whitehorses

member of the human network

WHITEBOOK

JHipster - snel aan de slag met Spring Boot en Angular

Maart 2018

Auteur:
Roger Goossens
INTEGRATIE SPECIALIST

Moderne applicaties bestaan vaak uit een verservicede serverlaag en een client-side UI. De eerste laag biedt een universele interface (meestal REST) naar de front-end aan, terwijl de tweede een reactieve UI (in tegenstelling tot server-side front-ends) aan de gebruiker serveert. Voordelen van een REST-exposed back-end is dat deze hiermee loosely coupled is richting front-end en deze laatste hierdoor makkelijk vervangen kan worden als zich een nieuw hip client framework aanbiedt. Daarnaast - niet geheel onbelangrijk – is het eenvoudig om meerdere UI's (denk bv. aan een Android en een IOS client) te ondersteunen.

Een veel gebruikte combinatie van back-end/front-end is die van Spring Boot en Angular. Beide technologieën vormen de basis van JHipster. JHipster is een genereertool die het mogelijk maakt om in zeer korte tijd een applicatie te genereren op basis van Spring Boot en Angular. In oudere versies van JHipster was dit een zgn. monolithische applicatie. Sinds JHipster versie 3.0.0 is het echter ook mogelijk om een applicatie te genereren die voldoet aan een microservice architectuur. Hierbij wordt veelal gebruikgemaakt van het Spring Cloud ecosysteem.

Om een indruk te geven van de kracht van JHipster: de prestigieuze Whitehorses Developer Derby in 2017 is gewonnen door Team LUHR (Laptop Uit Het Raam). LUHR bouwde de winnende applicatie waarvan de basis gegenereerd was m.b.v. JHipster.

In dit Whitebook proberen we een goed beeld te schetsen van de gegenereerde componenten van JHipster en de mogelijkheden en onmogelijkheden die de tool biedt. Hierbij zullen we een (monolithische) applicatie genereren, waarbij we kijken hoe makkelijk het is om zo'n applicatie uit te breiden met eigen maatwerkcode. De applicatiecode wordt beschikbaar gesteld via GitHub (https://github.com/rphgoossens/whitehorses/tree/master/whitebook_jhipster).

Spring Boot

De back-end code – de REST-services – gegenereerd door JHipster, maakt gebruik van Spring Boot-technologie. Spring Boot is een uitbreiding op het populaire Spring-framework. Belangrijke onderdelen van Spring zijn o.a. Dependency Injection (loose coupling, herbruikbaarheid en testbaarheid), Aspect Oriented Programming (cross-cutting concerns als logging en security) en Spring Templates (o.a. JMS en JDBC code uitschrijven zonder boilerplating).

Een groot nadeel van Spring is dat het initieel een hoop moeite vergt om de basis voor een project neer te zetten. Denk hierbij aan het verzamelen van de juiste dependencies (die met elkaar samen kunnen werken), het instantiëren van allerlei Spring Beans om functionaliteit (zoals templates) te ontsluiten en het configureren van de nodige descriptorfiles (web.xml, persistence.xml, etc.), kortom de boilerplating die uitgeschreven moet worden voordat er überhaupt businesscode geklopt kan worden.

De oplossing die Spring Boot hiervoor biedt, is autoconfiguration (geen beans en descriptors nodig) en starter dependencies (dependency hell is verleden tijd). Met bijzonder weinig regels aan configuratie kan een ontwikkelaar snel aan de slag met het werk dat ertoe doet: het coderen van de businesslogica!

Angular

Angular is een populair client-side javascript framework dat het mogelijk maakt om op een productieve wijze een front-end te bouwen. Angular font-ends zijn per definitie cross-platform. Angular biedt een CLI waarmee componenten gegenereerd kunnen worden. Angular-applicaties zijn modulair opgezet, waarmee het relatief eenvoudig is om de code te doorgronden.

Tenslotte wordt de code achter de Angular-HTML-componenten vaak in Typescript gebouwd. Ook JHipster genereert Typescript. Typescript is een compile-time-safe variant van Javascript, biedt IDE-ondersteuning en zorgt daarmee voor een verhoging van de productiviteit van ontwikkelaars. Kortom genoeg redenen waarom Angular in korte tijd zo populair is geworden.

Microservices

Sinds JHipster versie 3.0.0 is ondersteuning voor microservices toegevoegd. In dit Whitebook zullen we hier niet te diep op ingaan, omdat het anders te groot zou worden. In een notendop wordt er gebruikgemaakt van diverse bewezen Netflix technologieën, ondergebracht in het Spring Cloud-framework, waarbij microservices als Docker containers worden gedeployed. Onderstaand plaatje geeft een overzicht:

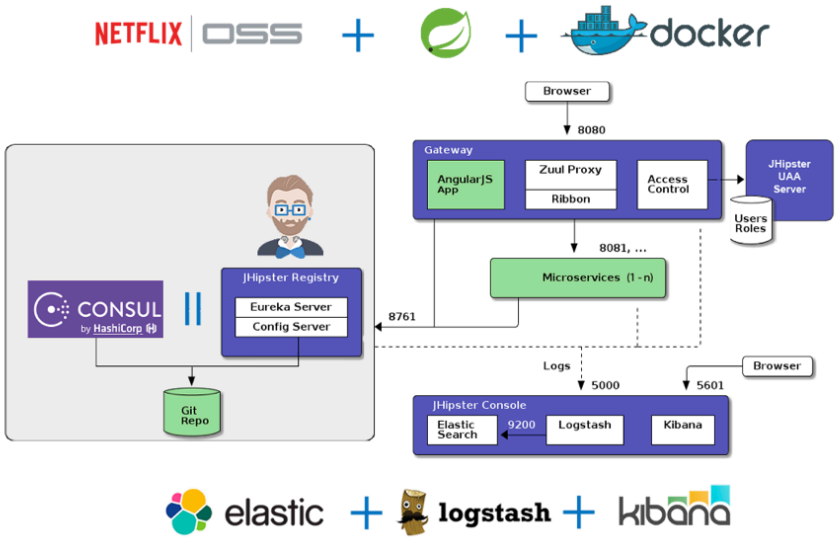

Fig. 1 JHipster-microservices-architectuur

JHipster – een eerste applicatie

Tot zover de theorie. Nu maar eens kijken hoe snel we een werkende applicatie kunnen produceren. Voor de installatie van de noodzakelijke componenten wordt verwezen naar de JHipster-homepage. In dit Whitebook gebruiken we JHipster versie 4.14.1.

In plaats van de gebruikelijke dept-emp applicatie, bouwen we voor dit Whitebook een brewery-beer applicatie. De applicatie bestaat dus in de basis uit 2 domeinobjecten. Het datamodel is ook de basis waarop onze JHipster-applicatie gegenereerd wordt. JHipsters CLI bevat een optie om zgn. entiteiten te creëren, maar het makkelijkst is het om gebruik te maken van JDL Studio, een online tool, waarmee via een eenvoudige syntax een datamodel gecreëerd kan worden. Zie hier het datamodel voor onze applicatie:

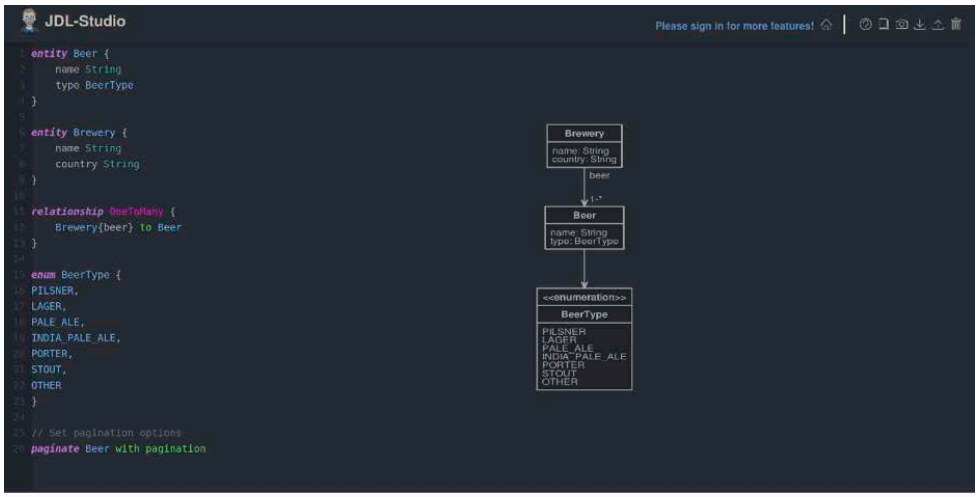

Fig. 2 JDL Studio

Het datamodel kan d.m.v. JDL Studio gedownload worden (dit is een textfile met de jh-extensie, waarbij de inhoud gelijk is aan de tekst aan de linkerkant van bovenstaand plaatje). De JDL-file gaan we later gebruiken. Allereerst genereren we een basis JHipster-applicatie. Gebruik hiervoor het onderstaande CLI-commando:

jhipster

De CLI wordt opgestart en stelt een aantal vragen. Als naam van de applicatie kiezen we **jh_app**, als package **nl.whitehorses.jhipster** en als extra taal - naast English - selecteren we Dutch. Bij de rest van de vragen houden we de defaults aan. De generatie duurt vervolgens zo'n 10 seconden. Hierna kan de applicatie meteen worden getest.
De Spring Boot-applicatie kan gestart worden middels

```
./mvnw
```

En de Angular-applicatie middels

yarn start

De kale JHipster-applicatie is nu te benaderen via localhost:9000 (zie hieronder een afbeelding van de applicatie nadat er is ingelogd met admin/admin):

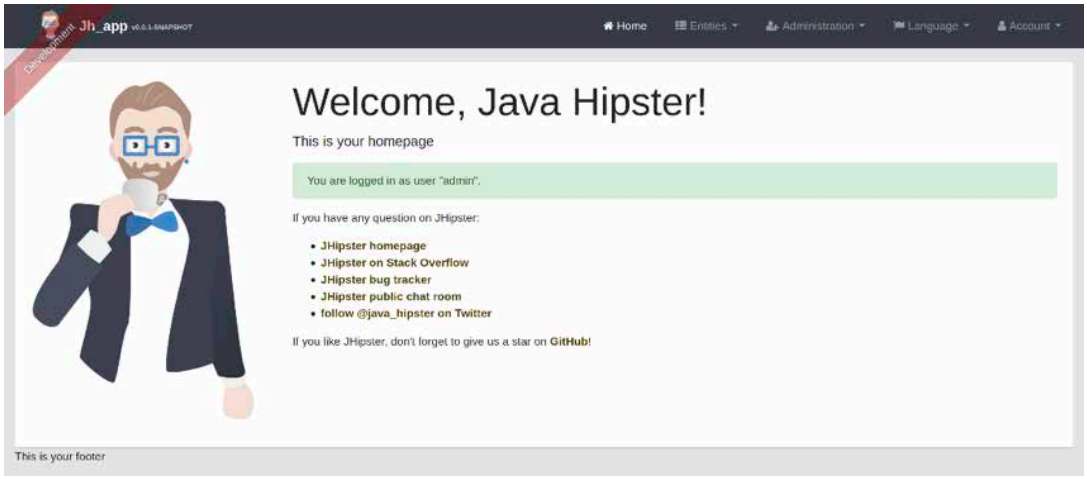

Fig. 3 Homepage

De JHipster-applicatie bevat in dit stadium een homepage, een loginscherm, een menu om de taal te wijzigen, een menu om je account te beheren en een Administration menu. Als onderdeel van dit laatste menu zijn o.a. diverse metrics te raadplegen, alsmede logfiles en een API-pagina waarin Swagger-definities worden getoond voor alle REST-services in de applicatie.

Het Entities menu voor de CRUD-schermen is nog leeg. Laten we de entiteiten toevoegen om daar verandering in aan te brengen (de jhipster-jdl.jh file verwijst naar de file die van JDL Studio is gedownload)(kies bij conflicten altijd voor overschrijven) :

```
jhipster jhipster-jdl.jh
```

Nadat alles weer opnieuw is opgestart, staan er 2 entries in het Entities menu. Hiermee kunnen de Brewery en Beer tabellen beheerd worden. Zie hieronder een screenshot van het Brewery scherm nadat er 2 records zijn toegevoegd:

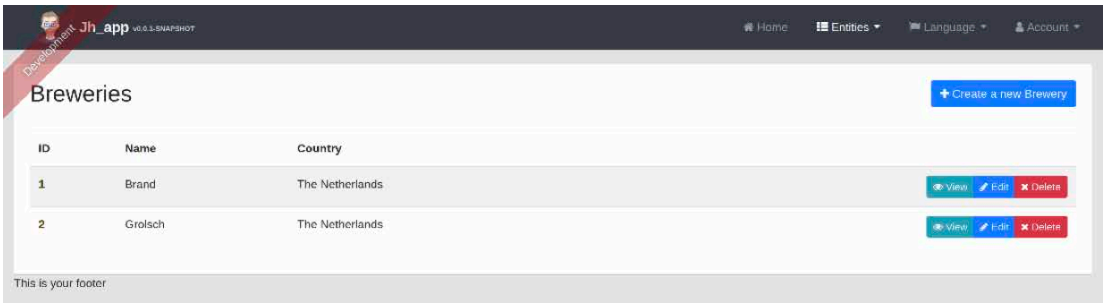

Fig. 4 Breweries scherm

JHipster – structuur

Spring Boot

Onderstaand plaatje geeft een overzicht van de verschillende lagen in de gegenereerde applicatie. De namen komen overeen met de folders waarin de bijbehorende Spring Boot-componenten zijn opgeslagen.

Van onder naar boven:

- domain: deze folder bevat de @Entity-klassen, JPA-klassen die corresponderen met de onderliggende databasetabellen;
- repository: deze folder bevat @Repository-klassen, DAO-klassen die de CRUD-operaties ontsluiten;
- service: deze folder bevat @Services-klassen, serviceklassen hebben een lagere granulariteit dan repositories en worden o.a. gebruikt om complexere operaties met meerdere domainobjecten te ondersteunen;
- service.dto: deze folder bevat zgn. DTO (Data Transfer Object) klassen, objecten welke middels de services worden uitgewisseld met de front-end;
- web.rest: deze folder bevat de @Controller-klassen, de klassen die de REST-operaties ontsluiten.

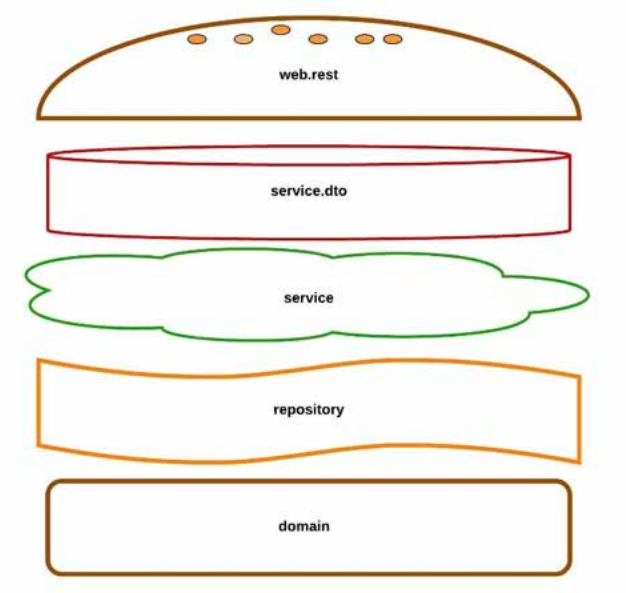

Fig. 5 JHipster-Spring Boot-architectuur

De service en service.dto-lagen worden voor de gegenereerde entities overgeslagen. Hierbij kletst de web.rest-laag direct tegen de repositorylaag aan en worden de domainobjecten als DTO's gebruikt (Als je wel service en DTO klassen wil genereren kun je dit aangeven in de JDL file).

Angular

Ook de Angular-applicatie is gelaagd opgezet. Van onder naar boven:

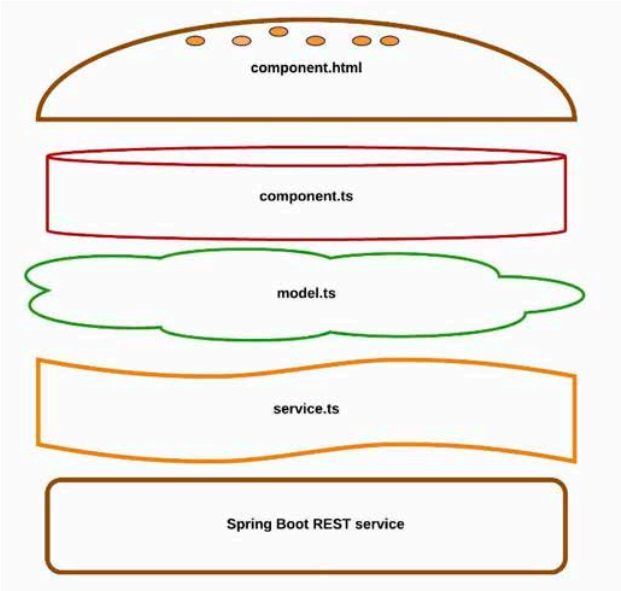

Fig. 6 JHipster-Angular-architectuur

- service.ts: hierin vindt communicatie met de REST-service plaats;
- model.ts: de JSON objecten die middels de REST-services uitgewisseld worden;
- component.ts: de backingcode voor de html-pagina, hierin worden de verschillende service-operaties aangeroepen. Ook bevat deze klasse de data die middels databinding via de UI wordt ontsloten;
- component.html: angular-pagina corresponderend met de overzichtspagina die voor een entiteit getoond wordt.

Andere belangrijke klassen zijn de **module.ts** (elke entitiiteit heeft zijn eigen Angular-module) en de **route.ts** (bevat alle Angular-routes die corresponderen met de menu en knoppen hyperlinks). De route.ts-klassen zorgen er meteen voor dat veel onderdelen van de applicatie gebookmarked kunnen worden (bv. http://localhost:9000/#/beer voor het beer-overzichtscherm).

JHipster – custom uitbreidingen

Toevoegen entity

Het toevoegen van nieuwe entiteiten (en optioneel nieuwe relaties) in JHipster is een fluitje van een cent. Het toevoegen van een entiteit Customer bv. gebeurt middels het volgende CLI-commando:

```
jhipster entity Customer
```

Nadat de vragen die de CLI stelt (o.a. de fieldnamen) zijn doorlopen, wordt de complete CRUD-functionaliteit voor de nieuwe entiteit aan de applicatie toegevoegd:

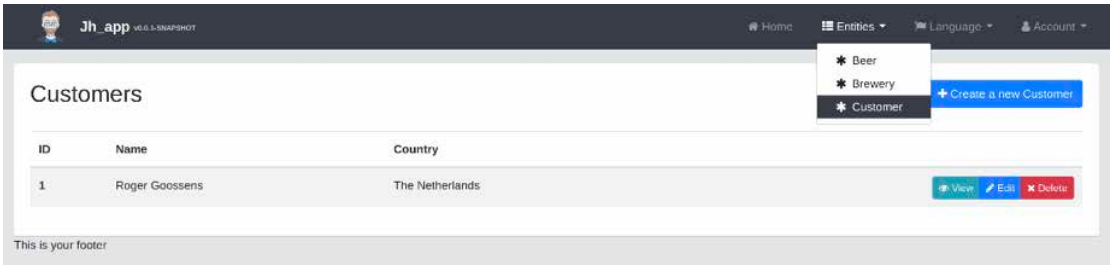

Fig. 7: Nieuw gegenereerde Customers-scherm

Custom back-end-code

Wat nou als we geen entiteit, maar een samengestelde service willen toevoegen aan de applicatie? Dit vereist het nodige maatwerk. JHipster biedt hier wel wat CLI-functionaliteit voor, maar dit is in de huidige versie nog redelijk minimaal.

Stel dat we een scherm willen toevoegen dat een overzicht geeft van de breweries met hun bijbehorende beers, welke we met een eigen REST-service willen ontsluiten?

De aanpak die we in dit Whitebook hebben gekozen is het bouwen van een serviceklasse die beide repositories (beer en brewery) combineert om het gewenste resultaat (brewerySummary) op te halen (De beer repository alleen bevat eigenlijk ook al de informatie die nodig is, maar in de code worden beide repositories geraadpleegd om aan te tonen dat dit ook tot mogelijkheden behoort). D.m.v. de JHipster CLI kan een service gecreëerd worden:

```
jhipster spring-service BrewerySummary
```

De gegenereerde code is vrij beperkt, enkel een lege klasse met een annotatie en een logger:

```
@Service
@Transactional
public class BrewerySummaryService {
  private final Logger log = LoggerFactory.getLogger(BrewerySummaryService.
class);
}
```

De data die opgehaald wordt door deze service wordt in een custom DTO-klasse gestopt. Hiervoor biedt JHipster helaas geen CLI-ondersteuning (er kan wel een 1 op 1 DTO voor een losse entiteit worden gecreëerd). Die zal dus handmatig moeten worden toegevoegd:

```
public class BrewerySummaryDTO implements Serializable {
  private String brewery;
  private List<String> beers;
…
}
```

Nadat de service-implementatie geschreven is, kan de controller worden gebouwd. Ook hier biedt JHipster een klein beetje hulp:

```
jhipster spring-controller BrewerySummary
```

Kies bij de vragen voor een GET methode (**getSummary**) en er wordt een BrewerySummaryResource klasse voor je gegenereerd (en een testklasse):

```
@RestController
@RequestMapping("/api/brewery-summary")
public class BrewerySummaryResource {

  private final Logger log = LoggerFactory.getLogger(BrewerySummaryResource.
class);

  /**
  * GET getSummary
  */
  @GetMapping("/get-summary")
  public String getSummary() {
    return "getSummary";
  }
}
```

Al met al vrij summier. Het autowiren en delegeren van methodes naar de service moet je zelf doen. Kortom wat de back-end-code aangaat zijn de hulpmiddelen van JHipster aanwezig, maar beperkt. Zie GitHub (https://github.com/rphgoossens/whitehorses/tree/master/whitebook_jhipster) voor de uiteindelijke code.

Het mooie is dat je met die laatste stap meteen (automatisch) een nieuwe API aan de Swagger-pagina hebt toegevoegd:

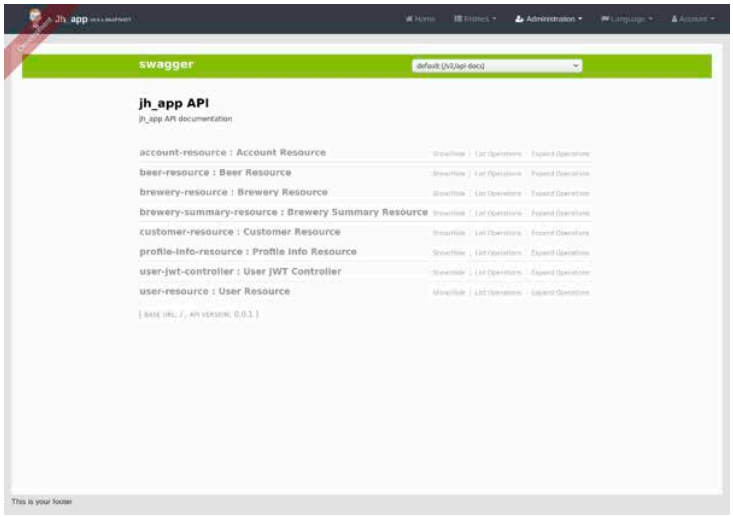

Fig. 8: API menu met nieuwe brewery-summart-resource API

In de Swagger-API staat alle informatie die je nodig hebt, om de REST-service een keer te testen. Zie hieronder een Postman verzoek (let erop dat je een Authorization header toevoegt, de waarde is te vinden in de Swagger-API-documentatie):

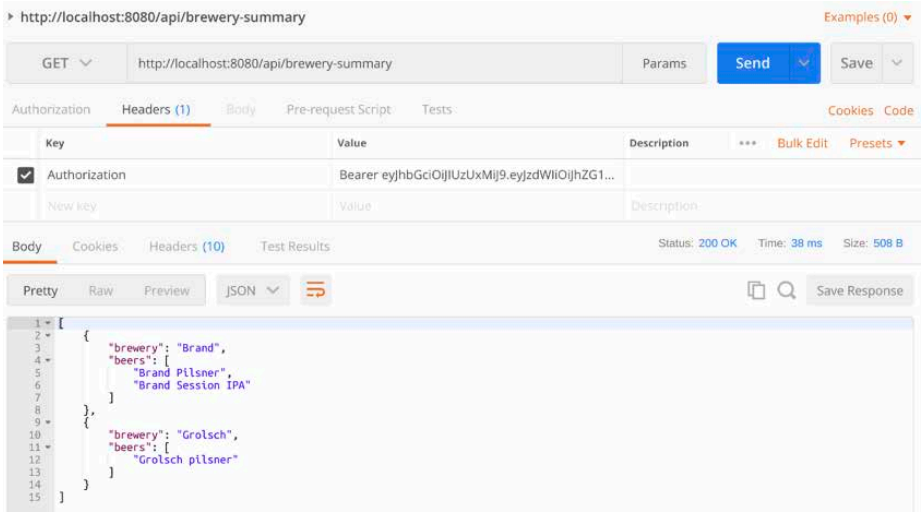

Fig. 9: Postman call naar nieuwe brewery-summary REST service

Custom front-end-code

Waren de JHipster-hulpmiddelen voor de back-end al beperkt, voor de front-end is er momenteel nauwelijks ondersteuning. Je kunt gebruikmaken van de Angular CLI, maar aangezien JHipster zijn eigen structuur heeft, is het voor de front-end aan te raden om de componenten zelf van scratch af aan op te bouwen, hierbij kan de structuur en code die JHipster al heeft gegenereerd voor de bestaande entiteiten als een blauwdruk dienen. Voor het BrewerySummary-scherm bouwen we een apart menu. Analoog aan wat JHipster voor de Beer-en Brewery-entiteiten gegenereerd heeft, bouwen we soortgelijke klassen en componenten voor onze BrewerySummary (zie GitHub voor de code):

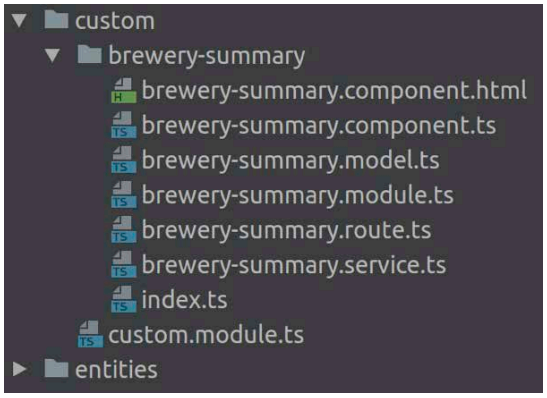

Fig. 10: brewery-summary Angular componenten

Delete en edit klassen (de popus en dialogs) zijn in dit geval niet nodig, omdat we enkel een read-only overzicht geven.

Naast bovenstaande nieuwe componenten, zijn er enkel nog aanpassingen nodig in de **app.module** klasse (declareer hierin de nieuwe Custom module), de **navbar.component.html** file (voeg hieraan het Custom menu toe) en de i18n folder (voeg hieraan de vertalingen van het nieuwe menu (**global.json**) en het nieuwe scherm (**BrewerySummary.json**) toe.

De applicatie heeft er nu een Custom menu bijgekregen met het nieuwe BrewerySummary scherm (netjes gebookmarked als http://localhost:9000/#/brewery-summary). Via de back-end REST-service die we in de vorige stap hebben toegevoegd, wordt het overzicht van Breweries met hun Beers opgehaald en in het nieuwe scherm getoond (voor de afwisseling wordt nu het Nederlandse scherm getoond).

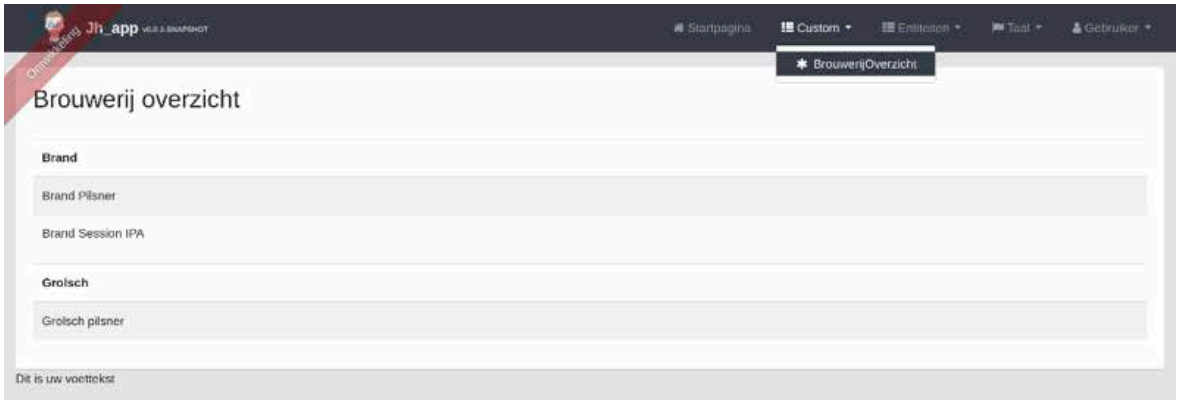

Fig. 11: Nieuw custom Brouwerij overzicht scherm

Conclusie

In dit Whitebook hebben we laten zien hoe eenvoudig het is om in zeer korte tijd een CRUD-applicatie te genereren m.b.v. JHipster. De applicatie wordt netjes gescheiden in een Spring Boot back-end en een Angular front-end. Beide applicaties zijn netjes gelaagd opgezet en voorzien van unit tests.

Naast de CRUD-schermen, krijg je o.a. een loginscherm cadeau en een Administration menu met daarin o.a. een API-pagina met Swagger-definities voor alle aanwezige REST-services. Ook i18n is standaard ingeregeld.

Het uitbreiden van een JHipster-applicatie is niet heel erg ingewikkeld. Een extra entiteit toevoegen compleet met CRUD-functionaliteit is heel eenvoudig. Zodra je echter aan de slag wil met complexere services en schermen, biedt de tool hiervoor zeker aan de front-end kant weinig extra hulpmiddelen. Allicht dat dit in nieuwere versies van de tool nog een stuk verbetert.

Desalniettemin krijg je veel cadeau bij het genereren van een JHipster-applicatie en kan het zeker als een solide basis dienen voor je eigen webapplicatie.

Referenties

- https://www.jhipster.tech
- https://projects.spring.io/spring-boot
- https://angular.io
- https://github.com/rphgoossens/whitehorses/tree/master/whitebook_jhipster