Angular Basiscursus

Versie 0.x, we zijn er nooit klaar mee (:

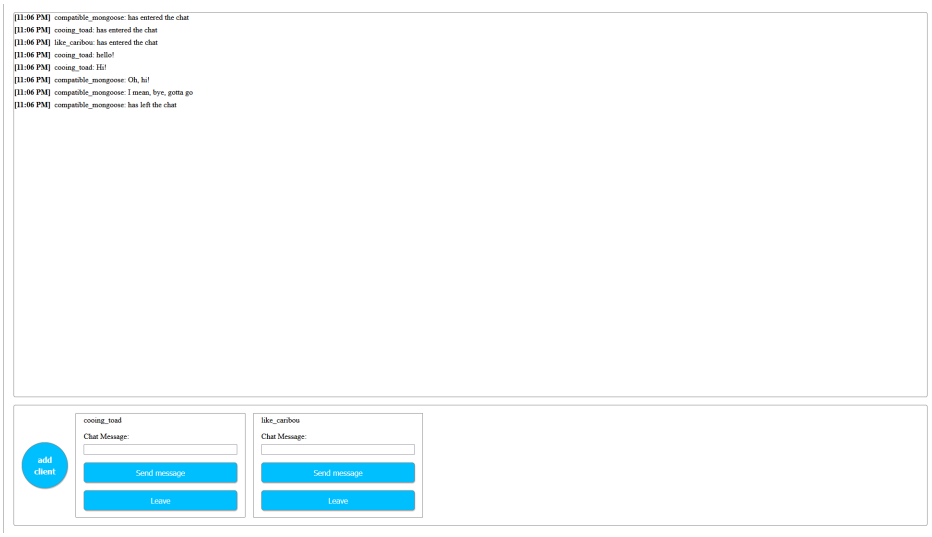
Roeland Kegel

Topicus Healthcare
BL 1e 2e Lijnszorg
9 maart 2023

Hoofdstuk 1

Dag 1

In het eerste deel van de cursus maken we een applicatie die locaal kan "chatten". Hiervoor duiken we in directives, components, pipes, styling en de angular life cycle.

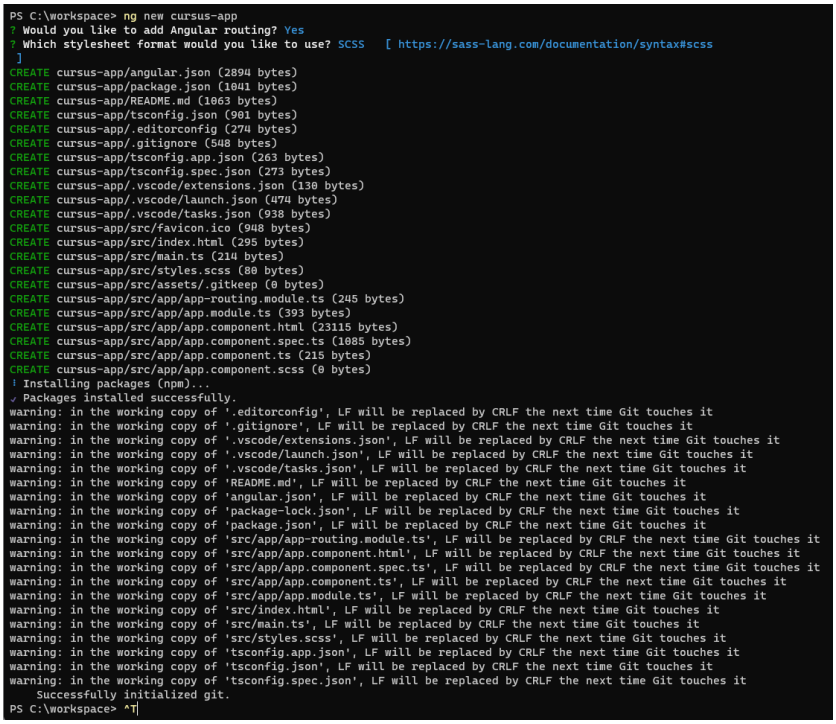


1.1 Project initialisatie: Een nieuwe (Chat)applicatie aanmaken

Zorg er eerst voor dat je een werkende node installatie hebt, het liefst `node >= 16` en `npm >=8`. Installeer dan de angular command line utilities met:

```
npm install -g @angular/cli
```

Angular heeft veel schematics, quick build targets voor het aanmaken van verschillende onderdelen van een Angular applicatie. Een nieuwe applicatie kan worden aangemaakt met: `ng new cursus-app`



Figuur 1.1: output van `ng new`

De angular applicatie kan worden gecompileerd naar een bundle van *tersed* JS, CSS en HTML met het ocmmando `ng build`. Deze files kunnen worden geserved als deel van je bestaande backend, of via een deployment tool worden uitgerold op S3, Firebase, Github Pages, of een andere distributieoplossing. Angular biedt mogelijkheden om dit proces te versimpelen door het `ng deploy` command. Voor meer informatie, zie `https://angular.io/guide/deployment`. Voor locaal ontwikkelen kunnen we een locale server opstarten die je applicatie toont. Deze update live met de veranderingen in je code:

```
ng serve
```

1.2 Components: Bouwblokjes in je applicatie

Eerst maken we een paar nieuwe componenten:

```
ng generate component chatScreen
ng g c chatClient
```

In de HTML template van het root component kunnen nu de componenten worden ingeladen door een tag met de selector van het child component: `<app-chat-screen></app-chat-screen>`.

1.2.1 Opdracht

Voeg een chatScreen en een paar chatClient componenten toe aan het root component.

1.3 Data binding: Heen en weer met data

Als we componenten in een hierarchie hebben kunnen we data *binden* aan ons component. We kunnen op een paar manieren refereren naar onze code:

- {{`value`}}: interpolation. Dit geeft waardes vanuit het aanliggende typescript component weer in het HTML template
- `[value]`: property binding: Dit geeft waardes door naar child components vanuit typescript. Child components kunnen vervolgens met een `@Input()` annotatie een waarde inladen in de tyepscript file van het child component.
- `value`: property binding zonder blokhaken: Dit geeft platte strings door naar het child component. Child components kunnen vervolgens met een `@Input()` annotatie een waarde inladen in de tyepscript file van het child component.
- `(value)`: event binding. Dit luistert naar events die gedefinieerd worden in child components. Child components kunnen vervolgens met de `@Output()` annotatie met een EventEmitter waardes naar een parent component sturen.
- `[(value)]`: *two-way* or *banana-box* binding. Dit zorgt ervoor dat wijzigingen tussen de waarde vanuit de HTML en typescript de variabele updaten. Wordt gebruikt bij input elementen om te luisteren naar de waarde *en* het updaten van de waarde vanuit typescript.

1.3.1 Opdracht

Geef wat vorm aan je chat applicatie:

- Geef een titel aan de clients via het root component,
- Gebruik `display: flex;` en `flex-direction: column;` om het chatscherm uit te lijnen boven de chat clients,
- Voeg een input toe met two way binding in de chatClients, met een knop om een bericht te sturen.
- Stuur via de output binding een string waarde met het chatbericht naar het root component.

- Voeg in het root component de chatregel toe aan een lijst met chatregels.
- Toon de chatregels op een gelijke wijze in het chatscreen component.
- Speel wat met de volgende CSS properties om je applicatie op te leuken:
  - `background-color`,
  - `color`,
  - `border`,
  - `border-radius`,
  - `margin`,
  - `padding`,
  - `overflow`,
  - `width`,
  - `height`

Als je wat meer wilt weten over CSS, zijn er goede resources te vinden op het internet voor flexbox, grid en css selectors in het algemeen.

1.4 Styling: Locaal en Globaal

Angular kent het concept *encapsulated styling*. Styles worden dan geladen via de stylesheet die bij een component hoort. Daarnaast kunnen styles ook ingeladen worden op globale wijze door ze aan een stylesheet toe te voegen die in de `angular.json` file. Het voordeel van encapsulated styling is dat regels binnen stylesheets alleen gelden voor binnen het component, waardoor styling makkelijker overzichtelijk blijft. Encapsulated styling kan worden aangepast van *emulated* (de standaard) naar *shadow dom* of *none*. Bij None wordt styling buiten het component ook toegepast. Bij ShadowDom worden global styles genegeerd.

1.4.1 Opdracht

Voeg styling toe aan de global `styles.scss` file:

```
.styled-button{
  background: deepskyblue;
  color: white;
  border: 1px solid gray;
  border-radius: 6px;
  box-shadow: rgba(0,0,0, 0.2) 1px 2px;
```

```
  line-height: 1.5rem;
  font-size: 1rem;
  padding: 8px 1rem;

  &:hover{
    background: cornflowerblue;
    &:active{
      background: dodgerblue;
    }
  }
}
```

Voeg daarna de styling toe aan de knoppen in je chat clients. Let op hoe deze class styling wel toegepast wordt in componenten, maar een simpele `div` styling binnen een component stylesheet andere componenten niet raakt.

1.5 Structural directives: Structuur toevoegen aan je applicatie

In Angular kennen we 3 soorten directives: Attribute, Component en Structural. Structural directives veranderen de elementen die getoond worden op een pagina. De meest voorkomende directives zijn:

- `*ngIf`: Toont/haalt een element weg gebaseerd op een boolean value.
- `*ngFor`: Voegt een bepaald template toe per element in een lijst.

onderwater gebruiken structural directives een `<ng-template>` tag. Dit is een inline definitie van een stukje injecteerbare HTML. Meestal wordt bij een `<ng-template>` of structural directive ook een `<ng-container>` tag gebruikt. Dit is een önzichtbaarëlement wat niet gerenderd wordt, maar wat gebruikt kan worden in combinatie met bijvoorbeeld een `*ngFor` directive.

1.5.1 Opdracht

- Laten we onze communicatie tussen componenten wat netter maken. definieer een chatbericht interface:

  ```
  export interface ChatMessageModel {
    clientName: string;
    message: string;
    timestamp: Date;
  }
  ```

en laat een chatclient ChatMessageModel objecten doorsturen i.p.v. platte strings.

- Schrijf daarna de initialisatie van chat clients om: in plaats van 3 `<app-chat-client>` tags kunnen deze ook met een `*ngFor="let client of clients"` structural directive aangemaakt worden, waarbij een client een simpele string is.
- Laat het chatcherm chatberichten tonen met een `*ngFor`.

1.6 Pipes: Performante functies in je HTML

Pipes worden gebruikt om op performance vriendelijke manier functies uit te voeren om dingen in een HTML template te laten zien. We gaan hier bij change detection iets dieper op in, maar in het kort: de functie wordt dan alleen uitgevoerd als de invoerwaarde verandert, aangezien een pipe als een pure function wordt beschouwd zonder side effects. Angular kent een aantal standaard pipes zoals `titleCase` en `date` om formatteren makkelijker te maken. Pipes worden toegepast in een template met een pipe operator:

```
{{value | date : ' short'}}
```

Waarbij de waarde na de — een referentie is naar een pipe, en waardes na de : bijkomende parameters zijn. Zo ziet een slice pipe (subsection van een list) er zo uit:

```
{{list | slice : 0 : 5}}
```

1.6.1 Opdracht

We hebben eerder onze chat berichten netjes omgeschreven naar een object. Dit kunnen we nu uitlezen en formatteren met pipes:

- Formatteer de timestamp met een `date` pipe,
- Formatteer messages met een `titleCase` pipe,
- Zet een maximale hoeveelheid berichten op het scherm met een `slice` pipe.

1.7 Attribute directives: Gedragsaanpassingen op pagina elementen

Attribute directives passen het gedrag of uiterlijk van een host tag/component aan. Ze kunnen luisteren naar click events, classes toevoegen aan een host element, of andere, wat geavanceerdere dingen doen zoals het toevoegen van tooltips of context menus aan elementen. Ze kunnen ook logica bevatten en daarmee dus ook conditioneel styling toepassen op een element.

1.7.1 Opdracht

Laten onze global styling net iets anders inladen. In plaats van een classname kunnen we ook een directive maken die met de `@HostBinding()` annotatie. Maak een directive met het commando `ng g d styledButton` en schrijf de bestaande knoppen om naar knoppen waar de class gezet wordt door middel van de appStyledButton directive.

1.8 Life cycle: Het begin en het einde van een component

De Angular lifecycle is een integraal onderdeel van snappen hoe je programmeert in Angular. Het zijn event hooks waar je op in kan haken bij het aanmaken, verwijderen en (re-)renderen van views binnen je applicatie. Belangrijke life cycle hooks die veel voorkomen zijn:

- `OnInit`: wordt aangeroepen als een component aangemaakt wordt, nadat de input properties vanuit een component zijn gezet
- `AfterViewInit`: Nadat het component voor het eerst gerenderd is
- `OnChanges`: Telkens als een input property verandert
- `DoCheck`: Elke change detection cycle, ongeveer 20x per seconde
- `OnDestroy`: Als het component wordt verwijderd, bijvoorbeeld door wegnavigeren van een pagina, of met een structural directive zoals `*ngIf`.

1.8.1 Opdracht

Laten we ook wat berichtjes sturen op het moment dat een nieuwe client de chatroom binnenkomt, of een client verwijderd wordt.

- Voeg knoppen toe om een chatclient aan te maken en verwijderen, zodat clients de "chatroom"kunnen binnenvallen en verlaten.
- **Bonus** installeer het unique-names-generator package met `npm i unique-name-generator` en geef aangemaakte chatclients automatisch een naam.
- Implementeer de OnInit interface in je chatClient component en laat het een bericht sturen naar de server als het component wordt aangemaakt
- Doe hetzelfde voor OnDestroy.

1.9 Custom pipes: Zelf functies klikken

Er bestaan voorgebouwde pipes binnen angular, maar het aanmaken van je eigen pipes is erg gemakkelijk. Het is een klasse die de PipeTransform interface implementeert. Pipes zijn standaard gemarkeerd als *pure*. Dat betekent dat alleen als het binnenkomende object van een transform verandert, de functie opnieuw wordt uitgevoerd. Als je pipe iets doet met observables of internet requests, zal deze snel als *impure* gemarkeerd worden. Dan is de functie al snel even performant als een normale tyepscript functie.
Belangrijk: primitives zoals strings zijn Pass by Value, maar objecten (zoals ons chatbericht) zijn Pass by Reference. Dat betekent dat een pure pipe niet zal worden uitgevoerd als een key binnen een object verandert.

1.9.1 Opdracht

Laten we het chatbericht formatteren met een custom pipe. Maak een nieuwe pipe aan met `ng g p formatChatMessage`. Laat een chatregel een geformatteerd tijd, naam en bericht regel tonen. Pas deze pipe vervolgens toe binnen het chatscherm.

Hoofdstuk 2

Dag 2

2.1 Routing

Routing wordt gebruikt in een angular applicatie om andere html files in een klassieke webpagina te simuleren. Angular gebruikt hier *content projection* voor, met de `<router-outlet>` tag. Routing wordt toegevoegd met de volgende stappen:

- Definieer een routing file. Kan aangemaakt worden samen met een module door `ng g m moduleName --routing`
- Definieer routes in de router file
- Zet routerlinks op elementen in je pagina
- Voeg een `<router-outlet>` toe aan je root component.

2.1.1 Opdracht

Laten we routing toevoegen aan onze applicatie:

- Voeg een routing module toe en importeer deze in je root module. Het kan zijn dat je applicatie hier al mee is geinitialiseerd.
- Maak een ChatPage component
- Maak een BeheerPage component
- Voeg 2 routes toe aan je angular applicatie: een voor het ChatComponent, en een voor het Beheer component.
- Laat een 3e default route redirecten naar het Chat component
- Voeg met de routerlink attribute een paar links of knoppen toe aan je root component als top bar, zodat je kan wisselen tussen de chat en beheerpagina's.

2.2 Services

Services worden in Angular gebruikt om state bij te houden in je applicatie, of externe bronnen te bevragen. Ze kunnen worden geimporteerd via dependency injection in de constructor van andere services of directives. Dependency injection kan je op verschillende niveaus doen om state te isoleren tussen classes of modules, maar het is gewoonlijk om services in de root te definieren, waardoor je effectief een Singleton object hebt voor de service.

2.2.1 Opdracht

We willen een service maken om gebruikers op te halen die we aan onze chat kunnen toevoegen. Hiervoor maken we een userService:

- Maak een userService aan met ng generate service userService
- Geef de service de `addUserToChat`, `removeUserFromChat` en `listUsers` methodes.
- Geef de service een activeUsers lijst met strings.
- Maak een lijstje met mock users. Voeg 1 of 2 van deze users toe aan activeUsers bij ngOnInit. Implementeer de remove, list en add methodes om te werken met de mockUsers lijst.
- Injecteer de userService in de chatApplicatie. Laat de chatApplicatie de activeUsers opvragen bij ngOnInit, zodat active Users gelijk aan de chat worden toegevoegd.

2.3 Forms

Angular gebruikt Reactive Forms om formulieren bij te werken, valideren en consolideren op pagina's. Forms bestaan uit instanties van *AbstractControl*, die vaak geimplementeerd worden door een van de volgende 3 elementen:

- *Form Controls*: Enkele elementen die aan een input element gebonden kunnen worden. Bevat een enkele waarde om aan te passen,
- *Form Groups*: Verzamelingen van meerdere `AbstractControl`s. Een FormGroup bevat doorgaans een paar Form Controls, maar kan ook nested `FormGroup`s of `FormArray`s bevatten,
- *Form Arrays*: Dynamisch te instantieren sets van *AbstractControl*s, bijvoorbeeld voor een lijst van kleine forms waar meer instanties van bijgeplust kunnen worden.

Alle AbstractControls kunnen validators toevoegen om bijvoorbeeld te zeggen dat een input niet leeg mag zijn, of een string van maximale lengte mag bevatten. Het houdt ook de staat van een control bij (pristine / dirty / touched / untouched / valid / invalid).

2.3.1 Opdracht

De beheerpagina kan een form gebruiken om gebruikers te zoeken, toe te voegen en te onderscheiden.

- Voeg een `User` interface toe met de properties actief en naam,
- Laat de bestaande chat client uitgaan van `User` objecten i.p.v. platte strings,
- Maak een FormGroup in de beheerpagina met controls voor hideActief, filterNaam en selectedUser,
- Installeer angular material met het commando `ng add @angular/material`,
- Voeg een `mat-select` toe voor user, en een `matInput` en `matCheckbox` voor de filters,
- Laad de lijst met gebruikers vanuit de user service in als opties voor de mat-select,
- Voeg een knop toe die de geselecteerde gebruiker toevoegt als actieve gebruiker.

2.4 Observables

Observables zijn een onderdeel van het RxJS project en zijn een kernonderdeel van Angular. Observables zijn een combinatie van het Observer en Iterator design pattern. De built-in HTTP client van Angular gebruikt ze, en het zijn ook geschikte constructies om o.a. eventstreams of websockets mee te onderhouden. Observables zijn long-lived functions die opties hebben tot multicasting of asynchrone processing. Standaard zijn het *cold* (unicast), *synchronous* functions die een functie uitvoeren als er naar gekeken wordt met een subscribe. Er bestaan veel operators en functies waar eventstreams mee kunt aanmaken en manipuleren. Voor een algemenere introductie wat streams zijn kan je ook kijken naar het prachtige (op Apache Kafka gebaseerd, een java equivalent) geillustreerde verhaal wat een speelse uitleg geeft over hoe complex het kan worden, en welke problemen observables op proberen te lossen.

2.4.1 Opdracht

Laten we eerst wat HTTP requests simuleren, om net te doen of onze user service daadwerkelijk een externe bron bevraagt:

- Laat de user service een `Observable<User[]>` terug geven. Door de userlijst met een `of()` te omsingelen maak je een Observable van je `User[]` variabele.
- Met de `delay` operator in een `pipe` kan je een http lag simuleren.

2.5 De HTTPClient en alles bij elkaar

De Angular `HttpClient` kan gebruikt worden om requests te sturen naar een externe bron. Je kan een `HttpParams` toevoegen om parameters aan je request toe te voegen, en een `HttpHeaders` object om headers toe te voegen. Je kan ook met een apart gedefinieerde `HttpInterceptor` standaard headers of parameters toevoegen aan een request, of een url proxyen naar een andere URL als je bijvoorbeeld in een testomgeving zit.

2.5.1 Opdracht

Laten we alles wat we tot nu toe hebben behandeld bij elkaar komen. We gaan de Github API bevragen om wat users te suggereren om toe te voegen aan onze chat:

- Maak een personal access token op github zodat je de github api kan bevragen en private repo's kan uitlezen. Indien je dit niet kan of wil, kan je ook de user search van github direct gebruiken,
- Laat je userService een Github search doen met de HttpClient:
  - Voeg je authentication toe in de authorization header,
  - Maak een input om te zoeken binnen een specifieke organisatie,
  - Maak daarna een input om te zoeken naar repositories binnen die organisatie,
  - Laat de search users actie nu collaborators zoeken binnen de gedefinieerde repository.
- Voeg de user toe aan de chat door het te converteren naar een User object voor je chat applicatie en het toe te voegen aan de actieve gebruikers.,
- **Bonus:** Voeg filters toe voor een minimale hoeveelheid contributions op de repository

- **Bonus:** Voeg error validatie toe die zoeken met een organisatie of repository naam van minder dan 3 letters niet toestaat, door gebruik te maken van de `minLength` validator van Angular
- **Bonus:** Gebruik de `distinctUntilChanged`, `debounce` rxJs operators en de `valueChanges` methode in AbstractControl om automatisch zoekopdrachten uit te voeren bij GitHub
- **Bonus:** Schrijf de HttpHeaders constructie in de userService om naar een `HttpInterceptor`