STRUCTUUR

```
├── client
│   ├── app                 - All of our app specific components go in here
│   ├── assets              - Custom assets: fonts, images, etc…
│   ├── components          - Our reusable components, non-specific to to our app
│
├── e2e                     - Our protractor end to end tests
│
└── server
    ├── api                 - Our apps server api
    ├── auth                - For handling authentication with different auth
strategies
    ├── components          - Our reusable or app-wide components
    ├── config              - Where we do the bulk of our apps configuration
    │   └── local.env.js    - Keep our environment variables out of source control
    │   └── environment     - Configuration specific to the node environment
    └── views               - Server rendered views
```

Client components in `client/app`

```
main
├── main.js                 - Routes
├── main.controller.js      - Controller for our main route
├── main.controller.spec.js - Test
├── main.html               - View
└── main.less               - Styles
quiz
├── quiz.js                 - Routes
├── quiz.controller.js      - Controller for our quiz route
├── quiz.controller.spec.js - Test
├── quiz.html               - View
└── quiz.less               - Styles
details
├── details.js                 - Routes
├── details.controller.js      - Controller for our details route
├── details.controller.spec.js - Test
├── details.html               - View
└── details.less               - Styles
admin
├── admin.js                - Routes
├── admin.controller.js     - Controller for our admin route
├── admin.controller.spec.js - Test
├── admin.html              - View
└── admin.less              - Styles
account
├──├login
│   ├── login.controller.js      - Controller for our login route
│   ├── login.html               - View
│
├──├settings
│   ├── settings.controller.js      - Controller for our settings route
│   ├── settings.html               - View
│
├──├signup
│   ├── signup.controller.js      - Controller for our signup route
│   ├── signup.html               - View
├──├account.js                  - Routes
```

Server components in `server/api`

```
thing
├── index.js                - Routes
├── thing.controller.js     - Controller for our `thing` endpoint
├── thing.model.js          - Database model
├── thing.socket.js         - Register socket events
└── thing.spec.js           - Test
detail
├── index.js                - 1 ▯▯1 1▯2▯▯▯Tm▯ )1▯ )-▯ )1▯ )-▯ )▯T▯▯ )▯T▯▯ )▯T▯▯▯1 ▯22▯▯/▯▯MID ▯▯
```

```
syncUpdates: function (modelName, array, cb) {
        cb = cb || angular.noop;

        /**
         * Syncs item creation/updates on 'model:save'
         */
        socket.on(modelName + ':save', function (item) {
          var oldItem = _.find(array, {_id: item._id});
          var index = array.indexOf(oldItem);
          var event = 'created';

          // replace oldItem if it exists
          // otherwise just add item to the collection
          if (oldItem) {
            array.splice(index, 1, item);
            event = 'updated';
          } else {
            array.push(item);// voegt onderaan de list toe
              //array.unshift(item);//boven aan de list
          }

          cb(event, item, array);
        });

        /**
         * Syncs removed items on 'model:remove'
         */
        socket.on(modelName + ':remove', function (item) {
          var event = 'deleted';
          _.remove(array, {_id: item._id});
          cb(event, item, array);
        });
      },
```

Server-side model
http://ebvr-ebvr.rhcloud.com/api/things

bevat bv.

details":["5693c433735c6c828472ecb9","
"

5693c433735c6c828472ecb9 =
http://ebvr-ebvr.rhcloud.com/api/details

5693c433735c6c828472ecb9

```
'use strict';

var mongoose = require('bluebird').promisifyAll(require('mongoose'));
var Schema = mongoose.Schema;


var ThingSchema = new Schema({
  name: String,
  info: String,
  password: String,
  currentQuestion: Number,
  details :
    [{
     type: Schema.Types.ObjectId,
     ref: 'Detail'
    }]
});

module.exports = mongoose.model('Thing', ThingSchema);
```

Hier link je detail database aan je thing database. Je verwacht de detail._id die je opslaat als array.

Zo kun je weten welke vraag bij welke vragenreeks hoort omdat zo elke vragenreeks de ids van de vragen mee in zijn database heeft zitten.

server/api/thing/thing.controller.js:

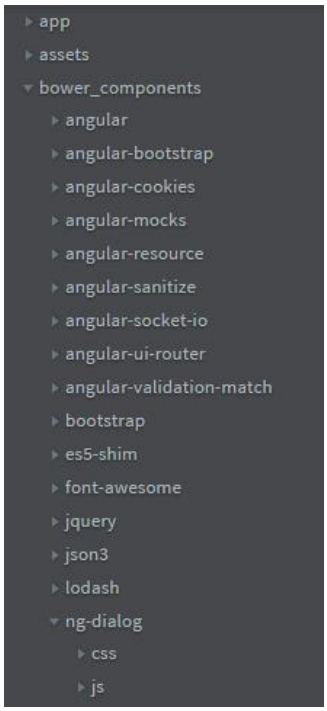
```
// Gets a single Thing from the DB
exports.show = function(req, res) {
  Thing.findById(req.params.id)
    .populate('details', 'question answer correctAnswer currentQuestion
groupId')
  .execAsync()
    .then(handleEntityNotFound(res))
    .then(responseWithResult(res))
    .catch(handleError(res));
};
```

.populate is belangerijk, anders zal het de database niet gevuld kunnen worden.

Toevoegen van components.

In dit project is bv. ng-Dialog toegevoegt, dit zorgt voor stijlbare alert boxes met controls.

Om dit te doen kun je in cmd gewoon binnen je project bower install ng-dialog typen.



in de client side wordt ng-dialog automatisch aangemaakt binnen u

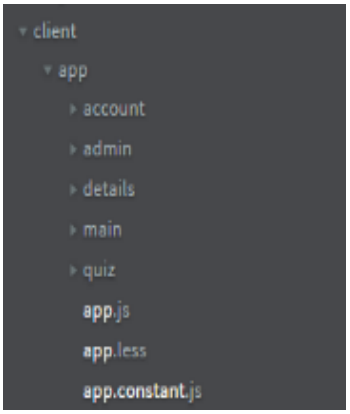
in client/app/app.js kun je dat aan je angular.module ngDialog toevoegen. Hierdoor kun het over al je html files

```
'use strict';

(function() {

class MainController {

  constructor($http, $scope, socket, $compile, ngDialog) {
```



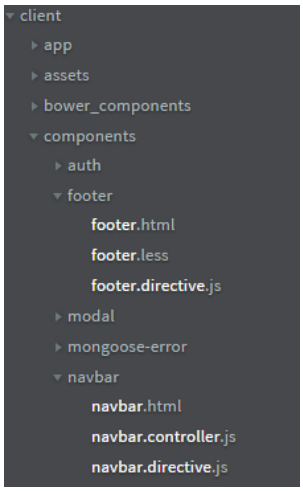

```
'use strict';

angular.module('ebvrApp', [
  'ebvrApp.auth',
  'ebvrApp.admin',
  'ebvrApp.constants',
  'ngCookies',
  'ngResource',
  'ngSanitize',
  'btford.socket-io',
  'ui.router',
  'ui.bootstrap',
  'ngDialog',
  'validation.match'
])
  .config(function($urlRouterProvider, $locationProvider) {
    $urlRouterProvider
      .otherwise('/');

    $locationProvider.html5Mode(true);
  });
```

Zo zou je nog meer elementen kunnen toevoegen op een makkelijke manier.

Directives



Footer en navbar zijn twee voorbeelden van directives.

Directives vertellen angularJS's html compiler ($compile, wordt gebruikt in de code) om gedrag aan een DOM element te hangen of te updaten.

Het voordeel hiervan is dat je gemakkelijk je code kunt hergebruiken en aan DOM manipulatie te doen.

In dit geval kan <navbar></navbar> op elke html pagina aangeroepen worden zodat deze overal hetzelfde eruit ziet.

Veel problemen zijn veroorzaakt door socket.io, dit is niet makkelijk te integreren. Sommige tutorials leggen het goed uit maar maken dan gebruik van jQuery bv. Hierdoor zijn we 3x terug opnieuw moeten beginnen, waarbij de laatste keer Yeoman voor ons de programma structuur (waar welke folder staat, wat front end en back end is) aangemaakt heeft. Vanaf dan was het vrij straight forward. Lijsten updaten was relatief makkelijk. Alleen zijn we op het laatste vastgelopen op een element updaten in het midden van een lijst. Doordat er heel weinig tutorials waren die zo specifiek waren hebben we er een week op vastgezeten zonder de oplossing te vinden tot heden.