Bachelor's thesis

Degree programme in Information Technology

Information Technology

2016

Erik Rigoberto Lanza Aplicano

BUILDING A WEB APPLICATION USING THE MEAN STACK

– A walk-through the process

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2016 | 25

Erik Rigoberto Lanza Aplicano

BUILDING A WEB APPLICATION USING THE MEAN STACK

- A walk through the process

This thesis is focused on the building of a web application using contemporary JavaScript-based technologies. A walkthrough explanation on how to set a simple environment that takes advantage of modern full-stack technologies. The technologies used were: AngularJS for the front-end, Node.js together with Express for the back-end and MongoDB for the database. To develop this application, tools, such as Vagrant, VirtualBox, and Mongoose among others were used throughout the process.

The application consists of a social platform that allows a user to register, login, post, and comment in different posts. The idea is similar to the widely used platforms like Reddit and HackerNews.

This application was created for experimentation purposes and although it is working and has different features, it can be improved visually and many other features can be added. Further testing and even trying to deploy the application in a real server could be areas for future development.

KEYWORDS:

JavaScript, AngularJS, Node.js, Express, MongoDB, front-end, back-end, Vagrant, VirtualBox, Mongoose.

CONTENTS

FIGURES

PICTURES

TABLES

LIST OF ABBREVIATIONS (OR) SYMBOLS

SASS	Syntactically Awesome Stylesheets
MEAN	MongoDB Express AngularJS Node.js
CRUD	Create Read Update Delete
CSS	Cascade Style Sheets
HTML	Hyper Text Markup Language
JSON	Java Script Object Notation
JWT	JSON Web Token
SPA	Single Page Application
OS	Operating System
CDN	Content Delivery Network

1 INTRODUCTION

Nowadays the need to create or build any kind of product has allowed new inventions to emerge and facilitate the old ways of creating things. Frameworks, pre-processors, and libraries are some examples of such inventions in the technology field. Almost every programming language and technology counts on some or several frameworks that help the developer to finalise a task.

In this thesis, the usage of different technologies and frameworks are the main subject and the way that can be used is explained during the process of making a web application. The application was built for experimentation purposes and although it can be used as a an application similar to Reddit or HackerNews, it is not meant to be utilised by an specific end-user.

My main motivation for building an application using MEAN stack, was for learning the technologies used in this framework and the process that requires making a single web application from end to end. The MEAN stack uses JavaScript-based technologies that allows the developer to build a consistent single page web application.

2 THE MEAN STACK COMPONENTS

The MEAN stack consists of 4 main technologies that fully compose the application. The front-end of the application is made using AngularJS, Node.js together with Express for the back-end and MongoDB for the database. These four technologies are used to build web applications. The MEAN stack works best for making single page web application (SPA) because its functionality is mainly focused on the client side. The following sections will be dedicated for introducing the technologies used throughout the making of the application.

2.1 MongoDB

MongoDB is a document-oriented, cross platform, and open source database. According to the MongoDB website: “MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.” (MongoDB 2016).

MongoDB is a NoSQL database that offers very versatile structure that can range from very basic databases to very complex ones. MongoDB can easily be scaled up and it does not require to have a pre-defined schema to start working with. This makes it very attractive for start-ups that can later on become large.

MongoDB uses a similar to JavaScript Object Notation (JSON) syntax and calls it BSON. Differing from well-known databases, such as MySQL, MongoDB does not make use of Tables and Rows, instead Mongo uses Collections and Documents.

To aid MongoDB, Mongoose was used throughout the building process of the web application. “Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box” (Valeri Karpov, Mongoose official website 2011).It helps the developer to model the application data in a simpler and smarter way than using plain Mongo.

Mongoose uses Schemas to define the distribution of the documents within a Mongo collection. There are different kinds of Schema types allowed in Mongoose: String, Number, Date, Buffer, Boolean, Mixed, ObjectId, and Array.

2.2 Express JS

The second component of the MEAN stack is Express. “Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.” (Express JS official website 2016). Due to its features, Express is the most popular framework for building web applications with Node.js.

Express is used for routing. That means that it manipulates the behavior of an application towards a client request and executes functions when the request’s Uniform Request Identifier (URI) matches the route.

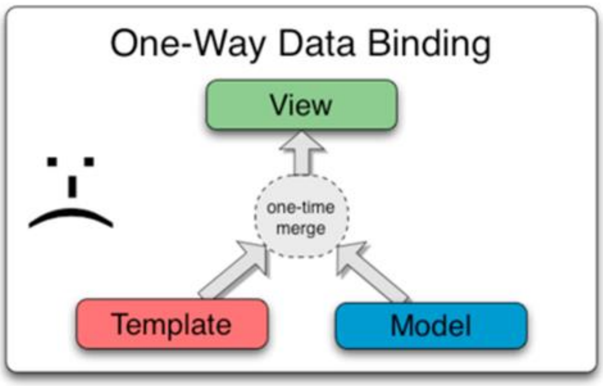
An easy and effective way to start building a Express application is by using the Express-generator which builds the application’s skeleton. The skeleton is the way the directories are distributed and created.

2.3 AngularJS

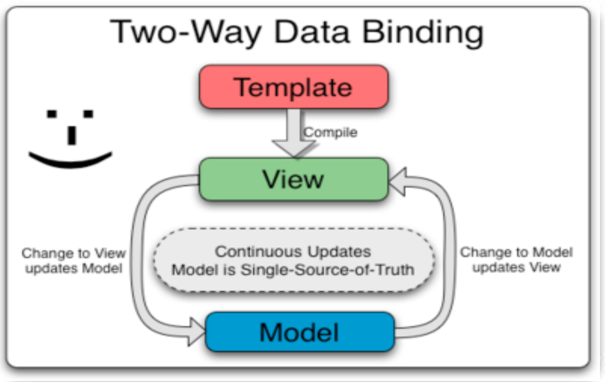
One of the most popular types of web applications are single web applications, AngularJS is a JavaScript front-end framework designed for making this kind of applications. AngularJS, or better known as Angular, is open source, cross platform and follows the MVC architectural pattern.

Angular offers two-way data binding, which is its main feature over other strong frameworks/libraries like React and Ember. According to Andrea Bresolin (2015): “We have a two way data binding when a model variable is bound to a HTML element that can both change and display the value of the variable. In general, we could have more than one HTML element bound to the same variable.”

What two-way data binding offers over one-way data binding is the ability to take a property’s value and display it on the view at the same time as it updates it in the model. On the other hand, in one-way data binding this is not possible. Picture 1 and Picture 2 illustrate the difference between one and two-way data binding.



Picture 1. One-Way Data Binding (AngularJS 2010).


Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates
Model is Single-Source-of-Truth
Change to Model updates View
Model

Picture 2. Two-way Data Binding (AngularJS 2010).

Angular offers a very broad set of personalizing an application's HTML by using directives. Directives control behaviours in the Document Object Model (DOM). Angular comes with a set of Directives that make a series of processes easier, for example, ng-repeat acts as a loop, ng-hide as a conditional statement and many others. The Directives can also be created by the developer to make repetitive behaviours reusable.

2.4 Node.js

The MEAN stack uses Node.js for the back-end and as its final component. Node.js is a run-time system written in JavaScript that, as the rest of the MEAN stack components, is open source and cross-platform.

Node.js uses a package manager for reusing code that other developers have written and update the code that is being written. This package manager is called Node Package

Manager (NPM). According to npmjs official website (2016): “It's a way to reuse code from other developers, and also a way to share your code with them, and it makes it easy to manage the different versions of code.”

Other Tools

In this project, many tools for making the process easier were used. One of them is Vagrant . In this project, many tools for making the process easier were used. One of them is Vagrant . Vagrant is an open source tool that lets the user to create a virtual machine, with the aid of a virtualization software like VirtualBox, with preset settings for RAM, CPU and HDD. By using Vagrant, the user saves time and avoids repetitive commands (Hashicorp 2016).

For the development of this application, Ubuntu Operating System (OS) was used. Ubuntu is a Debian-based operating system and free software based (Ubuntu 2016).

A fairly easy way to install all the MEAN stack requirements and layouts is to use a MEAN stack boilerplate. A boilerplate is a premade project using a stack of technologies, in this case the MEAN stack. The boilerplate usually counts with a already made layout of directories and basic feature of a common web application. There are two very popular boilerplates for the MEAN stack: the MEAN.js and the MEAN.io boilerplates both created by Armos Haviv.

It is difficult to state which one is the best option to start a project with, since both are very similar. Something to consider is the time each boilerplate option other has been used by the public. This may affect the amount of documentation created for each boilerplate.

The MEAN.io has been used longer than the MEAN.js, hence, the documentation may be superior.

Bootstrap was used to make the application responsive and decent- looking. Bootstrap is a CSS framework that facilitates the developer to make fully responsive websites. Bootstrap uses a grid layout that divides the page in 12 columns in a row and the developer chooses how many columns will be used for the content he/she picks in each row. For example, a paragraph can occupy 6 columns in a row when is seen from a desktop computer and a complimentary image the rest of the columns. This is able to change when the page is seen from a mobile device, the 12 columns will be occupied by the paragraph and other 12 columns in another row will be occupied by the complimentary image. This will make them be displayed in different rows (Twitter Bootstrap 2016).

3 DEVELOPING THE APPLICATION

There are different ways to start a MEAN stack application. One of them, as it was mentioned, is by using a MEAN boilerplate. In this application no boilerplate has been used due to various issues encountered while installing either of the boilerplates, hence, each of the components of the MEAN stack has been installed separately.

3.1 Creating an Express project

The first step taken for creating the MEAN stack application, was to create a layout for the directories. This layout was created using the Express-generator.

By typing the command sudo express -–-ejs, myThesisApp will create a new application layout with the name given. The ejs attribute changes the default templating engine Jade, to something more similar to pure HTML, ejs. This command will create a series of directories to store the files needed for the application to work. The layout looks as in Figure 1.

```
├── app.js
├── bin
|   └── www
├── package.json
├── public
|   ├── images
|   ├── javascripts
|   └── stylesheets
|       └── style.css
├── routes
|   ├── index.js
|   └── users.js
└── views
    ├── error.jade
    ├── index.jade
    └── layout.jade
```

Figure 1. Express-generator Layout.

The models directory was not created by the Express-generator; it was specifically created for using Mongoose. The Mongoose models will be stored in this directory.

The public directory stores all the data and files that will be accessed by the user when the app is ready to be published. Inside the public directory are stored directories like JavaScript, CSS, Images and other required data needed for the web app to work smoothly. The routes directory stores most of the back-end code and keeps the Node controllers.

In the views directory, the main HTML file is found. In this directory all the views for the web app are stored. It usually goes so that each view has its own HTML, but in the case of this application, inline templates were used. Using Inline templates means that different views of the web app are stored in the same HTML file.

3.2 Working with AngularJS

This project was started by creating the front-end of the application. Angular was used to build the whole front-end with some HTML and Bootstrap. Angular connects all the dots within the application.

3.2.1 Requirements

The building of the application was started by creating a basic HTML file and adding the Content Delivery Network (CDN) to the head of the HTML file for AngularJS to work. A JavaScript file was created with the name of angularApp.js inside of the JavaScript in the public directory. In this file, all the AngularJS code and logic was written. It is recommended to break down the code into different files for organizing the application in a clear way.

To create an Angular app, a module that handles all the app was created. According to Angular JS (2010) “A Module is a collection of services, directives, controllers, filters, and configuration information”. (see Table 1).

Table 1. Creating an Angular app.

```
var app = angular.module('myThesis', ['ui.router']);
```

A simple Angular app consists in a HTML file and a JavaScript file. The HTML file is the way the app will be seen by the user, it will show the information gathered by the JavaScript. The JavaScript's tasks are to collect information from the database and distribute it throughout the application.

3.2.2 Components

The JavaScript and the HTML in the application communicate with each other with directives. The directives are components of the Angular library that have specific logic that facilitate the tasks to be done when displaying the information in the HTML. For instance, the ng-repeat directive works as a loop to the array “posts” declared in the angularApp.js. Other directives, such as ng-click, triggers the function “incrementUpvotes(post)” when clicking the “^” symbol and the ng-show shows the link information in the post array if there is any available (see Table 2).

Table 2. Directives.

```
<div ng-repeat="post in posts | orderBy:'-upvotes'">
  <span class="glyphicon glyphicon-thumbs-up"
    ng-click="incrementUpvotes(post)"></span>
```

The ng-repeat directive was used to go through repetitive information like posts and comments made by the user. The directives are always declared within the HTML, not the JavaScript. As seen before, directives are declared as classes or attributes in HTML. There are many in-built directives in Angular, ng-class, ng-submit, ng-model are just a few of them. A very important directive is used to start any Angular application. The name of this directive is “ng-app” and should have the name of the app as the value that matches the module done previously. (see Table 3)

Table 3. ng-app Directive.

```
<body ng-app="myThesis">
```

The Controller in AngularJS has the task of providing the information to the directives so that they can manipulate the information and show it in the views. The controller needs to be declared through the ng-controller directive in the HTML so this can work properly. The controller uses the scope to display the data to the view. (see Table 4).

Table 4. Controllers.

```
app.controller('MainCtrl', [...
```

The main controller handles the posts added to the database through the views. Other controllers were made for handling the comments added to the posts (PostsCtrl), the authorization controller (AuthController) handles behaviors like login, register and such and the navigation controller (NavController) will take care of when the user has logged in to display a different navigation bar.

Angular services can be defined as JavaScript functions that are responsible to do specific tasks and work with persistent data. There are many different in-built services in AngularJS, among them we can find $http that is used to make AJAX calls to the server. Different ways to create an Angular service are: factories, services, and providers. For this application the factory method was used. For using Angular factories, an object was created, some properties were added to it and them returns the same object. (see Table 5).

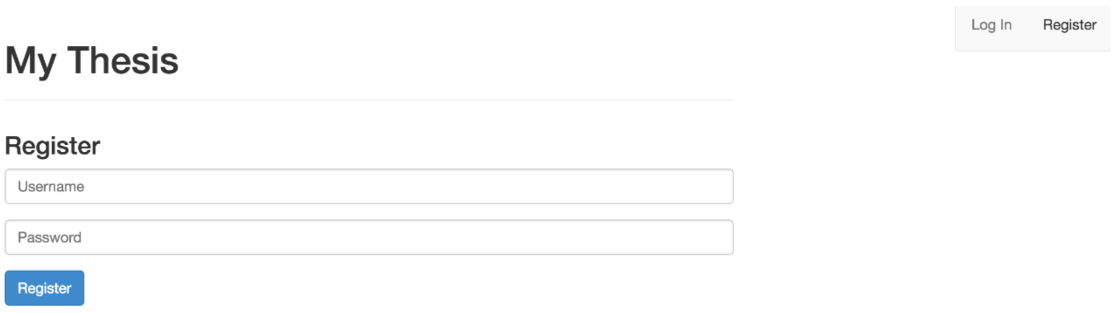
Table 5. Creating Services.

```
app.factory('posts',['$http', 'auth', function($http, auth) {
  var o = {
    posts: []
  };

  o.getAll = function() {
    return $http.get('/posts').success(function(data){
      angular.copy(data, o.posts);
    });
  };
[…]
  return o;
}]);
```

3.2.3 Behaviours

In this application, two services or factories were created: the authorization service and the posts service. The authorization (auth) service handles registrations, logins, logouts, users and verifications. While the posts service takes care of creating posts, adding comments, likes, and getting the information back. The registration page is shown in Picture 3.

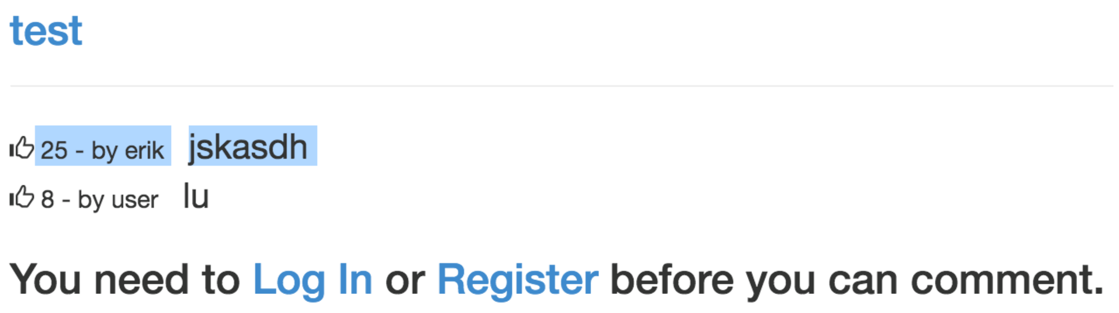


Picture 3. Registration Page.

To access the data handled by a service is necessary to inject the service to the controller. To inject the service to the controller, the name of the service was added as a parameter of the controller. As it is displayed in Table 6, the auth service is injected to the NavCrtl controller to check if a user is logged in.(see Picture 4).

Table 6. Injecting Services.

```
app.controller('NavCtrl', [
'$scope',
'auth',
function($scope, auth){
  $scope.isLoggedIn = auth.isLoggedIn;
  $scope.currentUser = auth.currentUser;
  $scope.logOut = auth.logOut;
}]);
```


test

25 - by erik jskasdh

8 - by user lu

You need to Log In or Register before you can comment.

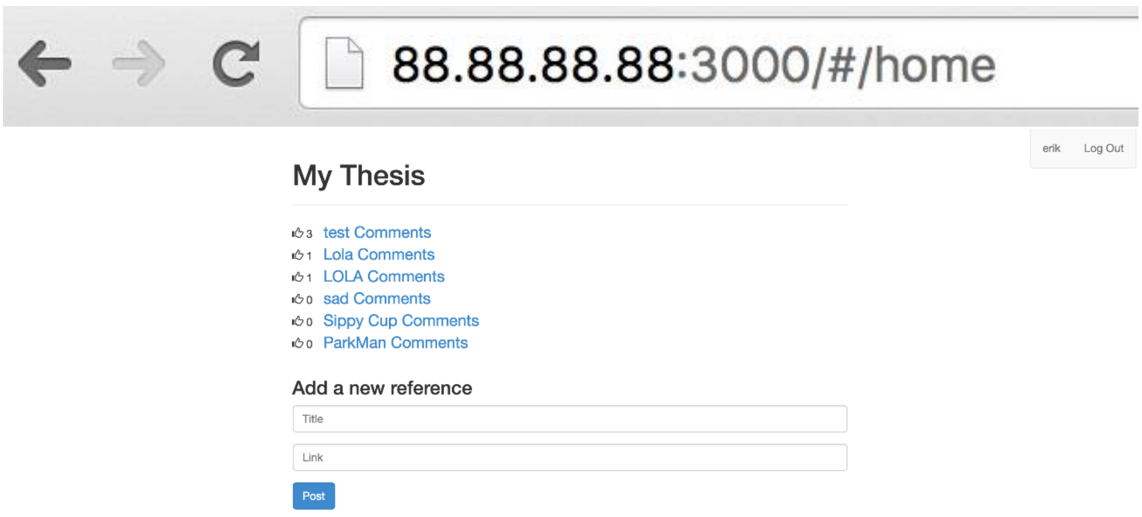
Picture 4. Login verification.

For managing different routes and controllers, AngularUI Router has been used. According to Angular UI Router (2013): “Angular UI-Router is a client-side Single Page Application routing framework for AngularJS”. AngularUI router is more flexible and provides more features than the module that Angular already counts on, the ng-route module.

To start using AngularUI Router, its CDN was added to the head section in the index.ejs file in the views directory of the application. Since this is an external module, it is also required to add a dependency to the app. This is implemented by adding the AngularUI as a parameter of the Angular module app.

The views for this project were made inline. This means that just one HTML file was used to display all the different views in the project. It was implemented in this way because it is a small project. However, for bigger projects different HTML files should be created.

Inline templates work as individual scripts inside the HTML file, in this case the index.ejs and they are routed using AngularUI Router to individual URLs that will be displayed in the browser (see Picture 5).



Picture 5. Routes.

To configure the AngularUI Router, Angular’s config() function was used. Within the config() function, the inline templates were assigned their respective routes in the application. For this, $stateProvider, which is part of AngularUI Router, was used to give the inline templates a place in the views. In other words, it makes those lines of code a visible HTML document in the browser (see Table 7).

Table 7. Creating Routes.

```
app.config([
 '$stateProvider',
 '$urlRouterProvider',
 function($stateProvider, $urlRouterProvider){

   $stateProvider
    .state('home', {
      url: '/home',
      templateUrl: '/home.html',
      controller: 'MainCtrl',
```

3.3 Setting Up Mongoose

For Mongoose to work, Node.js and MongoDB should be installed and running. If these requirements are not met, it is very possible that Mongoose will not work properly or at all. Mongoose was installed through the Node Package Manager (NPM).

In the app.js file, Mongoose is required by writing “mongoose connect” followed by the IP address where the node server is running, in this case 88.88.88.88, followed by the database name (Table 8). This will allow the project to access the database and be displayed when the address is typed in the browser.

Table 8. Initializing Mongoose.

```
mongoose.connect('mongodb://88.88.88.88/news');
```

After having Mongoose required and connected, a new file called Posts.js was created in the models directory created previously. As mentioned before, the models directory contains the models needed for Mongoose to work on retrieving and posting data from the database to the application and vice-versa.

A typical Mongoose model contains a Mongoose schema that handles an action aimed at the database. In this application different models were defined: The Posts.js model that handles the posts made by the user. This model links the posts to the respective comments written by a user using an Object ID. Comments.js handles the comments posted in the posts and the Users.js handles the users registered and their contributions to the posts (see Table 9).

Table 9. Mongoose Model.

```
var mongoose = require('mongoose');


var PostSchema = new mongoose.Schema({
  title: String,
  link: String,
  upvotes: {type: Number, default: 0},
  comments: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Comment' }]
});


/*Method for upvotes*/
PostSchema.methods.upvote = function(cb) {
  this.upvotes += 1;
  this.save(cb);
};


mongoose.model('Post', PostSchema);
```

3.4 Security

Security measures for the passwords and login details were set. In this app, we use the Password-Based Key Derivation Function 2 (PBKDF2) which comes with node’s native crypto module for hashing passwords. The first step is to require the crypto module, Mongoose and JsonWebToken (JWT) in the beginning of the Mongoose Users.js model (see Table 10).

Table 10. Security Dependencies.

```
var mongoose = require('mongoose');
var crypto = require('crypto');
var jwt = require('jsonwebtoken');
```

Hashing passwords was implemented by combining a password with a salt, which is randomly generated data, via PBKDF2. This was done so that the passwords are harder to crack (see Table 11).

Table 11. Creating Salt.

```
UserSchema.methods.setPassword = function(password){
  this.salt = crypto.randomBytes(16).toString('hex');
  this.hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64).toString('hex');
};
```

Later on the passwords saved will need to be verified when the user logs in. For this case, a method that compares the password entered by the user and the hashed password was written. The method will return a Boolean indicating if the password is correct or not (see Table 12).

Table 12. Verify Password.

```
UserSchema.methods.validPassword = function(password) {
  var hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64).toString('hex');
  return this.hash === hash;
};
```

For generating tokens for the users JSON Web Token was used. According to JWT official website (2014): “JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA.”

When the user is logged in, all the requests made by the user carry this token. Access, services, and resources will be available for the user thanks to this token. A method for generating the token is created and the expiration date was specified within it (See Table 13).

Table 13. Setting JASON Web Token.

```
UserSchema.methods.generateJWT = function() {
  // set expiration to 60 days
  var today = new Date();
  var exp = new Date(today);
  exp.setDate(today.getDate() + 60);
  return jwt.sign({
    _id: this._id,
    username: this.username,
    exp: parseInt(exp.getTime() / 1000),
  }, 'SECRET');
};
```

As a last step for this application, a passportjs was configured. According to PassportJS (2012): “Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password, Facebook, Twitter, and more.”

A passport-local strategy was used to handle username/password authentication. By running the command `npm install passport passport-local`, the passport will be installed. After the installation has been completed, the passport should be called in the app.js file.

A new directory in the root of the application was created and named “config”. Inside this directory, a JavaScript file was created which handles the passport configuration. (See Table 14).

Table 14. Passport Configuration.

```
var passport = require('passport');
var LocalStrategy = require('passport-local').Strategy;
var mongoose = require('mongoose');
var User = mongoose.model('User');


passport.use(new LocalStrategy(
  function(username, password, done) {
    User.findOne({ username: username }, function (err, user) {
      if (err) { return done(err); }
      if (!user) {
        return done(null, false, { message: 'Incorrect username.' });
      }
      if (!user.validPassword(password)) {
        return done(null, false, { message: 'Incorrect password.' });
      }
      return done(null, user);
    });
  }
));
```

4 CONCLUSION

The application made using MEAN stack is fully working and demonstrates that using this stack is a viable way to create a single page web application. Although the app is working, there a many features that can be added in the future, for example, deleting comments, adding images, creating groups and so on. For the sake of this thesis, the MEAN stack was tested as a reliable way of making a web application and more features could be added eventually.

The MEAN stack offers a complete set for making web applications. The consistency that JavaScript allows using it along with the whole application and makes it very practical and rewarding when executing the application correctly.

Making a full-stack web application can be underestimated by many young developers. A complete web application requires considering various factors, from designing the application, database, and structure to allow the project to grow in the future, until all the security implementations that should be carried out in any web app nowadays.

It is indeed interesting to learn from end to end how to make an application. This allows developers, designers, and anyone in general to be able to understand their team mates when working on a complete web application project.

We believe that the MEAN stack is a very reliable framework to familiarize with several current technologies and it is now one of the most popular stacks for making single-page applications. The MEAN stack will stay around for some time.

REFERENCES

AngularJS. Available at: https://angularjs.org/ Accessed 15.3.2016

AngularJS Modules. Available at: https://docs.angularjs.org/api/ng/function/angular.module Accessed 24.03.2016

AngularJS Data Binding. Available at: http://www.angularjshub.com/examples/basics/twowaydatabinding/ Accessed 26.03.2016

Angular UI. Available at: https://github.com/angular-ui/ui-router/ Accessed 10.04.2016

JSON Web Token. Available at: https://jwt.io/introduction/ Accessed 10.04.2016

MEAN IO. Available at: http://mean.io/ Accessed 15.3.2016

MEAN JS. Available at: http://meanjs.org/ Accessed 15.3.2016

MongoDB . Available at: https://docs.mongodb.org/manual/introduction/ Accessed 20.3.2016

Mongoose. Available at: http://mongoosejs.com/ Accessed 20.3.2016

NodeJS. Available at: https://nodejs.org/en/ Accessed 15.3.2015

Package Manager. Available at: https://en.wikipedia.org/wiki/Package_manager#/media/File:Pms.svg Accessed 04.04.2016

PassportJS. Available at: http://passportjs.org/docs Accessed 15.04.2016