

MEAN.IO GUIDE

Building Real World
Applications From
Scratch

mongoDB

express

 ANGULARJS
by Google™

 node.js™



Introduction to the MEAN Stack

Building real world applications from scratch.

Rick H.

This book is for sale at http://leanpub.com/intro_to_mean_stack

This version was published on 2015-01-18



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

©2015 Rick H.

Tweet This Book!

Please help Rick H. by spreading the word about this book on [Twitter](#)!

The suggested hashtag for this book is [#meanguide](#).

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

<https://twitter.com/search?q=#meanguide>

Contents

Introduction	i
Introduction to MEAN.IO	2
What are we building?	2
Learning MEAN.IO	2
Why MEAN.IO will fail in the long run	3
Setting up a Developing Environment	3
Chapter Two: Planning a real world application	6
What the heck are we building?	6
Use Cases	6
MEAN file structure	13
Chapter Three: Refactoring code from MEAN.IO	16
Refactoring existing code	16
Dealing with ERRORS	17
Chapter Four: Writing lines of code	19
Integrating a custom Bootstrap 3 theme	19
Creating a users authentication system	19
Creating public and private CRUD pages	20
Chapter Five: Integrating open source projects into a MEAN application	26
Tackling Authentication	26
Authentication with tokens	27
Creating a static pages with MEAN	28
Chapter Six: Moving over a MEAN application to production	29
Separation of concerns with the source code	33
Example of application	34
Chapter Seven: Next Steps	35
Good Resources	35
Conclusion	35

Introduction

I want to thank you and congratulate you for downloading the book, MEAN.IO GUIDE | Building Real World applications from scratch. This book contains solid strategies and techniques on how to build a real world application with the MEAN Stack using the MEAN.IO framework.

Want to learn the MEAN STACK?

In this book, we will be learning about four pieces of software (MongoDB, ExpressJS, AngularJS, and Node.js) and how they combine to make the great MEAN stack and take that knowledge to the next level utilizing the MEAN.IO framework in order to create a real world application.

What will I Learn?

- Building CRUD operations using MongoDB and Mongoose
- Building REST API from scratch
- Building dynamic Routes with Express
- Integrating open source projects Bootstrap 3, Ghost Blog, and a progress bar
- Building scalable software applications
- Learn to use build tools like GRUNT
- So much more!

Why this book?

My intention is to get you from 0mph to 100mph, building real world applications with the MEAN Stack and instead of wasting time configuring and setting up stand alone applications the “normal” way we will be doing things unconventionally, this is the reason why this book is so short. I want you to see the 80% of the potential of the MEAN stack and I will leave the remaining 20% of the learning up to you. The book is meant to get you up and going as fast as possible with the MEAN STACK, that being said it's crucial for your success that you take the time to explore and ask questions about each of the individual projects (MongoDB, ExpressJS, AngularJS, and Node.js) in the corresponding places.

Assumptions.

I am assuming that you will be following along on a Unix or Linux system, windows users you might have to look into the documentation to dig out system specific configurations.

Let's get into it....

Introduction to MEAN.IO

What are we building?

For 2015 I am really motivated to take massive actions to get closer to my goals in different aspects of my life, sadly enough the current applications that are in the market are more geared towards individual tracking of goals. I believe if you really want to achieve a goal you have to be SMART(Specific, Measurable, Attainable, Realistic, Time-line) about how you achieve them. I am going to be tackling some challenging goals for 2015, and one of the ways I can stay committed to actually achieving the goals is if I put my reputation on the line, to be exact it would be on-line. This is probably one of the scariest and frankly hardest part about goals, letting others know what you are working on, because the moment you do. You are putting your reputation on the line to go out and complete the goal. The current software solution that exists did not quite meet my expectations, so I decided to create my own application and use some of those MEAN tools, and see what all the “buzz” is about.

The requirements of the application are going to be straight forward and should be easy enough to get an actual MEAN application under our belts. Specifications as follows

- Goals page(s)
- Individual Goal tracking
- About page
- Blog in Markdown
- Contact Form

Now there is not one single tool that will do all this, and yes I could use Word-press or create a Rails application using the normal web application frameworks. But we are out to create the worlds best “goals setting” application.

Learning MEAN.IO

You are here to learn MEAN, and we will be implementing this application using the [MEAN.IO](http://mean.io/)¹ starter boiler plate forked right off the [git repo](https://github.com/linnovate/mean)². For those that are completely new to the MEAN stack we will be covering the from scratch moto for the application.

¹<http://mean.io/>

²<https://github.com/linnovate/mean>



BECAREFUL

MEAN STACK'S and frameworks are still relatively new and they are not as well tested and maintained.

Why MEAN.IO will fail in the long run

If any of you have try to setup your own Angular, Mongo, Express, Node programming stack you know the pains of setting up everything correctly to find out that it's not the “*right*” way to do things. If you have not used [MEAN.IO](http://mean.io/)³ go and check them out before continuing reading. After playing around with the master branch of MEAN.IO, I decided that this tool is great but it's trying to do about a million things all at the same time. Don't get me wrong their is plenty of good things and just enough bad things to get me motivated to write my own tool to fix the problem.

What MEAN.IO did right?

Some of my favorite parts of MEAN.IO, that I just simply enjoy using are the following.

- MEAN MVC design pattern
- MEAN packages
- MEAN CLI tool
- Folder scaffolding
- Grunt pre-configured

What MEAN.IO did *WRONG*

But with all great things there is a down side to them, the main vision behind the MEAN project is to create this larger eco system where packages can be created and exchange back and forward with anyone that uses that stack. What are you talking about that sounds like a great idea! To some point it does sound like a solid idea but in implementation it's a whole other thing. That being said MEAN.IO is a great boiler plate to get started quickly building node application.

Setting up a Developing Environment

I'm assuming that you are going to be following along on a Linux server, if you don't know what I am talking about check out [setting up a LAMP stack](http://codewithintent.com/videos/lamp-tutorial/)⁴. Once you have a LAMP box, let's jump right into installing the prerequisites.

³<http://mean.io/#/>

⁴<http://codewithintent.com/videos/lamp-tutorial/>

Install NodeJS, CLI tools and NPM

```
1 sudo apt-get update
2 sudo apt-get install nodejs
3 sudo apt-get install npm
4 sudo npm install -g mean-cli
```

If you are in ubuntu 14.04, the apt-get install should work flawless with your environment, other Linux distributions might require more configurations.

Setup Apache with nodejs

```
1 touch /etc/apache2/sites-available/knownrick.conf
2 vim /etc/apache2/sites-available/knownrick.conf
```

Example of an apache configuration

```
1 <VirtualHost *:80>
2     ServerAdmin info@knownrick.com
3     ServerName knownrick.com
4     ServerAlias www.knownrick.com
5     ProxyRequests off
6
7     <Proxy *>
8         Order deny,allow
9         Allow from all
10    </Proxy>
11
12    <Location />
13        ProxyPass http://localhost:3000/
14        ProxyPassReverse http://localhost:3000/
15    </Location>
16
17 </VirtualHost>
```

Activate the site

```
1 sudo a2enmod proxy_http
2 sudo a2ensite knowrick.conf
3 sudo service apache2 reload
```

Now on your local machine you, want to tell it about the newly created site on your local server. Let's get it done, with the following command.

Update your host file

```
1 sudo vim /etc/hosts
2 192.168.56.101 www.knowrick.com
```

Finally the last part let's kick off this application!

Initialize your MEAN application

```
1 cd /var/www
2 mean init knowRick
3 cd knowRick
4 npm install
5 bower install
6 grunt
```

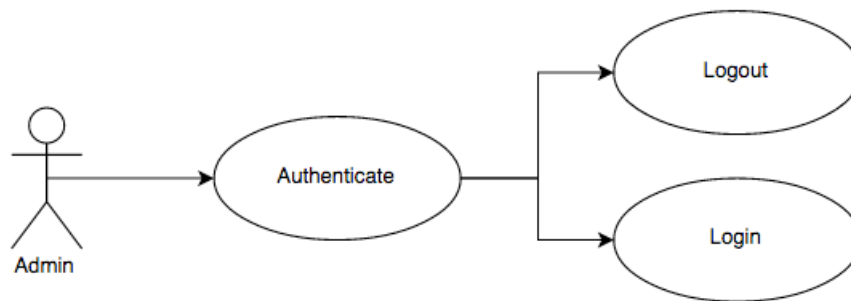
At this point you should have a complete MEAN application ready to go, and that will do it for Chapter 1! Congratulations give your self pat on the back and get ready for Chapter 2.

Chapter Two: Planning a real world application

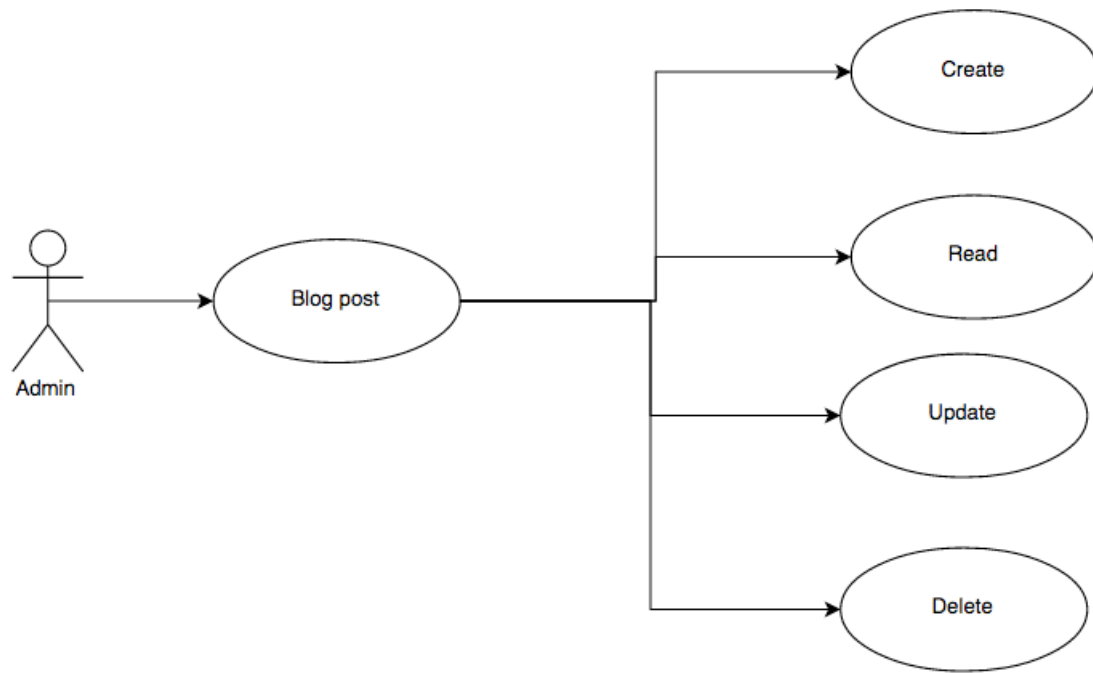
What the heck are we building?

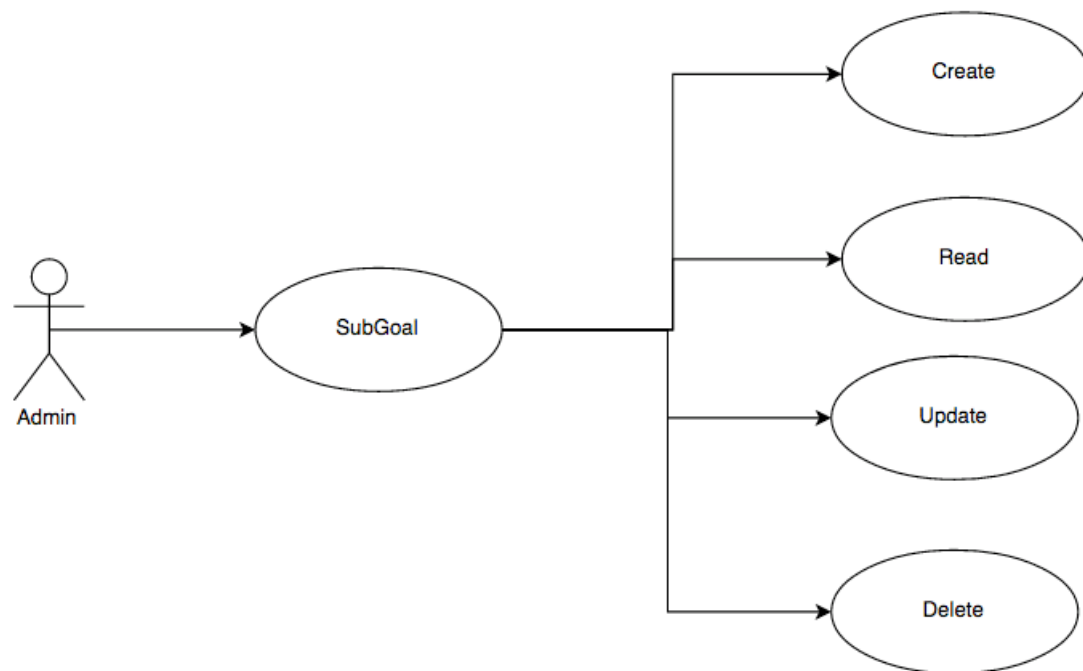
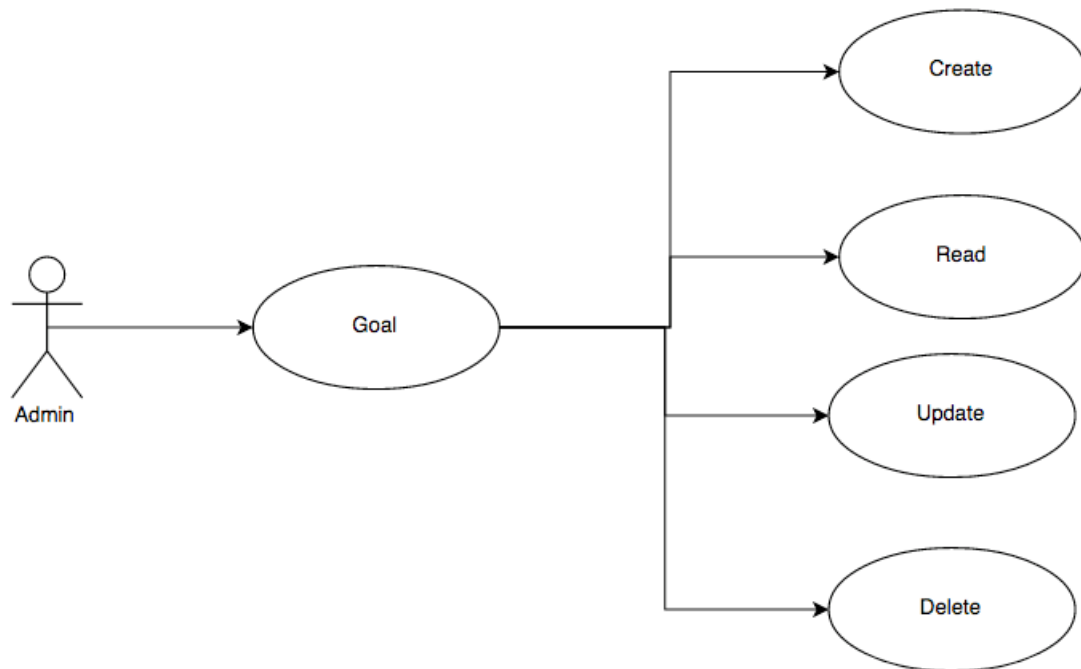
It's always a good practice to reflect on what you are building before you ever build anything. What am I talking about? "The fastest code is the one that you never write". Code is always more costly to modify after you have written it, this is why we are going to be reviewing the system requirements and use cases before we write any code.

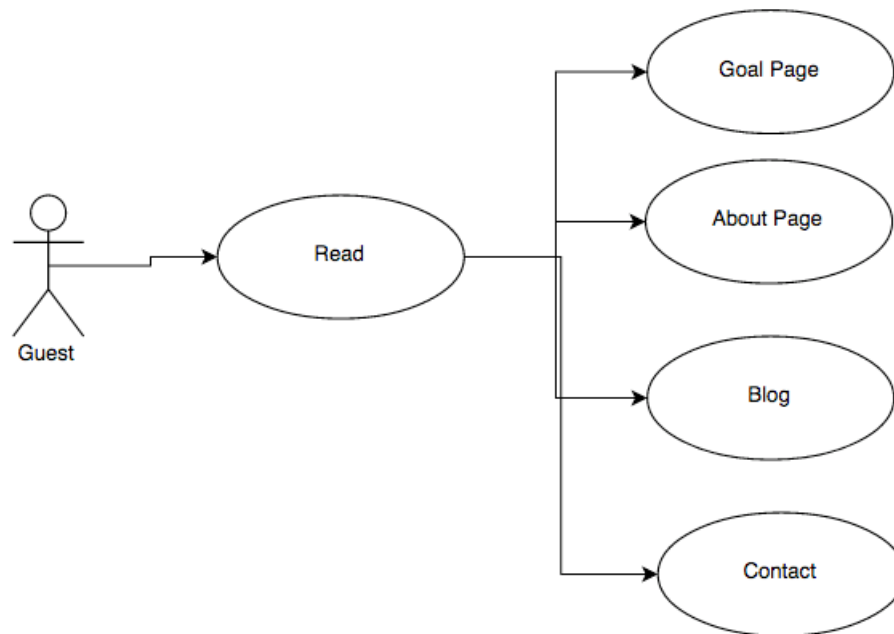
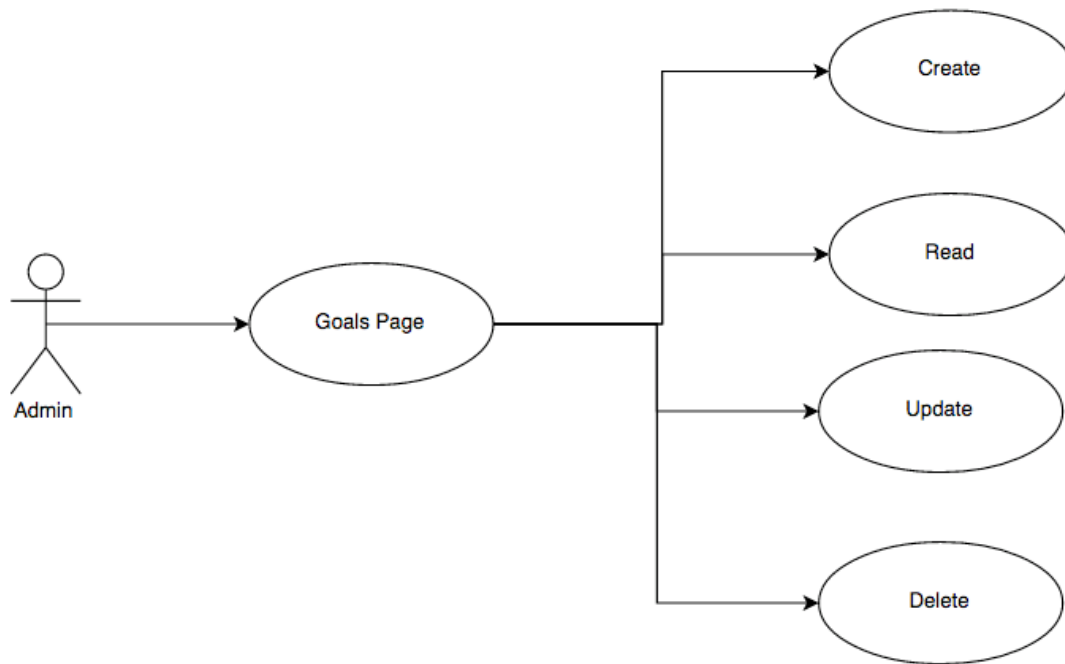
Use Cases

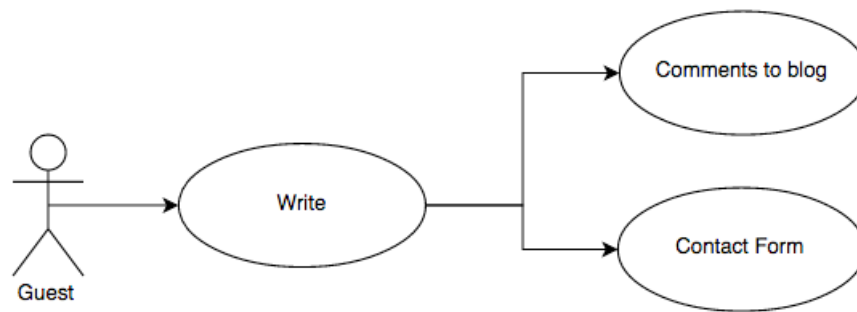


Authentication



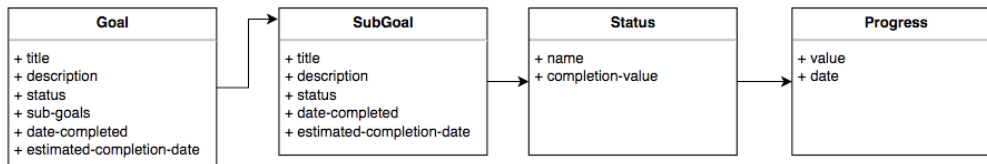
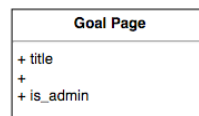
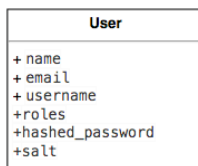




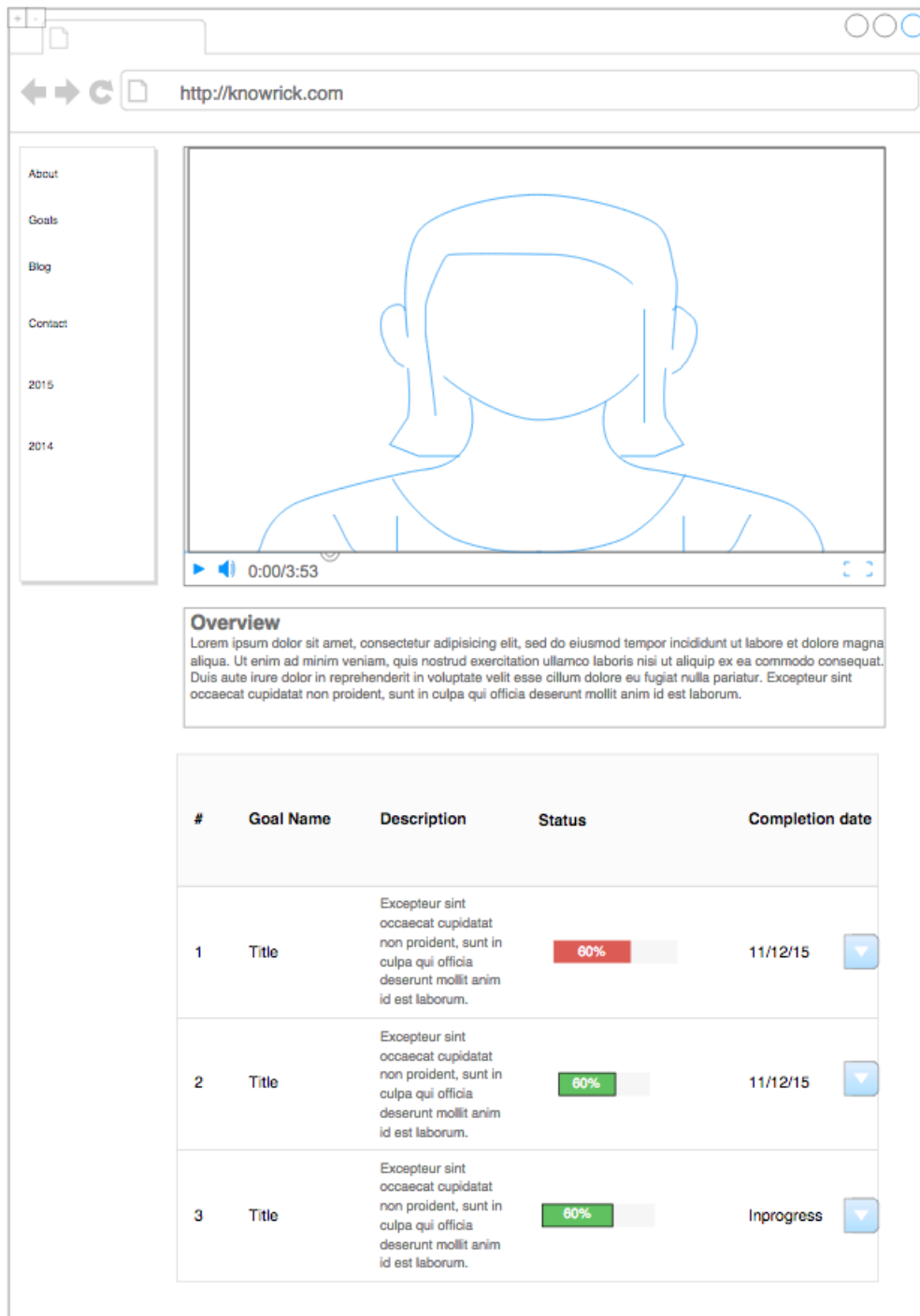


Database Design


Database Design



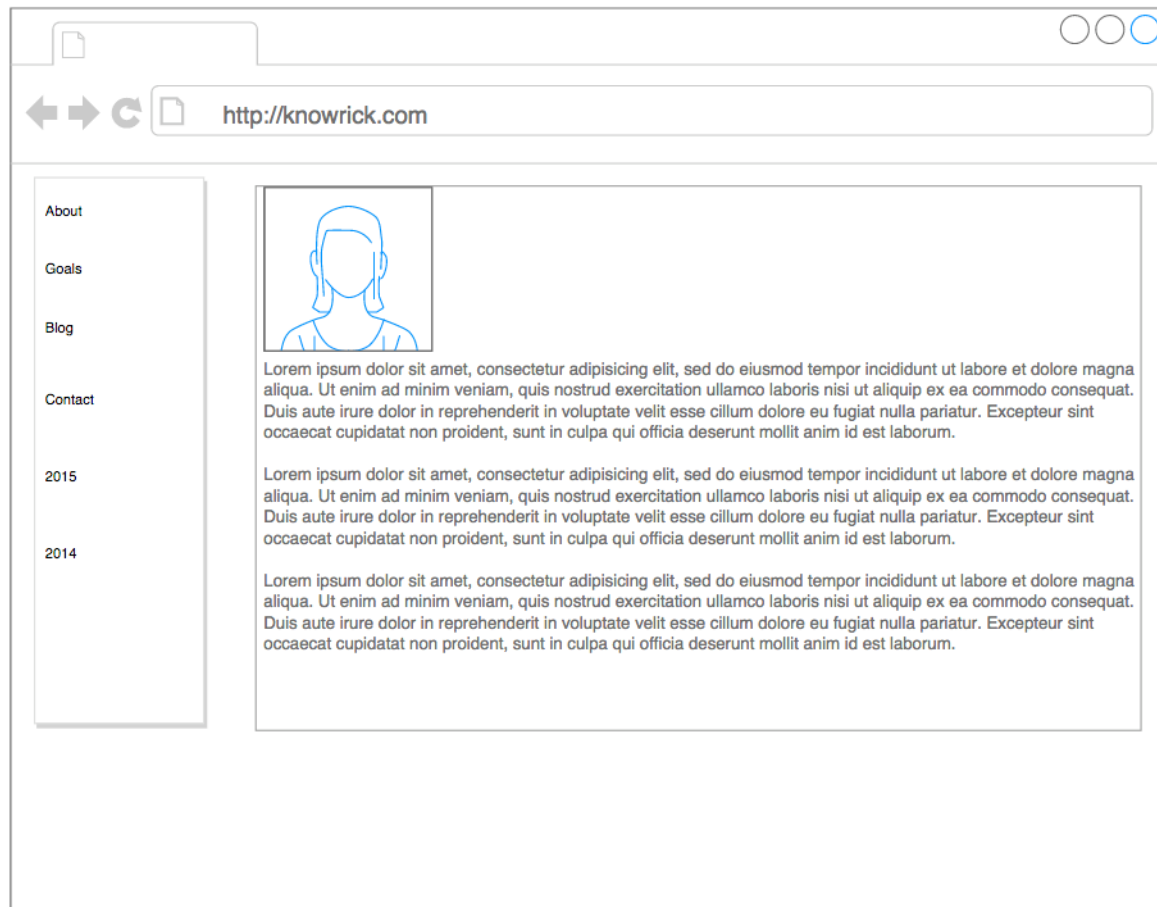
Prototypes



Subgoals

1	Title	Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.	60%	11/12/15	
Goal Name	Description	Status	Completed		
Goal Title	Excepteur sint occaecat cupidat.	60%	15 Sep, 8:56 AM (2013)		
Goal Title	Excepteur sint occaecat cupidat.	60%	15 Sep, 7:12 AM (2013)		
Goal Title	Excepteur sint occaecat cupidat.	60%	15 Sep, 4:34 AM (2013)		
Goal Title	Excepteur sint occaecat cupidat.	60%	15 Sep, 2:08 AM (2013)		
Goal Title	Excepteur sint occaecat cupidat.	60%	15 Sep, 8:56 AM (2013)		
Goal Title	Excepteur sint occaecat cupidat.	60%	15 Sep, 7:12 AM (2013)		

About Page



THIS IS NOT COMPLETE!!

I know that the following system design is not complete, and by all means I don't want it to be complete. I am creating this guide for a real application and as with many system that go through the software development cycle the applications tend to change and we find issues that we did not anticipate from the beginning this includes business rules, functional requirements, system testing and limitations that we do not perceive at this moment.

MEAN file structure

As good software engineers we need to clean up the project and prepare it for our application, that we will be creating.

Commit changes

```
1 sudo rm -R .git/
2 git clone https://github.com/rick4470/knowRick.git
3 mv knowRick/.git/ .
4 date
5 git commit -am "init project, forked mean.io Wed Dec 24 00:19:31 MST 2014"
```

Now that you got a good standing project, let's look over the file structure of the mean application and get familiar with what's what.



If you have not worked with a node application before this folder structure may look a bit intimidating. But remember at the end of the day they are just files and you own them. You did setup the right permissions? The break down goes as follows

***.bower**

You can ignore most of this folders, you might need to look inside once in while but rarely. If you would like to learn more about [bower](http://bower.io/)⁵ check them out.

config

Here is where a large amount of the application actually lives, you can think about this like a bootstrap file if you come from a zendframework world. If you don't think about it like your starting point to the entire application.

node_modules

All of your node packages and dependencies live here. This directory you more then likely never need to go inside of it, just like the bower directory.

Packages

Here is where your actual applications are going to be stored. Here is where we are going to be doing most of work, so make sure you get comfortable being in that folder.

Tools

This is a recent folder that the mean.io team recently put into the project, so if you know more advance features like testing, switching your build processes this is the folder to do it in.

That's it for Chapter 2! Congratulations give your self pat on the back and get ready for Chapter 3.

⁵<http://bower.io/>

Chapter Three: Refactoring code from MEAN.IO

Refactoring existing code

We are going to be needing to clean up the project before we can start adding any code to the application, in the ideal world this would be as simple as running the following command.

Remove a module

```
1 mean uninstall MODULENAME
```

Since the project has a very tight coupling dependent modules we can not just run this command and start cleaning up the boiler plate code so we have to slowly start to refactor in this case delete components that are not needed in the project. Make sure if you are following along with the git repo that you check out the right commit.

Check out the right commit

```
1 git checkout 2591bcc51cb18e817d329d74fb74c8515bf42194
```

Let's go ahead and remove the mean-admin module that comes by default with the project.

Remove package

```
1 mean uninstall mean-admin
```

Did you catch that? the following command did work although I mentioned that it would not. Don't worry this is because the module was install in the following directory.

```
1 knowRick/packages/contrib/mean-admin
```

Now let's remove the following directory we will no longer will be needing. Don't worry it's just an empty directory.

```
1 rm -R packages/contrib/
```

Let's go ahead and clean up this project even more, the following two commands are going to be deleting the themes folder and the articles module, our requirements don't call for this features so let's get rid of them.

```
1 rm -R articles
2 rm -R theme
```

Dealing with ERRORS

As previously mentioned the themes package is coupled with other modules so if you try to compile and run the application your console will log out a nasty error like the following one.

```
1 Error: [ng:areq] Argument 'ThemeController' is not a function, got undefined
2 http://errors.angularjs.org/1.3.2/ng/areq?p0=ThemeController&p1=not%20a%20functi\
3 on%2C%20got%20undefined
4   at http://knowrick.com/modules/aggregated.js:86:12
5   at assertArg (http://knowrick.com/modules/aggregated.js:1583:11)
6   at assertArgFn (http://knowrick.com/modules/aggregated.js:1593:3)
7   at http://knowrick.com/modules/aggregated.js:8339:9
8   at http://knowrick.com/modules/aggregated.js:7513:34
9   at forEach (http://knowrick.com/modules/aggregated.js:353:20)
10  at nodeLinkFn (http://knowrick.com/modules/aggregated.js:7500:11)
11  at compositeLinkFn (http://knowrick.com/modules/aggregated.js:6999:13)
12  at compositeLinkFn (http://knowrick.com/modules/aggregated.js:7002:13)
13  at compositeLinkFn (http://knowrick.com/modules/aggregated.js:7002:13)
```

No problem! Let's refactor this controller out of the project.

```
1 vim knowRick/packages/system/server/views/layouts/default.html
```

delete the following line

```
1 data-ng-controller="ThemeController"
```

Now that we have all of this out of the way the project should be somewhat of a workable project to start developing some of the features for the goal's application. Let's create a package for the application

```
1 mean package knowrick
```

This will create a directory in the packages parent directory `knowRick/packages/custom/knowrick`, Now I don't necessarily like having my project with in other folder so let's go ahead and move it up to the parent folder

```
1 mv knowRick/packages/custom/knowrick knowRick/packages
2 rm -R custom/
```

Now what I did is not necessarily the MEAN.IO way of doing things, and this is frankly frowned upon. But hey we forked off the project no looking back now. **ONWARDS!**

That's it for Chapter 3! Congratulations give your self pat on the back and get ready for Chapter 4.

Chapter Four: Writing lines of code

We will be completing some of the system requirements and really take our boiler plate to the next level.

```
1 git checkout 966dea79bf1a4a96f544e2edc89229d343e97390
```

We are not quite done cleaning up the project, we got to remove one more folder.

```
1 rm -R knowRick
```

Integrating a custom Bootstrap 3 theme

Let's go ahead and start by integrating a theme into our application, per the system requirements we need something that will fit our prototypes mockups. For the project we will be using the following [theme](#)⁶. Now we have to integrate the files that we receive from the designer into the MEAN application. The best location for this would be in the following folder

```
1 mkdir knowrick/packages/system/public/lib/
```

Now make sure you move over your assets into this folder, the reason why we are not using knowRick/packages/system/public/assets directory to store the files there is because our theme is different then our application. This way we can make changes and override anything from the [theme](#)⁷ with out cluttering our application with information that we don't need.

The theme should of provided the HTML markup that your application will be needing, let's go ahead and change the templates to fit our markup from our theme.

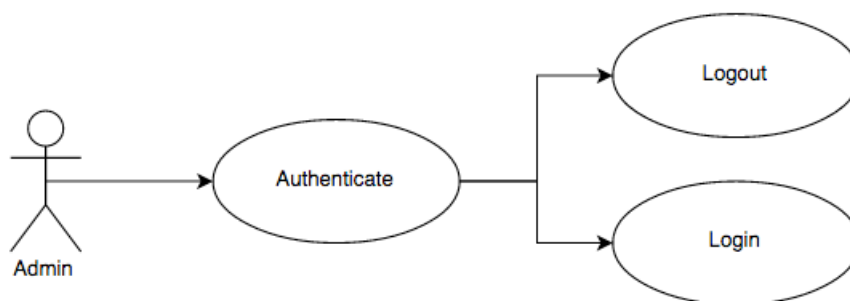
Creating a users authentication system

⁶<https://wrapbootstrap.com/theme/rebound-portfolio-responsive-template-WB0JSTNH4?ref=codewithintent>

⁷<https://wrapbootstrap.com/theme/rebound-portfolio-responsive-template-WB0JSTNH4?ref=codewithintent>


```
1 git checkout fc9b8b5b5517c9af06e6aa4da3963ea3ba5c8a44
```

After getting the theme in place let's get our authentication in order. If you recall from Chapter 2 of this tutorial our system requirements calls for an administrator.



auth example

Now we are not going to be allowing any more users to login into the application, by definition this will eliminate a major part of the authentication that the MEAN stack provides for us but not all of it. If you are not familiar with [passport](http://passportjs.org/)⁸ make sure you look over there documentation about strategies of authentication that you can use with your application, since our application only requires an administrator role to login to do updates on the data. We will be using a local strategy to authenticate the user.

One of the aspects of the authentication that I did not perceive when I originally mocked up the prototype, is how can we restrict access to everyone except only allow the right users to sign up and be part of the admin group? Since I am not thinking of opening the application to any one else besides those that want to get accesses to the admin functionality, I have created a work around of to only allowing users to register if they have a secrete invitation code. This way I can control the flow of users allowed for now.

Creating public and private CRUD pages

Now that we have our authentication working that way we want. We need to create our public facing pages and our CRUD pages that will allow us to manage our goals. We are going to be needing the following pages available with the right permissions.

```
1 git checkout ada4c9a1b5cdf6f57ee34dc3f90d280366ec1424
```

URL Mapping Client Side

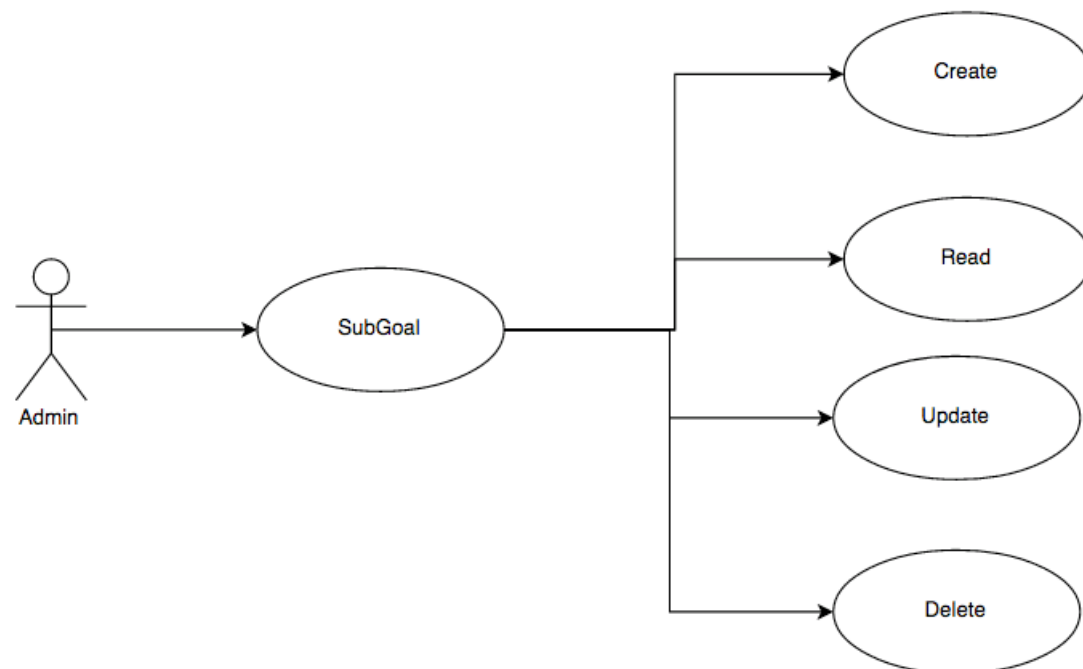
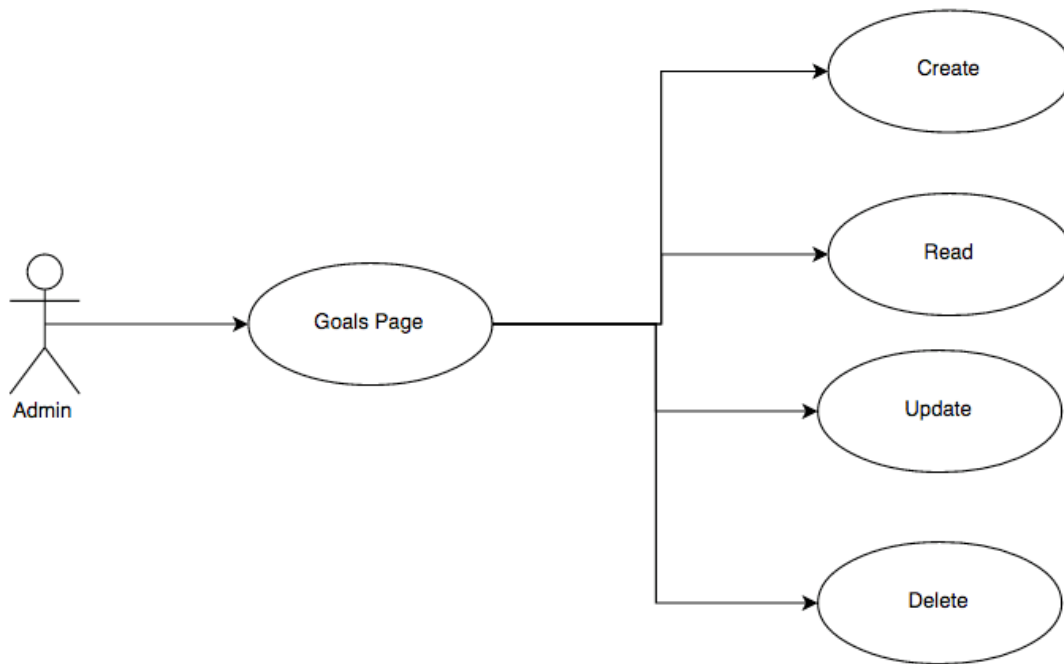
⁸<http://passportjs.org/>

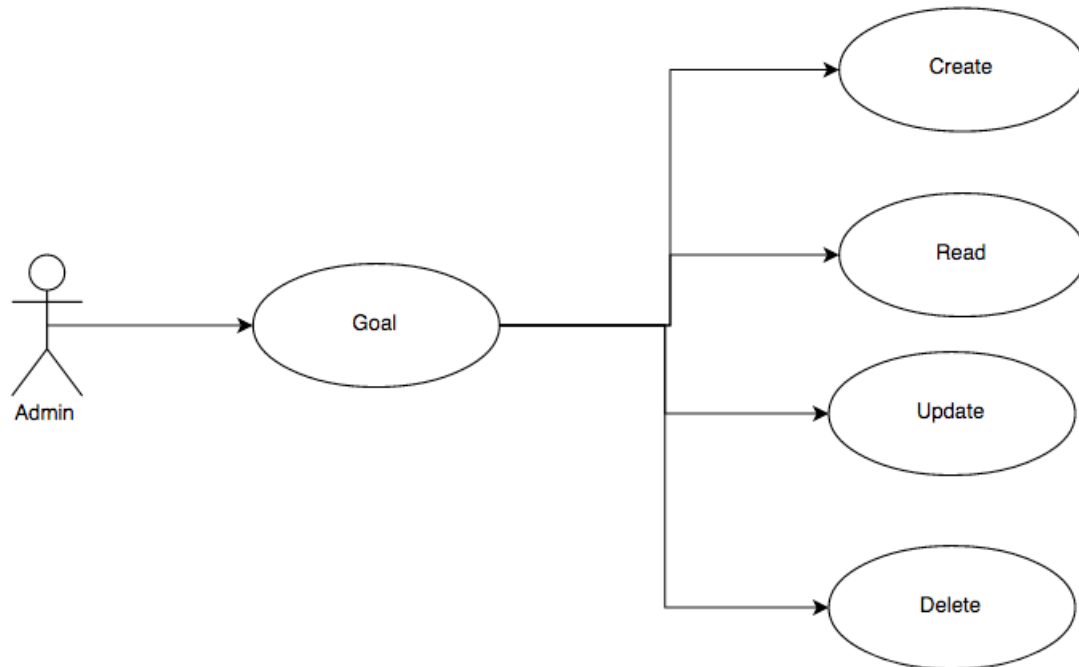
URL	Method	Description
/	GET	public current year goals page
/about	GET	public about page
/blog	GET POST PUT DELETE	public blog
/contact	GET POST	public contact page
/goals	GET	private displays actions to take
/goals/today	GET POST	private submits todays goals progress
/goals/create	GET PUT POST DELETE	private Manage the goals
/goals-page/	GET DELETE	private displays list of goals and actions to take
/goals-page/create	POST	private create a new page
/goals-page/update/:pageId	PUT GET	private updates selected page

URL Mapping Server Side

URL	Method	Description
/pages	GET	public get all pages
/latest-page	Get	public returns the latest page for current year
/pages	POST	private create app pages
/pages/:pageId	GET PUT DELTE	private show, create, update, delete
/goals/:goalId	GET POST PUT DELETE	private show, create, update, delete
/sub-goals/:subGoalId	GET POST PUT DELETE	private show, create, update, delete

The following url's where created based on the use cases.





Rethinking the database design

If you come from a SQL background my design from the previous chapter will make some sense but the structure is still not as clear as I would like it to be. We are using mongodb to store our database objects, and if you did not know we can not represent joins in a NOSQL environment. I had to rethink my data, and I did this by thinking about the output that I would like to have the ideal JSON object and this is what I came up with.

```
1  {
2    "goal": {
3      "name": "Earn $32,000 online",
4      "description": "Create enough value to earn 32K online with ease",
5      "isComplete": false,
6      "dateCompleted": "ongoing",
7      "completeBy": "12/12/15",
8      "goalTotal": 10,
9      "goalTotalCompleted": 2,
10     "subGoal": [{
11       "name": "Create 3 products",
12       "description": "Find 3 products to sell online",
```

```
13     "isComplete": false,
14     "goalTotal": 8,
15     "goalTotalCompleted": 3,
16     "progress" : [{
17         "note" : "found product 1",
18         "value": 1,
19         "date": "12/12/14"
20     }, {
21         "note" : "found product 2",
22         "value": 1,
23         "date": "12/22/14"
24     }, {
25         "note" : "found product 3",
26         "value": 1,
27         "date": "12/22/14"
28     }]
29 }, {
30     "name": "Publish 3 apps",
31     "description": "Develop 3 software apps",
32     "isComplete": false,
33     "goalTotal": 8,
34     "goalTotalCompleted": 3,
35     "progress" : [{
36         "note" : "Created NodeJS app",
37         "value": 1,
38         "date": "12/12/14"
39     }, {
40         "note" : "Created a PHP app",
41         "value": 1,
42         "date": "12/22/14"
43     }, {
44         "note" : "Created a Android app",
45         "value": 1,
46         "date": "12/22/14"
47     }]
48 }],
49 "page": {
50     "year": "2014",
51     "name": "Sweet 14",
52     "overview": "This year is going to be epic",
53     "video": "intro.mp4"
54 }
```

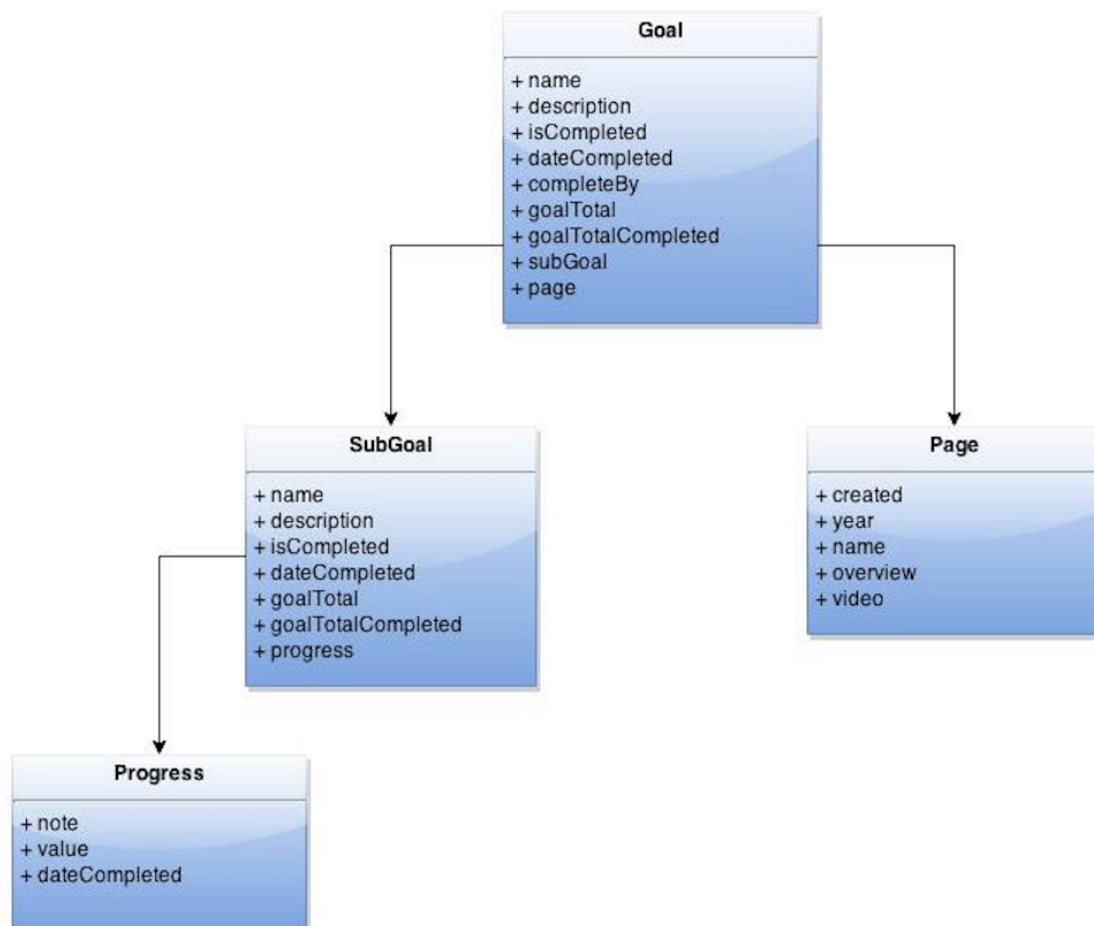
```

55   }
56   }

```

In turn our design would look something like this. Of course this does deviate for the original design but not by much.

DB Design



After setting up our models based on our JSON example, this is really straight forward using [mongoose](http://mongoosejs.com/)⁹, it's just setting the data types and there corresponding default values. From here the CRUD actions are straight forward to implement based on our URL mapping all we have to do is follow the angular way of pulling and fetching data back from our models based on our routes.

That's it for Chapter 4! Congratulations give your self pat on the back you definitely earned it. This was the longest section, now get ready for Chapter 5.

⁹<http://mongoosejs.com/>

Chapter Five: Integrating open source projects into a MEAN application

We are told not to re invent the wheel, and sometimes that is the right advice to take. Why go and create something that already exists? For this project, I wanted to create a blog in markdown and be able to post blogs about my goals and my progress, but like most things by the time I finish my blogging part of the application the year would be over! I decided to KISS and bite the bullet and find an open source project that had those implementations all ready baked in. Since node applications are still relatively new there is no mature projects, but there is a few blogging projects that caught my eye [hexo](http://hexo.io/)¹⁰ and [ghost](https://ghost.org)¹¹. Both of this projects have some similarity but yet are different blogging platforms.

The main differentiation between both of this blogging platforms is that one want's you to physically have .md files on the server and the other works with a data store to store the content of those files and then reads them in dynamically. I decided for my use case [ghost](https://ghost.org)¹² would be the best option, since I am planning on traveling this year and I don't necessarily want to pull up a SSH connection every-time I want to write a blog post. With this in mind they offer a hosted solution for those that are interested in it, you can find out more about it [here](https://ghost.org/pricing/)¹³ if you want to give it a try before getting to committed give it a [try](https://github.com/TryGhost/Ghost/blob/master/CONTRIBUTING.md#working-on-ghost-core)¹⁴.

Tackling Authentication

Ideally what I would like to be able to authenticate with my MEAN credentials and be able to blog that way, but according to the [documentation](https://github.com/TryGhost/Ghost/wiki/%5BWIP%5D-API-Documentation)¹⁵ token [authentication](https://github.com/TryGhost/Ghost/issues/4004)¹⁶ is still under development at this time of writing. So like most projects you have to improvise and work with what you got. I decided to setup ghost at blog.knowrick.com and have a custom angular application pull posts from that subdomain, this would look something like this.

Now the reason why opted out of modifying the core of the application to make it work with my MEAN application was due to the fact that I would like to continue to run updates on the blogging platform and be able to follow the progress of the project and maybe even [contribute](https://github.com/TryGhost/Ghost/blob/master/CONTRIBUTING.md#working-on-ghost-core)¹⁷.

¹⁰<http://hexo.io/>

¹¹<https://ghost.org>

¹²<https://ghost.org>

¹³<https://ghost.org/pricing/>

¹⁴<https://github.com/TryGhost/Ghost>

¹⁵<https://github.com/TryGhost/Ghost/issues/4004>

¹⁶<https://github.com/TryGhost/Ghost/wiki/%5BWIP%5D-API-Documentation>

¹⁷<https://github.com/TryGhost/Ghost/blob/master/CONTRIBUTING.md#working-on-ghost-core>

If you never deployed a ghost blog before the instructions are straight forward for [deployment](#)¹⁸ and installation on [linux](#)¹⁹. For my deployment machine I decided to go with Digital Ocean because of there is to use application for creating virtual machines, if you never used Digital ocean you can get a \$10 dollar credit right [here](#)²⁰ and give it a try.

Authentication with tokens

Since the application is going to be using the JSON that ghosts outputs, I kept running into this issue on the client side. [sourcecode language="html"] â€œNo 'Access-Control-Allow-Origin' header is present on the requested resourceâ€ [/sourcecode] I decided that authenticating and pulling content this way would not work out with out doing some heavy notifications to MEAN and setup [CORS](#)²¹, so I moved over my authenticating code back to the server and used this really handy package [request](#)²² which allows me to make HTTP calls with ease, you can set it up on your project by running.

```
1 npm install request --save
```

Now that you have that package installed, it's straight forward to implement the requests to the server. I did run into some callback hell, but I just separated some of the concerns to more requests. The code to fetch blog posts would be something like this.

```
1  var options = {
2    url: 'http://blog.knowrick.com/ghost/api/v0.1/posts',
3    headers: {
4      'Authorization': config.blog.key
5    }
6  };
7
8  request(options, function(error, response, body) {
9    if (!error && response.statusCode === 200) {
10     var posts = JSON.parse(body);
11     res.status(200).json(posts);
12   }else{
13     req.statusCode = response.statusCode;
14     next();
15   }
16 });
```

¹⁸<http://support.ghost.org/deploying-ghost/>

¹⁹<http://support.ghost.org/installing-ghost-linux/>

²⁰<https://www.digitalocean.com/?refcode=217fa23c64a1>

²¹https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

²²<https://www.npmjs.com/package/request>

If you can see from the above code `config.blog.key` is actually being pulled from `config/env` pretty sweet if you ask me, I did not have to do anything from my side besides calling the package, thank you MEAN.IO team.

Creating a static pages with MEAN

The last remaining part of our application is email, now you must be saying email and node.js? Yep! If you are not a professional in unix socket programming and using the SMTP protocol, I would recommend you to go and check out [mail gun](http://www.mailgun.com/)²³, if you never used MailGun before it's a nice tool that allows developers to quickly get up and going with email. Now the last part to use some abstraction to make this contact form even easier to create check out the handy node package [nodemailer-mailgun-transport](https://github.com/orliesaurus/nodemailer-mailgun-transport)²⁴, and now with a couple of lines of code you will have a contact form ready to go.

That's it for the application we are done!

Not really, we are never done. In Chapter 6 will cover linux init scripts and setting up node.js with a SSL certificate and moving over our code into production.

²³<http://www.mailgun.com/>

²⁴<https://github.com/orliesaurus/nodemailer-mailgun-transport>

Chapter Six: Moving over a MEAN application to production

For this setup I am using the following stack to serve my application to users.

- Ubuntu 14.04
- Node.js
- Apache
- SSL Certificates
- MongoDB

Now you have to run through Chapter 1 again to install prerequisites, after you do this you then have to clone the repository into the server.

```
1 git clone https://github.com/rick4470/knowRick.git
```

That will pull over our source code and just like before back in part one

```
1 cd knowrick
2 npm install
3 bower install
```

Now that we have all of those files on the server we have to be able to setup our development credentials on the server lets edit this file.

```
1 vim config/env/all.js
```

and make the following changes.

```
1  http: {
2    port: process.env.PORT || 3280
3  },
4  https: {
5    port: false,
6    // Paths to key and cert as string
7    ssl: {
8      key: '',
9      cert: ''
10   }
11 },
12 sessionSecret: '<super secrete code here>',
13 sessionCookie: {
14   path: '/',
15   httpOnly: true,
16   secure: false,
17   maxAge: null
18 },
19 blog:{
20   username: '<name>',
21   password: '<password>',
22   clientID: '<role>',
23   key: undefined
24 }
```

Write the changes to config/env/production.js

```
1  mailer: {
2    service: '',
3    auth: {
4      user: '<email>',
5      pass: '<password>'
6    },
7    domain: '<domainName>',
8    api_key: '<key>'
9  }
```

If you can tell we are running our application in a different port 3080 instead of the regular 3000 port, remember we do have our Ghost application on 3000 so we cant reuse that port. Now make sure you save your changes.

How to setup apache and nodejs with an SSL certificate

Here is the tricky part that took a bit of asking around in the mean community, I wanted to force all of the traffic arriving to my application to be sent over HTTPS and the way you would do this is by first purchasing an SSL certificate from a provider I used [godaddy](https://www.godaddy.com/)²⁵, then you have to generate key with openssl, more on that later. Here is what the apache configuration would look like.

```
1 <VirtualHost *:80>
2     ServerAdmin info@knowrick.com
3     ServerName knowrick.com
4     Redirect permanent / https://knowrick.com/
5 </VirtualHost>
6 <VirtualHost *:443>
7     ServerName knowrick.com
8     <Proxy *>
9         Order deny,allow
10        Allow from all
11    </Proxy>
12
13    SSLEngine on
14    SSLProxyEngine On
15    SSLCertificateFile /etc/apache2/ssl/<name>.crt
16    SSLCertificateKeyFile /etc/apache2/ssl/<name>.key
17
18    ProxyRequests Off
19    ProxyPreserveHost On
20    ProxyPass / http://localhost:3280/
21    ProxyPassReverse / http://localhost:3280/
22 </VirtualHost>
```

That's it all you have to do is enable the site and you are ready to go from a server stand point, if you don't recall how to do that go back to Chapter 1.

How to create an init script for a nodejs application

Now we need to test all of those configurations, from the MEAN.IO documentation you can easily do this by running the following command.

```
1 NODE_ENV=production grunt
```

²⁵<https://www.godaddy.com/>

This presents a problem though? How can we make this command as a service? or even better a daemon if we have to restart the server? If you never built a linux daemon before a really good boiler plate code can be found at `/etc/init.d/skeleton` so we will be using this skeleton to build our own service.

```
1 cp /etc/init.d/skeleton /etc/init/knowrick.conf
2 vim /etc/init/knowrick.conf
```

Now just make the changes to that file accordingly to your environment, where the files are located what user is going to be running it. Of course this lead more into system administration which this book is not about.

```
1 description "KnowRick.com node.js server"
2 author      "Rick - http://saltyslopes.com"
3 env PROGRAM_NAME="knowrickNode"
4 env FULL_PATH="<dir>/<dir>/knowRick.com"
5 env FILE_NAME="server.js"
6 env NODE_PATH="/usr/bin/node"
7 env USERNAME="www-data"
8 # used to be: start on startup
9 # until we found some mounts weren't ready yet while booting:
10 start on started mountall
11 stop on shutdown
12
13 # Automatically Respawn:
14 respawn
15 respawn limit 99 5
16
17 script
18     export HOME="/root"
19     export NODE_ENV=production
20     echo $$ > /var/run/$PROGRAM_NAME.pid
21     cd $FULL_PATH
22     exec sudo -u $USERNAME NODE_ENV=production $NODE_PATH $FULL_PATH/$FILE_NAME \
23 >> /var/log/$PROGRAM_NAME.log 2>&1
24 end script
25
26 post-start script
27 end script
28
29 pre-start script
30     # Date format same as (new Date()).toISOString() for consistency
```

```
31     echo "[`date -u +%Y-%m-%dT%T.%3NZ`] (sys) Starting" >> /var/log/$PROGRAM_NAME\
32 E.sys.log
33 end script
34
35 pre-stop script
36     rm /var/run/$PROGRAM_NAME.pid
37     echo "[`date -u +%Y-%m-%dT%T.%3NZ`] (sys) Stopping" >> /var/log/$PROGRAM_NAME\
38 E.sys.log
39 end script
```

Now that we have this handy script running all we have to do is start the application by running `start knowrick` and at any point if you want to stop it just run `stop knowrick`.

Separation of concerns with the source code

When ever you are working on a project that anyone can see your source code, it's important not to publish secrete tokens, passwords and the like on a public repository like Github, not only because bots are constantly scanning for keys, passwords and username on those repos so it's not a good idea. How do we keep using our git repo in development and in production? We have to remove files that we don't want to be tracked in the repository we do thy by running the following commands.

```
1 git rm --cached config/env/all.js
2 git rm --cached config/env/production.js
```

This will only remove the files from the repository but we still have to tell git not to track those files you can do this by running the following command.

```
1 echo config/env/all.js >> .gitignore
2 echo config/env/production.js >> .gitignore
```

Now that we have ignored those files just check in those changes

```
1 git commit -am "removed production files from being tracked"
2 git push
```

That's it don't forget to register those changes to node by doing a restart of the service.

```
1 stop knowrick  
2 start knowrick
```

Now we can quickly push code changes right to the repository with out doing any configurations.
AWESOME!

What's next for this project?

That will do it for this application if we call back to Chapter 1, of this book the entire point of this was for me to be able to track and manage my goals and learn MEAN. Check and check again, now that the application is in production it's time to start putting it to use, I will be tracking my goals all of 2015 and see if we get some results and possibly tackle this project again in 2016 with new and improved changes.

Example of application

Now what are you waiting for? go and give it a try!

[KnowRick.com](https://knowrick.com)²⁶

²⁶<https://knowrick.com>

Chapter Seven: Next Steps

Good Resources

Great job in completing the book, here is a list of resources that I compiled specially for you to keep learning more about the MEAN STACK.

- [Angularjs](#)²⁷
- [Node.JS](#)²⁸
 - [NPM](#)²⁹
- [Express.JS](#)³⁰
- [Bower](#)³¹
- [GruntJS](#)³²
- [Karma](#)³³
- [LESS](#)³⁴
- [Sublime Text](#)³⁵
- [Code With Intent](#)³⁶
- [Javascriptweekly](#)³⁷

Conclusion

Thank you again for downloading this book!

I hope this book was able to help you to get started using the MEAN STACK.

The next step to take would be to share your next project and feedback about this book with me, I'm mostly on twitter [@rick4470](#)³⁸.

²⁷<https://angularjs.org/>

²⁸<http://nodejs.org/>

²⁹<https://www.npmjs.com/>

³⁰<http://expressjs.com/>

³¹<http://bower.io/>

³²<http://gruntjs.com/>

³³<http://karma-runner.github.io/0.12/index.html>

³⁴<http://lesscss.org/>

³⁵<http://www.sublimetext.com/>

³⁶<http://codewithintent.com/>

³⁷<http://javascriptweekly.com/>

³⁸<https://twitter.com/rick4470>

Finally, if you enjoyed this book, then I'd like to ask you for a favor, would you be kind enough to leave a review for this book on Amazon? It'd be greatly appreciated!

Click [here](#)³⁹ to leave a review for this book on Amazon!

Thank you and best of luck with your next project.

Rick H.