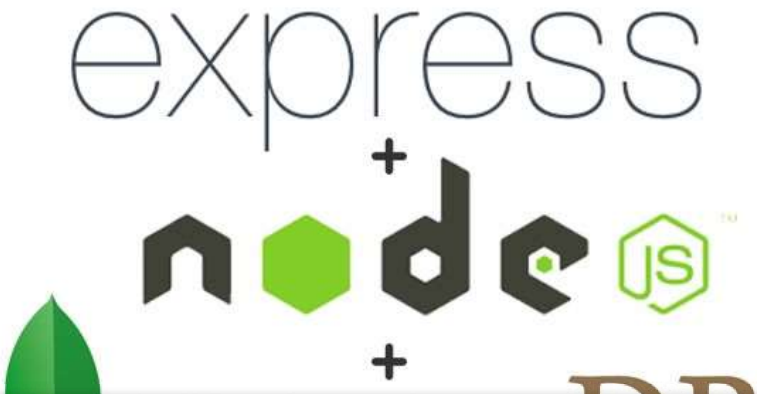
CRUD REST API using Node JS, Express and MongoDB

author: Ramesh Fadatare

mongodb nodejs restful api

In this tutorial, we'll learn how to develop a RESTful CRUD (Create, Retrieve, Update, Delete) API with **Node.js**, **Express**, and **MongoDB**. We'll use **Mongoose** for interacting with the MongoDB instance.



Before getting started, let's quickly see what is Express and Mongoose.

Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node-based Web applications.

> Read more about express js at **https://expressjs.com**.

MongoDB and Mongoose

MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need.

Mongoose is an ODM (Object Document Mapping) tool for Node.js and MongoDB. It helps you convert the objects in your code to documents in the database and vice versa.

Setup MongoDB

To work with MongoDB, you need to have MongoDB installed in your system. Check out the official **MongoDB doc** for installing MongoDB in your System.

You can also install the Zip version of MongoDB on Windows, check out this article at **Install MongoDB on Windows 10**.

Let's list the tools and technologies that we will use in this tutorial.

Tools and technologies used

1. node.js (npm)
2. express
3. body-parser
4. mongoose

3. Setting up the webserver
4. Configuring and Connecting to the database
5. Defining the Todo model in Mongoose
6. Defining Routes using Express
7. Developing the Restful APIs
8. Testing our APIs
9. Conclusion
10. Source code on GitHub repository

1. Creating an application

1. Open a terminal and create a new folder for the application.

```
$ mkdir todo-app
```

2. Initialize the application with a `package.json` file

Go to the root folder of your application and type `npm init` to initialize your app with a `package.json` file.

```
$ cd todo-app
$ npm init
```

Here is the complete `package.json` file:

```
{
  "name": "todo-app",
  "version": "1.0.0",
  "description": "Todo App",
  "main": "main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "Express",
    "RestAPI",
```

Note that I've specified a file named `main.js` as the entry point of our application. We'll create a `main.js` file in the next section.

2. Install dependencies

We will need `express`, `mongoose`, and `body-parser` modules in our application. Let's install them by typing the following command -

```
$ npm install express body-parser mongoose --save
```

We have used the `--save` option to save all the dependencies in the `package.json` file. The final `package.json` file looks like this -

```
{
  "name": "todo-app",
  "version": "1.0.0",
  "description": "Todo App",
  "main": "main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "Express",
    "RestAPI",
    "MongoDB",
    "Mongoose",
    "Todos"
  ],
  "author": "javaguides.net",
  "license": "MIT",
  "dependencies": {
    "body-parser": "^1.19.0",
    "express": "^4.17.1",
    "mongoose": "^5.9.2"
  }
}
```

3. Setting up the webserver

```
const express = require('express');
const bodyParser = require('body-parser');

// create express app
const app = express();

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

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

// define a simple route
app.get('/', (req, res) => {
    res.json({"message": "Welcome to Todo app"});
});

// listen for requests
app.listen(4000, () => {
    console.log("Server is listening on port 4000");
});
```

Let's now run the server and go to **http://localhost:4000** to access the route we just defined.

```
$ npm install
$ node main.js

Server is listening on port 4000
```

4. Configuring and Connecting to the database

Let's create a **config** folder with the following command:

```
$ mkdir config
$ cd config
```

Let's create `database.config.js` inside the **config** folder with the following

We'll now import the above database configuration in `main.js` and connect to the database using **mongoose**.

Add the following code to the `main.js` file after the `app.use(bodyParser.json())` line -

```
// Configuring the database
const dbConfig = require('./config/database.config.js');
const mongoose = require('mongoose');

mongoose.Promise = global.Promise;

// Connecting to the database
mongoose.connect(dbConfig.url, {
 useNewUrlParser: true
}).then(() => {
    console.log("Successfully connected to the database");
}).catch(err => {
    console.log('Could not connect to the database. Exiting now...', e
    process.exit();
});
```

The complete `main.js` file looks like:

```
const express = require('express');
const bodyParser = require('body-parser');

// create express app
const app = express();

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

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

// Configuring the database
const dbConfig = require('./config/database.config.js');
const mongoose = require('mongoose');
```

```
    console.log('Could not connect to the database. Exiting now...', e
    process.exit();
});

// define a simple route
app.get('/', (req, res) => {
    res.json({"message": "Welcome to Todo app"});
});

require('./app/routes/todo.routes.js')(app);

// listen for requests
app.listen(4000, () => {
    console.log("Server is listening on port 4000");
});
```

Let's fire the below command to make sure that you're able to connect to the database:

```
$ node main.js
Server is listening on port 4000
Successfully connected to the database
```

5. Defining the Todo model in Mongoose

Let's create an **app/models** folder inside the root folder of the application with the following command:

```
$ mkdir -p app/models
$ cd app/models
```

Now, create a file called `todo.model.js` inside the **app/models** folder with the following contents:

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

```
module.exports = mongoose.model('Todo', TodoSchema);
```

The **Todo** model is very simple. It contains a **name** and a **description** field. I have also added a **timestamps** option to the schema.

Mongoose uses this option to automatically add two new fields - **createdAt** and **updatedAt** to the schema.

6. Defining Routes using Express

Create a new folder called **routes** inside the **app** folder with the following command:

```
$ mkdir app/routes
$ cd app/routes
```

Now, create a new file called `todo.routes.js` inside the **app/routes** folder with the following contents:

```
module.exports = (app) => {
    const todos = require('../controllers/todo.controller.js');

    // Create a new todo
    app.post('/todos', todos.create);

    // Retrieve all todos
    app.get('/todos', todos.findAll);

    // Retrieve a single todo by id
    app.get('/todos/:id', todos.findOne);

    // Update a Todo with id
    app.put('/todos/:id', todos.update);

    // Delete a Todo by id
    app.delete('/todos/:id', todos.delete);
}
```

file. We'll define the controller file in the next section. The controller will contain methods for handling all the CRUD operations.

7. Developing the Restful APIs

Create a new folder called **controllers** inside the **app** folder, then create a new file called `todo.controller.js` inside the **app/controllers** folder.

Create a Todo

```
// Create and Save a new Todo
exports.create = (req, res) => {
    // Validate request
    if(!req.body.description) {
        return res.status(400).send({
            message: "Todo description can not be empty"
        });
    }

    // Create a Todo
    const todo = new Todo({
        name: req.body.name || "Untitled Todo",
        description: req.body.description
    });

    // Save Todo in the database
    todo.save()
    .then(data => {
        res.send(data);
    }).catch(err => {
        res.status(500).send({
            message: err.message || "Some error occurred while creatir
        });
    });
};
```

Retrieve all Todos

```
// Retrieve and return all todos from the database.
```

```
            message: err.message || "Some error occurred while retrie
        });
    });
};
```

Retrieve a single Todo by Id

```
// Find a single todo with a id
exports.findOne = (req, res) => {
    Todo.findById(req.params.id)
    .then(todo => {
        if(!todo) {
            return res.status(404).send({
                message: "Todo not found with id " + req.params.id
            });
        }
        res.send(todo);
    }).catch(err => {
        if(err.kind === 'ObjectId') {
            return res.status(404).send({
                message: "Todo not found with id " + req.params.id
            });
        }
        return res.status(500).send({
            message: "Error retrieving todo with id " + req.params.id
        });
    });
};
```

Update a Todo

```
// Update a todo identified by the id in the request
exports.update = (req, res) => {
    // Validate Request
    if(!req.body.description) {
        return res.status(400).send({
            message: "Todo description can not be empty"
        });
    }
```

```
            return res.status(404).send({
                message: "Todo not found with id " + req.params.id
            });
        }
        res.send(todo);
    }).catch(err => {
        if(err.kind === 'ObjectId') {
            return res.status(404).send({
                message: "Todo not found with id " + req.params.id
            });
        }
        return res.status(500).send({
            message: "Error updating todo with id " + req.params.id
        });
    });
};
```

Delete a Todo

```
// Delete a todo with the specified id in the request
exports.delete = (req, res) => {
    Todo.findByIdAndRemove(req.params.id)
    .then(todo => {
        if(!todo) {
            return res.status(404).send({
                message: "Todo not found with id " + req.params.id
            });
        }
        res.send({message: "Todo deleted successfully!"});
    }).catch(err => {
        if(err.kind === 'ObjectId' || err.name === 'NotFound') {
            return res.status(404).send({
                message: "Todo not found with id " + req.params.todo
            });
        }
        return res.status(500).send({
            message: "Could not delete todo with id " + req.params.id
        });
    });
};
```

```
// Create and Save a new Todo
exports.create = (req, res) => {
    // Validate request
    if(!req.body.description) {
        return res.status(400).send({
            message: "Todo description can not be empty"
        });
    }

    // Create a Todo
    const todo = new Todo({
        name: req.body.name || "Untitled Todo",
        description: req.body.description
    });

    // Save Todo in the database
    todo.save()
    .then(data => {
        res.send(data);
    }).catch(err => {
        res.status(500).send({
            message: err.message || "Some error occurred while creatir
        });
    });
};

// Retrieve and return all todos from the database.
exports.findAll = (req, res) => {
    Todo.find()
    .then(todos => {
        res.send(todos);
    }).catch(err => {
        res.status(500).send({
            message: err.message || "Some error occurred while retriev
        });
    });
};

// Find a single todo with a id
exports.findOne = (req, res) => {
    Todo.findById(req.params.id)
    .then(todo => {
        if(!todo) {
```

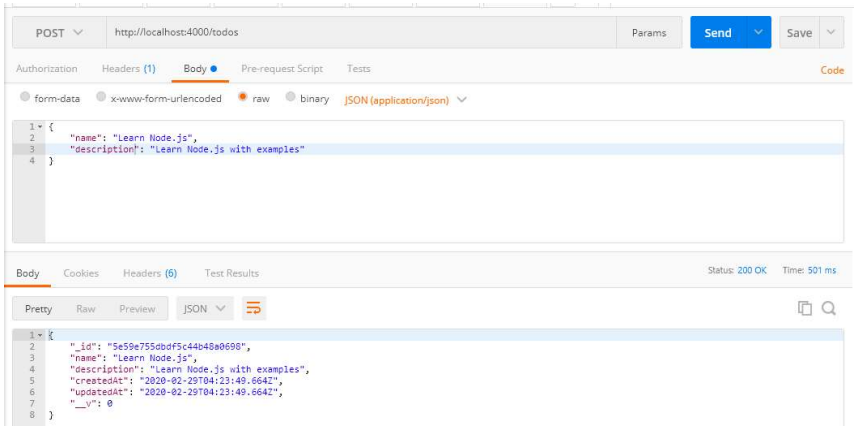
```
            if(err.kind === 'ObjectId') {
                return res.status(404).send({
                    message: "Todo not found with id " + req.params.id
                });
            }
            return res.status(500).send({
                message: "Error retrieving todo with id " + req.params.id
            });
        });
};

// Update a todo identified by the id in the request
exports.update = (req, res) => {
    // Validate Request
    if(!req.body.description) {
        return res.status(400).send({
            message: "Todo description can not be empty"
        });
    }

    // Find todo and update it with the request body
    Todo.findByIdAndUpdate(req.params.id, {
        title: req.body.name || "Untitled Todo",
        description: req.body.description
    }, {new: true})
    .then(todo => {
        if(!todo) {
            return res.status(404).send({
                message: "Todo not found with id " + req.params.id
            });
        }
        res.send(todo);
    }).catch(err => {
        if(err.kind === 'ObjectId') {
            return res.status(404).send({
                message: "Todo not found with id " + req.params.id
            });
        }
        return res.status(500).send({
            message: "Error updating todo with id " + req.params.id
        });
    });
};
```

```
            });
        }
        res.send({message: "Todo deleted successfully!"});
    }).catch(err => {
        if(err.kind === 'ObjectId' || err.name === 'NotFound') {
            return res.status(404).send({
                message: "Todo not found with id " + req.params.todo
            });
        }
        return res.status(500).send({
            message: "Could not delete todo with id " + req.params.id
        });
    });
};
```

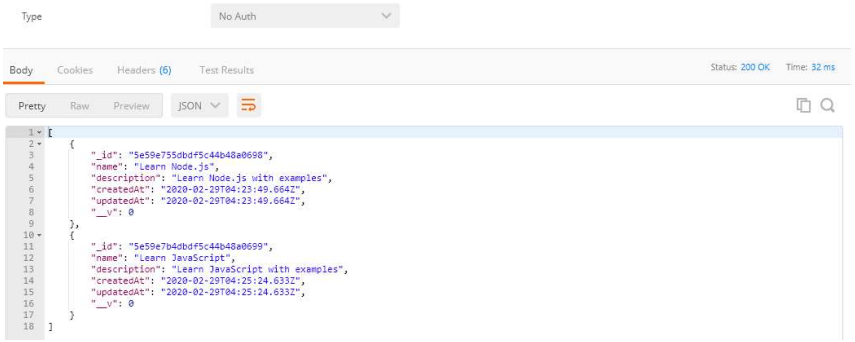
8. Testing our APIs

Let's now test all the APIs one by one using postman.

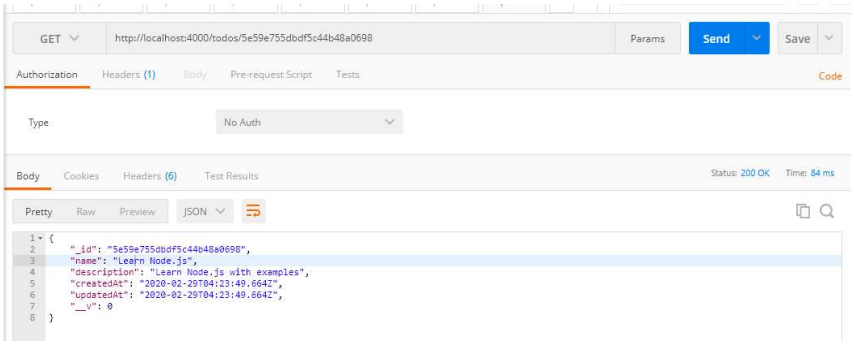
Create a new Todo using POST /todos API



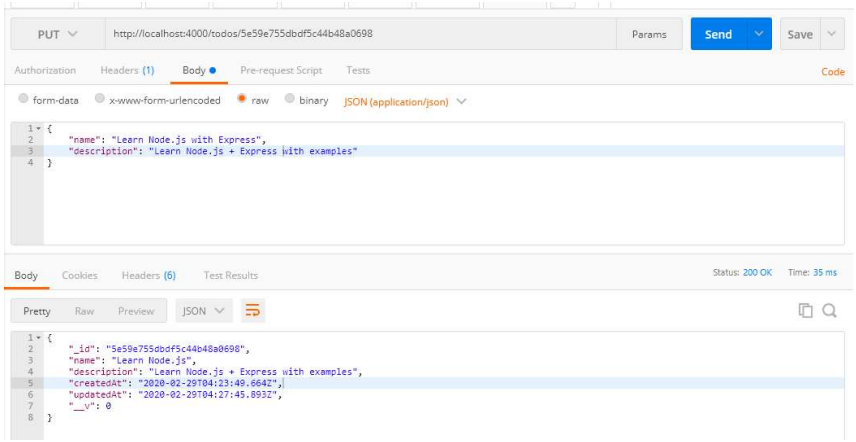
Retrieving all Todos using GET /todos API



Retrieving a single Todo using GET /todos/:id API



Updating a Todo using PUT /todos/:id API



Deleting a Todo using DELETE /todos/:id API

In this tutorial, we have developed a RESTful CRUD (Create, Retrieve, Update, Delete) API with **Node.js, Express**, and **MongoDB**. We have seen how to use Mongoose for interacting with the MongoDB instance.

> If you found this tutorial useful then follow me at **https://www.javaguides.net/p/about-me.html**.

10. Source code on GitHub repository

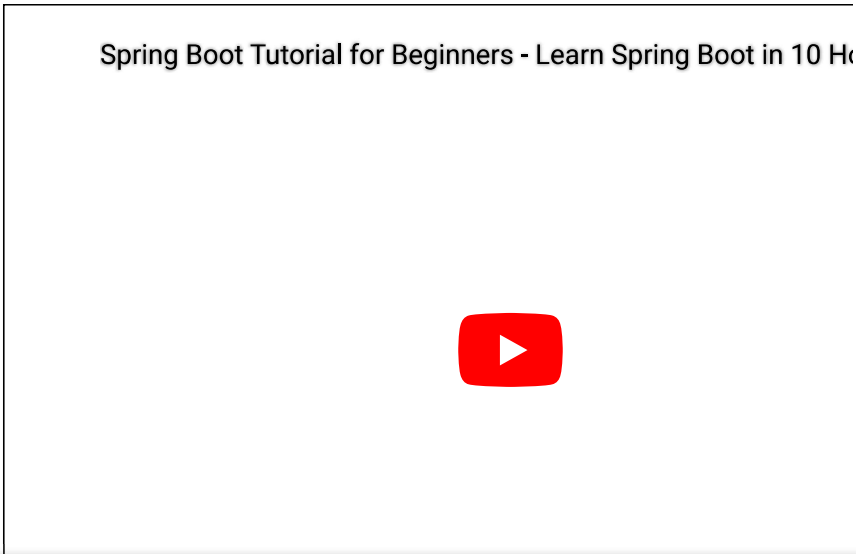
The source code of this tutorial is available on my GitHub repository at **https://github.com/RameshMF/node-todo-app**

mongodb nodejs restful api

Free Spring Boot Tutorial | Full In-depth Course | Learn Spring Boot in 10 Hours

---

Watch this course on YouTube at **Spring Boot Tutorial | Fee 10 Hours Full Course**



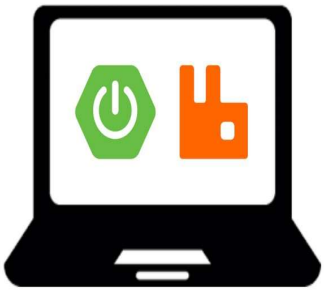
MONGODB NODEJS RESTFUL API

Spring Boot Thymeleaf Real-Time Web Application Course

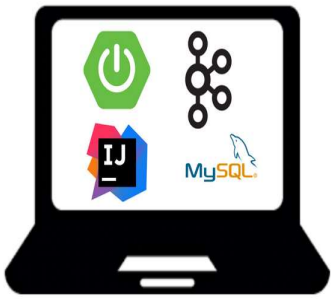
My Udemy Course - Testing Spring Boot Application with JUnit and Mockito

My Udemy Course - Building Real-Time REST APIs with Spring Boot

My Udemy Course - Spring Boot RabbitMQ Course - Event-Driven Microservices



My Udemy Course - Spring Boot + Apache Kafka Course

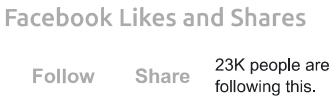


About Me

Hi, I am **Ramesh Fadatare**. I am VMWare Certified Professional for Spring and Spring Boot 2022.

I am founder and author of this blog website **JavaGuides**, a technical blog dedicated to the Java/Java EE technologies and Full-Stack Java development.

All the articles, guides, tutorials(2000 +) written by me so connect with me if you have any questions/queries. Read more about me at **About Me.**



Free Courses on YouTube

Spring Boot Tutorial
Java Collections Framework
Spring Boot AWS Deployment
Spring MVC Tutorial Course
5 Spring Boot Projects in 10 Hours
Spring Boot Restful Web Services Tutorial
Event-Driven Microservices using Spring Boot and Kafka
Spring Boot Kafka Real-World Project Tutorial
Spring Boot + Apache Kafka Course
Angular + Spring Boot CRUD Full Stack
ReactJS + Spring Boot CRUD Full Stack
Spring Boot Thymeleaf Full Stack

My Udemy Courses

Building Microservices with Spring Boot and Spring Cloud
Building Real-Time REST APIs with Spring Boot
Testing Spring Boot Application with JUnit and Mockito
Master Spring Data JPA with Hibernate
Spring Boot + Apache Kafka - The Quickstart Practical Guide
Spring Boot + RabbitMQ (Includes Event-Driven Microservices)
Spring Boot Thymeleaf Real-Time Web Application - Blog App

Connect

YouTube
Twitter
Facebook
GitHub
Linkedin
StackOverflow

Courses

Udemy Courses
YouTube Channel
Free Courses
Java API Guides
All 2K+ Java Tutorials
Udemy Profile