

Project 2: WorkOut App

Requirements

Deze workout app is een planning voor fitness oefeningen om binnen de 5 minuten 10 oefeningen achtereenvolgens uit te voeren. Dus we moeten 10 oefeningen voorzien waarbij elke oefening 30 seconden duurt, gevolgd door een rust periode. Na de rust wordt automatisch de volgende oefening gestart.

- De workout starten
- Voorzie step-by-step instructies hoe men de oefening moet uitvoeren
- Een timer om te kijken hoeveel tijd nog rest voor de oefening
- Notificatie aan de gebruiker wanneer de oefening gedaan is, en automatisch naar de volgende oefening overgaan


Om de juiste taal te spreken is een workout een serie oefeningen uitgevoerd in een bepaalde volgorde met een bepaalde tijdsduur.

Een oefening heeft:

- Naam
- Titel (aan de gebruiker getoond)
- Beschrijving
- Instructies

Model

Uit de requirements halen we ons model maken:

```
 function Oefening(args) {  
    this.naam = args.naam;  
    this.titel = args.titel;  
    this.beschrijving = args.beschrijving;  
}  
  
function WorkOutPlan(args) {  
    this.naam = args.naam;  
    this.oefeningen = [];  
    this.titel = args.titel;  
    this.RustTussenOefeningen = args.RustTussenOefeningen;  
}
```

(Eigenlijk hebben we in deze oefening misschien geen WorkOutPlan nodig, omdat we maar 1 workoutplan implementeren met daarin enkele oefeningen, dus hebben we met de klasse Oefening meer dan genoeg. Maar voor de toekomst kunnen we het met de klasse WorkOutPlan makkelijker uitbreiden met nieuwe plannen en oefeningen.

We starten met onze app.js file als onze hoofdmodule, maar maken hier ook een tweede module aan "workout" om ons programma overzichtelijker te maken:

1. `angular.module("app", ["workout"]);`
2. `angular.module("workout", []);`

In regel 1 maken we gebruik van de constructie `["workout"]` wat een dependency injection is.

Bouwen van de workout controller

Welke functionaliteit moet de controller bevatten? Allereerst moeten we een workout kunnen starten. Nadat een oefening uit het workoutplan gedaan is moet de volgende oefening opgestart kunnen worden (na elke oefening voorzien we een rust periode!). En dit proces moet herhaald worden zolang er oefeningen in het workout plan zitten.

Maak een javascript bestand: workout.js

```
angular.module("workout").controller("WorkOutCtrl", ['$scope', function ($scope)
{
    var rustOefening;
    var workoutPlan;

    var init = function ()
    {
        console.log("init functie");
        startWorkOut();
    }
}
```

In de init functie starten we de startWorkOut. Hier gaan we eerst een workoutplan aanmaken en uiteindelijk dit plan met hierin de oefeningen te starten.

```
### startWorkOut

var startWorkOut = function ()
{
    workoutPlan = CreateWorkOut();
    console.log(workoutPlan is aangemaakt);
    rustOefening =
    {
        details: new Oefening({
            naam: "Rust",
            beschrijving: "Rust een beetje",
            titel: "Rust!"
        }),
        duration: 10
    }
}
```

```
};

//Start Oefening
startOefeningen(workoutPlan.oefeningen.shift());
}
```

Je merkt dat in de startWorkOut ook een rustOefening object wordt aangemaakt. Meer info hoe dit object wordt aangemaakt op:

https://developer.mozilla.org/nl/docs/Web/JavaScript/Reference/Operators/Object_initializer
(https://developer.mozilla.org/nl/docs/Web/JavaScript/Reference/Operators/Object_initializer) .

Structuur van een JSON object

- Data wordt georganiseerd als key-value pairs
- Documenten bevatten name-value pairs, gescheiden door een komma
- Een document start met { en eindigt met }
- Namen zijn strings: bijvoorbeeld: customer_id, adres,..
- Waarden kunnen numbers, strings, Booleans, arrays, objecten, NULL zijn.
- De waarden van een array worden opgelijst beginnende met [en eindigen met]
- De waarden van een object zijn opnieuw key-value pairs beginnend met { en eindigen met }

Dus een document is een set key-value pairs. De keys worden als strings voorgesteld, terwijl de values basic types kunnen bevatten, of structuren kunnen zijn (arrays, objecten).


Documenten bevatten zowel de structuur als de data. JSON en XML zijn 2 formaten die hiervoor vaak gebruikt worden. Meerdere documenten worden in een collectie gestopt. Collecties zijn dus een lijst van documenten. Document database designers moeten denken aan het zo snel mogelijk kunnen toevoegen, verwijderen, updaten en zoeken naar documenten. Belangrijk te weten is dat documenten binnen een collectie niet persé dezelfde structuur moeten aannemen.

Op het einde van de functie wordt de functie startOefeningen opgeroepen met als argument: workoutPlan.oefeningen.shift(). De shift methode op een array zal het eerste item uit de array verwijderen en dit item retourneren.

Ter info: Een object kan als volgt aangemaakt worden: obj = {} . Dit is een leeg object obj2 = { naam: "Peeters", voornaam: "Tom" }

obj3 = { naam: "peeters", adres: { straat: "test", woonplaats: "test" } }

CreateWorkOut

```
 var CreateWorkOut = function ()
{
    console.log("create workout");
    var workout = new WorkOutPlan(
```

```

        {
            naam: "5min workout",
            titel : "5 minute workout",
            RustTussenOefeningen: 10
        });

workout.oefeningen.push(
    {
        details: new Oefening(
            {
                naam: "oef 1 springen",
                titel: "oef 1: springen",
                beschrijving: "spring constant"
            }
        ),
        duration: 10
    },

    {
        details: new Oefening(
            {
                naam: "oef 2 pompen",
                titel: "oef 2: pompen",
                beschrijving: "pomp constant"
            }
        ),
        duration: 10
    }
);
console.log(workout);
return workout;
}

```

In de CreateWorkOut instantiëren we een nieuw object van WorkOutPlan: `var workout = new WorkOutPlan()` Als we naar de constructor kijken zien we dat we hierin een array van oefeningen inbrengen:

```

function WorkOutPlan(args) {
    this.naam = args.naam;
    this.oefeningen = [];
    this.titel = args.titel;
    this.RustTussenOefeningen = args.RustTussenOefeningen;
}

```

Met de push methode brengen we nieuwe oefeningen in.

startoefening

```

var startOefeningen = function (oefening) {
    console.log("start oefeningen")

    $scope.huidigeOefening = oefening;
    $scope.huidigOefeningDuration = 0;
}

```

```
};
```

Niet alle properties moeten aan het \$scope object gehangen worden, enkel deze die in de View getoond worden.

We kunnen de view aanmaken om te testen of we werkelijk de eerste oefening te zien krijgen:

de view - part 1.

```
<body ng-app="app">
  <div >
    <div >
      <div>
        <h1>7 Minute Workout</h1>
      </div>
    </div>
  </div>
  <div>

    <div ng-controller="WorkOutCtrl">
      <pre>Huidige Oefening: {{huidigeOefening.details | json}}</pre>
      <pre>Time Left: {{huidigOefeningDuration}}</pre>
    </div>

  </div>
</body>
</div>
<div>

  <div ng-controller="WorkOutCtrl">
    <pre>Huidige Oefening: {{huidigeOefening.details | json}}</pre>
    <pre>Time Left: {{huidigOefeningDuration}}</pre>
  </div>

</div>
</body>
</html>
```

Interval Service

Nu willen we dat na de gekregen tijdsduur van de oefening we automatisch de volgend oefening laten zien.

Bij de aanmaak van een oefening:

```
workout.oefeningen.push(
  {
    details: new Oefening(
```

```

        {
            naam: "oef 1 springen",
            titel: "oef 1: springen",
            beschrijving: "spring constant"
        })),
        duration: 10
    }
}

```

geven we een property "duration" mee die bepaald hoe lang de oefening duurt. Angular heeft een \$interval service die een wrapper is over de window.setInterval methode. Het hoofddoel van deze service is het continu oproepen van een bepaalde functie met een specifiek tijdsinterval:

```

$interval(function () {

    $scope.huidigOefeningDuration++;
},
1000,
$scope.huidigOefening.duration);

```

Hier roepen we de \$interval op en geven als eerste argument ene callback functie mee die na een tijdsinterval (dit is de tweede parameter) opgeroepen wordt en dit voor een aantal maal (dit is de derde parameter die meegegeven wordt).

Om de \$interval service te gebruiken moeten we deze injecteren in de controller:

```

angular.module("workout").controller("WorkOutCtrl", ['$scope', '$interval', function
($scope, $interval)

```

test je programma opnieuw uit (zorg ervoor dat je de \$scope.huidigOefeningDuration gelinkt hebt in de UI).

\$watch

Om de overgang naar de volgende oefening te maken moeten we de waarde van huidigeOefeningDuration volgen. Wanneer deze waarde de limiet bereikt (gemeten in de \$interval service) moeten we overgaan naar de volgende oefening. We weten dat Angular de view update wanneer het model wordt aangepast dankzij de databinding capaciteit, en meer Angular heeft een watch functionaliteit die gebruikt kan worden om veranderingen op te volgen in de controller. Deze watch functie laat ons toe om een "listener" te registreren die opgeroepen wordt wanneer de \$scope verandert (of een bepaalde waarde krijgt).

```

$scope.$watch("huidigOefeningDuration", function (waarde)
{
    if (waarde == $scope.huidigOefening.duration)
    {
        startVolgendeOefening();
    }
})

```

```

var startVolgendeOefening = function ()
{
    if ($scope.huidigeOefening == rustOefening)
        startOefeningen(workoutPlan.oefeningen.shift());
    else
        startOefeningen(rustOefening);
}

```

Test deze code uit.

Dependency Injection

Modules dienen om code te organiseren en zijn containers voor bijvoorbeeld controllers, services, directives, filters. Angular voorziet in een mechanisme om dependencies te beheren door gebruik te maken van Dependency Injection.

Het idee achter dependency injection is dat een object zijn eigen dependencies niet creëert of beheert. De dependencies worden van buitenaf voorzien.

Een voorbeeld ter verduidelijking:

```

function Component()
{
    var logger = new Logger();
}

```

De dependency is hardgecodeerd in de klasse Component. Nu laten we de dependency van buitenaf komen:

```

function Component(l)
{
    var logger = l;
}

```

De impact is groot, want door de dependency van buitenaf te voorzien hebben we de mogelijkheid om het logging gedrag te beïnvloeden zonder extra codering:

```

var v1 = new Component(new DBlogger);
var v2 = new Component( new FileLogger);

```

We hebben 2 variabelen aangemaakt met een verschillende log capaciteit zonder de Component klasse implementatie te veranderen.

<> Hoe in Angular