

**Nghi Le Thanh**

**MEAN STACK WEB DEVELOPMENT**

**Thesis**

**CENTRIA UNIVERSITY OF APPLIED SCIENCES**

**Information Technology**

**May 2016**

## ABSTRACT

<b>Centria University of Applied Sciences</b>	<b>Date</b> May 2017	<b>Author</b> Nghị Lê Thanh
<b>Degree programme</b> Information Technology		
<b>Name of thesis</b> MEAN STACK WEB DEVELOPMENT		
<b>Instructor</b> Kauko Kolehmainen	<b>Pages</b> 43	
<b>Supervisor</b> Kauko Kolehmainen		
<p>The aim of the thesis is to provide a universal website using JavaScript as the main programming language. It also shows the basic parts anyone need to create a web application. The thesis creates a simple CMS using MEAN stack.</p> <p>MEAN is a collection of JavaScript based technologies used to develop web application. It is an acronym for MongoDB, Express, AngularJS and Node.js. It also allows non-technical users to easily update and manage a website's content. But the application also lets other developers connect with REST APIs.</p>		
<b>Key words</b> MEAN STACK, modern web development, RESTful API		

**ABSTRACT**  
**ABBREVIATIONS**  
**CONTENTS**

<b>1 INTRODUCTION .....</b>	<b>1</b>
<b>2 WIREDDELTA – THE PROMISING STARTUP .....</b>	<b>2</b>
<b>3 MODERN WEB APPLICATION.....</b>	<b>3</b>
<b>3.1 Single Page Application .....</b>	<b>3</b>
<b>3.2 The MEAN Stack .....</b>	<b>4</b>
<b>3.2.1 Node.js.....</b>	<b>5</b>
<b>3.2.2 Express .....</b>	<b>7</b>
<b>3.2.3 AngularJS .....</b>	<b>7</b>
<b>3.2.4 MongoDB .....</b>	<b>7</b>
<b>3.3 REST API and JSON.....</b>	<b>8</b>
<b>3.3.1 REST API .....</b>	<b>8</b>
<b>3.3.2 JSON .....</b>	<b>9</b>
<b>3.4 Content Management System .....</b>	<b>9</b>
<b>4 PLANING PHASE.....</b>	<b>10</b>
<b>4.1 Goals and Target Users .....</b>	<b>10</b>
<b>4.2 Site map and Wire frame Creation .....</b>	<b>10</b>
<b>5 INSTALLATION.....</b>	<b>12</b>
<b>5.1 Node.js.....</b>	<b>12</b>
<b>5.1.1 Installing with GUI .....</b>	<b>12</b>
<b>5.1.2 Using terminal .....</b>	<b>13</b>
<b>5.2 MongoDB .....</b>	<b>14</b>
<b>5.3 Utility Modules .....</b>	<b>15</b>
<b>5.3.1 Mongoose .....</b>	<b>16</b>
<b>5.3.2 Async and Q.....</b>	<b>17</b>
<b>6 IMPLEMENTATION .....</b>	<b>19</b>
<b>6.1 Application Architecture .....</b>	<b>19</b>
<b>6.2 Database Architecture .....</b>	<b>20</b>
<b>6.3 JSON Sample Files, Data Importation.....</b>	<b>20</b>
<b>6.4 Basic Operation .....</b>	<b>23</b>
<b>6.4.1 Retrieving Data .....</b>	<b>24</b>
<b>6.4.2 Adding Product .....</b>	<b>25</b>
<b>6.4.3 Editing Product .....</b>	<b>28</b>
<b>6.4.4 Deleting Product.....</b>	<b>33</b>
<b>6.5 Authenticating with JSON Web Token .....</b>	<b>34</b>
<b>7 API TESTING.....</b>	<b>38</b>
<b>7.1 Testing retrieving data.....</b>	<b>38</b>
<b>7.1 Testing creating data .....</b>	<b>40</b>
<b>7.3 Testing editing data.....</b>	<b>40</b>
<b>7.4 Testing removing data .....</b>	<b>41</b>

<b>8 CONCLUSION .....</b>	<b>43</b>
---------------------------	-----------

<b>REFERENCES.....</b>	<b>44</b>
------------------------	-----------

## **GRAPHS**

GRAPH 1. MEAN Stack elements .....	4
GRAPH 2. Diagram of the MEAN stack.....	5
GRAPH 3. Node.js Single Thread Event Model .....	6
GRAPH 4. MongoDB data scheme .....	8
GRAPH 5. REST API design .....	9
GRAPH 6. The sitemap of CMS Application.....	11
GRAPH 7. The wire frame of CMS.....	11
GRAPH 8. Node.js download page .....	13
GRAPH 9. Checking Node.js on local machine.....	14
GRAPH 10. MongoDB successful connection on local machine.....	15
GRAPH 11. Working with MongoDB in terminal on local machine.....	16
GRAPH 12. Callback example .....	17
GRAPH 13. Application architecture .....	19
GRAPH 14. Database architecture .....	20
GRAPH 15. Product CRUD flow chart .....	23
GRAPH 16. Product List .....	25
GRAPH 17. Add new product form .....	27
GRAPH 18. Edit Product Form .....	33
GRAPH 19. REST API with login system .....	36
GRAPH 20. Login Activity .....	37
GRAPH 21. cURL example.....	38
GRAPH 22. All Products.....	39
GRAPH 23. Adding new product .....	40
GRAPH 24. Error response.....	41
GRAPH 25. Removing product .....	42

## **ABBREVIATIONS**

MEAN	MongoDB, Express, AngularJS, Node.js
HTTP	Hyper Text Transfer Protocol
XML	eXtensible Markup Language
RPC	Remote Procedure Call
REST	Representational State Transfer
JSON	JavaScript Object Notation
API	Application Interface
WD	Wiredelta
HTML	Hypertext Markup Language
PHP	Hypertext Preprocessor
SPA	Single Page Application
MVC	Model – View – Controller
MVVM	Model – View – View Model
I/O	Input / Output
CRUD	Create, Read, Update, Delete
ECMA	European Computer Manufacturers Association
CMS	Content Management System

## 1 INTRODUCTION

Websites are one of the most effective ways to present and propagate information to people around the world. The web browser is a software application for retrieving, presenting and traversing information resources on the World Wide Web. More than ten years ago, a website was constructed with raw HTML document and served by simple HTTP request. The birth of CSS and JavaScript made web application become more and more convenient in displaying information. In this period, developing server side of a web application required developer know one of many programming languages, which are PHP, Java, C#. Otherwise, JavaScript is the primary scripting language to make reactive website. The development process is not an easy task for junior developers because they have to learn many programming languages along with client side markup language.

The thesis is about building a content management system as internal project with Wiredelta. It is divided into three parts. The first part discusses about modern web application, single page application, and the MEAN stack, the second part writes about website construction, implementation and validation.

## 2 WIREDELTA – THE PROMISING STARTUP

Wiredelta (hereafter WD) was formed in the fall of 2012 by Mark Dencket and Thibaut Delarbre after experiencing numerous difficulties in finding affordable solutions for building a high-quality web application. WD's core business is to provide cost-efficient web and mobile applications for internet entrepreneurs while matching the quality offered by European web-agencies (Wiredelta, 2012).

The company's mission is to transform volatile ideas into impressive products at cost-efficient prices. WD realized how quickly and effectively developing web application with Node.js engine. There are plenty of successful testimonials displayed on the company website. To meet the needs of the market, WD has spent the majority of the company's time and effort in training the next generation of talented developers in Wiredelta Institute, hiring the best and growing a portfolio of web and mobile projects (Wiredelta, 2012).

Wiredelta offers a large amount of internships in Copenhagen, Denmark every year. It is a great chance for any students who are willing to gain more real world experience. Additionally, all Wiredelta projects come from the micro, small and medium business segment. Developers will receive more challenges when working on those business scales. They also have chances to meet and work directly with customers. Gaining soft skills is critically important as well (Wiredelta, 2012).

### 3 MODERN WEB APPLICATION

The strategy in web development has changed dramatically in the recent years, thanks to the improvement of information technology. The browser is no longer used to serve static information. JavaScript is used heavily to serve dynamic elements in the browser. The browser is becoming a kind of mini-operating system, which avails itself in the net of various data sources- the services of developer's servers (Jörg, 2016).

The modern web application, which is called web app, connects to the server to retrieve data dynamically. It only exists in the browser. When users approach the app for the first time, the app is rendered by the server and supports its services such as database access or transaction (Mozilla Inc, 2016). Deriving from modern web application, Wiredelta's internal project is designed to build application service provider called an APIs (Application Programming Interfaces) and use JSON (JavaScript Object Notation) as formatting language.

Atwood (2017), as a collar to Tim Bernes-Lee's Rule of Least Power, proposed Atwood's Lay which states that any application that can be written in JavaScript, will eventually be written in JavaScript. Ryan Dahl (2009), wrote Node.js, an open-source, cross-platform JavaScript runtime environment. After several years of enhancement, it has combined with Google's V8 JavaScript engine, an event loop and low-level I/O API, and has supported executing JavaScript code in the server side.

#### 3.1 Single Page Application

Instead of using numbers of HTML pages to display, SPA is a web application or website that is embedded in a single HTML page. The main goal of SPA is providing a more fluid user experience and richer interface. There is no limitation for the SPA. It can be a small application which simply provides CREATE, UPDATE, DELETE service, like a to-do list. SPA can scale into much more complex level with large amounts of requests. A Single Page Application contains countless libraries, templates and scripts in many folders depending on its complexity (Code School LLC, 2015).

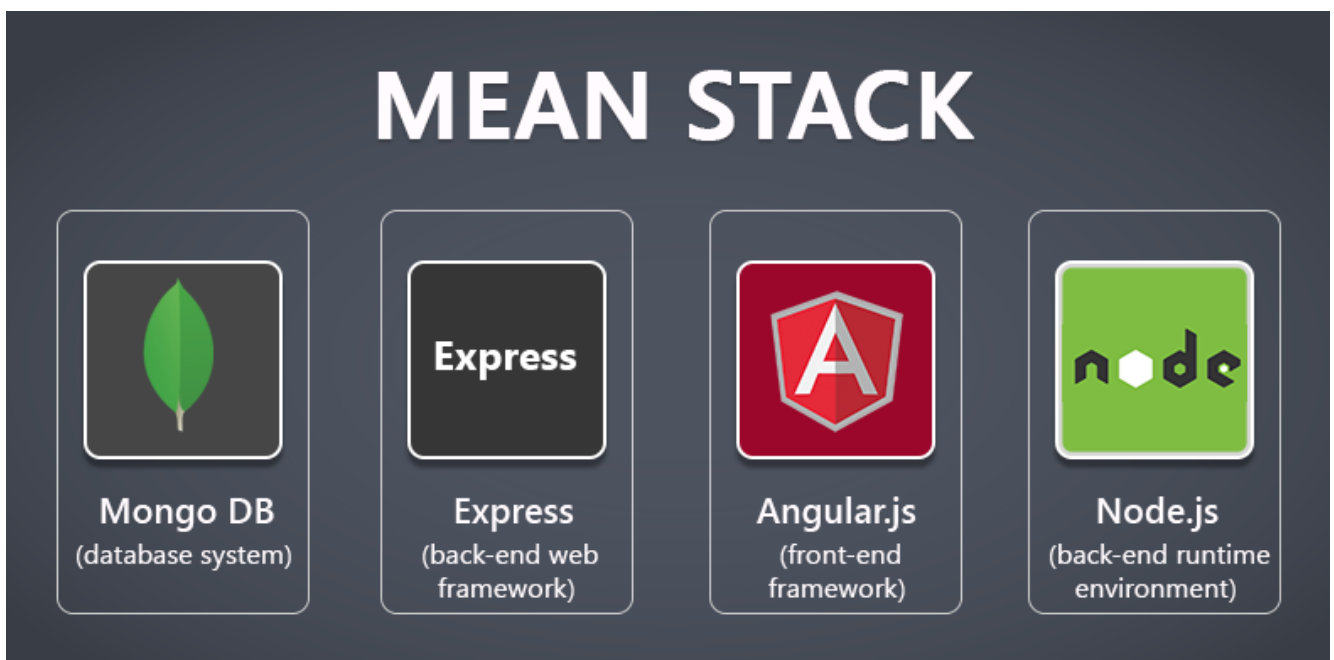
The advantage of choosing this type of web application is to be able to update parts of a web site without sending request and reloading a full page. This feature is believed to be the most interesting feature of



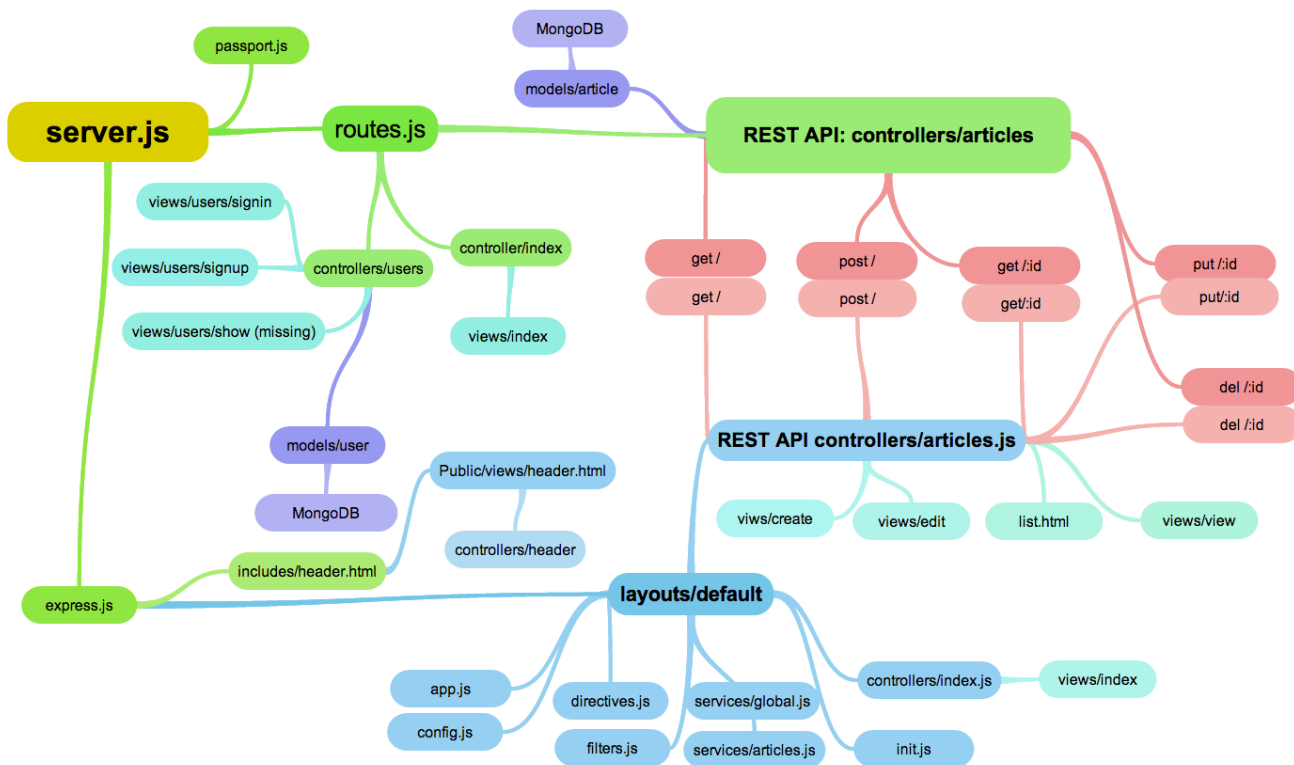
SPA. Nowadays, more developers are starting to approach SPA. It is much easier to develop and it is the best suit for RESTful API. From the start, everything is coded from the front-end of a web application using Ajax (to interacting with the server), HTML templates, a good MVC/MVVM framework, and of courses, an enormous amount of JavaScript (Wasson, 2013).

### 3.2 The MEAN Stack

In a full stack web application, JavaScript is the consistent language and syntax. The MEAN stack is comprised of four main JavaScript oriented technologies. MongoDB is the database, Express is back-end web framework, AngularJS is front-end web framework and Node.js is the back-end runtime environment. The reason why the stack is selected because developers will have improved view of greater picture by understanding the different areas and how they fit together. Frameworks and libraries are no longer limited in supporting user interfaces but also can be combined with other layers of applications. However, wrong architecture, low-level design and unbalance foundation in initial construction can dramatically affect the development process (Linnovate, 2014). Graph 1 explains the information of the MEAN Stack.



Graph 1 MEAN Stack elements (Prat, 2015)



Graph 2 Diagram of the MEAN stack (100PercentJS, 2013)

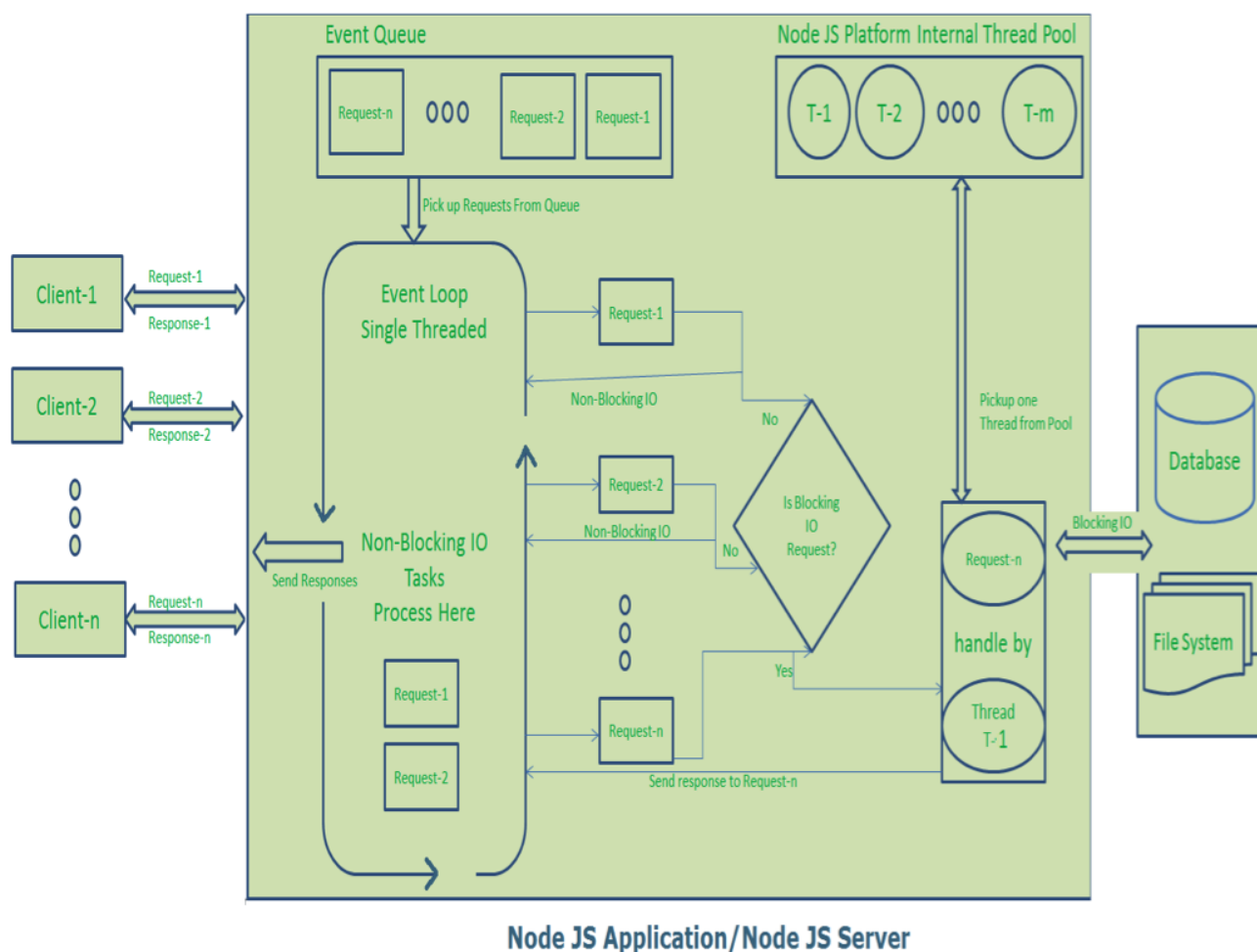
### 3.2.1 Node.js

Node.js is a software platform allowing developers to create web servers and build web applications on it. Meanwhile PHP needs to run separate web server program such as Apache or Internet Information Services, Node.js server can be configured when building an application. However, it is not limited in web projects. More phone apps and desktop applications are built on top of Node.js. (Node.js Foundation, 2012).

Node.js provides many utilities' packages in order to create a powerful web server that runs JavaScript on the server side, for example, asynchronous I/O, single-threaded or multi-threaded, sockets, and HTTP connections. When building web applications using other server-side programming languages, such as Java, PHP, and .NET, the server creates a new thread for each new connection. Assumed that there is a popular website receiving more than two million requests daily, developers probably need more servers or even to invest in more hardware. This approach works well as long as the server has enough hardware

infrastructure to provide services to the customers. When the number of clients exceed the server's ability, it starts to slow down and the customers have to wait (Holmes, 2015).

The single threaded, non-blocking I/O changes how a request is handled by the server. Instead of creating a new thread for each connection, the web server receives client requests and places them in a queue, which is known as Event Queue. The Event Loop picks up one client request from the queue and starts process it. If the request does not include any blocking I/O operation, the server process everything, creates a response and sends it back to the client. In the case that the request can block the I/O operation, there is a different approach. An available thread from the thread pool will be picked up and assigned to the client request. That thread has then processed the blocking I/O, processed the request and created a response to send back to the client (Jörg, 2016). Node servers can support tens of thousands of simultaneous connections. It does not allocate one thread per connection model, but uses a model process per connection, creating only the memory that is required for each connection (Monteiro 2014, 32). Grap 3 explains single threaded application.



Graph 3 Node.js Single Thread Event Model (Jörg, 2016)

### 3.2.2 Express

Even though Node.js is a great choice to make a web server, it is just a run time environment with many built-in modules. The default Node API is not able to handle complex applications as route management. A web framework is needed to do the heavy work. Express is a mature and flexible web framework to build web applications on top of the Node eco system. By default, the Express framework uses the Pug engine for supporting templates (Node.js Foundation, 2010).

Furthermore, Express reduces difficulties in setting up a webserver to handle incoming requests and return relevant response. That is the reason why it is the best choice to render HTML on the server side for large-scale application. It still provides an elegant set of features to deal with including SPA, the RESTful API, and the MEAN stack. Express follows good development practices, an oriented RESTful architecture that uses the main methods of the HTTP protocol (such as GET, POST, PUT, and DELETE), and very difficult-to-build complex web applications (Monteiro, 2014, 39).

### 3.2.3 AngularJS

At the moment, there is a enormous amount of front end libraries and frameworks supporting front end web development. AngularJS is one of them, which is designed to work with data directly in the front end. It allows developers to use HTML as template language and extend HTML's syntax. There are two core features: Data binding and Dependency injection. AngularJS reduces most of codes, which are needed to write. Developers can use Node.js, Express, MongoDB to build a data-drive web application and pass data via HTTP protocols. Angular works with data directly in the frontend and puts the HTML templates together based on the provided data. Developers need less resources with this approach (Google, 2010).

### 3.2.4 MongoDB

MongoDB is one of the most popular open source NoSQL databases written by C++. In 02/2015, MongoDB ranked at 4<sup>th</sup> in list of common databases. MongoDB is document-oriented, and it stores data in the key-value format, using binary JSON (BSON). It is agile, scalable and high performance. Node.js

and MongoDB unleashed all the power for high-performance web applications with Express (MongoDB Inc, 2008).

```

{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}

```

The diagram illustrates the structure of a MongoDB document. On the left, a JSON document is shown with keys in blue and values in black. On the right, four arrows point from the labels "field: value" to the corresponding key-value pairs: "name: 'sue'", "age: 26", "status: 'A'", and "groups: [ 'news', 'sports' ]".

Graph 4 MongoDB data scheme (MongoDB Inc, 2008)

Document in MongoDB has the same structure as JSON, which means developers can map data as a key-value. We can understand the document as a record in MySQL. Because MongoDB is written in C++, it is able to make a high-speed calculation, unlike other database management systems. Instead of writing a massive amount of SQL commands, MongoDB supports fairly complex and powerful queries in most use cases (MongoDB Inc, 2008).

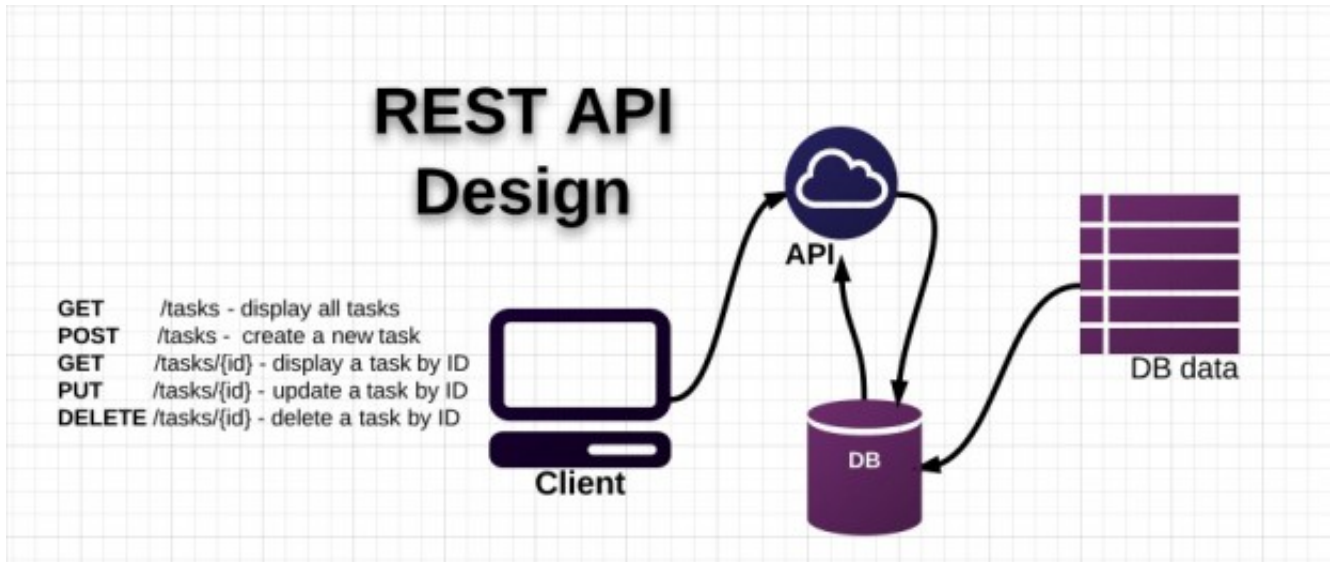
### 3.3 REST API and JSON

In legacy applications, XML-RPC is the method to transfer data via HTTP protocols. It is extremely complicated and uses a large number of bytes to encode objects. However, REST API that uses JSON become the universal connector for data on the internet. The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language (W3Schools, 2013).

#### 3.3.1 REST API

REST stands for Representation State Transfer, which is a stateless connection between clients and servers. It is modern client-server architecture using the HTTP protocol to communicate and JSON as

a formatting language. In a simple CRUD application as an example, REST API is used to create stateless interfaces for POST (create), PUT (update), GET (retrieve) and DELETE activities. The graph below shows the example of typical REST API design (Hunter II, 2013).



Graph 5 REST API design (Hunter II, 2013)

### 3.3.2 JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. No matter what knowledge of JavaScript developer has, it is easy to read and write. There is no problem for machines to parse and generate. Compared to XML, it is JavaScript-friendly. It also has the advantage of being generally easier to write by hand than XML. JSON has better JavaScript support and is much simpler to write (Ecma-International, 2001).

## 3.4 Content Management System

Content is information produced through the editorial process and ultimately intended for human utilization via a publication. A CMS is a server-based, multiuser software that provides several effective content management automatically. Editors can create new content, edit an existing one, or perform changes in content. Content manager will see differences right after commitment (Barker, 2016).

## **4 PLANING PHASE**

The problem came from Mark Dencket, the CEO of Wiredelta, after the training period. The developer who created the company's website left long time ago and it was inconvenient whenever he wanted to change the content of the website. It would be a grammar mistake, updating the image or adding new content. Hence, a task was an internal project before working with real clients, which is creating a simple content management system.

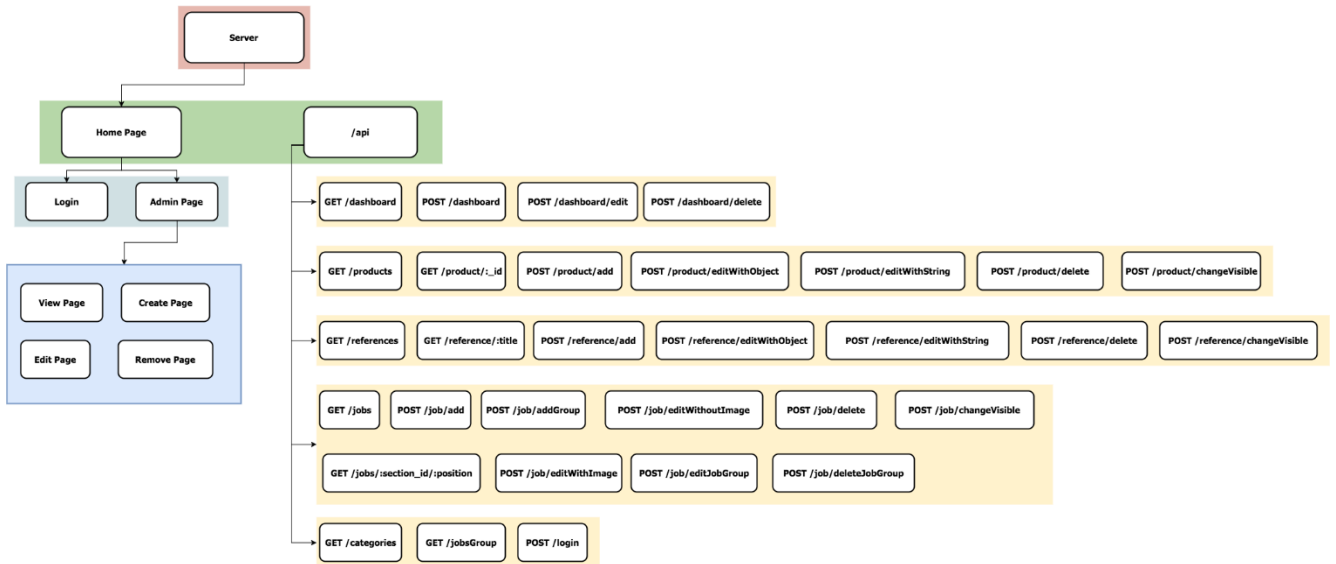
### **4.1 Goals and Target Users**

The main goal of the project was helping the CEO managing the website by building a web application. The website's requirements were creating a web application and using MEAN stack as mandatory for practicing purpose. The layout of the website was asked to be simple and modern enough to use. JSON was the main formatting language of the API and other developers could connect to the CMS via REST API. The target users were Mark Dencket and Aaron Morley (COO and Project Manager).

It was a tough and important challenge because there was only one developer responsible for the whole project. On the optimistic way, it was a chance to understand and learn how to make a complete website from scratch. Because there was only one developer in the team, tasks were required to be made on both server side and client side.

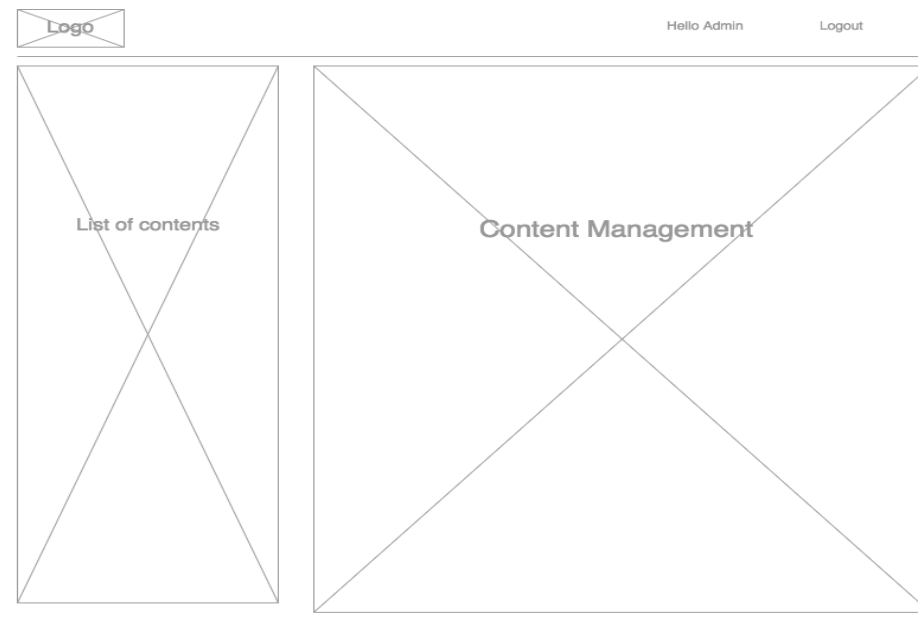
### **4.2 Site map and Wire frame Creation**

At this point of development process, sample data was created, which could give a customer an opportunity to judge how the entire site would look like. Based on the information gathered at the first stage, the sitemap was created. The graph below is the sitemap of the CMS website.



Graph 6 The site map of the CMS Application

The sitemap describes how the CMS was designed. From the view of clients using the web application, there are two main pages; log in and Admin Page. The Admin Page contain four main sub-pages for each content needed to be managed. Each sub-page corresponds to the CRUD elements. From the view of the developers who need to work with the API the site map displays URL end point and METHOD for each content of the company’s website. However, the site map allows clients to understand what the inner structure of the project looks like. It does not give any hints about the user interface. A wire frame is created to provide a visual presentation of the user interface. It only describes the position of elements that will be added to the pages. Graph 7 displays the wire frame of CMS.



Graph 7 The wire frame of CMS



## 5 INSTALLATION

Before moving forward, Node.js, MongoDB and other necessary modules need to be installed. There must be a stable internet connection during installation and development process. MacOS is the main operating system and other Linux distros can follow the installation guide. Windows users can easily find a guide on the internet.

### 5.1 Node.js

As the main purpose is coding the web application on the Node.js runtime environment, it is crucial to install Node.js. Furthermore, npm is a tool to install modules. npm makes it easy for JavaScript developers to share the code that they have created to solve particular problems, and for other developers to reuse that code in their own applications (npm Inc, 2010).

#### 5.1.1 Installing with GUI

Graph 8 displays the official page to download Node.js installation package. Depending on the local OS, users can download the necessary one. Developers should choose the most stable version in the LTS section. In the current tab, the page provides latest features for any operating system. However, they are not fully supported and can contain potential bugs. Downloading the most released version is not a wise move unless developers have been working for several years (npm Inc, 2010).

The screenshot shows the Node.js download page with the following content:

**Downloads**  
 Latest LTS Version: v6.10.3 (includes npm 3.10.10)  
 Download the Node.js source code or a pre-built installer for your platform, and start developing today.

**LTS**  
 Recommended For Most Users

**Current**  
 Latest Features

**Windows Installer**  
 node-v6.10.3-x86.msi

**Macintosh Installer**  
 node-v6.10.3.pkg

**Source Code**  
 node-v6.10.3.tar.gz

**Windows Installer (.msi)**

32-bit	64-bit	
32-bit	64-bit	
64-bit		
64-bit		
32-bit	64-bit	
ARMv6	ARMv7	ARMv8
node-v6.10.3.tar.gz		

**Additional Platforms**

**SunOS Binaries**

32-bit	64-bit
Official Node.js Docker Image	
64-bit le	64-bit be
64-bit	
64-bit	

**Docker Image**

**Linux on Power Systems**

**Linux on System z**

**AIX on Power Systems**

Graph 8 Node.js download page (Node.js Foundation, 2012)

## 5.1.2 Using terminal

Homebrew, the package manager on MacOS, is needed to install Node.js. It is easy to find how to install Homebrew at the official web site. When Homebrew is downloaded, *brew install node* is the command that allows a user to download and install Node.js (McFarland, 2014). Graph 9 will show how to check if node is installed on the local machine. The two commands are used to check both node and npm version.

A terminal window titled "1. nghi.thanh@dk-mac-ngth: ~ (zsh)" showing the following commands and outputs:

```
Last login: Mon May 8 19:42:31 on ttys000
nghi.thanh@dk-mac-ngth ~ > node -v
v6.10.0
nghi.thanh@dk-mac-ngth ~ > npm -v
3.10.10
nghi.thanh@dk-mac-ngth ~ > □
```

Graph 9 Checking Node.js on local machine

## 5.2 MongoDB

Following the demonstration with Node.js, MongoDB can be installed using GUI application or terminal commands. <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/> displays the best way for favorite choice. After MongoDB is extracted, the following command shows how to connect to the mongod using command line. `mongod --dbpath pathToStoringDataDirectory` is the command to start MongoDB in a local machine. The `pathToStoringDataDirectory` indicates where MongoDB can store a database in a directory. (MongoDB inc, 2008). If MongoDB is installed correctly, Graph 10 will show the result and connection port to MongoDB on local machine.

```

1. mongod --dbpath /Users/nghi.thanh/mongo (mongod)
nghi.thanh@dk-mac-ngth ~$ mongod --dbpath /Users/nghi.thanh/mongo
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten] MongoDB starting : pid=4143 port=27017 dbpath=/Users/nghi.t
hanh/mongo 64-bit host=dk-mac-ngth.local
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten] db version v3.4.4
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten] git version: 888390515874a9debd1b6c5d36559ca86b44babd
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.2k 26 Jan 2017
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten] allocator: system
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten] modules: none
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten] build environment:
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten]     distarch: x86_64
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten]     target_arch: x86_64
2017-05-08T20:43:06.740+0200 I CONTROL [initandlisten] options: { storage: { dbPath: "/Users/nghi.thanh/mongo" } }
2017-05-08T20:43:06.741+0200 I - [initandlisten] Detected data files in /Users/nghi.thanh/mongo created by t
he 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTiger'.
2017-05-08T20:43:06.741+0200 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=7680M,session_max
=20000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,pa
th=journal,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_lo
g=(wait=0),
2017-05-08T20:43:07.060+0200 I CONTROL [initandlisten]
2017-05-08T20:43:07.060+0200 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-05-08T20:43:07.060+0200 I CONTROL [initandlisten] **          Read and write access to data and configuration
is unrestricted.
2017-05-08T20:43:07.060+0200 I CONTROL [initandlisten]
2017-05-08T20:43:07.063+0200 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directo
ry '/Users/nghi.thanh/mongo/diagnostic.data'
2017-05-08T20:43:07.063+0200 I NETWORK [thread1] waiting for connections on port 27017

```

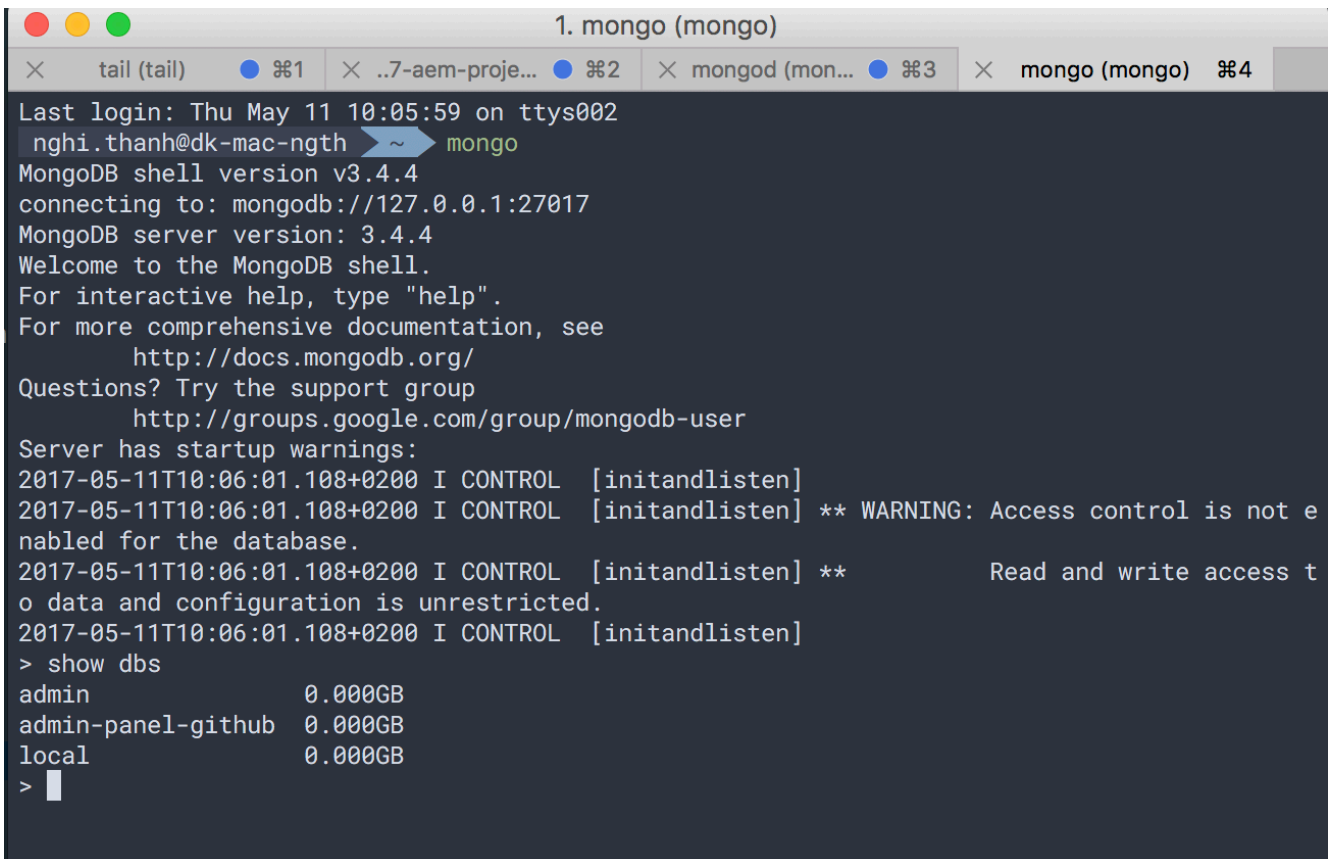
Graph 10 MongoDB successful connection on local machine.

### 5.3 Utility Modules

This is the last stage of the installation process. There are three main modules which are extremely useful during the development process and can be used as command line. *Bower* is another package manager for front end development. *Nodemon* monitors any changes in source and restarts the server. *Express-generator* quickly creates an application skeleton. At this point, everything is ready to create a powerful MEAN stack application. `npm install -g express-generator bower nodemon` is the command, which shows how to use npm to install node modules globally. After the command is executed, utility modules can be invoked as command lines.

### 5.3.1 Mongoose

When MongoDB is running and providing a port to connect as shown in Graph 10, developer can open the database and simply work with it by running the *mongo* command. It is the same as connecting to MySQL with terminal command line. A developer can create a new table and modify a data. Graph 11 shows work with MongoDB in a terminal on a local machine.



```

1. mongo (mongo)
tail (tail)  #1  ..7-aem-proje...  #2  mongod (mon...  #3  mongo (mongo)  #4
Last login: Thu May 11 10:05:59 on ttys002
nghi.thanh@dk-mac-ngth ~ mongo
MongoDB shell version v3.4.4
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.4
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
Server has startup warnings:
2017-05-11T10:06:01.108+0200 I CONTROL [initandlisten]
2017-05-11T10:06:01.108+0200 I CONTROL [initandlisten] ** WARNING: Access control is not e
nabled for the database.
2017-05-11T10:06:01.108+0200 I CONTROL [initandlisten] **          Read and write access t
o data and configuration is unrestricted.
2017-05-11T10:06:01.108+0200 I CONTROL [initandlisten]
> show dbs
admin                0.000GB
admin-panel-github   0.000GB
local                 0.000GB
>

```

Graph 11 Working with MongoDB in terminal on local machine

However, creating and removing data manually is not preferred at all. It takes time and contains syntax errors easily. That is why Mongoose Module is preferred. Mongoose provides a straightforward, schema-based solution to model an application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box (Automattic, 2011).



The other way to avoid callback function is using a promise object. It is one of the ECMAScript 2015 (6<sup>th</sup> Edition, ECMA-262) built-in support. If a function cannot return a value or throw an exception without blocking, it can return a promise instead. A promise is an object that represents the return value or the throw an exception that the function may eventually provide (Kowal, 2009). Kowal also wrote Q, a promise library for JavaScript. Compared to default promise object, Kowal's library provides more documentations, API and resources to follow. In the CMS application for Wiredelta, the promise is used in data immigration, making requests in front end development.

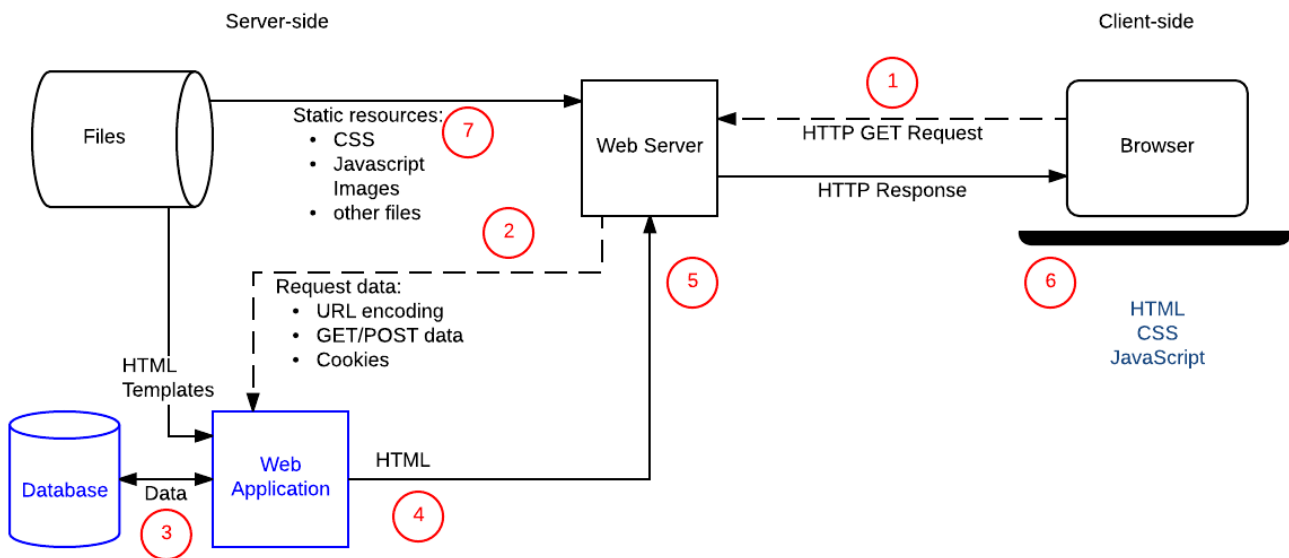
## 6 IMPLEMENTATION

For an implementation process to be successful, many tasks between different departments need to be accomplished in sequence. Companies strive to use proven methodologies and enlist professional help to guide them through the implementation of a system, but the failure of many implementation processes often comes from the lack of accurate planning in the beginning stages of the project due to inadequate resources or unforeseen problems that arise (Rouse, 2015).

The process of developing the application is divided into two main parts; server-side development and client-side construction. The goals of the server are rendering the main single HTML file and providing RESTful API. Meanwhile, the task of the front-end side is displaying the information and CRUD interface to manage the content. In personal opinion, the whole project is not difficult, but it contains an enormous amount of jobs to do before deploying.

### 6.1 Application Architecture

In this project, a browser retrieves data using the Hyper Text Transport Protocol (HTTP) to the web server. Other events happens in the browser (e.g. form submitted), an HTTP request is sent from the browser to the target server. The defined request handlers in receiving client’s request messages, process them and answer the web browser with an HTTP response message. Clients get the status code and message indicating whether or not the request succeeded (Mozilla Inc, 2016).

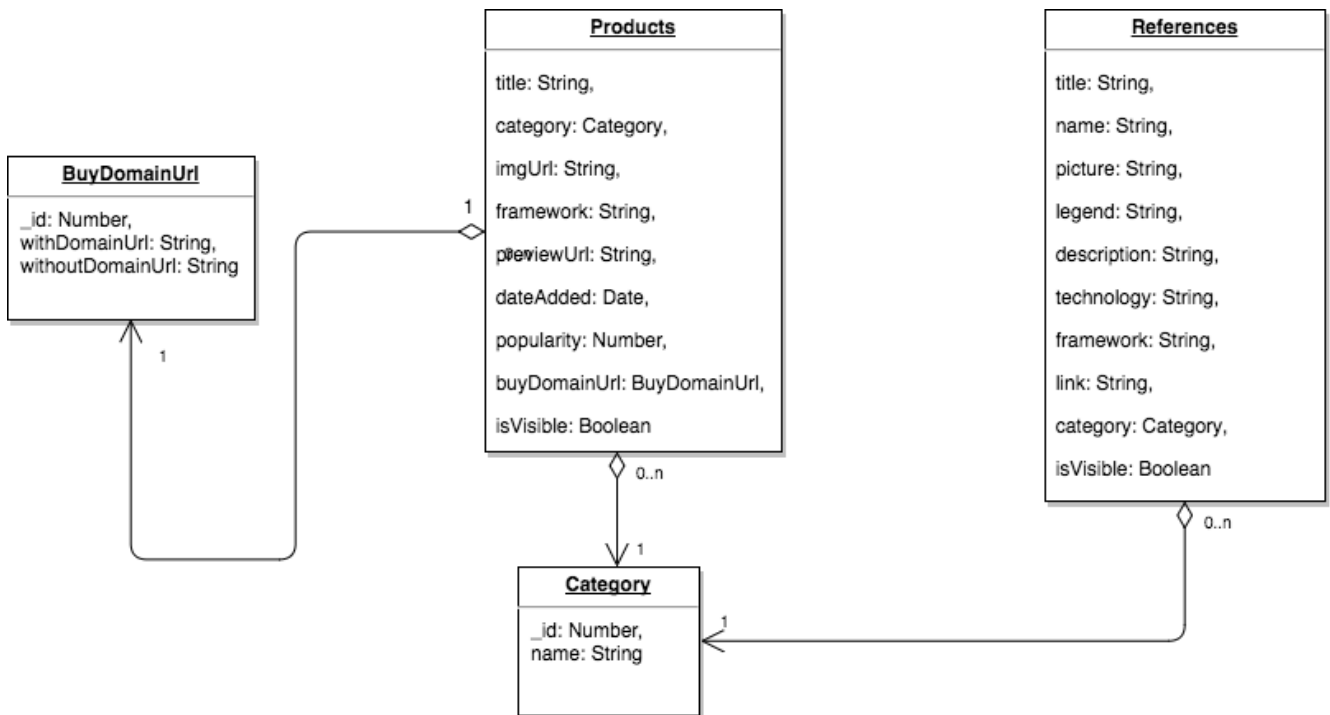


Graph 13 Application architecture (Mozilla Inc, 2016)



## 6.2 Database Architecture

Graph 12 describes the data types for each schema. Because MongoDB use JSON typed data, it had a different design than MySQL. Below is the UML diagram which describes schemas and their relationship. A product will have one category and buydomainurl object. Meanwhile, category and buydomain can have reference to many products. A reference has its own definition and it does not require buydomainurl.



Graph 14 Database architecture

## 6.3 JSON Sample Files, Data Importation

When the database is set up with schema connected with the server, database is needed to be imported before creating API. The database architecture in Graph 12 shows the idea how to make sample JSON. For example, the products.json contains an array of Product typed objects. There is no need to create sample BuyDomainUrl because it will not be used straightly. It is the advantages of MongoDB that developers can put sub objects in the main object without references. For each Product object in the products array, a new product is created. It retrieves properties and the value from the Product object. The object is then saved to the database and the process continues until the last element. The following piece of code reads the products.json file in assets/products and saves to database.

```
'user strict';

var path = require('path');
var jsonfile = require('jsonfile');
var mongoose = require('mongoose');
var Products = require('../models/products');
var Q = require('q');
var Async = require('async');

var productsPath = path.join(__dirname, '../assets/products/products.json');
var productsArrJson;

mongoose.connect('mongodb://localhost/admin-panel-github', function(err) {
  if (err) throw err;
  console.log('Mongoose database connected!!');

  Products.remove({}, function (err) {
    if(err) {
      throw err;
    } else {
      jsonfile.readFile(productsPath, function (err, products) {
        Async.each(products, function (value, callback) {
          var product = new Products();
          product.title = value.title;
          product.category = {
            _id: value.category._id,
            name: value.category.name
          };
          product.framework = value.framework;
          product.imgUrl = value.imgUrl;
          product.popularity = value.popularity;
          product.previewUrl = value.previewUrl;
          product.buyDomainUrl = {
            withDomainUrl: value.buyDomainUrl.withDomainUrl,
```

```

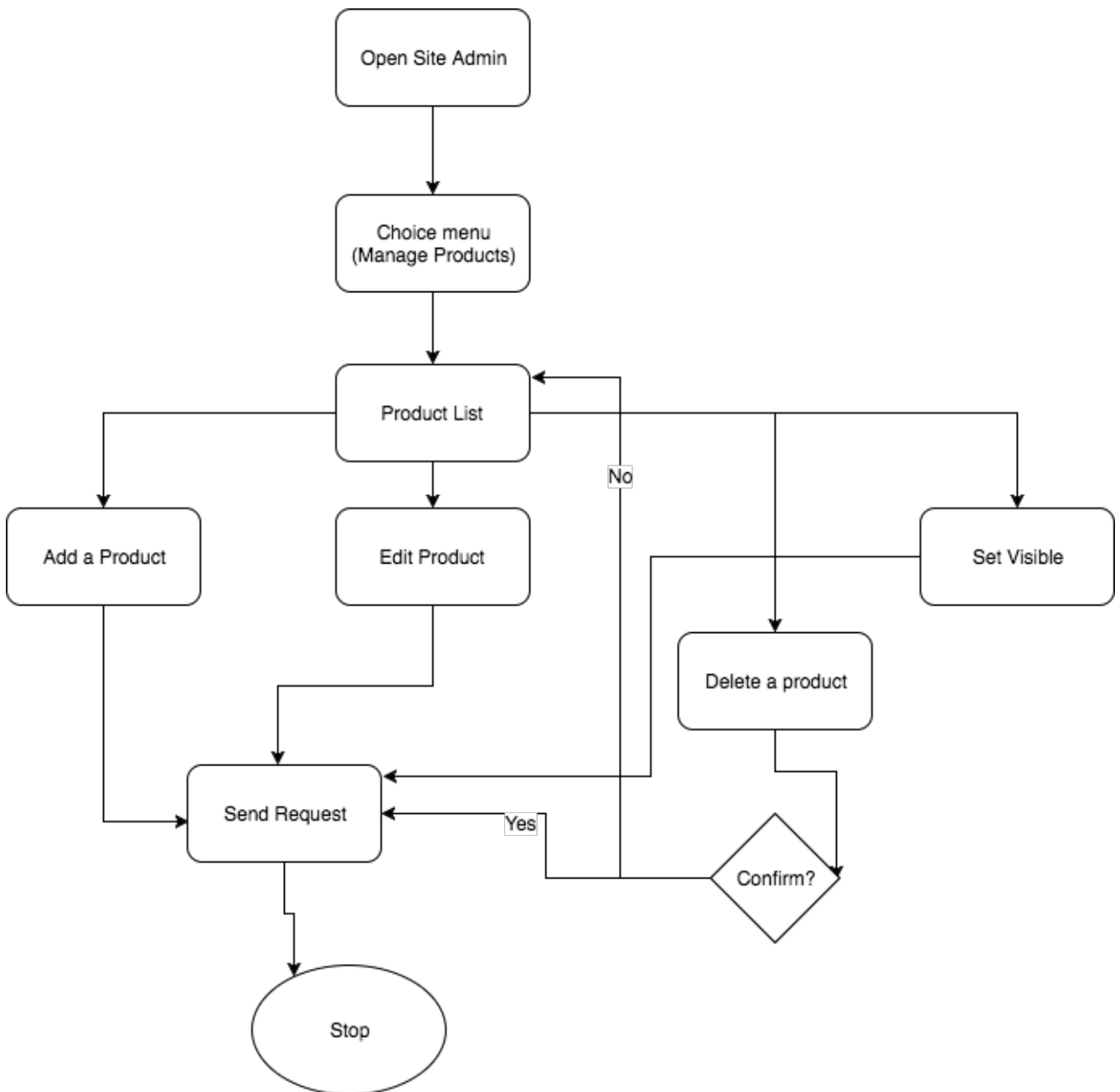
        withoutDomainUrl: value.buyDomainUrl.withoutDomainUrl
    };
    product.isVisible = value.isVisible;

    product.save(function (err) {
        if(err) {
            callback(err);
        } else {
            callback();
        }
    });
}, function (err) {
    if(err) {
        throw err
    } else {
        console.log('Initializing Products collection done!!!!');
        mongoose.connection.close(function(err) {
            if (err) {
                throw err;
            }
            console.log("Collection inserted and close db connection");
        });
    }
});
});
}
});
});

```

## 6.4 Basic Operation

After sample json files are integrated to MongoDB, its content can be modified. Graph 13 shows how the data creation and editing works in the Simple CMS. Starting at the site admin, users choose the content they would like to edit. The graph selects product as an example. Users will see Product List as the first view. In this page, users can go to Add a product, Edit Product and Delete a product. Set Visible is simply changes the checkbox value for each product in the Product List.



Graph 15 Product CRUD flow chart

### 6.4.1 Retrieving Data

When a client requires all products in the database, a browser sends a get request to `/api/products` without any extra request's configurations. In Express application, for each URL endpoint, there is a handler function accepting the request and responding result corresponding to client's demand and server's answer. The `Products` object represents the `Products` table in the database, and it will get every product and choose properties, which are selected, and return an array of `Products`. If there are any errors, the server will send errors in JSON format with status code. The following piece of code creates a function to return all products in the database.

```
var getProducts = function(req, res) {
  Products.find({}, {
    _id: 1,
    title: 1,
    category: 1,
    imgUrl: 1,
    framework: 1,
    previewUrl: 1,
    dateAdded: 1,
    popularity: 1,
    buyDomainUrl: 1,
    isVisible: 1
  }, function(err, products) {
    if (err) {
      return helpers.sendJsonResponse(res, 400, err);
    }
    helpers.sendJsonResponse(res, 200, products);
  });
};
```

Below is the result of retrieving data from the server. The page displays all products in a table. Each row of the table is title, framework, product's visibility on the main website, edit and delete button. In this web interface, users can go to add new product page by clicking Add button. Users can sort by framework, title or category with the red button on the top left of the table.

	Title	Framework	Category	Visible?	
1	example-0	express	Promotional	<input checked="" type="checkbox"/>	<a href="#">Edit</a> <a href="#">Delete</a>
2	example-1	angular	Promotional	<input checked="" type="checkbox"/>	<a href="#">Edit</a> <a href="#">Delete</a>
3	example-2	express	Ecommerce	<input checked="" type="checkbox"/>	<a href="#">Edit</a> <a href="#">Delete</a>
4	example-3	wordpress	Ecommerce	<input checked="" type="checkbox"/>	<a href="#">Edit</a> <a href="#">Delete</a>
5	example-4	angular	Promotional	<input checked="" type="checkbox"/>	<a href="#">Edit</a> <a href="#">Delete</a>
6	example-5	ios	Mobile	<input checked="" type="checkbox"/>	<a href="#">Edit</a> <a href="#">Delete</a>
7	example-6	android	Mobile	<input checked="" type="checkbox"/>	<a href="#">Edit</a> <a href="#">Delete</a>
8	example-7	html	Promotional	<input checked="" type="checkbox"/>	<a href="#">Edit</a> <a href="#">Delete</a>

Graph 16 Product List

### 6.4.2 Adding Product

When the user adds a new product, it is hidden in the company's website by default. The administrator needs to set product's visibility. To add a new product after fulfilling the form and sending POST request, the server will do two parallel tasks. The first one is initializing a new Product Object with information from the request and save. The second one will save the image in the file system. The following codes create a new product in the database and handle image upload at the same time. Below is the codes explain how to add a new product. After the codes is Graph 17, which displays the form to add a new product.

```
var addProduct = function(req, res) {
  upload(req, res, function(err) {
    if (err) {
      return helpers.sendJsonResponse(res, 400, err);
    }
  })
}
```

```

Async.parallel([
  function(callback) {
    var product = new Products({
      title: req.body.title,
      framework: req.body.framework,
      dateAdded: new Date(),
      category: {
        name: req.body.category,
        _id: helpers._idForCategory(req.body.category)
      },
      buyDomainUrl: {
        withDomainUrl: req.body.withDomainUrl,
        withoutDomainUrl: req.body.withoutDomainUrl
      },
      popularity: req.body.popularity,
      previewUrl: req.body.previewUrl,
      imgUrl: helpers.imgName(req.file.mimetype, req.body.title),
      isVisible: true
    });

    product.save(function(err, savedProduct) {
      if (err) {
        callback(err, null);
      } else {
        callback(null, 'save new product done!');
      }
    });
  },
  function(callback) {
    var pathForTempPic = path.join(pathForUploadPic, req.file.filename);
    var newName = path.join(pathForUploadPic, helpers.imgName(req.file.mimetype,
req.body.title));

```

```

fs.rename(pathForTempPic, newName, function(err) {
  if (err) {
    callback(err, null);
  } else {
    callback(null, 'rename new image done!');
  }
});
}
], function(err, results) {
  if (err) {
    return helpers.sendJsonResponse(res, 500, err);
  }
  delete req.file;
  helpers.sendJsonResponse(res, 200, results);
});
});
};

```

Graph 17 Add new product form



### 6.4.3 Editing A Product

This activity requires more efforts to finish. When adding a new product, the client must provide an image of the product as required. However, in editing mode, changing a product's image is not necessary. Hence, the case is divided into two handler functions. When the user only wants to change the detail of the product, the `editProductWithStringInput` function is invoked. The function will execute a series of sub functions. Firstly, the image's name is changed. Secondly, the result of finding a product by id will replace its content with the data from the body of request. In the case that client wants to change the product's image, the server invokes `editProductWithObjectInput` to handle image uploading. In this case, the series of three helper functions will be executed one by one. At first, the server will remove the old image. Secondly, the old product object will be replaced by modified one from the client. In the last stage, the server saves the new image to the file system. The following code shows how to edit products with or without an image.

```
var editProductWithStringInput = function(req, res) {
  Products.findById(req.body._id, function(err, product) {
    if (err || !product) {
      return helpers.sendJsonResponse(res, 404, 'Not found!');
    }

    Async.series([
      function(callback) {
        var oldName = path.join(pathForUploadPic, req.body.imgUrl);
        var newName = path.join(pathForUploadPic, req.body.title + req.body.imgUrl.slice(req.body.imgUrl.indexOf('.')));
        fs.rename(oldName, newName, function(err) {
          if (err) {
            callback(err, null);
          } else {
            callback(null, 'rename old image done!');
          }
        });
      },
      function(callback) {
```

```

product.title = req.body.title;
product.category = {
  _id: helpers._idForCategory(req.body.category),
  name: req.body.category
};
product.framework = req.body.framework;
product.buyDomainUrl = {
  withoutDomainUrl: req.body.withoutDomainUrl,
  withDomainUrl: req.body.withDomainUrl
};
product.dateAdded = new Date(req.body.dateAdded);
product.popularity = req.body.popularity;
product.previewUrl = req.body.previewUrl;
product.imgUrl = req.body.title + req.body.imgUrl.slice(req.body.imgUrl.indexOf('.'));

product.save(function(err) {
  if (err) {
    callback(err, null);
  } else {
    callback(null, 'saved to database done!');
  }
});
},
function(err, results) {
  if (err) {
    return helpers.sendJsonResponse(res, 500, err);
  }
  helpers.sendJsonResponse(res, 200, results);
});
});
};

var editProductWithObjectInput = function(req, res) {

```

```

upload(req, res, function(err) {
  if (err) {
    return helpers.sendJsonResponse(res, 400, err);
  }

  fs.readdir(pathForUploadPic, function(err, files) {
    if (err) {
      return helpers.sendJsonResponse(res, 400, err);
    }

    var index = _.findIndex(files, function(file) {
      return file === req.file.originalname;
    });
    var pathForTempPic = path.join(pathForUploadPic, req.file.filename);

    if (index >= 0) {
      fs.unlink(pathForTempPic, function(err) {
        if (err) {
          return helpers.sendJsonResponse(res, 500, err);
        }
        helpers.sendJsonResponse(res, 500, 'Duplicate filename, please rename the file before
post!!!!');
      });
    } else {
      Products.findById(req.body._id, function(err, product) {
        if (!product || err) {
          return helpers.sendJsonResponse(res, 404, 'Product not found!');
        }
        Async.series([
          function(callback) {
            fs.unlink(path.join(pathForUploadPic, product.imgUrl), function(err) {
              if (err) {
                callback(err, null);
              } else {

```



```
        } else {
            delete req.file;
            callback(null, 'saved to database done!');
        }
    });
}
], function(err, results) {
    if (err) {
        return helpers.sendJsonResponse(res, 500, err);
    }
    helpers.sendJsonResponse(res, 201, results);
});
});
}
});
});
};
```

Graph 18 below is the form of editing specific product. The edit page is divided into two parts. The first one displays input fields, which have default value. The form has an informative area to let user know the size of an image. Above the form is a light orange button. This button will redirect the user to the products list. The second part shows the image of the product.

The screenshot shows a web browser window with the URL `localhost:3000/admin/product/5908a795b084cc6f01e5dbc8`. The page title is 'Admin panel'. On the left is a sidebar with links for 'Dashboard', 'Products', 'References', and 'Jobs'. The main content area features a 'Back' button and a blue error message: 'Image must have Min/Max Width: 1349px; Min Height: 900px; Max Height: 6000px'. Below this is a form with the following fields:

- Title:
- Framework:
- Category:
- Date Added:
- With Domain Uri:
- Without Domain Uri:
- Image:
- Preview Uri:
- Popularity:

At the bottom of the form are 'Cancel' and 'Submit' buttons. To the right of the form is a placeholder image with the text '200x200'.

Graph 18 Edit Product Form

#### 6.4.4 Deleting A Product

There is a function which performs deletion based on the `_id` in the body of client's request. When the user clicks on the delete button, a POST is sent to the server. The function also makes sure to remove the product's image. A product is allowed to have only one image, which makes the function not complex. The following code shows how it works:

```
var deleteProduct = function(req, res) {
  Products.findByIdAndRemove(req.body._id, function(err, product) {
    if (err || !product) {
      return helpers.sendJsonResponse(res, 404, err);
    }

    fs.unlink(path.join(pathForUploadPic, product.imgUrl), function(err) {
      if (err) {
```

```

        return helpers.sendJsonResponse(res, 400, err);
    }
    helpers.sendJsonResponse(res, 200, {
        message: 'Product deleted'
    });
});
}6.);
};

```

## 6.5 Authenticating with JSON Web Token

As this is an internal project, the default design does not require to add strict authorization to REST API, but login system is mandatory for the web interface. In RESTful application, the session is not used to save user's information. In this case, JSON Web Token, an open standard (RFC 7519), 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 (Auth0, 2015). The following codes show how to handle a login request:

```

var login = function (req, res) {
    var post = {
        username: req.body.username,
        password: req.body.password
    };

    Admin.findOne({
        username: post.username
    }, function(err, admin) {
        if (err) {
            return helpers.sendJsonResponse(res, 500, err);
        }
        if (!admin) {
            return helpers.sendJsonResponse(res, 404, {

```

```

        message: 'Username not found'
    });
}

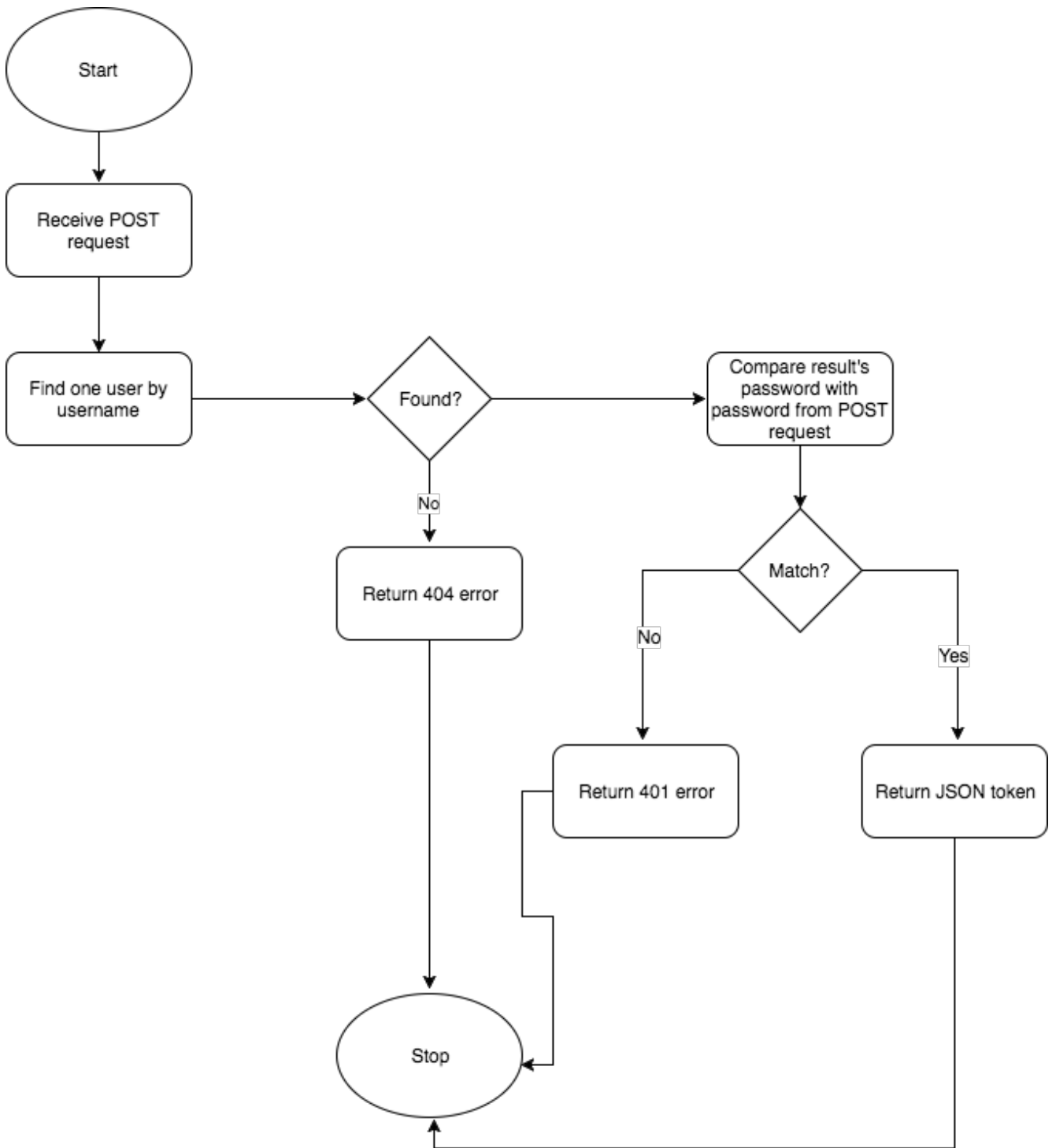
admin.checkPassword(post.password, function(err, match) {
    if (err) {
        helpers.sendJsonResponse(res, 500, err);
    } else if (!match) {
        helpers.sendJsonResponse(res, 500, {
            message: 'Invalid password'
        });
    } else {
        var token = jwt.sign({
            _id: admin._id,
            username: admin.username
        }, secretKey, {
            expiresIn: 2 * 24 * 60 * 60
        });
        helpers.sendJsonResponse(res, 200, token);
    }
});
});
};

```

Graph 19 is the result of the successful login request. The server responds to the client with a single hashed string. That string, when decoded, will display the id and username of admin and how long the token will last. In this application, the token will be saved at browser's local storage.



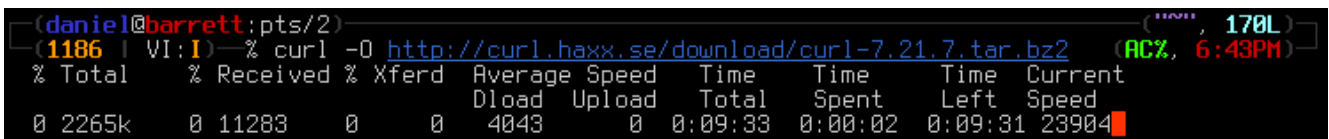




Graph 20 Login Activity

## 7 API TESTING

Before creating a web interface for users to work with the API, it is needed to make sure that it works as planned and there are errors sent back if needed. Developers normally can use the cURL command to transfer data using various protocols. cURL is a tool to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET and TFTP). The command is designed to work without user interaction (Haas, 2017). Graph 20 displays an example.



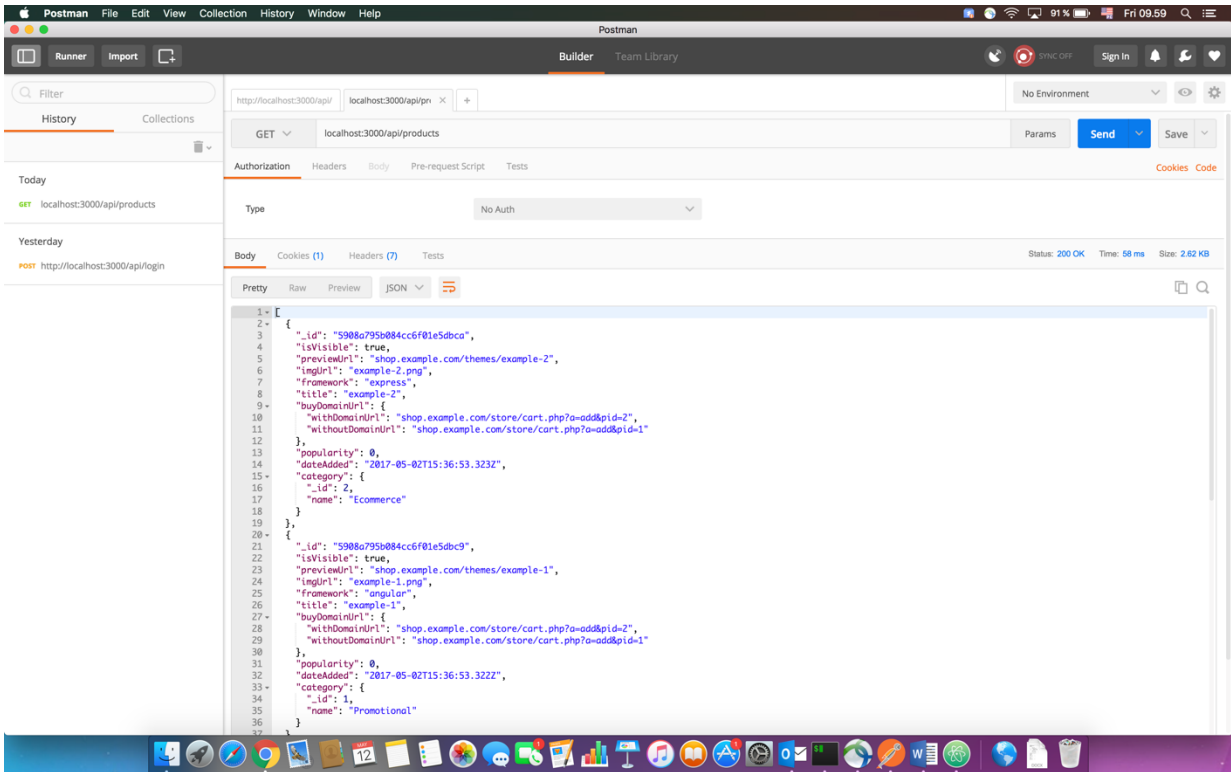
```
(daniel@barrett:pts/2)
(1186 | VI:~) % curl -O http://curl.haxx.se/download/curl-7.21.7.tar.bz2 (AC%, 6:43PM)
% Total % Received % Xferd Average Speed Time Time Time Current
      Dload Upload Total Spent Left Speed
  0 2265k  0 11283  0  0 4043  0 0:09:33 0:00:02 0:09:31 23904
```

Graph 21 cURL example

However, writing a cURL command requires an effort and it is easy to appear syntax errors. Postman is a Google Chrome app for interacting with HTTP APIS. It presents developers with a friendly GUI for constructing requests and reading responses. The people behind Postman also offer an add-on package called Jetpacks, which includes automation tools and, most crucially, a JavaScript testing library (Yockey, 2015).

### 7.1 Testing retrieving data

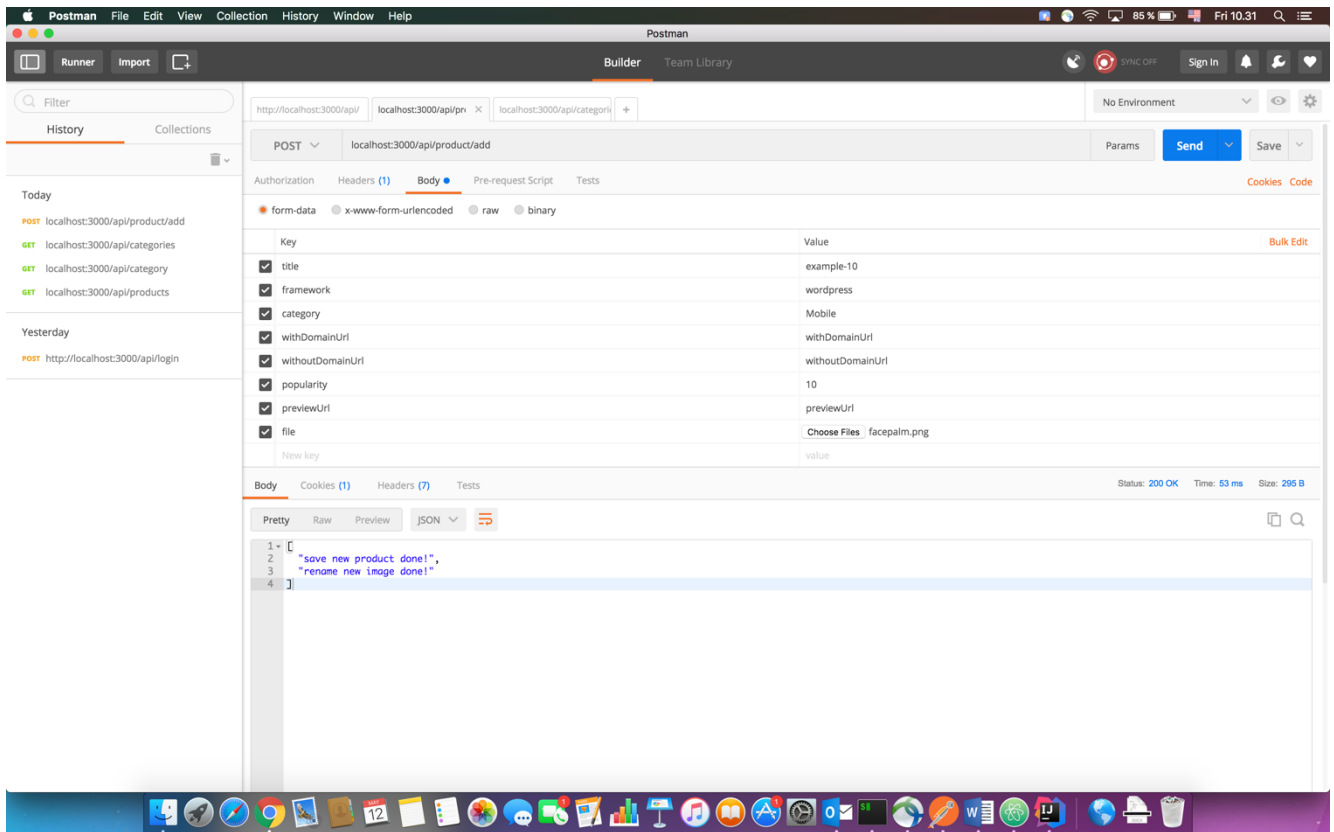
If front end developers want to get all the available products in the database, they have to send GET request to /api/products. The server does not require any authorization and header data in request. An array of objects is quickly sent back to them. However, in some case there are more than thousands of products in database, and front end developers have to set limits in query to optimize user experience. Graph 22 displays the successful request to get all products in the database.



Graph 22 All Products

## 6.2 Testing creating new data

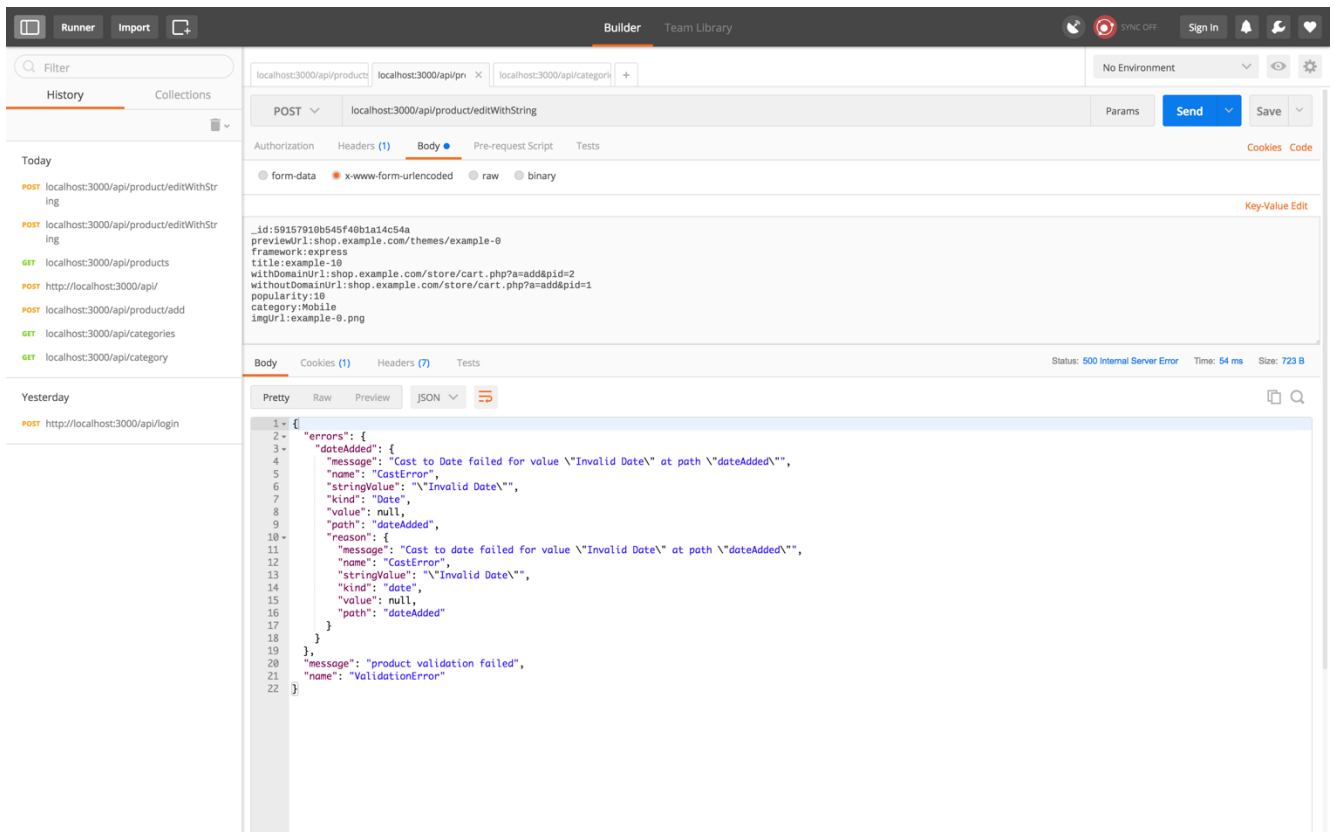
This section aims to add a new product by sending a POST request to `/api/product/add`. When adding a new product, front end developers must provide all required properties of the product in the body of the request; title, framework, category. Some details cannot be left as black, and they can be filled with sample string. The sever will make validations to avoid hacking issues. The graph 23 below shows how to use Postman to add a new product.



Graph 23 Adding new product

### 7.3 Testing editing data

In this case, a fail request will be made to make sure the server will respond with errors in JSON format. It is expected to receive an error with status code. The HTTP code 500 means Internal Server Error. MongoDB encounters a problem when validating data in request and cannot process the request further. The reason why the error appears because there is a missing field in the request, which leads to Internal Error. Graph 24 displays fail the request.



Graph 24 Error response

## 7.4 Testing removing data

Due to some reasons, the client no longer wants to keep the old products and they must be removed from the database. The web app sends post request having the id of product in its body, and the server will find it and send back the response when it finishes. At the moment, the implemented API in the server side is well tested. As long as the server responds with a success or error message, developers can start to create a web interface for users with high confidence. The graph 25 below displays the result when deleting a product.

The screenshot displays the Postman interface for a REST client. The active request is a POST method to the endpoint `localhost:3000/api/product/delete`. The request body is set to `x-www-form-urlencoded` and contains a single key-value pair: `_id` with the value `59157910b545f40b1a14c54b`. The response status is `200 OK`, with a response time of `37 ms` and a size of `273 B`. The response body is displayed in a pretty-printed JSON format:

```
1 - {  
2   "message": "Product deleted"  
3 }
```

The left sidebar shows a history of requests, including several POST requests to `localhost:3000/api/product/delete` and `localhost:3000/api/product/editWithString`, and GET requests to `localhost:3000/api/products`, `localhost:3000/api/categories`, and `localhost:3000/api/category`. A POST request to `http://localhost:3000/api/login` is also visible under the 'Yesterday' section.

Graph 25 Removing product

## 8 CONCLUSION

The aim of the thesis was to express the idea how to make a JavaScript-oriented application. It was a challenge, but contained a massive number of tasks needs to be finished. Meanwhile, there was only one developer in the team. The system was built from scratch as it was the best opportunity to learn MEAN stack. The key features when building a Web based CMS were user-friendly interface and easy to use API.

The application took two months to finish before deployment. The CMS successfully covered three main sections of the company's website. The primary features of Simple CMS are user authentication, fast and responsive web interface, editable content, and it is easy to use the REST API. This project will be useful for both non-technical users and developers. Every authorized user can create, update, delete data and set data visibility. Developers can directly retrieve or modify simply with HTTP protocol.

However, there are many opportunities for future development in this project. Incoming developer can provide multilingualism because WD aims to be an international competitor instead of being limited in the Danish market. This feature helps data displaying in multiple languages. The WYSIWYG HTML Editor is one of the mandatory features for every CMS software. It allows users to manage data description as it may contain HTML code when needed. It was one of the greatest achievements when the COO expressed that the project contributed incredible values to the company.

The internship at Wiredelta lasted for three months. The first month was focusing on learning and the CMS application took two months to finish. Hence, there was not enough time to have practical applications with clients. The site is still under improvement by new interns because the projects is considered as the valuable resource. For fellow students at Centria, Wiredelta not only offers attractive opportunities for anyone who would like to have more experiences in web development but business students also have chances to work with them.



## REFERENCES

- 100PercentJS. 2013. Introducing The MEAN stack. Available: <http://www.100percentjs.com/introducing-the-mean-stack>. Accessed 8 May 2017.
- Atwood, J. 2007. The Principle of Least Power. Available at: <https://blog.codinghorror.com/the-principle-of-least-power>. Accessed 8 May 2017.
- Auth0, Inc. 2015. What is JSON Web Token? Available: <https://jwt.io/introduction>. Accessed 9 May 2017.
- Automattic Inc. 2011. Mongoose Documentation. Available: <http://mongoosejs.com/>. Accessed 10 May 2017.
- Code School LLC, 2015. Single-page Applications. Available: <https://www.codeschool.com/beginners-guide-to-web-development/single-page-applications>. Accessed 10 May 2017.
- Cunningham, C. 2015. First Class. Available: <http://wiki.c2.com/?FirstClass>. Accessed 14 May 2017.
- Ecma-International, 2001. Introducing JSON. Available: <http://www.json.org/>. Accessed 9 May 2017.
- Google. 2010. AngularJS. Available: <https://angularjs.org>. Accessed 13 May 2017.
- Haas, J. 2017. What Is Curl And Why Would You Use It?. Available: <https://www.lifewire.com/curl-definition-2184508>. Accessed 14 May 2017.
- Holmes, S. 2015. Getting MEAN with Mongo, Express, Angular and Node. Available: <https://www.manning.com/books/getting-mean-with-mongo-express-angular-and-node>. Accessed 12 May 2017.
- Hunter II, T. 2013. Principles of good RESTful API. Available: <https://codeplanet.io/principles-good-restful-api-design>. Accessed 9 May 2017.

- Jörg, K. 2016. Programming Web Application with Node, Express and Pug. Available: <http://www.apress.com/us/book/9781484225103>. Accessed 9 May 2017.
- Kowal, K. 2009. Q Documentation. Available: <http://documentup.com/kriskowal/q>. Accessed 14 May 2017.
- Linnovate, 2014. MEAN.IO. Available: <http://mean.io>. Accessed 11 May 2017.
- McMahon, C. 2014. Async Documentation. Available: <http://caolan.github.io/async>. Accessed 14 May 2017.
- McFarland, D. 2014. How to Install Node.js and NPM on a Mac. Available: <http://blog.teamtreehouse.com/install-node-js-npm-mac>. Accessed 10 May 2017.
- MongoDB, Inc. 2008. Documents. Available: <https://docs.mongodb.com/manual/core/document>. Accessed 9 May 2017.
- MongoDB, Inc. 2008. What is MongoDB. Available: <https://www.mongodb.com/what-is-mongodb>. Accessed 9 May 2017.
- Monteiro, F. 2014. Learning Single-page Web Application Development. Available at: <https://www.packtpub.com/web-development/learning-single-page-web-application-development>. Accessed 8 May 2017.
- Mozilla Foundation. 2016. Introduction to the server side. Available: [https://developer.mozilla.org/en-US/docs/Learn/Server-side/First\\_steps/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction). Accessed 8 May 2017.
- Node.js Foundation. 2010. Express. Available: <https://expressjs.com>. Accessed 10 May 2017.
- Node.js Foundation. 2012. Downloads. Available: <https://nodejs.org/en/download>. Accessed 10 May 2017.
- npm, Inc. 2014. What is npm? Available: <https://docs.npmjs.com/getting-started/what-is-npm>. Accessed 14 May 2017.

npm, Inc. 2014. Build amazing things. Available: <https://www.npmjs.com>. Accessed 14 May 2017.

Odgen, M. 2012. Callback Hell. Available: <http://callbackhell.com>. Accessed 14 May 2017.

Posa, R. 2016. NodeJS Architecture – Single Threaded Event Loop. Available: <http://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop>. Accessed 8 May 2017.

Prat, R. 2015. 7 Good Reasons to use MEAN Stack in your next web project. Available: <http://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop>. Accessed 8 May 2017.

Raj, J. 2014. An Introduction to the MEAN Stack. Available: <https://www.sitepoint.com/introduction-mean-stack>. Accessed 10 May 2017.

Rouse, M. 2015. Implementation Definition. Available: <http://searchcrm.techtarget.com/definition/implementation>. Accessed 10 May 2017.

Stenberg, D. 1997. cURL. Available: <https://en.wikipedia.org/wiki/CURL>. Accessed 14 May 2017.

Wasson, M. 2013. ASP.NET – Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. Available: <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>. Accessed 10 May 2017.

W3Schools. 2013. JavaScript JSON. Available: [https://www.w3schools.com/js/js\\_json.asp](https://www.w3schools.com/js/js_json.asp). Accessed 14 May 2017.

Wiredelta, 2012. About us. Available: <https://wiredelta.com>. Accessed 12 May 2017.

Yockey, M. 2015. API Testing With Postman. Available: [https://seesparkbox.com/foundry/api\\_testing\\_with\\_postman](https://seesparkbox.com/foundry/api_testing_with_postman). Accessed 12 May 2017.