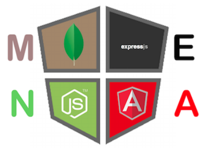
MEAN Stack Tutorial MongoDB ExpressJS AngularJS NodeJS (Part III)

Oct 3rd, 2014 6:59 am | Comments

Table of Contents

This is the last part of three-series tutorial. We are going to build a full-stack Todo App using the MEAN (MongoDB, ExpressJS, AngularJS and NodeJS).

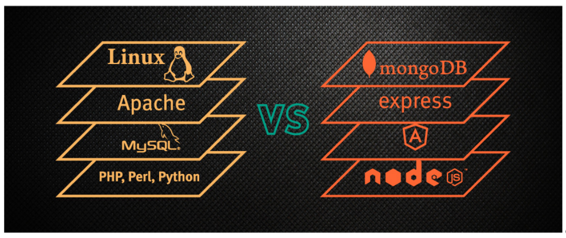
Part III: MEAN Stack



Brief Background

TL; DR: NodeJS has been built from bottom up a non-blocking I/O paradigm, which gives you more efficiency per CPU core than using threads in other languages like Java. Get started here.

LAMP (Linux-Apache-MySQL-PHP) has dominated web application stack for many years now. Well-known platforms such as Wikipedia, Wordpress, and even Facebook uses it or started with it. Enterprise, usually, used go down the Java path: Hibernate, Spring, Struts, JBoss. More agile frameworks also have been widely used such as Ruby on Rails and for Python Django and Pylon.



(Image from backand.com)

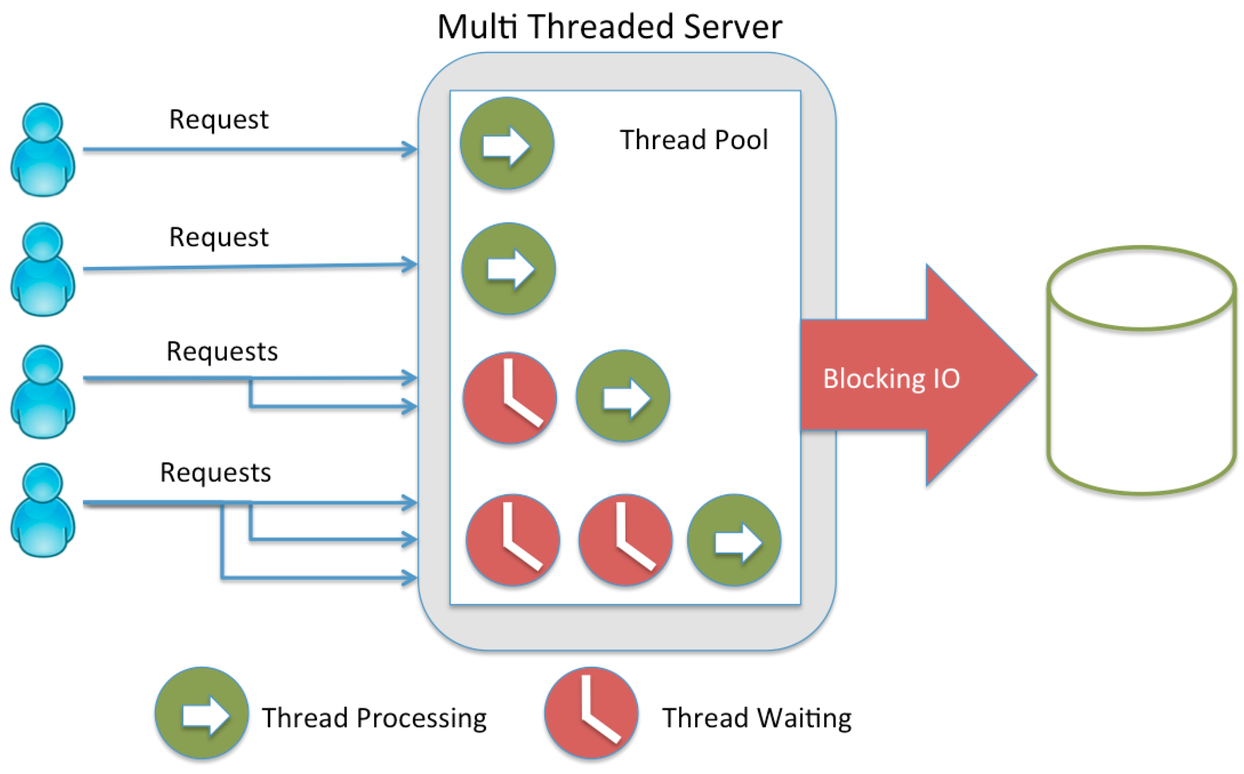
Why MEAN stack then?

Ubiquitous

Well, it turns out, that JavaScript it is everywhere: smartphones, computers, in the browser, in the server, robotics, Arduino, RaspberryPi and growing. Thus, it does not matter what technology of stack you use to build web applications you need to be familiar with Javascript. In that case then, it is a time saver to use wherever it fits, especially for building web applications. MEAN stack is embracing that, using Javascript to create end-to-end web applications.

Non-blocking architecture

JavaScript is a dynamic, object-oriented, and functional scripting language. One of the features that make it win over Java Applets in the browser scripting war decades ago, it was its lightness and non-blocking event loop. Bocking means that when one line of code is executing the rest of it is locked waiting to finish. On the other hand, non-blocking gives to each line of code a shot and then through callbacks it can come back when an event happens. Programming languages that are blocking (Java, Ruby, Python, PHP, …) overcomes concurrency using multiple threads of execution while JavaScript handles it using non-blocking event loop in a single thread.



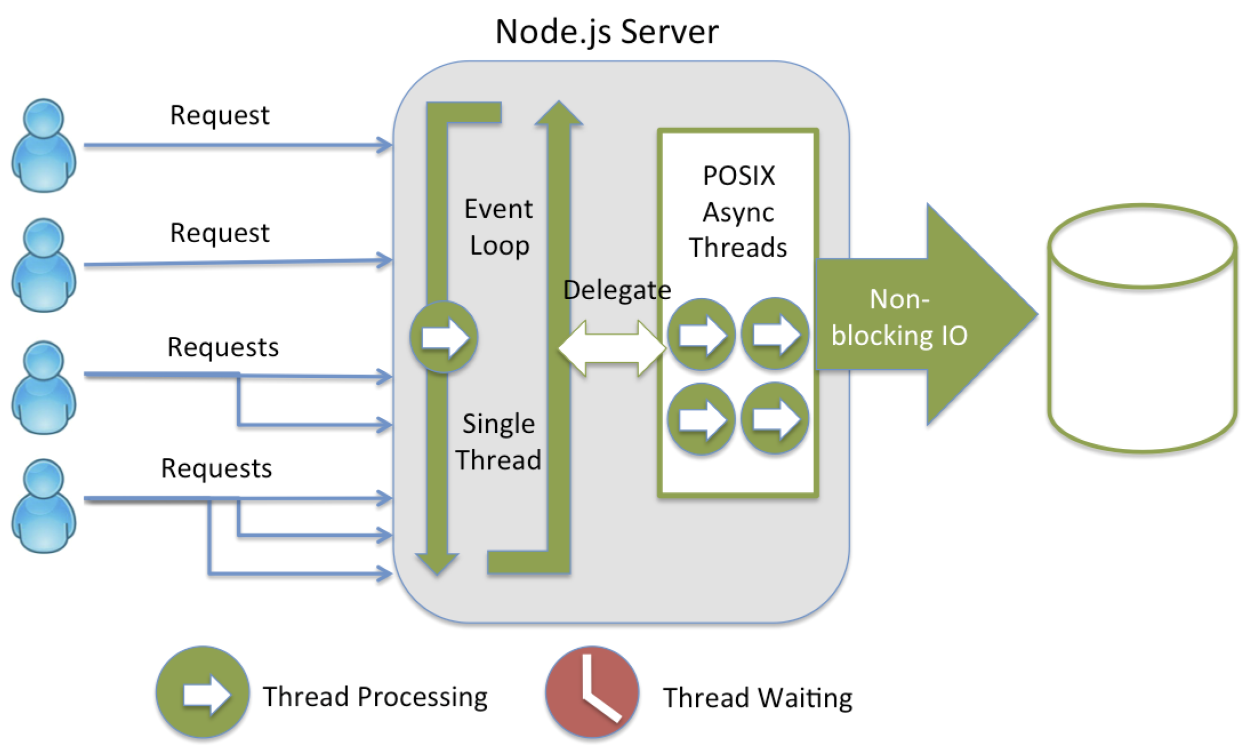


Image from strongloop.com

Some companies like [Paypal]() moved from Java backend to NodeJS and reported a increased performance, lower average response times, and development speed gains. Similarly happens to [Groupon]() that came from Java/Rails monoliths.

Agile and vibrant community

The community behind Javascript is quite vibrant and has permeated in almost all the fields of technology: data visualization, client-side frameworks, server-side frameworks, databases, robotics, building tools and many more.

Setup

MEN for MongoDB, ExpressJS and NodeJS

In the [previous post](), we have gone through the process of building a RESTful API and we are going to be building on top of that. [Repository here]().

Getting the back-end code build on Part II

```
git clone https://github.com/amejiarosario/todoAPIjs.git
```

A for AngularJS

Similarly, we have build a very lean todoApp in the [first part]() of this tutorial. You can [download the file]() to follow along and see it in action [here](). You might notice the angularJS app is very simple and even it is entirely in one file for simplicity sake. In further tutorials, we are going to make it more modular, split in files, add tests and stylesheets.

Let's go first to the ExpressJS app (todoAPIjs) and review the default routing system:

1. `app.js` loads the all the routes.
2. The root path (`/`) is mounted on the `routes/index.js`
3. `routes/index.js` sets the variable title and renders `index.ejs`.

Tracing ExpressJS index route

```
// app.js
var routes = require('./routes/index');
app.use('/', routes);

// ./routes/index.js
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
});

// ./views/index.ejs
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
```

The best place to load our `./views/index.ejs`. So let's copy the body content from ngTodo.html content in there and change in `./routes/index.js` title to "ngTodo App". Don't forget to add ng-app on the top. `<html ng-app="app">`.

[diff]()

Wiring it together

AngularJS CRUD

AngularJS Read with $http

As you might notice, in the factory, we have a fixed array. We need to change it to communicate with the API that we just build.

`$http` is Anguar core sevice that allow to make `XMLHttpRequest` or `jsonp` request. You can either pass an object with http verb and url or call call $http.verb (`$http.get`, `$http.post`).

`$http` returns a promise which has a `success` and `error` function.

AngularJS $HTTP Usage Example

```
$http({method: 'GET', url: '/todos'}).
  success(function(data, status, headers, config) {
    // this callback will be called asynchronously
    // when the response is available.
    console.log('todos: ', data );
  }).
  error(function(data, status, headers, config) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
    console.log('Oops and error', data);
  });
```

Let's try it out in our app. Go to `views/index.ejs` and do this changes:

Using $http to retrieve data from database

```
    // Service
    .factory('Todos', ['$http', function($http){
      return $http.get('/todos');
    }])

    // Controller
    .controller('TodoController', ['$scope', 'Todos', function ($scope, Todos) {
      Todos.success(function(data){
        $scope.todos = data;
      }).error(function(data, status){
        console.log(data, status);
        $scope.todos = [];
      });
    }])
```

If you have data in MongoDB you see them listed in the main page. If you not you can follow the steps in [here]() to get some in.

[diff]()

AngularJS Read with $resource

If you click in one of the Todo elements and get redirected to the detail page, you will not see anything yet. We need to update the `TodoDetailCtrl` first. Even though, we already have the GET verb working, we have a slightly different URL requirement for `/todos/:id`. There's an Angular service that has a higher level of abstraction to deal with RESTful requests: `$resource`.

Initialize as: `$resource(url, [paramDefaults], [actions], options);`

It comes with the following actions already defined; it is missing one though… Can you tell?

$resource actions

```
{ 'get':    {method:'GET'},  // get individual record
  'save':   {method:'POST'}, // create record
  'query':  {method:'GET', isArray:true}, // get list all records
  'remove': {method:'DELETE'}, // remove record
  'delete': {method:'DELETE'} }; // same, remove record
```

The instances are used in the following way (examples will come later):

- GET: `Resource.get([parameters], [success], [error])`
- Non-GET: `Resource.action([parameters], postData, [success], [error])`
- Non-GET: `resourceInstance.$action([parameters], [success], [error])`

`$resource` is not part of the Angular core, so it requires to `ngResource` and the dependency. We can get it from the CDN:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular-resource.min.js"></script>
```

This is what need to set it up:

$resource.query()

```
// add ngResource dependency
angular.module('app', ['ngRoute', 'ngResource'])

      .factory('Todos', ['$resource', function($resource){
        return $resource('/todos/:id', null, {
          'update': { method:'PUT' }
        });
      }])

      .controller('TodoController', ['$scope', 'Todos', function ($scope, Todos) {
        $scope.todos = Todos.query();
      }])
```

Notice that `$resource` does not return a promise like `$http` but an empty reference instead. Angular will render an empty `$scope.todos`, however, when `Todos.query()` comes with the data from the server it will re-render the UI automatically.

diff

AngularJS Create

We will need to create a new text box, a button to send a `POST` request to server and add it to the `$scope`.

New textbox for adding Todos

```
New task <input type="text" ng-model="newTodo"><button ng-click="save()">Create</button>
```

Notice that we are using a new directive `ng-click`, this one executes a function when it clicked. Angular makes sure that the behaviour is consistent across different browsers.

Save function $resource.$save(…)

```
.controller('TodoController', ['$scope', 'Todos', function ($scope, Todos) {
  $scope.todos = Todos.query();

  $scope.save = function(){
    if(!$scope.newTodo || $scope.newTodo.length < 1) return;
    var todo = new Todos({ name: $scope.newTodo, completed: false });

    todo.$save(function(){
      $scope.todos.push(todo);
      $scope.newTodo = ''; // clear textbox
    });
  }
}])
```

diff

Show Todo details

Every time you click a todo link, it is showing an empty fields. Let’s fix that. First we need set the real `_id` to the links instead of `$index`.

Change the ID link in the “/todos.html” template

```
<li ng-repeat="todo in todos | filter: search">
  <input type="checkbox" ng-model="todo.completed">
  <a href="#/"></a>
</li>
```

Update TodoDetailCtrl with $resource.get

```
.controller('TodoDetailCtrl', ['$scope', '$routeParams', 'Todos', function ($scope, $routeParams, Todos) {
  $scope.todo = Todos.get({id: $routeParams.id });
}])
```

Now you should be able to see the details :)

diff

AngularJS Update (in-line editing)

This is going to be a very cool feature. Meet these new directives:

- **ng-show**: is directive that shows the element in which it is declared if attribute is true or hide it when the attribute become false.
- **ng-change**: directive for input elements that evaluates the expression after any change.

Template todos.html

```
<!-- Template -->
<script type="text/ng-template" id="/todos.html">
  Search: <input type="text" ng-model="search.name">
  <ul>
    <li ng-repeat="todo in todos | filter: search">
      <input type="checkbox" ng-model="todo.completed" ng-change="update($index)">
      <a ng-show="!editing[$index]" href="#/"></a>
      <button ng-show="!editing[$index]" ng-click="edit($index)">edit</button>

      <input ng-show="editing[$index]" type="text" ng-model="todo.name">
      <button ng-show="editing[$index]" ng-click="update($index)">Update</button>
      <button ng-show="editing[$index]" ng-click="cancel($index)">Cancel</button>
    </li>
  </ul>
  New task <input type="text" ng-model="newTodo"><button ng-click="save()">Create</button>
</script>
```

We added a new variable `$scope.editing` which shows or hides the form to edit the values. Furthermore, notice ng-click functions: edit, update and cancel. Let’s see what they do.

Todo Controller

```
.controller('TodoController', ['$scope', 'Todos', function ($scope, Todos) {
  $scope.editing = [];
  $scope.todos = Todos.query();

  $scope.save = function(){
    if(!$scope.newTodo || $scope.newTodo.length < 1) return;
    var todo = new Todos({ name: $scope.newTodo, completed: false });

    todo.$save(function(){
      $scope.todos.push(todo);
      $scope.newTodo = ''; // clear textbox
    });
  }

  $scope.update = function(index){
    var todo = $scope.todos[index];
    Todos.update({id: todo._id}, todo);
    $scope.editing[index] = false;
  }

  $scope.edit = function(index){
    $scope.editing[index] = angular.copy($scope.todos[index]);
  }

  $scope.cancel = function(index){
    $scope.todos[index] = angular.copy($scope.editing[index]);
    $scope.editing[index] = false;
  }
}])
```

While were are editing notice that we copy the original todo task into the editing variable. This server for two purposes: (1) it evaluates to `true` and show the forms with `ng-show` and (2) it holds a copy of the original value in case we press cancel.

Now, going to the Todo Details. We would like that to be updated as well and add notes.

Todo Details

```
<script type="text/ng-template" id="/todoDetails.html">
  <h1></h1>
  completed: <input type="checkbox" ng-model="todo.completed"><br>
  note: <textarea ng-model="todo.note"></textarea><br><br>

  <button ng-click="update()">Update</button>
  <a href="/">Cancel</a>
</script>
```

Similarly, we added an update method. However, this time we do not need to pass any index, since it is just one todo at a time. After it has been saved, it goes back to root path `/`.

Todo Details

```
.controller('TodoDetailCtrl', ['$scope', '$routeParams', 'Todos', '$location', function ($scope, $routeParams, Todos, $location) {
  $scope.todo = Todos.get({id: $routeParams.id });

  $scope.update = function(){
    Todos.update({id: $scope.todo._id}, $scope.todo, function(){
      $location.url('/');
    });
  }
}])
```

- `$location.url([url])` is a getter/setter method that allows us to change url, thus routing/view.

diff

AngularJS Delete

These are the changes added to implement the remove functionality. Pretty straight forward. Notice when we remove elements from the todos array `$scope.todos.splice(index, 1)` they also disappear from the DOM. Very cool, huh?

Delete functionality (diff)

```
diff --git a/views/index.ejs b/views/index.ejs
index 9c3ef46..afb37e1 100644
--- a/views/index.ejs
+++ b/views/index.ejs
@@ -22,6 +22,7 @@
            <input type="checkbox" ng-model="todo.completed" ng-change="update($index)">
            <a ng-show="!editing[$index]" href="#/"></a>
            <button ng-show="!editing[$index]" ng-click="edit($index)">edit</button>
+           <button ng-show="!editing[$index]" ng-click="remove($index)">remove</button>

            <input ng-show="editing[$index]" type="text" ng-model="todo.name">
            <button ng-show="editing[$index]" ng-click="update($index)">update</button>
@@ -37,6 +38,7 @@
        note: <textarea ng-model="todo.note"></textarea><br><br>

        <button ng-click="update()">update</button>
+       <button ng-click="remove()">remove</button>
```

```
      <a href="/">Cancel</a>
    </script>

@@ -85,6 +87,13 @@
            $scope.todos[index] = angular.copy($scope.editing[index]);
            $scope.editing[index] = false;
          }
+
+         $scope.remove = function(index){
+           var todo = $scope.todos[index];
+           Todos.remove({id: todo._id}, function(){
+             $scope.todos.splice(index, 1);
+           });
+         }
        }])

        .controller('TodoDetailCtrl', ['$scope', '$routeParams', 'Todos', '$location', function ($scope, $routeParams, Todos, $location) {
@@ -95,6 +104,12 @@
              $location.url('/');
            });
          }
+
+         $scope.remove = function(){
+           Todos.remove({id: $scope.todo._id}, function(){
+             $location.url('/');
+           });
+         }
        }])

        //---------------
```

diff

Congratulations! You are now a MEAN developer!

What’s next?

Learn how to use GruntJS to automate repetitive tasks in your MEAN Stack workflow.

GruntJS Tutorial

Also, you can learn more about full-stack framework solutions.

Full-Stack Javascript Web Frameworks

What we did in these three series tutorial could have been done with just few keystrokes in the comamnd line ;). However, it’s good to know what’s going on. But at this point you do. So, I will introduce you to some frameworks that can save you a lot of time.

Using MEAN.io

MeanIO uses a customized CLI tool: ‘mean’. Its approach for modularity is leaned towards self-contained packages that have code for both client and server files. At moment of writing this, it has nine packages ranging from MEAN-Admin, Translation, file uploads, image crop and more.

Using MEAN.js

MeanJS it is a fork from the creator of MEAN.IO, it uses Yeoman generators to generate Angular’s CRUD modules, routes, controllers, views, services, and more. Also has generators for Express: models, controllers, routes and tests. It has excellent documentation.

Others Frameworks to look at

- Meteor - Meteor is an open-source platform for building top-quality web apps in a fraction of the time, whether you’re an expert developer or just getting started.
- Sails - The web framework of your dreams. for your next web application.
- Yahoo! Mojito - A JavaScript MVC framework for mobile applications, one of the Yahoo! Cocktails.
- Tower.js - Small components for building apps, manipulating data, and automating a distributed infrastructure.

Posted by Adrian Mejia Oct 3rd, 2014 6:59 am agile frameworks, angularjs, apache, javascript, lamp, linux, mean stack, mongodb, mysql, nodejs, tutorials, web development, web frameworks

Tweet 0 g+1 0 Like Share Sign Up to see what your friends like.

« Creating RESTful APIs with NodeJS and MongoDB Tutorial (Part II) Grunt JS tutorial from Beginner to Ninja »

Comments

0 Comments Adrian Mejia's Blog Login

Sort by Best Share Favorite

Start the discussion…

ALSO ON **ADRIAN MEJIA'S BLOG** WHAT'S THIS?

Algorithms for Dummies (Part 1): Big-O Notation and Sorting
4 comments • 9 months ago
AvatarOdumba500 — is there anything on reverse order of a sentence and how to write step by step instructions? without code?

Cheap Airplay Receiver With Raspberry Pi - Adrian Mejia's [code]Blog
6 comments • 9 months ago
AvatarScott — A bit of research and I got it working by doing the following. (Now my AirPi starts automatically on boot)STEP

Microsoft Zune Failure Analysis
1 comment • 3 months ago
Avatartomcarrdpi — Got this 30GB Zune for a steal, especially for the storage size - http://www.amazon.com/dp/B00oF... it

Creating a RESTful API Tutorial With NodeJS and MongoDB
2 comments • a month ago
Avataradriansky — Thanks, I fixed it and also added the repository for full file examples and diff: https://github.com

Subscribe Add Disqus to your site Privacy

Recent Posts

- Grunt JS Tutorial From Beginner to Ninja
- MEAN Stack Tutorial MongoDB ExpressJS AngularJS NodeJS (Part III)
- Creating RESTful APIs With NodeJS and MongoDB Tutorial (Part II)
- AngularJS Tutorial for Beginners With NodeJS ExpressJS and MongoDB (Part I)
- How Company X Make Money?

RSS

adrianmejia

@amejiarosario

@amejiarosario

Google+

Subscribe

Enter your email to receive updates

Subscribe

GitHub Repos

- todoAPIjs

  NodeJS, ExpressJS and MongoDB RESTful API Tutorial

- amejiarosario.github.io
- impulsideas
- landing-impulsideas
- soshop

  social shop

@amejiarosario on GitHub