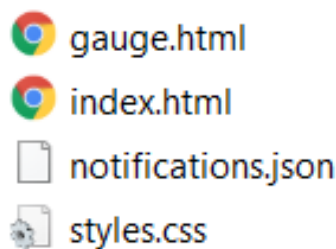


Labo Gebruikersinterfaces

Reeks 5: Angular (deel 2)

1 Getting started

1. Maak een nieuw angular project aan, en geef het de naam **smarthome**.
2. Compileer het project en start de webserver vanuit de juiste map.
3. Bekijk het resultaat in de browser.
4. Op Ufora vind je een aantal losse bestanden (verpakt in een zip).



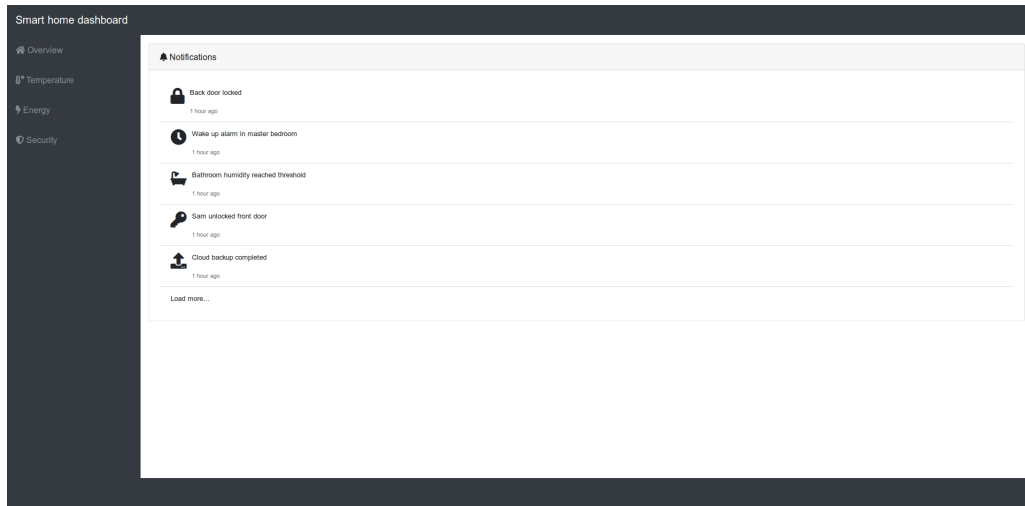
Deze startcode bevat een statische versie van de applicatie die we gaan ontwikkelen; in dit labo passen we dit aan tot een dynamische angular applicatie. Verhuis de html code uit het gegeven bestand *index.html*:

- het gedeelte binnen de **<head>**-tag (zonder de tag zelf) gaat naar *src/index.html*,
- het gedeelte binnen de **<body>**-tag (zonder de tag zelf) gaat naar *src/app/app.component.html*. Verhuis de inline css-code gewoon mee. Je mag de code die Angular gegenereerd heeft verwijderen uit *src/app/app.component.html*.
- En verhuis/kopieer tenslotte nog de attributen die in het gegeven bestand bij de **<body>**-tag staan.

Kopieer de code vanuit *styles.css* naar *styles.css* in je Angular project.

Tip: bij foutmeldingen controleer je of de link naar **styles.css** er nog staat (die mag weg), en of de code **<base href="/">** te vinden is in **index.html** (dat moet er wél staan).

Als IntelliJ bovenaan in het venster met de code een waarschuwing geeft over TSLint, dan klik je helemaal rechts op diezelfde regel, en laat je dit automatisch installeren. Daarna IntelliJ afsluiten en opnieuw opstarten.



2 Angular components

1. De overview pagina toont een lijst van notifications. Voorlopig zijn deze hard gecodeerd in *src/app/app.component.html* (met inline css).
2. We willen dat deze notifications dynamisch aan de pagina worden toegevoegd. We manipuleren het DOM niet rechtstreeks via JavaScript, maar gebruiken de elegantere aanpak van Angular.
3. Maak een nieuwe component “notification” aan.
4. Binnen *src/app* is er nu een mapje “notification” met daarin:
 - (a) *notification.component.html*: html template van de component
 - (b) *notification.component.scss*: (s)css styling van de component
 - (c) *notification.component.spec.ts*: unit tests voor de component
 - (d) *notification.component.ts*: De javascript (typescript) code voor de component.
5. Verplaats de html-code voor één notification naar de *notification.component.html* file. Knip de andere notifications uit *app.component.html* en plak die in een voorlopig document voor later gebruik.
6. Op de plaats van de verdwenen notifications includeer je nu de nieuwe component:


```
<app-notification></app-notification>
```
7. Je zou één enkele notification moeten zien in de browser. Let ook eens op het verschil tussen de code die je gemaakt hebt, de source code die je ziet in je browser (Ctrl+U) en de DOM die je ziet met developer tools (F12).
8. De gegeven html code van de notification bevat inline css styling. Verplaats deze naar de *notification.component.css* file. Gebruik de juiste selector in de css-code. Daarvoor moet je eerst even nagaan welke klassen er toegekend werden aan de bewuste tag. Er staan twee klassen: de klasse **fas** is bij elke notification hetzelfde. De andere klasse is altijd verschillend. Inspecteer het html-bestand in de browser. Dan zie je dat de selector **.fas** gebruikt wordt in het css-bestand *all.css*. Nieuwsgierigen achterhalen nu met wat online zoekwerk waar die tweede klasse voor staat.

Als de css-code verhuisd is, controleer je het resultaat in de browser.

9. We weten dat de html-code voor die ene notification die verhuisd is naar *notification.component.html* op drie plaatsen verschillend is van de andere notifications: de boodschap, het attribuut dat voor het icoontje zorgt, en de tijd.

Pas de *component.ts* aan zodat elke component drie instantievariabelen heeft: een boodschap, een icoontype en een timestamp. Gebruik voorlopig hardgecodeerde waarden om de boodschap en het icoontype te initialiseren. De timestamp initialiseer je ook hardgecodeerd met *new Date()*. (Er zal dus gewoon een datum komen, geen *1 hour ago*.)

10. Gebruik deze variabelen uit *component.ts* in de html code (angular interpolation).
11. Verwijder nu de hardgecodeerde waarden voor de variabelen voor het bericht en de icon in *notification.component.ts* en plaats *@Input()* ervoor.

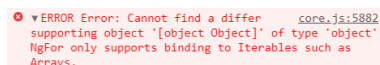
@Input() icon: string;

Dit laat ons toe om de waarde van dit attribuut in te stellen in de code van *app.component.html*:

```
<app-notification icon="fa-clock"></app-notification>
```

Tip: vergeet niet *Input* te importeren in *notification.component.ts*

12. Voeg de array “notifications” uit de gegeven startcode in “notifications.json” toe aan de AppComponent-klasse die je in *app.component.ts* vindt. Het is de bedoeling dat de AppComponent-klasse de eigenschap **notifications** heeft, waarvan de initialisatie gebeurt aan de hand van hardgecodeerde waarden.
13. Gebruik *structural directives* om voor elk object in deze array een nieuwe notification component toe te voegen aan de pagina.
Krijg je deze foutmelding?



Controleer dan of je de array notifications juist geïntialiseerd hebt.

3 Angular routing

In dit onderdeel willen we de verschillende tabbladen van de website alvast bereikbaar maken. Hun inhoud komt later.

Op de hoofdpagina zullen we het overzicht van de notifications laten zien (dus zoals nu), maar hetzelfde overzicht zullen we ook laten zien als er naar het adres *http://localhost:4200/overview* gesurft wordt. De adressen *http://localhost:4200/temperature*, *.../energy* en *.../security* zullen elk een andere inhoud tonen. De nodige bestanden (css, html, spect.ts en ts) om deze vier inhouden te verzorgen, zullen elk verzameld zitten in een aparte map (net zoals nu de map *notification* de inhoud verzorgt voor één notification). Elke map bevat dus alle nodige gegevens voor de betreffende nieuwe component van de website.

1. Maak nieuwe componenten voor de verschillende tabbladen (overview, temperature, energy en security). (Gebruik **ng generate**.)
2. Bekijk de code in de gegenereerde html-bestanden, bijvoorbeeld *energy.component.html*. Daar staat een html-tekst, enkel omgeven door **<p>**-tags. Omdat deze tekst helaas niet

zichtbaar zal zijn (en dat is gelegen aan de gebruikte Bootstrap-opmaak), zet je er best nog deze tags rond:

```
<div class="content-wrapper">
  <div class="container-fluid">
    ...
  </div>
</div>
```

3. Verplaats de html code die de lijst toont vanuit *app.component.html* naar de overview component. Vergeet ook de lijst met data niet te verplaatsen vanuit *app.component.ts* naar het overeenkomstig bestand bij de overview component. (Pak heel het stuk mee: vanaf de div-tag met attribuut `class="content-wrapper"`.)
4. We willen er nu voor zorgen dat je de verschillende tabbladen kan zien door te surfen naar `"/overview"`, `"/security"`, ... Dat kan heel eenvoudig door `href="/overview",...` in te stellen. Doe dat.
5. Zorg ervoor dat de router-outlet tag in de *app.component.html* staat. (Zie onderaan, misschien staat die er al?)
6. Pas de routing configuratie aan zodat de urls gemapt worden op deze nieuwe componenten.
7. Controleer of je de juiste componenten te zien krijgt als je naar de urls surft. Pas de links in de navigatiebalk aan.
8. Er wordt nu steeds een refresh gedaan van de pagina als je naar een ander tabblad surft. Pas de links aan zodat ze *routerLink* gebruiken in plaats van een *href*. Hiervoor moet je wel de navigatiebalk verhuizen vanuit *index.html* naar een angular component, als dat nog niet gebeurd was. Waarom is dit het geval ?

4 Two way data binding

1. Maak een nieuwe component *TemperatureGauge* aan, en zorg ervoor dat (de bijdrage van) deze component zichtbaar zal zijn in (de html-pagina van) de component *temperature*.
2. Gebruik de html- en css-code vanuit het gegeven bestand *gauge.html* als view voor deze component.
3. Merk in het html-bestand op dat er twee temperaturen getoond worden: de *current* temperatuur is gewone tekst, de *target* temperatuur is de waarde van een invoerveld.
4. Maak twee attributen aan in de corresponderende klasse: één voor de titel en één voor de temperatuur, en geef ze een defaultwaarde.
5. Zorg voor 2 way data binding van de waarde van het invoerveld en het attribuut in de klasse. Gebruik normale property binding voor de andere temperatuur. Beide temperaturen gebruiken hetzelfde attribuut in de klasse. Vergeet niet om de **FormsModule** toe te voegen aan het bestand *app.module.ts* - IntelliJ geeft je trouwens die hint.
6. Merk op dat het effect nu het volgende is: als je in de website de *target* temperatuur aanpast, dat gaat ook onmiddellijk de *current* temperatuur zich aanpassen.

5 Event binding

1. Zorg ervoor dat je slechts de eerste 5 notifications toont.
2. Door op de knop onderaan de lijst te klikken kan je er 5 extra tonen.
3. Toon deze knop enkel als er nog notifications zijn die nog niet getoond worden.

Een tip: laat de lus in de html-code ongewijzigd, ga daar geen voorwaarde op het aantal te tonen componenten toevoegen. Doe de grootste aanpassing in de klasse `OverviewComponent`: maak daar een nieuw attribuut aan, dat een kopie bewaart van alle *zichtbare* notifications. De lengte van dit attribuut zal dus variëren.

6 Angular routing (deel 2)

1. Maak een nieuwe component *NotificationDetail* aan. Deze component zal de details tonen van één specifieke notification.
2. Definieer de route *notification/:id* die naar deze component gaat.
3. Gebruik de routerLink om deze route te volgen als je op een notification klikt.
4. Toon de id van de notification in de detail pagina.

7 Service

1. Om de applicatie nu af te werken gaan we de notifications ophalen via AJAX.
2. Hiervoor gaan we een service gebruiken. Een service is een Angular component die gebruikt kan worden vanuit verschillende andere componenten. Hier zal de service verantwoordelijk zijn voor het ophalen en bijhouden van de notifications.
3. Maak een nieuwe service aan (**ng generate**) en gebruik dependency injection om een referentie te hebben binnen de overview component.
4. Maak twee methodes aan in de service, één om alle notifications terug te geven en één om één specifieke terug te geven. Voorlopig kan je de notifications nog hard coderen in de service.
5. Pas de overview component aan zodat de data vanuit de service gebruikt wordt.
6. Pas ook de notification detail component aan zodat die de juiste notification toont.
7. Pas nu je service aan zodat de data niet meer hardgecodeerd staat maar dat ze dynamisch opgehaald wordt vanop <http://www.mocky.io/v2/5be453402f00002c00d9f48f>

