

## DeepDive Monteban – deel 3.1

### Benodigdheden:

- Een webserver (eventueel lokaal) waarop PHP en een Database-applicatie kan draaien;
  - o Tip: XAMPP voor een localhost
    - Bevat Apache (server applicatie die PHP ondersteund)
    - Bevat phpmyadmin (MYSQL-database)
    - Bestanden in htdocs zetten (zie een van de screenshots bijvoorbeeld)
- Node.js
  - o Draait lokaal
  - o Installeer daarop angular en voeg enkele modules toe, zoals @angular/http
- winRAR/winZIP/7z om de bestanden uit te kunnen pakken
  - o In het .rar bestand zit ook een leesmij.txt met waar je wat moet uitpakken ☺



Allereerst veel plezier met deze speeltuin en hopelijk heb je er wat aan!!

---

### INTRODUCTIE

Welkom bij het derde deel van de tofste DeepDive. In dit deel zal je vast en zeker dingen tegen komen waarvan je denkt, *dit kan ik al*. Dat is niet erg, het is goed om te herhalen. Wat gaan we exact doen in dit deel?

We gaan een administratie-applicatie maken (of in zekere zin opnieuw opbouwen) voor <https://www.interndepion.nl>. Momenteel is deze ontwikkelt (of ontwikkeld?) in PHP en totaal niet robuust opgebouwd. Sterker nog, er is nog geen eens gebruik gemaakt van OOP. Een doodzonde.

De applicatie bestaat uit drie delen, zie onderstaand screenshot

#### Admin Pagina

A screenshot of a web application's admin page. It features a list of players with their IDs and names, a 'Dr Nice bevesten' button, and sections for 'Afmeldingen' and 'Scores'.

Spelers	
11	Dr.Nice
12	Narco
13	Bombasto
14	Celeritas
15	Magneta
16	RubberMan
17	Dynama
18	Dr.IQ
19	Magma
20	Tornado

Dr Nice bevesten

11

---

Afmeldingen

---

Scores

---

---

1 Dit screenshot toont alleen de opbouw, niet de functionaliteit. De delen zijn dus: Spelers, Afmeldingen en Scores

In de administratiepagina willen we de volgende functionaliteit hebben:

- Details van de spelers aanpassen (naam en e-mailadres bijvoorbeeld);
- Afmeldingen van spelers corrigeren als de nieuwe datum nog niet bevestigd is;
- Scores van wedstrijden bevestigen of corrigeren.

Bij deze DeepDive zal een databestand worden verstrekt met basale afmeldingen en scores. We gaan daarnaast gebruik maken van een database waar onze applicatie connectie mee moet maken. Die connectie gaan we zelf bouwen. Dat is dan ook de eerste stap van deze DeepDive.

## HOOFDSTUK 1 - REST API

Omdat we tussen verschillende applicaties zullen moeten communiceren (we hebben onze database en de applicatie die de data er uit haalt én de applicatie die onze administratie-applicatie laat zien), hebben we een API nodig. We maken een REST API. REST staat voor REpresentational State Transfer. Kortweg hoe we onze data representeren aan de clientside van de applicatie.

Onze API zal diverse JSON-objecten gaan genereren die we vervolgens kunnen gebruiken in de administratie-applicatie.

Omdat je niet alle tijd van de wereld zult hebben, is een groot deel van onze API toegevoegd aan de DeepDive. Superwicked dat dit gelukt is, want dit was iets wat al een lange tijd op ons wensenlijstje stond.

De API heeft (uiteindelijk) de volgende functionaliteiten, wellicht kun jezelf meer functionaliteit bedenken!:

- Spelers uitlezen
- Spelers wijzigen
- Afmeldingen uitlezen
- Afmeldingen wijzigen
- Afmeldingen aanmaken
- Scores uitlezen
- Scores aanpassen

Spelers bestaan uit de volgende elementen:

- id
- naam
- code (een wachtwoord)
- email
- knsb\_id (het nummer van de speler bij de schaakbond)

Afmeldingen bestaan uit de volgende elementen:

- id
- pid (partij id)
- afmelder (het knsb\_id)
- datum (datum waarop de afgemelde wedstrijd eventueel gespeeld zal worden)
- bevestig (een uniek nummer op het moment dat gevuld wordt als de tegenstander de nieuwe datum bevestigd)
- bevestigd (indien bevestigd wordt dit 1 anders 0)

Scores (en daarmee in feite ook partijen) bestaan uit de volgende elementen (opgebouwd vanuit een andere applicatie en aangevuld voor deze applicatie):

- Round
- Board
- wit (knsb\_id van witte speler)
- zwart (knsb\_id van zwarte speler)
- NoW (nr van wit in het speelschema)
- NoB (nr van zwart in het speelschema)
- ResW (score van wit)
- ResB (score van zwart)
- Res (totale score-overzicht van de partij)
- Mno (gevuld met knsb\_id van de afmelder indien afgemeld)
- id

Je kunt de code van de API rustig doorlezen. Het is voorzien van comments. De API bouwt de data uit de database, met behulp van PHP, om naar JSON-objecten. Ter lering en vermaak zal hieronder wél een van de delen van de API uitgelegd worden.

#### HOOFDSTUK 1.1 – STRUCTUUR EN BASISFUNCTIONALITEIT VAN DE API

Naam	Gewijzigd op	Type	Groot
afmeldingen	6-7-2019 21:46	Bestandsmap	
config	6-7-2019 20:48	Bestandsmap	
objects	6-7-2019 21:30	Bestandsmap	
spelers	6-7-2019 21:00	Bestandsmap	

Zo ziet de map structuur van de api er uit (let wel, scores is nog niet gemaakt).

Uit de structuur blijkt al dat ieder element zijn eigen map krijgt. Maar er zijn twee algemene mappen, het config-mapje en het objects-mapje. Config bevat een bestand waar in connectie gelegd wordt met de database.

```

<?php
class Database{

    // specify your own database credentials
    private $host = "localhost";
    private $db_name = "api_db";
    private $username = "root";
    private $password = "";
    public $conn;

    // get the database connection
    public function getConnection(){

        $this->conn = null;

        try{
            $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db_name, $this->username, $this->password);
            $this->conn->exec("set names utf8");
        }catch(PDOException $exception){
            echo "Connection error: " . $exception->getMessage();
        }

        return $this->conn;
    }
}
?>

```

De connectie an sich is niet zo spannend. Het is vrij standaard en bevat gegevens waar onze database gehost is, hoe hij heet en hoe je er bij mag komen. In mijn geval is dat dus localhost (ik draai alles lokaal tijdens het ontwikkelen door middel van XAMPP (google maar eens als je dat nog niet hebt gedaan 😊)).

In het mapje objects definiëren we onze objecten. Momenteel waren dat dus:

Afmeldingen

Spelers

Afmeldingen ziet er bijvoorbeeld als volgt uit:

```

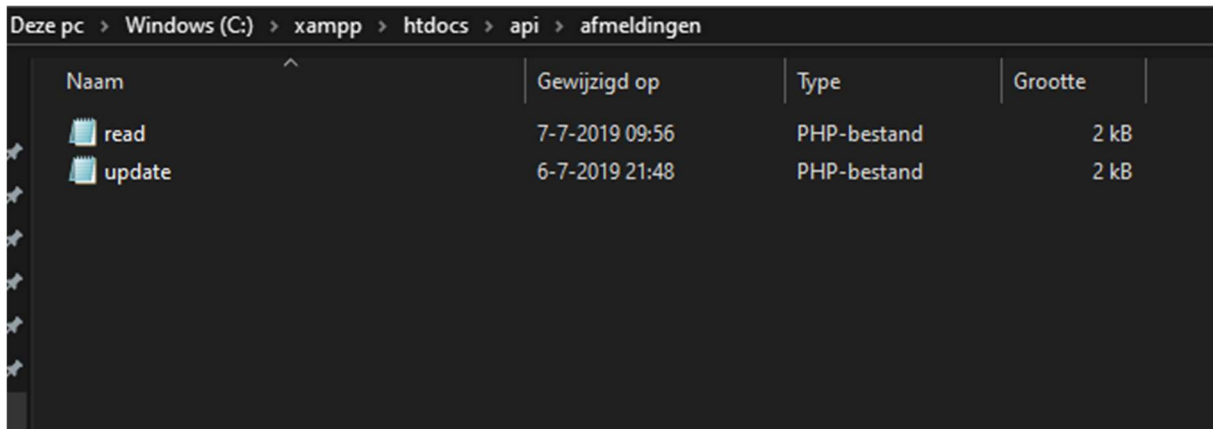
1 <?php
2 class Afmeldingen{
3     // database connection and table name
4     private $conn;
5     private $table_name = "afmeldingen";
6
7     // object properties
8     public $id;
9     public $pid;
10    public $afmelder;
11    public $datum;
12    public $bevestig;
13    public $bevestigd;
14    public $wit;
15    public $zwart;
16
17    // constructor with $db as database connection
18    public function __construct($db){
19        $this->conn = $db;
20    }
21
22    // read afmeldingen (and think of something clever... join it on partijen to get wit and zwart more easily then in the system now)
23    function read(){
24
25        // select all query
26        $query = "SELECT
27            a.id as id, a.pid, a.afmelder, a.datum, a.bevestig, a.bevestigd, (SELECT naam FROM spelers WHERE knab_id = p.wit) as wit, (SELECT naam FROM spelers WHERE knab_id = p.zwart)
28            FROM
29                ". $this->table_name . " a
30            INNER JOIN partijen p
31            WHERE a.pid = p.id
32            ";
33
34        // prepare query statement
35        $stmt = $this->conn->prepare($query);
36
37        // execute query
38        $stmt->execute();
39
40        return $stmt;
41    }
42
43    // update the afmelding
44    function update(){
45
46        // update query, be aware of the fact you do not need to update the wit, zwart-elements.
47        $query = "UPDATE
48            ". $this->table_name . "
49            SET
50                pid = :pid,
51                afmelder = :afmelder,
52                datum = :datum,
53                bevestig = :bevestig,
54                bevestigd = :bevestigd
55            ";

```

Afmeldingen is een klasse waar aan enkele elementen worden gekoppeld (zie de opsomming hierboven) en enkele functies. Een van de functies is de read-functie. Daarmee lezen we alles uit uit de database. Zodra we dus read aanroepen, krijgen we een dataset die de elementen vult.

## HOOFDSTUK 1.2 – HET JSON-OBJECT

Maar met alleen deze dataset kunnen we niks, voor iedere functie converteren we het naar een JSON-object om er in de frontend wel iets mee te kunnen.



Naam	Gewijzigd op	Type	Grootte
read	7-7-2019 09:56	PHP-bestand	2 kB
update	6-7-2019 21:48	PHP-bestand	2 kB

In de afmeldingen-map hebben twee bestanden die ieder een van de functies van onze klasse aanroept.

```

// instantiate database and afmeldingen object
$database = new Database();
$db = $database->getConnection();

// initialize object
$afmelding = new afmeldingen($db);

// query products
$stmt = $afmelding->read();
$num = $stmt->rowCount();

// check if more than 0 record are found
if($num>0){

    // afmeldingen array
    $afmeldingen_arr=array();
    $afmeldingen_arr["records"]=array();

    // retrieve our table contents
    // fetch() is faster than fetchAll()
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
        // because we are lazy, we extract $row to use $id instead of $row['id']
        extract($row);

        // now add our data of the database to an item
        $afmelding_item=array(
            "id" => $id,
            "pid" => $pid,
            "afmelder" => $afmelder,
            "datum" => $datum,
            "bevestig" => $bevestig,
            "bevestigd" => $bevestigd,
            "wit" => $wit,
            "zwart" => $zwart
        );

        // push the item to the array
        array_push($afmeldingen_arr["records"], $afmelding_item);
    }

    // set response code - 200 OK
    http_response_code(200);

    // show data in json format
    echo json_encode($afmeldingen_arr);
}

```

Dit is het functionele stuk van het read-bestand. Allereerst roepen we onze database aan en koppelen we deze aan onze afmeldingen-klasse zodat de query die we uitvoeren ook daadwerkelijk uitgevoerd kan worden op de database.

De query zit in de functie read() die we aanroepen en vervolgens willen we weten of wel een resultaat terug hebben gehad. Dat is zo als, een standaardfunctie, rowCount(), een resultaat van meer dan 0 oplevert.

Vervolgens willen we onze resultaat gaan ombouwen naar het JSON-object dat we kunnen aanroepen in de frontend-applicatie. Het JSON-object is een array dus moeten we al onze rijen in de array gaan zetten. Per item doen we dat en bepalen we de namen in de array. Zolang we nieuwe resultaten in \$row krijgen, stoppen we in de totale array genaamd "records" onze items.

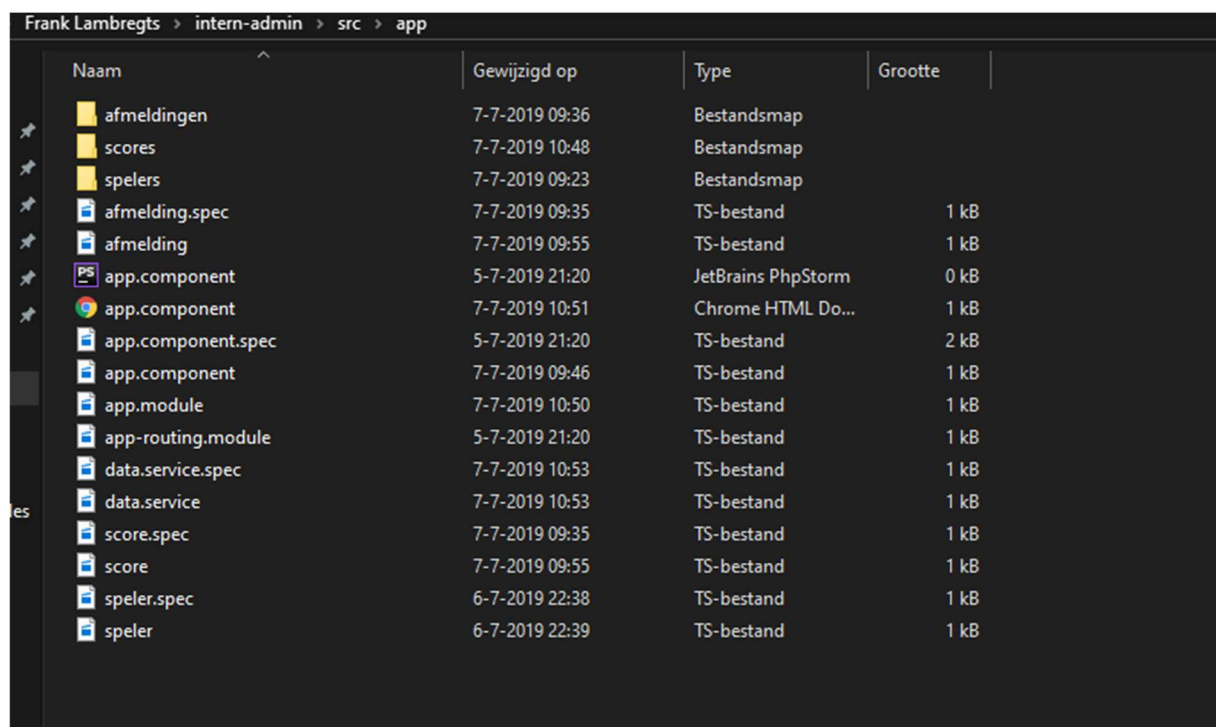
Daarna willen we de gevulde array omzetten in het JSON-object. Dat kan gelukkig met een standaardfunctionaliteit: `json_encode`. Het wordt gecodeerd als JSON-object en levert dan bijvoorbeeld dit op:

```
{ "records": [{"id": "34", "pid": "74", "afmelder": "8090687", "bevestig": "2019-03-22", "bevestigd": "1", "wit": "Erik van Elven", "zwart": "Frank Lambregts"}, {"id": "35", "pid": "75", "afmelder": "8090687", "bevestig": "2019-03-08", "bevestigd": null, "wit": "Frank Lambregts", "zwart": "Erik van Elven"}]}
```

We zien de totale array-naam terug, "records". Deze heeft als elementen verschillende aparte arrays met terugkerende namen. Die hebben we immers zo gedefinieerd. Dit formaat array is prima te lezen door onze frontend en komt je vast en zeker ook bekend voor, immers hebben we eerder met een array gewerkt met angular. Het is nu alleen nog de vraag, hoe krijgen we het daar in?

## HOOFDSTUK 2 – ANGULAR APPLICATIE

Voordat we in detail treden, eerst even een overzicht hoe de Angular-applicatie uiteindelijk opgebouwd zal zijn.



Naam	Gewijzigd op	Type	Grootte
afmeldingen	7-7-2019 09:36	Bestandsmap	
scores	7-7-2019 10:48	Bestandsmap	
spelers	7-7-2019 09:23	Bestandsmap	
afmelding.spec	7-7-2019 09:35	TS-bestand	1 kB
afmelding	7-7-2019 09:55	TS-bestand	1 kB
app.component	5-7-2019 21:20	JetBrains PhpStorm	0 kB
app.component	7-7-2019 10:51	Chrome HTML Do...	1 kB
app.component.spec	5-7-2019 21:20	TS-bestand	2 kB
app.component	7-7-2019 09:46	TS-bestand	1 kB
app.module	7-7-2019 10:50	TS-bestand	1 kB
app-routing.module	5-7-2019 21:20	TS-bestand	1 kB
data.service.spec	7-7-2019 10:53	TS-bestand	1 kB
data.service	7-7-2019 10:53	TS-bestand	1 kB
score.spec	7-7-2019 09:35	TS-bestand	1 kB
score	7-7-2019 09:55	TS-bestand	1 kB
speler.spec	6-7-2019 22:38	TS-bestand	1 kB
speler	6-7-2019 22:39	TS-bestand	1 kB

De applicatie heeft als naam gekregen intern-admin. Zoals je weet bestaat de Angular-applicatie uit drie componenten, deze componenten zitten in de mappen.

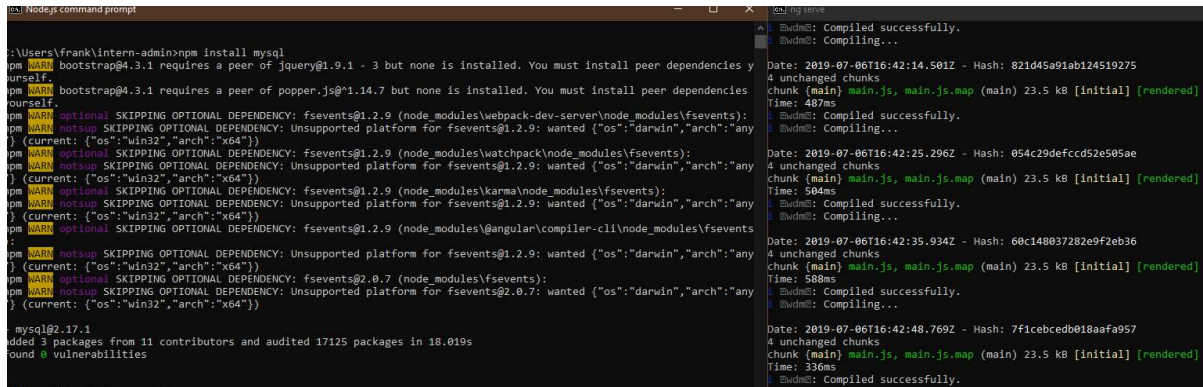
De bestanden `afmelding`, `score` en `speler` zijn slechts de definitie wat een afmelding, score en speler is. Zo is een speler:

```
export class Speler {
  constructor(
    public id: number,
    public naam: string,
    public code: string,
    public email: string,
    public knsb_id: number
  ) {}
}
```

Je hebt daarnaast de standaard zaken als `app.module` en `app.component`. De hoofdbestanddelen. Daarnaast zie je een `data.service`-bestand. Dat is het nieuwe stukje.

## HOOFDSTUK 2.1 - SERVICES

Een Angular-applicatie bestaat uit verschillende componenten, dat zijn de eigen bouwstenen. Maar onze applicatie wilt gebruik maken van bouwstenen die door een andere applicatie zijn gemaakt. Er wordt een dienst aangeleverd, services. Ken je onze commandline nog?



```
Users\frank\Intern-admin>npm install mysql
pm WARN bootstrap@4.3.1 requires a peer of jquery@1.9.1 - 3 but none is installed. You must install peer dependencies yourself.
pm WARN bootstrap@4.3.1 requires a peer of popper.js@^1.14.7 but none is installed. You must install peer dependencies yourself.
pm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\webpack-dev-server\node_modules\fsevents):
pm WARN node-sass SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
pm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\karma\node_modules\fsevents):
pm WARN node-sass SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
pm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\angular\compiler-cli\node_modules\fsevents):
pm WARN node-sass SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
pm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.0.7 (node_modules\fsevents):
pm WARN node-sass SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.0.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

mysql@2.17.1
added 3 packages from 11 contributors and audited 17125 packages in 18.019s
found 0 vulnerabilities

ng serve
Date: 2019-07-06T16:42:14.501Z - Hash: 821d45a91ab124519275
4 unchanged chunks
chunk (main) main.js, main.js.map (main) 23.5 kB [initial] [rendered]
Time: 487ms
Bundle: Compiled successfully.
Bundle: Compiling...
Date: 2019-07-06T16:42:25.296Z - Hash: 054c29defcccd52e505ae
4 unchanged chunks
chunk (main) main.js, main.js.map (main) 23.5 kB [initial] [rendered]
Time: 504ms
Bundle: Compiled successfully.
Bundle: Compiling...
Date: 2019-07-06T16:42:35.934Z - Hash: 60c148037282e9f2eb36
4 unchanged chunks
chunk (main) main.js, main.js.map (main) 23.5 kB [initial] [rendered]
Time: 580ms
Bundle: Compiled successfully.
Bundle: Compiling...
Date: 2019-07-06T16:42:48.769Z - Hash: 7f1cecbcd0b18aafa957
4 unchanged chunks
chunk (main) main.js, main.js.map (main) 23.5 kB [initial] [rendered]
Time: 336ms
Bundle: Compiled successfully.
```

Die hebben we weer nodig. In de DeepDive-map zit alles al behalve het servicebestand. Die mag je nog maken via de console. Maar hoe doen we dat?

### Opgave 1

*ng g service [naam]*

In de `app`-folder van ons Angular-project komen twee bestanden; `[naam].service.spec` (niet zo boeiend aangezien dat puur een testbestand is) & `[naam].service`. Ik heb hem dus `data` genoemd, immers de service zal ons van `data` kunnen voorzien.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class TestService {

  constructor() { }
}
```

Dit is wat je ongeveer zult zien. Dat doet natuurlijk niets. Het enige wat het momenteel doet is zeggen dat de service in de basis van onze applicatie aanwezig moet zijn.

Wat moet onze service voor de rest doen? Hij moet onze bedienen van informatie vanuit de backend. Dat is de API welke we gaan kunnen benaderen door te verwijzen naar

[http://localhost/api/\[object\]/\[functie\].php](http://localhost/api/[object]/[functie].php)

Het is belangrijk om dit te bedenken, want er zit cruciale informatie in die url. We hebben `data` klaar staan in een pagina die door middel van HTTP naar ons toekomt. Angular heeft momenteel een vrij complex systeem om daar mee om te gaan, maar gelukkig zijn we niet de enigen op de wereld en zijn er packages beschikbaar om de boel wat te versimpelen.

Via de console installeren eerst we die package; `npm install @angular/http`



## HOOFDSTUK 2.2 – DE SERVICE

Nu kunnen we echt aan de slag om onze service op te bouwen.

```
import { Injectable } from '@angular/core';
import { Http, Response, Headers, RequestOptions } from '@angular/http';
import { Observable } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { Speler } from './speler';
import { map } from 'rxjs/operators';
import { Afmelding } from './afmelding';
import { Score } from './score';

@Injectable()
export class DataService {
  // We need Http to talk to a remote server.
  constructor(private _http : Http) { }

  // Get list of spelers from remote server.
  leesSpelers(): Observable<Speler[]>{
    return this._http
      .get("http://localhost/api/spelers/read.php")
      .pipe(map((res: Response) => res.json()));
  }

  // Get list of afmeldingen from remote server.
  leesAfmeldingen(): Observable<Afmelding[]>{
    return this._http
      .get("http://localhost/api/afmeldingen/read.php")
      .pipe(map((res: Response) => res.json()));
  }

  // Get list of scores from remote server.
  leesScores(): Observable<Score[]>{
    return this._http
      .get("http://localhost/api/scores/read.php")
      .pipe(map((res: Response) => res.json()));
  }
}
```

Het screenshot laat zien wat de service in de basis wordt. Een aantal zaken zijn gehighlight, die zijn het belangrijkste.

Allereerst de import vanuit de zojuist geïnstalleerde package, we halen daarmee enkele mogelijkheden op, zoals het kunnen uitlezen van een http-url, maar ook het, straks verzenden van verzoeken naar zo'n http-url.

Daarnaast hebben we Observable geïmporteerd.

### HOOFDSTUK 2.2.1 - Observables

Een observable is eigenlijk niets anders dan er voor zorgen dat data die we gebruiken continue ge-update wordt. Stel er wordt door twee personen hetzelfde object gemuteerd, dan wil je niet dat als X het e-mailadres aanpast en Y de naam aanpast, een van de wijzigingen wordt overschreven. X ziet de wijziging van Y wellicht niet, maar onze applicatie wel en slaat hem op en, op de achtergrond, overschrijft de data die X muteert. Resultaat, ook al ziet X niet dat de naam anders wordt, als hij op opslaan drukt, wordt niet de oude naam die hij nog zag, opgeslagen.

### HOOFDSTUK 2.2.2 - Constructor & leesSpelers()

Om met onze API verbinding te leggen, definiëren we eerst dat `_http` (of iedere andere willekeurige naam) een object is van het geïmporteerde `Http`. Vervolgens gaan we onze data lezen.

We zullen straks een functie hebben die `leesSpelers` heet, die bevat een `Observable`-array van `Speler`

(dus een array waarin een Speler zit (of meerdere Spelers 😊) en die door meerdere partijen tegelijk bereikbaar moet zijn).

De functie moet, als we hem aanroepen, onze leesfunctie van de API aanroepen.

We gebruiken daarvoor `.get`. En wat getten we, een JSON-object. Daar hadden we de informatie namelijk naar geconverteerd. We verwachten dus als Response van de gekregen informatie een JSON-object.

Dat dat ook echt zo is, hebben we overigens in de API bepaald 😊

```
<?php
// required headers to make it a json object
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
```

### **Hoofdstuk 2.2.2.1 - Intermezzo**

Het HTTP-protocol kent verschillende methoden, de belangrijkste, en verreweg meest gebruikte zijn GET en POST. GET betekent dat we informatie willen krijgen, POST is dat we informatie willen versturen, bijvoorbeeld bij een inlogformulier, dan post je een gebruikersnaam en wachtwoord naar een server waar het verwerkt wordt.

Met GET kun je bijvoorbeeld ook een url als

<https://www.facebook.com/franklambregts?ref=bookmarks> uitlezen. Het stukje na het vraagteken is informatie voor een GET-functionaliteit. Hoe ben ik op mijn facebook-timeline gekomen? Blijkbaar via bookmarks, die informatie kan geGET worden.

### **HOOFDSTUK 2.2.3 – Benodigheden**

Onze service moet natuurlijk wel verweven zitten in de applicatie. Dat gaat gebeuren in de verschillende componenten. Hier is een screenshot van het spelers-component:

```

import { Component, OnInit } from '@angular/core';
import { DataService } from 'src/app/data.service';
import { Observable } from 'rxjs';
import { Speler } from 'src/app/speler';

@Component({
  selector: 'app-spelers',
  templateUrl: './spelers.component.html',
  styleUrls: ['./spelers.component.css'],
  providers: [DataService]
})
export class SpelersComponent implements OnInit {

  selectedSpeler: Speler;
  // store list of spelers
  spelers: Speler[];

  // initialize SpelersService to retrieve list service in the ngOnInit()
  constructor(private spelersService: DataService) { }

  // methods to be created later on
  maakSpeler() {}
  leesEenSpeler(id) {}
  updateSpeler(id) {}
  verwijderSpeler(id) {}

  ngOnInit() {
    this.spelersService.leesSpelers()
      .subscribe(spelers =>
        this.spelers=spelers['records']
      );
  }

  onSelect(speler: Speler): void{
    this.selectedSpeler = speler;
  }
}

```

We importeren naast de standaard imports, onze service, opnieuw de observable én onze Speler, want anders weet onze component waar een speler uit bestaat.

Daarnaast hebben we nu een provider. In het @Component deel, vertellen we ons component dat de DataService ons data toespeelt. Het is een provider.

Daarnaast hebben we in dit component (en ook de anderen) gedefinieerd wat we met leesSpelers() doen. leesSpelers bevatte een observable. Aan een observable moet eerst worden deelgenomen door gebruik te maken van subscribe. Je zegt als het ware, ja ik wil geobserveerd kunnen worden. We kunnen daar ook nog allerlei operaties aan toevoegen. We kunnen filteren zodat bijvoorbeeld alleen spelers met een id > 5 geobserveerd kunnen worden of zeggen dat alle namen voorafgegaan moeten worden door "Dhr." Door gebruik te maken van mapping. De syntax daarvan is als (bijvoorbeeld) volgt:

```

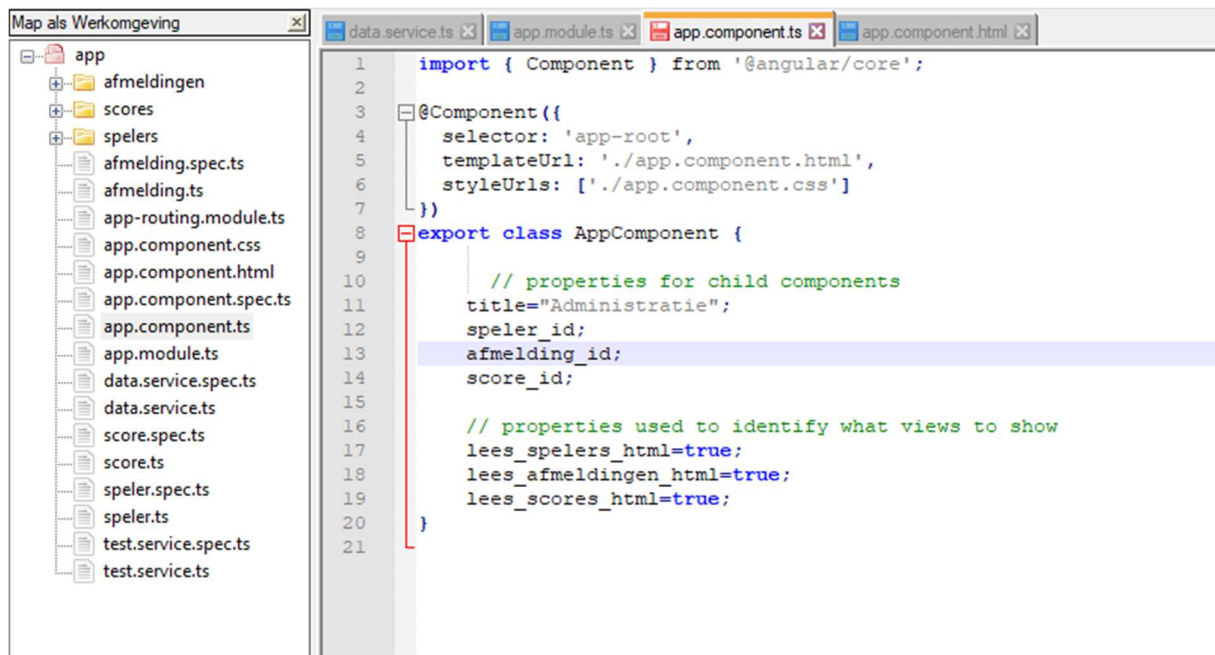
.filter((person) => person.id > 5)
.map((person) => "Dr. " + person.name)

```

Waarbij person uiteraard veranderd moet worden.

## HOOFDSTUK 2.3 – DE APPLICATIE-OPBOUW

We hebben nu de functionaliteit van de service gehad, maar hoe komt terug in de applicatie? De twee hoofdbestanddelen van de applicatie zien er als volgt uit:



```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9
10    // properties for child components
11    title="Administratie";
12    speler_id;
13    afmelding_id;
14    score_id;
15
16    // properties used to identify what views to show
17    lees_spelers_html=true;
18    lees_afmeldingen_html=true;
19    lees_scores_html=true;
20 }
21
```

Voor de volledigheid definieer we de verschillende ids (gaan we later nog iets meedoen) én we definiëren lees\_spelers\_html en de anderen als true! Waarom zul je je afvragen?



```
1 <div>
2   <h1>
3     {{ title }}
4   </h1>
5 </div>
6 <ul>
7
8   <app-spelers *ngIf="lees_spelers_html"></app-spelers>
9   <app-afmeldingen *ngIf="lees_afmeldingen_html"></app-afmeldingen>
10  <app-scores *ngIf="lees_scores_html"></app-scores>
11 </ul>
12
13 <router-outlet></router-outlet>
14
15
16
```

Dit is de html-pagina van de applicatie. Iedere tag heeft een bijzonder if-statement in de tag. Binnen de tag kunnen we if-statements maken waardoor de tag en het geen wat er binnen staat alleen zichtbaar is als het if-statement true is.

In de applicatie is ook een Else-element beschikbaar. In de html-pagina van het Afmeldingen-component is dit opgenomen:

```

<!-- depending on the value of selectedAfmelding.bevestigd, show text -->
<div *ngIf="selectedAfmelding.bevestigd == 1; else ElseBlock">bevestigd.</div>
<!-- Only if not confirmed, the Afmelding can be changed or deleted. -->
<ng-template #ElseBlock>niet bevestigd.
<p>
  <!-- edit Afmelding button -->
  <a (click)="updateAfmelding(afmelding.id)" class='btn btn-info m-r-5px'>
    <span class='edit'> Wijzig Afmelding</span>
  </a>

  <!-- delete Afmelding button -->
  <a (click)="verwijderAfmelding(afmelding.id)" class='btn btn-danger m-r-5px'>
    <span class='edit'> Verwijder Afmelding (kan nog niet)</span>
  </a></p></ng-template>

<p>
  <!-- read one Afmelding button -->
  <a (click)="leesEenAfmelding(afmelding.id)" class='btn btn-primary m-r-5px'>
    <span class='edit'> Lees 1 Afmelding (kan nog niet)</span>
  </a>
</p>
</div>

```

Afhankelijk van de status van een Afmelding laten we een tekst zien. Het is daarmee heel eenvoudig om delen wel of niet te laten zien.

Naast de ApplicatieComponent, hebben we ook de Applicatie Modules. Hierin worden alle andere componenten gekoppeld.

```

1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { HttpClientModule } from '@angular/http';
4
5  import { AppRoutingModule } from './app-routing.module';
6  import { AppComponent } from './app.component';
7  import { AfmeldingenComponent } from './afmeldingen/afmeldingen.component';
8  import { ScoresComponent } from './scores/scores/scores.component';
9  import { SpelersComponent } from './spelers/spelers/spelers.component';
10
11  @NgModule({
12    declarations: [
13      AppComponent,
14      AfmeldingenComponent,
15      ScoresComponent,
16      SpelersComponent
17    ],
18    imports: [
19      BrowserModule,
20      AppRoutingModule,
21      HttpClientModule
22    ],
23    providers: [],
24    bootstrap: [AppComponent]
25  })
26  export class AppModule { }
27

```

## HOOFDSTUK 3 – EN NU VERDER?

Als we op basis van bovenstaand alles draaiend hebben, krijgen we dit (bij het maken van dit screenshot was de API nog niet uitgebreid naar scores, ondertussen wel, daarnaast is de opmaak enigszins aangepast en zijn e-mailadressen gestandaardiseerd):

### Administratie

Spelers Nieuwe Speler (kan nog niet)

1	Frank Lambregts
4	René van den Broek
5	Ted van Eck
6	Laurens Quinten
7	Pim Eerens
8	Ivo Kok
9	Koen Riemens
10	Lewon Gevoorkjan
11	Bas Robben
12	Erik van Elven
13	Nieuwe Speler

afmeldingen Nieuwe Afmelding (kan nog niet)

34	Erik van Elven - Frank
35	Frank Lambregts - Erik van Elven

Scores Nieuwe Score (kan nog niet)

En als we dan op een speler klikken, verschijnt:

Nieuwe Speler bewerken

ID Naam E-mail KNSB-ID

13 Nieuwe Speler nieuwespeler@gmail.com 123123

Lees 1 speler (kan nog niet)

Wijzig Speler

Verwijder Speler (kan nog niet)

Maar we kunnen hier nog niks meedoen. Dat was wel de bedoeling.

### Opdracht 2

Maak een formulier waarmee een speler daadwerkelijk bewerkt kan worden.

Een klein beetje op weg heb ik je al geholpen door het formulier in ieder geval te in html te maken.

*Tips en hints mogen gevraagd worden.*

*Lever Opdracht 2 in ter beoordeling. Deel 3.2 van DeepDiveMonteban volgt na het inleveren van Opdracht 2.*

Succes!