Node.js, Express & MongoDb: Build a CRUD Rest

You should install MongoDB in your machine first. The installation instructions can be found at Official MongoDB installation manual.

Create Node.js App

First, we create a folder:

```
$ mkdir nodejs-express-mongodb
$ cd nodejs-express-mongodb
```

Next, we initialize the Node.js App with a *package.json* file:

```
npm init

name: (nodejs-express-mongodb)
version: (1.0.0)
description: Node.js Restful CRUD API with Node.js, Express and MongoDB
entry point: (index.js) server.js
test command:
git repository:
keywords: nodejs, express, mongodb, rest, api
author:
license: (ISC)

Is this ok? (yes) yes
```

We need to install necessary modules: `express`, `mongoose` and `cors`.

Run the command:

```
npm install express mongoose cors --save
```

The *package.json* file should look like this:

```
{
  "name": "node-express-mongodb",
  "version": "1.0.0",
  "description": "Node.js Restful CRUD API with Node.js, Express and
MongoDB",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
```

```
    "nodejs",
    "express",
    "rest",
    "api",
    "mongodb"
  ],
  "author":            ,
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mongoose": "^5.8.10"
  }
}
```

Setup Express web server

In the root folder, let's create a new *server.js* file:

```
const express = require("express");
const cors = require("cors");

const app = express();

var corsOptions = {
  origin: "http://localhost:8081"
};

app.use(cors(corsOptions));

// parse requests of content-type - application/json
app.use(express.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));

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

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

What we do are:

– import `express` and `cors` modules:

- Express is for building the Rest apis
- cors provides Express middleware to enable CORS with various options.

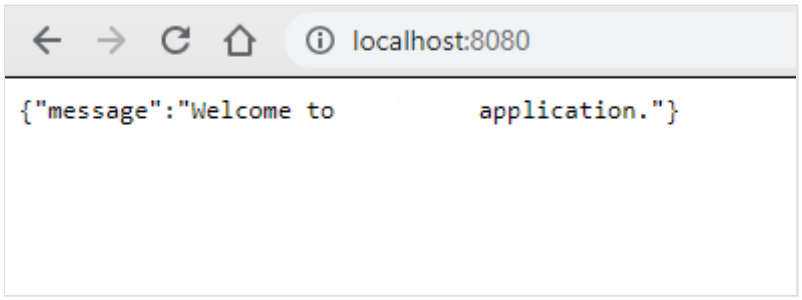
– create an Express app, then add body-parser (`json` and `urlencoded`) and `cors` middlewares using `app.use()` method. Notice that we set origin: `http://localhost:8081`.

– define a GET route which is simple for test.

– listen on port 8080 for incoming requests.

Now let's run the app with command: `node server.js`.

Open your browser with url http://localhost:8080/, you will see:



Yeah, the first step is done. We're gonna work with Mongoose in the next section.

Configure MongoDB database & Mongoose

In the *app* folder, we create a separate *config* folder for configuration with *db.config.js* file like this:

```
module.exports = {
  url: "mongodb://localhost:27017/db"
};
```

Define Mongoose

We're gonna define Mongoose model (*tutorial.model.js*) also in **app**/**models** folder in the next step.

Now create **app**/**models**/*index.js* with the following code:

```
const dbConfig = require("../config/db.config.js");

const mongoose = require("mongoose");
mongoose.Promise = global.Promise;

const db = {};
db.mongoose = mongoose;
```

```
db.url = dbConfig.url;
db.tutorials = require("./tutorial.model.js")(mongoose);

module.exports = db;
```

Don't forget to call `connect()` method in *server.js*:

```
...
const app = express();
app.use(...);

const db = require("./app/models");
db.mongoose
  .connect(db.url, {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => {
    console.log("Connected to the database!");
  })
  .catch(err => {
    console.log("Cannot connect to the database!", err);
    process.exit();
  });
```

Define the Mongoose Model

In *models* folder, create *tutorial.model.js* file like this:

```
module.exports = mongoose => {
  const Tutorial = mongoose.model(
    "tutorial",
    mongoose.Schema(
      {
        title: String,
        description: String,
        published: Boolean
      },
      { timestamps: true }
    )
  );

  return Tutorial;
};
```

This Mongoose Model represents **tutorials** collection in MongoDB database. These fields will be generated automatically for each Tutorial document: *_id*, *title*, *description*, *published*, *createdAt*, *updatedAt*, *__v*.

```
{
  "_id": "5e363b135036a835ac1a7da8",
  "title": "Js Tut#",
  "description": "Description for Tut#",
  "published": true,
  "createdAt": "2020-02-02T02:59:31.198Z",
  "updatedAt": "2020-02-02T02:59:31.198Z",
  "__v": 0
}
```

If you use this app with a front-end that needs *id* field instead of *_id*, you have to override `toJSON` method that map default object to a custom object. So the Mongoose model could be modified as following code:

```
module.exports = mongoose => {
  var schema = mongoose.Schema(
    {
      title: String,
      description: String,
      published: Boolean
    },
    { timestamps: true }
  );

  schema.method("toJSON", function() {
    const { __v, _id, ...object } = this.toObject();
    object.id = _id;
    return object;
  });

  const Tutorial = mongoose.model("tutorial", schema);
  return Tutorial;
};
```

And the result will look like this-

```
{
  "title": "Js Tut#",
  "description": "Description for Tut#",
  "published": true,
  "createdAt": "2020-02-02T02:59:31.198Z",
  "updatedAt": "2020-02-02T02:59:31.198Z",
  "id": "5e363b135036a835ac1a7da8"
}
```