How to Create Node.js, Express.js and MongoDB CRUD Web Application

by Didin J. on Mar 01, 2017

> Tutorial of creating Node.js, Express.js and MongoDB CRUD Web Application from scratch using Express.js basic feature.

This is the tutorial of how to create Node.js, Express.js and MongoDB CRUD Web Application from scratch with basic use of Express.js. For data modeling, we will use Mongoose.js module and for UI (User Interface) or front end, we use "ejs" instead of jade that comes with Express.js project generation. Don't waste your time, let's get started.

1. Create Web Application Project

As usual, in every of our tutorial always starting with project creation. Right now, we are creating new web application project using Express.js project generator that works in command line. We assume that you are already installed required tools or read the previous tutorial. Open the terminal or cmd then type this command.

```
express node-crud --view=ejs
```

That command will create a project folder called "node-crud" with view template using "ejs" extension.

Next, go to newly created project folder then install node modules.

```
cd node-crud && npm install
```

Next, we are installing Mongoose.js module by this command.

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

Now, open and edit app.js from the root of the project folder. Add Mongoose.js to 'require' and call connection to MongoDB.

```
var mongoose = require('mongoose');
mongoose.Promise = global.Promise;

mongoose.connect('mongodb://localhost/product')
  .then(() =>  console.log('connection succesful'))
  .catch((err) => console.error(err));
```

We have to test connection to MongoDB server by running the application using "nodemon". If there's no "nodemon" installed, run this command first.

```
sudo npm install -g nodemon
```

After that, run the application.

```
nodemon
```

You should see this message in console or terminal if everything is ok.

```
[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node ./bin/www`
connection succesful
```

2. Create Mongoose.js Model

To create Mongoose.js model, first, create a models folder.

```
mkdir models
```

Then create new Javascript file that uses for Mongoose.js model.

```
touch models/Employee.js
```

That command will create Employee.js which means we will create a model for Employee data. Now, open and edit that file and add Mongoose require.

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

Then add model fields like this.

```
var EmployeeSchema = new mongoose.Schema({
  name: String,
  address: String,
  position: String,
  salary: Number,
  updated_at: { type: Date, default: Date.now },
});
```

That Schema will mapping to MongoDB collections called product. If you want to know more about Mongoose Schema Datatypes you can find it here. Next, export that schema.

```
module.exports = mongoose.model('Employee', EmployeeSchema);
```

3. Create Controller for CRUD Operations

To make our application more modular and meet MVC pattern, we have to create the controller for CRUD operations. For that, create new "controllers" folder then create new Javascript file as the controller.

```
mkdir controllers
touch controllers/EmployeeController.js
```

Open and edit EmployeeController.js file. Add this require to the file.

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

Add model require.

```
var Employee = mongoose.model("Employee");
```

Create controller object for CRUD operations.

```
var employeeController = {};
```

Add show list of employees function.

```
employeeController.list = function(req, res) {
  Employee.find({}).exec(function (err, employees) {
    if (err) {
      console.log("Error:", err);
    }
    else {
      res.render("../views/employees/index", {employees: employees});
    }
  });
};
```

Add show single employee by id function.

```
employeeController.show = function(req, res) {
  Employee.findOne({_id: req.params.id}).exec(function (err, employee) {
    if (err) {
      console.log("Error:", err);
    }
    else {
      res.render("../views/employees/show", {employee: employee});
    }
  });
};
```

Add create employee function, it just redirects to create the page.

```
employeeController.create = function(req, res) {
  res.render("../views/employees/create");
};
```

Add save new employee function.

```
employeeController.save = function(req, res) {
  var employee = new Employee(req.body);

  employee.save(function(err) {
    if(err) {
      console.log(err);
      res.render("../views/employees/create");
    } else {
      console.log("Successfully created an employee.");
      res.redirect("/employees/show/"+employee._id);
    }
  });
};
```

Add edit employee by id function, it just redirects to edit page.

```
employeeController.edit = function(req, res) {
  Employee.findOne({_id: req.params.id}).exec(function (err, employee) {
    if (err) {
      console.log("Error:", err);
    }
    else {
      res.render("../views/employees/edit", {employee: employee});
    }
  });
};
```

Add update employee function for updating currently edited employee.

```
employeeController.update = function(req, res) {
  Employee.findByIdAndUpdate(req.params.id, { $set: { name: req.body.name, address: req.body.address, position: req.body.position, salary: req.body.sa
lary }}, { new: true }, function (err, employee) {
    if (err) {
      console.log(err);
      res.render("../views/employees/edit", {employee: req.body});
    }
    res.redirect("/employees/show/"+employee._id);
  });
};
```

Add delete employee by id function for remove single employee data.

```
employeeController.delete = function(req, res) {
  Employee.remove({_id: req.params.id}, function(err) {
    if(err) {
      console.log(err);
    }
    else {
      console.log("Employee deleted!");
      res.redirect("/employees");
    }
  });
};
```

Finally, export employee controller as a module.

```
module.exports = employeeController;
```

4. Create Routes

To redirect the request to the controller before call query or display page, add routes for employee module. Now, create new Javascript file inside routes folder.


```
touch routes/employees.js
```

Open and edit employee.js. First, add express require and create router.

```
var express = require('express');
var router = express.Router();
```

Add require that point to Employee controller.

```
var employee = require("../controllers/EmployeeController.js");
```

Next, add all routes to CRUD functions like below.

```
// Get all employees
router.get('/', employee.list);

// Get single employee by id
router.get('/show/:id', employee.show);

// Create employee
router.get('/create', employee.create);

// Save employee
router.post('/save', employee.save);

// Edit employee
router.get('/edit/:id', employee.edit);

// Edit update
router.post('/update/:id', employee.update);

// Edit update
router.post('/delete/:id', employee.delete);
```

Now, export router as a module.

```
module.exports = router;
```

Next, open and edit app.js then add employee route as require after users require.

```
var employees = require('./routes/employees');
```

Then add use after use of users.

```
app.use('/employees', employees);
```

5. Create Views for CRUD User Interface.

Because we are creating separate views for employee CRUD, add a new folder in views folder and name it employee.

```
mkdir views/employee
```

Create new index.ejs inside views/employee folder.

```
touch views/employee/index.ejs
```

Open and edit index.ejs then add HTML codes like this.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Employee List</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <div class="container">
      <h3><a href="/employees/create">Create Employee</a></h3>
      <h1>Employee List</h1>
      <% if(employees.length>0) { %>
      <table>
        <thead>
          <tr>
            <th>Employee Name</th>
            <th>Position</th>
          </tr>
        </thead>
        <tbody>

          <% for(var i=0; i<employees.length;i++) { %>
            <tr>
              <td><a href="/employees/show/<%= employees[i]._id%>"><%= employees[i].name%></a></td>
              <td><%= employees[i].position%></td>
            </tr>
          <% } %>
        </tbody>
      </table>
      <% } else { %>
        <div>No employees found.</div>
      <% } %>
    </div>
  </body>
</html>
```

Create new HTML file for show employee page.

```
touch views/employees/show.ejs
```

Open and edit show.ejs then add this lines of HTML codes.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Employee Detail</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <div class="container">
      <h3><a href="/employees">Employee List</a></h3>
      <h1>Employee Detail</h1>
      <table>
        <tbody>
          <tr>
            <td>Name</td>
            <td>:</td>
            <td><%= employee.name %></td>
          </tr>
          <tr>
            <td>Address</td>
            <td>:</td>
            <td><%= employee.address %></td>
          </tr>
          <tr>
            <td>Position</td>
            <td>:</td>
            <td><%= employee.position %></td>
          </tr>
          <tr>
            <td>Salary</td>
            <td>:</td>
            <td>$<%= employee.salary %>/year</td>
          </tr>
        </tbody>
      </table>
      <h3><a href="/employees/edit/<%= employee._id%>">EDIT</a></h3>
      <form action="/employees/delete/<%= employee._id%>" method="post">
        <button type="submit">DELETE</button>
      </form>
    </div>
  </body>
</html>
```

Create new HTML file for creating new employee form page.

```
touch views/employees/create.ejs
```

Open and edit create.ejs then add this lines of HTML codes.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Create Employee</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <div class="container">
      <h3><a href="/employees">Employee List</a></h3>
      <h1>Create New Employee</h1>
      <form action="/employees/save" method="post">
        <table>
          <tbody>
            <tr>
              <td>Name</td>
              <td><input type="text" name="name" /></td>
            </tr>
            <tr>
              <td>Address</td>
              <td><textarea name="address"></textarea></td>
            </tr>
            <tr>
              <td>Position</td>
              <td><input type="text" name="position" /></td>
            </tr>
            <tr>
              <td>Salary</td>
              <td><input type="number" name="salary" /></td>
            </tr>
            <tr>
              <td colspan="2"><input type="submit" value="Save" /></td>
            </tr>
          </tbody>
        </table>
      </form>
    </div>
  </body>
</html>
```

Create new HTML file for edit current employee.

```
touch views/employees/edit.ejs
```

Open and edit file edit.ejs then add this lines of HTML codes.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Edit Employee</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <div class="container">
      <h3><a href="/employees">Employee List</a></h3>
      <h1>Edit Employee</h1>
      <form action="/employees/update/<%= employee._id%>" method="post">
        <table>
          <tbody>
            <tr>
              <td>Name</td>
              <td><input type="text" name="name" value="<%= employee.name %>" /></td>
            </tr>
            <tr>
              <td>Address</td>
              <td><textarea name="address"><%= employee.address %></textarea></td>
            </tr>
            <tr>
              <td>Position</td>
              <td><input type="text" name="position" value="<%= employee.position %>" /></td>
            </tr>
            <tr>
              <td>Salary</td>
              <td><input type="number" name="salary" value="<%= employee.salary %>" /></td>
            </tr>
            <tr>
              <td colspan="2"><button type="submit">Update</button></td>
            </tr>
          </tbody>
        </table>
      </form>
    </div>
  </body>
</html>
```

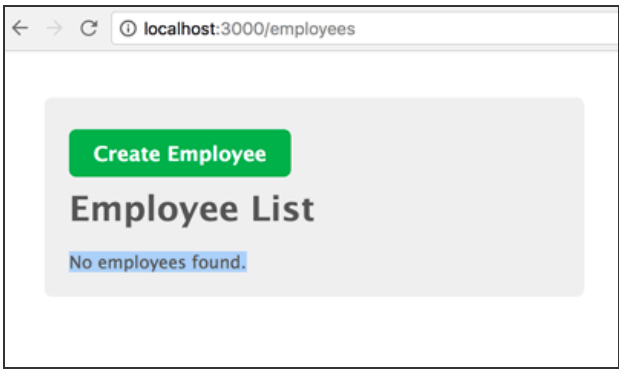
The last thing to do is styling the views. Just open and edit public/stylesheets/style.css then replace all CSS code with this codes.

```
body {
  padding: 30px;
  font: 14px "Lucida Grande", Helvetica, Arial, sans-serif;
  color: #555555;
}

a {
  color: #00B7FF;
}

.container {
  padding: 20px;
  background-color: #efefef;
  width: 400px;
  border-radius: 6px;
}
.container h3 a {
  text-decoration: none;
  padding: 10px 20px;
  background-color: #00AE4D;
  border-radius: 6px;
  color: #FFFFFF;
}

.container h3 a:hover, .container table tr td button:hover {
  background-color: #78F5AE;
  color: #999999;
}

.container table {
  width: 360px;
}

.container table tr td {
  padding: 5px;
}

.container table tr td input, .container table tr td textarea {
  border: solid 1px #dddddd;
  padding: 5px 10px;
  border-radius: 6px;
  width: 100%;
  color: #777777;
}

.container table tr td button {
  border: solid 1px #AABF5C;
  background-color: #00AE4D;
  border-radius: 6px;
  text-transform: uppercase;
  color: #FFFFFF;
  margin: 10px 0;
  padding: 10px 20px;
}

.container form button {
  border: solid 1px #CB2027;
  background-color: #DD4B39;
  border-radius: 6px;
  text-transform: uppercase;
  color: #FFFFFF;
  margin: 10px 0;
  padding: 10px 20px;
}

.container .list-table {
  width: 360px;
  border: solid 2px #DDDDDD;
  border-spacing: 0;
}

.container .list-table th {
  border-bottom: solid 2px #DDDDDD;
}
```

Finally, you just can run and test the application CRUD functions.

```
nodemon
```

Open your browser and go to this URL "localhost:3000/employees". If you see the page with the message "No employees found." and a create button, it is mean your CRUD function is ready to test in the browser.



> Notes: This tutorial just the basic use of express.js and mongoose.js on the top of node.js. For further simple and quicker method, you can find a lot of npm modules and library that ready for using in your real projects.

Here's the full source code from Github.

Thanks.

Follow

Previous Article

How to Create REST API Easily using Node.js, Express.js, Mongoose.js and MongoDB

Next Article

Node.js, Express.js, Mongoose.js and Passport.js Authentication

Related Articles

- How to Create REST API Easily using Node.js, Express.js, Mongoose.js and MongoDB
- Node.js, Express.js, Mongoose.js and Passport.js Authentication
- Tutorial Building CRUD App from Scratch using MEAN Stack (Angular 2)
- Getting Started Angular 4 using Angular CLI
- Building Chat Application using MEAN Stack (Angular 4) and Socket.io
- Node, Express, Mongoose and Passport.js REST API Authentication
- Node Express Passport Facebook Twitter Google GitHub Login
- Building CRUD Web Application using MERN Stack

- MEAN Stack (Angular 5) CRUD Web Application Example
- Mongo Express Vue Node.js (MEVN Stack) CRUD Web Application
- Node.js and MongoDB Slack Bot Example
- Setup Node.js, Nginx and MongoDB on Ubuntu 16.04 for Production
- Securing MEAN Stack (Angular 5) Web Application using Passport
- Securing MERN Stack Web Application using Passport

Programming Blog

Node.js (15)

Popular Articles: