2015-2016

Project : Labinventory

Cloud Applications

Lorenz Put & Thomas van Havere
ARTESIS PLANTIJN HOGESCHOOL

Voorwoord

Wij, Thomas van Havere en Lorenz Put, zijn leerlingen van 3EA2 op de Artesis Plantijn Hogeschool Antwerpen. Ons project voor Cloud applications bestond uit het creëren van een website om de inventaris van het labo elektronica te beheren. Deze website maakt gebruik van de MEAN stack, voor zowel de back-end als de front-end.

Om dit project te verwezenlijken hebben wij samen gewerkt met De heer Tim Dams en Tom Peeters. Wij danken deze voor de hulp en de steun geleverd bij dit project.

Table of Contents

Doelstellingen

Het doel van de website is om makkelijk de componenten / toestellen aanwezig in het lab elektronica te kunnen beheren.

- Research & development naar MongoDB
- Research & development naar Express
- Research & development naar AngularJS
- Research & development naar NodeJS

Taakverdeling

Thomas van Havere

- MongoDB (via mongoLabs)
- NodeJS

Lorenz Put

- AngularJS
- Express

Voorbereiding

We hebben onderzoek gedaan naar hoe we een website moeten schrijven met behulp van de MEAN stack of met behulp van een REST API. Uit ons onderzoek bleek dat de MEAN stack voor ons de beste keuze was. Dit omdat de MEAN stack gebruik maakt van MongoDB. In dit soort databases heb je geen vaste structuur en kan je gewoon velden toevoegen die je nodig hebt voor bijvoorbeeld een bepaalde component te beschrijven.

Om dit alles te bereiken hebben we samen gebrainstormd. Zo zijn we op het idee gekomen om de site in een soort van tabelvorm te laten weergeven waarin de professor de componenten in een overzichtelijke lijst te zien krijgt. Hierdoor is het gemakkelijk componenten om toe te voegen.

Benodigdheden

- Laptop
- Account op mongoLabs
- MongoDB server (mongoLabs)
- Account op Heroku
- hosting server (Heroku, labinventory.herokuapp.com)

Prijskaartje

Omdat de school niet over de nodige materialen beschikte voor het ontwikkelen van de terminals voor studenten, hebben we onze eigen componenten aangekocht. Deze componenten kun je terug vinden in de onderstaande tabel samen met respectabele prijs en hoeveelheid.

WAT	HOEVEELHEID	KOST
INNOLUX 7" TOUCHSCREEN	1	€31.00
BARCODE SCANNER	1	€16.00
RPI GPIO EXTENSION BOARD	1	€3.00
RPI B+	1	€40.00
WIPI	1	€20.00
SIGAREN KISTJE	1	€0.00
LICENTIE MONGODB	1	€0.00
LICENTIE HEROKU	1	€0.00
TOTAAL	8	€110.00

Zoals in bovenstaande tabel te zien is, hebben we in totaal €110.00 gespendeerd aan dit project. Dit is niet de kost gemaakt voor dit vak. Door het feit dat we dit project ook voor het vak Internet of Things doen en deze dus met elkaar verbonden zijn, hebben we de totaalprijs vermeldt. We kunnen we niet anders dan een klein beetje informatie verschaffen over de verschillende delen.

Bovenstaande tabel is vooral het prijskaartje van de componenten die we gebruiken voor Internet of Things. Dit gedeelte is een terminal gebouwd met een Raspberry Pi waarmee je met een barcode scanner componenten kunt inscannen en op gaat zoeken in de database.

Voor Cloud applications zijn enkel de licenties voor MongoDB en Heroku gebruikt. Wij hebben gebruik gemaakt van de gratis licenties, maar deze kunnen echter ook betalend zijn.

De MEAN stack

Zoals hiervoor al vermeld hebben we gebruik gemaakt van de MEAN stack. De grootste reden waarom we gebruik maken van de MEAN stack is omdat deze gebruik maakt van MongoDB. Deze is zeer handig voor onze doeleinden omdat we nooit op voorhand welke eigenschappen een component of toestel moet hebben.

M for MongoDB

Hier zullen we uitleggen hoe we connectie maken naar een MongoDB en hoe de basis structuur van een document in onze MongoDB eruit ziet.

Connectie vanuit NodeJS

Vooraleer we een connectie naar de database kunnen leggen moeten we eerst de juiste modules aanspreken. Dit doen we door de eerste regel in onderstaande blok uit te voeren. Deze regel zorgt ervoor dat de mongoose module door middel van de variabele mongoose aangesproken zal kunnen worden. Met behulp van deze variabele kunnen we dus een connectie vanuit mongoose op zetten.

```
var mongoose = require("mongoose");
mongoose.connect('mongodb://<dbusername>:<dbpassword>@ds054308.mongolab.com
:54308/labinventory');
```

De tweede regel in bovenstaande kader zal een connectie op zetten met de database. Dit gebeurt aan de hand van een connectiestring. Deze string wordt gegenereerd op mongolabs, maar moet echter nog aangepast worden met de juiste dbusername en dbpassword vooraleer er connectie met de database mogelijk is. Deze dbusername en dbpassword maak je aan op mongolabs onder de users tab. Helemaal vanachter in de connectiestring zet je de naam van de database waarmee je wilt verbinden. In ons geval is dit de Labinventory database.

Document structuur

Op de eerste twee regels zien we de structuur van een document voor een componenten. Verder in de code wordt ook nog een structuur aangemaakt voor users. De manier waarop deze structuren aangemaakt worden zijn het zelfde, alleen de veldnamen en de types van de variabele verschillen. Het componentSchema bestaat uit volgende velden :

- Type : Dit is de component of apertuur dat is in gescand vb. : een weerstand
- Value: Dit is de waarde van een bepaalde component of een eigenschap
- Quantity: Dit is de hoeveelheid aanwezig in het labo
- Barcode: Samenstelling van type en waarde.
- Note: Note is notitie toevoegen over component (in ontwikkeling).

```
var componentScheme = mongoose.Schema({Type: String, Value: String,
Quantity: Number, Barcode: String, Note: String});

var componentsmodel = mongoose.model('component', componentScheme);
```

De derde regel zorgt ervoor dat dit model in componentSchema wordt gehandhaafd bij het verzenden van component eigenschappen naar de MongoDB.

E for Express

Express is een NPM package voor het bouwen van web servers. Deze wordt meest gebruikt om web applicaties met NodeJS te bouwen. Express geeft ons een veel overzichtelijkere en gestructureerde code.

Initialiseren Express app

In de eerste regel wordt de Express module opgehaald. Deze hebben we op voorhand geïnstalleerd met de NPM package manager. Op de tweede regel creëren we de variabele waar we dan de Express module aan toe wijzen.

```
var express = require("express");
var app = express();
```

Related express modules

De eerste lijn code (express.static) wordt gebruikt om een statische route op te zetten naar de public map. Op deze manier kunnen CSS files, afbeeldingen, javascript files,… die zich in deze directory bevinden opgehaald en gebruikt worden.
Op de tweede lijn code (bodypaser.json) wordt de json module van bodyparser geïnitialiseerd. Door hiervan gebruik te maken, wordt het bson formaat dat we standaard terug krijgen van server geparsed naar json. Deze json wordt in AngularJS ontvangen en verwerkt, maar hier wordt later dieper op in gegaan.
De laatste regel wordt gebruikt voor het initialiseren van de paswoordencryptie.

```
//set up static routes
app.use(express.static(__dirname + "/public"));

//Use json body-parser
app.use(bodyparser.json());

app.use(passport.initialize());
```

Login pagina

Initialisatie users

In de eerste regel wordt een variabele user gecreëerd van het MongoDB schema dat we in het hoofdstuk “M for MongoDB” uitgelegd hebben. Hierna gaan we in database kijken of we een user vinden. Indien er zich geen users in de database bevinden, wordt er automatisch 3 standaard users aangemaakt. In het volgende stuk gaan we uitleggen hoe we één van deze users aanmaken.

```
var user = mongoose.model('User', userSchema);

user.find({}).exec(function(err,collection)
{
   if(collection.length == 0)
   {
      var salt, hash;

      salt = createSalt();
      hash = hashPwd(salt, 'Lorenz');
      user.create({schoolId: "s079368",firstName: "Lorenz",
      lastName: "Put", userName : "Lorenz", salt : salt,
      hashed_pwd: hash});
```

Aanmaken standaard user

Eerst maken we een variabele salt en hash aan. Deze variabelen zullen worden gebruikt om een resultaat van een functie op te slagen. Voor de variabele salt zullen we de functie “createSalt()” oproepen. In deze functie gebruiken we de crypto module van NodeJS om een string van 128 random bytes te genereren.

Daarna stellen we de variabele hash gelijk aan de functie “hashPwd(salt, ‘Lorenz’)”. Deze functie geven we de variabele salt (string 128 bytes) en het gewenste passwoord, in dit geval Lorenz mee. In deze functie worden de variabele salt en het passwoord door het “SHA1” versleutelingsalgoritme gehaald. Daarna wordt het nog eens naar de desbetreffende hexadecimale waarde geconverteerd. Het resultaat wordt geretourneerd en komt in de variabele hash terecht.

Authentication

Onderstaande code wordt gebruikt om te controleren of de gebruiker het juiste wachtwoord heeft ingegeven. Het ingegeven passwoord wordt gehashed zoals hierboven beschreven en wordt daarna vergeleken met het passwoord dat is opgeslagen in de database.

```
userSchema.methods = {
    authenticate: function (passwordToMatch){
        return hashPwd(this.salt, passwordToMatch) === this.hashed_pwd;
    }
}
```

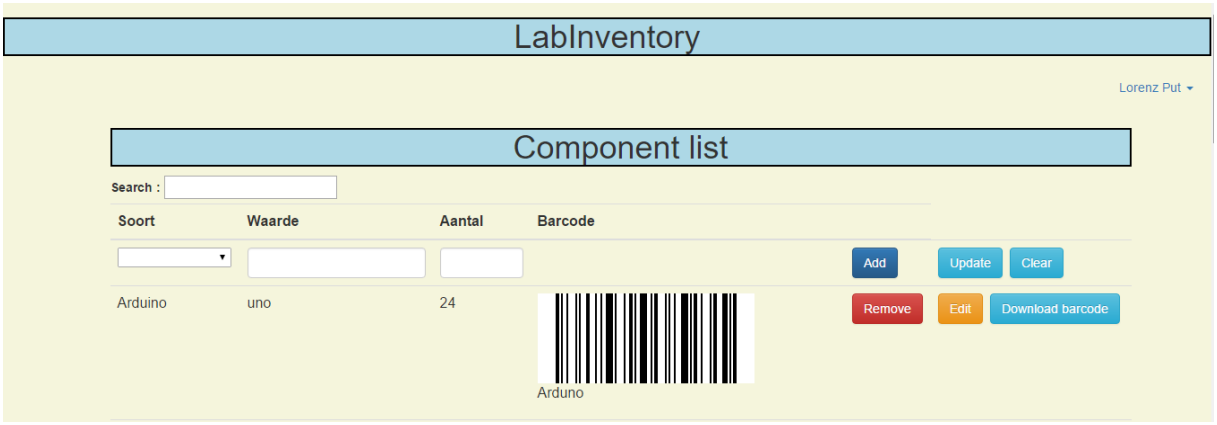
Deze authenticate methode wordt aangehaald wanneer je wilt inloggen op de pagina. Vooraleer je kan inloggen zal eerst nog gecontroleerd worden of dat de user bestaat. Indien deze niet bestaat zal er een error tevoorschijn komen. Dit gebeurt allemaal aan de hand van onderstaande code. Wanneer de username en het wachtwoord correct zijn zal je de administratie pagina te zien krijgen.

```
app.post('/login',function(req,res,next)
{
    var auth = passport.authenticate('local', function(err,user)
    {
        if(err)
        {
            return next(err);
        }
        if(!user)
        {
            res.send({success: false});
        }
        req.logIn(user,function (err)
        {
            if(err)
            {
                return next(err);
            }
            res.send({success : true, user: user})
        })
    })
    auth(req,res, next);
});
```

Betreffende de authenticatie bevindt er zich ook nog een deel in AngularJS. Deze code is echter te complex om in dit verslag uit te leggen.

Administratie pagina

Op de administratie pagina vindt je in de rechterbovenhoek de naam van de persoon die is ingelogd. Wanneer je op deze naam klikt, zullen de opties uitloggen en registreren van een nieuwe gebruiker tevoorschijn komen. Wanneer we verder naar beneden gaan op de pagina zullen we een zoekbalk zien. Deze kan gebruikt worden om te zoeken op het type van component. Onder de zoekbalk zien we drie lege velden staan met daarnaast drie knoppen : add, update en clear. Wanneer we de velden invullen, moeten we in het eerste veld de soort ingeven. Dit gebeurt aan de hand van een dropdown menu dat gehardcoded component types bevat vb. Arduino. Daarnaast zullen we de waarden invullen vb. uno. Als laatste zullen we de hoeveelheid die beschikbaar is in het labo invullen. Wanneer we dan op de knop add drukken zal de administratie pagina eruit zien als onderstaand screenshot.



Uitloggen en registreren

Onderstaande code zorgt ervoor dat de sessie wordt vernietigd en dat je wordt uitgelogd.

```
app.post('/logout', function (req,res) {
   req.logout();
   res.end();
});
```

Wanneer er op de administratie pagina op register new user wordt geklikt, zal er een pagina tevoorschijn komen waarin je de informatie zult kunnen invullen van de te registreren gebruiker. Zoals in het hoofdstuk “Initialisatie users” zal het paswoord gehasht worden met de desbetreffende functies. De rest van de onderstaande code zal de velden invullen met de data verkregen van AngularJS en pushen naar de database.

```
app.post('/register', function (req,res) {
   salt = createSalt();
   hash = hashPwd(salt, req.body.password);
   user.create({
      schoolId : req.body.SchoolId,
      firstName: req.body.firstname,
      lastName: req.body.lastname,
      userName: req.body.username,
      salt: salt,
      hashed_pwd: hash
   });
   res.end();

});
```

A for AngularJS

In dit hoofdstuk zullen de meeste delen van de AngularJS code worden toegelicht.

Laden pagina

Wanneer de pagina geladen wordt, wordt er in AngularJS gecontroleerd of alle velden ingevuld zijn. Indien dit het geval is zal er in de onderstaande code een get request gedaan worden naar de server. De server verwerkt deze request en stuurt alle documenten in de collectie componenten terug naar de client. In de client wordt via het ng-repeat directive alle documenten in een mooie tabel gegoten.

```
$http.get('/componentlist').success(function(response)
{
   $scope.componentList = response;
   $scope.component = "";

});
```

Search bar

De search bar bestaat uit een input veld dat via ng-model gekoppeld is aan de variabele searchText.

```
<label>Search : <input ng-model="searchText"></label>
```

In het ng-repeat directive wordt terug gebruikt gemaakt van de searchText variabele. Door gebruik te maken van het filter statement en de variabele zal de output van de ng-repeat automatisch worden aangepast.

```
<tr ng-repeat = "component in componentList | filter: searchText |
orderBy:'Type'" >
```

Add knop

Wanneer er op add geklikt wordt, wordt er in AngularJS gecontroleerd of alle velden ingevuld zijn. Indien dit het geval is wordt er in onderstaande code een post request gedaan. Deze post wordt door de server ontvangen en verwerkt. Vervolgens stuurt de server terug dat de post succesvol was en wordt in de AngularJS code de refresh functie uitgevoerd. De refresh functie bestaat simpelweg uit de get functie die hierboven beschreven werd.

```
$scope.addComponent = function (type,value)
{
      $http.post('/componentlist', $scope.component)
.success(function (response) {
         refresh();
      });
};
```

Update / edit knop

Wanneer we een eigenschap van een component zouden willen veranderen. Dan gaan we eerst deze component zoeken in de tabel op de administratie pagina. Wanneer we de component hebben die we willen bewerken, klik je op de knop edit. Nu zal je zien dat de velden bovenaan de pagina ingevuld zijn met de respectievelijke velden van de component die je wilt bewerken.

```
$scope.edit = function(id)
{
   $http.get('/componentlist/' + id).success(function(response)
   {
      $scope.component = response;
   });
};
```

Bovenstaande AngularJS code zorgt ervoor dat de velden worden ingevuld. Vervolgens kan je de component naar wens aanpassen. Wanneer je nu op de update knop klikt zal volgende code worden uitgevoerd.

```
$scope.update = function()
{
   $http.put("/componentlist/" + $scope.component._id,
$scope.component).success(function(response)
   {
      refresh();
   });
};
```

Bovenstaande code zal via het component._id het juiste document aanpassen met de nieuwe waarden. Wanneer dit opgeslagen is in de database zal de pagina automatisch refreshen.

Clear knop

Wanneer er op deze knop wordt geklikt zullen de velden die zijn ingevuld zijn, geledigd worden. Dit zal niets veranderen aan de database.

Remove knop

Wanneer er naast de component op de remove knop geklikt wordt, zal dit de component verwijderen uit de database. Dit gebeurt aan de hand van het component._id. Wanneer het verwijdert is, zal de pagina refreshen waardoor het ook van de administratie pagina verdwijnt.

```
$scope.remove = function(id)
{
     $http.delete("/componentlist/" + id).success(function(response){
         refresh();
   });
};
```

Barcode generator

De applicatie maakt gebruik van een barcode generator. Deze generator bestaat uit een AngularJS bibliotheek, die we van het internet hebben gehaald. De bibliotheek is met de html pagina gelinkt met behulp van AngularJS directives. We kunnen dus zeggen dat de generatie van deze barcodes dus niets te maken heeft met de database. De enige communicatie die met de database gebeurt is het verzenden van de string die ook gebruikt wordt om de barcodes te genereren. Deze string zal later ook gebruikt worden bij het Internet of Things gedeelte om componenten terug te vinden.

De functie converttype neemt als argument het type van de component converteert dit naar een string die zowel het type als de waarde van de component bevat.

```
<div id="barcode" barcode-generator="{{converttype(component.Type) +
component.Value}}" style="height:100px; text-align: center; width: auto;">
</div>
```

Download barcode knop

De download barcode knop roept de functie downloadbarcode aan in de mainController. Dit geeft als argument $index mee. We geven de $index mee omdat we meerdere download barcode knoppen hebben die elks een overeenkomstige barcode downloaden. Door deze $index kunnen we de juiste barcode aan de juiste download barcode knop linken.

```
<button ng-click="downloadbarcode($index)"  class="btn btn-info">Download
barcode</button>
```

Wanneer er op de knop geklikt wordt, wordt de onderstaande code uitgevoerd. De meegegeven id wordt gebruikt om de div naar een canvas te converteren. Dit gebeurt met behulp van een javascript bibliotheek. Wanneer de conversie klaar is, wordt het gegenereerde canvas gedownload als een png bestand, maar wel zonder extentie.

```
$scope.downloadbarcode = function(id) {
   html2canvas($("#" + id), {
      onrendered: function (canvas) {
         Canvas2Image.saveAsPNG(canvas);
      }
});
```

N for NodeJS

NodeJS is de backend van onze applicatie. Het is gebaseerd op de Javascript engine van Google. Deze engine zet de Node code om in machine code waardoor het door de server geïnterpreteerd en uitgevoerd kan worden. De grote voordelen van NodeJS zijn I/O events, netwerk communicatie, file transfer.

Besluit

Om dit project tot een goed einde te brengen hebben we voor heel wat problemen gestaan. Hoewel het niet altijd gemakkelijk om de problemen op te lossen, hebben we nooit getwijfeld aan het feit dat we Labinventory tot een goed einde konden brengen.

Enkele dingen die ons gefrustreerd hebben tijdens dit project was het leren kennen van mongoose. Dit deed in het begin van het project helemaal niks. Dit hebben we dan samen bekeken en uiteindelijk opgelost. Een ander probleem was bij het downloaden van de barcode. Het barcode element (een div) was niet downloadbaar. Dit hebben we opgelost door het eerst te converteren naar een canvas alvorens te downloaden.

Al bij al met af en toe een beetje miscommunicatie, hebben we er toch een mooi project van gemaakt en zijn we toch zeer tevreden met het resultaat. Wij hopen dat de school en met name De heer Smets & van Houtven hier later nog veel aan zullen hebben.