



Spring Boot en Hibernate

Eerste stappen in Fullstack
development



Leerdoelen

- het MVC model en applicatie architectuur, Fullstack development, en de frameworks Spring en Hibernate
- opzetten en configureren van een Spring Boot applicatie
- opzetten van een database server (XAMPP)
- aansluiten van je applicatie op de db server
- opzetten van een model
- toepassen van Annotations
- opzetten van een repository interface
- opzetten van een service klasse
- aanspreken van endpoints via de browser en met API test software
- bouwen van REST endpoints (GET, POST) in een controller klasse

Planning

- Theorie
- Spring Boot Applicatie configureren
- Endpoints bouwen
- Opzetten Database Server (Xampp)
- Tweede Spring Boot Applicatie configureren
- Back-End Applicatie verder uitbouwen
- Applicatie laten interacteren met DB

Software architecture

Een architectural pattern in software development is een patroon hoe je een applicatie kunt bouwen.

→ sneller applicaties bouwen

- denk aan de aanleg van een nieuwbouwwijk met dezelfde huizen, die hebben architectural pattern

→ structuur in grote applicaties met veel verschillende classes/entiteiten

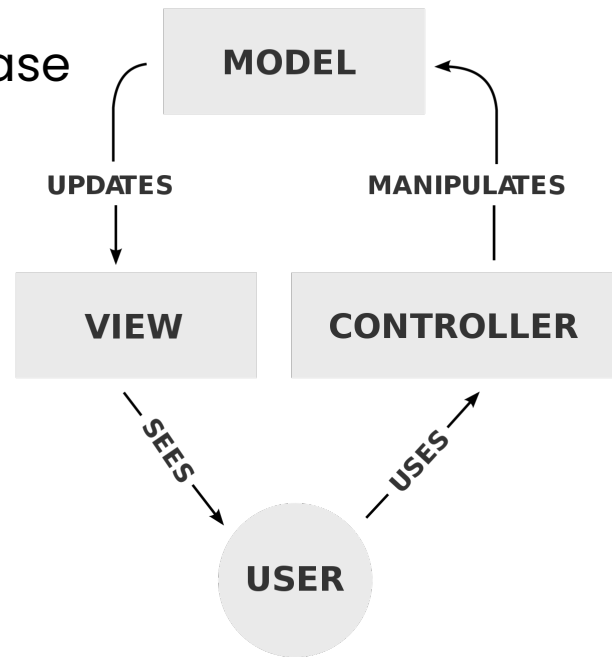
- voorbeeld: Amazon of bol.com → Klant, Product, Bestelling, Afbeelding, etc.

Voorbeelden: Client-Server pattern, Peer-to-Peer pattern, Master-slave pattern, Pipe-filter pattern

Voor ons fullstackers : Model View Controller Pattern

Model-View-Controller pattern

- MVC is een 'design pattern'
 - AKA een sjabloon voor je applicatie architectuur
- Bepaald interactie tussen gebruiker en database



Model

- Structuur voor het type data in je DB (Repository) en de gebruiker
- Een Java Class is een blauwdruk van die data
- Back-End / Repository

```
Persoon.java
1 public class Persoon {
2     String naam;
3     Integer leeftijd;
4     String favoKleur;
5 }
```

The screenshot shows the phpMyAdmin interface. On the left, the database structure tree is visible, showing a database named 'fase3' containing a table named 'persoon'. The main panel displays the 'Structure' tab for the 'persoon' table, showing the columns: 'naam', 'leeftijd', and 'favo_kleur'. Below the structure, a table of data is shown, displaying 7 rows of data.

naam	leeftijd	favo_kleur
Rinse	40	paars
Jan	24	geel
Jan	24	geel
Jan	24	geel
Suus	35	groen
Jannetje	18	cyaan
Sterre	27	Indigo

Controller

- Handelt de input van de gebruiker af via een webapplicatie die draait in een webbrowser
- Endpoints waar de webapplicatie mee communiceert
- Alle manipulaties van de data in de DB
 - Bijvoorbeeld: een gebruiker wilt weten hoeveel gebruikers er in de database opgeslagen zijn
 - Tellen van de gebruikers doet de controller
- Back-end

View

- Bepaald wat de gebruiker te zien krijgt
- Meestal: web-applicatie in browser
- Front-End
- Zonder webapplicatie: Swing / JavaFX → zeer weinig gebruikte libraries voor Java

Front-End

- HTML, CSS, JS, Angular, React, etc.
- Eigenlijk: een website (zoals jullie al gemaakt hebben) die interacteert met een Back-End
- Client

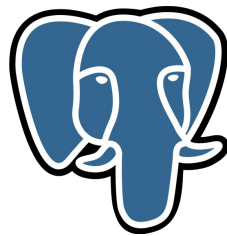
Back-end

- Java (PHP, C#, .Net, etc.)
- Applicatie die zorgt dat gebruikersinput vertaald wordt naar verzoeken aan de Repository
- Create Read Update Delete
 - GET requests – data opvragen uit repository
 - POST requests – data opslaan in/wegschrijven naar repository
 - DELETE requests – data verwijderen uit de repository
 - PUT requests – data updaten/aanpassen in de repository
- Server

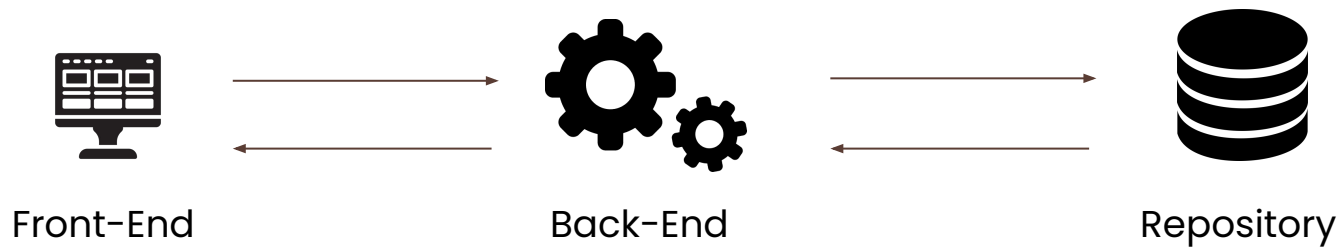
Repository



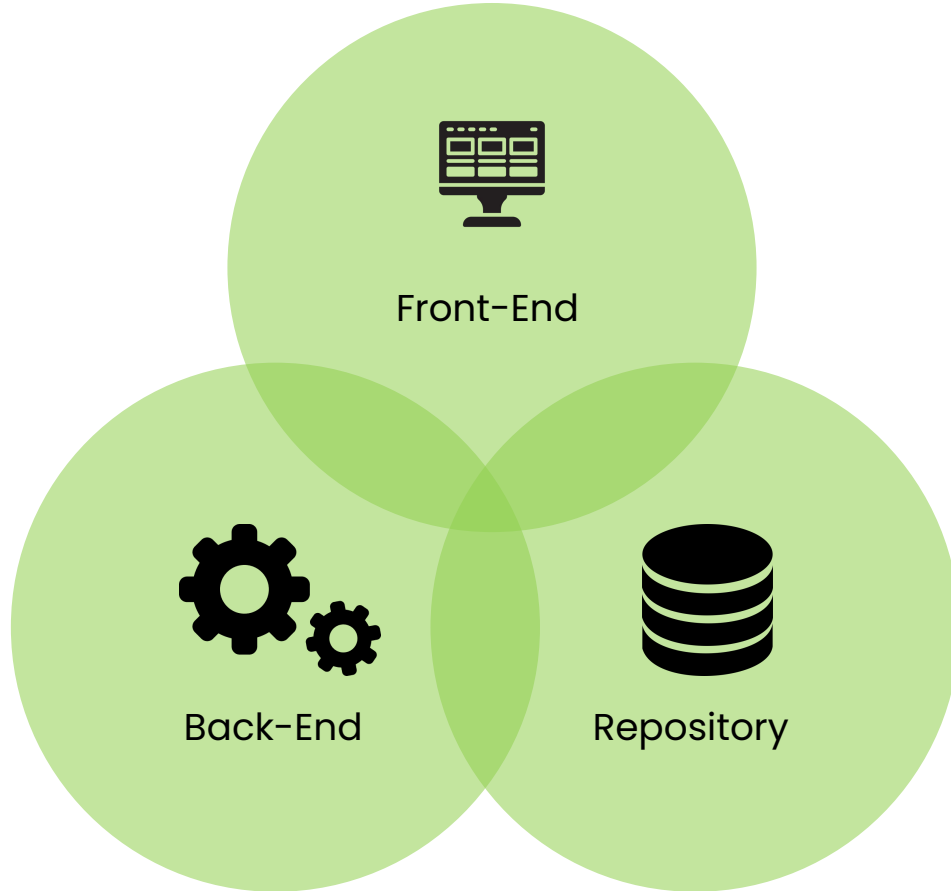
- Meestal: SQL
- Database → nodig om informatie gestructureerd te persisteren
 - Persisteren = **opslaan buiten het werkgeheugen** (bijv. op een harde schijf)
- Server, waar Back-end mee communiceert (kan dezelfde server zijn)
- Wat we hier gaan doen: leren een applicatie te bouwen die informatie kan opslaan in een database



Full Stack



Full Stack Developer:





- Application Framework voor Java
- Back-End
- Ingericht op het bouwen van MVC type architectuur
- Maakt het makkelijk om MVC applicatie voor de Back-End te maken

```
@SpringBootApplication
@RestController
public class DemoApplication {

    @GetMapping("/helloworld")
    public String hello() {
        return "Hello World!";
    }
}
```



- Projecten hebben een directory/package structuur
- Properties bestanden → configureren van connecties naar:
 - Repository
 - Front-End
 - Externe API's
- Dependencies
 - Stukken libraries waar jouw applicatie gebruik van maakt, worden bijgehouden in software project management tools. De twee belangrijkste zijn:
 - Maven: pom.xml
 - Gradle: gradle.built
- Integratie Front-end applicatie in de back-end → static
 - Deployable package



Dit is een extensie van Spring om gemakkelijk en snel Spring applicaties maken. Er zijn veel instellingen van tevoren geconfigureert (op de beste manier, volgens Spring)

→ Doel: Snel applicaties bouwen voor productie

- Stukken standaard programmeerwerk = *Boilerplate code*
- Spring Boot maakt gebruik van libraries en frameworks om Boilerplate te kunnen overslaan



Vermindering boilerplate

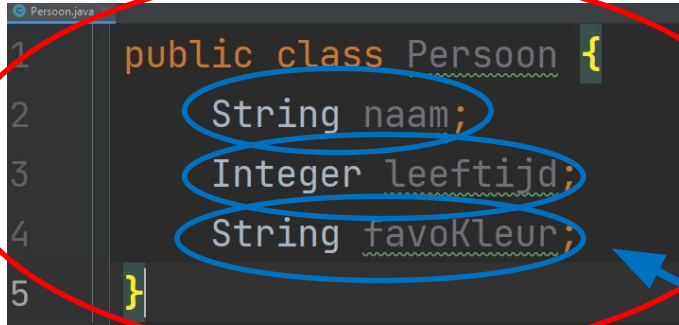
```
1 public class Student {
2     private String naam;
3     private Persoon leraar;
4
5     public Student (String naam, Persoon leraar){
6         this.naam = naam;
7         this.leraar = leraar;
8     }
9
10    public String getNaam() {
11        return naam;
12    }
13
14    public void setNaam(String naam){
15        this.naam = naam;
16    }
17
18    public Persoon getLeraar(){
19        return leraar;
20    }
21
22    public void setLeraar(Persoon leraar){
23        this.leraar = leraar;
24    }
25 }
26
```

Wordt met Spring
Boot en Annotations:

```
3 @AllArgsConstructor
4 @Getter
5 @Setter
6 public class Student {
7     private String naam;
8     private Persoon leraar;
9 }
```

Van 26 lijnen code
naar 9!

Model, Entities, Fields???!?!?



```
1 public class Persoon {  
2     String naam;  
3     Integer leeftijd;  
4     String favoKleur;  
5 }
```

Model of Entiteit / Entity

=

Class

=

Een soort Blauwdruk

Velden / Fields

=

datatype en naam

=

Geeft eigenschappen
aan de entiteit

```
Persoon.java
1 public class Persoon {
2     String naam;
3     Integer leeftijd;
4     String favoKleur;
5 }
```

fase3

- New
- persoon
- information_schema
- mysql
- performance_schema
- phpmyadmin
- test

`SELECT * FROM `persoon``

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Cancel](#)]

☐ Show all | Number of rows

+ Options

naam	leeftijd	favo_kleur
Rinse	40	paars
Jan	24	geel
Jan	24	geel
Jan	24	geel
Suus	35	groen
Jannetje	18	cyaan
Sterre	27	Indigo

Spring Boot naar Database

- Hoe wordt het model in jouw Spring Boot Applicatie verbonden met een tabel in de database?





HIBERNATE

Simpel gezegd: Hibernate is een framework waarmee je de tabellen in je database kunt koppelen aan het Model (Classes / Entities in Back-End - zoals Persoon) van je Back-End applicatie

Door Annotations te gebruiken kun je aangeven welke Class aan welke tabel in de database correspondeert

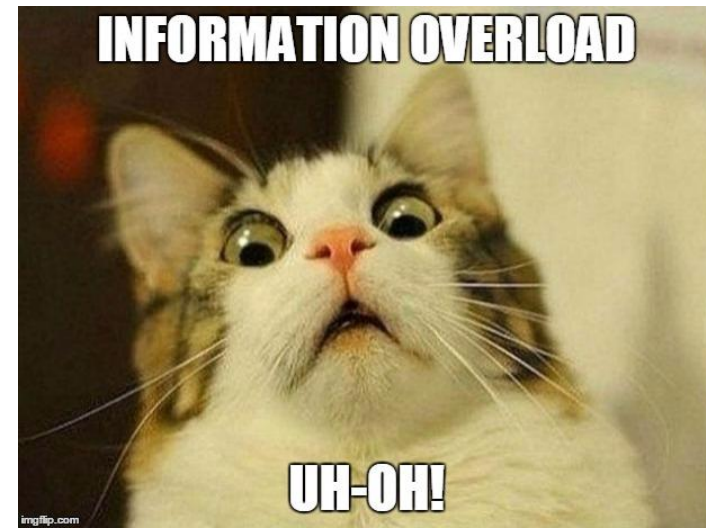
En ook welk veld aan welke kolom

Samenvatting

- MVC is een vorm van software architectuur
- Spring is een framework die het makkelijker maakt om een MVC vorm applicatie te maken
- Spring is een Back-End Framework voor Java om jouw applicatie met een Front-End en een Repository te laten communiceren
- Spring Boot is een extensie waarmee je makkelijk Spring applicaties kunt opzetten
- Hibernate is een framework wat heel vaak samen wordt gebruikt met Spring
- Hibernate zorgt voor de gelijkschakeling tussen de Modellen / Entiteiten in je Back-End applicatie en de tabellen in de Repository/Database
- Daarbij maken we gebruik van Annotations

Genoeg theorie !

- Theorie blijft terugkomen bij het bouwen
- Nu: let's start programming 😊



Wat moeten we doen?

- Spring Boot applicatie bouwen met Spring Initializr
- Applicatie en laten draaien in de IDE
- Eerste Hello World applicatie maken met endpoint



- Surf naar start.spring.io

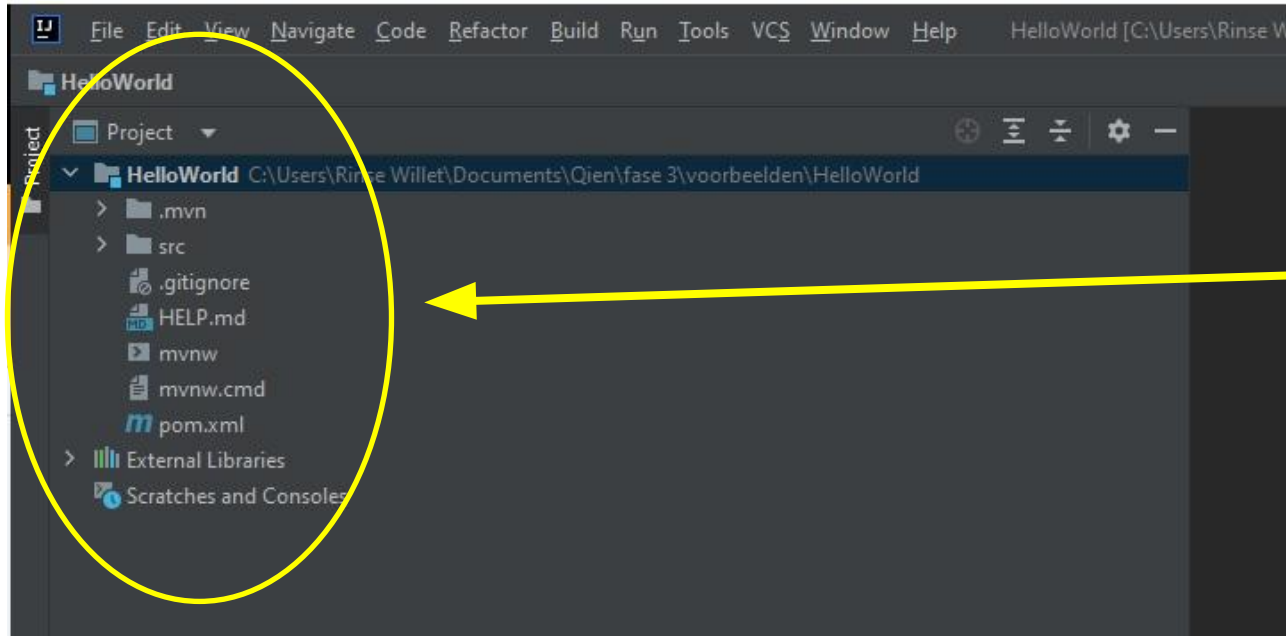
A screenshot of the Spring Initializr web application in a browser. The browser's address bar shows "start.spring.io". The page has a light gray header with the Spring Initializr logo and a hamburger menu icon on the left, and a settings icon on the right. The main content area is divided into three columns. The left column contains the "Project" section with radio buttons for "Gradle Project" and "Maven Project" (selected), and "Language" with radio buttons for "Java" (selected), "Kotlin", and "Groovy". Below this is the "Spring Boot" section with radio buttons for versions: "3.0.0 (SNAPSHOT)", "3.0.0 (RC1)", "2.7.6 (SNAPSHOT)", "2.7.5" (selected), "2.6.14 (SNAPSHOT)", and "2.6.13". The middle column contains the "Project Metadata" section with input fields for "Group" (com.oefening), "Artifact" (HelloWorld), "Name" (HelloWorld), "Description" (Spring Boot Hello World oefening), and "Package name" (com.oefening.HelloWorld). It also has a "Packaging" section with radio buttons for "Jar" (selected) and "War", and a "Java" section with radio buttons for versions "19", "17", "11" (selected), and "8". The right column contains the "Dependencies" section with a button "ADD DEPENDENCIES... CTRL + B". Below this is the "Spring Web" section with a "WEB" tag and a description: "Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container." At the bottom of the page, there is a light blue footer with three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and "SHARE...".



- Project: Maven
- Language: Java
- Spring Boot: 2.7.5 (of default instelling)
- Project Metadata:
 - Group: com.oefening
 - Artifact: HelloWorld
 - Eventueel: Description: vul hier je beschrijving van de app in
- Packaging: Jar
- Java: 8
- Dependencies: Spring Web
- Staat alles goed? → Generate

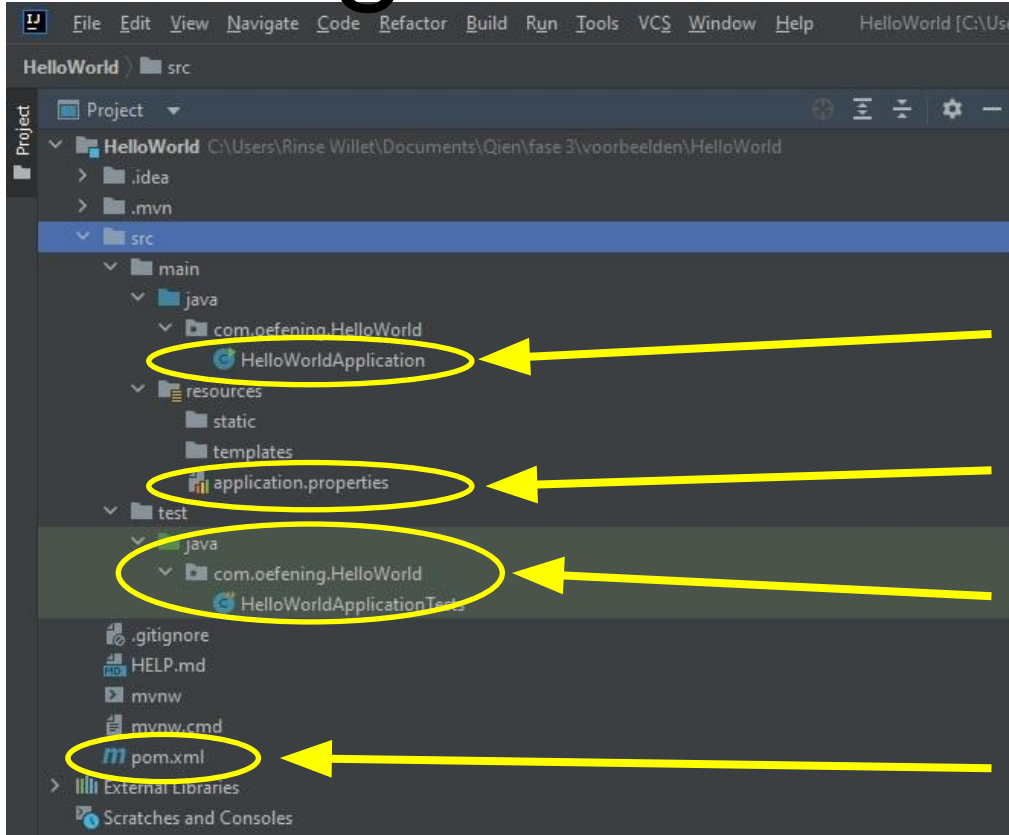
Applicatie laten draaien

- Zip-file uitpakken en uitgepakte map openen in IDE (hier IntelliJ Ideas Community)



Package Structuur

Package Structuur Maven



Main class met
Main Method

Application.properties – hierin staan de
configuratie gegevens van je applicatie

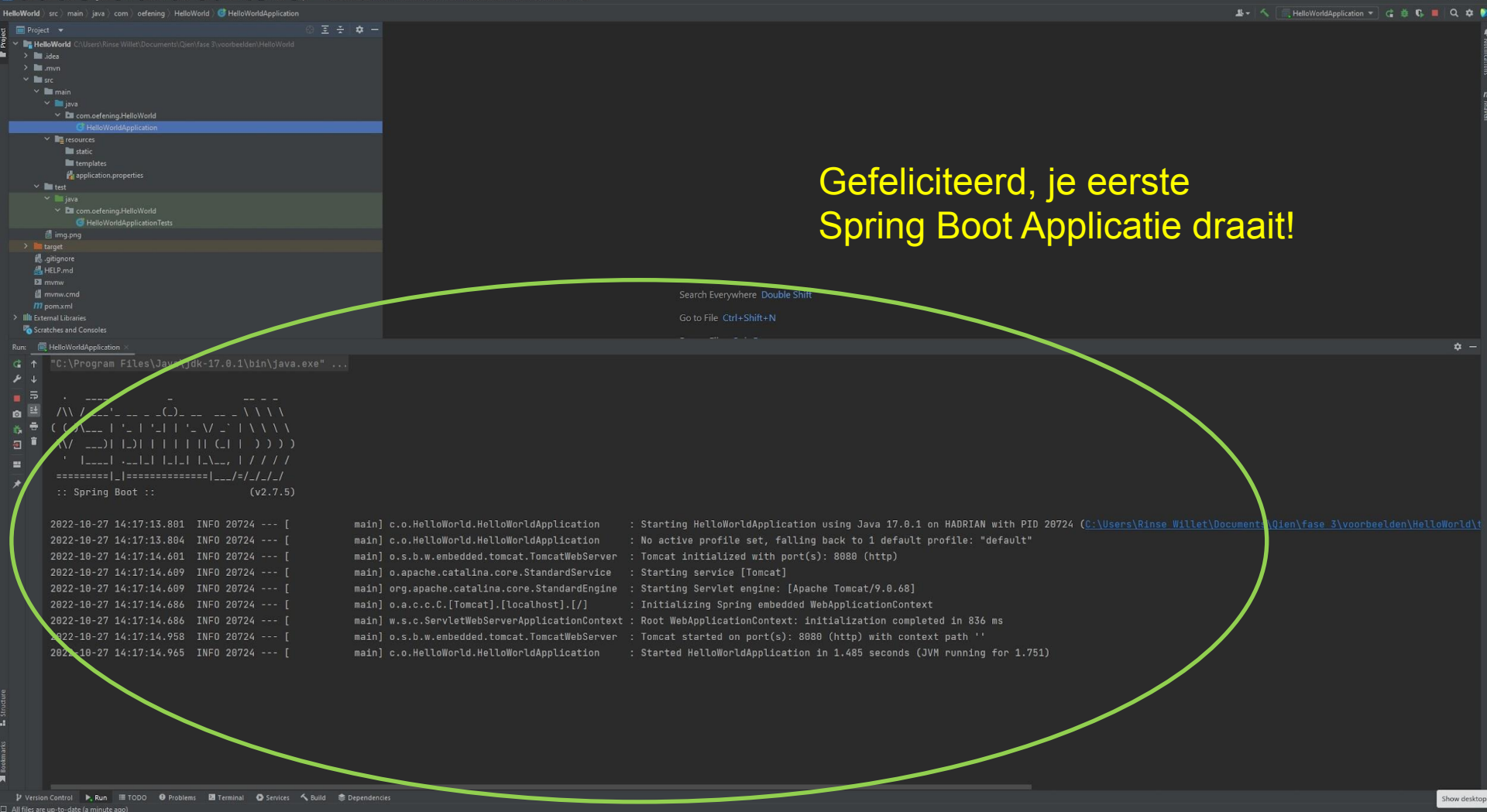
Application Tests – in deze package
kunnen de (JUnit) tests komen te staan

Pom.xml – hierin staan de dependencies
opgeijst die je applicatie gebruikt
(Bijvoorbeeld Spring Web)

Alles klaar? Draaien maar!

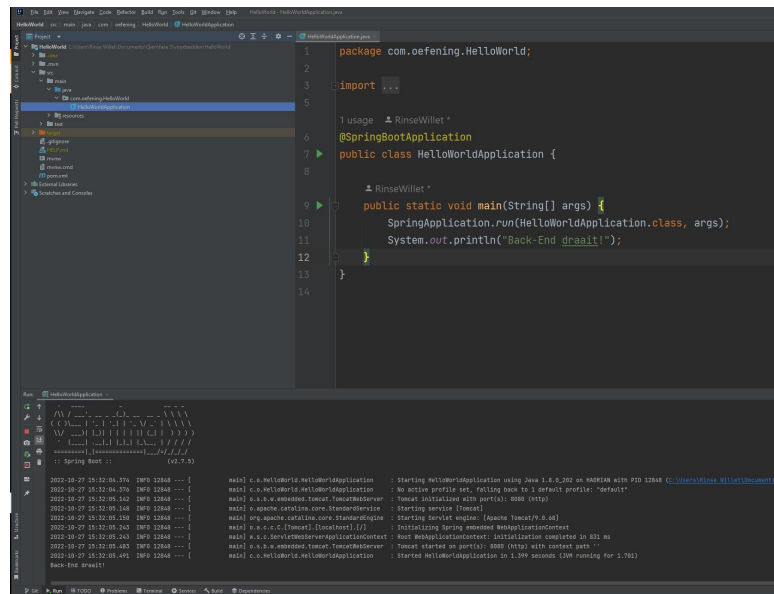
- Vaak worden bij het openen heel veel dependencies gedownload/geresolved
- Build application (vaak een hamertje als icoon)
- Vaak is er nog geen Run configuration gemaakt -> Bij IntelliJ Ideas wordt dat automatisch gedaan door op de rechtermuisknop op de HelloWorldApplication (main class) te klikken → dan Run 'HelloWorldApplication' of Ctrl + Shift + F10

Gefeliciteerd, je eerste
Spring Boot Applicatie draait!



En nu verder

- Stap 0 eventueel: alles naar Git pushen en hier continu je stappen bijhouden
- Stap 1: tekstje toevoegen dat je Back-End draait
 - In: HelloWorldApplication



The screenshot shows an IDE with the following content:

```
package com.oefening.HelloWorld;

import ...

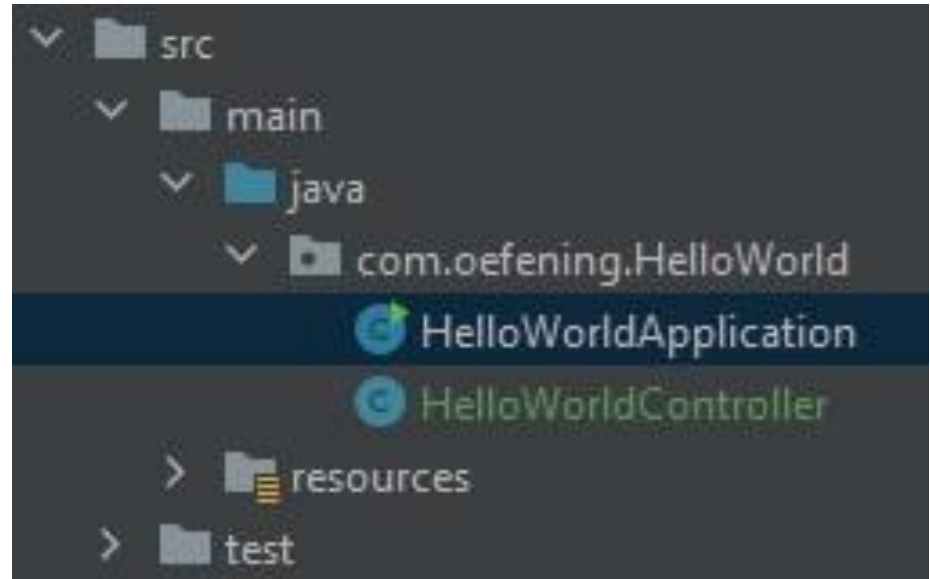
1 usage: RunasWillet *
2
3 @SpringBootApplication
4
5 public class HelloWorldApplication {
6
7     RunasWillet *
8     public static void main(String[] args) {
9         SpringApplication.run(HelloWorldApplication.class, args);
10        System.out.println("Back-End draait!");
11    }
12
13
14 }
```

The console output shows the application running successfully:

```
2022-10-27 15:12:04.374 INFO 12868 --- [main] c.o.h.HelloWorldApplication : Starting HelloWorldApplication using Java 17.0.2 on Windows with PID 12868 (C:\Users\Fine\HelloWorld\bin\HelloWorldApplication.jar)
2022-10-27 15:12:04.374 INFO 12868 --- [main] c.o.h.HelloWorldApplication : No active profile set, falling back to 1 default profile: "default"
2022-10-27 15:12:04.374 INFO 12868 --- [main] c.o.h.HelloWorldApplication : Tomcat initialized with port(s): 8080 (http)
2022-10-27 15:12:04.374 INFO 12868 --- [main] c.o.h.HelloWorldApplication : Starting Tomcat
2022-10-27 15:12:04.374 INFO 12868 --- [main] c.o.h.HelloWorldApplication : Starting server engine (Apache Tomcat/9.0.40)
2022-10-27 15:12:04.374 INFO 12868 --- [main] c.o.h.HelloWorldApplication : Initializing Spring embedded WebApplicationContext
2022-10-27 15:12:04.374 INFO 12868 --- [main] c.o.h.HelloWorldApplication : Root WebApplicationContext: initialization completed in 802 ms
2022-10-27 15:12:04.374 INFO 12868 --- [main] c.o.h.HelloWorldApplication : Tomcat started on port(s): 8080 (http) with context path ''
2022-10-27 15:12:04.374 INFO 12868 --- [main] c.o.h.HelloWorldApplication : Starting HelloWorldApplication in 1.396 seconds (JVM running for 1.761s)
Back-End draait!
```

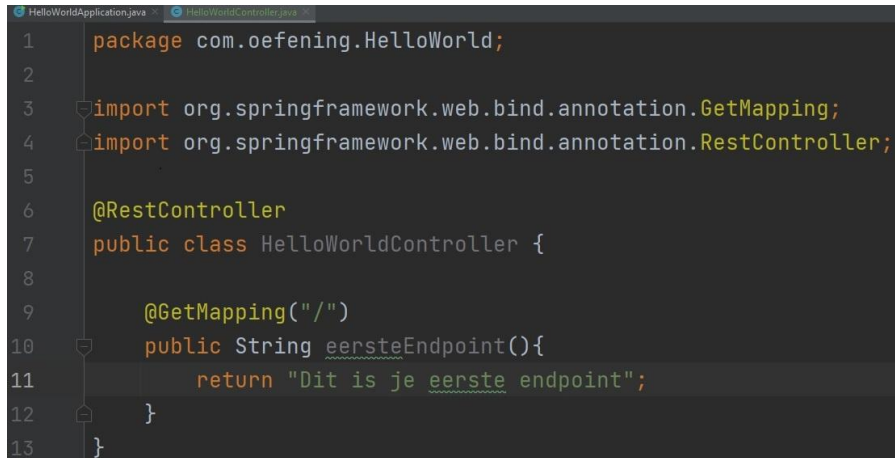
Eerste Endpoint maken

- Endpoint = een methode in de Back-End die aan een URL gekoppeld wordt
- In `com.oefening.HelloWorld` een nieuwe java.class maken:
`HelloWorldController`



Eerste Endpoint maken

- Zet `@RestController` direct boven public class `HelloWorldController`
- Zet in de scope van de class een public method die geen argumenten nodig heeft en een string returnt
- Direct boven deze methode `@GetMapping("/")`



```
1 package com.oefening.HelloWorld;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class HelloWorldController {
8
9     @GetMapping("/")
10    public String eersteEndpoint(){
11        return "Dit is je eerste endpoint";
12    }
13 }
```

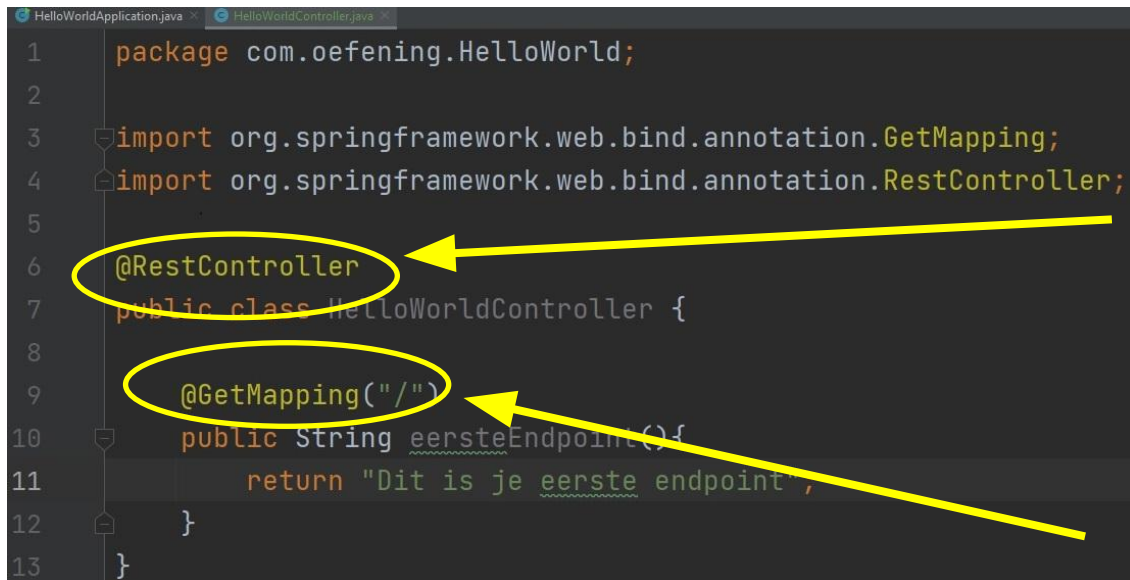
Endpoint testen

- Draai je applicatie
- Open een browser en type localhost:8080
- Als het goed is wordt de String die je methode retournt weergegeven



Dit is je eerste endpoint

Is het gelukt? Wat heb je nu gemaakt?



```
1 package com.oefening.HelloWorld;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class HelloWorldController {
8
9     @GetMapping("/")
10     public String eersteEndpoint() {
11         return "Dit is je eerste endpoint";
12     }
13 }
```

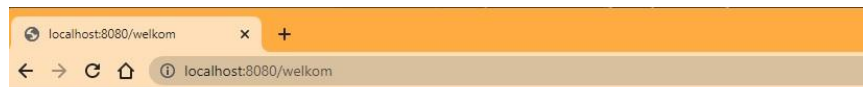
- Annotation die aangeeft dat de class zich als een REST controller moet gedragen (Representational State Transfer Application Program Interface)
- Communicatie architectuur tussen Back-End en Front-End (GET, POST, PUT, DELETE)

Annotation die aangeeft dat een GetRequest gedaan kan worden en op welke **url** dit gedaan kan worden (hier dus: ip-adres/ of **localhost:8080/**)

Nog een Endpoint maken

- Maak nog een public method en laat die een String returnen
- Direct boven deze methode `@GetMapping("/welkom")`
- In een browser: `localhost:8080/welkom`

```
6  @RestController
7  public class HelloWorldController {
8
9      @GetMapping("/")
10     public String eersteEndpoint(){
11         return "Dit is je eerste endpoint";
12     }
13
14     @GetMapping("/welkom")
15     public String welkomEndpoint(){
16         return "Welkom op de Spring Boot Back-End";
17     }
18 }
```



Welkom op de Spring Boot Back-End

Opdracht

- Maak zelf twee endpoints erbij (of meer), op discrete urls
- Laat de ene een Integer returnen en de ander een ArrayList met je favoriete auto's
- Test je endpoints via een browser
- Experimenteer met het endpoints die andere datatypes returned

Gelukt?

```
20
21     @GetMapping("/getal")
22     public Integer getalEndpoint(){
23         return 12;
24     }
25
26     @GetMapping("/arraylist")
27     public ArrayList<String> arrayListEndpoint(){
28         ArrayList<String> arrayList = new ArrayList<>();
29         arrayList.add("Saab");
30         arrayList.add("Honda");
31         arrayList.add("Porsche");
32         return arrayList;
33     }
34
35     @GetMapping("/double")
36     public Double doubleEndpoint(){
37         return 12.556;
38     }
```

Wat heb je gemaakt

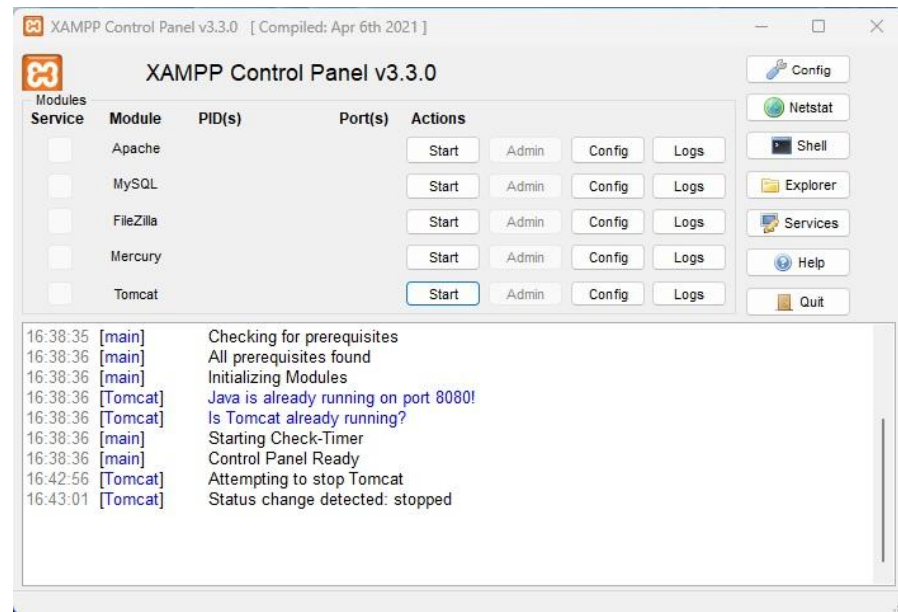
- ✓ Spring Boot applicatie met Spring Initializr
 - ✓ Applicatie laten draaien in de IDE
 - ✓ Hello World applicatie met meerdere endpoints en deze kunnen
 - ✓ Al een stukje van de View en Controller is gebouwd
- Nu: Repository opzetten, Back-end verder uitbouwen

De repository:



XAMPP

- Ga naar <https://www.apachefriends.org/> en installeer XAMPP voor jouw systeem
- Start XAMPP op

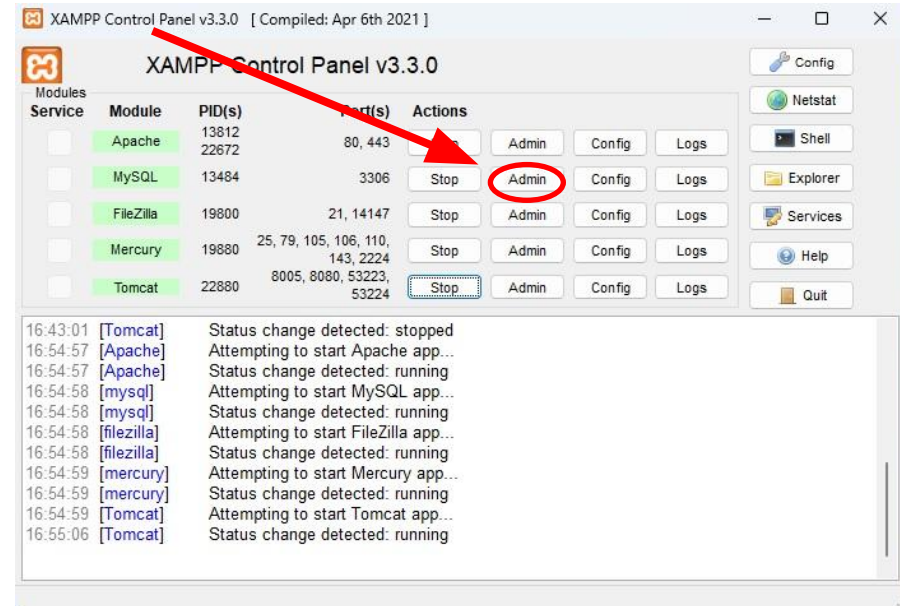


De repository:



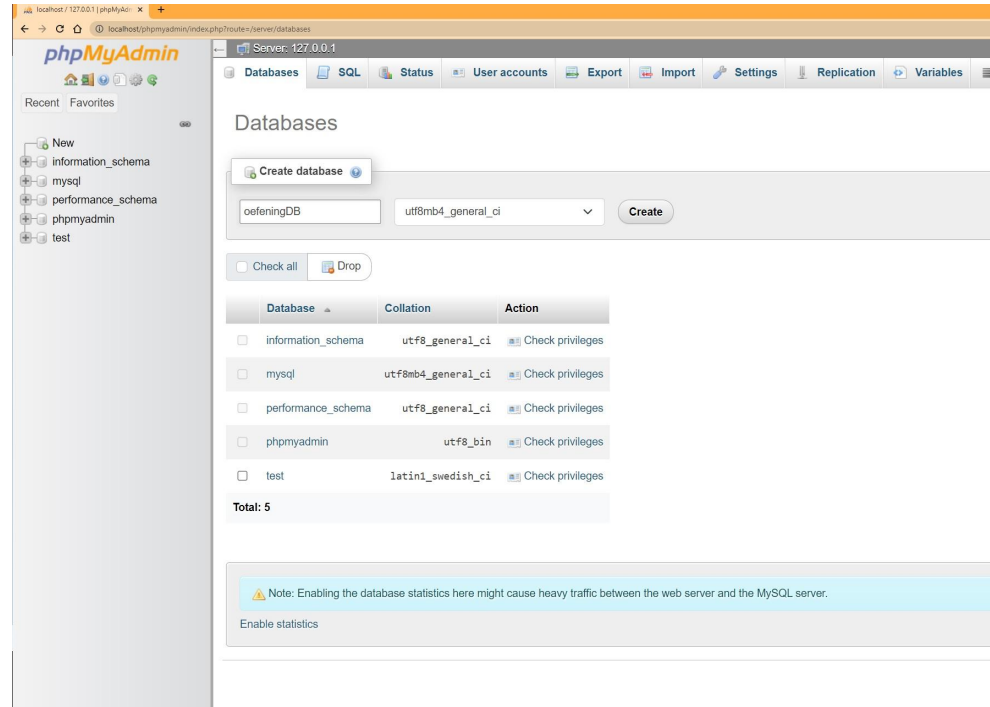
XAMPP

- Start alle services op
- Daarna: start de Admin van MySQL op



In phpMyAdmin

- Klik New en noem je nieuwe Database oefeningDB
- Klik Create



In phpMyAdmin

- Klik op de oefeningDB
- Klik Create new table
- Noem de tabel bedrijf, 3 kolommen
 - Eerste kolom: name id, Type INT, length 10
 - Tweede kolom: name naam, Type VARCHAR, length 255
 - Derde kolom: name aantal_werknemers, Type INT, length 10
- Klik Save

The screenshot shows the 'Create new table' interface in phpMyAdmin. The browser address bar indicates 'Server: 127.0.0.1' and 'Database: oefeningdb'. The interface has tabs for 'Structure', 'SQL', 'Search', 'Query', and 'Export'. The 'Table name' is 'bedrijf' and 'Add' is 1. The columns are defined as follows:

Name	Type	Length/Values
id	INT	10
naam	VARCHAR	255
aantal	INT	10

Below the columns, there are fields for 'Table comments:' and 'Collation:'. The 'PARTITION definition:' section is also visible, with 'Partition by:' and 'Partitions:' fields. At the bottom, there are 'Preview SQL' and 'Save' buttons.

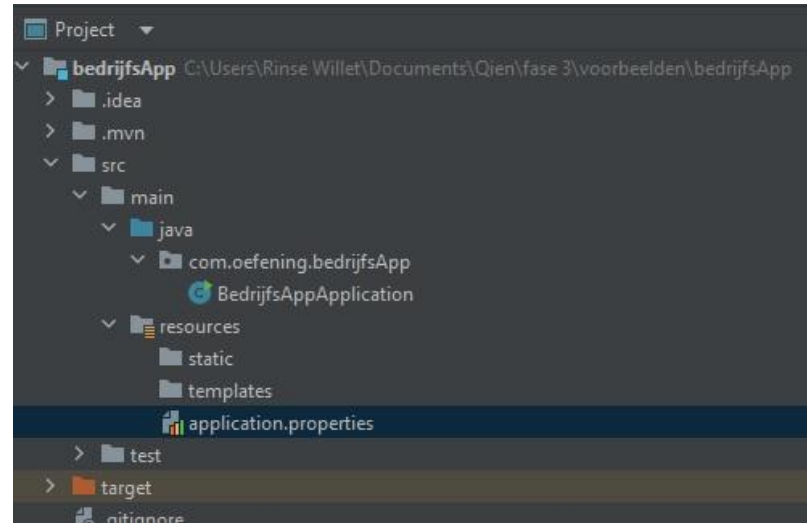
→ Kijk daarna of er een tabel is toegevoegd aan oefeningDB – als het goed is zitten daar geen records in

Opdracht – maak een nieuwe Back-End

- In Spring Initializr, maak een applicatie die:
 - Een Maven Project is en in Java 8 draait met Spring Boot 2.7.5 (of default), packaging JAR
 - `com.oefening.bedrijfsApp` heet
 - De volgende dependencies heeft:
 - Spring Web
 - MySQL Driver
 - Spring Data JPA
 - Spring Data JDBC
- Zorg dat je de App geopend hebt in de IDE (hoeft nog niet te draaien)

Gelukt?

- Bij het draaien zie je dat er een fout is, omdat er nog geen DataSource (url) is geconfigureerd (zie console)
- Daarom: applications.properties configureren (onder src/main/resources)



DB verbinding configureren

- In application.properties (copy/paste van centraal bestand):

```
#hier staan de 'inloggegevens' om verbinding te maken met de repository en de database (XAMPP)
spring.datasource.url=jdbc:mysql://localhost:3306/oefeningDB?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=

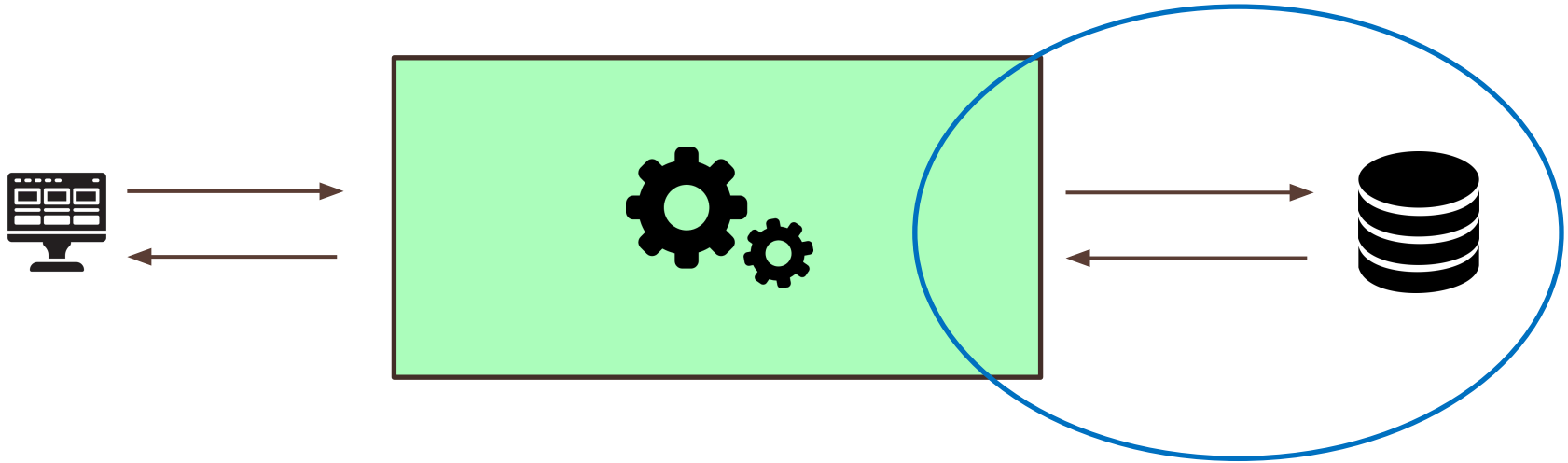
spring.jpa.database_platform=org.hibernate.dialect.MySQL5Dialect

#Zorgt dat de DB leeg is iedere keer als de Back-End opgestart wordt
spring.jpa.hibernate.ddl-auto=create-drop

#bepaald de poort waar de back-end op te bereiken is voor de Front-End
server.port=8082
```

→ Als dat gelukt is → save → en probeer te draaien

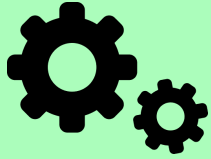
Waar staan we nu?



→ Nu: Back-End raamwerk
verder uitbouwen

- ✓ Repository server en Database + tabel
- ✓ Back-End raamwerk + verbinding Repository

Structuur Back-End



Model Class
"Bedrijf"

Dit wordt de blauwdruk
van de data die door de
Back-End gebruikt wordt

Controller Class
"BedrijfController"

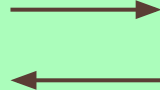
Deze bevat de endpoints
en behandelt het verkeer
tussen Front-End en
Service-Repository

Service Class
"BedrijfService"

Hierin worden alle
binnenkomende verzoeken
behandeld en
uitgespecificeerd,
mutaties uitgevoerd
*'hierin staan alle methods
die echt wat doen'*

Repository Interface
"BedrijfRepository"

Deze behandelt het
verkeer tussen Repository
Server en Back-End



Model maken

- Back-end is running boodschap in BedrijfsAppApplication
- In com.oefening.bedrijfsApp maken we een nieuwe package (bedrijf – kleine letters wegens conventie)
- Daarin: class die Bedrijf heet
 - Dit wordt het model / entity
- Volgende annotations op de twee regels direct boven de class:
- Deze vertellen de applicatie dat deze class een entiteit/model is en dat die gelinkt wordt in de database aan tabel "bedrijf"

```
@Entity
@Table(name =
    "bedrijf")
public class Bedrijf
{
```

Model maken

- In de class Bedrijf maken we de volgende velden:
- Het is een goede gewoonte in Model of Entity voor de velden Wrapper classes te gebruiken (om NullPointerExceptions te voorkomen in combinatie met de Apache Commons Lang3 library)
- Boven het veld id plaatsen we deze Annotations:
- Deze zorgen ervoor dat dit veld het ID veld is, en dat hier een waarde voor wordt aangemaakt bij het instantiatie
- Vervolgens bouwen we de constructor:

```
private long id;  
private String naam;  
private Integer  
aantalWerknemers;
```

```
public Bedrijf(String naam, Integer  
aantalWerknemers) {  
    this.naam = naam;  
    this.aantalWerknemers =  
aantalWerknemers;  
}
```

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

Model maken

- Als laatste: de Getter en Setter methods voor de velden maken
 - Bij IntelliJ Ideas → ALT + Insert → Getter and Setter → velden naam en aantalWerknemers selecteren
- Klaar!

```
package com.oefening.bedrijfsApp.bedrijf;

import javax.persistence.*;

@Entity
@Table(name = "bedrijf")
public class Bedrijf {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String naam;

    private Integer aantalWerknemers;

    public String getNaam() {
        return naam;
    }

    public void setNaam(String naam) {
        this.naam = naam;
    }

    public Integer getAantalWerknemers() {
        return aantalWerknemers;
    }

    public void setAantalWerknemers(Integer
aantalWerknemers) {
        this.aantalWerknemers = aantalWerknemers;
    }

    public Bedrijf(String naam, Integer
aantalWerknemers) {
        this.naam = naam;
        this.aantalWerknemers = aantalWerknemers;
    }
}
```

Repository Interface maken

- In package bedrijf maak je een java interface die BedrijfRepository heet
- Boven de interface zet je de Annotation @Component en je laat de interface de CrudRepository extenden
- Deze interface gaat alle interacties met de Repository in XAMPP afhandelen
- @Component laat zien dat dit een Component van een Spring MVC applicatie is
- De CrudRepository van Spring heeft heel veel functies ingebakken, findById, findAll, etc.
- Achteraan staan de datatypes voor het verkeer tussen Back-End en Repository:
 - Bedrijf objecten en Long (om bedrijven met id nummer te vinden)

```
package com.oefening.bedrijfsApp.bedrijf;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Component;

@Component
public interface BedrijfRepository extends CrudRepository <Bedrijf, Long>{
}
```

Controller maken

- In package bedrijf maak je een java class die BedrijfController heet
- Boven de class zet je devolgende Annotations:

```
@RestController
@CrossOrigin
@RequestMapping("/api/bedrijf")
public class BedrijfController {

}
```

- @CrossOrigin voorkomt allerlei problemen rondom CORS
- @RequestMapping geeft aan dat alle endpoints in de Controller worden aangesproken met een url beginnende met ip-adres/api/bedrijf

Controller maken

- In de class gaan we de service verbinden:

```
@RestController
@CrossOrigin
@RequestMapping("/api/bedrijf")
public class BedrijfController {

    @Autowired
    BedrijfService bedrijfService;
```

- Met @Autowired zorgen we dat de BedrijfService klasse meteen geïnstantieerd wordt bij het oproepen van de Controller
- Endpoints daarmee methods in de service aanroepen

Controller maken

- Maak een testpoint dat alle bedrijven in de DB teruggeeft:

```
@GetMapping("/findAll")
public Iterable<Bedrijf> findAll(){
    System.out.println("findAll endpoint
aangesproken");
    return
    bedrijfService.getAllBedrijf();
}
```

- Vervolgens maken we een endpoint die een testbedrijf maakt:

```
@GetMapping("/test")
public void maakTestBedrijf(){
    bedrijfService.maakTestBedrijf();
    System.out.println("maakTestbedrijf endpoint
aangesproken");
}
```


Service maken

- In package bedrijf maak je een java class die BedrijfService heet
- Boven de class zet je de volgende Annotations en Autowire je de BedrijfRepository:

```
@Service
@Transactional
public class BedrijfService {

    @Autowired
    BedrijfRepository bedrijfRepository;
```

- @Service geeft aan dat dit een service class is in een Spring Boot MVC stijl applicatie
- @Transactional geeft de standaard van transacties weer, en dit voorkomt dat je allerlei extra constructors moet maken in je Model.

Service maken

- Maak de getAllBedrijf method

```
public Iterable<Bedrijf> getAllBedrijf()
{
    System.out.println("in service:
getAllBedrijf");
    return bedrijfRepository.findAll();
}
```

- De functie findAll() ingebakken in de bedrijfRepository retournt een Iterable<Bedrijf>
- Maak de maakTestBedrijf method

```
public Bedrijf maakTestBedrijf() {
    Bedrijf testBedrijf = new
Bedrijf("TestBedrijf", 99999);
    System.out.println("We hebben een
test bedrijf gemaakt in de service");
    return
bedrijfRepository.save(testBedrijf);
}
```

Opdracht: de App testen

- Zorg dat je XAMPP draait
- Zorg dat je Back-End draait

→ Back-End testen

- In een browser, bezoek **localhost:8082/api/bedrijf/findall**
 - We hebben de poort namelijk aangepast in application.properties
- In een browser, bezoek localhost:8082/api/bedrijf/test
- In een browser, bezoek localhost:8082/api/bedrijf/findall
- Bezoek nu via XAMPP phpMyAdmin en bekijk de tabel bedrijf – wat zie je?
- Wat zie je in de console van je IDE staan?

Wat hebben we nu gemaakt?

- ✓ Spring Boot Back-End applicatie die Bedrijven kan aanmaken en opvragen uit een Repository
- ✓ Via een Browser kunnen we de endpoint bevragen en informatie terugkrijgen en/of de data in de repository aanpassen
 - Nu: variabele meegeven in request
 - Requests vanaf de Front-End wat netter simuleren met Postman
 - Back-End voor POST, DELETE, PUT requests

Variabele meegeven aan request 1

- Aan Bedrijf (model) een default constructor (no arguments) toevoegen:

```
public  
Bedrijf(){  
    super();  
}
```

Variabele meegeven aan request 2

- Aan BedrijfController extra endpoint:

```
@GetMapping("/id/{id}")
public Bedrijf
getBedrijfById(@PathVariable(value="id")
long id) {
    System.out.println("In endpoint:
getBedrijfById met ID " + id);
    return bedrijfService.getById(id);
}
```

Hiermee geef je aan dat je in de url een getal zet, wat de variabele long id wordt

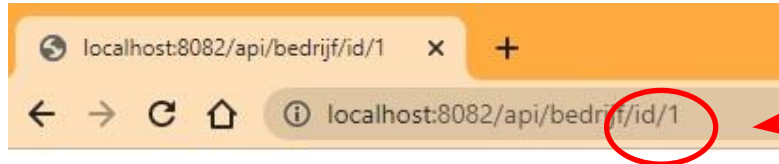
- Aan BedrijfService extra method:

```
public Bedrijf getById(long id) {
    System.out.println("in service:
getById");
    return
    bedrijfRepository.findById(id).get();
}
```

De findById method in de repository retournt een zogenaamde Optional (later meer). Met get() halen we daar het Bedrijf object uit

Variabele meegeven aan request 3

- App herstarten
- Aantal testbedrijven maken (localhost:8082/api/bedrijf/test in browser)
- In de browser een variabele voor id meegeven: localhost:8082/api/bedrijf/id/1
- Opdracht: experimenteer!



Hiermee geef je in url de id variabele mee – het Back-End maakt hier zelf een long van

```
{  
  naam: "TestBedrijf",  
  aantalWerknemers: 99999  
}
```


Variabele meegeven aan request 6

- Daarom: Optional<Bedrijf>
 - Een Optional is een workaround voor null-waarden in deze situatie: bij een null waarde wordt een leeg object gereturt

In BedrijfController:

```
@GetMapping("/id/{id}")
public Optional<Bedrijf>
getBedrijfById(@PathVariable(value="id")
long id) {
    System.out.println("In endpoint:
getBedrijfById met Id " + id);
    return bedrijfService.getById(id);
}
```

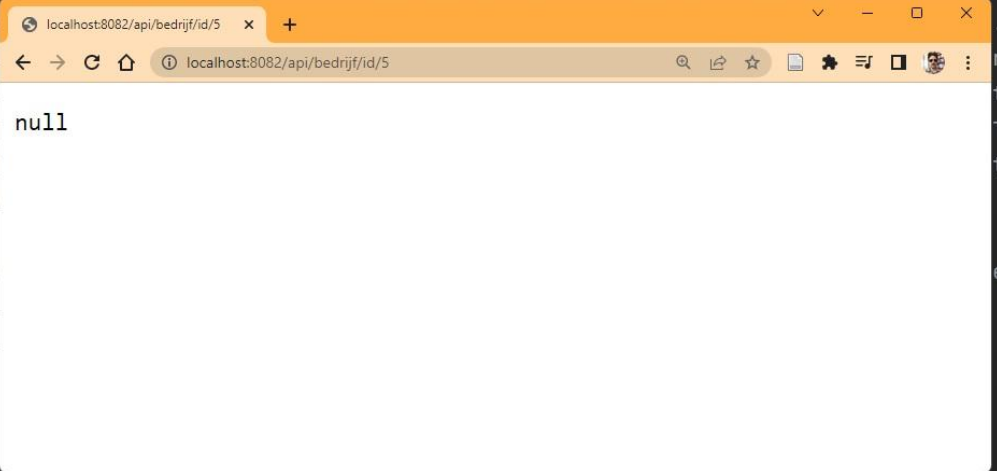
In BedrijfService:

```
public Optional<Bedrijf> getById(long id) {
    System.out.println("in service:
getById");
    return bedrijfRepository.findById(id);
}
```

Variabele meegeven aan request 7

- Met Optional: null is de return waarde bij een niet bestaande id

```
2022-10-31 12:08:24.151 INFO 23760 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-10-31 12:08:24.263 INFO 23760 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-10-31 12:08:24.274 INFO 23760 --- [main] org.hibernate
2022-10-31 12:08:24.679 INFO 23760 --- [main] o.h.e.t.j.p.i
2022-10-31 12:08:24.686 INFO 23760 --- [main] j.LocalContai
2022-10-31 12:08:24.879 WARN 23760 --- [main] JpaBaseConfig
2022-10-31 12:08:25.236 INFO 23760 --- [main] o.s.b.w.embed
2022-10-31 12:08:25.243 INFO 23760 --- [main] c.o.bedrijfsA
Back-end up and running
2022-10-31 12:09:58.887 INFO 23760 --- [nio-8082-exec-1] o.a.c.c.C.[To
2022-10-31 12:09:58.887 INFO 23760 --- [nio-8082-exec-1] o.s.web.servl
2022-10-31 12:09:58.888 INFO 23760 --- [nio-8082-exec-1] o.s.web.servl
In endpoint: getBedrijfById met Id 5
in service: getById
```



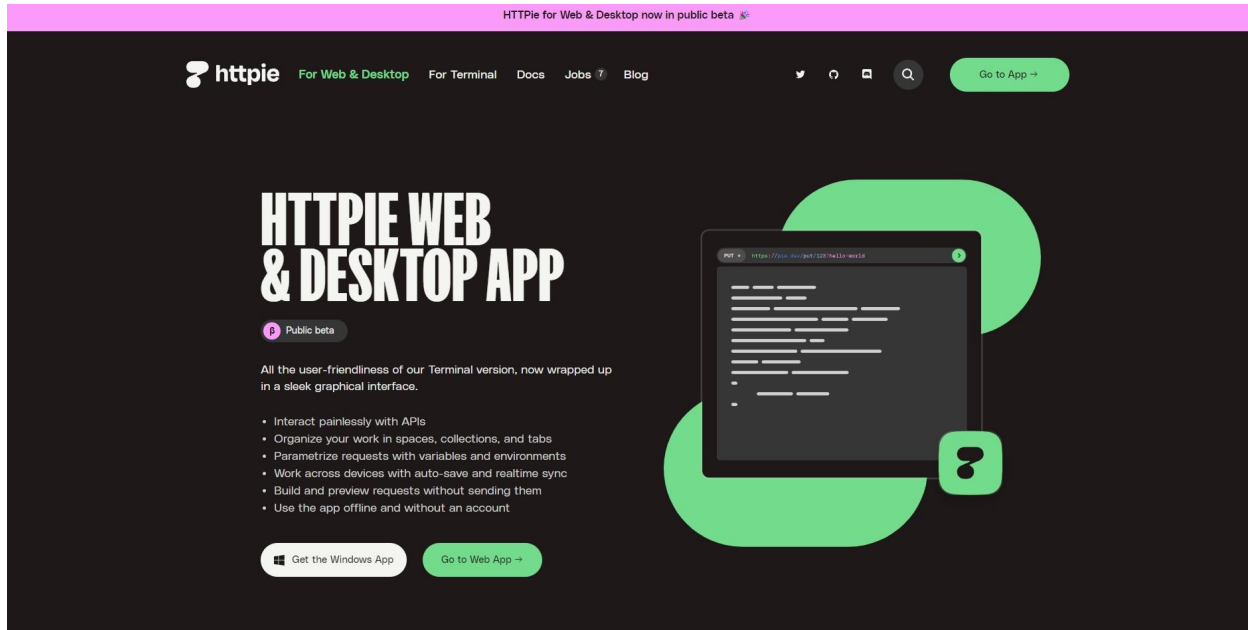
The screenshot shows a web browser window with the address bar displaying 'localhost:8082/api/bedrijf/id/5'. The browser's response area shows the word 'null'. The browser's address bar also shows the full URL 'localhost:8082/api/bedrijf/id/5'.

Simuleren Front-End requests

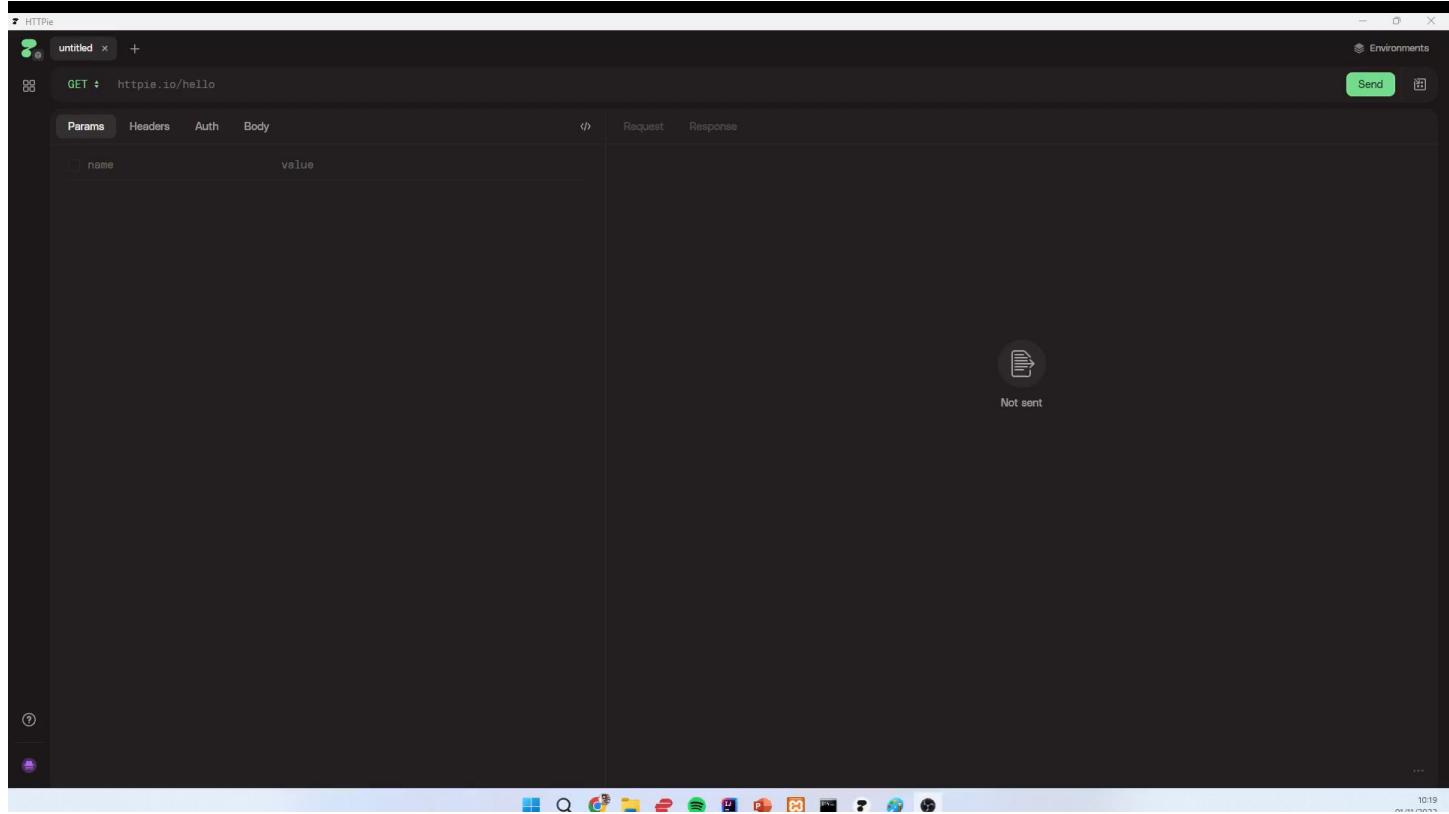
- API testing
- Vanaf een browser werkt, maar complexere requests onhandig
- Daarom: API testing software
 - Postman → moet je een account voor maken (ook voor Free versie)
 - SwaggerUI → Open Source, draait in Node of Docker
 - SoapUI → Dedicated Desktop App + Open Source – soms niet intuïtief
 - **HTTPIe** → **Open Source, redelijk intuïtief, maar Desktop app wel Beta**
 - Hier gekozen voor HTTPIe

HTTPIe

- Installatie → <https://httpie.io/product> → 'Get Windows App' (staan alle platformen onder)



HTTPIe – testen en collectie maken



Naar volledige REST: POST

In BedrijfController:

```
@PostMapping("/new")
public Bedrijf addBedrijf(@RequestBody
Bedrijf bedrijf) {
    System.out.println("In endpoint:
addBedrijf");
    return
bedrijfService.addBedrijf(bedrijf);
}
```

Hier wordt als variabele een body
meegegeven i.p.v. een variabele in de URL

Een body is meestal een stuk JSON tekst

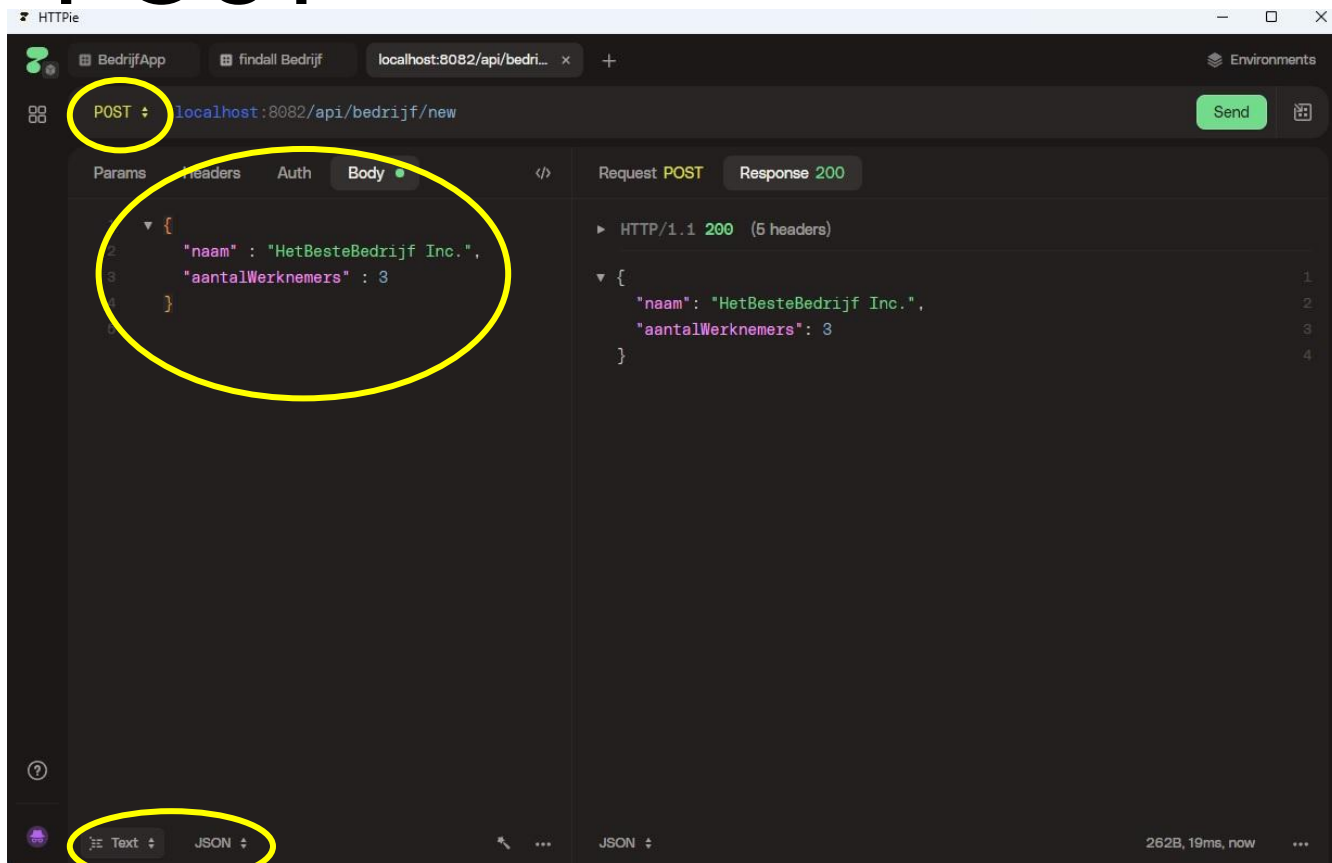
```
{
  "naam" : "LeukBedrijf",
  "aantalWerknemers" : 24
}
```

Een body geef je in de regel enkel mee aan
een endpoint bij POST en PUT requests

- In BedrijfService:

```
public Bedrijf addBedrijf(Bedrijf
bedrijf){
    System.out.println("in service:
addBedrijf");
    return
bedrijfRepository.save(bedrijf);
}
```

POST



- Check dat je een POST verzoek maakt
- Check dat je een Tekst (JSON) als Body meegeeft
- Inhoud Body:

```
{  "naam": "Jouwbedrijf",  "aantalWerknemers": 333}
```

Wat hebben we vandaag behandeld?

- ✓ het MVC model en applicatie architectuur, Fullstack development, en de frameworks Spring en Hibernate
- ✓ opzetten en configureren van een Spring Boot applicatie
- ✓ opzetten van een database server (XAMPP)
- ✓ aansluiten van je applicatie op de db server
- ✓ opzetten van een model
- ✓ toepassen van Annotations
- ✓ opzetten van een repository interface
- ✓ opzetten van een service klasse
- ✓ aanspreken van endpoints via de browser en met API test software
- ✓ bouwen van REST endpoints (GET, POST) in een controller klasse

Opdracht: verder testen

- Probeer een Bedrijf te maken en te vinden
- Configureer nu je eigen Spring Boot Applicatie voor iets dat je tof lijkt (bijv. gameDB) met Spring Initializr
- Bouw een eigen DB met daarin een tabel (bijv. voor game) met velden voor
 - id
 - Naam
 - Etc.
- Sluit je App aan op de DB (application.properties)
- Bouw een eigen Model (bijv. Game), Controller, Service, Repository-Interface in je Back-End
- Maak Endpoints om games op te zoeken en te maken

Vragen?

- E-mail mij op voornaam.achternaam@code-cafe.nl!
- Join de Code-Café community op discord!

