Eerste Angular Applicatie opzetten

Opdracht 1.1 – Nieuwe App met CLI

Gebruik de Angular CLI om je eerste app te bouwen

a. Installeer Angular CLI (gebruik de nieuwste versies van node v.12.16.0 en npm v.6.13.7):

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

b. Open een commando prompt waar je het project wilt maken, en gebruik de Angular CLI om aan het project te beginnen:

```
ng new angular-workshop
```

De CLI stelt 2 vragen; of je Angular Routing wilt gebruiken (kies voor deze workshop het antwoord 'nee'), en welke Stylesheet Format je wilt gebruiken (voor deze workshop wordt SCSS gebruikt)

c. Voeg Bootstrap toe aan *package.json* als dependency ("bootstrap": "latest") en aan de styles array in *angular.json* (regel 30: "styles": ["styles.css", "./node_modules/bootstrap/dist/css/bootstrap.css"]). Voer het commando `npm install` uit vanuit de project folder.

d. Run je app met het volgende commando:

```
ng serve
```

De app is nu beschikbaar in je browser op http://localhost:4200. Wijzig de titel in app.component.ts en kijk hoe de nieuwe html in de browser wordt getoond.

Opdracht 1.2 – Nieuw component maken

Maak een nieuw component aan en voeg deze toe aan de app.

a. Maak een nieuwe folder aan (app/header), maak daarin een nieuw TypeScript bestand (header.component.ts). Maak een nieuwe class met de @Component() decorator, en geef een waarde voor `selector` en `template`. Voeg een console.log() toe aan de ngOnInit methode.

```
@Component( {selector: 'app-header', template: ''} )
export class HeaderComponent implements OnInit {

  ngOnInit() { console.log('het component werkt!'); }

}
```

b. Importeer het nieuwe component in app.module.ts door het toe te voegen aan de lijst met `declarations`. Plaats vervolgens het component in je app.component.html (`<app-header></app-header>`), en dan zou je de log in je browser moeten kunnen zien.

c. Geef nu ook waardes aan de @Component decorator voor `template` en `styles` (bijv. `'<h1>Header </h1>'` als template en `['h1 { color: blue }']` voor styles)

Opdracht 1.3 – TemplateUrl en StyleUrls

a. Maak de nieuwe bestanden header.component.html en header.component.scss aan. Voeg hieraan de code toe die je in je header wilt zien, bijvoorbeeld in het html bestand:

```
<ul class="nav nav-tabs">
   <li class="nav-item">
      <a class="nav-link active">Ingredients</a>
   </li>
   <li class="nav-item">
      <a class="nav-link">Recipes</a>
   </li>
</ul>
```

En in het SCSS bestand:

```
:host {
   display: block;
   margin: 2em 0;
}
```

b. Vervang in de @Component decorator `template` met `templateUrl` en `styles` met `styleUrls`:

```
@Component({
   selector: 'app-header',
   templateUrl: './header.component.html',
   styleUrls: ['./header.component.scss']
})
```

Data Binding

Opdracht 2.1 – Nieuwe componenten ingredient-list en ingredient-add

Maak een nieuw component ingredients, en daarin twee nieuwe componenten ingredient-list en ingredient-add

a. Gebruik de CLI om in totaal 3 nieuwe componenten aan te maken (bijv `ng generate component ingredients --skipTests` en `ng g c ingredients/ingredient-list --skipTests` voor de eerste 2 componenten).

b. Plaats `<app-ingredients>` in app.component.html, en plaats in app.ingredients.html beide andere componenten:

```
<h3>Here you will find all the ingredients!</h3>
<div class="row">
   <div class="col-sm">
      <app-ingredient-list></app-ingredient-list>
   </div>
   <div class="col-sm">
      <app-ingredient-add></app-ingredient-add>
   </div>
</div>
```

Opdracht 2.2 – String interpolation en two-way-binding

Voeg input velden toe aan de html en toon de waardes van de input met string interpolation

a. Maak in ingredient-add.component.ts twee nieuwe variabelen, `ingredientName` en `ingredientAmount`.

b. Maak in de bijbehorende html 2 input velden. Voeg een input toe voor Ingredient Name (zoals hieronder) en voeg ook een input toe voor Ingredient Amount:

```
<h4>Add new ingredients</h4>
<div class="form-group">
   <div class="row">
      <div class="col-sm">
         <label>Ingredient name:</label>
      </div>
      <div class="col-sm">
         <input type="text" class="form-control">
      </div>
   <!-- TODO: add html for Ingredient Amount as well -->
```

c. Plaats onder de input velden een <p> element waarin de waardes getoond kunnen worden (`<p>Ingredient name: {{ingredientName}}</p>` en `<p>Ingredient amount: {{ingredientAmount}}</p>`)

d. Voeg tenslotte de two-way-binding toe aan de input elementen (bijv `[(ngModel)]="ingredientName"`). Denk er aan dat het voor ngModel nodig is om FormsModule als import toe te voegen aan app.module.ts. In de browser zou je nu de waardes van je input velden geprint moeten zien worden.

Opdracht 2.3 – Event-binding en property-binding

Voeg een knop toe die de waardes van de input velden kan printen, en die disabled is wanneer er geen waarde voor beide velden is opgegeven

a. Plaats een knop onder de input velden (`<button type="button" class="btn btn-primary">Add ingredient</button>`). Geef de knop een event-binding mee, waarmee je naar een click event luistert en een nieuwe methode aanroept: `(click)="addIngredient()"`

b. Voeg de methode `addIngredient()` toe aan het TypeScript bestand, waarin je eerst een console.log() gebruikt om de `ingredientName` en `ingredientAmount` te loggen, en waarin je daarna beide waardes op `null` zet.

c. Voeg tenslotte ook een property-binding toe aan de knop, op de property disabled, waarmee je een methode isDisabled aanroept: `[disabled]="isDisabled()"`. In deze nieuwe methode controleer je of waardes voor naam en aantal bestaan (`return !this.ingredientName || !this.ingredientAmount;`).

Input en Output decorators

Opdracht 3.1 – Nieuw component met *ngFor

Maak een model voor Ingredient, en gebruik deze om een lijst van ingrediënten te tonen in ingredient-list.component

a. Maak een nieuwe folder app/models, en maak hierin een nieuw bestand ingredient.model.ts. Plaats hierin een constructor waar naam en aantal kunnen worden meegegeven:

```
export class Ingredient {
  constructor (public name: string, public amount: number = 1) {}
```

b. Maak in ingredient-list.component.ts een variabele: `ingredients: Ingredient[];`. Maak in de bijbehorende html een lijst van deze ingredients met behulp van de structural directive *ngFor:

```
<h4>Ingredients list:</h4>
<ul class="list-group">
   <li class="list-group-item list-group-item-action"
        *ngFor="let ingredient of ingredients">
      {{ingredient.name}} / {{ingredient.amount}}
   </li>
</ul>
```

Opdracht 3.2 – Input decorator

Maak een lijst van ingrediënten in ingredients.component.ts, en koppel deze aan de ingredient-list component met behulp van input decorators

a. Maak en initialiseer een lijst van Ingredients aan in ingredients.component.ts:

```
ingredients: Ingredient[] = [
   new Ingredient('apples', 12), new Ingredient('bananas', 3)
];
```

b. Voeg aan de variable ingredients in ingredient-list.component.ts een `@Input()` decorator toe.

```
@Input() ingredients: Ingredient[];
```

Geef daarna aan het component in de html (ingredients.component.html) de waarde voor de input door met behulp van property-binding:

```
<app-ingredient-list [ingredients]="ingredients"></app-ingredient-list>
```

De lijst zou nu zichtbaar moeten worden in je browser.

Opdracht 3.3 – Output decorator

Voeg een eventEmitter toe aan ingredient-add waarmee een nieuw ingrediënt kan worden toegevoegd aan de lijst in ingredients.component

a. Voeg een eventEmitter met een `@Output()` decorator toe in ingredient-add.component.ts, en noem deze `ingredientAdded`:

```
@Output() ingredientAdded = new EventEmitter<{name:string, amount:number}>();
```

b. Roep deze eventEmitter aan vanuit de functie addIngredient:
`this.ingredientAdded.emit({ name: this.ingredientName, amount: this.ingredientAmount }).`

c. Dit event kun je nu opvangen met event-binding op het element, waarmee je vervolgens een nieuwe methode in het parent-element kunt aanroepen. In ingredients.component.html:

```
<app-ingredient-add (ingredientAdded)="onIngredientAdded($event)"></app-ingredient-add>
```

En in ingredients.component.ts:

```
onIngredientAdded(data): void {
   this.ingredients.push(new Ingredient(data.name, data.amount));
}
```

Services

Opdracht 4.1 – Een nieuw component recipes

Maak een nieuw component Recipes, en bouw een simpele link om tussen de componenten te kunnen schakelen

a. Gebruik de Angular CLI om het nieuwe component recipes te maken. Maak in het nieuwe TypeScript bestand een variabele recipes, zoals het voorbeeld hieronder, en voeg hier wat extra recepten aan toe:

```
recipes = [
   { name: 'pizza', ingredients: [
      new Ingredient('tomatoes', 10), new Ingredient('cheese', 12)
   ]}
];
```

b. Voeg in de html van het Recipes component een lijst toe, waar de recepten kunnen worden weergegeven:

```
<h3>Recipes:</h3>
<div class="list-group">
   <div class="list-group-item" *ngFor="let recipe of recipes">
      <h4>{{recipe.name}}</h4>
      <div class="row">
         <div class="col-sm-8">
            <span *ngFor="let ingredient of recipe.ingredients">
               {{ingredient.name}}: {{ingredient.amount}} /
            </span>
         </div>
      </div>
   </div>
</div>
```

c. Verderop in deze workshop wordt op een nette manier navigatie tussen de componenten toegevoegd (hoofdstuk 6: Routing), maar voor nu wordt een simpele en snelle manier toegepast met de Angular directive `*ngIf`. Voeg hiervoor eerst in app.component.html twee links toe, die met event-binding een waarde aan variabele `page` kunnen meegeven (voeg deze variabele ook toe aan app.component.ts):

```
<a (click)="page = 0">Ingredients</a> / <a (click)="page = 1">Recipes</a>
```

d. Voeg de componenten selectors van Recipes en Ingredients toe, en gebruik een `*ngIf` om te bepalen welk component getoond moet worden:

```
<app-ingredients *ngIf="page === 0"></app-ingredients>
<app-recipes *ngIf="page === 1"></app-recipes>
```

Opdracht 4.2 – Ingredients service

Maak een service die de ingrediënten kan opslaan en vrijgeven, waardoor de Recipes en Ingredients componenten dezelfde ingrediënten kunnen delen.

a. Maak een nieuw bestand ingredients.service.ts binnen de folder /ingredients. Maak hierin een nieuwe class aan, en voeg een lijst met ingrediënten toe.

```
export class IngredientsService {
   private ingredients: Ingredient[] = [
      new Ingredient('salami', 8),
      new Ingredient('peppers', 3)
   ];
}
```

b. Voeg twee methodes toe: getIngredients (`return this.ingredients.slice()`) en addIngredient (`this.ingredients.push(ingredient)`)

c. Voeg de IngredientService vervolgens toe aan app.module, in de *providers* lijst.

Opdracht 4.3 – Ingredients service in Recipes component

Voeg een methode toe in recipes.component die ingrediënten kan toevoegen aan de service

a. In de html van recipes, waar de lijst met recepten wordt getoond, moet een link worden toegevoegd met een event-binding die een nieuwe methode aanroept:

```
<div class="col-sm-4">
   <button type="button" class="btn btn-link"
           (click)="addIngredients(recipe.ingredients)">
      Add ingredients to list
   </button>
</div>
```

b. De methode in recipes.component.ts geeft de ingrediënten door aan de service. Voeg eerst de service als parameter toe aan de constructor (`constructor (private ingredientsService: IngredientsService) {}`), en roep dan addIngredient van de service aan om de ingrediënten daar toe te voegen

```
addIngredients(ingredients: Ingredient[]): void {
   ingredients.forEach( (ingredient) => {
      this.ingredientsService.addIngredient(ingredient);
   });
}
```

Opdracht 4.4 – Ingredients service in Ingredients component

Roep de IngredientsService aan vanuit het ingredients component, om de lijst weer te kunnen geven

a. Ga naar ingredients.component.ts, en voeg de IngredientsService toe als parameter in de constructor.

b. Voeg in de ngOnInit functie een aanroep toe naar de getIngredients() methode van de service: `this.ingredients = this.ingredientsService.getIngredients();`

Opdracht 4.5 – Ingredients service en RxJS Subject()

Voeg een RxJS Subject() toe aan de ingredientsService, zodat de ingredient-list mooi up-to-date blijft met de lijst

a. Verwijder eerst de eventEmitter die de communicatie tussen de ingredient-add.component en de ingredients.componenten mogelijk maakt (verwijder de eventListener `(ingredientAdded)="onIngredientAdded"` uit ingredients.component.html, de methode `onIngredientAdded` uit ingredients.component.ts, en de `eventEmitter ingredientAdded` uit ingredient-add.component.ts).

b. Voeg nu de ingredientsService toe aan ingredient-add.component.ts, en voeg in de methode `addIngredient()` een aanroep toe naar de service:

```
const newIngr = new Ingredient(this.ingredientName, this.ingredientAmount);
this.ingredientsService.addIngredient(newIngr);
```

c. Zoals je ziet, komen nieuw toegevoegde ingrediënten nu niet meer in de lijst te staan. Om dit op te lossen, kan gebruik gemaakt worden van een observable Subject () (te importeren op deze manier: `import { Subject } from 'rxjs';`). Deze Subject moet worden toegevoegd aan ingredients.service:

```
ingredientsChanged = new Subject<Ingredient[]>();
```

d. Vervolgens moet Subject worden gebruikt zodra de lijst wordt ge-update, oftewel in de methode `addIngredient`, nadat `this.ingredients.push` is uitgevoerd, met behulp van de methode `next()`:

```
this.ingredientsChanged.next(this.ingredients.slice());
```

e. Tenslotte moet naar het `Subject` en diens `next()` methode worden geluisterd vanuit de ingredients.component, zodat de lijst kan worden bijgewerkt zodra wijzigingen zijn doorgevoerd. In de functie ngOnInit van ingredients.component.ts:

```
this.ingredients = this.ingredientsService.getIngredients();
this.ingredientsService.ingredientsChanged
    .subscribe((ingredients: Ingredient[]) => {
       this.ingredients = ingredients;
    }
 );
```

HttpClient

Opdracht 5.1 – HttpClientModule

Maak in de ingredients.service gebruik van de HttpClientModule om data op te halen van en weg te schrijven naar een backend service

a. Voeg de HttpClientModule toe aan de imports lijst in app.module.ts

b. Verwijder in de ingredients.service de waardes die worden toegeschreven aan `ingredients`, gebruik de decorator `@Injectable()` en voeg een constructor aan de service toe waarmee de `HttpClient` wordt geïnjecteerd:

```
@Injectable()
export class IngredientsService {
    private ingredients: Ingredient[];
    constructor(private httpClient: HttpClient) {}
```

Opdracht 5.2 – getIngredients met httpClient.get()

In `getIngredients()` kan nu worden gekeken of de lijst met ingrediënten al waardes heeft en kan worden teruggegeven, of dat de waardes van een backend gehaald moeten worden

a. Voeg een `if()` statement toe waarbinnen een bestaande lijst wordt teruggegeven:

```
if (this.ingredients) {
   return this.ingredients.slice();
```

b. Voeg een `else()` statement toe waarbinnen de `httpClient.get()` methode wordt aangeroepen:

```
} else {
   this.httpClient.get<Ingredient[]>(
      'https://ordina-fontys-workshop.firebaseio.com/ingredients.json'
   )
```

c. Voeg een `subscribe()` methode toe aan de `httpClient.get()` en schrijf daarbinnen de response naar de `ingredients` lijst, en roep de `next()` functie aan op het `ingredientsChanged` Subject:

```
.subscribe((response) => {
    this.ingredients = response;
    this.ingredientsChanged.next(this.ingredients.slice());
})
```

Opdracht 5.3 – addIngredient met httpClient.put()

In `addIngredient()` kan nu, nadat `ingredients` in de service is bijgewerkt, de nieuwe lijst naar de backend worden gestuurd

a. Voeg na de ingredients.push() statement de volgende regel toe, waarmee naar de backend een PUT request gemaakt wordt:

```
this.httpClient.put(
   'https://ordina-fontys-workshop.firebaseio.com/ingredients.json',
   this.ingredients
)
```

b. Voeg na de httpClient.put() weer een subscribe() methode toe, waarbinnen met de response opnieuw de `next()` functie van het `ingredientsChanged` Subject aangeroepen kan worden:

```
.subscribe((response: Ingredient()) => {
   this.ingredientsChanged.next(response);
});
```

Routing

Opdracht 6.1 – Routing module

Voeg een nieuwe module toe waar de routes kunnen worden geconfigureerd

a. Maak een nieuw bestand app/app-routing.module.ts. Maak hierin een class `AppRoutingModule`, en geef deze een `@NgModule` decorator.

b. Maak in datzelfde bestand een `const routes: Routes = []` aan (buiten de AppRoutingModule class), en vul deze array met vier nieuwe routes (een redirect naar /ingredients, de ingredients en recipes componenten, en een catch welke ook weer naar /ingredients redirect)

```
{ path: '', redirectTo: '/ingredients', pathMatch: 'full' },
{ path: 'ingredients', component: IngredientsComponent },
{ path: 'recipes', component: RecipesComponent },
{ path: '**', redirectTo: '/ingredients' }
```

c. Geef de @NgModule decorator een imports (`imports: [RouterModule.forRoot(routes)]`) en exports (`exports: [RouterModule]`) array mee

d. Voeg de nieuwe AppRoutingModule toe als import in app.module

Opdracht 6.2 – Router outlet en navigatie

Voeg een router-outlet element toe waar de componenten kunnen worden weergegeven, en gebruik de navigatie uit de header om de juiste paden op te geven

a. Voeg de router-outlet toe in app.component.html, waarbij de <app-ingredients> en <app-routes> kunnen worden verwijderd.

```
<app-header></app-header>
<router-outlet></router-outlet>
```

b. Navigatie in de browser werkt nu al, maar de links werken nog niet. Voeg hiervoor een property-binding toe aan de <a> elementen in header.component.html. Bind hierbij aan de property [routerLink], en geef hier een array aan mee waarin je de paths van de componenten meegeeft:

```
<li><a [routerLink]="['ingredients']">Ingredients</a></li>
```

c. Om de links active te maken, kan de Angular directive routerLinkActive worden gebruikt. Geef aan beide <a> elementen deze directive mee (`<a routerLinkActive="active">`)

d. Om vanuit Recipes meteen te kunnen navigeren naar Ingredients wanneer je ingrediënten toevoegt, moet navigatie vanuit de class mogelijk zijn. Voeg in de constructor van recipes.component.ts daarvoor eerst de module Router toe (`constructor (private router: Router) {}`). Deze router kan vervolgens worden gebruikt in de methode `addIngredients()`, nadat de `ingredientsService` is aangeroepen om de ingrediënten op te slaan:

```
this.router.navigate(['ingredients']);
```

Opdracht 6.3 – Child paths

Voeg sub paden toe vanaf het pad /ingredients, zodat meer gedetailleerde navigatie mogelijk wordt

a. Voeg 2 nieuwe paden toe in app-routing.module, als children van het pad /ingredients:

```
{ path: 'ingredients', component: IngredientsComponent, children: [
      { path: 'new', component: IngredientAddComponent },
      { path: 'edit/:id', component: IngredientAddComponent }
   ]},
```

b. Vervang het component `<app-ingredient-add>` in ingredients.component.html met een `router-outlet`. Beide paden openen nu (nog) hetzelfde component.

c. De volgende stap is om beide nieuwe paden beschikbaar te maken vanuit de html, en dat doen we in ingredient-list.component.html. Om naar ingredients/new te navigeren, voegen we een button toe onder de lijst, met een routerLink property:

```
<button class="btn btn-outline-primary btn-block" [routerLink]="['new']">
    New ingredient
</button>
```

d. Om naar ingredients/edit/:id te navigeren, voegen we een (click) event-binding toe aan de list elementen. Gebruik `let i = index` in de *ngFor directive (in ingredient-list.component.html), en roep daarmee de methode aan `(click)="editIngredient(i)"`.

e. Om in de methode `editIngredient()` te navigeren, hebben we de modules Router en ActivatedRoute nodig; voeg deze toe aan de constructor van ingredient-list.component.ts. Daarna kan vanuit de methode worden genavigeerd naar de nieuwe route:

```
constructor(private router: Router, private route: ActivatedRoute) {}

editIngredient(index): void {
    this.router.navigate(['edit', index], {relativeTo: this.route});
}
```

Opdracht 6.4 – Route params uitlezen

Update het ingredient-add.component zodat nieuwe ingrediënten kunnen worden toegevoegd, en bestaande ingrediënten kunnen worden aangepast.

a. De eerste stap is om in ingredient-add.component.ts te kijken naar wat het pad is waar de gebruiker op zit. Hiervoor is de module ActivatedRoute nodig; voeg deze toe aan de constructor:

```
constructor (private ingredientsService: IngredientsService,
             private route: ActivatedRoute) {}
```

b. Van ActivatedRoute kan `params` worden aangeroepen, welke een observable teruggeeft. Hier kan een `subscribe` methode op aangeroepen worden, en in de callback functie kunnen de params van ActivatedRoute worden opgeslagen. Deze params worden gebruikt om een variabele `ingredientIndex` gelijk te zetten aan de `param:id`, of om deze op `null` te zetten:

```
ingredientIndex: number = null;

ngOnInit() {
    this.route.params.subscribe(
       (params: Params) => {
          if (params['id']) {
             this.ingredientIndex = +params['id'];
          } else {
             this.ingredientIndex = null;
          }
    })
 }
```

c. Met de waarde ingredientIndex kunnen we nu in de html verschil maken tussen ingredients/add en ingredients/edit/:id. Bijvoorbeeld met verschillende titels en knoppen (de methodes voor de Update en Delete knoppen worden later nog toegevoegd):

```
<h4>{{ ingredientIndex === null ? 'Add new' : 'Edit'}} ingredients</h4>

...

<button *ngIf="ingredientIndex === null" (click)="addIngredient()"
        [disabled]="isDisabled()" class="btn btn-primary">
   Add ingredient
</button>
<button *ngIf="ingredientIndex !== null" (click)="updateIngredient()"
        [disabled]="isDisabled()" class="btn btn-primary">
   Update ingredient
</button>
<button *ngIf="ingredientIndex !== null" (click)="deleteIngredient()"
        [disabled]="isDisabled()" class="btn btn-outline-primary">
   Delete ingredient
</button>
```

Opdracht 6.5 – Ingredient-add component afmaken

De laatste stap is om onder het pad ingredients/edit de mogelijkheid te bieden om ingrediënten uit de lijst te wijzigen en te verwijderen.

a. In de ingredientsService moeten twee nieuwe methodes worden toegevoegd, die deze functionaliteit kunnen aanbieden. Eerst moeten we een `getIngredient()` toevoegen, die met een index als input parameter een enkel Ingredient kan teruggeven:

```
getIngredient(index: number): Ingredient {
   return this.ingredients
      ? this.ingredients.slice(index, index + 1)[0]
      : null;
}
```

b. De tweede methode die we nodig hebben in de ingredientsService is een `updateIngredients()`, met `index` en optioneel `ingredient` als input parameters (optioneel wordt aangegeven met `?`). Wanneer geen ingredient is opgegeven, moet het item in de lijst met hetzelfde index nummer worden verwijderd, en anders moet het item met hetzelfde index nummer worden vervangen door het meegegeven ingredient. Voor zowel vervangen als verwijderen wordt eerst de lijst in de service bijgewerkt, en wordt daarna een PUT request gemaakt, om de lijst in de database ook bij te werken:

```
updateIngredients(index, ingredient?): void {
   if (ingredient) {
      this.ingredients[index] = ingredient;
   } else {
      this.ingredients.splice(index, 1);
   }
   this.httpClient.put<Ingredient[]>(
      'https://ordina-fontys-workshop.firebaseio.com/ingredients.json',
      ingredient
   ).subscribe((response) => {
      this.ingredientsChanged.next(response);
   });
}
```

c. Deze methodes kunnen nu vanuit het ingredient-add component worden aangeroepen. Eerst in de OnInit functie, binnen de if-loop van de `params.subscribe()` wanneer een params[‘id’] beschikbaar is:

```
this.ingredientIndex = +params['id'];
const ingredient =
    this.ingredientsService.getIngredient(this.ingredientIndex);
if (ingredient) {
    this.ingredientName = ingredient.name;
    this.ingredientAmount = ingredient.amount;
} else {
    this.router.navigate(['ingredients']);
}
```

Let op dat in de else-loop `router.navigate()` wordt aangeroepen; wanneer geen id beschikbaar is (bijvoorbeeld bij verversen van de pagina) wordt de gebruiker terug naar de Ingredients pagina gestuurd.

d. Tenslotte worden twee nieuwe methodes toegevoegd, namelijk `deleteIngredient()` en `updateIngredient()`, welke allebei de IngredientsService aanroepen en daarna terug navigeren naar de Ingredients pagina. Deze methodes worden zelf aangeroepen door middel van de `<button>` elementen die al eerder in de html zijn toegevoegd (zie opdracht 6.4.c):

```
deleteIngredient(): void {
   this.ingredientsService.updateIngredients(this.ingredientIndex,
      null);
   this.router.navigate(['ingredients']);
}

updateIngredient(): void {
   const newIngredient =
      new Ingredient(this.ingredientName, this.ingredientAmount);
   this.ingredientsService.updateIngredients(this.ingredientIndex,
      newIngredient);
   this.router.navigate(['ingredients']);
}
```