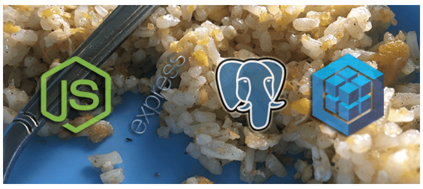
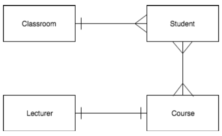
Node.js, Express.js, Sequelize.js and PostgreSQL RESTful API

by Didin J. on Jul 24, 2018



Step by step tutorial on building RESTful API using Node.js, Express.js, Sequelize.js and PostgreSQL.

A comprehensive step by step tutorial on building RESTful API using Node.js, Express.js, Sequelize.js, and PostgreSQL. In this tutorial, we will show how to create a little complex table association or relationship with CRUD (Create, Read, Update, Delete) operations. So, the association or relationship will be like this diagram.



Above diagrams describe:

- One classroom has many students and a student has one classroom
- Many students take many of courses and vice-versa
- One course has one lecturer and one lecturer teach one course

That the simple explanation for what we will do in the steps of this tutorial.

Table of Contents:

The following tools, frameworks, and modules are required for this tutorial:

- Node.js
- PostgreSQL Server
- Express.js
- Sequelize.js
- Terminal or Command Line
- Text Editor or IDE

We assume that you have installed PostgreSQL server in your machine or can use your own remote server (we are using PostgreSQL 9.5.13). Also, you have installed Node.js in your machine and can run `node`, `npm` or `yarn` command in your terminal or command line. Next, check their version by type this commands in your terminal or command line.

```
node -v
v8.11.1
npm -v
6.1.0
yarn -v
1.7.0
```

That the versions that we are uses. Let's continue with the main steps.

1. Create Express.js Project and Install Required Modules

Open your terminal or node command line the go to your projects folder. First, install express generator using this command.

```
sudo npm install express-generator -g
```

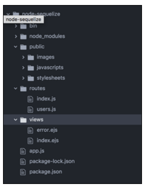
Next, create an Express.js app using this command.

```
express node-sequelize --view=ejs
```

This will create Express.js project with the EJS view instead of Jade view template because using '--view=ejs' parameter. Next, go to newly created project folder then install node modules.

```
cd node-sequelize && npm install
```

You should see the folder structure like this.



There's no view yet using the latest Express generator. We don't need it because we will create a RESTful API.

2. Add and Configure Sequelize.js Module and Dependencies

Before installing the modules for this project, first, install Sequelize-CLI by type this command.

```
sudo npm install -g sequelize-cli
```

To install Sequelize.js module, type this command.

```
npm install --save sequelize
```

Then install the module for PostgreSQL.

```
npm install --save pg pg-hstore
```

Next, create a new file at the root of the project folder.

```
touch .sequelizerc
```

Open and edit that file then add this lines of codes.

```
const path = require('path');

module.exports = {
  "config": path.resolve('./config', 'config.json'),
  "models-path": path.resolve('./models'),
  "seeders-path": path.resolve('./seeders'),
  "migrations-path": path.resolve('./migrations')
};
```

That files will tell Sequelize initialization to generate config, models, seeders and migrations files to specific directories. Next, type this command to initialize the Sequelize.

```
sequelize init
```

That command will create `config/config.json`, `models/index.js`, `migrations` and `seeders` directories and files. Next, open and edit `config/config.json` then make it like this.

```
{
  "development": {
    "username": "djamware",
    "password": "dj@mw@r3",
    "database": "node_sequelize",
    "host": "127.0.0.1",
    "dialect": "postgres"
  },
  "test": {
    "username": "root",
    "password": "dj@mw@r3",
    "database": "node_sequelize",
    "host": "127.0.0.1",
    "dialect": "postgres"
  },
  "production": {
    "username": "root",
    "password": "dj@mw@r3",
    "database": "node_sequelize",
    "host": "127.0.0.1",
    "dialect": "postgres"
  }
}
```

We use the same configuration for all environment because we are using same machine, server, and database for this tutorial.

Before run and test connection, make sure you have created a database as described in the above configuration. You can use the `psql` command to create a user and database.

```
psql postgres --u postgres
```

Next, type this command for creating a new user with password then give access for creating the database.

```
postgres-# CREATE ROLE djamware WITH LOGIN PASSWORD 'dj@mw@r3';
postgres-# ALTER ROLE djamware CREATEDB;
```

Quit `psql` then log in again using the new user that previously created.

```
postgres-# \q
psql postgres -U djamware
```

Enter the password, then you will enter this `psql` console.

```
psql (9.5.13)
Type "help" for help.

postgres=>
```

Type this command to creating a new database.

```
postgres=> CREATE DATABASE node_sequelize;
```

Then give that new user privileges to the new database then quit the `psql`.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE node_sequelize TO djamware;
postgres=> \q
```

3. Create or Generate Models and Migrations

We will use Sequelize-CLI for generate a new model. Type this command to create a model for `Classroom`, `Student`, `Lecturer`, `Course` and `StudentCourse`.

```
sequelize model:create --name Classroom --attributes class_name:string
sequelize model:create --name Student --attributes classroom_id:integer,student_name:string
sequelize model:create --name Lecturer --attributes lecturer_name:string
sequelize model:create --name Course --attributes lecturer_id:integer,course_name:string
sequelize model:create --name StudentCourse --attributes student_id:integer,course_id:integer
```

That command creates a model file to the model's folder and a migration file to folder migrations. Next, modify `models/classroom.js` then add association with `Student` model inside `associate` function.

```
Classroom.associate = function(models) {
  Classroom.hasMany(models.Student, {
    foreignKey: 'classroom_id',
    as: 'students',
  });
};
```

Next, modify `models/student.js` then add association with `Classroom` and `Coursemodel` models inside `associate` function.

```
Student.associate = function(models) {
  Student.belongsTo(models.Classroom);
  Student.belongsToMany(models.Course, {
    through: 'StudentCourse',
    as: 'courses',
    foreignKey: 'student_id'
  });
};
```

Next, modify `models/lecturer.js` then add the association with `Course` model inside `associate` function.

```
Lecturer.associate = function(models) {
  Lecturer.hasOne(models.Course, {
    foreignKey: 'lecturer_id',
    as: 'lecturer',
  });
};
```

Next, modify `models/course.js` then add association with `Student` and `Lecturer` models inside `associate` function.

```
Course.associate = function(models) {
  Course.belongsToMany(models.Student, {
    through: 'StudentCourse',
    as: 'students',
    foreignKey: 'course_id'
  });
  Course.belongsTo(models.Lecturer);
};
```

Finally, for migrations, there's nothing to change and they all ready to generate the table to PostgreSQL Database. Type this command to generate the table to the database.

```
sequelize db:migrate
```

4. Create Controller and Router for Classroom Model

To create the controller, first create a folder for controllers and new Javascript file by type this commands.

```
mkdir controllers
touch controllers/classroom.js
```

Open and edit `controllers/classroom.js` then add this lines of codes.

```
const Classroom = require('../models').Classroom;
const Student = require('../models').Student;

module.exports = {
  list(req, res) {
    return Classroom
      .findAll({
        include: [{
          model: Student,
          as: 'students'
        }],
        order: [
          ['createdAt', 'DESC'],
          [{ model: Student, as: 'students' }, 'createdAt', 'DESC'],
        ],
      })
      .then((classrooms) => res.status(200).send(classrooms))
      .catch((error) => { res.status(400).send(error); });
  },

  getById(req, res) {
    return Classroom
      .findById(req.params.id, {
        include: [{
          model: Student,
          as: 'students'
        }],
      })
      .then((classroom) => {
        if (!classroom) {
          return res.status(404).send({
            message: 'Classroom Not Found',
          });
        }
        return res.status(200).send(classroom);
      })
      .catch((error) => res.status(400).send(error));
  },

  add(req, res) {
    return Classroom
      .create({
        class_name: req.body.class_name,
      })
      .then((classroom) => res.status(201).send(classroom))
      .catch((error) => res.status(400).send(error));
  },

  update(req, res) {
    return Classroom
      .findById(req.params.id, {
        include: [{
          model: Student,
          as: 'students'
        }],
      })
      .then(classroom => {
        if (!classroom) {
          return res.status(404).send({
            message: 'Classroom Not Found',
          });
        }
        return classroom
          .update({
            class_name: req.body.class_name || classroom.class_name,
          })
          .then(() => res.status(200).send(classroom))
          .catch((error) => res.status(400).send(error));
      })
      .catch((error) => res.status(400).send(error));
  },

  delete(req, res) {
    return Classroom
      .findById(req.params.id)
      .then(classroom => {
        if (!classroom) {
          return res.status(400).send({
            message: 'Classroom Not Found',
          });
        }
        return classroom
          .destroy()
          .then(() => res.status(204).send())
          .catch((error) => res.status(400).send(error));
      })
      .catch((error) => res.status(400).send(error));
  },
};
```

In that controller, we have all CRUD (Create, Read, Update and Delete) functions. To make this controller available via controllers folder, add this files for declaring this controller file and other controllers files.

```
touch controllers/index.js
```

Open and edit that file then add this lines of Javascript codes.

```
const classroom = require('./classroom');

module.exports = {
  classroom,
};
```

For the router, we will use the existing router that generated by Express Generator. Open and edit `routes/index.js` then declare the Classroom controller after another variables.

```
const classroomController = require('../controllers').classroom;
```

Add this routes after existing route for Classroom controller.

```
router.get('/api/classroom', classroomController.list);
router.get('/api/classroom/:id', classroomController.getById);
router.post('/api/classroom', classroomController.add);
router.put('/api/classroom/:id', classroomController.update);
router.delete('/api/classroom/:id', classroomController.delete);
```

5. Create Controller and Router for Student Model

Type this command to create a controller and router file for Student model.

```
touch controllers/student.js
```

Open and edit `controllers/student.js` then add this lines of codes that contains full CRUD function for the Student model.

```
const Student = require('../models').Student;
const Classroom = require('../models').Classroom;
const Course = require('../models').Course;

module.exports = {
  list(req, res) {
    return Student
      .findAll({
        include: [{
          model: Classroom,
          as: 'classroom'
        },{
          model: Course,
          as: 'courses'
        }],
        order: [
          ['createdAt', 'DESC'],
          [{ model: Course, as: 'courses' }, 'createdAt', 'DESC'],
        ],
      })
      .then((students) => res.status(200).send(students))
      .catch((error) => { res.status(400).send(error); });
  },

  getById(req, res) {
    return Student
      .findById(req.params.id, {
        include: [{
          model: Classroom,
          as: 'classroom'
        },{
          model: Course,
          as: 'courses'
        }],
      })
      .then((student) => {
        if (!student) {
          return res.status(404).send({
            message: 'Student Not Found',
          });
        }
        return res.status(200).send(student);
      })
      .catch((error) => res.status(400).send(error));
  },

  add(req, res) {
    return Student
      .create({
        classroom_id: req.body.classroom_id,
        student_name: req.body.student_name,
      })
      .then((student) => res.status(201).send(student))
      .catch((error) => res.status(400).send(error));
  },

  update(req, res) {
    return Student
      .findById(req.params.id, {
        include: [{
          model: Classroom,
          as: 'classroom'
        },{
          model: Course,
          as: 'courses'
        }],
      })
      .then(student => {
        if (!student) {
          return res.status(404).send({
            message: 'Student Not Found',
          });
        }
        return student
          .update({
            student_name: req.body.student_name || classroom.student_name,
          })
          .then(() => res.status(200).send(student))
          .catch((error) => res.status(400).send(error));
      })
      .catch((error) => res.status(400).send(error));
  },

  delete(req, res) {
    return Student
      .findById(req.params.id)
      .then(student => {
        if (!student) {
          return res.status(400).send({
            message: 'Student Not Found',
          });
        }
        return student
          .destroy()
          .then(() => res.status(204).send())
          .catch((error) => res.status(400).send(error));
      })
      .catch((error) => res.status(400).send(error));
  },
};
```

Next, open and edit `controllers/index.js` then register Student controller in that file.

```
const classroom = require('./classroom');
const student = require('./student');

module.exports = {
  classroom,
  student,
};
```

Next, open and edit `routes/index.js` then add a required variable for student controller.

```
const studentController = require('../controllers').student;
```

Add the routes for all CRUD function of student controller.

```
router.get('/api/student', studentController.list);
router.get('/api/student/:id', studentController.getById);
router.post('/api/student', studentController.add);
router.put('/api/student/:id', studentController.update);
router.delete('/api/student/:id', studentController.delete);
```

6. Create Controller and Router for Lecturer Model

Type this command to create a controller and router file for Lecturer model.

```
touch controllers/lecturer.js
```

Open and edit `controllers/lecturer.js` then add this lines of codes that contains full CRUD function for Lecturer model.

```
const Lecturer = require('../models').Lecturer;
const Course = require('../models').Course;

module.exports = {
  list(req, res) {
    return Lecturer
      .findAll({
        include: [{
          model: Course,
          as: 'course'
        }],
        order: [
          ['createdAt', 'DESC'],
          [{ model: Course, as: 'course' }, 'createdAt', 'DESC'],
        ],
      })
      .then((lecturers) => res.status(200).send(lecturers))
      .catch((error) => { res.status(400).send(error); });
  },

  getById(req, res) {
    return Lecturer
      .findById(req.params.id, {
        include: [{
          model: Course,
          as: 'course'
        }],
      })
      .then((lecturer) => {
        if (!lecturer) {
          return res.status(404).send({
            message: 'Lecturer Not Found',
          });
        }
        return res.status(200).send(lecturer);
      })
      .catch((error) => res.status(400).send(error));
  },

  add(req, res) {
    return Lecturer
      .create({
        lecturer_name: req.body.lecturer_name,
      })
      .then((lecturer) => res.status(201).send(lecturer))
      .catch((error) => res.status(400).send(error));
  },

  update(req, res) {
    return Lecturer
      .findById(req.params.id, {
        include: [{
          model: Course,
          as: 'course'
        }],
      })
      .then(lecturer => {
        if (!lecturer) {
          return res.status(404).send({
            message: 'Lecturer Not Found',
          });
        }
        return lecturer
          .update({
            lecturer_name: req.body.lecturer_name || classroom.lecturer_name,
          })
          .then(() => res.status(200).send(lecturer))
          .catch((error) => res.status(400).send(error));
      })
      .catch((error) => res.status(400).send(error));
  },

  delete(req, res) {
    return Lecturer
      .findById(req.params.id)
      .then(lecturer => {
        if (!lecturer) {
          return res.status(400).send({
            message: 'Lecturer Not Found',
          });
        }
        return lecturer
          .destroy()
          .then(() => res.status(204).send())
          .catch((error) => res.status(400).send(error));
      })
      .catch((error) => res.status(400).send(error));
  },
};
```

Next, open and edit `controllers/index.js` then register Lecturer controller in that file.

```
const classroom = require('./classroom');
const student = require('./student');
const lecturer = require('./lecturer');

module.exports = {
  classroom,
  student,
  lecturer,
};
```

Next, open and edit `routes/index.js` then add a required variable for student controller.

```
const lecturerController = require('../controllers').lecturer;
```

Add the routes for all CRUD function of lecturer controller.

```
router.get('/api/lecturer', lecturerController.list);
router.get('/api/lecturer/:id', lecturerController.getById);
router.post('/api/lecturer', lecturerController.add);
router.put('/api/lecturer/:id', lecturerController.update);
router.delete('/api/lecturer/:id', lecturerController.delete);
```

7. Create Controller and Router for Course Model

Type this command to create a controller and router file for Course model.

```
touch controllers/course.js
```

Open and edit `controllers/course.js` then add this lines of codes that contains full CRUD function for Course model.

```
const Course = require('../models').Course;
const Student = require('../models').Student;
const Lecturer = require('../models').Lecturer;

module.exports = {
  list(req, res) {
    return Course
      .findAll({
        include: [{
          model: Student,
          as: 'students'
        },{
          model: Lecturer,
          as: 'lecturer'
        }],
        order: [
          ['createdAt', 'DESC'],
          [{ model: Student, as: 'students' }, 'createdAt', 'DESC'],
        ],
      })
      .then((courses) => res.status(200).send(courses))
      .catch((error) => { res.status(400).send(error); });
  },

  getById(req, res) {
    return Course
      .findById(req.params.id, {
        include: [{
          model: Course,
          as: 'course'
        }],
      })
      .then((course) => {
        if (!course) {
          return res.status(404).send({
            message: 'Course Not Found',
          });
        }
        return res.status(200).send(course);
      })
      .catch((error) => res.status(400).send(error));
  },

  add(req, res) {
    return Course
      .create({
        course_name: req.body.course_name,
      })
      .then((course) => res.status(201).send(course))
      .catch((error) => res.status(400).send(error));
  },

  update(req, res) {
    return Course
      .findById(req.params.id, {
        include: [{
          model: Course,
          as: 'course'
        }],
      })
      .then(course => {
        if (!course) {
          return res.status(404).send({
            message: 'Course Not Found',
          });
        }
        return course
          .update({
            course_name: req.body.course_name || classroom.course_name,
          })
          .then(() => res.status(200).send(course))
          .catch((error) => res.status(400).send(error));
      })
      .catch((error) => res.status(400).send(error));
  },

  delete(req, res) {
    return Course
      .findById(req.params.id)
      .then(course => {
        if (!course) {
          return res.status(400).send({
            message: 'Course Not Found',
          });
        }
        return course
          .destroy()
          .then(() => res.status(204).send())
          .catch((error) => res.status(400).send(error));
      })
      .catch((error) => res.status(400).send(error));
  },
};
```

Next, open and edit `controllers/index.js` then register Lecturer controller in that file.

```
const classroom = require('./classroom');
const student = require('./student');
const lecturer = require('./lecturer');
const course = require('./course');

module.exports = {
  classroom,
  student,
  lecturer,
  course,
};
```

Next, open and edit `routes/index.js` then add a required variable for course controller.

```
const courseController = require('../controllers').course;
```

Add the routes for all CRUD function of lecturer controller.

```
router.get('/api/course', courseController.list);
router.get('/api/course/:id', courseController.getById);
router.post('/api/course', courseController.add);
router.put('/api/course/:id', courseController.update);
router.delete('/api/course/:id', courseController.delete);
```

8. Advance Route and Function for Association

Now, we have to make the association more useful. To make a Classroom include the students, add this function to `controllers/classroom.js`.

```
addWithStudents(req, res) {
  return Classroom
    .create({
      class_name: req.body.class_name,
      students: req.body.students,
    }, {
      include: [{
        model: Student,
        as: 'students'
      }]
    })
    .then((classroom) => res.status(201).send(classroom))
    .catch((error) => res.status(400).send(error));
},
```

Next, add this new function to the route file `routes/index.js`.

```
router.post('/api/classroom/add_with_students', classroomController.addWithStudents);
```

To add a lecturer include a course, add this function to `controllers/lecturer.js`.

```
addWithCourse(req, res) {
  return Lecturer
    .create({
      lecturer_name: req.body.lecturer_name,
      course: req.body.course
    }, {
      include: [{
        model: Course,
        as: 'course'
      }]
    })
    .then((lecturer) => res.status(201).send(lecturer))
    .catch((error) => res.status(400).send(error));
},
```

Next, add this new function to the route file `routes/index.js`.

```
router.post('/api/lecturer/add_with_course', lecturerController.addWithCourse);
```

To add course for student, add this function to `controllers/student.js`.

```
addCourse(req, res) {
  return Student
    .findById(req.body.student_id, {
      include: [{
        model: Classroom,
        as: 'classroom'
      },{
        model: Course,
        as: 'courses'
      }],
    })
    .then((student) => {
      if (!student) {
        return res.status(404).send({
          message: 'Student Not Found',
        });
      }
      Course.findById(req.body.course_id).then((course) => {
        if (!course) {
          return res.status(404).send({
            message: 'Course Not Found',
          });
        }
        student.addCourse(course);
        return res.status(200).send(student);
      })
    })
    .catch((error) => res.status(400).send(error));
},
```

Next, add this new function to the route file `routes/index.js`.

```
router.post('/api/student/add_course', studentController.addCourse);
```

That's a few of Association feature that might be useful for your project. We will add another useful function to this article later.

9. Run and Test The RESTful API

Type this command for run the application.

```
nodemon
```

Open the new terminal tab or command line tab then type this command for save or persist classroom data include with students.

```
curl -i -X POST -H "Content-Type: application/json" -d '{ "class_name":"Class A","students": [{ "student_name":"John Doe" },{ "student_name":"Jane Doe" },{ "student_name":"Doe Doel" }] }' localhost:3000/api/classroom/add_with_s
tudents
```

To see data persist to PostgreSQL table, open new terminal tab then run `psql`.

```
psql postgres -U djamware
```

Connect to the database then running the queries.

```
postgres=> \c node_sequelize
node_sequelize=> SELECT * FROM public."Classrooms";

 id | class_name |         createdAt          |         updatedAt
----+------------+----------------------------+----------------------------
  2 | Class A    | 2018-07-24 09:18:30.062+07 | 2018-07-24 09:18:30.062+07
(1 row)

node_sequelize=> SELECT * FROM public."Students" WHERE classroom_id=2;

id | classroom_id | student_name |         createdAt          |         updatedAt
----+--------------+--------------+----------------------------+----------------------------
 1 |            2 | John Doe     | 2018-07-24 09:18:30.125+07 | 2018-07-24 09:18:30.125+07
 2 |            2 | Jane Doe     | 2018-07-24 09:18:30.125+07 | 2018-07-24 09:18:30.125+07
 3 |            2 | Doe Doel     | 2018-07-24 09:18:30.125+07 | 2018-07-24 09:18:30.125+07
(3 rows)
```

Using `curl` you just get a classroom then the students will included with the response.

```
curl -i -H "Accept: application/json" localhost:3000/api/classroom/2

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 512
ETag: W/"200-9RPafOJtDdkqqMBVkSNCFoQ3p9s"
Date: Tue, 24 Jul 2018 03:18:45 GMT
Connection: keep-alive

{"id":2,"class_name":"Class A","createdAt":"2018-07-24T02:18:30.062Z","updatedAt":"2018-07-24T02:18:30.062Z","students":[{"id":1,"classroom_id":2,"student_name":"John Doe","createdAt":"2018-07-24T02:18:30.125Z","updatedAt":"201
8-07-24T02:18:30.125Z"},{"id":2,"classroom_id":2,"student_name":"Jane Doe","createdAt":"2018-07-24T02:18:30.125Z","updatedAt":"2018-07-24T02:18:30.125Z"},{"id":3,"classroom_id":2,"student_name":"Doe Doel","createdAt":"2018-07-2
4T02:18:30.125Z","updatedAt":"2018-07-24T02:18:30.125Z"}]}
```

Run this `curl` for save or persist Lecturer, Course and Student/Course data.

```
curl -i -X POST -H "Content-Type: application/json" -d '{ "lecturer_name":"Kylian Mbappe","course": { "course_name":"English Grammar" }}' localhost:3000/api/lecturer/add_with_course
curl -i -X POST -H "Content-Type: application/json" -d '{ "student_id":1,"course_id": 1}' localhost:3000/api/student/add_course
```

Now, you can see the data exists using `psql` query for each table.

That's it, the Node.js, Express.js, Sequelize.js and PostgreSQL RESTful API. You can get the full working source code on our GitHub (https://github.com/didinj/node-express-postgresql-sequelize.git).

That just the basic. If you need more deep learning about MEAN Stack, Angular, and Node.js, you can find the following books:

For more detailed on MEAN stack and Node.js, you can take the following course:

Thanks!

🡰 Previous Article

MEAN Stack Angular 6 CRUD Web Application

(/post/5b00bb9180aca726dee1fd6d/mean-stack-angular-6-crud-web-application)

Next Article 🡲

MongoDB, Express, Vue.js 2, Node.js (MEVN) and SocketIO Chat App

(/post/5b6a681f80aca76a2cbd98fb/mongodb-express-vuejs-2-nodejs-mevn-and-socketio-chat-app)

Related Articles

- MongoDB, Express, Vue.js 2, Node.js (MEVN) and SocketIO Chat App (/post/5b6a681f80aca76a2cbd98fb/mongodb-express-vuejs-2-nodejs-mevn-and-socketio-chat-app)
- MEAN Stack Angular 6 CRUD Web Application (/post/5b00bb9180aca726dee1fd6d/mean-stack-angular-6-crud-web-application)
- Angular 6 Tutorial: Getting Started Build Angular 6 Web Application (/post/5afa31a780aca726dee1fd6c/angular-6-tutorial-getting-started-build-angular-6-web-application)
- Securing MEVN Stack (Vue.js 2) Web Application using Passport (/post/5ac8338780aca714d19d5b9e/securing-mevn-stack-vuejs-2-web-application-using-passport)
- Securing MERN Stack Web Application using Passport (/post/5a90c37980aca7059c14297a/securing-mern-stack-web-application-using-passport)
- Securing MEAN Stack (Angular 5) Web Application using Passport (/post/5a878b3c80aca7059c142979/securing-mean-stack-angular-5-web-application-using-passport)
- Setup Node.js, Nginx and MongoDB on Ubuntu 16.04 for Production (/post/5a593bfc80aca7059c142975/setup-nodejs-nginx-and-mongodb-on-ubuntu-1604-for-production)
- Node.js and MongoDB Slack Bot Example (/post/5a500c9380aca7059c142973/nodejs-and-mongodb-slack-bot-example)
- Mongo Express Vue Node.js (MEVN Stack) CRUD Web Application (/post/5a1b779f80aca75eadc12d6e/mongo-express-vue-nodejs-mevn-stack-crud-web-application)
- MEAN Stack (Angular 5) CRUD Web Application Example (/post/5a0673c880aca7739224ee21/mean-stack-angular-5-crud-web-application-example)
- Building CRUD Web Application using MERN Stack (/post/59faec0a80aca7739224ee1f/building-crud-web-application-using-mern-stack)
- Node Express Passport Facebook Twitter Google GitHub Login (/post/59a6257180aca768e4d2b132/node-express-passport-facebook-twitter-google-github-login)

1 Comment Djamware - Fullstack Programming Tutorials 1 Login

Recommend 2 Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

APOORV BANKEY
18 days ago
After running the command which includes classroom data with students, i got an error saying students is not associated with classroom.
Reply

ALSO ON DJAMWARE - FULLSTACK PROGRAMMING TUTORIALS

Building Spring Boot, MongoDB and React.js CRUD Web
4 comments · 5 months ago
Bathula Nagendra — Sir I am getting the below error.please help me to resolve this issue. It is

Spring Boot, Security, and Data MongoDB Authentication Example
1 comment · 2 months ago
keles0glu — Why you use the 'set' data structure for hold Role?

Build Ionic 3 Angular 5 Calendar UI with Event Integration
25 comments · 9 months ago
chica85 — los puntitos de los eventos no se ven tras seguir los pasos y verificarlo en github,

Getting Started Build Angular 6 Web Application
5 comments · 3 months ago
xzatech — You may generate the Navigation Component using this command.

Subscribe Add Disqus to your siteAdd DisqusAdd DISQUS
Disqus' Privacy PolicyPrivacy PolicyPrivacy

Programming Blog
(/post-category/595f867edbd39e7571e183dc/programming-blog)
Javascript
(/post-sub-category/583d6d30fcbe614473a4c4e9/javascript)
MongoDB
(/post-sub-category/5845677b80aca7763489d871/mongodb)
Groovy and Grails
(/post-sub-category/585b3fa380aca73b19a2efd4/groovy-and-grails)
Node.js
(/post-sub-category/58a9196f80aca748640ce352/nodejs)
HTML 5 Tutorial
(/post-sub-category/584209dffcbe618f680bdc5c/html-5-tutorial)
React Native
(/post-sub-category/5b4aa0b480aca707dd4f65a6/react-native)
Ionic Framework
(/post-sub-category/5845691a80aca7763489d872/ionic-framework)
CSS 3
(/post-sub-category/584249bde4d5d303658d1ecf/css-3)
Java
(/post-sub-category/583d6c37fcbe614473a4c4e8/java)
All Articles
(/public/allArticles)

Popular Articles:
- MEAN Stack (Angular 5) CRUD Web Application Example (/post/5a0673c880aca7739224ee21/mean-stack-angular-5-crud-web-application-example)
- MEAN Stack Angular 6 CRUD Web Application (/post/5b00bb9180aca726dee1fd6d/mean-stack-angular-6-crud-web-application)
- Ionic 3 Consuming REST API using New Angular 4.3 HttpClient (/post/59924f9080aca768e4d2b12e/ionic-3-consuming-rest-api-using-new-angular-43-httpclient)
- Node.js, Express.js, Sequelize.js and PostgreSQL RESTful API (/post/5b56a6cc80aca707dd4f65a9/nodejs-expressjs-sequelizejs-and-postgresql-restful-api)
- Ionic 3, Angular 4 and SQLite CRUD Offline Mobile App (/post/59c53a1280aca768e4d2b143/ionic-3-angular-4-and-sqlite-crud-offline-mobile-app)
- Ionic 3 and Angular 4 Mobile App Example (/post/58e657b680aca764ec903c2d/ionic-3-and-angular-4-mobile-app-example)
- How to Upload File on Ionic 3 using Native File Transfer Plugin (/post/599da16580aca768e4d2b130/how-to-upload-file-on-ionic-3-using-native-file-transfer-plugin)
- Ionic 3 and Angular 5 Mobile App Example (/post/59fc9da680aca7739224ee20/ionic-3-and-angular-5-mobile-app-example)
- Build Ionic 3, Angular 5 and Firebase Simple Chat App (/post/5a629d9880aca7059c142976/build-ionic-3-angular-5-and-firebase-simple-chat-app)
- Angular 6 Tutorial: Getting Started Build Angular 6 Web Application (/post/5afa31a780aca726dee1fd6c/angular-6-tutorial-getting-started-build-angular-6-web-application)