



# Spring Boot en Hibernate

Verder bouwen aan  
de Back-End



# Leerdoelen

- Derived queries
- Simpele View bouwen (Front-End)
- RequestBodies en PathVariables
- REST methodes toepassen op Front-End
  - Database records aanmaken
  - Database records opvragen
  - Database records verwijderen

# Planning

- Derived queries
- View maken met een simpel Front-End
- Bedrijf opvragen en Testbedrijf aanmaken
- Form maken om nieuw bedrijf aan te maken
- Bedrijf opzoeken met Id of Naam
- Bedrijf aanpassen vanaf View
- Bedrijf verwijderen

# De Kracht van Spring JPA – derived queries

- Nog een Get endpoint – in BedrijfController:

```
@GetMapping("/name/{name}")
public Bedrijf getBedrijfByNaam(@PathVariable(value="name") String
naam){
    System.out.println("In endpoint: getBedrijfByName");
    return bedrijfService.getByNaam(naam);
}
```

- In BedrijfService:

```
public Bedrijf getByNaam(String naam) {
    System.out.println("in service: getByNaam");
    return bedrijfRepository.findByName(naam);
}
```

# De Kracht van Spring JPA – derived queries

- Dit draait nog niet! – in de interface BedrijfRepository:

```
@Component
public interface BedrijfRepository extends CrudRepository
<Bedrijf, Long>{
    Bedrijf findByNaam(String naam);
}
```

→ dit mag wel! Spring JPA laat het toe om zo DB queries te maken  
Begint met findBy en dan een naam van een veld uit het model

```
Bedrijf findByAantalWerknemers (Integer aantalWerknemers);
```

# De Kracht van Spring JPA – derived queries

- MAAR: Exceptions als er meerdere bedrijven met dezelfde naam of zelfde aantal medewerkers zijn
  - Oplossing: maak `Iterable<Bedrijf>` van return type

```
@GetMapping("/name/{name}")  
public Iterable<Bedrijf>  
getBedrijfByNaam(@PathVariable(value="name") String  
naam){  
    System.out.println("In endpoint:  
getBedrijfByName");  
    return bedrijfService.getByNaam(naam);  
}
```

In BedrijfController

```
public Iterable<Bedrijf> getByNaam(String naam) {  
    System.out.println("in service: getByNaam");  
    return bedrijfRepository.findByName(naam);  
}
```

In BedrijfService

```
Iterable<Bedrijf> findByName(String naam);
```

In BedrijfRepository

# De Kracht van Spring JPA – derived queries

- `Iterable<E>` → interface die o.a. wordt geïmplementeerd door de `ArrayList` (en alle andere collections)
- Wordt in de Spring architectuur vaak ingezet als een Optional voor lijstjes
  - D.w.z. een `Iterable` kan een lijst met Object 'bevatten'
  - Maar kan en mag ook leeg zijn
    - (en daarom zo nuttig bij Repository communicatie – je weet niet van tevoren of er een object gereturt gaat worden)

```
Iterable<Bedrijf> bedrijfIterable =  
bedrijfRepository.findByAantalWerknemers(Integer aantalWerknemers);  
for(Bedrijf bedrijf : bedrijfIterable){  
    System.out.println("Dit is bedrijf " + bedrijf.getNaam());  
}
```

# De Kracht van Spring JPA – derived queries

- `Iterable<E>` → je kunt het aantal objecten tellen met een for-each loop:

```
int counter = 0;
for (Bedrijf bedrijf : bedrijfIterable){
    counter++;
}
```

- Maar als je gechecked hebt dat de `Iterable` een instance is van een `Collection` → `size()`

```
int counter = 0;
if (bedrijfIterable instanceof Collection){
    counter = ((Collection<?>) bedrijfIterable).size();
}
```

\* Nog veel meer mogelijkheden met Iterables d.m.v. externe libraries, zoals Apache Commons `IterableUtils`

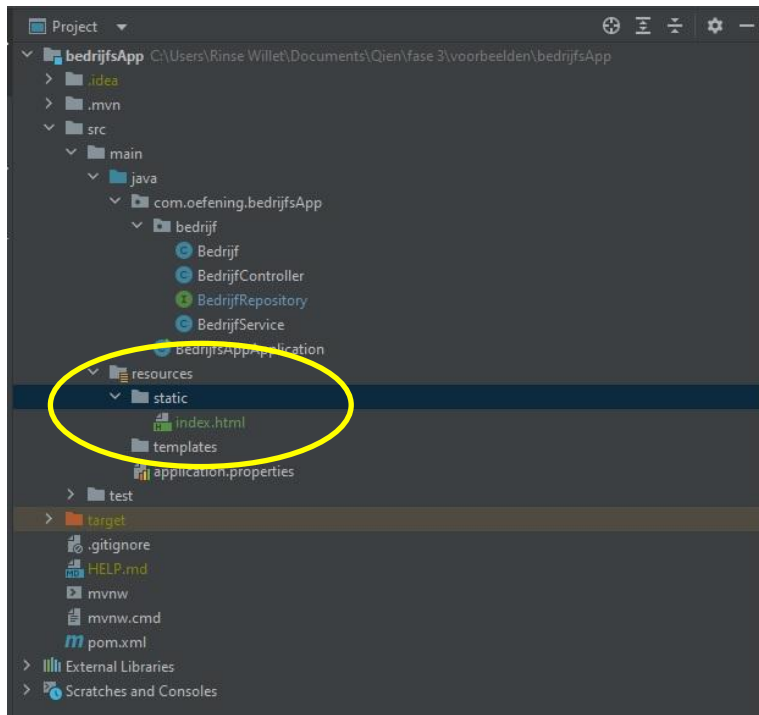


# De Kracht van Spring JPA – derived queries

- Derived queries zijn extra zoekopdrachten die gebruik maken van de CrudRepository in de Repository Interface
  - findBy, countBy, deleteBy, existsBy
- Je kunt ook condities toevoegen:
  - findByNaamIsNot(String naam)
  - findByNaamContaining(String tekst)
  - findByNaamStartingWith(String begin)
  - findByNaamEndingWith(String eind)
  - findByAantalWerknemersLessThan(Integer maximum)
  - findByAantalWerknemersGreaterThan(Integer minimum)
  - findByAantalWerknemersGreaterThanEqual(Integer minimum)
  - findByAantalWerknemersBetween(Integer min, Integer max)
  - findByNaamAndAantalWernemersGreaterThan(String naam, Integer minimum)
  - findByNaamOrderByNaamAsc(String naam)
  - etc. etc.
    - [www.Baeldung.com/spring-data-derived-queries](http://www.Baeldung.com/spring-data-derived-queries)

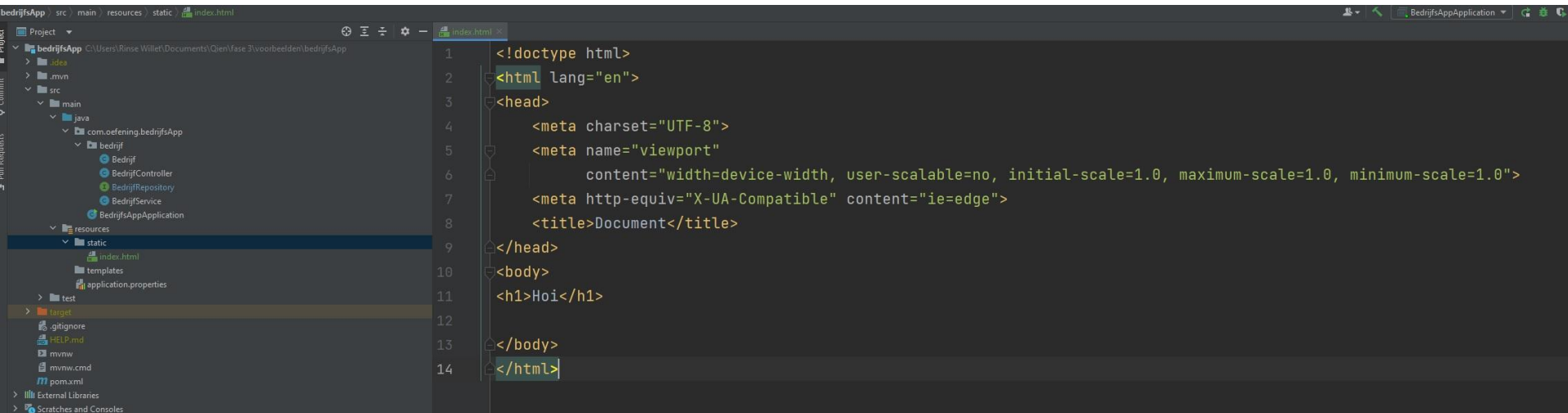
# MVC: View – Front-End

- Maak bestand index.html aan in resources/static



# MVC: View – Front-End

- Simpele inhoud (! + tab)



The screenshot shows an IDE with a project named 'bedrijfsApp'. The file explorer on the left shows the project structure, with 'index.html' selected under 'resources' > 'static'. The main editor displays the content of 'index.html' with line numbers 1 through 14. The code is a simple HTML document with a head section containing meta tags for charset, viewport, and compatibility, and a body section with a single h1 element containing the text 'Hoi'.

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport"
6     content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>Document</title>
9 </head>
10 <body>
11   <h1>Hoi</h1>
12
13 </body>
14 </html>
```

→ opslaan, App draaien en testen in browser → localhost:8082

# Kleine aanpassing controller

- String teruggeven voor Front-End

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

# Index.html

- Maak een knop om een testbedrijf te maken

```
<h1>Welkom bij de BedrijfApp</h1>
<p>Maak met deze knop een testbedrijf aan</p>
<button type="button" onclick="maakTestBedrijf()">Maak
Testbedrijf</button>
<p id="testBedrijf" />
```

# Index.html

- JavaScript (Ajax XMLHttpRequest) om endpoint aan te spreken

```
<script>
  function maakTestBedrijf(){
    const xhttp = new XMLHttpRequest();
    xhttp.onload = function(){
      document.getElementById("testBedrijf").innerHTML =
this.responseText;
    }

    xhttp.open("GET", "http://localhost:8082/api/bedrijf/test");
    xhttp.send()
  }
</script>
```

Let op:  
http:// voor het  
url, anders krijg  
je CORS  
problemen

→ Opslaan, app draaien en testen via localhost:8082

# Index.html

- Nu verder: alle Bedrijven opvragen:

```
<p>Wil je weten welke bedrijven er in de Database  
zijn? Druk op de knop!</p>  
<button type="button" onclick="findAllBedrijf()">Alle  
bedrijven</button>  
<p id="resultaat" />
```

```
function findAllBedrijf(){  
    const xhttp = new XMLHttpRequest();  
    xhttp.onload = function(){  
        document.getElementById("resultaat").innerHTML =  
this.responseText;  
    }  
  
    xhttp.open("GET", "http://localhost:8082/api/bedrijf/f  
indall");  
    xhttp.send()  
}
```

# Je eerste MVC!

→ Opslaan, draaien, testen via localhost:8082 in browser



## Welkom bij de BedrijfApp

Maak met deze knop een testbedrijf aan

Maak Testbedrijf

Testbedrijf aangemaakt

Wil je weten welke bedrijven er in de Database zijn? Druk op de knop!

Alle bedrijven

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

Je hebt nu de basis van het Model, de Controller en de View gemaakt!



# Alle bedrijven in een tabel

- Extra knop toevoegen:

```
<p>Wil je alle bedrijven uit de DB in een tabel? Druk op de knop!</p>
<button type="button" onclick="showTable()">Alle bedrijven</button>
<table>
  <tr>
    <th>Naam</th>
    <th>aantal werknemers</th>
  </tr>
  <tbody id="tabel" />
</table>
```

# Alle bedrijven in een tabel

- Extra functie aan script – showTable():

```
const showTable = () => {  
  const xhttp = new XMLHttpRequest();  
  xhttp.onload = function () {  
    let data = JSON.parse(this.responseText);  
    buildTable(data);  
  }  
  xhttp.open("GET", "http://localhost:8082/api/bedrijf/findall");  
  xhttp.send();  
}
```

Deze toevoeging knipt de objecten uit de JSON-string  
(*parsing*) in losse objecten  
Vervolgens wordt met die opgeknipte data de buildTable  
functie aangeroepen

# Alle bedrijven in een tabel

- Nog een extra functie aan script – buildTable(data):

```
const buildTable = (data) => {  
  var table = document.getElementById("tabel");  
  table.innerHTML = "";  
  for (var i = 0; i < data.length; i++) {  
    var row = `  
      <td>${data[i].naam}</td>  
      <td>${data[i].aantalWerknemers}</td>  
    </tr>`  
    table.innerHTML += row;  
  }  
}
```

Deze functie itereert over de objecten in data en construeert iedere keer een tabel-rij (LET-OP: back-ticks!)

→vervolgens worden de rijen als html toegevoegd aan tbody "tabel"

# Opdracht: experimenteer met HTTPie en kijk of nieuwe / aangepaste bedrijven in de tabel verschijnen



## Welkom bij de BedrijfApp

Maak met deze knop een testbedrijf aan

Maak Testbedrijf

Testbedrijf aangemaakt

Wil je weten welke bedrijven er in de Database zijn? Druk op de knop!

Alle bedrijven

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

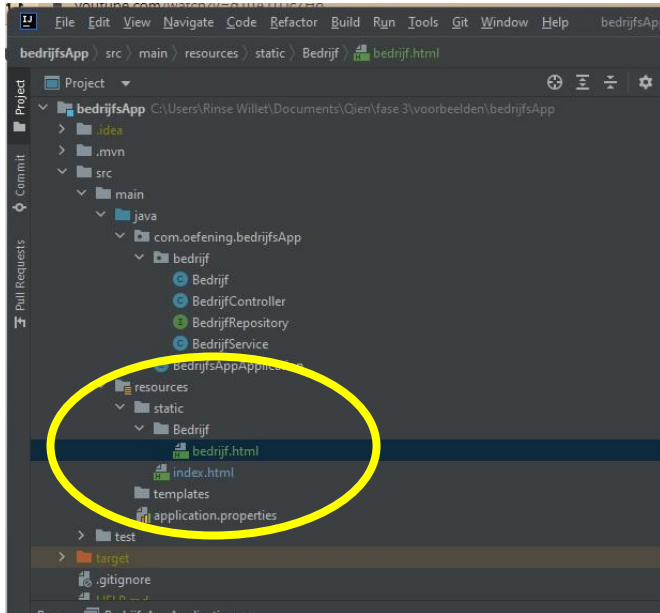
Wil je alle bedrijven uit de DB in een tabel? Druk op de knop!

Alle bedrijven

Naam	aantal werknemers
TestBedrijf	99999
TestBedrijf	99999
TestBedrijf	99999

# Bedrijven toevoegen – simpel Formulier

- Extra map bedrijf en bestand bedrijf.html toevoegen aan resources/static



Link toevoegen aan index.html

```
<a href="/bedrijf/bedrijf.html"> Voeg een bedrijf toe</a>
```

# Bedrijf.html

- Boilerplate html (! + tab) en simpele opzet met titel, form tags en link terug naar index.html

```
<body>
  <h2>Voeg een nieuw bedrijf toe</h2>

  <form>

  </form>

  <a href="/../index.html">Terug</a>
</body>
</html>
```


# Bedrijf.html

- Tussen de form tags maken we de labels en input velden en submit en reset knoppen

```
<label for="naam">Naam Bedrijf</label><br>
<input type="text" id="naam" name="naam"><br>
<label for="werknemers">Aantal werknemers</label><br>
<input type="number" id="werknemers" name="werknemers"><br>
<input type="submit" value="Indienen">
<input type="reset" value="opnieuw">
```

# Bedrijf.html

- Eerste form tag aanpassen: bij onsubmit – functie newBedrijf() aanroepen

```
<h2>Voeg een nieuw bedrijf toe</h2>  
<form onsubmit="newBedrijf()">   
  <label for="naam">Naam Bedrijf</label><br>
```



# Bedrijf.html

- newBedrijf() toevoegen tussen script tags

```
<script>
  const newBedrijf = () => {
    var data = {};
    data.naam = document.getElementById("naam").value;
    data.aantalwerknemers = document.getElementById("werknemers").value;
    objJSON = JSON.stringify(data);
  }
</script>
```

→ functie leest de waarden uit de inputvelden uit (id="naam" en id="werknemers") en stopt die in de variabele data als JavaScript Object  
→ JavaScript Object Notation AKA JSON!

# Bedrijf.html

- regels toevoegen aan newBedrijf() om te posten

```
var xhr = new XMLHttpRequest();  
xhr.onreadystatechange = function(){  
    console.log("newBedrijf");  
}  
xhr.open("POST", "http://localhost:8082/api/bedrijf/new", true);  
xhr.setRequestHeader("Content-type", "application/json")  
xhr.send(objJSON);
```

- opent een POST request aan url /api/bedrijf/new
- geeft in de Header aan dat het request van het type application/json is
- de laatste regel geeft aan dat het objJSON wordt meegegeven en verstuurt
- dit klopt ook, want het endpoint verwacht een @RequestBody op de Back-End d.w.z. een object

# Bedrijf.html – opzoeken

- Extra regels om een bedrijf met het ID (findById) op te zoeken (na </form>)

```
<h2>Zoek bedrijf op</h2>
<label for="id">Id Bedrijf</label><br>
<input type="number" id="id" name="id">
<button type="button" onclick="findById()">Opzoeken</button>
<p id="zoekresultaat" />
```

# Bedrijf.html – opzoeken

- Extra functie om resultaat op te zoeken:

```
const findBedrijfById = () => {  
  var id = document.getElementById("id").value;  
  
  const xhttp = new XMLHttpRequest();  
  xhttp.onload = function () {  
    document.getElementById("zoekresultaat").innerHTML = this.responseText;  
  }  
  xhttp.open("GET", "http://localhost:8082/api/bedrijf/id/" + id);  
  xhttp.send()  
}
```

→ hier wordt de variabele id uit de invoer aan het url van het endpoint geconcateneert

→ dit klopt ook, want het endpoint op de Back-End verwacht een @PathVariable d.w.z. er wordt een variabele uit het url-path gelezen

# Opdracht

- Test de zoekfunctie uit
- Maak een eigen zoekfunctie die op Naam zoekt ipv Id
  - `@PathVariable` of `@RequestBody` op endpoint?
  - Input type is een text

# Update en View

- Moeten een PutRequest kunnen doen – dit kan al met HTTPie
- MAAR: Gebruiker moet kunnen zien wat aangepast wordt
- Eerst zorgen dat het aan te passen Bedrijf opgezocht wordt
  - Bedrijf.html aanpassen dat op naam gezocht kan worden
  - findBedrijf() aanpassen dat herkend wordt of op Id of Naam gezocht wordt
  - zoekresultaten retourneren in <p>

# Bedrijf.html – vervolg – updaten

- Uitbreiding om ook op naam te zoeken:

```
<h2>Zoek bedrijf op</h2>
<label for="id">Id Bedrijf</label><br>
<input type="number" id="id" name="id"><br>
<label for="naamSearch">Naam Bedrijf</label><br>
<input type="text" id="naamSearch" name="naamSearch">
<button type="button" onclick="findBedrijf()">Opzoeken</button>
<p id="zoekresultaat"></p>
```

# Bedrijf.html – vervolg – updaten

- Uitbreiding findBedrijf() – variabelen aanmaken:

```
const findBedrijf = () => {  
  var id = document.getElementById("id").value;  
  var naam = document.getElementById("naamSearch").value;  
  var url = "http://localhost:8082/api/bedrijf/";  
}
```



# Bedrijf.html – vervolg – updaten

- Toevoegen If statements om te checken welk inputveld is ingevoerd:

```
if (naam === "" && id === "") {  
    document.getElementById("zoekresultaat").innerHTML = "Voer een zoekterm in";  
} else {  
    if (id !== "") {  
        url += "id/" + id;  
    } else {  
        url += "name/" + naam  
    }  
}
```

# Bedrijf.html – vervolg – updaten

- Toevoegen GET request binnen eerste If statement

```
if (naam === "" && id === "") {
    document.getElementById("zoekresultaat").innerHTML = "Voer een zoekterm in";
} else {
    if (id !== "") {
        url += "id/" + id;
    } else {
        url += "name/" + naam
    }
    const xhttp = new XMLHttpRequest();
    xhttp.onload = function () {
        document.getElementById("zoekresultaat").innerHTML = this.responseText;
    }
    xhttp.open("GET", url);
    xhttp.send();
}
```

# Opdracht

- Nog eens testen – komen de zoekresultaten binnen?
- Probleem:
  - findById endpoint geeft een Optional<Bedrijf> terug,
  - findByName endpoint geeft een Iterable<Bedrijf> terug (oftewel – mogelijk meerdere resultaten)
  - Optional geeft een responsetekst JSON,
  - Iterable geeft een responsetekst in Array (JSON Array)

# Zoekresponse in formulier

- Formulier voor het uitlezen en updaten maken in html
- Check maken waarop gezocht wordt

# Bedrijf.html – eerst: formulier maken

- Update form maken onder invoer zoeken
  - `<p id="zoekresultaat"></p>` wordt vervangen

```
<form onsubmit="updateBedrijf()">
  <label for="idUpdate">Id Bedrijf</label><br>
  <input type="number" id="idUpdate" name="idUpdate" readonly><br>
  <label for="naamUpdate">Naam Bedrijf</label><br>
  <input type="text" id="naamUpdate" name="naamUpdate"><br>
  <label for="werknemersUpdate">Aantal werknemers</label><br>
  <input type="number" id="werknemersUpdate" name="werknemersUpdate"><br>
  <input type="submit" value="Update">
</form>
```

In dit inputveld kan een waarde staan, maar die is readonly (mag je dus niet aanpassen)

# boodschap teruggeven

- In `findBedrijf()` zorgen dat zoek boodschap in het formulier veld `naamUpdate` verschijnt

```
if (naam === "" && id === "") {  
    document.getElementById("naamUpdate").value = "Voer een zoekterm in";  
    document.getElementById("werknemersUpdate").value = "";  
    document.getElementById("idUpdate").value = "";  
}
```

- `.value` in plaats van `.innerHTML` → zorgt ervoor dat de waarde van het veld wordt aangepast (dus niet de HTML)

# Response laten parsen

- Response laten parsen als JSON en lege variabele maken

```
xhttp.onload = function () {  
    let data = JSON.parse(this.responseText);  
    let updateValues;  
  
}
```

# check JSON Array

Als response data een Array is (maar slechts 1 element bevat) → wijs het eerste ( $= [0]$ ) element uit de Array aan de lege variabele toe

- Check maken om te zien of er een JSON Array teruggegeven is:

```
if (Array.isArray(data) && data.length > 1) {  
    document.getElementById("naamUpdate").value = "Meer bedrijven met deze naam";  
    document.getElementById("werknemersUpdate").value = "";  
    document.getElementById("idUpdate").value = "";  
} else if (Array.isArray(data)) {  
    updateValues = data[0];  
}
```

Als response data een Array is ( $\text{Array.isArray(data)}$ ) EN deze array heeft meer elementen dan 1 ( $\text{data.length} > 1$ ) → meerdere bedrijven met dezelfde naam → boodschap naar formulier



# check JSON

- Nog een laatste clause als er geen JSON Array maar een JSON is teruggegeven:

```
} else if (Array.isArray(data)) {  
  updateValues = data[0];  
} else {  
  updateValues = data;  
}
```

→ JSON direct toewijzen aan lege variabele updateValues

# Zoekwaarden aan formulier geven

- Als updateValues niet leeg is:

```
if (updateValues) {  
    document.getElementById("idUpdate").value = updateValues["id"];  
    document.getElementById("naamUpdate").value = updateValues["naam"];  
    document.getElementById("werknemersUpdate").value = updateValues["aantalWerknemers"];  
}
```

→ waarden aan velden formulier toewijzen

```

const findBedrijf = () => {
    var id = document.getElementById("id").value;
    var naam = document.getElementById("naamSearch").value;
    var url = "http://localhost:8082/api/bedrijf/";

    if (naam === "" && id === "") {
        document.getElementById("naamUpdate").value = "Voer een zoekterm in";
        document.getElementById("werknemersUpdate").value = "";
        document.getElementById("idUpdate").value = "";
    } else {
        if (id !== "") {
            url += "id/" + id;
        } else {
            url += "name/" + naam
        }
        const xhttp = new XMLHttpRequest();
        xhttp.onload = function () {
            let data = JSON.parse(this.responseText);
            let updateValues;

            if (Array.isArray(data) && data.length > 1) {
                document.getElementById("naamUpdate").value = "Meer bedrijven met deze naam";
                document.getElementById("werknemersUpdate").value = "";
                document.getElementById("idUpdate").value = "";
            } else if (Array.isArray(data)) {
                updateValues = data[0];
            } else {
                updateValues = data;
            }
            if (updateValues) {
                document.getElementById("idUpdate").value = updateValues["id"];
                document.getElementById("naamUpdate").value = updateValues["naam"];
                document.getElementById("werknemersUpdate").value = updateValues["aantalWerknemers"];
            }
        }
        xhttp.open("GET", url);
        xhttp.send();
    }
}

```

# Opdracht

- Test de zoekfunctie verder uit
- `findBedrijf()` functie aardig lang -> op te splitsen meerdere functies?
- Werkt het? Denk na over de te maken functie `updateBedrijf()`
  - `@PathVariable` of `@RequestBody` op endpoint?
  - Checks als er bij ongeluk op de knop wordt gedrukt
    - Waarde van `naamUpdate` is dan "" of "Voer een zoekterm in"

# updateBedrijf()

- Eerste check voor ontbrekende waarden

```
const updateBedrijf = () => {  
    var id = document.getElementById("idUpdate").value;  
    var naam = document.getElementById("naamUpdate").value;  
    var aantalWerknemers = document.getElementById("werknemersUpdate").value;  
  
    if (id === "" || naam === "" || aantalWerknemers === "") {  
        console.log("geen waardes ingevuld");  
    } else {  
  
    }  
}
```

# updateBedrijf()

- Als id, naam, en aantalWerknemers wel een waarde bevatten: PUT request maken

```
else {  
    var data = {};  
    data.aantalWerknemers = aantalWerknemers;  
    data.naam = naam;  
    objJSON = JSON.stringify(data);  
  
    const xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function () {  
        console.log("newBedrijf");  
    }  
    xhttp.open("PUT", "http://localhost:8082/api/bedrijf/update/" + id, true);  
    xhttp.setRequestHeader("Content-type", "application/json")  
    xhttp.send(objJSON);  
}
```

# Wat heb je nu gemaakt?

- Endpoints voor GET, POST, PUT requests gemaakt voor Bedrijf (Model) in Back-End (Controller)
- View → je hebt een Front-End gemaakt die communiceert met deze endpoints
- Data uitlezen, aanmaken, en aanpassen
- Opdracht: maak een DELETE functie voor je View in Bedrijf.HTML

# Verwijderen

- Input veld en knop

```
<h2>Bedrijf verwijderen</h2>  
<label for="idDelete">Id Bedrijf</label><br>  
<input type="number" id="idDelete" name="idDelete"><br>  
<button type="button" onclick="deleteBedrijf()">Verwijderen</button><br>
```



# Verwijderen

- Functie aan script toevoegen

```
const deleteBedrijf = () => {  
  var id = document.getElementById("idDelete").value;  
  if(id === ""){  
    console.log("geen waarde ingevuld");  
  } else {  
    const xhttp = new XMLHttpRequest();  
    xhttp.onload = function () {  
      console.log("Bedrijf verwijderd");  
    }  
    xhttp.open("DELETE", "http://localhost:8082/api/bedrijf/delete/" + id);  
    xhttp.send();  
  }  
}
```

# 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

# Back to the Back-End

# Vragen?

- E-mail mij op [voornaam.achternaam@code-cafe.nl](mailto:voornaam.achternaam@code-cafe.nl)!
- Join de Code-Café community op discord!

