FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Labo Webtechnologieën
Reeks 2: AJAX, REST, Spring

15 + 23 + 30 oktober 2019

1 Inleiding

In de vorige labo's maakten we kennis met HTML5 en Javascript. De combinatie van beiden laat toe om complexe client-side webapplicaties te maken. In deze reeks beginnen we met server-side programmeren. Er bestaan verschillende technologieën (binnen verschillende programmeertalen) om dit te doen maar binnen Java zijn Servlets nog altijd de basiscomponenten.

Servlets laten toe om code aan de server kant uit te voeren. Deze genereert dan uitvoer die een webbrowser rechtstreeks kan weergeven (HTML uitvoer) of eerst nog zal verwerken (JSON, XML, ... uitvoer).

Een servlet is niets meer dan een standaard Java-klasse (erft over van een Servlet-klasse bv: HttpServlet). In deze klasse wordt er dan een methode overschreven die een request kan afhandelen. De servlets draaien binnen een webcontainer die verantwoordelijk is voor het aanmaken van de servlet-objecten en voor het toewijzen van verzoeken aan deze objecten. In dit labo gebruiken we Apache Tomcat als webcontainer.

Servlets bieden dus een vrij low level manier aan om webapplicaties te bouwen. Manueel een html pagina opbouwen binnen een Servlet is een achterhaalde manier van werken. Tegenwoordig zijn er betere oplossingen zoals het Spring framework in Java. Deze gebruikt wel nog steeds Servlets achter de schermen, maar biedt een productievere omgeving aan de developer.

In deze reeks implementeren we een blog met een achterliggend Content Management System. De beheerder van de blog zal eenvoudig met een webformulier nieuwe artikels kunnen toevoegen aan de blog zonder dat er iets moet veranderen aan de broncode. Al deze functionaliteit wordt geïmplementeerd via een REST api. Vanuit Javascript kunnen we aan de hand van Ajax requests communiceren met de backend. Hiervoor gebruiken we de nieuwe "fetch" api.

2 Spring Framework

Spring is een open-source, lightweight container en framework voor het bouwen van Java enterprise applicaties. Dit framework zorgt ervoor dat je als developer kan focussen op het probleem

dat je wil oplossen met een zo min mogelijk aan configuratie. Het Spring ecosystem bestaat uit verschillende projecten (Zie http://www.spring.io/projects) In dit labo maken we gebruik van Spring Boot, Spring Web (MVC) and Spring Security.

3 Nieuw project

- Maak een nieuw project aan met Spring Initializr en open het in je IDE (Je kan Netbeans of InteliJ gebruiken). https://start.spring.io
  - Voeg bij dependency "Spring Web"toe ("Spring Security" voegen we later manueel toe).
  - Optioneel: update project metadata.
- Voeg de static files ("index.html", "admin.html", "posts.json", "index.js" vanop Ufora toe aan dit project onder "/src/main/resources/static".
- Start de server en bekijk de logs. Als alles goed ging, heeft onze applicatie een Tomcat Server op poort 8080 gestart. http://localhost:8080

4 Ophalen van blogposts met AJAX

De blogposts mogen niet meer hardgecodeerd staan in de html broncode maar worden dynamisch opgehaald via AJAX.

- We hebben de server side logica om de blogposts op te halen nog niet geschreven, voorlopig werken we dus met dummy data. Het bestand "posts.json" bevat meerdere blogposts.
- Schrijf de javascript code die bij het laden van index.html een AJAX request doet naar dit bestand. Gebruik de developer tools om te debuggen. Maak gebruik van de "fetch" api.
- Verwijder alle hardgecodeerde blogposts uit je html bestand en zorg ervoor dat de data uit de json getoond wordt. Je kan handig gebruik maken van de ES6 template strings: http://wesbos.com/template-strings-html/

5 Toevoegen van blogposts met AJAX

- Implementeer de javascript code om een nieuwe blogpost via AJAX door te geven aan de backend als de gebruiker het formulier indient.
- Voorlopig hebben we nog geen backend. Gebruik daarom https://beeceptor.com als dummy backend.
- Je kan ook de network monitor in de development tools gebruiken om te zien welke data doorgestuurd wordt.

6 REST api

Alle blogposts worden op de server bijgehouden. Nieuwe posts aanmaken of bestaande posts opvragen verloopt via een REST api. Spring laat ons toe om eenvoudig een REST api aan te maken zonder dat we zelf moeten instaan voor de conversie van en naar JSON.

- Voorlopig focussen we op de REST api en maken we geen gebruik van een externe database. Alle blogposts mogen voorlopig in memory bijgehouden worden.
- Maak een nieuwe java klasse die een **blogpost** voorstelt. Zorg ervoor dat deze klasse een default constructor heeft en getters en setters voor alle attributen.
- Maak een **blogpost DAO** klasse die de blogberichten bijhoudt, deze klasse simuleert een verbinding met een databank. Hou een aantal BlogPosts bij in het geheugen en voorzie methodes om posts toe te voegen, te verwijderen en op te vragen. Annoteer deze klasse met **@Service**, dit zorgt ervoor dat Spring een bean aanmaakt voor deze klasse en kan gebruikt worden voor dependency injection.
- Maak een **Controller** klasse die de http requests zal afhandelen.
  - Voeg de annotatie **@RestController** toe aan deze klasse (Zie https://spring.io/guides/gs/rest-service/.
  - Injecteer de BlogpostDAO met dependency injection. Hiervoor maak je een constructor aan die een parameter van het type "BlogPostDao" verwacht. Spring zorgt er dan voor dat je bij constructie de dao binnen krijgt.
  - Alle data wordt uitgewisseld als JSON.
- Implementeer volgende REST functionaliteit in de **Controller**, voor elk endpoint van je API maak je een functie die je annoteert met de juiste annotatie (**"@GetMapping", "@DeleteMapping", "@PutMapping", "@PostMapping"**).
  - Alle blogberichten ophalen
  - Eén specifiek blogbericht ophalen. Indien een onbestaande blogpost wordt opgehaald moet er een IllegalArgumentException gegooid worden. Deze exceptie wordt dan opgevangen door een handler en resulteert in een 404 not found http status. Maak gebruik van **"@PathVariable"** om een dele van de url op te vragen.
  - Een bericht toevoegen. Maak gebruik van **"@RequestBody"** om toegang te krijgen tot de meegstuurde data. het http antwoord bevat de locatie header en http status 201: created (zie https://howtodoinjava.com/spring-boot2/rest/rest-api-example/).

```
URI location = ServletUriComponentsBuilder
        .fromCurrentRequestUri()
        .path("/{id}")
        .buildAndExpand(post.getId())
        .toUri();
return ResponseEntity.created(location).build();
```

  - Een bericht verwijderen, resulteert in een 204 http status.
  - Een bericht aanpassen, resulteert in een 204 http status.

7 Testen van de REST api

De REST api kan manueel getest worden met de client uit het eerste deel van dit labo, een cli-tool (ex. curl), een gui (ex. postman) of met geautomatiseerde tests. Vanuit Java code kan je gebruikmaken van de klasse "(Test)RestTemplate". Deze laat ons toe om eenvoudig een REST api aan te spreken.

- Open het bestand in de test folder. Deze klasse heeft de annotatie "@SpringBootTest". Voor elke test maak je een methode aan die je annoteert met "@Test". Binnen deze methodes gebruik je de RestTemplate klasse om te communiceren met de REST api. De resultaten kan je valideren met een "Assert" methode. Zie ook https://www.baeldung.com/rest-template.
- Implementeer een test voor elke CRUD operatie van onze api.

8 Data laag

Nu hebben we een volledig functionele blog geïmplementeerd. We zijn begonnen met hard gecodeerde blog posts in de HTML code, daarna deden we fetch naar een statisch JSON bestand op onze server en tot slot stuurden we REST calls naar de api zodat de blog posts dynamisch konden gebruikt worden. De REST api houdt nu alle blog posts bij in het geheugen. Meer realistisch is het opslaan van de data in een database. Voor deze opdracht maken we gebruik van H2 embedded database. https://www.baeldung.com/spring-boot-h2-database

- Voeg de dependency voor de H2 database toe aan de dependencies voor dit project.
- Bij de interactie met de database maakt het Spring framework ons het leven veel makkelijker. We moeten enkel de data klasse correct annoteren en een interface aanmaken die zelf overerft van "JpaRepository<T,D>". Spring zal dan een bean aanmaken waarmee we de database kunnen aanspreken. Op https://spring.io/guides/gs/accessing-data-jpa/ vind je een voorbeeld hoe dit moet. In deze code erft de repository over van "CrudRepository<T,D>" maar het principe is het zelfde, "JpaRepository<T,D>" biedt gewoon nog extra functionaliteit aan zoals paginering. De types T en D zijn de types van respectievelijk het data object en zijn id.
- Maak een nieuwe klasse aan die exact dezelfde functionaliteit aanbiedt als onze blogpost DAO, maar nu gebruik maakt van de net aangemaakte repository om de data op te slaan in de databank. Vergeet deze klasse niet te annoteren met **"@Service"**. Om helemaal volgens de regels te programmeren kan je een interface definiëren die geïmplementeerd wordt door beide DAOs. De repository krijg je opnieuw binnen via dependency injection in de constructor.
- De applicatie bevat nu 2 manieren om blogposts op te slaan. Op het moment dat de applicatie gestart wordt, beslissen we welke gebruikt moet worden. Hiervoor maken we gebruik van **profiles**. Maak een profile "test" aan die de blogposts opslaat in het geheugen. Als dit profile niet actief is, maken we gebruik van de database. https://www.baeldung.com/spring-profiles
- Test de applicatie. Verander het profiel en ga na of er nu andere data getoond wordt.

9 Beveiliging

Uiteraard kan beveiling niet ontbreken in onze blog. Het admin-gedeelte mag enkel bereikbaar zijn nadat een admin succesvol is ingelogd.

- Voor de security maken we gebruik van “Spring Security”. Voeg de dependency manueel toe aan “pom.xml” (maven). Door deze dependency toe te voegen worden alle pagina’s default doorgestuurd naar een login-pagina. In de logs vind je een random password voor de default user “user”. Hiermee zou je moeten kunnen inloggen. Zie ook https://www.baeldung.com/spring-security-login.

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

- We voegen 2 hard gecodeerde users toe, met de rol ADMIN en USER. Maak een klasse aan die overerft van “WebSecurityConfigurerAdapter” en annoteer die met “@Configuration”. Vergeet de paswoorden niet the encrypteren.
- In dit zelfde configuratie bestand kunnen we ook configureren welke pagina’s beveiligd moeten worden. Zorg ervoor dat iedereen de index pagina kan bekijken maar dat enkel admins de admin pagina kunnen raadplegen.
- Zorg ervoor dat iedereen een GET kan uitvoeren op de API maar dat enkel admins posts kunnen verwijderen en toevoegen. Maak hiervoor gebruik van beveiliging op de methode zelf. https://www.baeldung.com/spring-security-method-security
- Doordat we de default instellingen nu overschrijven, moeten we ook explicit de login en logout instellen.
- Test uit of de rechten correct gezet zijn voor de user en de admin.

10 Uitbreidingen (optioneel)

Nu hebben we een werkend content management system voor een eenvoudige blog maar er zijn nog veel uitbreidingen mogelijk.

- Zorg voor paginering.
- Implementeer de zoekfunctionaliteit.