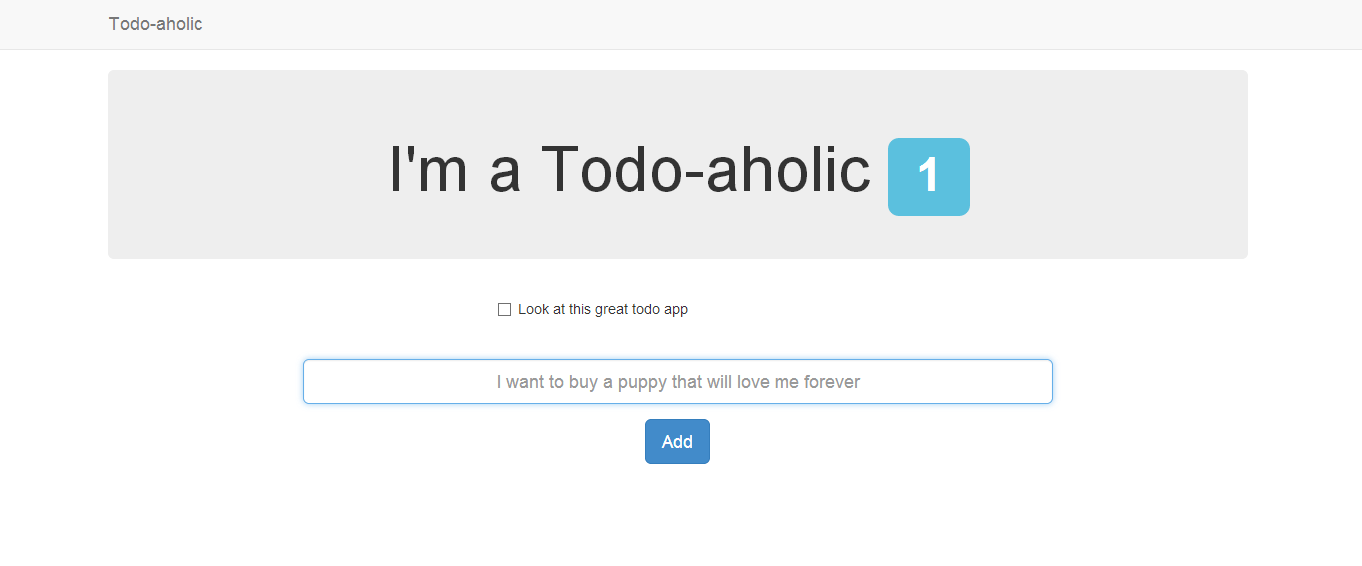
aToday we will be creating a very simple Todo application using the MEAN (Mongo, Express, Angular, Node) stack. We will be creating:

* Single page application to create and finish todos
* Storing todos in a MongoDB using Mongoose
* Using the Express framework
* Creating a RESTful Node API
* Using Angular for the frontend and to access the API

This article has been updated for [**ExpressJS 4.0**](https://scotch.io/bar-talk/expressjs-4-0-new-features-and-upgrading-from-3-0).

While the application is simple and **beginner to intermediate** level in its own right, the concepts here can apply to much more advanced apps. The biggest things we should focus on is using Node as an API and Angular as the frontend. Making them work together can be a bit confusing so this tutorial should help alleviate some confusion. Buckle those seatbelts; this could be a long one.

**What We’ll Be Building**

[](https://cask.scotch.io/2013/11/todoaholic.png)

[**Base Setup**](https://scotch.io/tutorials/creating-a-single-page-todo-app-with-node-and-angular#base-setup)

**File Structure**

We are going to keep the file structure very simple and put most of the code for our Node application into theserver.js file. In larger applications, this should be broken down further to separate duties. [Mean.io](http://mean.io/) is a good boilerplate to see best practices and how to separate file structure. Let’s go ahead and create our simpler file structure and edit the files as we go along.

- public <!-- holds all our files for our frontend angular application -->

----- core.js <!-- all angular code for our app -->

----- index.html <!-- main view -->

- package.json <!-- npm configuration to install dependencies/modules -->

- server.js <!-- Node configuration -->

**Installing Modules**

In Node, the package.json file holds the configuration for our app. Node’s package manager (npm) will use this to install any dependencies or modules that we are going to use. In our case, we will be using [Express](http://expressjs.com/)(popular Node framework) and [Mongoose](http://mongoosejs.com/) (object modeling for MongoDB).

{

"name" : "node-todo",

"version" : "0.0.0",

"description" : "Simple todo application.",

"main" : "server.js",

"author" : "Scotch",

"dependencies" : {

"express" : "~4.7.2",

"mongoose" : "~3.6.2",

"morgan" : "~1.2.2",

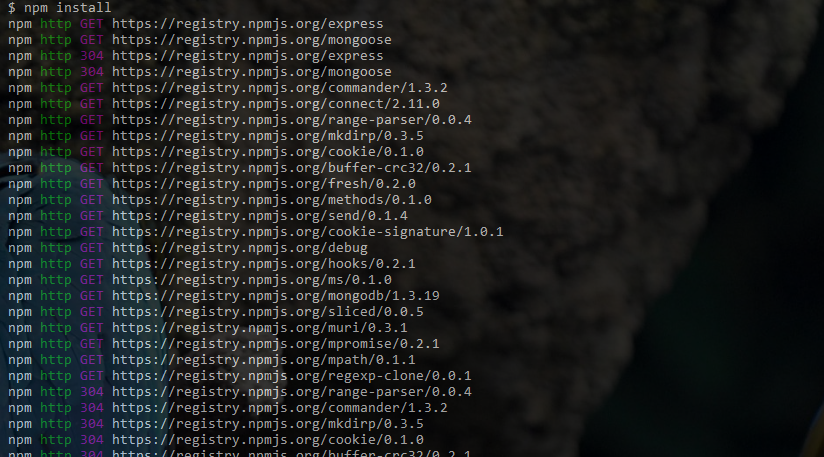
"body-parser": "~1.5.2",

"method-override": "~2.1.2"

}

}

Now if we run npm install, npm will look at this file and install Express and Mongoose.

[](https://cask.scotch.io/2013/11/npm-install.png)

**Node Configuration**

In our package.json file, we told it that our main file would be server.js. This is the main file for our Node app and where we will configure the entire application.

This is the file where we will:

* Configure our application
* Connect to our database
* Create our Mongoose models
* Define routes for our RESTful API
* Define routes for our frontend Angular application
* Set the app to listen on a port so we can view it in our browser

For now, we will just configure the app for Express, our MongoDB database, and listening on a port.

// server.js

// set up ========================

var express = require('express');

var app = express(); // create our app w/ express

var mongoose = require('mongoose'); // mongoose for mongodb

var morgan = require('morgan'); // log requests to the console (express4)

var bodyParser = require('body-parser'); // pull information from HTML POST (express4)

var methodOverride = require('method-override'); // simulate DELETE and PUT (express4)

// configuration =================

mongoose.connect('<mongodb://node:nodeuser>@mongo.onmodulus.net:27017/uwO3mypu'); // connect to mongoDB database on modulus.io

app.use(express.static(\_\_dirname + '/public')); // set the static files location /public/img will be /img for users

app.use(morgan('dev')); // log every request to the console

app.use(bodyParser.urlencoded({'extended':'true'})); // parse application/x-www-form-urlencoded

app.use(bodyParser.json()); // parse application/json

app.use(bodyParser.json({ type: 'application/vnd.api+json' })); // parse application/vnd.api+json as json

app.use(methodOverride());

// listen (start app with node server.js) ======================================

app.listen(8080);

console.log("App listening on port 8080");

Just with that bit of code, we now have an HTTP server courtesy of Node. We have also created an app with Express and now have access to many benefits of it. In our app.configure section, we are using express modules to add more functionality to our application.

**Database Setup**

We will be using a remote database hosted on[Modulus.io](https://modulus.io/). They provide a great service and give you $15 upfront to use as you see fit. This is great for doing testing and creating databases on the fly.

Modulus will provide the database URL you need and you can use mongoose.connect to connect to it. That’s it.

**Start Your App!**

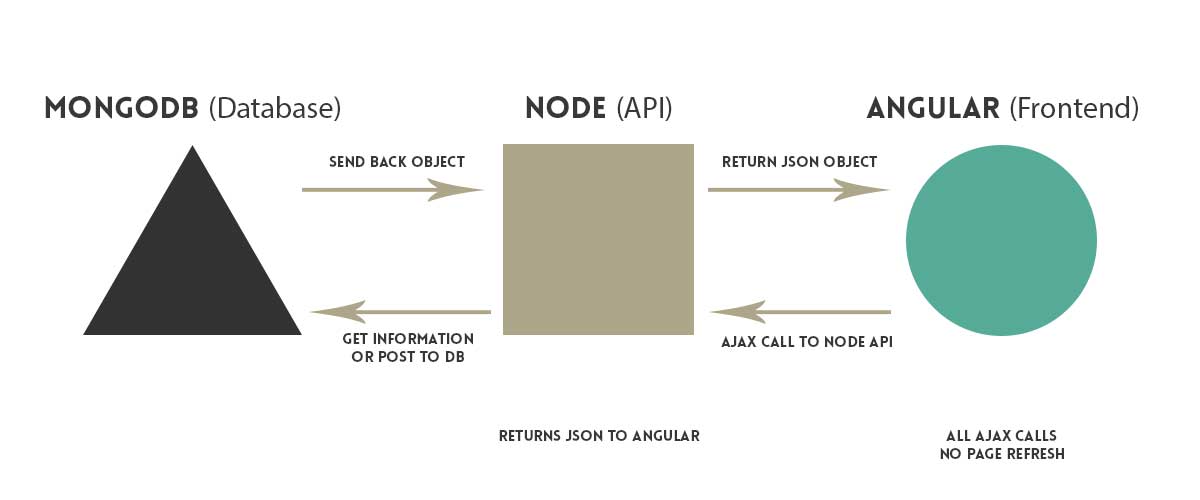
Now that we have our package.json and server.jsstarted up, we can start up our server and see what’s going on. Just go into your console and use the following command:

node server.js Now you have a server listening on port 8080. You can’t see anything in your browser at**http://localhost:8080** yet since we didn’t configure our application to output anything. But it’s a start!

**Automatically restart server when files change:** By default, node will not monitor for file changes after your server has been started. This means you’d have to shut down and start the server every time you made a file change. This can be fixed with **nodemon**. To use: install nodemon globally npm install -g nodemon. Start your server with nodemon server.jsnow. Smooth sailing from there.

[**Application Flow**](https://scotch.io/tutorials/creating-a-single-page-todo-app-with-node-and-angular#application-flow)

Now a brief overview of how all our moving parts will work together. There are a lot of different ideas and technologies involved in this application that it is easy to get mixed up with them all. In our diagram below, we explain a bit of the separation of tasks and how the parts tie in together.

[](https://cask.scotch.io/2013/11/mean.jpg)

Angular is on its own in the frontend. It accesses all the data it needs through the Node API. Node hits the database and returns JSON information to Angular based on the RESTful routing.

This way, you can separate the frontend application from the actual API. If you want to extend the API, you can always build more routes and functions into it without affecting the frontend Angular application. This way you can eventually build different apps on different platforms since you just have to hit the API.

[**Creating Our Node API**](https://scotch.io/tutorials/creating-a-single-page-todo-app-with-node-and-angular#creating-our-node-api)

Before we get to the frontend application, we need to create our RESTful API. This will allow us to have an api that will **get all todos**, **create a todo**, and **complete and delete a todo**. It will return all this information in JSON format.

**Todo Model**

We must define our model for our Todos. We’ll keep this simple. After the **configuration section** and before the**listen** section, we’ll add our model.

// define model =================

var Todo = mongoose.model('Todo', {

text : String

});

That is all we want. Just the text for the todo. MongoDB will automatically generate an \_id for each todo that we create also.

**RESTful API Routes**

Let’s generate our Express routes to handle our API calls.

// server.js

...

// routes ======================================================================

// api ---------------------------------------------------------------------

// get all todos

app.get('/api/todos', function(req, res) {

// use mongoose to get all todos in the database

Todo.find(function(err, todos) {

// if there is an error retrieving, send the error. nothing after res.send(err) will execute

if (err)

res.send(err)

res.json(todos); // return all todos in JSON format

});

});

// create todo and send back all todos after creation

app.post('/api/todos', function(req, res) {

// create a todo, information comes from AJAX request from Angular

Todo.create({

text : req.body.text,

done : false

}, function(err, todo) {

if (err)

res.send(err);

// get and return all the todos after you create another

Todo.find(function(err, todos) {

if (err)

res.send(err)

res.json(todos);

});

});

});

// delete a todo

app.delete('/api/todos/:todo\_id', function(req, res) {

Todo.remove({

\_id : req.params.todo\_id

}, function(err, todo) {

if (err)

res.send(err);

// get and return all the todos after you create another

Todo.find(function(err, todos) {

if (err)

res.send(err)

res.json(todos);

});

});

});

...

Based on these routes, we’ve built a table to explain how a frontend application should request data from the API.

| HTTP Verb | URL | Description |
| --- | --- | --- |
| GET | /api/todos | Get all of the todos |
| POST | /api/todos | Create a single todo |
| DELETE | /api/todos/:todo\_id | Delete a single todo |

Inside of each of our API routes, we use the Mongoose actions to help us interact with our database. We created our Model earlier with var Todo = mongoose.model and now we can use that to **find**, **create**, and **remove**. There are many more things you can do and I would suggest looking at the official [docs](http://mongoosejs.com/docs/guide.html) to learn more.

Our API is done! Rejoice! If you start up your application, you can interact with it at localhost:8080/api/todos to get all the todos. There won’t be anything currently since you haven’t added any.

[**Frontend Application with Angular**](https://scotch.io/tutorials/creating-a-single-page-todo-app-with-node-and-angular#frontend-application-with-angular)

We have **created a Node application**, **configured our database**, **generated our API routes**, and **started a server**. So much already done and still a little bit longer to go!

The work that we’ve done so far can stand on its own as an application. It can be an API we use let applications and users connect with our content.

We want to be the first to use our brand new API that we’ve just created. This is one of my favorite terms that I learned about last month: We will be [dogfooding](http://en.wikipedia.org/wiki/Eating_your_own_dog_food" \t "_blank). We could treat this as we are our very first client to use our new API. We are going to keep this simple so we’ll have just our index.html and core.js to define our frontend.

**Defining Frontend Route**

We have already defined our API routes. Our application’s API is accessible from /api/todos, but what about our frontend? How do we display the index.html file at our home page?

We will add one route to our server.js file for the frontend application. This is all we need to do since Angular will be making a single page application and handle the routing.

After our API routes, and before the app.listen, add this route:

// server.js

...

// application -------------------------------------------------------------

app.get('\*', function(req, res) {

res.sendfile('./public/index.html'); // load the single view file (angular will handle the page changes on the front-end)

});

...

This will load our single index.html file when we hitlocalhost:8080.

**Setting Up Angular**core.js

Let’s go through our Angular setup first. We have to**create a module**, **create a controller**, and **define functions to handle todos**. Then we can **apply to view**.

// public/core.js

var scotchTodo = angular.module('scotchTodo', []);

function mainController($scope, $http) {

$scope.formData = {};

// when landing on the page, get all todos and show them

$http.get('/api/todos')

.success(function(data) {

$scope.todos = data;

console.log(data);

})

.error(function(data) {

console.log('Error: ' + data);

});

// when submitting the add form, send the text to the node API

$scope.createTodo = function() {

$http.post('/api/todos', $scope.formData)

.success(function(data) {

$scope.formData = {}; // clear the form so our user is ready to enter another

$scope.todos = data;

console.log(data);

})

.error(function(data) {

console.log('Error: ' + data);

});

};

// delete a todo after checking it

$scope.deleteTodo = function(id) {

$http.delete('/api/todos/' + id)

.success(function(data) {

$scope.todos = data;

console.log(data);

})

.error(function(data) {

console.log('Error: ' + data);

});

};

}

We create our Angular module (scotchApp) and controller (mainController).

We also create our functions to **get all todos**, **create a todo**, and **delete a todo**. All these will be hitting the API we just created. On page load, we will GET /api/todosand bind the JSON we receive from the API to$scope.todos. We will then loop over these in our view to make our todos.

We’ll follow a similar pattern for **creating and deleting**. Run our action, remake our todos list.

**Frontend View**index.html

Here we will keep it simple. This is the HTML needed to interact with Angular. We will:

* Assign Angular module and controller
* Initialize the page by getting all todos
* Loop over the todos
* Have a form to create todos
* Delete todos when they are checked

<!-- index.html -->

<!doctype html>

<!-- ASSIGN OUR ANGULAR MODULE -->

<html ng-app="scotchTodo">

<head>

<!-- META -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1"><!-- Optimize mobile viewport -->

<title>Node/Angular Todo App</title>

<!-- SCROLLS -->

<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css"><!-- load bootstrap -->

<style>

html { overflow-y:scroll; }

body { padding-top:50px; }

#todo-list { margin-bottom:30px; }

</style>

<!-- SPELLS -->

<script src="//ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script><!-- load jquery -->

<script src="//ajax.googleapis.com/ajax/libs/angularjs/1.0.8/angular.min.js"></script><!-- load angular -->

<script src="core.js"></script>

</head>

<!-- SET THE CONTROLLER AND GET ALL TODOS -->

<body ng-controller="mainController">

<div class="container">

<!-- HEADER AND TODO COUNT -->

<div class="jumbotron text-center">

<h1>I'm a Todo-aholic <span class="label label-info">{{ todos.length }}</span></h1>

</div>

<!-- TODO LIST -->

<div id="todo-list" class="row">

<div class="col-sm-4 col-sm-offset-4">

<!-- LOOP OVER THE TODOS IN $scope.todos -->

<div class="checkbox" ng-repeat="todo in todos">

<label>

<input type="checkbox" ng-click="deleteTodo(todo.\_id)"> {{ todo.text }}

</label>

</div>

</div>

</div>

<!-- FORM TO CREATE TODOS -->

<div id="todo-form" class="row">

<div class="col-sm-8 col-sm-offset-2 text-center">

<form>

<div class="form-group">

<!-- BIND THIS VALUE TO formData.text IN ANGULAR -->

<input type="text" class="form-control input-lg text-center" placeholder="I want to buy a puppy that will love me forever" ng-model="formData.text">

</div>

<!-- createToDo() WILL CREATE NEW TODOS -->

<button type="submit" class="btn btn-primary btn-lg" ng-click="createTodo()">Add</button>

</form>

</div>

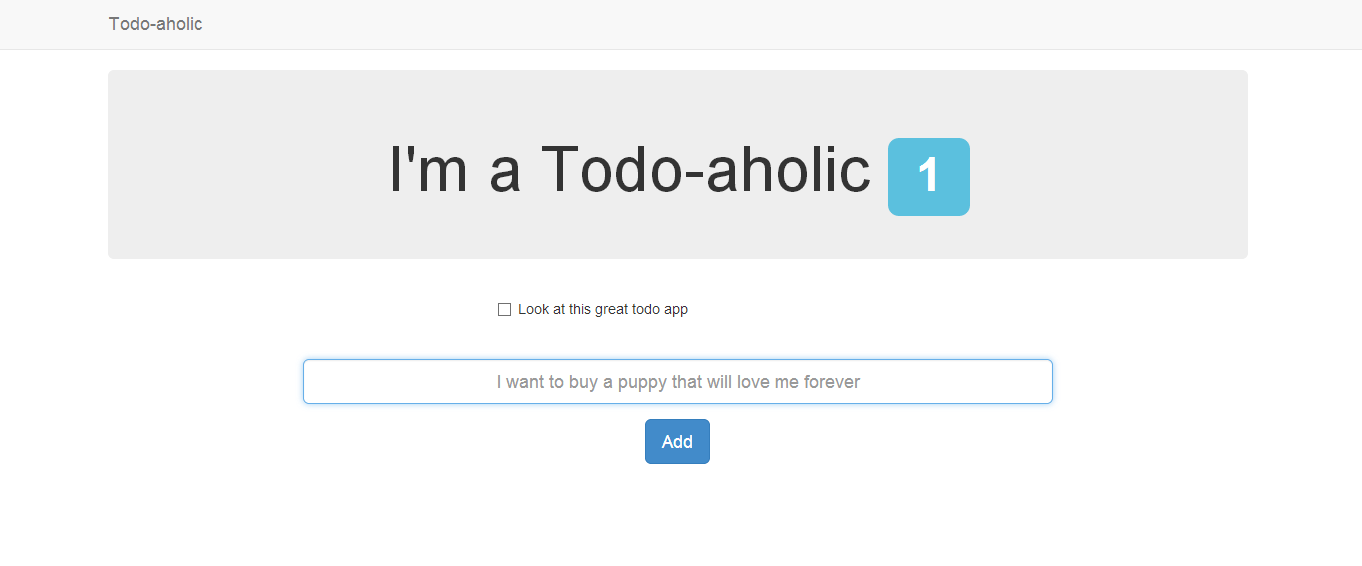
</div>

</div>

</body>

</html>

Take a look at what we have.

[](https://cask.scotch.io/2013/11/todoaholic.png)

[**Conclusion**](https://scotch.io/tutorials/creating-a-single-page-todo-app-with-node-and-angular#conclusion)

Now we have a fully working application that will show, create, and delete todos all via API (that we built!). That was quite a day. We’ve done so much. Just an overview of what we’ve accomplished:

* RESTful Node API using Express
* MongoDB interaction using mongoose
* Angular AJAX $http calls
* Single page application w/ no refreshes
* Dogfooding (sorry, I really like that word)

**Test the Application**

Go ahead and download the code on [Github](https://github.com/scotch-io/node-todo" \t "_blank) and tweak it or test it. To get it all up and running:

1. Make sure you have [Node and npm](http://nodejs.org/) installed
2. Clone the repo: git clone git@github.com:scotch-io/node-todo
3. Install the application: npm install
4. Start the server: node server.js
5. View in your browser at http://localhost:8080

I hope this was insightful on how to have lots of moving parts work together. In the future, we will look at separating our server.js file since that got a little crazy.

**Further Reading If you are interested in more MEAN stack applications, we’ve written up a guide to get you started in building your own MEAN stack foundation.**

[Setting Up a MEAN Stack Single Page Application](https://scotch.io/bar-talk/setting-up-a-mean-stack-single-page-application)

**Edit #1:** Removing ng-init

This article is part of our [**Node and Angular To-Do App**](https://scotch.io/series/node-and-angular-to-do-app) series.

1. Creating a Single Page To-do App with Node and Angular
2. [**Node Application Organization and Structure**](https://scotch.io/tutorials/javascript/node-and-angular-to-do-app-application-organization-and-structure)
3. [**Angular Modules: Controllers and Services**](https://scotch.io/tutorials/javascript/node-and-angular-to-do-app-controllers-and-services)