MEAN.JS - Quick Guide

MEAN.JS - Overview

What is MEAN.js?

The term **MEAN.js** is a full stack JavaScript open-source solution, used for building dynamic websites and web applications. MEAN is an acronym that stands for **M**ongoDB, **E**xpress, **N**ode.js and **A**ngularJS, which are the key components of the MEAN stack.

It was basically developed to solve the common issues with connecting those frameworks (Mongo, Express Nodejs, AngularJS), build a robust framework to support daily development needs, and help developers use better practices while working with popular JavaScript components.

Stack means using the database and web server in the back end, in the middle you will have logic and control for the application and interaction of user at the front end.

- **MongoDB** − Database System
- **Express** − Back-end Web Framework
- **Node.js** − Web Server Platform
- **AngularJS** − Front-end Framework

History

MEAN name was coined by *Valeri Karpov*, a MongoDB developer.

Why to use MEAN.js?

- It is an open source framework which is free to use.
- It can be used as standalone solution in a whole application.
- It provides lower development cost and increases the developer flexibility and efficiency.
- It supports MVC pattern and uses the JSON for transferring data.
- It provides additional frameworks, libraries and reusable modules to increase the development speed.

Before we begin with further concepts, we will see the basic building blocks of *MEAN.JS* application.

Introduction to MongoDB

In *MEAN* acronym, **M** stands for MongoDB, which is an open source NoSQL database that saves the data in JSON format. It uses the document oriented data model to store the data instead of using table and rows as we use in the relational databases. It stores data in binary JSON (JavaScript Serialized Object Notation) format to pass the data easily between client and server. MongoDB works on concept of collection and document. For more information, refer to this link MongoDB .

Introduction to Express

In *MEAN* acronym, **E** stands for *Express*, which is a flexible Node.js web application framework used to make development process easier. It is easy to configure and customize, that allows building secure, modular and fast applications. It specifies the routes of an application depending on the HTTP methods and URLs. You can connect to databases such as MongoDB, MySQL, Redis easily. For more information, refer to this link Express .

Introduction to AngularJS

In *MEAN* acronym, **A** stands for *AngularJS*, which is a web frontend JavaScript framework. It allows creating dynamic, single page applications in a clean Model View Controller (MVC) way. AngularJS automatically handles JavaScript code suitable for each browser. For more information, refer to this link AngularJS .

Introduction to Node.js

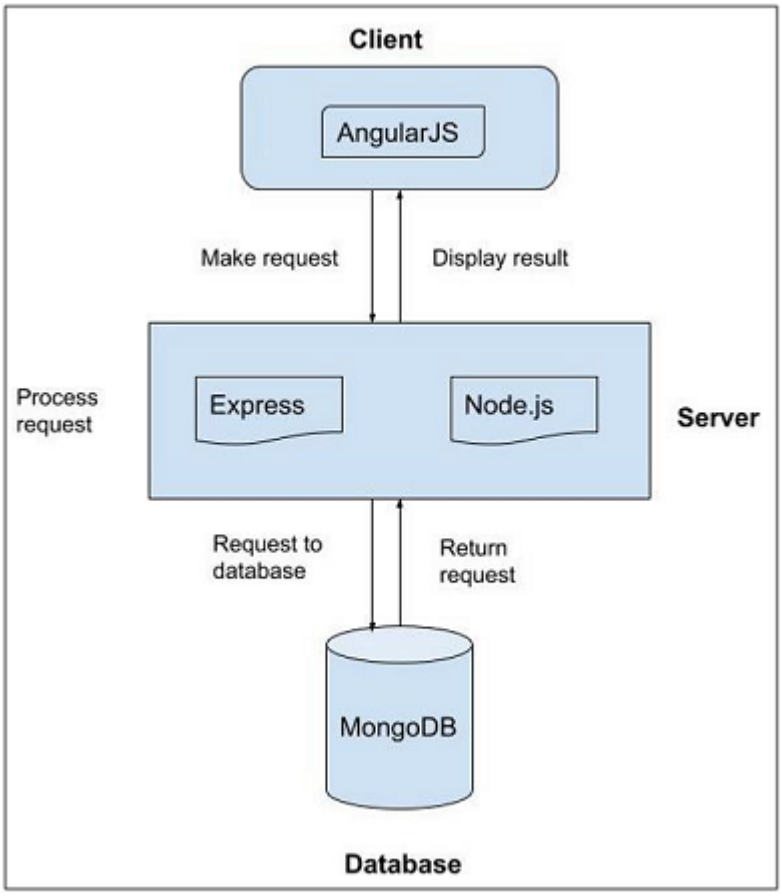
In *MEAN* acronym, **N** stands for *Node.js*, which is a server side platform used for development of web applications like video streaming sites, single-page applications, and other web applications. It provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent. It is built on Google Chrome's V8 JavaScript Engine, so it is very fast in code execution. For more information, refer to this link Node.js .

MEAN.JS - Architecture

MEAN is an open source JavaScript framework, used for building dynamic websites and web applications. It includes following four building blocks to build an application.

- **MongoDB** − It is a document database, that stores data in flexible, JSON-like documents.
- **Express** − It is web application framework for Nodejs.
- **Node.js** − It is Web Server Platform. It provides rich library of various JavaScript modules which simplifies the development of web applications.
- **AngularJS** − It is a web frontend JavaScript framework. It allows creating dynamic, single page applications in a clean Model View Controller (MVC) way.

For more information on these, you can refer the overview chapter. The below diagram depicts architecture of MEAN stack application.



As shown in the above image, we have AngularJS as client side language which processes the request of a client.

- Whenever a user makes a request, it is first processed by AngularJS.
- Next, request enters second stage, where we have Node.js as server side language and *ExpressJS* as backend web framework.
- *Node.js* handles the client/server requests and *ExpressJS* makes request to the database.
- In the last stage, *MongoDB* (database) retrieves the data and sends the response to ExpressJS.
- ExpressJS returns the response to Nodejs and in turn to AngularJS and then displays the response to user.

MEAN.JS - MEAN Project Setup

This chapter includes creating and setting up a MEAN application. We are using NodeJS and ExpressJS together to create the project.

Prerequisites

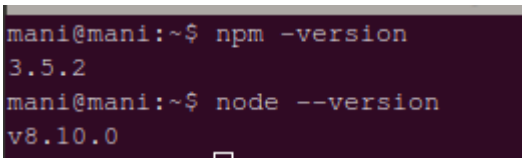
Before we begin with creating a MEAN application, we need to install required prerequisites.

You can install latest version of Node.js by visiting the Node.js website at Node.js (This is for Windows users). When you download Node.js, npm will get installed automatically on your system. Linux users can install the Node and npm by using this link .

Check the version of Node and npm by using the below commands −

```
$ node --version
$ npm --version
```

The commands will display the versions as shown in the below image −



Creating Express Project

Create a project directory by using mkdir command as shown below −

```
$ mkdir mean-demo //this is name of repository
```

The above directory is the root of node application. Now, to create package.json file, run the below command −

```
$ cd webapp-demo
$ npm init
```

The init command will walk you through creating a package.json file −

This utility will walk you through creating a package.json file. It only covers the most common items, and tries to guess sensible defaults.

```
See `npm help json` for definitive documentation on these fields and exactly what they
Use `npm install --save` afterwards to install a package and save it as a dependency i

Press ^C at any time to quit.
name: (mean-demo) mean_tutorial
version: (1.0.0)
description: this is basic tutorial example for MEAN stack
entry point: (index.js) server.js
test command: test
git repository:
```

```
keywords: MEAN,Mongo,Express,Angular,Nodejs
author: Manisha
license: (ISC)
About to write to /home/mani/work/rnd/mean-demo/package.json:

{
   "name": "mean_tutorial",
   "version": "1.0.0",
   "description": "this is basic tutorial example for MEAN stack",
   "main": "server.js",

   "scripts": {
      "test": "test"
   },
   "keywords": [
      "MEAN",
      "Mongo",
      "Express",
      "Angular",
      "Nodejs"
   ],
   "author": "Manisha",
   "license": "ISC"
}
```

```
Is this ok? (yes) yes
```

Click yes and a folder structure as below will be generated −

```
-mean-demo
   -package.json
```

The *package.json* file will have the following info −

```
{
   "name": "mean_tutorial",
   "version": "1.0.0",
   "description": "this is basic tutorial example for MEAN stack",
   "main": "server.js",
   "scripts": {
      "test": "test"
   },
   "keywords": [
      "MEAN",
```

```
      "Mongo",
      "Express",
      "Angular",
      "Nodejs"
   ],
   "author": "Manisha",
   "license": "ISC"
}
```

Now to configure the Express project into current folder and install configuration options for the framework, use the below command −

```
npm install express --save
```

Go to your project directory and open package.json file, you will see the below information −

```
{
   "name": "mean_tutorial",
   "version": "1.0.0",
   "description": "this is basic tutorial example for MEAN stack",
   "main": "server.js",
   "scripts": {
      "test": "test"
   },
   "keywords": [
      "MEAN",
      "Mongo",
      "Express",
      "Angular",
      "Nodejs"
   ],
   "author": "Manisha",
   "license": "ISC",
   "dependencies": {
      "express": "^4.17.1"
   }
}
```

Here you can see express dependency is added to the file. Now, the project structure is as below −

```
-mean-demo
   --node_modules created by npm install
   --package.json tells npm which packages we need
   --server.js set up our node application
```

Running Application

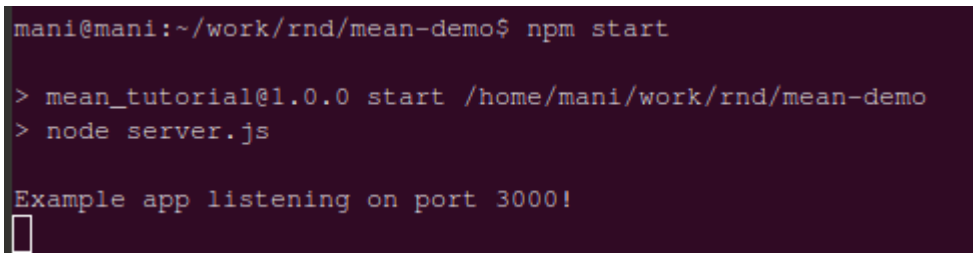
Navigate to your newly created project directory and create a server.js file with below contents.

```
// modules =================================================
const express = require('express');
const app = express();
// set our port
const port = 3000;
app.get('/', (req, res) ⇒ res.send('Welcome to Tutorialspoint!'));

// startup our app at http://localhost:3000
app.listen(port, () ⇒ console.log(`Example app listening on port ${port}!`));
```

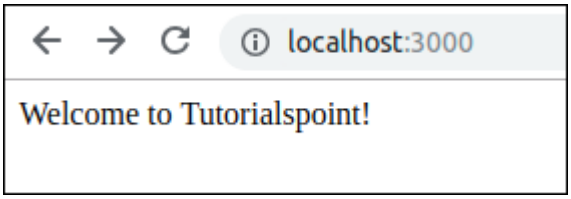
Next, run the application with the below command −

```
$ npm start
```

You will get a confirmation as shown in the image below −



It informs that Express application is running. Open any browser and access the application using **http://localhost:3000**. You will see Welcome to Tutorialspoint! text as shown below −



MEAN.JS - Building Static Route Node Express

This chapter demonstrates building route for an application with **Node** and **Express**.

In the previous chapter, we created a node-express application. Navigate to project directory called *mean-demo*. Go to the directory by using below command −

```
$ cd mean-demo
```

Setting Up Routes

Routes are used as mapping service by using URL of an incoming request. Open the **server.js** file and setup the routing as shown below −

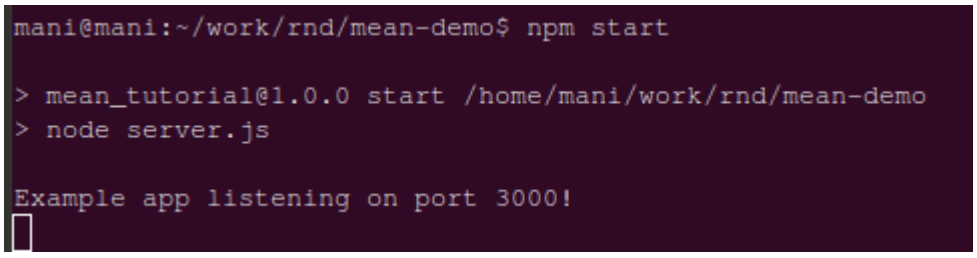
```
// modules =================================================
const express = require('express');
const app = express();

// set our port
const port = 3000;
app.get('/', (req, res) ⇒ res.send('Welcome to Tutorialspoint!'));

//defining route
app.get('/tproute', function (req, res) {
   res.send('This is routing for the application developed using Node and Express...')
});

// startup our app at http://localhost:3000
app.listen(port, () ⇒ console.log(`Example app listening on port ${port}!`));
```

Running Application

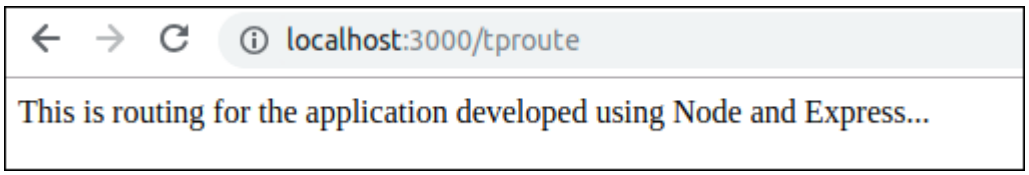
Next, run the application with the below command −

```
$ npm start
```

You will get a confirmation as shown in the image below −



Now, go to browser and type **http://localhost:3000/myroute**. You will get the page as shown in the image below −



MEAN.JS - Build Data Model

In this chapter, we shall demonstrate how to use data model in our Node-express application.

MongoDB is an open source NoSQL database that saves the data in JSON format. It uses the document oriented *data model* to store the data instead of using table and rows as we use in the relational databases. In this chapter, we are using Mongodb to build data model.

Data model specifies what data is present in a document, and what data should be there in a document. Refer the Official MongoDB installation , to install the MongoDB.

We shall use our previous chapter code. You can download the source code in this link . Download the zip file; extract it in your system. Open the terminal and run the below command to install npm module dependencies.

```
$ cd mean-demo
$ npm install
```

Adding Mongoose to Application

Mongoose is a data modelling library that specifies environment and structure for the data by making MongoDB powerful. You can install Mongoose as an npm module through the command line. Go to your root folder and run the below command −

```
$ npm install --save mongoose
```

The above command will download the new package and install it into *the node_modules* folder. The *--save* flag will add this package to *package.json* file.

```
{
   "name": "mean_tutorial",
   "version": "1.0.0",
   "description": "this is basic tutorial example for MEAN stack",
   "main": "server.js",
   "scripts": {
      "test": "test"
   },
   "keywords": [
      "MEAN",
      "Mongo",
      "Express",
      "Angular",
      "Nodejs"
   ],
```

```
   "author": "Manisha",
   "license": "ISC",
   "dependencies": {
      "express": "^4.17.1",
      "mongoose": "^5.5.13"
   }
}
```

Setting up Connection File

To work with data model, we will be using *app/models* folder. Let's create model *students.js* as below −

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

// define our students model
// module.exports allows us to pass this to other files when it is called
module.exports = mongoose.model('Student', {
   name : {type : String, default: ''}
});
```

You can setup the connection file by creating the file and using it in the application. Create a file called *db.js* in *config/db.js*. The file contents are as below −

```
module.exports = {
   url : 'mongodb://localhost:27017/test'
}
```

Here *test* is the database name.

Here it is assumed that you have installed MongoDB locally. Once installed start Mongo and create a database by name test. This db will have a collection by name students. Insert some data to this colection. In our case, we have inserted a record using db.students.insertOne({ name: 'Manisha' , place: 'Pune', country: 'India'});

Bring the *db.js* file into application, i.e., in *server.js*. Contents of the file are as shown below −

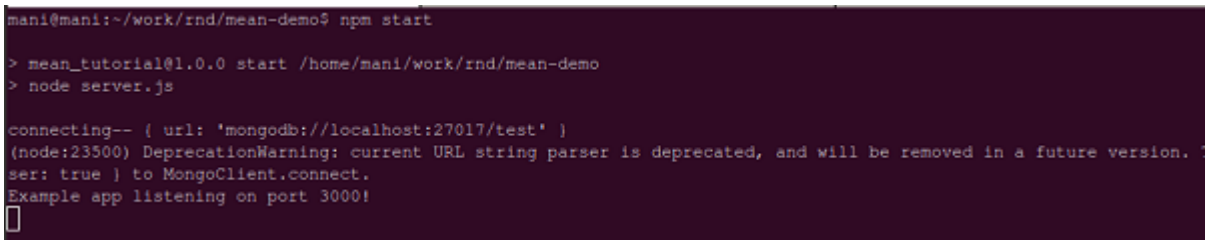
```
// modules =================================================
const express = require('express');
const app = express();
var mongoose = require('mongoose');
// set our port
const port = 3000;
// configuration ===========================================

// config files
```

```
var db = require('./config/db');
console.log("connecting--",db);
mongoose.connect(db.url); //Mongoose connection created

// frontend routes =========================================================
app.get('/', (req, res) => res.send('Welcome to Tutorialspoint!'));

//defining route
app.get('/tproute', function (req, res) {
   res.send('This is routing for the application developed using Node and Express...')
});

// sample api route
// grab the student model we just created
var Student = require('./app/models/student');
app.get('/api/students', function(req, res) {
   // use mongoose to get all students in the database
   Student.find(function(err, students) {
      // if there is an error retrieving, send the error.
      // nothing after res.send(err) will execute
      if (err)
         res.send(err);
      res.json(students); // return all students in JSON format
   });
});
// startup our app at http://localhost:3000
app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

Next, run the application with the below command −

```
$ npm start
```

You will get a confirmation as shown in the image below −



Now, go to browser and type **http://localhost:3000/api/students**. You will get the page as shown in the image below −

MEAN.JS - REST API

In this chapter, we will see our application interacting via a REST API with our database by using HTTP methods. The term *REST* stands for REpresentational State Transfer, which is an architectural style designed to communicate with web services and *API* stands for Application Program Interface that allows interacting applications with each other.

First, we will create RESTful API to get all items, create the item and delete an item. For each item, *_id* will be generated automatically by MongoDB. The below table describes how application should request data from API −

HTTP Method	URL Path	Description
GET	**/api/students**	It is used to get all the students from collection Student.
POST	**/api/students/send**	It is used to create a student record in collection Student.
DELETE	**/api/students/student_id**	It is used to delete a student record from collection Student.

RESTful API Routes

We will first discuss Post Method in RESTful API Routes.

POST

First let's create a record in the collection Student via our REST API. The code for this particular case can be found in *server.js* file. For reference, a part of code is pasted here −

```
app.post('/api/students/send', function (req, res) {
   var student = new Student(); // create a new instance of the student model
   student.name = req.body.name; // set the student name (comes from the request)
   student.save(function(err) {
      if (err)
         res.send(err);
         res.json({ message: 'student created!' });
   });
});
```

Execution

You can download the source code for this application in this link . Download the zip file; extract it in your system. Open the terminal and run the below command to install npm module dependencies.

```
$ cd mean-demon-consuming_rest_api
$ npm install
```

To parse the request, we would need body parser package. Hence, run the below command to include in your application.

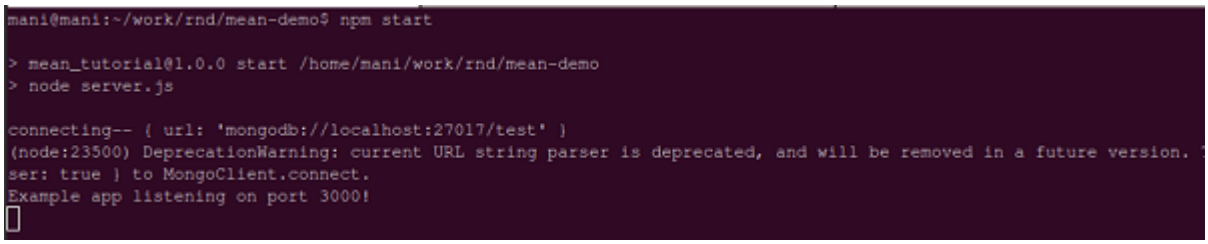
```
npm install --save body-parser
```

The attached source code already has this dependency, hence no need to run the above command, it is just for your info.

To run the application, navigate to your newly created project directory and run with the command given below −

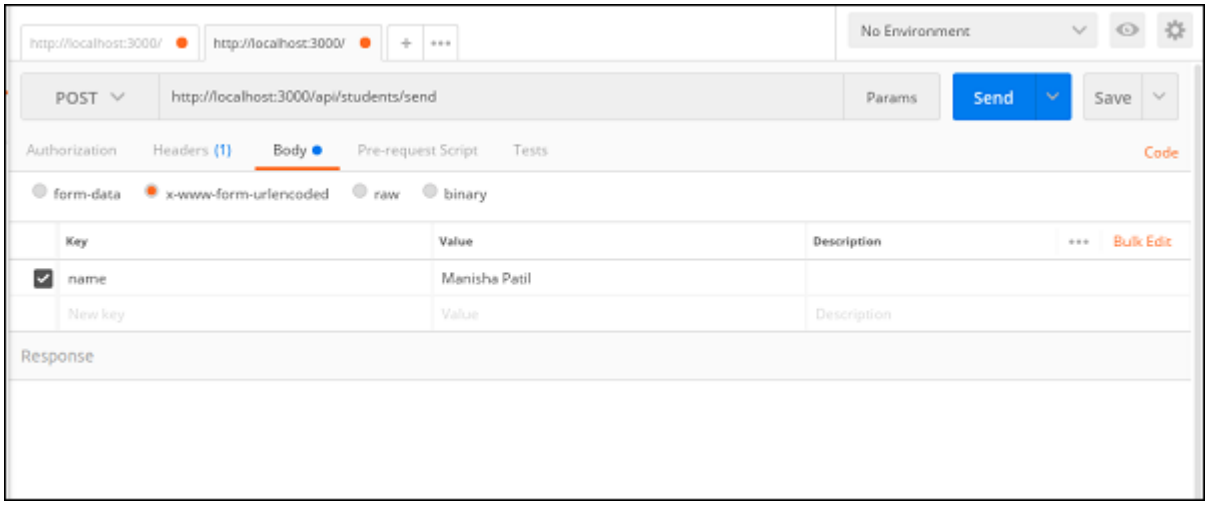
```
npm start
```

You will get a confirmation as shown in the image below −



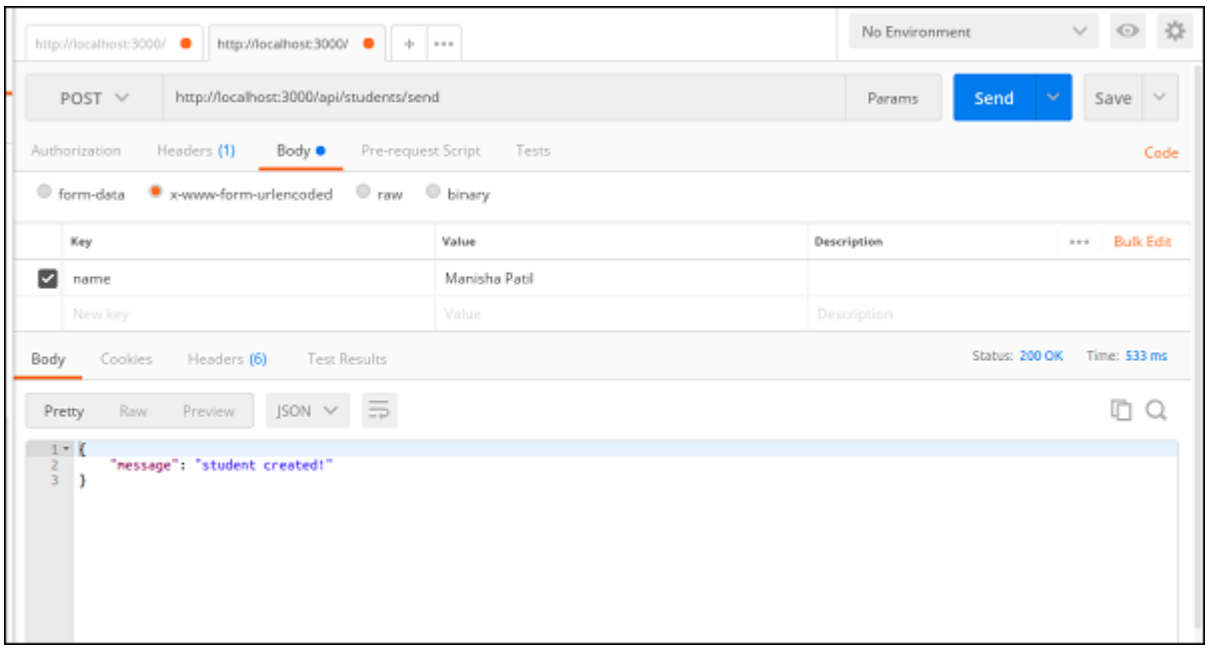
There are many tools to test the API calls, here we are using one of the user friendly extensions for Chrome called *Postman REST Client*.

Open the Postman REST Client, enter the URL as ***http://localhost:3000/api/students/send***, select the *POST method*. Next, enter request data as shown below −



Notice that we are sending the name data as *x-www-form-urlencoded*. This will send all of our data to the Node server as query strings.

Click on the *Send* button to create a student record. A success message will appear as shown below −



GET

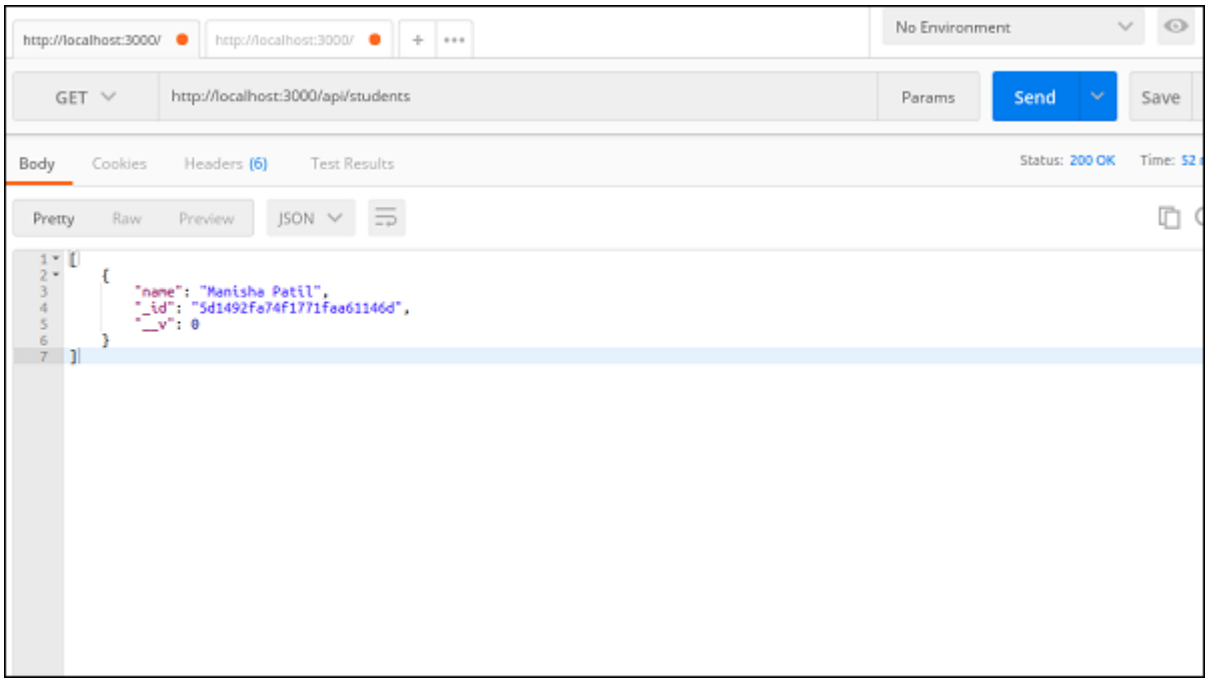
Next, let's get all the student records from the mongodb. Following route needs to be written. You can find full code in *server.js* file.

```
app.get('/api/students', function(req, res) {
   // use mongoose to get all students in the database
   Student.find(function(err, students) {
      // if there is an error retrieving, send the error.
      // nothing after res.send(err) will execute
      if (err)
```

```
         res.send(err);
      res.json(students); // return all students in JSON format
   });
});
```

Next, open the Postman REST Client, enter the URL as

http://localhost:3000/api/students, select the *GET* method and click on the Send button to get all the students.



DELETE

Next, let's see how to delete a record from our mongo collection via REST api call.

Following route needs to be written. You can find full code in *server.js* file.

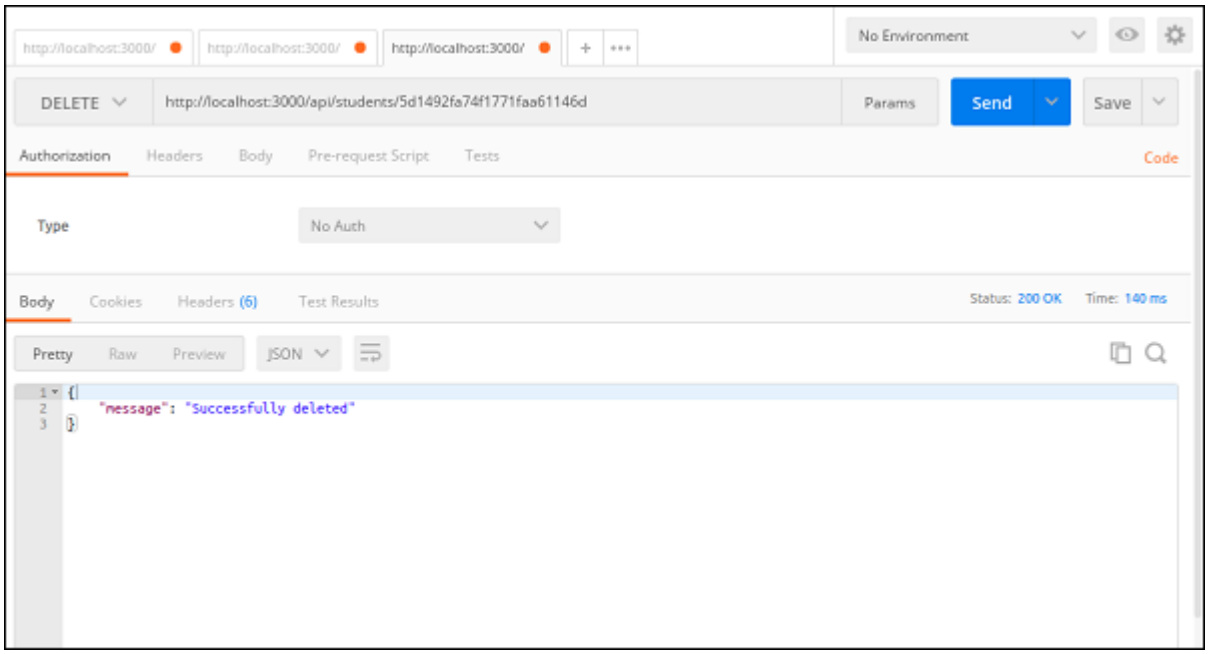
```
app.delete('/api/students/:student_id', function (req, res) {
   Student.remove({
      _id: req.params.student_id
   }, function(err, bear) {
      if (err)
         res.send(err);
      res.json({ message: 'Successfully deleted' });
   });
});
```

Next, open the Postman REST Client, enter the URL as

http://localhost:3000/api/students/5d1492fa74f1771faa61146d

(here 5d1492fa74f1771faa61146d is the record we will be deleting from the collection Student).

Select the *DELETE* method and click on the *Send* button to get all the students.



You can check the MongoDB for the deleted data, by making GET call to ***http://localhost:3000/api/students/5d1492fa74f1771faa61146d***.

MEAN.JS - Angular Components in App

In this chapter, we will add angular components to an application. It is a web front end JavaScript framework, which allows creating dynamic, single page applications by using Model View Controller (MVC) pattern. In the MEAN.JS architecture chapter, you have seen how AngularJS will process the client request and get the result from database.

Getting to know AngularJS

AngularJS is an open-source web application framework that uses HTML as template language and extends the HTML's syntax to express your application components clearly. AngularJS provides some basic features such as data binding, model, views, controllers, services etc. For more information on AngularJS, refer to this link .

You can make the page an Angular application by adding Angular in the page. It can be added just by using an external JavaScript file, which can be either downloaded or can be referenced directly with a CDN version.

Consider we have downloaded file and referenced it locally by adding to the page as follows −

```
<script src="angular.min.js"></script>
```

Now, we need to tell Angular that this page is an Angular application. Therefore, we can do this by adding an attribute, ng-app to the <html> or <body> tag as shown below −

```
<html ng-app>
or
<body ng-app>
```

The ng-app can be added to any element on the page, but it is often put into the <html> or <body> tag so that Angular can work anywhere within the page.

Angular Application as a Module

To work with an Angular application, we need to define a module. It is a place where you can group the components, directives, services, etc., which are related to the application. The module name is referenced by ng-app attribute in the HTML. For instance, we will say Angular application module name as myApp and can be specified in the <html> tag as shown below −

```
<html ng-app="myApp">
```

We can create definition for the application by using below statement in an external JavaScript file −

```
angular.module('myApp', []); //The [] parameter specifies dependent modules in the m
```

Defining Controller

AngularJS application relies on controllers to control the flow of data in the application. A controller is defined by using *ng-controller* directive.

For instance, we will attach the controller to the body by using ng-controller directive, along with name of the controller you want to use. In the below line, we are using name of the controller as "myController".

```
<body ng-controller="myController">
```

You can attach a controller (myController) to an Angular module (myApp) as shown below −

```
angular
.module('myApp')
.controller('myController', function() {
   // controller code here
});
```

It is better to use named function instead of an anonymous function for readability, re-usability, and testability. In the below code, we are using new named function "myController" to hold the controller code −

```
var myController = function() {
   // controller code here
};
angular
.module('myApp')
.controller('myController', myController);
```

For more information on controllers, refer to this link .

Defining Scope

Scope is a special JavaScript object that connects controller with views and contains model data. In controllers, model data is accessed via $scope object. The controller function takes $scope parameter which has been created by Angular and it gives direct access to the model.

The below code snippet specifies how to update controller function to receive the $scope parameter and sets the default value −

```
var myController = function($scope) {
   $scope.message = "Hello World...";
};
```

For more information on controllers, refer to this link . In the next chapter, we will start creating single page application by using Angular.

MEAN.JS - Building Single Page with Angular

In the MEAN stack, Angular is known as second JavaScript framework, which allows creating single page applications in a clean Model View Controller (MVC) way.

AngularJS as a front-end Framework uses following things −

- Uses Bower to install files and libraries
- Uses controllers and services for Angular application structure
- Creates different HTML pages
- Uses *ngRoute* module to handle routing and services for AngularJS application
- Uses Bootstrap to make an application attractive

Setting Up Our Angular Application

Let us build a simple application that has a Node.js backend and an AngularJS frontend. For our Angular application, we will want −

- Two different pages (Home, Student)
- A different angular controller for each
- No page refresh when switching pages

Bower and Pulling in Components

We will need certain files for our application like bootstrap and angular. We will tell bower to fetch those components for us.

First, install bower on your machine executing the below command on your command terminal −

```
npm install -g bower
```

This will install bower and make it accessible globally on your system. Now place the files .bowerrc and bower.json under your root folder. In our case it is **mean-demo**. Contents of both the files are as below −

.bowerrc - This will tell Bower where to place our files −

```
{
   "directory": "public/libs"
}
```

bower.json - This is similar to package.json and will tell Bower which packages are needed.

```
{
   "name": "angular",
   "version": "1.0.0",
   "dependencies": {
      "bootstrap": "latest",
      "angular": "latest",
      "angular-route": "latest"
   }
}
```

Next, install Bower components by using the below command. You can see bower pull in all the files under *public/libs*.

```
$ bower install
```

Our directory structure would be as follows −

```
mean-demo
   -app
   -config
   -node_modules
   -public
      -js
         --controllers
   -MainCtrl.js
   -StudentCtrl.js
      --app.js
      --appRoutes.js
   -libs
   -views
      --home.html
   --student.html
      -index.html
   -bower.json
   -package.json
   -server.js
```

Angular Controllers

Our controller (public/js/controllers/MainCtrl.js) is as follows −

```
angular.module('MainCtrl', []).controller('MainController', function($scope) {
   $scope.tagline = 'Welcome to tutorials point angular app!';
});
```

Controller public/js/controllers/StudentCtrl.js is as follows −

```
angular.module('StudentCtrl', []).controller('StudentController', function($scope) {
   $scope.tagline = 'Welcome to Student section!';
});
```

Angular Routes

Our routes file (public/js/appRoutes.js) is as follows −

```
angular.module('appRoutes', []).config(['$routeProvider',
   '$locationProvider', function($routeProvider, $locationProvider) {
   $routeProvider
      // home page
```

```
      .when('/', {
         templateUrl: 'views/home.html',
         controller: 'MainController'
      })
      // students page that will use the StudentController
      .when('/students', {
         templateUrl: 'views/student.html',
         controller: 'StudentController'
      });
   $locationProvider.html5Mode(true);
}]);
```

Now that we have our controllers, and routes, we will combine them all and inject these modules into our main *public/js/app.js* as follows −

```
angular.module('sampleApp', ['ngRoute', 'appRoutes', 'MainCtrl', 'StudentCtrl']);
```

View File

Angular uses the template file, which can be injected into the <div ng-view> in the index.html file. The ng-view directive creates a place holder, where a corresponding view (HTML or ng-template view) can be placed based on the configuration. For more information on angular views, visit this link .

When you are ready with routing, create smaller template files and inject them into *index.html* file. The *index.html* file will have following code snippet −

```
<!doctype html>
<html lang="en">
   <head>
      <meta charset="UTF-8">
      <base href="/">
      <title>Tutorialspoint Node and Angular</title>

      <!-- CSS -->
      <link rel="stylesheet" href="libs/bootstrap/dist/css/bootstrap.min.css">
      <link rel="stylesheet" href="css/style.css"> <!-- custom styles -->

      <!-- JS -->
      <script src="libs/angular/angular.min.js"></script>
      <script src="libs/angular-route/angular-route.min.js"></script>

      <!-- ANGULAR CUSTOM -->
      <script src="js/controllers/MainCtrl.js"></script>
      <script src="js/controllers/StudentCtrl.js"></script>
      <script src="js/appRoutes.js"></script>
```

```
      <script src="js/app.js"></script>
   </head>
   <body ng-app="sampleApp" ng-controller="MainController">
      <div class="container">

         <!-- HEADER -->
         <nav class="navbar navbar-inverse">
            <div class="navbar-header">
               <a class="navbar-brand" href="/">Tutorial</a>
            </div>
            <ul class="nav navbar-nav">
               <li><a href="/students">Students</a></li>
            </ul>
         </nav>
         <!-- ANGULAR DYNAMIC CONTENT -->
         <div ng-view></div>
      </div>
   </body>
</html>
```

Running Application

Execution

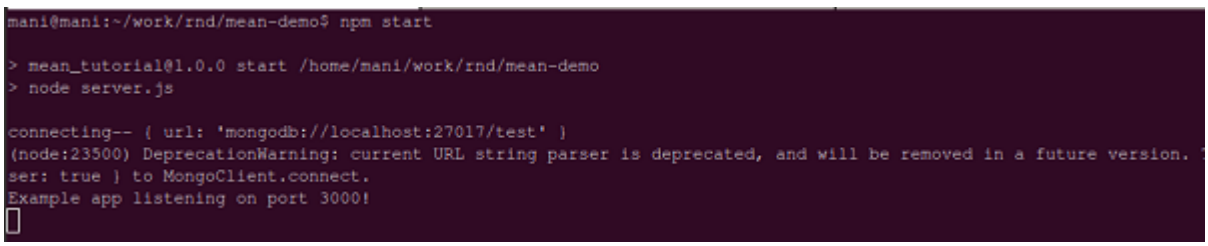
You can download the source code for this application in this link . Download the zip file; extract it in your system. Open the terminal and run the below command to install npm module dependencies.

```
$ cd mean-demo
$ npm install
```

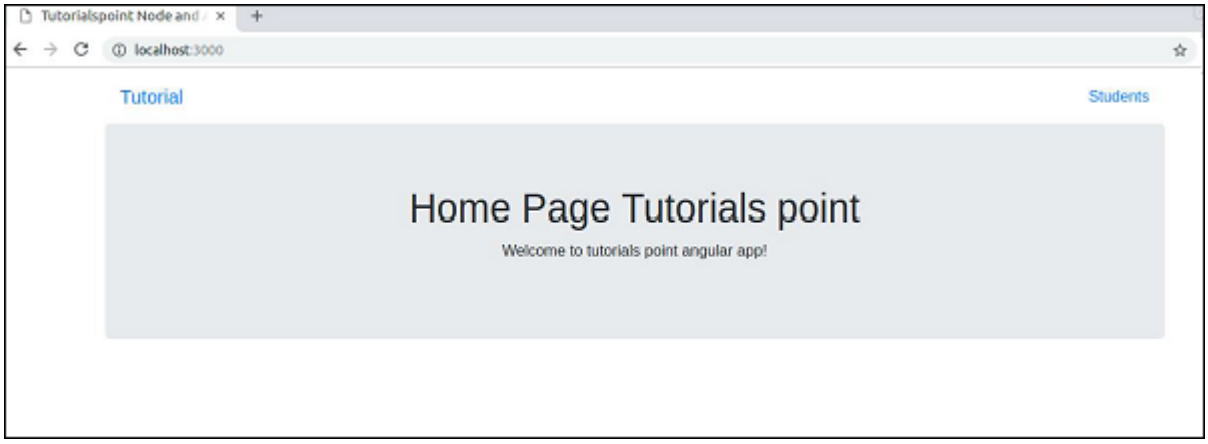
Next run the below command −

```
$ node start
```

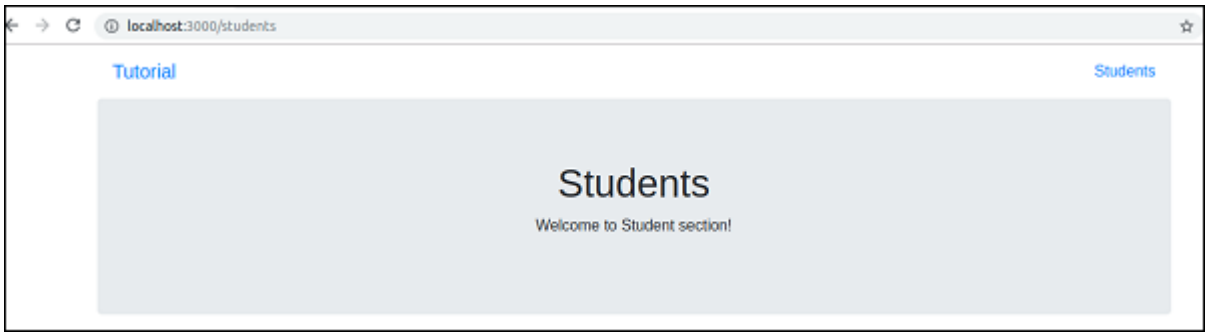
You will get a confirmation as shown in the image below −



Now, go to browser and type **http://localhost:3000**. You will get the page as shown in the image below −



Click on *Students* link, you will see screen as below −



Our Angular frontend will use the template file and inject it into the <div ng-view> in our *index.html* file. It will do this without a page refresh.

MEAN.JS - Building an SPA: The next level

In the previous chapter, we have seen creation of single page meanjs application using Angularjs. In this chapter, let's see how Angular application uses API to get the data from Mongodb.

You can download the source code for this application in this link . Download the zip file; extract it in your system.

Directory structure of our source code is as follows −

```
mean-demo
   -app
      -models
         -student.js
   -config
      -db.js
   -public
```

```
        -js
           -controllers
              -MainCtrl.js
              -StudentCtrl.js
           -services
              -StudentService.js
           -app.js
           -appRoutes.js
        -views
           -home.html
           -student.html
        -index.html
     -.bowerrc
     -bower.json
     -package.json
     -server.js
```

In this application, we have created a view (home.html), which will list all students from collection Student, allow us to create a new **student** record and allow us to delete the student record. All these operations are performed via REST API calls.

Open the terminal and run the below command to install npm module dependencies.

```
$ npm install
```

Next, install Bower components by using the below command. You can see bower pull in all the files under public/libs.

```
$ bower install
```

The node configuration for an application will be saved in the server.js file. This is main file of node app and will configure the entire application.

```
// modules ================================================
const express = require('express');
const app = express();
var bodyParser = require('body-parser');
var mongoose = require('mongoose');
var methodOverride = require('method-override');
// set our port
const port = 3000;
// configuration ==========================================
// configure body parser
app.use(bodyParser.json()); // parse application/json

// parse application/vnd.api+json as json
```

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

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

// override with the X-HTTP-Method-Override header in the request.
app.use(methodOverride('X-HTTP-Method-Override')); simulate DELETE/PUT

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

// config files
var db = require('./config/db');
console.log("connecting--",db);
mongoose.connect(db.url); //Mongoose connection created

// grab the student model
var Student = require('./app/models/student');
function getStudents(res) {
   Student.find(function (err, students) {
      // if there is an error retrieving, send the error. nothing after res.send(err)
         if (err) {
            res.send(err);
         }
         res.json(students); // return all todos in JSON format
      });
   };
app.get('/api/studentslist', function(req, res) {
   getStudents(res);
});
app.post('/api/students/send', function (req, res) {
   var student = new Student(); // create a new instance of the student model
   student.name = req.body.name; // set the student name (comes from the request)
   student.save(function(err) {
      if (err)
         res.send(err);
         getStudents(res);
   });
});
app.delete('/api/students/:student_id', function (req, res) {
   Student.remove({
      _id: req.params.student_id
      }, function(err, bear) {
         if (err)
            res.send(err);
         getStudents(res);
      });
```

```
});
// startup our app at http://localhost:3000
app.listen(port, () ⇒ console.log(`Example app listening on port ${port}!`));
```

Defining Frontend Route

The *public/index.html* file will have following code snippet −

```
<!doctype html>
<html lang="en">
   <head>
      <meta charset="UTF-8">
      <base href="/">
      <title>Tutorialspoint Node and Angular</title>

      <!-- CSS -->
      <link rel="stylesheet" href="libs/bootstrap/dist/css/bootstrap.min.css">
      <link rel="stylesheet" href="css/style.css"> <!-- custom styles -->

      <!-- JS -->
      <script src="libs/angular/angular.min.js"></script>
      <script src="libs/angular-route/angular-route.min.js"></script>

      <!-- ANGULAR CUSTOM -->
      <script src="js/controllers/MainCtrl.js"></script>
      <script src="js/controllers/StudentCtrl.js"></script>
      <script src="js/services/StudentService.js"></script>
      <script src="js/appRoutes.js"></script>
      <script src="js/app.js"></script>
   </head>
   <body ng-app="sampleApp" ng-controller="MainController">
      <div class="container">
         <!-- HEADER -->
         <nav class="navbar navbar-inverse">
            <div class="navbar-header">
               <a class="navbar-brand" href="/">Tutorial</a>
            </div>
            <ul class="nav navbar-nav">
               <li><a href="/students">Students</a></li>
            </ul>
         </nav>

         <!-- ANGULAR DYNAMIC CONTENT -->
         <div ng-view></div>
      </div>
```

```
   </body>
</html>
```

We have written a service to make the API calls and execute the API requests. Our service, *StudentService* looks as below −

```
angular.module('StudentService', [])
// super simple service
// each function returns a promise object
.factory('Student', ['$http',function($http) {
   return {
      get : function() {
         return $http.get('/api/students');
      },
      create : function(student) {
         return $http.post('/api/students/send', student);
      },
      delete : function(id) {
         return $http.delete('/api/students/' + id);
      }
   }
}]);
```

Our controller (MainCtrl.js) code is as below −

```
angular.module('MainCtrl', []).controller('MainController',
   ['$scope','$http','Student',function($scope, $http, Student) {
   $scope.formData = {};
   $scope.loading = true;
   $http.get('/api/studentslist').
   then(function(response) {
      $scope.student = response.data;
   });
   // CREATE
   // when submitting the add form, send the text to the node API
   $scope.createStudent = function() {
      // validate the formData to make sure that something is there
      // if form is empty, nothing will happen
      if ($scope.formData.name != undefined) {
         $scope.loading = true;
         // call the create function from our service (returns a promise object)
         Student.create($scope.formData)
         // if successful creation, call our get function to get all the new Student
         .then(function (response){
            $scope.student = response.data;
            $scope.loading = false;
```

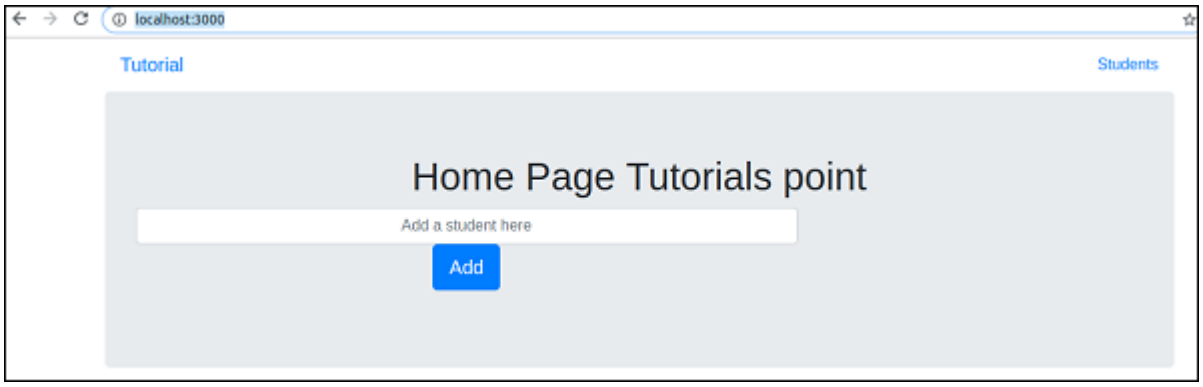
```
            $scope.formData = {}
         },    function (error){
         });
      }
   };
   // DELETE
   ==================================================================
   // delete a todo after checking it
   $scope.deleteStudent = function(id) {
      $scope.loading = true;
      Student.delete(id)
      // if successful delete, call our get function to get all the new Student
      .then(function(response) {
         $scope.loading = false;
         new list of Student
      });
   };
}]);
```

Running Application

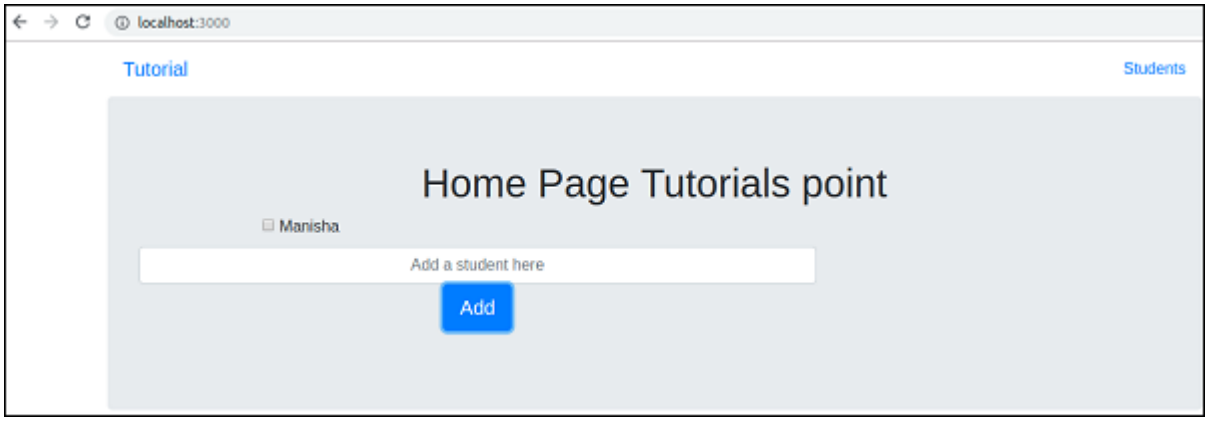
Navigate to your project directory and run the command given below −

```
$ npm start
```

Now navigate to **http://localhost:3000** and you will get the page as shown in the image below −



Enter some text in the text box and click on **Add** button. A record gets added and displayed as follows −



You can delete the record by checking the check box.