super();
19        this.todoService = todoService;
20    }
21
22    @RequestMapping("list-todos")
23    public String listAllTodos(ModelMap model) {
24        List<Todo> todos = todoService.findByUsername("in28minutes");
25        model.put("todos", todos);
26        return "listToDos";
27    }
28
29    @RequestMapping(value = "add-todo", method = RequestMethod.GET)
30    public String showNewToDoPage() {
31        return "todo";
32    }
33
34    @RequestMapping(value = "add-todo", method = RequestMethod.POST)
35    public String addNewToDo() {
36        return "redirect:list-todos";
37    }
38
39 }
```

Validations

Validations with Spring Boot

- 1: Spring Boot Starter Validation
  - pom.xml
- 2: Command Bean (Form Backing Object)
  - 2-way binding (todo.jsp & TodoController.java)
- 3: Add Validations to Bean
  - Todo.java
- 4: Display Validation Errors in the View
  - todo.jsp

2-way binding:
BEAN <--> FORM

Toevoegen Spring Boot Starter Validation

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

Spring Form Tag Library
https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/view.html

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

Binden van object aan form via "modelAttribute":
modelAttribute naam moet hetzelfde zijn als de waarde in het ModelMap
```
<form:form method="POST" modelAttribute="iptTodo">
```

In de controller moet bij het openen van de pagina (dus NIET in de add-methode).
Een object in de ModelMap gestoken worden.

```
@RequestMapping(value = "add-todo", method = RequestMethod.GET)
public String showNewToDoPage(ModelMap model) {
    String username = (String) model.get("name");
    Todo newTodo = new Todo(0, username, "", LocalDate.now().plusYears(1), false);
    model.put("todo", newTodo);
    return "todo";
}

@RequestMapping(value = "add-todo", method = RequestMethod.POST)
public String addNewToDo(ModelMap model, @Valid Todo newTodo, BindingResult bindingResult) {
    if (bindingResult.hasErrors()) {
        return "todo";
    }
    String username = (String) model.get("name");
    todoService.addToDo(username, newTodo.getDescription(), LocalDate.now().plusYears(1), false);
    return "redirect:list-todos";
}
```

In de JSP:

```
<h1>Enter details</h1>
<form:form method="POST" modelAttribute="todo">
    Description: <form:input type="text" path="description" required="true"/
    <!-- errors -->
    <form:errors path="description" cssClass="text-warning"/>
    <!-- hidden -->
```

Binden aan inputveld via "path" (= naam van het veld van het object)
```
<form:input type="text" path="description" required="true"/>
```

Bean-Validation

Via Annotations (Size, PastOrPresent, NotEmpty, …)
https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/constraints/package-summary

Onderdeel van starter-validation package.

Voorbeeld:
```
@Size(min = 10, message = "Add at least 10 characters")
private String description;
```

Ook nodig om @Valid annotation te gebruiken bij input parameter!
+ BindingResult
```
public String addNewToDo(ModelMap model, @Valid Todo newTodo, BindingResult
bindingResult) {
```

Tonen in JSP:
```
<form:errors path="description" cssClass="text-warning"/>
```

Niet vergeten: check op bindingResult.hasErrors() om op zelfde pagina te blijven:

```java
@RequestMapping(value = "add-todo", method = RequestMethod.POST)
public String addNewToDo(ModelMap model, @Valid Todo newTodo, BindingResult bindingResult) {
    if (bindingResult.hasErrors()) {
        return "todo";
    }
    String username = (String) model.get("name");
    todoService.addToDo(username, newTodo.getDescription(), LocalDate.now().plusYears(1), false);
    return "redirect:list-todos";
}
```

Implementing Delete

Service:

```java
public void deleteById(int id) {
    Predicate<? super Todo> predicate = todo -> todo.getId() == id;
    todos.removeIf(predicate);
}
```

Controller:

```java
@RequestMapping(value = "delete-todo")
public String deleteToDo(@RequestParam int id) {
    todoService.deleteById(id);
    return "redirect:list-todos";
}
```

JSP:

```
<a href="delete-todo?id=${todo.id}" class="btn btn-warning">Delete</a>
```

Implementing UPDATE

Grotendeels idem zoals delete.
In JSP verwijzen naar een update-todo link die verwijst naar controller methode voor het laden van het Todo object en **door te sturen naar todo.jsp:**

```java
@RequestMapping(value = "update-todo")
public String showUpdateToDoPage(@RequestParam int id, ModelMap model) {
    // todoService.deleteById(id);
    Optional<Todo> toDoToBeUpdated = todoService.findById(id);
    model.put("todo", toDoToBeUpdated.get());
    return "todo";
```

En uiteindelijke Actie:

```java
@RequestMapping(value = "update-todo", method = RequestMethod.POST)
public String updateToDo(ModelMap model, @Valid Todo updatedTodo, BindingResult bindingResult) {
    if (bindingResult.hasErrors()) {
        return "todo";
    }
    updatedTodo.setUsername((String) model.get("name"));
    todoService.updateTodo(updatedTodo);
    return "redirect:list-todos";
}
```

Bootstrap datepicker

https://bootstrap-datepicker.readthedocs.io/en/latest/
https://mvnrepository.com/artifact/org.webjars/bootstrap-datepicker

```xml
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap-datepicker</artifactId>
    <version>1.9.0</version>
</dependency>
```

```
<link
href="webjars/bootstrap-datepicker/1.9.0/css/bootstrap-
datepicker.standalone.min.css"
rel="stylesheet">
```

```
<script
src="webjars/bootstrap-datepicker/1.9.0/js/bootstrap-datepicker.min.js"></script>
```

```
<script type="text/javascript">
		$('#targetDate').datepicker({
			format : 'dd/mm/yyyy'
		});
</script>
```

Include common code (JPSF):

```
<%@ include file="common/navigation.jspf" %>
```

Spring Security

Dependency

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

In console zie je standaard ww (**standaard login is = user**)

```
Using generated security password: e8c06b04-5281-4b1b-936e-a3ca40833ad2
```

En er wordt automatisch loginpagina toegevoegd aan de applicatie.
Je geraakt op geen enkele URL zonder passeren van /login
standaard

Ook /logout is standaard geïmplementeerd.

Door toevoegen van SpringSecurityConfiguration.java (naam is niet belangrijk) met @Configuration annotation.
Met daarin een @Bean : InMemoryUserDetailsManager.
Wordt gebruiker en pswd toegevoegd:

```
@Configuration
public class SpringSecurityConfiguration {

    @Bean
    public InMemoryUserDetailsManager crateUserDetailsManager() {
        Function<String, String> pswdEncoder = input -> this.passwordEncoder().encode(input);
        UserDetails userDetails = User.builder().passwordEncoder(pswdEncoder).username("GVV").password("dummy")
                .roles("USER", "ADMIN").build();

        return new InMemoryUserDetailsManager(userDetails);
    }
```

InMemoryUserDetailsManager kan meerdere users als parameter hebben:

```
return new InMemoryUserDetailsManager(userDetails1, userDetails2);
```

Combinatie: @Configuration + @Bean InMemoryUserDetailsManager volstaat zodat spring deze config gebruikt voor security.

Idem voor encoding: @Bean + passwordEncoder;

Gebruik van SecurityContextHolder voor ophalen authentication gegevens:

```
SecurityContextHolder.getContext().getAuthentication().getName();
```

Spring Boot Starter Data JPA

Dependency:

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

Voor memory database H2
Note: scope is ingevuld zodat dit enkel in runtime bestaat.

```
<dependency>
	<groupId>com.h2database</groupId>
```

```
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Vaste URL instellen in application.properties:
`spring.datasource.url=jdbc:h2:mem:testdb`

Configuratie Spring Security

Default zijn alle Url's afgeschermd.
Een login formulier is getoond voor unauthorized requests.

Om te configureren wat afgeschermd is door loginformulier en wat niet:

SecurityFilterChain (interface)
Defines a filter chain which is capable of being matched against an HttpServletRequest. In order to decide whether it applies to that request.
Returns DefaultSecurityFilterChain.
--> Methode toevoegen aan SpringSecurityConfiguration.

When we override SecurityFilterChain, we need to define entire chain again!

(**HttpSecurity** httpSec)

1) Zeker zijn dat elke request authenticated is:
   `httpSec.authorizeHttpRequests(auth -> auth.anyRequest().authenticated());`
2) Form laten tonen
   `httpSec.formLogin(withDefaults());`

Expliciet voor h2o-database - disable x-frames
`httpSec.csrf(csrf -> csrf.disable());`
`httpSec.headers(headers -> headers.frameOptions(frameOptions -> frameOptions.disable()));`

```
40●     @Bean
41      public SecurityFilterChain filterChain(HttpSecurity httpSec) throws Exception {
42          httpSec.authorizeHttpRequests(auth -> auth.anyRequest().authenticated());
43          httpSec.formLogin(withDefaults());
44
45  //      Deprecated since 6.1 :
46  //      httpSec.csrf().disable();
47  //      httpSec.headers().frameOptions().disable();
48
49          // X-Frame-Optionas enabled ==> Frames cannot be used.
50          // h2-console uses frames ==> Disable X-Frame-Options header
51          httpSec.csrf(csrf -> csrf.disable());
52          httpSec.headers(headers -> headers.frameOptions(frameOptions -> frameOptions.disable()));
53
54          return httpSec.build();
55
56      }
```

Making Entity

@Entity annotation toevoegen aan Todo class.

Default data toevoegen:
Creeër data.sql in src/main/resources.
By default wordt dit uitgevoerd voor het aanmaken van de tabellen in h2o.
Om dit te vermijden toevoegen aan application properties:
`spring.jpa.defer-datasource-initialization=true;`

Creating Repository

= een interface met als naam <entityNaam>Repository

… extends JpaRepository<className,type ID>:

```
public interface TodoRepository extends JpaRepository<Todo, Integer> {
```

Spring weet automatisch door naamgeving de functionaliteit.
Bvb findBy<naamKolom>(..)

```
public List<Todo> findByUsername(String username);
```

JpaRepository actions

```
.save(…)
.deleteById(..)
```

Connecteren met MySQL:

(zie DOCKER voor opzetten MySQL database)

!! Zet eerst h2database dependency in pom.xml in commentaar.

Dependency:
https://mvnrepository.com/artifact/mysql/mysql-connector-java

```
<dependency>
     <groupId>mysql</groupId>
     <artifactId>mysql-connector-java</artifactId>
     <version>8.0.33</version>
</dependency>
```

Settings in application.properties:
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/todos
spring.datasource.username=todos-user
spring.datasource.password=dummytodos

Dit is deprecated in nieuwere versie:
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

Mysqlsh kan gebruikt worden voor commando's

mysqlsh commands
mysqlsh
\connect todos-user@localhost:3306
\sql
use todos
select * from todo;
\quit

8. REST API

donderdag 6 maart 2025 11:50

@RestController
Annotation makes controller a REST-controller.

@RequestMapping / @GetMapping
Methode definiëren als request.
`@RequestMapping(path = "hello-world", method = RequestMethod.GET)`
Of alternatief voor get:
`@GetMapping(path="hello-world-bean")`

Returning bean

Geeft automatisch JSON-response

```
@GetMapping(path = "hello-world-bean")
public HelloWorldBean helloWorldBean() {
    return new HelloWorldBean("Hello World");
}
```

What happens in the background?

(enable logging to see it in console)

How are requests handled?
All requests are being handled by **DispatcherServlet** (=Front controller Patern)
Doet mapping tussen url en juiste methode.
Autoconfiguration (**DispatcherServletAutoConfiguration**)

How does bean object get converted to JSON?
@ResponseBody
--> Gebruikt door @RestController

Auto configuration via: **JacksonHttpMessageConverters**

Who is configuring error mapping?
Auto configuration: **ErrorMvcAutoConfiguration**

How are all jars available?
Spring, Spring MVC, Jackson, Tomcat,…
--> via starter projects.

Path parameters

@Pathvariable:
Te gebruiken voor parameter in methode.

In het path komt dit overeen met {…}
Bvb @Pathvariable String name --> path="blabla/{name}"

```
@GetMapping(path = "/hello-world/path-variable/{name}")
public HelloWorldBean helloWorldPathVariable(@PathVariable String name) {
    return new HelloWorldBean(String.format("Hello World, %s", name));
}
```

Request Methods for REST API

- GET
- POST
- PUT (update an existing resource)
- PATCH (update part of a resource)
- DELETE

POST

@PostMaping(path="…")

@RequestBody kan gebruikt worden als parameter van methode.

```
@PostMapping(path="/users")
public void createUser(@RequestBody User user) {
    dao.save(user);
}
```

Response Status

404: resource not found

500: server exception
400: Bad request / validation error
200: success
201: created
204: no content
401: unauthorized
404: resource not found

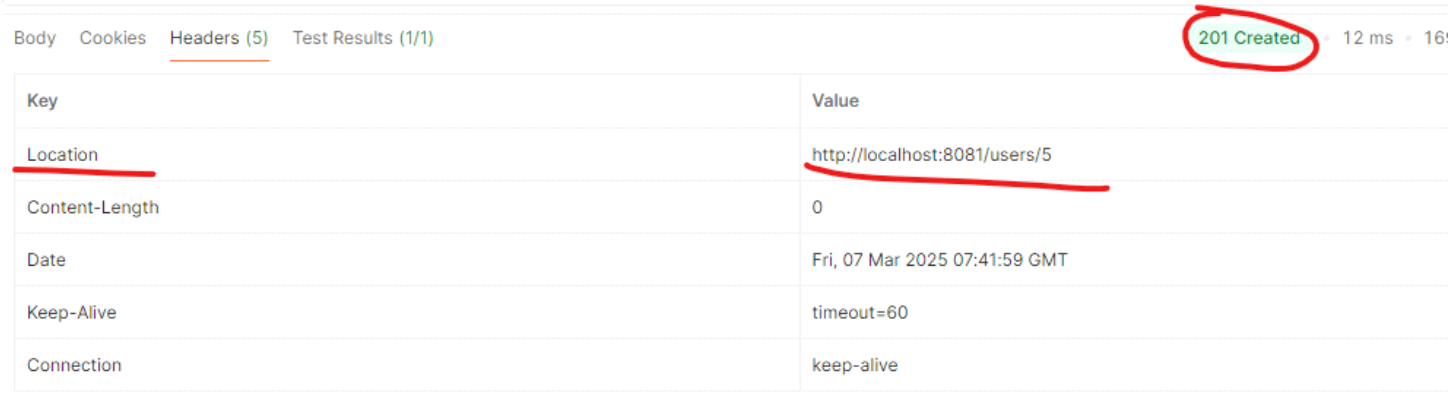
Return in method via ResponseEntity

```
return ResponseEntity.created(null).build();
```
---> method return value moet ResponseEntity<ObjectType> zijn
-----> .created returns "201"

In created(..) kan de naar te verwijzen url gezet worden.
Bvb na aanmaak user willen we naar /users/<id nieuwe user> om de gegevens van de nieuwe user te tonen.
Dit doen we door een URI te maken via ServletUriComponentsBuilder.

```java
@PostMapping(path = "/users")
public ResponseEntity<User> createUser(@RequestBody User user) {
    User newUser = dao.save(user);
    URI location = ServletUriComponentsBuilder.fromCurrentRequest() // get current url (/users)
            .path("/{id}") // add '/{id}' to the url
            .buildAndExpand(newUser.getId()) // replace {id} by the id of new user
            .toUri(); // convert to URI

    return ResponseEntity.created(location).build();
}
```

Resultaat na submit:



Assign response status to Exception-class
Voorbeeld, ik doe dit wanneer user niet gevonden:

```
throw new UserNotFoundException("id: " + id);
```

De Exception class moet dan de annotation **@ResponseStatus(code = HttpStatus.xxx)** bevatten:

```java
@ResponseStatus(code = HttpStatus.NOT_FOUND)
public class UserNotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public UserNotFoundException(String message) {
        super(message);
    }
}
```

Generic/Custom Exception handling

Nodig:
- Class om exception details bij te houden.
- Class die extends doet van ResponseEntityExceptionHandler.
- In die class nieuwe implementatie van de methode handleException (naam veranderen!)

@ControllerAdvice
Annotation op class.
Specialization of @Component for classes that declare f.e. ExceptionHandler methods to be shared accross multipe @Controller classes

@ExceptionHandler(Exception.class) --> annotation bovenop handleException methode. Wil zeggen: behandelt alle exceptions

```java
// Dit komt van de superclass ResponseEntityExceptionHandler
@ExceptionHandler(Exception.class)
public final ResponseEntity<ErrorDetails> handleAllExceptions(Exception ex, WebRequest request) throws Exception {
    ErrorDetails errorDetails = new ErrorDetails(LocalDateTime.now(), ex.getMessage(),
            request.getDescription(false));

    return new ResponseEntity<ErrorDetails>(errorDetails, HttpStatus.INTERNAL_SERVER_ERROR);
}
```

Zo kan voor elke soort exception een methode aangemaakt worden.
Door het veranderen van de parameter van @ExceptionHandler.
bvb:

```java
@ExceptionHandler(UserNotFoundException.class)
public final ResponseEntity<Object> handleUserNotFoundException…
```

Validations for REST API

--> additional dependency: **spring-boot-starter-validation**

@Valid annotation: toe te voegen voor parameter:

```java
public ResponseEntity<User> createUser(@Valid @RequestBody User user) {
```

In combinatie met validatie-tags op de object-attributen.
bvb: @Size(min = 2) / @Past (om te voorkomen dat datum in verleden ligt)
Evt met message:

```java
@Past(message = "Birthdate should be in the past")
```

--> standaard resultaat is een 400 (bad request) response.
----> handleMethodArgumentNotValid
Om dit custom op te vangen is het nodig om deze methode te overriden in je eigen "CustomizedResponseEntityExceptionHandler"

Ex.getFieldErrors : contains all errors and their messages.

```java
@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
        HttpHeaders headers, HttpStatusCode status, WebRequest request) {

    StringJoiner sj = new StringJoiner(" / ");
    for (FieldError fe : ex.getFieldErrors()) {
        sj.add(fe.getField() + ": " + fe.getDefaultMessage());
    }

    ErrorDetails errorDetails = new ErrorDetails(LocalDateTime.now(), sj.toString(), request.getDescription(false));

    return new ResponseEntity(errorDetails, HttpStatus.BAD_REQUEST);
}
```

REST API Documentation

Automatisch genereren documentatie:

- Open API
- Swagger

Open API is gebaseerd op Swagger.
OpenAPI = standaardisering van swagger.

Swagger UI: Visualize & interact with REST API

Springdoc-openapi
https://springdoc.org/

Examins the application on runtime to infer API semantics.

```
<dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
        <version>2.8.5</version>
</dependency>
```

Locatie UI:
http://server:port/context-path/swagger-ui.html
---> http://localhost:8081/swagger-ui.html

JSON DOC:
http://localhost:8081/v3/api-docs

Content Negotiation

Same Resource - Same URI - Different representations
Different content Type (XML or JSON), Different Language

Accept header (MIME types - application/xml, application/json, …)
Accept-Language header (en, nl, fr, …)

Voorbeeld: XML toevoegen aan REST-applicatie.

```
<dependency>
     <groupId>com.fasterxml.jackson.dataformat</groupId>
     <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

Daarna in Postman, in de request-header:
Key: Accept
Value: application/xml <--> application/json

Internationalization (-i18n)

HTTP Request Header: Accept-Language
(en, nl, fr, …)

Property files voor languages in: src/main/resources:

- messages.properties
- messages**_nl**.properties

In controller variabele van type **MessageSource** (injected in constructor van Controller)

```
private MessageSource messageSource;
```

Ophalen Locale:

```
Locale locale = LocaleContextHolder.getLocale();
```

Message ophalen:

```
messageSource.getMessage("good.morning.message", null, "Default Message", locale);
```

2de argument zijn parameters voor message in formaat:

```
Object[] {"param1","param2"}
```

In message:

```
Test.message = dit is een test {0} {1}
```

Versioning

Verschillende mogelijkheden

- Via URL
- Via Request parameter
- Via header
- Media Type

URL-Versioning

Bvb /v1/person

```
@GetMapping(path = "/v1/person")

@GetMapping(path = "/v2/person")
```

Request parameter versioning

Bvb: /person?version=2
Gebruik van "params".

```
@GetMapping(path = "/person", params = "version=1")
```

(Custom) headers versioning

Beide versies zelfde URL maar andere header met key X-API-VERSION.
Gebruik van "headers".

```
@GetMapping(path = "/person/header", headers = "X-API-VERSION=2")
```

Media type versioning

Ref. "content negiotation" or "accept header"
Zelfde URL, produces = application/vnd.company.app-v2+json
Gebruik van "produces"

```
@GetMapping(path = "/person/accept", produces = "application/vnd.company.app-v2
+json")
```

In header:
Key=accept
Value=
application/vnd.company.app-v2+json

Factors to consider:

- URI pollution
- Misuse of HTTP Headers
- Caching
- Can we execute request on browser?
- API documentation

- Summary: no perfect solution

Think about versioning even before you need it!

HATEOAS

Hypermedia As The Engine Of Application State

In het kort: met de response van je REST-call ook info (url) toevoegen om verdere acties te ondernemen.
Bvb na het creeëren van een user de url terugsturen met de lijst van alle users.

Enhancing REST API to tell consumers how to perform subsequent actions.

Standard implementation: HAL (JSON Hypertext Application Language).
--> Simple format that gives a consistent and easy way to hyperlink between resources in your API.

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

EntityModel:
A simple EntityModel wrapping a domain object and adding links to it.

```
EntityModel<User> em = EntityModel.of(foundUser);
```

WebMvcLinkBuilder
Builder to ease building Link instances pointing to Spring MVC controllers

```
WebMvcLinkBuilder link = linkTo(methodOn(this.getClass()).retrieveAllUsers());
```

Voorbeeld:

```
    @GetMapping(path = "/users/{id}")
    public EntityModel<User> retrieveUser(@PathVariable int id) {
        User foundUser = dao.findOne(id);

        if (foundUser == null) {
            throw new UserNotFoundException("id: " + id);
        }

        EntityModel<User> em = EntityModel.of(foundUser);
        // Link to specific method:
        WebMvcLinkBuilder link = linkTo(methodOn(this.getClass()).retrieveAllUsers());
        em.add(link.withRel("all-users"));

        return em;
    }
```

Resultaat:

```
{
    "id": 4,
    "name": "Kobe",
    "birthDate": "1981-03-11",
    "_links": {
        "all-users": {
            "href": "http://localhost:8081/users"
        }
    }
}
```

Customizing REST API Responses

@JsonProperty("new_name")
Andere naam geven aan veld in JSON-response

```
@JsonProperty("user_name")
String name;
```

Return only selected fields (=Filtering)

- **Static Filtering**: same filtering for a bean across diferent REST API
  @JsonIgnoreProperties, @JsonIgnore
- **Dynamic Filtering**: Customize filtering for a bean for specific REST API
  @JsonFilter with FilterProvider

Static Filtering
Gebeurt in de bean.
Boven een veld van je class **@JsonIgnore** toevoegen.
Dit veld zal dan niet in de JSON-response teruggestuurd worden.

Of boven de class: **@JsonIgnoreProperties("naamVeld")**

Dynamic Filtering
Gebeurt in de REST-controller.
Filtering die afhangt van de REST-API.
bvb voor de ene het veld wel terugsturen, voor een andere niet.

1) Nodig in de bean:

@JsonFilter
Moet bovenop de de object-class:

```
@JsonFilter("SomeBeanFilter")
public class SomeBean {
```

2) In de REST-controller-method:

SimpleBeanPropertyFilter
Definieert welke velden uit te filteren:
(filterOutAllExcept kan ook andere methoden zijn)

```
SimpleBeanPropertyFilter filter =
SimpleBeanPropertyFilter.filterOutAllExcept("field1", "field2");
```

FilterProvider
Linkt de filter aan de te filteren bean.

```
FilterProvider filters = new SimpleFilterProvider().addFilter("SomeBeanFilter",
filter);
```

MappingJacksonValue
Allows to add serialization logic.
Hangt de filtering aan het bean-object:

```
MappingJacksonValue mappingJacksonValue = new MappingJacksonValue(someBean);
```

```
@GetMapping(path = "/filtering-dynamic")
public MappingJacksonValue filteringDynamic() {
    SomeBean someBean = new SomeBean("value1", "value2", "value3");

    SimpleBeanPropertyFilter filter = SimpleBeanPropertyFilter.filterOutAllExcept("field1", "field2");
    FilterProvider filters = new SimpleFilterProvider().addFilter("SomeBeanFilter", filter);

    MappingJacksonValue mappingJacksonValue = new MappingJacksonValue(someBean);
    mappingJacksonValue.setFilters(filters);

    return mappingJacksonValue;
}
```

Spring Boot Actuator

Spring Boot Starter Actuator dependency.

Localhost:8081/actuator

Alle endpoints openzetten (in application.properties)

```
management.endpoints.web.exposure.include=*
```

Voorbeelden interessante endpoints:

- actuator/**health**
- actuator/**beans**
- actuator/**env**
- actuator/**metrics**
- actuator/**mappings**

HAL explorer

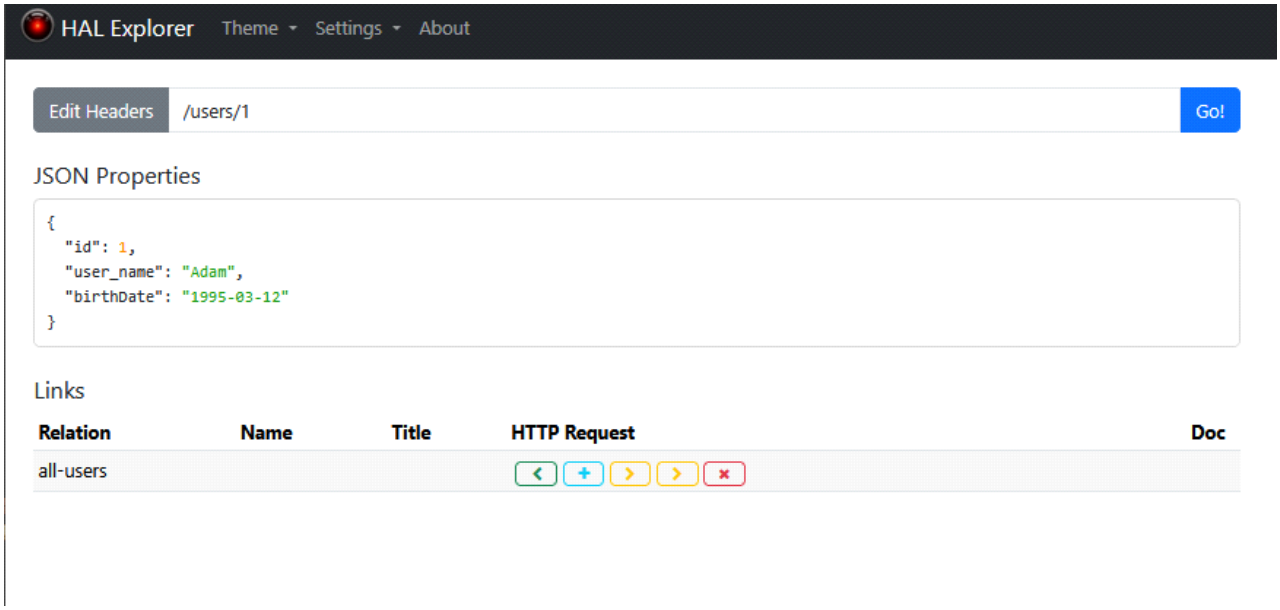
JSON Hypertext Application Language

Enable non-technical teams to play with API's.
Je kan navigeren door de API's

Spring Boot HAL Explorer

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-hal-explorer</artifactId>
</dependency>
```

URL:
http://localhost:8081/explorer



Implementing basic authentication

Met spring-boot-starter-security

Default user: user
Default pswd: zie console

Instellen via application.properties:

```
spring.security.user.name=username
spring.security.user.password=test
```

Enhancing Spring Security Configuration for basic authentication

When you send a request. Spring Security intercepts that request.
Filter chains:

1) All requests should be authenticated
2) If request not authenticated, a web page (login form) is shown
3) CSRF --> POST, PUT

TO DO:

- Voor REST: login form moet niet getoond worden
- Allow CSRF

Oplossing: Configuration bean aanmaken: SpringSecurityConfiguration.
Dit overrides de standaard security.

Deze bevat een methode filterChain:

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
```

Met daarin:
1. Er voor zorgen dat alle requests authenticated moeten zijn

```
http.authorizeHttpRequests(auth -> auth.anyRequest().authenticated());
```

Dit zal er voor zorgen dat er een 403 response teruggestuurd wordt.

2. Er voor zorgen dat er pop-up voor login getoond wordt indien nog niet geauthenticeerd
Import:

```
import static org.springframework.security.config.Customizer.withDefaults;
```

Deze withDefault gebruiken bij httpBasic:

```
http.httpBasic(withDefaults());
```

3. Disable CSRF

```
http.csrf(csrf -> csrf.disable());
```

9. Full Stack Application (REACT)

donderdag 13 maart 2025 10:15

ES: ECMASCRIPT
- EcmaScript is standard
- JavaScript is implementation

Npm: Package manager for Javascript. Similar to Maven/Gradle in Java.
Npx: Package executer: execute JS packages directly, withou installing.
Package.json: dependency definitions.

Opzeten Frontend project

npm -init
Genereert package.json

npm install jquery
(in dezelfde folder als waar je package.json staat)
--> Voegt dependency toe aan de package.json

npx create-react-app todo-app
Creeërt folder "todo-app"

npm-start
Start project (in dev mode)

Troubleshooting commands
npm uninstall -g create-react-app
npx clear-npx-cache

npm test: run unit tests

npm run build
Build a production deployable unit.
Creeërt 3 files.

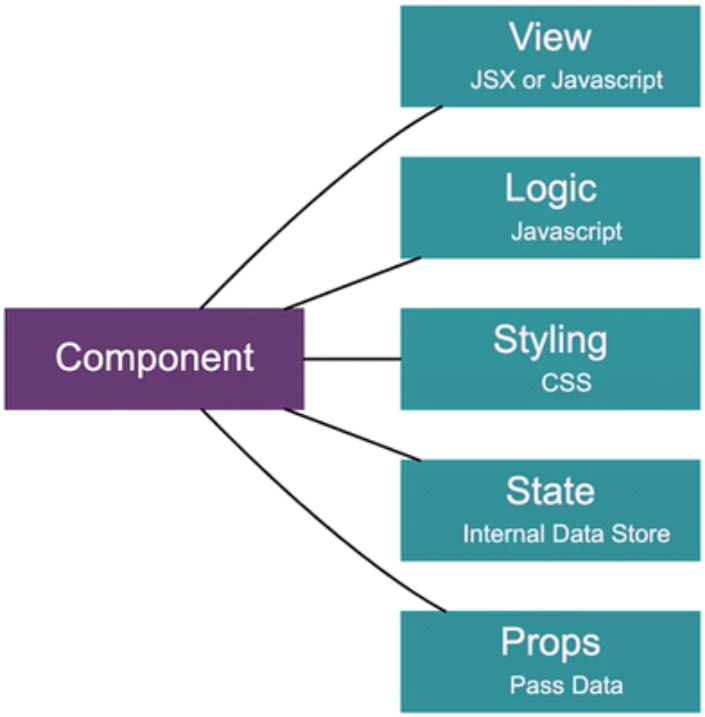
npm install --save react-router-dom
Add a dependency to your project.

Folder structure REACT

Package.json: dependencies
Node_modules: folder where dependencies downloaded to

- **README.md**: Documentation
- **package.json**: Define dependencies (similar to Maven `pom.xml`)
- **node_modules**: Folder where all the dependencies are downloaded to
- **React Initialization**
  - **public/index.html**: Contains root div
  - **src/index.js**: Initializes React App. Loads App component.
    - `src/index.css` - Styling for entire application
  - **src/App.js**: Code for App component
    - `src/App.css` - Styling for App component
    - `src/App.test.js` - Unit tests for App component
      - Unit test is right along side production code (Different to Java approach)
- **Remember**: Syntax might look little complex
  - Different from typical Java code (imports, ...)

REACT Components



Create Component

Function components.
Gebruik van "function"

Binnenin App.js:

```
function FirstComponent() {
  return(
    <div className='FirstComponent'>First Component</div>
  )
}
```

Om te gebruiken:
<FirstComponent/>

Class components
Class die Component extends.
Met daarin een render() method.

```
class ThirdComponent extends Component {
  render() {
    return(
      <div className='ThirdComponent'>Third Component</div>
    )
  }
}
```

In earlier versions of React, ONLY class components can have state.
Hooks were introduced in React 16.8.
useState hook allows to add state to function components.

Getting started with JSX - Views with React.

JSX is Stricter than HTML.
Closing tags verplicht.
JSX expects to have a parent tag. (bvb 2 aparte divs returnen kan niet, er moet 1 div boven staan).
Empty tags kunnen ook top-level zijn (<> </>)

Wat binnen de return of een function staat is JSX.

Different browsers have different support levels.
How to ensure backward compatibility for JS code?
--> Babel
Babel also converts JSX to JS.
https://babeljs.io/

JSX is at the end generated to JS

Tags in kleine letter: en niet <Div> of <DIV>
Custom components: starten met hoofdletters, bvb <FirstComponent>
--> zo is er duidelijk verschil tussen html tags en eigen componenten.

Gebruik className voor css (dus niet class="...")

Javascript Best Practices

Each component in its own file (or module)
Voorbeeld: FirstComponent.jsx
De module bevat een: **export default** function xxx() { ... }
In de module waar het gebruikt moet worden doe je import:

```
import FirstComponent from './components/learning-examples/FirstComponent';
```

```
export default function FirstComponent() {
    return(
      <div className='FirstComponent'>First Component</div>
    )
  }
```

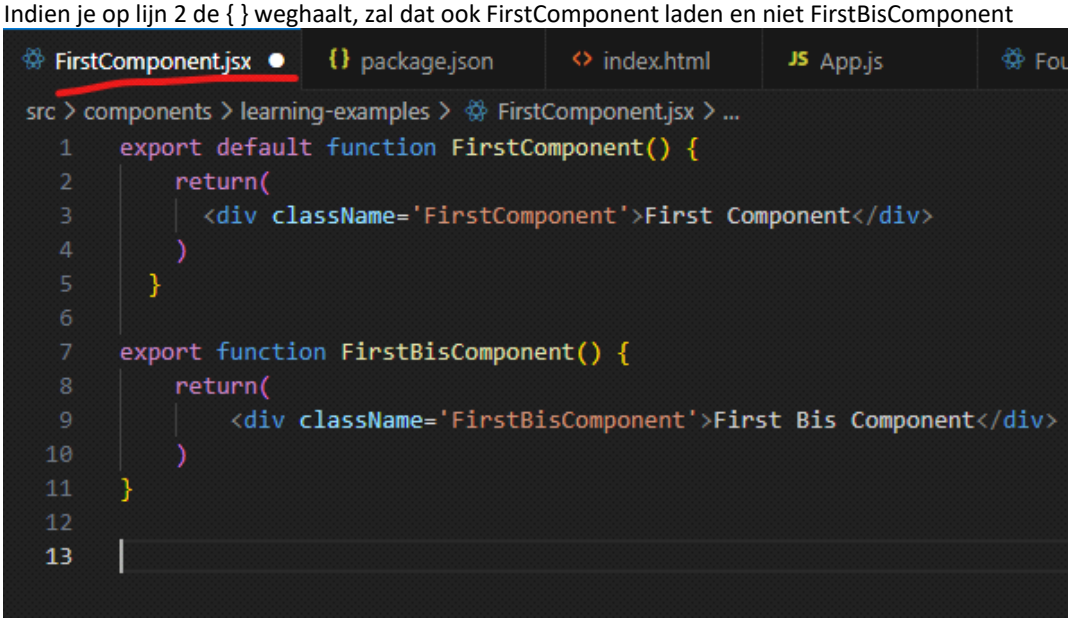
Only one default export per module.
Maar je kan meerdere exports hebben. Maar dan zonder "default" ervoor.

Je kan ook alle functions definiëren en dan apart:
export default <naamFunction> zetten

Indien je geen { } voor import gebruikt --> default component wordt gebruikt.

```
1   import FirstComponent from './components/learning-examples/FirstComponent';
2   import {FirstBisComponent} from './components/learning-examples/FirstComponent';
```

Indien je op lijn 2 de { } weghaalt, zal dat ook FirstComponent laden en niet FirstBisComponent



Exploring Javascript further

Object definiëren:
const person = { name: 'Gert' , address: {line 1: 'bla', line2: 'blabla'}}

Gebruik in JSX: **{person.address.line1}**

Array: **profiles: ['twitter','linkedin']**

Ook functions kunnen in object staan:
printProfile: () => { console.log(person.profiles[0]); }
Gebruik: {person.printProfile()}

```
const person = {
    name: 'Gert',
    address: {
        line1: 'Jeruzalemstraat 41',
        line2: '9420 Erpe-Mere'
    },
    profiles: ['twitter','linkedin','instagram'],
    printProfile: () => {
        person.profiles.map(
            (profile) => console.log(profile)
        )
    }
}
```

No need to use semicolon!

10. (REACT) Digging deeper into Components

vrijdag 14 maart 2025 7:47

State = Internal data store
Props / properties = pass data to a component

Calling a function

```
onClick={incrementCounter}
```

Geen '( )'!! Anders wordt de functie direct opgeroepen bij laden pagina en niet bij klik knop.

Define CSS

Optie 1: Via style: tussen {{ … }}

```
style={{borderRadius:"30px"}}
```

De css kan ook in een constante die dan gebruikt wordt:

```
const cssBtn = {borderRadius:"30px"};
style={cssBtn}
```

```
!!! CSS-in-JS is niet hetzelfde als gewoon CSS
```

```
const cssBtn = {
    borderRadius:"30px",
    fontSize:"15px",
    backgroundColor:"#00a5ab",
    width:"100px",
    margin:"10px",
    color:"white",
    padding:"15px",
};
```

Optie 2: Via aparte .css file en className

Aparte .css file met css class:

```
.counterButton{
    border-radius:30px;
    font-size:15px;
    background-color:#00a5ab;
    width:100px;
    margin:10px;
    color:white;
    padding:15px;
}
```

En dan gebruik in bvb button:

```
<button className="counterButton"
```

!! CSS moet geïmporteerd worden

```
import "./Counter.css"
```

React State

State: Built-in React object used to contain data or information about the component.

!! In earlier versions of React, ONLY Class Components can have state (and implementing was very complex).

Hooks introduced in React 16.8

- Easy to use
- useState hook allows adding state to Function Components.

useState returns 2 things:

- Current state
- A function to update state

Each instance of component has it's own state.

Voorbeeld van een Hook: **useState**
Import nodig:

```
import { useState } from "react";
```

```
const state = useState(0);
```

State bevat een array met 2 values: [0, f]

- De initiële value
- Een functie om de state te updaten.

DUS: gebruik maken van deconstruction.

Definiëren als een array bestaande uit een waarde en een functie:

```
const [count,setCount] = useState(0);
```

"count" kan dan gebruikt worden als variabele
setCount als methode:

```
setCount(count + 1);
```

Voorbeeld:

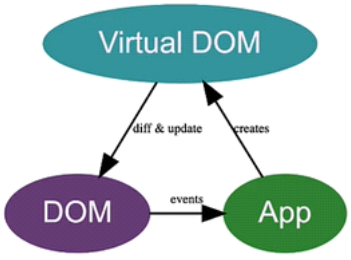
```
export default function Counter() {

    // Definieer Hook
    const [count,setCount] = useState(0);

    return(
        <div className="Counter">
            <span className="count">{count}</span>
            <div>
                <button className="counterButton" onClick={incrementCounter}>+1</button>
            </div>
        </div>
    )

    function incrementCounter(){
        setCount(count + 1);
        console.log('increment clicked',count);
    }
}
```

What happens in background?
We updated the state ==> REACT updated the view

- How can you update an HTML Element?
  - A HTML page is represented by DOM (Document Object Model)
  - Each element in a HTML page is a node in the DOM
  - You need to update the DOM to update the element
  - HOWEVER, writing code to update the DOM can be complex and slow!

- **REACT Takes a different approach:**
  - Virtual DOM: "virtual" represenetation of a UI (kept in memory)
  - React code updates Virtual DOM
  - React identifies changes and synchronizes them to the HTML page
    1) React creates Virtual v1 on load page
    2) You performan an action
    3) React creates Virtual Dom v2 as a result of your action
    4) React performs a diff between v1 and v2
    5) React synchronizes changes (updates HTML page)

We are NOT updating the DOM directly!
React identifies changes and efficiently updates the DOM.



React Props

You can pass "props" (short for properties) object to a React Component.
Used for things that remain a constant during lifetime of a component.

```
<PlayingWithProps property1="value1" property2="value2"/>
```

De function die het component voorstelt, ontvangt dan een array met 2 elementen: property1 en property2.

```
function PlayingWithProps({property1,property2}) {
```

Door de parameter te definiëren als {property1,property2} (deconstruction).
Om dan te gebruiken als variabelen property1 en property2.

Cijfers doorsturen, bvb:
propertyNaam={1}

Je kan ook **propTypes** toewijzen (Werkt niet???)

```
> export default function Counter({by}) {···
}

Counter.propTypes = {
    by: PropTypes.number
}

Counter.defaultProps = {
    by: 1
}
```

Moving React State Up

Voorbeeld:

- Counter: parent component
  - Geeft aan child function door via property.
    Bvb:

    <CounterButton by={1} someMethodInParent={someMethodInParent}/>

- CounterButton: child component
  - Neemt de function in de parent als parameter binnen:
    export function CounterButton({by, someMethodInParent}) {
  - Vervolgens kan die gebruikt worden binnen de functions van het child component:
    someMethodInParent();

Voorbeeld:

```
export default function Counter() {
    // Definieer Hook
    const [count,setCount] = useState(0);

    function incrementCounterParentFunction(by) {
        setCount(count + by);
    }

    function decrementCounterParentFunction(by) {
        setCount(count - by);
    }

    return(
        <>
            <span className="totalCount">{count}</span>
            <CounterButton by={1} incrementMethod={incrementCounterParentFunction} decrementMethod={decrementCounterParentFunction}/>
            <CounterButton by={2} incrementMethod={incrementCounterParentFunction} decrementMethod={decrementCounterParentFunction}/>
            <CounterButton by={5} incrementMethod={incrementCounterParentFunction} decrementMethod={decrementCounterParentFunction}/>
        </>
    )
}

export function CounterButton({by, incrementMethod, decrementMethod}) {

    // Definieer Hook
    const [count,setCount] = useState(0);

    function incrementCounter(){
        setCount(count + by);
        incrementMethod(by);
    }

    function decrementCounter(){
        setCount(count - by);
        decrementMethod(by);
    }

    return(
        <div className="Counter">
            <div>
                <span className="count">{count}</span>
                <button className="counterButton" onClick={incrementCounter}>+{by}</button>
                <button className="counterButton" onClick={decrementCounter}>-{by}</button>
            </div>
        </div>
    )
}
```

Het is ook mogelijk om vanuit het child-component direct een methode uit de paren-component op te roepen.
Via arrow-function, vb:

```
onClick={() => incrementMethod(by)}
```

Child-component:

```
export function CounterButton({by, incrementMethod, decrementMethod}) {
    return(
        <div className="Counter">
            <div>
                <button className="counterButton" onClick={() => incrementMethod(by)}>+{by}</button>
                <button className="counterButton" onClick={() => decrementMethod(by)}>-{by}</button>
            </div>
        </div>
    )
```

Parent:

```
export default function Counter() {
    // Definieer Hook
    const [count,setCount] = useState(0);

    function incrementCounterParentFunction(by) {
        setCount(count + by);
    }

    function decrementCounterParentFunction(by) {
        setCount(count - by);
    }

    function resetCounter() {
        setCount(0);
    }

    return(
        <>
            <span className="totalCount">{count}</span>
            <CounterButton by={1} incrementMethod={incrementCounterParentFunction} decrementMethod={decrementCounterParentFunction}/>
            <CounterButton by={2} incrementMethod={incrementCounterParentFunction} decrementMethod={decrementCounterParentFunction}/>
            <CounterButton by={5} incrementMethod={incrementCounterParentFunction} decrementMethod={decrementCounterParentFunction}/>
            <button className="counterButton" onClick={resetCounter}>Reset</button>
        </>
    )
}
```

11 Building Full Stack Todo (Spring+React) - FRONTEND

dinsdag 18 maart 2025 14:33

REACT

woensdag 19 maart 2025 8:38

Koppelen input aan userstate

Hook opzetten:

```
const [username,setUsername] = useState('in28minutes');
```

In het input-veld via value en onChange:

```
<input type="text" name="username" value={username} 
onChange={handleUserNameChange}/>
```

De onChange methode heeft een event als parameter.
De ingegeven waarde bevind zich in event.target.value:

```
    function handleUserNameChange(event) {
        setUsername(event.target.value);
    }
```

Vergelijken Strings: === (= strict equality)

```
if(username==='in28minutes' && pswd==='test'){
```

Tonen/verbergen via boolean + &&

Door combinatie boolean + && + stukje html code:

```
{showSuccessMessage && <div className="succesMessage">Authentication 
Successfully.</div>}
```

Router DOM

npm install react-router-dom

Gebruik van BrowserRouter, Route en Routes.

```
import { BrowserRouter, Route, Routes } from 'react-router-dom';
```

Elke Route heeft een path (het url-pad) en een element (component te tonen)

```
<BrowserRouter>
    <HeaderComponent/>
    <Routes>
        <Route path='/' element={<LoginComponent/>}/>
        <Route path='/login' element={<LoginComponent/>}/>
        <Route path='/welcome/:username' element={<WelcomeComponent/>}/>
        <Route path='/todos' element={<ListTodosComponent/>}/>
        <Route path='/logout' element={<LogoutComponent/>}/>

        <Route path='*' element={<ErrorComponent/>}/>
    </Routes>
    <FooterComponent/>
</BrowserRouter>
```

!!! Alles wat <Link wil gebruiken moet binnen BrowserRouter staan.
Dus zoals in dit voorbeeld HeaderComponent en FooterComponent staan binnen BrowserRouter-tags.

--> Je kan ook een basis url zetten: <BrowserRouter basename="/testReact"

useNavigate hook (ook import van react-router-dom)

```
const navigate = useNavigate();
```

Kan gebruikt worden om naar Route te navigeren:

```
navigate('/welcome');
```

Alle andere paths afdekken (door bvb error component te tonen):
path = '*'

Params meegeven met URL.
via hook:
import 'useParams' van react-router-dom

In de Route path via url/:variabele
```
<Route path='welcome/:username'
```

In het Component gebruik maken van useParams hook:
```
const params = useParams();
```

De variabele van in het path staat dan in deze constante:
```
{params.username}
```

Of via deconstruction:
```
const {username} = useParams();
```

Gebruik variabele in navigate.
Gebruik van (back)ticks:
```
navigate(`/welcome/${username}`);
```
--> template literal --> bouwen van dynamische strings in javascript/jsx

Navigeren naar andere pagina:
<a href=… : niet zo goed want dan wordt hele pagina herlade
```
<a href="/todos">Manage your Todo's </a>
```

Beter met: <Link to=…
---> is een import van react-router-dom:
```
<Link to="/todos">Manage your Todo's</Link>
```

Tabel afdrukken
Gebruik van map op array met gegevens:

```
<tbody>
    {
        todos.map(
            todo => (
                <tr key={todo.id}>
                    <td>{todo.id}</td>
                    <td>{todo.description}</td>
                </tr>
            )
        )

    }
</tbody>
```

Datum

```
const today = new Date();

const targetDate = new Date(today.getFullYear()+
12,today.getMonth(),today.getDay());
```

Tonen in jsx:

```
{todo.targetDate.toDateString()}
```

React Bootstrap

npm install bootstrap
--> installeert enkel de standaard bootstrap CSS & JS-bestanden
---> er bestaat ook react-bootstrap bootstrap, dan kan je alles gebruiken als react-componenten (https://react-bootstrap.netlify.app/)

Import in index.js:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Sharing components with multiple components

Contexte

Creëer een aparte file: AuthContext.js.

Creeër een Hook voor de context met **createContext()** (import van "react")
Deze moet export zijn zodat ze gebruikt kan worden in andere componenten.

```
export const AuthContext = createContext();
```

Hierin wordt een hook gedefinieerd (useAuth)
Gebruik maken van AuthContext.Provider die de children omvat.
En die AuthContext.Provider bevat value = , dit kan zowel variabelen als methodes bevatten.

```jsx
import { createContext, useState, useContext } from "react";

//1: Create a Context
export const AuthContext = createContext();

//Create Hook
export const useAuth = () => useContext(AuthContext);


//Put some state in the context
//Share the crated context with other components

export default function AuthProvider({ children }) {

    const [number, setNumber] = useState(10);

    return (
        <AuthContext.Provider value={ {number} }>
            {children}
        </AuthContext.Provider>
    )
}
```

<AuthContext> gebruiken als component rondom alles in de app.

```jsx
<AuthProvider>
    <BrowserRouter>
        <HeaderComponent/>
        <Routes>
            <Route path='/' element={<LoginComponent/>}/>
            <Route path='/login' element={<LoginComponent/>}/>
            <Route path='/welcome/:username' element={<WelcomeComponent/>}/>
            <Route path='/todos' element={<ListTodosComponent/>}/>
            <Route path='/logout' element={<LogoutComponent/>}/>

            <Route path='*' element={<ErrorComponent/>}/>
        </Routes>
        <FooterComponent/>
    </BrowserRouter>
</AuthProvider>
```

--> de hook die we gecreeërd hebben in AuthProvider (useAuth) kan gebruikt worden in alle children

```jsx
import { Link } from 'react-router-dom';
import { useAuth } from './security/AuthContext';

export default function HeaderComponent() {

    const AuthContext = useAuth();
    console.log(AuthContext.number);

    return(
        <header className="border-bottom border-light bor
```

Show/Hide Link

```
{isAuthenticated  && <Link className="nav-link" to="/todos">Todos</Link>}
```

Protecting Secure React Routes using Authenticated Route

Een function aanmaken die rond het route element komt.
Die returned enkel de "children" indien authenticated.
Indien niet authenticated : redirect naar de root "/" (= loginpagina)

```
function AuthenticatedRoute({children}) {
    const authContext = useAuth();
    if(authContext.isAuthenticated){
        return (
            <>{children}</>
        )
    }else {
        return <Navigate to="/"/>
    }
}

export default function TodoApp() {
    return (
        <div className="TodoApp">
            <AuthProvider>
                <BrowserRouter>
                    <HeaderComponent/>
                    <Routes>
                        <Route path='/' element={<LoginComponent/>}/>
                        <Route path='/login' element={<LoginComponent/>}/>

                        <Route path='/welcome/:username' element={
                            <AuthenticatedRoute>
                                <WelcomeComponent/>
                            </AuthenticatedRoute>
                        }/>

                        <Route path='/todos' element={<ListTodosComponent/>}/>
                        <Route path='/logout' element={<LogoutComponent/>}/>
```

<Navigate to="..." is een tag van de react-router-dom library voor het redirecten.

12 Connecting Spring boot with REACT

vrijdag 21 maart 2025 13:20

[REACT] Call REST-API from React

Via Axios
npm install axios

Import:
```
import axios from 'axios';
```

```
axios.get('http://localhost:8081/hello-world')
```

Gebruik van promises
--> callback methods van axios.get:

- Then : succesvolle respons
- Catch : error respons
- Finally : altijd opgeroepen (voor bvb clean-up)

```
axios.get('http://localhost:8081/hello-world')
    .then((response) => successfulResponse(response))
    .catch((error) => errorResponse(error))
    .finally(() => console.log('cleanup'));
```

Via axios.create kan je o.a. de baseURL instellen:

```
const apiClient = axios.create(
    {
        baseURL: 'http://localhost:8081'
    }
);
```

Idealiter wordt de .get in in een aparte service gestoken (bvb HelloWorldApiService.js):

HelloWorldApiService.js | AuthContext.js | TodoApp.jsx | package.json | WelcomeComponent.jsx | HeaderComponent.jsx

src > components > todo > api > HelloWorldApiService.js > ...

```
1  import axios from "axios";
2
3  const apiClient = axios.create(
4      {
5          baseURL: 'http://localhost:8081'
6      }
7  );
8
9  export const retrieveHelloWorldBean = () => apiClient.get('/hello-world-bean');
10
11 export const retrieveHelloWorldPathVariable = (username) => apiClient.get(`/hello-world/path-variable/${username}`);
12
13
```

Je kan ook makkelijk parameter meesturen: ${naamParameter}

```
export const retrieveHelloWorldPathVariable = (username) =>
apiClient.get(`/hello-world/path-variable/${username}`);
```

[SPRING] Enabeling CORS

Cross Origin Requests
By default: denied

Aan te passen in Spring-project.

WebMvcConfigurer --> addCorsMappings : dit moet override worden
In de root application (@SpringBootApplication):

@Bean toevoegen dat **WebMvcConfigurer** retourneert met daarin methode addCorsMappings die CorsRegistry object als parameter neemt.

Daarop addMapping doen met allowedMethods en allowedOrigins (url van consumer)

```
@Bean
public WebMvcConfigurer corsConfigurer() {
    return new WebMvcConfigurer() {
        public void addCorsMappings(CorsRegistry registry) {
            registry.addMapping("/**").allowedMethods("*").allowedOrigins("http://localhost:3000");
        }
    };

}
```

[REACT] Consume & handle API

1) useState constante maken:
   `const [todos,setTodo] = useState([]);`

2) useEffect - hook
   Wordt uitgevoerd indien component klaar.
   Eerste parameter is de uit te voeren functie, de 2de zijn de dependencies (waarop hook moet reageren). Indien dit enkel bij de rendering van component is is die parameter []
   `useEffect (() => refreshTodos());`

3) Function voor oproep van API

```
function refreshTodos() {
    retrieveAllTodosForUsername('in28minutes')
    .then(
        (response) => setTodos(response.data)
    )
    .catch((error) => console.log(error));
}
```

[SPRING] ResponseEntity

Bvb voor delete kan je ResponseEntity<Void> terugsturen .noContent()

```
34    @DeleteMapping("/users/{username}/todos/{id}")
35    public ResponseEntity<Void> deleteTodo(@PathVariable String username, @PathVariable int id) {
36        todoService.deleteById(id);
37        return ResponseEntity.noContent().build();
38    }
```

Je kan ook gewoons status 200 (void) terugsturen.

[REACT] Update formulier maken

ComponentNaam.jsx aanmaken met daarin
export default function ComponentNaam() { return (…) }
Met daarin ook het gebruik van de eventuele parameter: const {eventueleParameter} = useParams();

Route path toevoegen:
<Route path='/componentNaam/:eventueleParameter' element = {<ComponentNaam>}/>

Naar navigeren:
navigate(`todos/${eventueleParameter}`)
--> navigate is const van useNavigate()

FORMIK & MOMENT

Npm install formik
Npm install moment

```
import { Formik } from "formik";
```

Formik formulier wordt opgebouwd als een arrow-functie met als parameter (props).
Daarin een <Form> (van formik library!).
Form kan als properties hebben:

- initialValues
- enableReinitialize={true} is nodig zodat het formulier geupdate wordt na update van de useStates.
- onSubmit={naamFunction} : op te roepen function bij het klikken op knop van type="submit" in het form
  De function in de onSubmit krijgt als parameter de values (json formaat)

Per formulierveld is er dan een <fieldset> dat een <label> en <Field> bevat:

```
<Formik initialValues={ { description, targetDate }}
    enableReinitialize={true}
    onSubmit = {onSubmit}>
    {
        (props) => (
            <Form>
                <fieldset className="form-group">
                    <label>Description</label>
                    <Field type="text" className="form-control" name="description"></Field>
                </fieldset>
                <fieldset className="form-group">
                    <label>Target Date</label>
                    <Field type="date" className="form-control" name="targetDate"></Field>
                </fieldset>
                <div>
                    <button className="btn btn-success" type="submit">Save</button>
                </div>
            </Form>
        )
    }

</Formik>
```

[REACT] Validation

Toevoegen aan de Formik-tag:
<Formik …. validate={validatieMethode} ../>

Deze validatieMethode ontvangt de values van de velden.
En returned een "errors" object dat bevat bvb: { naamVeld: 'foutboodschap…'}.
Indien errors is leeg wordt de submit uitgevoerd. Indien error niet leeg wordt enkel de validatiemethode uitgevoerd.
Standaard wordt validatie uitgevoerd bij elke actie, dit kan uitgezet worden door:

- validateOnChange = {false}
- validateOnBlur = {false}

```
<Formik initialValues={ { description, targetDate }}
    enableReinitialize={true}
    onSubmit = {onSubmit}
    validate={onValidate}
    validateOnChange={false}
    validateOnBlur={false}>
```

```
function onValidate(values) {
    let errors = {};
    if(values.description.length < 5) {
        errors.description = 'Enter at least 5 characters';
    }
    return errors;
}
```

Tonen errors
Via Formik-tag: <ErrorMessage/>
Met als attributen:

- name (=de naam van het veld),
- Component
- className

```
(props) => (
    <Form>
        <ErrorMessage
            name="description"
            component="div"
            className="alert alert-warning"
        />
```

[REACT] PUT/POST

PUT: .put(<url>,<body>)
Naast de URL wordt ook de body (het te creeëren/updaten object) meegestuurd.

```
export const updateTodoApi = (username, idTodo, updatedTodo) =>
```

```
apiClient.put(`/users/${username}/todos/${idTodo}`,updatedTodo);
```

POST: .post(<url>,body)

```
export const createTodoApi = (username, newTodo) => apiClient.post(`/users/
${username}/todos`, newTodo);
```

Voor create in REACT: oproepen update-pagina met als parameter id= -1
```
navigate(`/todos/-1`);
```

Submit formulier:

```
function onSubmit(values) {
    const updatedTodo = {
        id: id,
        username: authContext.username,
        description: values.description,
        targetDate: values.targetDate,
        done: false
    };

    console.log('test update',updatedTodo);

    if(id == -1) {
        createTodoApi(authContext.username,updatedTodo)
        .then((response) => { nav('/todos') })
        .catch((error) => console.log('ERROR!',error));
    }else{
        updateTodoApi(authContext.username,id,updatedTodo)
        .then((response) => { nav('/todos') })
        .catch((error) => console.log('ERROR!',error));
    }
}
```

[SPRING] Authentication

Spring-boot-security-starter
Standaard paswoord in console.
Default username: user

@Configuration class maken
Met daarin methode (@Bean) van return-type SecurityFilterChain:
```
public SecurityFilterChain filterChain(HttpSecurity http) {
```

Als je daarin zet:
```
return http.build();
```
Dan wordt security eigenlijk disabled.

Dus moet er vanalles toegepast worden op die http build:

- Filter chain: naam methode
- Authenticate all requests: anyRequest().authenticated()
- Allow OPTIONS requests (requestMatchers(HttpMethod.OPTIONS)
  (anders fout : 'response to preflight request doesn't pass acesss control check)
- Basic authentication (.httpBasisc(Customizer.withDefaults())
- Disabling csrf (http.csrf(csrf -> csrf.disable())
- Stateless rest api (.sessionManagement (session ->
  session.sesssionCreationPolicy(SessionCreationPolicy.STATELESS)

```java
@Configuration
public class BasicAuthenticationSecurityConfiguration {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        // Authenticate all requests
        // + requestMatchers (in older Spring: antMatchers)
        // --> If an OPTIONS requests comes in: to an of the url's
        http.authorizeHttpRequests(
                auth ->
                    auth
                    .requestMatchers(HttpMethod.OPTIONS, "/**").permitAll()
                    .anyRequest().authenticated());
        // Shows pop-up for basic authentication
        http.httpBasic(Customizer.withDefaults());
        // Stateless
        http.sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
        // Disable csrf
        http.csrf(csrf -> csrf.disable());

        return http.build();
    }

}
```

Je hebt ook een basicAuthCheck() - REST-methode nodig:
(mag in om het even welke controller staan, zolang beschikbaar via REST)

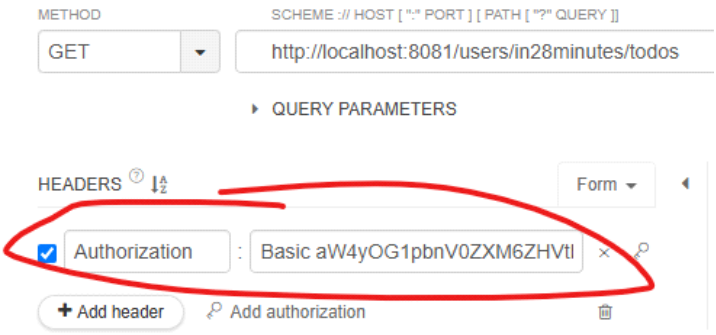
```java
    @GetMapping(path = "/basicauth")
    public String basicAuthCheck() {
        return "Success";
    }
```

[REACT] Authentication

Header met Authorization wordt meegestuurd als parameter van bvb .get.
Formaat:

```
{ headers: {Authorization: 'Basic aW4yOG1pbnV0ZXM6ZHVtbXk='} }
```

(de 'Basic ….' String is de encoded username & pswd, dit kan teruggevonden worden via o.a. Talend API tester)



To do in React-applicatie (in AuthContext / Authprovider)

1 - Service voorzien voor oproep van /basicauth met authorization-token.

```
export const executeBasicAuthenticationService = (token) =>
apiClient.get(`/basicauth`,{ headers: {Authorization: token} });
```

2 - Opbouw token dmv window.btoa - Hoofdletter gevoelig! Spaties belangrijk!

```
const baToken = 'Basic ' + window.btoa(username + ":" + password);
```

3 - oproep vn basicauth-api met deze token

4 - indien success-response: bijhouden token in een hook

```
async function login(username, pswd) {
    // Construct Token
    const baToken = 'Basic ' + window.btoa(username + ":" + pswd);

    try {
        // Call basicauth-api with token
        const response = await executeBasicAuthenticationService(baToken);

        if(response.status == 200){
            setAuthenticated(true);
            setUsername(username);
            setToken(baToken);
            return true;
        }else{
```

5 - deze token exposen zodat hij kan gebruikt worden voor alle api-calls.

--> zie volgende stappen voor async/await en axios interceptor

[REACT] Async/Await

De uitvoer van een API-call is een promise.
In de methode wordt niet gewacht tot de response alvorens verder te gaan met de rest van de logica.

Oplossing: async function en gebruik van await + response.status checken

Functions die async function oproepen moeten ook als async function gedefinieerd worden.

```
const response = await executeBasicAuthenticationService(baToken);
```

```
async function login(username, pswd) {
    // Construct Token
    const baToken = 'Basic ' + window.btoa(username + ":" + pswd);

    try {
        // Call basicauth-api with token
        const response = await executeBasicAuthenticationService(baToken);

        if(response.status == 200){
            setAuthenticated(true);
            setUsername(username);
            setToken(baToken);
            return true;
        }else{
            logout();
            return false;
        }
    } catch(error) {
            logout();
            return false;
    }

}

function logout() {
    setAuthenticated(false);
    setUsername(null);
    setToken(null);
}
```

Function die dit oproept wordt ook async en moet ook await bevatten bij oproep van de async function:

```
async function handleSubmit() {
    if(await authContext.login(username,pswd))
        navigate(`/welcome/${username}`);
    }else{
        setShowErrorMessage(true);
    }
}
```

[REACT] Axios Interceptor

Client maken met daarin de axios.create

ApiClient.js

```
export const apiClient = axios.create (
    {
        baseURL: 'http://localhost:8081'
    }
)
```

Deze importen in de API-services.

Bij login-methode een interceptor plaatsen op apiClient.
Deze interceptor zorgt er voor dat voor elke API-call, de authorization-header toegevoegd wordt.

```
apiClient.interceptors.request.use(
   (config) => {
                console.log('intercepting and adding a token');
                config.headers.Authorization=baToken;
                return config;
            }
)
```

Dus expliciet doorgeven van de token is enkel nodig voor basicauth api.

Getting Started with JWT
[SPRING]

Json Web Token
Open, industry standard for representing claims securely between two parites.
Can contain User Details and Authorizations.

JWT Contains:

- Header
  - Type: JWT
  - Hashing algorithm: HS512
- Payload
  - Standard attributes
    - Iss: the issuer
    - Sub: the subject
    - Aud: the audience
    - Exp: when does it expire?
    - Iat: when was token issued?
  - Custom attributes
- Signature
  - Includes a secret

==> wordt in detail uitgelegd in security-gedeelte

[REACT]
Aparte AuthetnicationApiService.js

Oproep van spring /authentication API.
Neemt als body {username,password}
POST! method

```
export const executeJwtAuthenticationService = (username,password) =>
apiClient.post(`/authenticate`,{ username, password });
```

Deze geeft token terug in respons.
De string 'Bearer ' + token = de token die gebruikt kan worden voor alle api-aanroepen.

```
async function login(username, pswd) {

    try {
        // Call basicauth-api with token
        const response = await executeJwtAuthenticationService(username,pswd);

        if(response.status == 200){
            const jwtToken = 'Bearer ' + response.data.token;
            setAuthenticated(true);
            setUsername(username);
            setToken(jwtToken);
            apiClient.interceptors.request.use(
                (config) => {
                    console.log('intercepting and adding a token');
                    config.headers.Authorization=jwtToken;
                    return config;
                }
```

13 Connecting full stack to JPA

woensdag 2 april 2025 14:03

1 - Dependencies in POM toevoegen:
- Spring-boot-starter-data-jpa
- (eventueel) com.h2database

2 - @Entity en @Id toevoegen aan Todo Class

3 - @Repository maken (=interface)
Dit extends JpaRepository<naamVanDeEntity,typeVanDeId>

```
public interface TodoRepository extends JpaRepository<Todo, Long>{
```

4 - Resource file (= @RestController) aanpassen zodat hij de Repository gebruikt.

Repository:

```
public interface TodoRepository extends JpaRepository<Todo, Integer> {

    List<Todo> findByUsername(String username);

}
```

Resource:

```
@RestController
public class TodoJpaResource {

    private TodoRepository todoRepos;

    public TodoJpaResource(TodoRepository todoRepos) {
        this.todoRepos = todoRepos;
    }

    @GetMapping("users/test")
    public String test() {
        return "test";
    }

    @GetMapping("/users/{username}/todos")
    public List<Todo> retrieveTodos(@PathVariable String username) {

        return todoRepos.findByUsername(username);

    }

    @GetMapping("/users/{username}/todos/{id}")
    public Todo retrieveTodo(@PathVariable String username, @PathVariable int id) {

        Optional<Todo> optTodo = todoRepos.findById(Integer.valueOf(id));

        return optTodo.isPresent() ? optTodo.get() : null;

    }

    @DeleteMapping("/users/{username}/todos/{id}")
    public ResponseEntity<Void> deleteTodo(@PathVariable String username, @PathVariable int id) {
        todoRepos.deleteById(id);
        return ResponseEntity.noContent().build();
    }

    @PutMapping("/users/{username}/todos/{id}")
    public Todo updateTodo(@PathVariable String username, @PathVariable int id, @RequestBody Todo iptTodo) {

        return todoRepos.save(iptTodo);
    }

    @PostMapping("/users/{username}/todos")
    public Todo createTodo(@PathVariable String username, @RequestBody Todo iptTodo) {
        iptTodo.setUsername(username);
        iptTodo.setId(null); // When ID null, .save know it's a create-action
```

```java
    @PostMapping("/users/{username}/todos")
    public Todo createTodo(@PathVariable String username, @RequestBody Todo iptTodo) {
        iptTodo.setUsername(username);
        iptTodo.setId(null); // When ID null, .save know it's a create-action

        return todoRepos.save(iptTodo);
    }

}
```

14. Junit

donderdag 3 april 2025 14:37

Junit 5

Testing: Check app behavior against expected behavior

System Testing or Integration Testing
Deploy the complete application and test.

Unit testing
Test specific units of application code independently.

Nieuwe test-file maken
Locatie: In source folder apart van src/

New --> Junit Test Case (New Junit Jupiter test)

Naamgeving: <naamClass>Test

+ add to java build path

Assert methods
Assertions-class

- assertEquals(expected,actual, <optionalMessage>)
- assertTrue(boolean, <optionalMessage>)
- assertFalse(boolean, <optionalMessage>)
- assert(Not)Null
- assertArrayEquals (expected,actual,<optionalMessage>)
  Geeft terug op welke index eerste verschil is.
- …

Junit Annotations

Elke test heeft @Test annotation.
Junit garandeert NIET dat tests in volgorde worden uitgevoerd zoals ze in de test-class staan.

Andere:

- @BeforeEach : voor elke @Test uitgevoerd
- @AfterEach: na elke @Test uitgevoerd
- @BeforeAll en @AfterAll
  Must be static! method

15. Mockito

donderdag 3 april 2025 15:43

Mockito = Unit testing framework

Standaard bij generatie spring-boot-project wordt dependency toegevoegd:
spring-boot-starter-test

Deze zorgt voor folder:
src/test/java

In effective pom kan je zien staan:

```
<mockito.version>5.14.2</mockito.version>
```

Stub-class: an example class.

Voorbeeld:
SomeBusinessImpl verwacht dat een DataService object injected wordt.

```java
public class SomeBusinessImpl {

    private DataService dataService;

    public SomeBusinessImpl(DataService dataService) {
        super();
        this.dataService = dataService;
    }

    public int findTheGreatestFromAllData() {
        int[] data = dataService.retrieveAllData();
        int greatestValue = Integer.MIN_VALUE;
        for (int value : data) {
            if (value > greatestValue) {
                greatestValue = value;
            }
        }

        return greatestValue;
    }
}

interface DataService {
    int[] retrieveAllData();
}
```

In de Test kan er dan gebruik gemaakt worden van een stub-class die deze DataService-interface implementeert en opvult:

```java
    @Test
    void testFindGreatestBasicScenario() {
        DataServiceStub stub = new DataServiceStub();
        SomeBusinessImpl businessImpl = new SomeBusinessImpl(stub);

        int result = businessImpl.findTheGreatestFromAllData();

        assertEquals(25, result);
    }

}

class DataServiceStub implements DataService {
    @Override
    public int[] retrieveAllData() {
        return new int[] { 25, 15, 5 };
    }
}
```

NADEEL: elke keer nieuwe methodes toegevoegd worden aan de DataService-interface, moet de Test-class aangepast worden.
+ Moeilijk voor veel tests, dan zou je meerdere stub-classes moeten aanmaken.

OPLOSSING: Mocks

Mocks met Mockito

mock(<className>.class)

```java
DataService dataServiceMock = mock(DataService.class);
```

when(<mock>.methode()).thenReturn(…)

```java
when(dataServiceMock.retrieveAllData()).thenReturn(new int[] {25,15,5});
```

Mockito Annotations

@Mock op de service-interface en @InjectMocks op de class die de service gebruikt.
Hierdoor is het ni tmeer nodig van mock() te gebruiken en constructor te gebruiken.
!!! Vereist @ExtendWith(MockitoExtension.class) op de Test-class (vroeger in Junit 4: @RunWith(MockitoJUnitRunner.class))

```java
@ExtendWith(MockitoExtension.class)
class SomeBusinessImplMockTest {

    @Mock
    private DataService dataServiceMock;
    @InjectMocks
    private SomeBusinessImpl businessImpl;

    @Test
    void testFindGreatestBasicScenario() {
//      Replaced by @Mock & @InjectMocks
//      DataService dataServiceMock = mock(DataService.class);
//      SomeBusinessImpl businessImpl = new SomeBusinessImpl(dataServiceMock);

        when(dataServiceMock.retrieveAllData()).thenReturn(new int[] { 25, 15, 5 });

        int result = businessImpl.findTheGreatestFromAllData();

        assertEquals(25, result);
    }
```

Multiple returns
.thenReturn(…).thenReturn(…)
Bij de eerste aanroep zal hij de eerste thenReturn terugsturen, bij de 2de de tweede.
Alles daarna zal hij steeds de laatste thenReturn terugsturen.

Mockito.any…
.anyInt; .anyString; .anyBoolean; …

16. Spring Security (TO DO - STUK OVERGESLAAN)

vrijdag 4 april 2025 8:21

1) **Trust nothing**
   Validate every request
2) **Assign least privileges**
   clear picture of users, roles and accesses.
   Assign minimum possible privileges at all levels
3) **Have complete mediation (1 way in)**
   Apply a well implemented security filter. Test the role and access of each user.
4) **Have defense in depth.**
   Multiple levels of security.
5) **Have economy of mechanism**
   Security architecture should be simple
6) **Ensur openness of design.**
   Easier to identify and fix security flaws.

Getting started with Spring Security

- Spring security intercepts all requests

Spring security executes a series of filters.
Filters provide these features:

- **Authentication**: is it a valid user (ex: basicAuthenticationFilter)
- **Authorization**: does the user have right access? (ex: AuthorizationFilter)
- Other Features:
  - **Cross-Origin Resource Sharing (CORS)** - CorsFilter
    - Should you allow AJAX calls from other domains?
  - **Cross Site Request Forgery (CSRF) -** CsrfFilter
    - A malicious website making use of previous authentication on your website
    - Default: CSRF protection enabled for update requests - POST, PUT, …
  - **Default Login Page, Logout Page**
    - LogoutFilter, DefaultLoginPageGeneratingFilter, DefaultLogoutPageGeneratingFilter
  - **Translating exceptions into proper Http Responses (ExceptionTranslationFilter)**
- **Order of filters is important**
  1) Basic check filters: CORS, CSRF, …
  2) Authentication filters
  3) Authorization filters

21. Getting started with Cloud and AWS

donderdag 24 april 2025 15:02

Amazon Web Services
Competitors: Microsoft Azure and Google Cloud

Root User: user we created our AWS account with -> used to login to console
Always create an IAM user, don't use the root user for services.

IAM Group --> IAM User

- Zoek op IAM in AWS console
- Attach permission: AdministratorAccess
- New user: in28minutes_dev - k!VZ2b%2
- Add to group

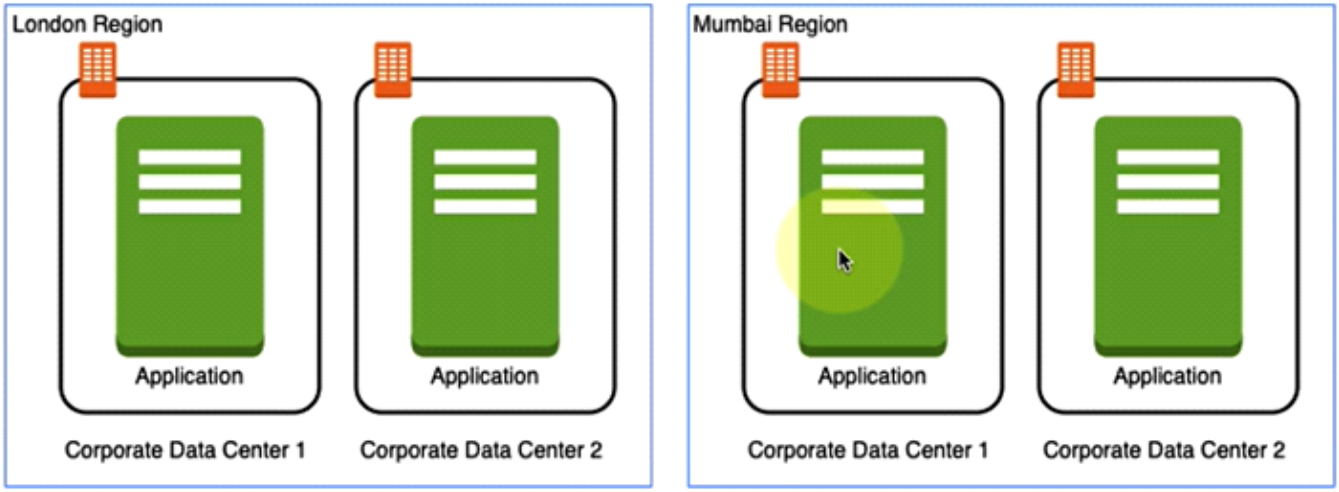
IAM gebruikt andere url voor login! (staat in console) --> bevat de account ID
https://653719116742.signin.aws.amazon.com/console

Regions and Zones

Challenges:

- High latency: slow access for users from other parts of the world
- Low availability: What if data center crashes? Application goes down:

Solution: multiple Regions with each multiple datacenters



AWS provides 25+ regions

Advantages Regions:

- High availability
- Low latency
- Global footprint
- Adhere to government regulations

Advantages Zones:

- Each AWS region consists of multiple AZ's
- Each Availability Zone:
    - Can have 1+ discrete data centers
    - Has independent & redudant power,networking and connectivity
- Azs in a Region are connected through low-latency links
- Increase availability and fault tolerance of applications in same reagion.

Region Code	Region	Availability Zones	Availability Zones List
us-east-1	US East (N. Virginia)	6	us-east-1a us-east-1b us-east-1c us-east-1d us-east-1e us-east-1f
eu-west-2	Europe (London)	3	eu-west-2a eu-west-2b eu-west-2c
ap-south-1	Asia Pacific(Mumbai)	3	ap-south-1a ap-south-1b ap-south-1c

22. Exploring compute services in AWS

donderdag 24 april 2025 15:53

EC2 (Elastic Compute Cloud)

EC2 instances = Virtual machines in AWS
EC2 Service = Provision EC2 instances or virtual machines

Important features:
- **Create & manage lifecycle** of EC2 instances
- **Attach storage** to your EC2 instances
- **Load balancing** for multiple EC2 instances

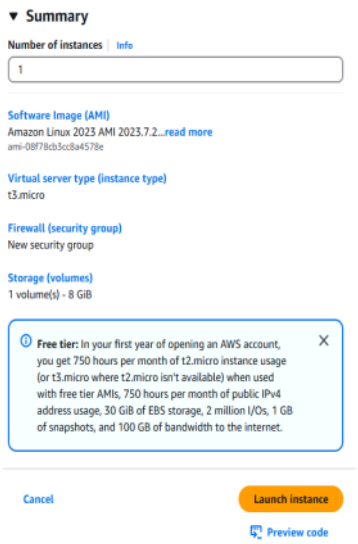
Instance

Te configureren voor creeëren instance:
- **AMI** = Amazone Machine Image
  (Amazon Linux, macOS, Ubuntu, windows,…)
  What operating system and what software do you want to run on instance?

- **Instance type / Instance family** = de hardware
- **Intance size** (t3.nano, t3.micro, …) = quantity of software
- **Elastic Block store**: attach disks to E2C Instances

- **Key pair (login)** --> wordt ook gedownload. Iedereen met key pair kan connecteren met instance.
  public/private key

- **Network settings** --> Firewall --> Create security group - Allow SSH traffic from Anywhere
  control incoming/outgoign traffic to/from AWS resources (E2C instances, databases, …)



Nu kan je connecteren met deze instance.

Instance states:
Stop = temp. Stoppen
Terminate = stop + delete

Tags: belangrijk om bvb aan te duiden welke environment en business unit

Best Practice: stop instance(s) wanneer niet mee bezig

Setting up webserver

1) Launch an instance
2) Connect to instance

3) Installeren Apache
    a. Hiervoor moet je super user zijn, command:
       sudo su
    b. Gebruik van package manager voor installatie.
       bvb yum
       ---> eerst updaten:
       yum update -y
       ---> dan install appache
       yum install httpd
4) Starten server (via systemcontrol - systemctl):
   systemctl start httpd
5) Connecteren naar server: op dashboard bij instance staat er een Public ip-adres.
   Paste in server.
   Standaard werkt dit niet want security group is enkel ingesteld op ssh.
6) Wijzigen security group: edit inbound rules.
   (Dashboard --> Instance aanklikken --> Security --> klik op security groups url
   Toevoegen rule: HTTP - 0.0.0.0/0

IaaS vs PaaS

IAAS: Infrastructure as a Service
Only use infrastructure from cloud provider.
ex: Using VM Service to deploy our apps/databases
Cloud provider is responsible for:
- Hardware, networking & virtualization

You are responsible for:
- OS upgrades & patches
- Application code & runtime
- Configuring load balancing
- Auto scaling
- Availability
- …

PAAS: Platform as a Service
Use a platform provided by the cloud.
Cloud provider is responsible for:
- Hardware, networking & virtualization
- OS (incl. upgrades & patches)
- Application runtime
- Auto scaling, availibility & load balancing etc..

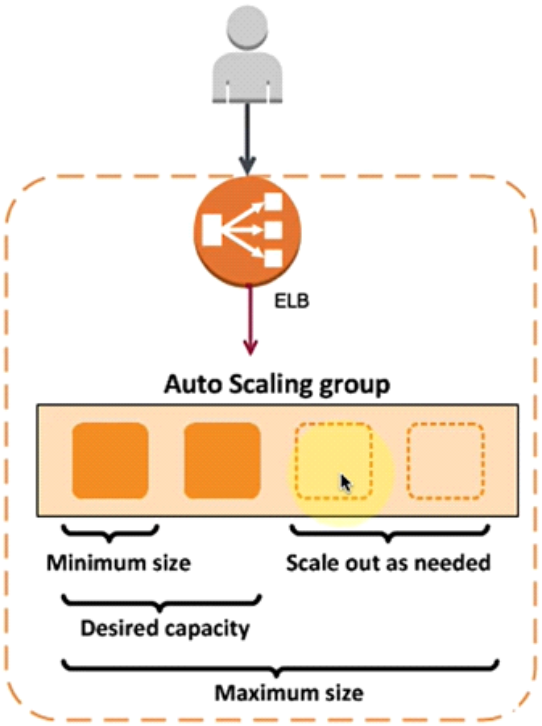
You are responsible for:
- Configuration (of application & servies)
- Application code (if needed)

Examples:
Compute: AWS Elastic Beanstalk, Azure App Service, Google App Engine
Databases: Relational & NoSQL (Amazon RDS, Google Cloud SQL, Azure SQL Database, etc..)
Queues, AI, ML, Operations etc…

AWS Elastic BeanStalk

Simplest way to deploy and scale your webapplications in AWS.
Provides E2E web application managment.
Automatic load balancing, auto scaling, …

Same application is deployed to multiple VMs
Auto Scaling Groups.
ELB = Elastic Load Balancing (Distribute traffic across multipe EC2 instances)

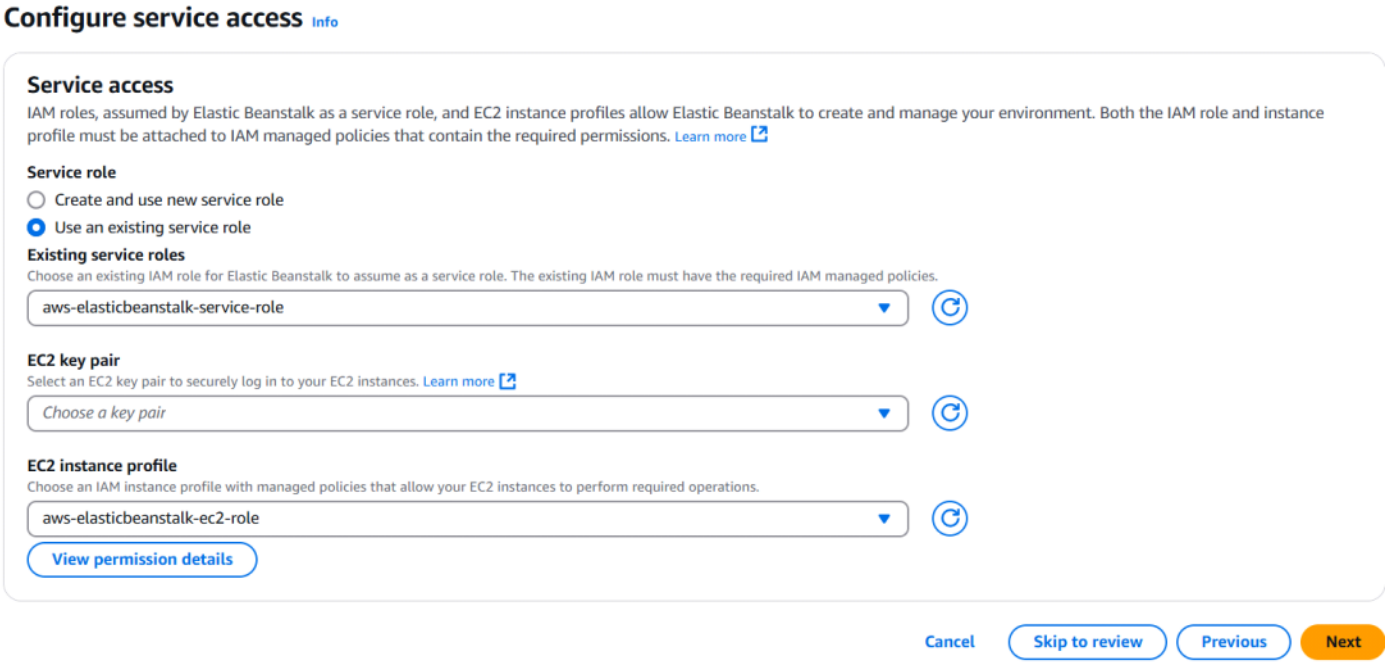


Setting up webapplication

Zoek in AWS naar "Elastic Beanstalk" --> Create application
Hier moet je een platform kiezen (Docker, Java, Python, PHP, ...)

BELANGRIJK!!!! Eerst aws-elasticbeanstalk-service-role aanmaken:
https://stackoverflow.com/questions/30790666/error-with-not-existing-instance-profile-while-trying-to-get-a-django-project-ru/76620598#76620598
---> via IAM een role met die exacte naam en permissions:
AWSElasticBeanstalkWebTier, AWSElasticBeanstalkWorkerTier, AWSElasticBeanstalk MulticontainerDocker

----> deze role toewijzingen als service role tijdens aanmaak application/environment:



Bij Capacity --> Auto scaling group: load balanced met min 1 en max 2.

Elastic Beanstalk creeërt:
Een instance
+ environment
+ deployed de demo applicatie daarop.
+ Auto Scaling Group
+ Load balancers

Een application kan meerdere environments hebben (één voor DEV, ACC, …)
Via Elastic Beanstalk --> klik op application --> dit geeft een lijst met environments en de knop "create new environment".
Bij het uploaden van de applicatie moet je dan de environment kiezen.

MicroServices
Flexibility to innovate and build applications in different programming languages.

BUT deployment becomes complex!
---> oplossing: **containers**

Voor elke microservice een docker container.

3 containers --> 1 DockerEngine --> 1 HostOS --> 1 Cloud infrastructure

Docker is cloud neutral.
Docker containers are light weight --> compared to VM as they don't have a Guest OS.

Container Orchestration
Typical Features:
- Auto Scaling: scale containers based on demand
- Service Discovery: Help microservices find one another
- Load Balancer: distribute load among multiple instances of a microservices
- Self Healing: do health checks and replace failing instances
- Zero downtime deployments: release new versions without downtime.

Container Orchestration Options:
- Amazon EKS: Cloud Neutral --> geen free tier
  Amazon Elastic Kubernetes Service
  Based on Kubernetes: open source container orchesetartion
- Amazon ECS: AWS Specific.
  Amazon Elastic Container Service
- Fargate: Serverless ECS/EKS --> geen free tier

Elastic Container Service (ECS)
Using Fargate: NIET free tier

Cluster --> Service --> Task definition --> Container definition

Cluster is een groep services, service is groep task definitions, etc…

Service: run and maintain a specified number of simaltunaeous instances of a task

Serverless in AWS --> Lambda
Does NOT mean no servers.
You don't worry about infrastructure (ZERO visibility into infrastructure)
Pay for use: Zero requests --> zero costs.

AWS Lambda
- Truly serverless
- Don't worry about servers, scaling or availability
- Pay for what you use (number of requests, duration of requests, memory)
- Supported languages: Java, Go, Node.js, C#, …

Je creeërt eerst een "Function" en kiest daar een Runtime (Node.js, java, …) en Architecture (x86

_64 of arm64)

Je kan code testen door het creeëren van Test event

Metrics - Logs - Traces
Cloudwatch: logging
WS X-Ray: Traces --> moet enabled worden via configuration

28. Deploying Spring Boot Applications to AWS

dinsdag 6 mei 2025 14:06

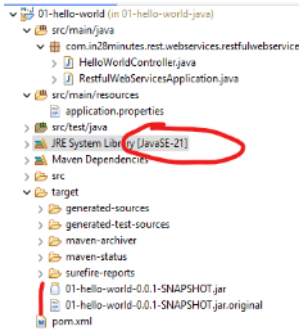
https://github.com/in28minutes/master-spring-and-spring-boot/tree/main/91-aws

Server port
In application.properties staat de port ingesteld op 5000, is het meest ideale voor Elastic Beanstalk.
`server.port=5000`

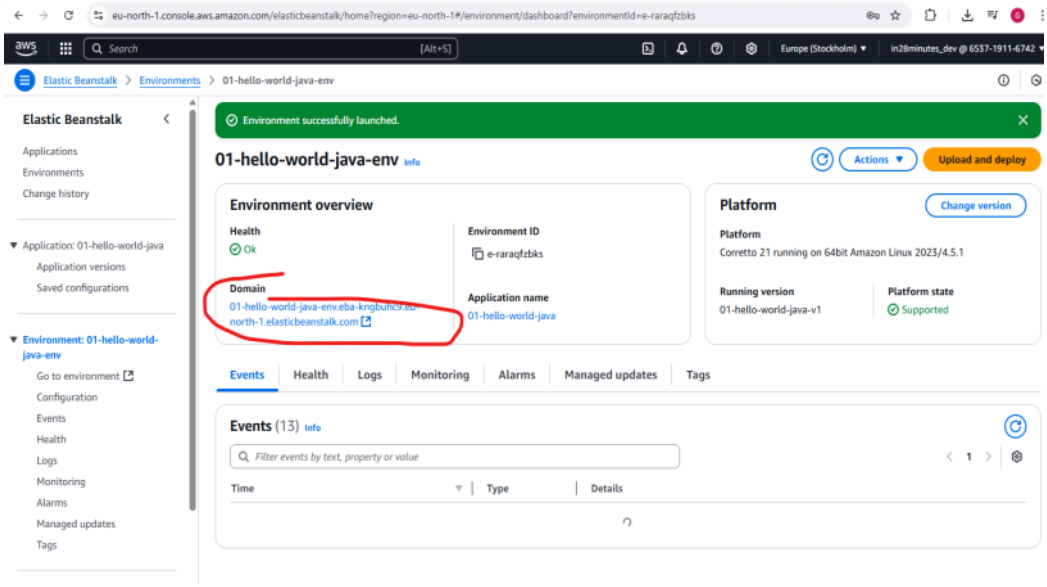
Extract JAR
In eclipse rechtsklik op Spring-project: Run As --> Maven Build...
Bij Goals: clean package
De JAR staat in /target folder

In Elastic Beanstalk
--> Create Application
--> Create Environment
Platform: Java
Platform branch: moet >= java versie zijn van Spring-project (dit zie je aan de JRE System Library folder)

Application code: Local file --> upload.
Zet hierin de .jar uit de target folder (maven-project/target)



Op het environment scherm staat de link naar de applicatie:



In EC2 bij Instances kan je zien dat er 1 instance running is voor de zonet gemaakte application environment.

Je kan meerdere environments creeëren per application.

Application versions geeft overzicht van de verschillende versies.
Hier kan je nieuwe versie uploaden en dan via Actions --> Deploy die versie deployen.
(Of gewoon via Appications --> environment --> upload and deploy

OM KOSTEN TE VERMIJDEN STEEDS APPLICATION VERWIJDEREN NA TESTEN!

REST API met SQL

Readme voor docker:
https://github.com/in28minutes/master-spring-and-spring-boot/tree/main/91-aws/02-rest-api-mysql

Belangrijke wijzigingen

Port: 5000

In application.properties dynamische datasource gegevens:

```
spring.datasource.url=jdbc:mysql://${RDS_HOSTNAME:localhost}:${RDS_PORT:3306}/
${RDS_DB_NAME:social-media-database}
spring.datasource.username=${RDS_USERNAME:social-media-user}
spring.datasource.password=${RDS_PASSWORD:dummypassword}
```

Deze environment variables zijn automatisch aangemaakt op de AWS server wanneer je database toevoegt aan je applicatie in Elastic Beanstalk:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html

Wat na de : komt, is de default value. Bvb ${RDS_HOSTNAME:localhost} -> default value is 'localhost'

==> mysql staat in een docker container:
docker run --detach --env MYSQL_ROOT_PASSWORD=dummypassword --env MYSQL_USER=social-media-user --env MYSQL_PASSWORD=dummypassword --env MYSQL_DATABASE=social-media-database --name mysql --publish 3306:3306 mysql:8-oracle

In Elastic Beanstalk --> create new application.
In de stap 3 "Set up networking, database, and tags" kan je database configureren
Dit creeërt een database in "Aurora and RDS" (https://eu-north-1.console.aws.amazon.com/rds/home?region=eu-north-1#databases:)

Rest-api oproepen:
Domain - url + /hello-world

Security Groups
Via EC2 --> Security --> bij Security groups staat een url.
Wanneer je daar op klikt zie je details.
Belangrijk: Inbound rules. By default enkel poort 80 voor HTTP is open.
Dit configureert welk dataverkeer toegestaan is.

Voor de database (te bekijken via "Aurora and RDS" --> databases --> database aanklikken --> Connectivity & security --> VPC security Groups
Hier staat bij inbound rules dat enkel dataverkeer toelaat vanuit de REST-applicatie!!

Full Stack application

Voorbereidingen in Spring-project:
Er moet een health-check voorzien worden (REST-mapping die verwijst naar de root "/")
In **HelloWorldController**:

```
@GetMapping(path = "/")
public String returnSomethingAtRootUrl() {
    return "Congratulations!";
}
```

In **RestfulWebServicesApplication:**
Configuratie van cors.
Idealiter bij .allowedOrigins toevoegen van de URL van de frontend-applicatie.
In voorbeeld is dit voor het gemak op * gezet:

```
@Bean
public WebMvcConfigurer corsConfigurer() {
    return new WebMvcConfigurer() {
        public void addCorsMappings(CorsRegistry registry) {
            registry.addMapping("/**")
                .allowedMethods("*")
                .allowedOrigins("*");//#CHANGE //NOT RECOMMENDED FOR PRODUCTION
        }
    };
}
```

In **application.properties**:
Poort zetten op default poort voor elastic beanstalk (= 5000)

```
server.port=5000
```

Samenvatting werking backend-authenticatie:
POST wordt verstuurd naar http://localhost:5000/authenticate met als body username en password.
Die retourneert een body met Token in.
Die Token moet meegegeven worden met alle toekomstige requests via de header van het type "Authorization" met als waarde: Bearer <deToken>

In de Frontend React:
In ApiClient.js de baseURL aanpassen --> zie onder deploy REST-API

DEPLOY REST-API
Create JAR-file (Run as --> maven Build -> goals: clean package)
Uploaden naar Elastic Beanstalk

Wanneer gedaan moet de url gekopieerd worden en in de Frontend gezet
(link staat in E.B. onder "domain" bij environment)
Vb: http://03-rest-api-full-stack-h2-env.eba-xiizsagg.eu-north-1.elasticbeanstalk.com/
--> moet in ApiClient.js gezet worden

```
src > components > todo > api > JS ApiClient.js > ...
1  import axios from 'axios'
2
3  export const apiClient = axios.create(
4      {
5          //baseURL: 'http://localhost:5000' //#CHANGE
6          baseURL: 'http://03-rest-api-full-stack-h2-env.eba-xiizsaqg.eu-north-1.elasticbeanstalk.com/'
7      }
8  );
```

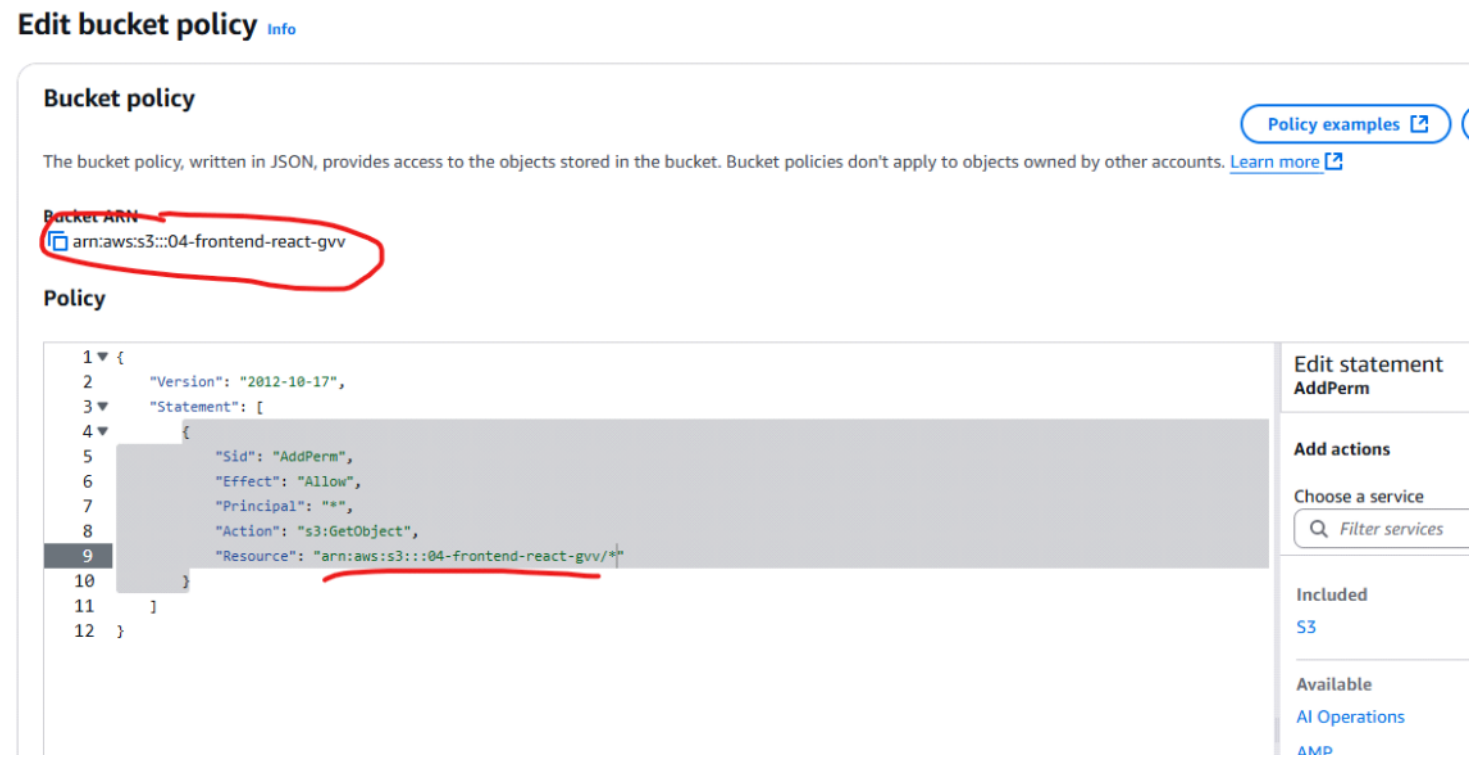
DEPLOY FRONTEND
--> Amazon S3

In Angular creeër een productieklare applicatie via commando:
npm run build
Wordt gegenereerd naar de build-folder van het project

Amazon S3 --> Create Bucket
Disable "Block all public access"
Nadat bucket gecreëerd is, die selecteren en "Upload Files".
Uploaden alle files in de /build folder van het frontend-project

Daarna in Tabblad **Properties** --> Static website hosting
Index document invullen (index.html)
Dan wordt een bucket website endpoint getoond (bvb: http://04-frontend-react-gvv.s3-website.eu-north-1.amazonaws.com)
!! Dit geeft nog een access denied (forbidden) fout

In Tabblad **Permissions:**
Edit bucket policy --> add statement:



Nadien terminate environment!

Functional programming

vrijdag 16 mei 2025 8:08

Method Reference

```
numbers.stream().forEach(System.out::println);
--> static methods

==> vervangt "x -> System.out.println(x)"
```

Lambda Expression

```
numbers.stream()
       .filter(number -> number%2==0)
       .forEach(System.out::println);
```

Mapping

```
numbers.stream()
       .map(t->t*2)
       .forEach(System.out::println);
```

Operations on Streams

Intermediate Operations

Transform or filter elements and return a new stream.
These operations do not execute until a terminal operation is called.
Examples of some common intermediate operations are **filter**, **map**, **sorted**, **distinct**, **limit**, **flatMap**, **skip**, etc.

Terminal Operations

Consumes the stream and produces a result.
Examples of some common terminal operations are **collect**, **forEach**, **reduce**, **count**, **anyMatch**, **allMatch**, **noneMatch**, etc.

Operation	Return Type	Type Of Operation	What It Does?
filter()	Stream<T>	Intermediate	Returns a stream of elements which satisfy the given predicate.
map()	Stream<R>	Intermediate	Returns a stream consisting of results after applying given function to elements of the stream.
distinct()	Stream<T>	Intermediate	Returns a stream of unique elements.
sorted()	Stream<T>	Intermediate	Returns a stream consisting of elements sorted according to natural order.
limit()	Stream<T>	Intermediate	Returns a stream containing first *n* elements.
skip()	Stream<T>	Intermediate	Returns a stream after skipping first *n* elements.
forEach()	void	Terminal	Performs an action on all elements of a stream.
toArray()	Object[]	Terminal	Returns an array containing elements of a stream.
reduce()	type T	Terminal	Performs reduction operation on elements of a stream using initial value and binary operation.
collect()	Container of type R	Terminal	Returns mutable result container such as List or Set.
min()	Optional<T>	Terminal	Returns minimum element in a stream wrapped in an Optional object.
max()	Optional<T>	Terminal	Returns maximum element in a stream wrapped in an Optional object.
count()	long	Terminal	Returns the number of elements in a stream.
anyMatch()	boolean	Terminal	Returns true if any one element of a stream matches with given predicate.
allMatch()	boolean	Terminal	Returns true if all the elements of a stream matches with given predicate.
noneMatch()	boolean	Terminal	Returns true only if all the elements of a stream doesn't match with given predicate.
findFirst()	Optional<T>	Terminal	Returns first element of a stream wrapped in an Optional object.
findAny()	Optional<T>	Terminal	Randomly returns any one element in a stream.

Things to remember about Streams:

- Don't store data
- Are lazy - intermediate operations are not executed immediately
- Are immutable
- Cannot be reused

Functional interfaces

Any interface with a SAM(Single Abstract Method) is a functional interface, and its implementation may be treated as lambda expressions.

- Consumer<T>: Processes an input
- Predicate<T>: tests a condition and returns boolean
- BiPredicate<T,U>: tests two inputs and returns boolean (f.e. equals)
- Function<T,R>: Transforms data and returns the result
- BiFunction<T,U,R>: combines two inputs to one result
- BinaryOperationor<T>: 2 inputs, 1 output, all the same type
  BinaryOperator<Integer> sum = (a,b) -> a + b
- Supplier<T> : provides a value without any input (bvb Random)
- UnaryOperator<T>:Single input, same type (bvb vierkantswortel)

Inleiding

vrijdag 19 september 2025 14:03

https://github.com/in28minutes/full-stack-with-angular-and-spring-boot

Ng new <projectname>

Ng serve

In README.md staan alle basis-commando's

Ng test
---> unit tests (*.spec.ts)

E2E zijn niet meer inbegrepen.
Dus commando "ng e2e" werkt niet meer.
Je kan wel bvb Cypress toevoegen aan het project.

ng generate component welcome --skip-import --standalone

Formulier

maandag 22 september 2025 14:35

Banana in a box: [(…)]

2-way databinding
Gebruik [(ngModel)] voor databinding.
(import te doen in .ts: FormsModule)
<input type="text" name="username" [(ngModel)]="username">

Overzicht

maandag 22 september 2025 14:40

(click)	Component event. Functie binden aan bvb input
[..] = **(..) =**	Input van een component Output van een component
[(ngModel)] = "…"	Banana in a box principe. Component 2-way databinding aan een Component property in .ts bestand.
===	Strict equality. Vergelijkt waarde én type. (tegenhanger is '==' -> loose equality)
***ngIf**	Deprecated vanaf v17
@If (…) { …. }	Manier voor if vanaf angular 17 (oude manier: *ngIf)
App.routs.ts	Plek waar de routing geconfigureerd wordt. voorbeeld: export const routes: Routes = [{path: 'login', component: Login}];
@for (item of items; track $index){…}	Manier voor for vana angular 17 (oude manier: *ngFor="let item of items")
{{item.targetDate | date}}	Pipe. In dit voorbeeld DatePipe. Heeft import nodig van DatePipe in TS-bestand.
@Injectable({providedIn: 'root'})	Maakt component een service.
sessionStorage.setItem('authenticatedUser',username);	Iets opslaan in sessie. ook: .getItem(..), .removeItem(..)
inject(…)	Manier om inject te doen (vanaf angular 14). Before: constructor(private service: MyService) { … } After: private service = inject(MyService)
Message = signal<String>('')	Initialiseren van signal
cdr = inject(ChangeDetectorRef) **this.cdr.detectChanges()**	Kan gebruikt worden om refresh van pagina te forceren. Bvb na ophalen data.
searchTerm = input<string>('');	Input van components

<app-component-result [searchTerm] = "searchTerm()"/> **mandatoryField = input.required<string>()**	Aanduiden input van component dmv […] = Afdwingen dat input geïnitialiseerd is.
searchTermChanged = output<string>(); **<input #searchField (input)="searchTermChanged.emit(searchField.value)"/>** **<app-component-search (searchTermChanged)="searchTerm.set($event)"/>**	Output van components Via .emit wordt een event gecreeërd dat de data verstuurt. Aanduiden input van component dmv (..) =
searchTerm = model<string>(); **<app-test-model-input-search [(searchTerm)]="searchTerm"/>**	Initialisatie van een model. Gebruikt voor 2-way binding. Model is een signal die vanuit de html zowel gelezen als gewijzigd kan worden.
withComponentInputBinding()	Ivm Routing. Te definiëren in provideRouter(routes, withComponentInputBinding())
computed() **filteredProducts = computed(() => this.products().filter(p => p.category.toLowerCase === this.category().toLowerCase));**	Computed signal. Is read-only en haalt zijn waarde uit andere signal(s). Voorbeeld: een gefilterde lijst is een computed() op basis van de basis lijst en de zoekwaarde:
[class]="cat === category() ? 'text-white' : null"	Dynamic class (css)
signalStore()	Onderdeel van ngrx/signals Bevat o.a. **withState** (definiëren van de basis-state), **withComputed** (automatische bewerkingen die moeten gebeuren) en **withMethods** (definiëren van methodes die mogelijk zijn op de store)
patchState(store, { category: 'blabla' }	Onderdeel van ngrx/signals In dit voorbeeld wordt de category in de store geüpdated met "blabla".
[omHetEvenWat] = "blabla() **??** null"	nullish coalescing operator Retourneert de rechter operand wanneer de linker operand null of undefined is.
<ng-content />	Content projection. Placeholder om aan te tonen waarin content moet komen, aangereikt door de parent component.
cartItems()**.reduce((accumulator, item) => accumulator+ item.quantity,0)**	Reduce-methode doorloopt een array en zet alle elementen om naar één resultaat.

	Dit kan in allerlei formaten zijn. Het resultaat is wat in de accumulator staat.
@Directive({ **selector: '[appViewPanel]',** **host: {** **class: 'border border-gray-200 rounded-xl p-6 bg-white'** **}** **})** **Gebruik:** **<div appViewPanel>**	Zelf een directive maken. Dit voorbeeld geeft aan alle DOM-elementen die de directive "appViewPanel" gebruiken, een bepaalde css class mee.

!! Immutable updates
Je maakt copy en daar maak je wijzigingen op.
-->

Pipes

vrijdag 14 november 2025 13:20

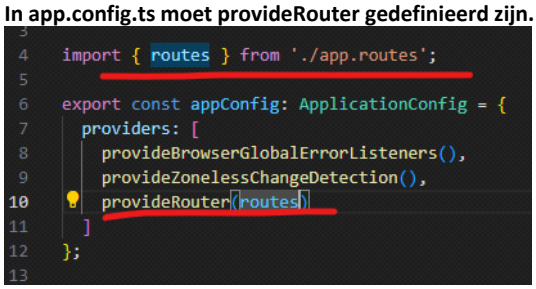
TitleCasePipe (@angular/common)
Zet letters in start met hoofdletter.

{{category | titlecase}}

Routing

woensdag 24 september 2025 11:23

In app.config.ts moet provideRouter gedefinieerd zijn.



```
import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideZonelessChangeDetection(),
    provideRouter(routes)
  ]
};
```

Basis is app.routes.ts

Hierin configureren routing:

export const routes: Routes = [

{ path: '', component: Login },//canActivate, RouteGuardService

{ path: 'login', component: Login },

{ path: 'welcome/:name', component: Welcome},

{ path: '', component: ErrorComponent }**

];

!!! Volgorde is belangrijk, '' moet als laatste. Paths hierna zullen niet werken.**

Gebruik in app:
<router-outlet/>

Navigatie vanuit componenten:

Toevoegen object van type "Router" aan constructor:
constructor(private router: Router) {}

Navigeren:
this.router.navigate(['welcome'])
Mét parameter:

Navigeren vanuit HTML:
!! RouterModule moet geïmporteerd zijn in de .TS
<a routerLink="/todos">HERE

Gebruik parameters
In path:
{ path: 'welcome/:name', component: Welcome},

Parameter meesturen:
this.router.navigate(['welcome', this.username]);

Toevoegen object van type "ActivatedRoute" aan constructor van component waar parameter gebruikt moet worden:
constructor(private route:ActivatedRoute) {}

Ophalen parameter(s):
this.route.snapshot.params['name'];
---> params[]is een Map

Beveiligen pagina's

Aanmaken service die CanActivate implenteert.

export class RouteGuard implements CanActivate{

In path "canActivate" toevoegen:

{ path: '', component: Login, canActivate:[RouteGuardService] },

```
export class RouteGuardService implements CanActivate{

  constructor(private hardcodedAuthenticationService: HardcodedAuthentication, private router: Router) {}
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): MaybeAsync<GuardResult> {
    if(this.hardcodedAuthenticationService.isUserLoggedIn()){
      return true;
    }else {
      this.router.navigate(['login']);
      return false;
    }
  }
```

redirect

vrijdag 24 oktober 2025 14:09

In app.routes.ts:

```
{
      path: '',
      pathMatch: 'full',
      redirectTo: 'products'
   }
```

loadComponent (lazy loading)

vrijdag 24 oktober 2025 14:04

Dit zorgt ervoor dat een pagina pas echt geladen wordt als een gebruiker er naartoe gaat.

Het opgeroepen component moet daarvoor gedefinieerd zijn als "export default class".

Voorbeeld:
export default class ProductsGrid {

In Routing:

```
{
      path: 'products',
      loadComponent: () => import('./pages/products-grid/products-grid')
   }
```

withComponentBinding

vrijdag 7 november 2025 8:22

[RouterLink]

vrijdag 14 november 2025 13:04

Je kan een routerlink samenstellen uit een array:
<a [routerLink]="['/products', category.toLowerCase()]"> test

Eerst het basis path (/products), alle volgende elementen in de array worden er aan geplakt met automatisch "/" tussen.
Dus bvb:
[routerLink] = ['/products', 'bla', 'boem'] geeft:
/products/bla/boem

API oproepen

donderdag 25 september 2025 14:47

Aanpassen app.config.ts
Toevoegen bij providers:
provideHttpClient(withInterceptorsFromDi())

Service creeëren
ng g s service/data/welcomeData

Inject van HttpClient in service:
constructor(private http:HttpClient) {}
(!! provideHttpClient() moet toegevoegd zijn in app.config.ts ==> providers)

.get-methode van HttpClient retourneert een **Observable**.
Tussen <…> geef je aan wat het type is van die observable.
this.http.get<HelloWorldBean>('http://localhost:5000/hello-world-bean');

Wanneer je path-variabele hebt moet je ticks gebruiken ipv '
this.http.get<HelloWorldBean>(`http://localhost:5000/hello-world/path-variable/${name}`);

OBSERVABLE
Beste approach om asynchroon te implementeren.
Er gebeurt niks zolang je niet subscribed.
.subscribe() is een asynchrone call.
Maar je moet dan nog zeggen wat je wil doen met de response.

Consumen van de Observable:
Voorbeeld: overzetten naar een String

MANIER 1: met Pipe en doorgeven Observable aan template:
Definieer de String als een Observable:
messageFromService$!: Observable<string>;

Opvullen:
this.messageFromService$ = this.dataService.executeHelloWorldBeanService().pipe(map(response=>
response.message));

Op deze manier reageert de frontend wanneer de String effectief opgevuld is.
Moest het gewoon een String zijn dan kan het zijn dat de frontend refreshed nog voor hij opgevuld is en zie je dus lege String.

Afdrukken in html (AsyncPipe moet imported worden in .ts)
```
@if(messageFromService$ | async; as msg){
  {{msg}}
}
```

De async pipe doet drie dingen voor je:
1. Subscribe op de Observable.
2. Geeft de laatste waarde door aan de template.
3. Unsubscribe automatisch wanneer de component verdwijnt → geen memory leaks.

De ;as messageFromService geeft de string-waarde vd observable een lokale variabele binnen het block.

🔑 **Kort samengevat**
messageFromService$ = stream (Observable)
| async = haal de laatste waarde eruit
as messageFromService = geef die waarde een naam binnen dit block
@if (...) { ... } = toon de HTML alleen als er effectief een waarde is

Eventueel met Error-handling:

```
this.messageFromService$ = this.dataService.executeHelloWorldBeanService()
  .pipe(
    map(response => response.message),
    catchError(error => {console.error('Error occured:',error); return of('Error occured');})
  ); //end pipe
```

MANIER 2: doorgeven String aan template.
Met signal.
En met next, error, complete:

```
msgString = signal<string>('');
```

```
this.dataService.executeHelloWorldBeanService().subscribe({
  next: (response) => this.msgString.set(response.message),
  error: (error) => console.log(error),
  complete: () => console.log('Request completed')
});
```

Afdrukken in template is gewoon met {{msgString()}}

MET Path Variable

Communicatie tussen componenten

woensdag 22 oktober 2025 16:05

Content projecten

dinsdag 18 november 2025 10:48

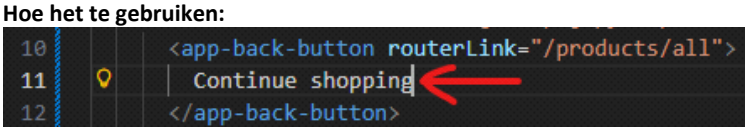
Met <ng-content />

Voorbeeld:

Het re-usable component:

```
@Component({
  selector: 'app-back-button',
  imports: [MatAnchor, MatIcon, RouterLink],
  template: `
    <button matButton="text" [routerLink]="navigateTo() ?? null" class="ms-2 flex items-center gap-1">
      <mat-icon>arrow_back</mat-icon>
      <ng-content/>
    </button>
  `,
  styles: `
```

Hoe het te gebruiken:



```
<app-back-button routerLink="/products/all">
  Continue shopping
</app-back-button>
```

Communicatie via input() en output()

maandag 20 oktober 2025 10:28

Samenvatting

Input/output variabelen
searchTerm = **input('')**

searchTermChanged=**output<string>('')**

Koppelen aan apps
Output: tussen haakjes (…)
Input: tussen vierkante haakjes […]

<…. **(searchTermChanged)** = "searchTerm.set($event)" />

<… **[**searchTerm**]** = "searchTermSignal()" /> (

SIMPEL VOORBEELD

"Master"-component

Bevat een search en een result gedeelte.
Search heeft als output een searchTerm.
Result heeft als input de searchTerm.

```
@Component({
  selector: 'app-master-component',
  imports: [ComponentSearch, ComponentResult],
  template:`
    <app-component-search (searchTermChanged)="searchTerm.set($event)"/>
    <app-component-result [searchTerm] = "searchTerm()"/>
  `
})
export class MasterComponent {

  searchTerm = signal('');

}
```

Search-component

searchTermChanged is een output<String>()
Van zodra iets ingegeven wordt in de input, gaat er een .emit gedaan worden.

```
@Component({
  selector: 'app-component-search',
  imports: [],
  template: `
    @if(!search()){
      <button type="button" class="btn btn-primary" (click)="search.set(true)">
          Search
      </button>
    } @else {
        <input #searchField (input)="searchTermChanged.emit(searchField.value)"/>
        <button type="button" class="btn btn-primary" (click)="search.set(false)">
            Search
        </button>
    }
`
})
export class ComponentSearch {

  search = signal(false);

  searchTermChanged = output<string>();

}
```

Result-component

Het result component bevat een searchTerm die een input() is.
Van zodra searchTerm wijzigt gaat er een filter uitgevoerd worden via de filteredContacts die een computed() is.

```
@Component({
  selector: 'app-component-result',
  imports: [],
  template: `
    <h1>Contacts List</h1>
    <ul>
    @for(contact of filteredContacts(); track $index){

        <li>{{contact.name}} - {{contact.phone}} - {{contact.email}}</li>

    }
    </ul>
  `
})
export class ComponentResult {
  searchTerm = input('');


  contacts = signal<Contact[]>([ ···
  ]);

  filteredContacts = computed(() => {
    if(!this.searchTerm()){
      //If no searchTerm: return whole list
      return this.contacts();
    }

    //Filter list
    return this.contacts().filter(c => c.name.toLowerCase().includes(this.searchTerm().toLowerCase()));

  })

}
```

Via "model" (ipv output)

donderdag 23 oktober 2025 13:57

SIMPEL VOORBEELD

Aanmaken model variabele:
searchTerm = model<string>();

Een model variabele kan zowel gelezen als geschreven worden vanuit template.
(signal kan enkel gelezen worden vanuit template)

Koppelen aan app:
[(...)] (banana in a box principe)

"Master"-component

Search-app heeft als attribuut de model variabele (via banana in a box).
De input daarvan is de signal met de zoekterm.
(dus je kan een signal toewijzen aan een model).

```
6   @Component({
7     selector: 'app-test-model-input-master',
8     imports: [TestModelInputResult, TestModelInputSearch],
9     template:`
10      <app-test-model-input-search [(searchTermModel)]="searchTermSignal"/>
11      <app-test-model-input-result [searchTermInput] = "searchTermSignal()"/>
12    `
13  })
14  export class TestModelInputMaster {
15
16    searchTermSignal = signal('');
17
18  }
19
```

"Search"-component

Hierin wordt de model-variabele aangemaakt en vervolgens aangepast vanuit de template.

```
4   @Component({
5     selector: 'app-test-model-input-search',
6     imports: [],
7     template:`
8       @if(!searchSignal()){
9         <button type="button" class="btn btn-primary" (click)="searchSignal.set(true)">
10            Search
11        </button>
12      } @else {
13          <input #searchField (input)="searchTermModel.set(searchField.value)"/>
14          <button type="button" class="btn btn-primary" (click)="searchSignal.set(false)">
15              Search
16          </button>
17      }
18    `
19  })
20  export class TestModelInputSearch {
21
22    searchSignal = signal(false);
23
24    searchTermModel = model<string>();
25
26  }
```

"Result"-component

Hier blijft gebruik gemaakt worden van een input(..)

```
@Component({
  selector: 'app-test-model-input-result',
  imports: [],
  template:`
    <h1>Contacts List</h1>
    <ul>
    @for(contact of filteredContacts(); track $index){

        <li>{{contact.name}} - {{contact.phone}} - {{contact.email}}</li>

    }
    </ul>
  `
})
export class TestModelInputResult {
  searchTermInput = input('');

  contacts = signal<Contact[]>([ ···
  ]);

  filteredContacts = computed(() => {
    if(!this.searchTermInput()){
      //If no searchTerm: return whole list
      return this.contacts();
    }

    //Filter list
    return this.contacts().filter(c => c.name.toLowerCase().includes(this.searchTermInput().toLowerCase()));

  })

}
```

Via provider en service

donderdag 23 oktober 2025 15:09

De searchTerm wordt gedeclareerd in een service als signal

```ts
import { Injectable, signal } from '@angular/core';

@Injectable()
export class SearchService {

  searchTerm = signal('');

  updateSearchTerm(term: string) {
    this.searchTerm.set(term);
  }

}
```

In hetmaster/parent component, ga je die service als "provider" zetten.
Dit zorgt ervoor dat alle onderliggende components die ook kunnen gebruiken.

```ts
@Component({
  selector: 'app-test-model-input-master',
  imports: [TestComWithServiceResult, TestComWithServiceResult, TestComWithServiceSearch],
  providers: [SearchService],
  template:`
    <app-test-com-with-service-search/>
    <app-test-com-with-service-result/>
  `
})
export class TestComWithServiceMaster {
```

In de search-module, ga je de service injecteren.
En de listener op het input-veld gaat de service methode oproepen:

```
@Component({
  selector: 'app-test-com-with-service-search',
  imports: [],
  template:`
    @if(!search()){
      <button type="button" class="btn btn-primary" (click)="search.set(true)">
          Search via service
      </button>
    } @else {
      <input #searchField (input)="searchService.updateSearchTerm(searchField.value)"/>
      <button type="button" class="btn btn-primary" (click)="search.set(false)">
          Search
      </button>
    }
  `
})
export class TestComWithServiceSearch {

  /*
  * VIDEO: https://www.youtube.com/watch?v=MtTAfjiZxtk
  (vanaf minuut 11)
  */

  searchService = inject(SearchService);

  search = signal(false);

```

De result-module gaat ook de service injecteren en de variabele uit die service gebruiken:

```
@Component({
  selector: 'app-test-com-with-service-result',
  imports: [],
  template:`
    <h1>Contacts List</h1>
    <ul>
    @for(contact of filteredContacts(); track $index){

      <li>{{contact.name}} - {{contact.phone}} - {{contact.email}}</li>

    }
    </ul>
  `
})
export class TestComWithServiceResult {
  searchTerm = inject(SearchService).searchTerm;

```

Interface - Model - Type

vrijdag 24 oktober 2025 8:49

Model

- Representeert data structuur EN business logica
- Verzamelt properties en methodes gerelateerd aan specifieke entiteit of concept
- Gebruikt "export class"
- Meestal datastructuur + constructor + evt andere methodes om data te manipuleren

```
// User Model
export class User {
  id: number;
  username: string;
  email: string;

  constructor(id: number, username: string, email: string) {
    this.id = id;
    this.username = username;
    this.email = email;
  }
}
```

Extends van een class:

```
export class ContactFysiek extends Contact{
  geslacht: string;
  haarkleur: string;
  geboorteDatum: Date;

  constructor(name: string, phone: string, email: string,geslacht: string, haarkleur: string, geboorteDatum: Date){
    super(name,phone,email);
    this.geslacht = geslacht;
    this.haarkleur = haarkleur;
    this.geboorteDatum = geboorteDatum;
  }

}
```

Interface

- Definieert structuur van een object.
- Specifieert properties en hun types, maar voorziet GEEN implementatie
- Gebruikt om bepaalde vorm van objecten te forceren.

```
// User Interface
export interface User {
  id: number;
  username: string;
  email: string;
}
```

Interface kan ook extend worden

Type

```ts
export type Product = {
    id: string;
    name: string;
    description: string;
    price: number;
    imageUrl: string;
    rating: number;
    reviewCount: number;
    inStock: boolean;
    category: string;

}
```

Geen extend maar gebruikt intersection (&):

```ts
type Animal = {
  name: string;
};

type Dog = Animal & {
  breed: string;
};
```

Wanneer gebruik je wat?
Gebruik Interface voor:

Object shapes (bijv. voor classes, React props, Angular services)

Wanneer gebruik je wat?

Gebruik Interface voor:

- Object shapes (bijv. voor classes, React props, Angular services)
- Public API definities
- Wanneer hersamensmelting nuttig kan zijn

Gebruik Type voor:

- Union types, tuples, mapped types
- Complexe type transformaties
- Wanneer je meer flexibiliteit nodig hebt

In de praktijk maakt het vaak niet uit, maar voor object-definities heeft interface de voorkeur in de Angular/TypeScript gemeenschap vanwege betere foutmeldingen en extensiemogelijkheden.

NgMaterial

vrijdag 24 oktober 2025 12:06

- Voor ngMaterial sass gebruiken
- Tailwind css

Theme-kleuren aanpassen

In styles.scss
Prebuilt color pallettes:
https://material.angular.dev/guide/theming

Font aanpassen

Via google fonts
--> Get embed code
Daar staat een link href die je kan toevoegen in je index.html:
<link href="https://fonts.googleapis.com/css2?family=Inter:ital,opsz,wght@0,14..32,100..900;1,14..32,100..900&display=swap" rel="stylesheet">

Daaran in styles.scss aanpassen:
typography: Inter,

En in body { ….. Font-family: Inter … }

Button met icoon

```
import { MatButton, MatIconButton} from '@angular/material/button';
import { MatIcon } from '@angular/material/icon';
```

Met icoon (https://material.angular.dev/components/icon/overview)

```
<button matIconButton>
  <mat-icon>favorite</mat-icon>
</button>
```

Gewone knop (https://material.angular.dev/components/button/overview)

```
<button matButton="filled">
  Sign Up
</button>
```

Ngrx Signals Store

maandag 17 november 2025 12:58

Lightweight store based on signals

- withState
- withComputeds
- withMethods
- withHooks

!! Als je wijzigingen doet op de state is het de bedoeling dat je immutable updates doet.
Je maakt copy's van originele data en doet daar wijzigingen op.
Op die manier propageren de signals correct.
Hiervoor kan je Immer JS gebruiken: https://immerjs.github.io/immer/

(1) Add package ngrx/signals
ng add @ngrx/signals

(2) Create store file
Gewoon een .ts bestand.
bvb ecommerce-store.ts

(3) Export type (definitie van de state)
Opbouwen van de state, de objecten die nodig zijn in de state:

```
5   export type EcommerceState = {
6       products: Product[];
7       category: string;
8   }
```

(4) export const (= signalStore)
Een signalStore waarin gedefinieerd wordt:

- providedIn: 'root' (= waar de store provided moet zijn, root is alles)
- withState (= de 'basis' state), gedefinieerd "as EcommerceState
- withComputed (= wat er moet computed worden in de state…)

Dit is wat je gaat injecteren in je component.

```
12  export const EcommerceStore = signalStore(
13      {
14          providedIn: 'root' //= 1 instantie geshared over de hele
15      },
16      withState({
17  >       products: [···
128         ],
129         category: 'all',
130         wishlistItems: []
131     } as EcommerceState), //end withState
132     withComputed(({ category, products }) => ({
133         filteredProducts: computed(() => {
134             if (category() === "all") {
135                 return products();
136             } else {
137                 return products().filter((p) => p.category.toLowerCase() === category().toLowerCase())
138             }
139         })
140     })), //end withComputed
```

(5) Access to the store
Via Dependency injection.
Design pattern to help reuse logic/code
"Provide" store at different levels (wordt gedefinieerd in de providedIn van d e store.

In het component waar je het wil gebruiken moet je gewoon inject doen:
store = inject(EcommerceStore);

Vervolgens kan je aan de objecten in de state door bvb:
store.category

(6) Updating the state
patchState --> update any state of the store

Simple
Vs
Reactive
--> rxMethod
--> signalMethod

Toevoegen **withMethods** aan de signalStore.
Daarin definieer je een **signalMethod**:

```
        })
    })), //end withComputed
    withMethods((store) => ({
        setCategory: signalMethod<string> ((category: string) => {
            patchState(store, { category });
        })//end setCategory)
    }) //end store
    ) //end withMethods
```

==> withMethods heeft de "store" als parameter.
De "setCategory" methode heeft als input een string (= de category) en gaat daarmee een patchState() doen. Het gaat in de "store" de "category" updaten..

Gebruik in het component:
this.store.setCategory(this.category);

Unidirectional Data Flow
Alle data komt van de store.
Geen functionaliteit in het component.
Maakt testen makkelijker