Activity

**Aim:** Create Rest api & test rest api using postman

**Learning outcome**: Able to develop the real time scenarios based on Node JS applications.

**Duration**: 2 hours

## List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Linux OS - Ubuntu 18.04 LTS
2. NodeJS, ExpressJS, Joi, Nodemon, Postman software installed

## Program:

**Step 1:** Create a project directory, which will consist of all the files present in the project. Then, open commands prompt and navigate to the project directory. Refer below.



**Step 2:** Now, call npm using the below command. This will initialize the npm modules in your system.

**npm init**

Once you hit enter, Node.js, will ask you to enter a few details related to the project. These details will basically be the metadata for your project.

Here you can define your entry point along with several other information. For this demo, I will be using script.js as an entry point.

It will then, ask you for a confirmation for the data you must have mentioned. Just press on Y to confirm.

**Step 3:** Next, you have yo install Express.js using the below command:

**npm i express**

Express is a web framework which can be used along with Node.js. This web framework will allow you to create Restful APIs, with the help of helper methods, middle layers to configure your application.

**Step 3.1:** Similarly, you have to install Joi.

**npm i joi**

This package allows you to create blueprints for JavaScript objects which store information to ensure validation of key information.

**Step 3.2:** Finally, install the node monitoring package nodemon, using the below command.

**npm i -g nodemon**

Nodemon, keeps a watch on all the files with any type of extension present in this folder. Also, with nodemon on the watch, you don’t have to restart the Node.js server each time any changes are made. Nodemon will implicitly detect the changes and restart the server for you.

**package.json**

{

"name": "restapidemo",

"version": "1.0.0",

"description": "Creation of REST API",

"main": "script.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "sahiti\_kappagantula",

"license": "ISC",

"dependencies": {

"express": "^4.17.1",

"joi": "^14.3.1"

}

}

**script.js**

const express = require('express'); //Import Express

const Joi = require('joi'); //Import Joi

const app = express(); //Create Express Application on the app variable

app.use(express.json()); //used the json file

//Give data to the server

const customers = [

{title: 'George', id: 1},

{title: 'Josh', id: 2},

{title: 'Tyler', id: 3},

{title: 'Alice', id: 4},

{title: 'Candice', id: 5}

]

//Read Request Handlers

// Display the Message when the URL consist of '/'

app.get('/', (req, res) => {

res.send('Welcome to Edurekas REST API!');

});

// Display the List Of Customers when URL consists of api customers

app.get('/api/customers', (req,res)=> {

res.send(customers);

});

// Display the Information Of Specific Customer when you mention the id.

app.get('/api/customers/:id', (req, res) => {

const customer = customers.find(c => c.id === parseInt(req.params.id));

//If there is no valid customer ID, then display an error with the following message

if (!customer) res.status(404).send('<h2 style="font-family: Malgun Gothic; color: darkred;">Ooops... Cant find what you are looking for!</h2>');

res.send(customer);

});

//CREATE Request Handler

//CREATE New Customer Information

app.post('/api/customers', (req, res)=>; {

const { error } = validateCustomer(req.body);

if (error){

res.status(400).send(error.details[0].message)

return;

}

//Increment the customer id

const customer = {

id: customers.length + 1,

title: req.body.title

};

customers.push(customer);

res.send(customer);

});

//Update Request Handler

// Update Existing Customer Information

app.put('/api/customers/:id', (req, res) => {

const customer = customers.find(c=> c.id === parseInt(req.params.id));

if (!customer) res.status(404).send('<h2 style="font-family: Malgun Gothic; color: darkred;">Not Found!! </h2>');

const { error } = validateCustomer(req.body);

if (error){

res.status(400).send(error.details[0].message);

return;

}

customer.title = req.body.title;

res.send(customer);

});

//Delete Request Handler

// Delete Customer Details

app.delete('/api/customers/:id', (req, res) =>; {

const customer = customers.find( c=> c.id === parseInt(req.params.id));

if(!customer) res.status(404).send('<h2 style="font-family: Malgun Gothic; color: darkred;">Not Found!!</h2>'</span>);

const index = customers.indexOf(customer);

customers.splice(index,1);

res.send(customer);

});

//Validate Information

function validateCustomer(customer) {

const schema = {

title: Joi.string().min(3).required()

};

return Joi.validate(customer, schema);

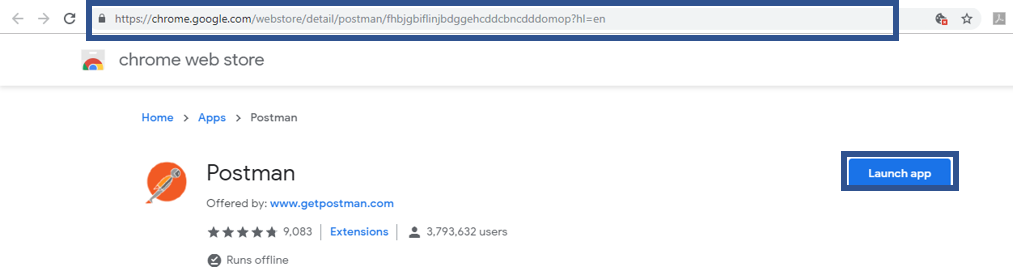
}

//PORT ENVIRONMENT VARIABLE

const port = process.env.PORT || 8080;

app.listen(port, () => console.log(`Listening on port ${port}..`));

**Step 4:** Now, the next step is to check whether the handlers are working properly or not. For that, we will use a Chrome extension called Postman. To install Postman you can search for postman in google web store and click on ‘Add to Chrome’.



**Step 5:** Now, once you have installed Postman, open it to test your application.

**Step 6:** But before that you have to start your server. To start your server, type the following command.

**node script.js**

You would see the output as below:

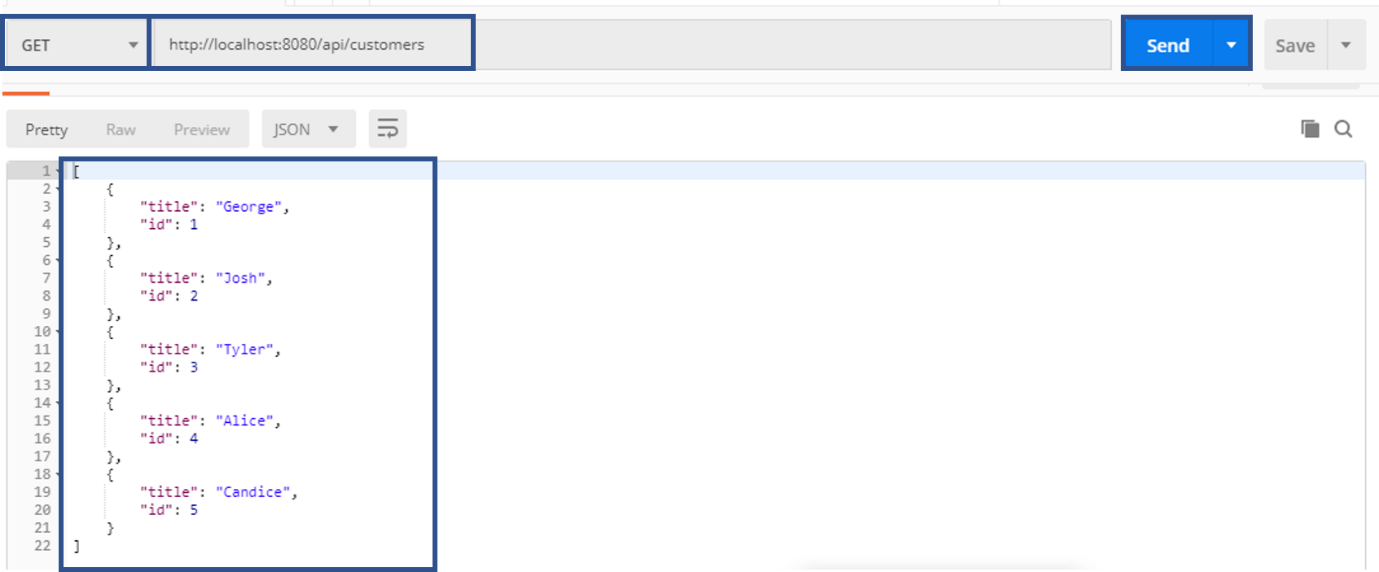


## Output/Results snippet:

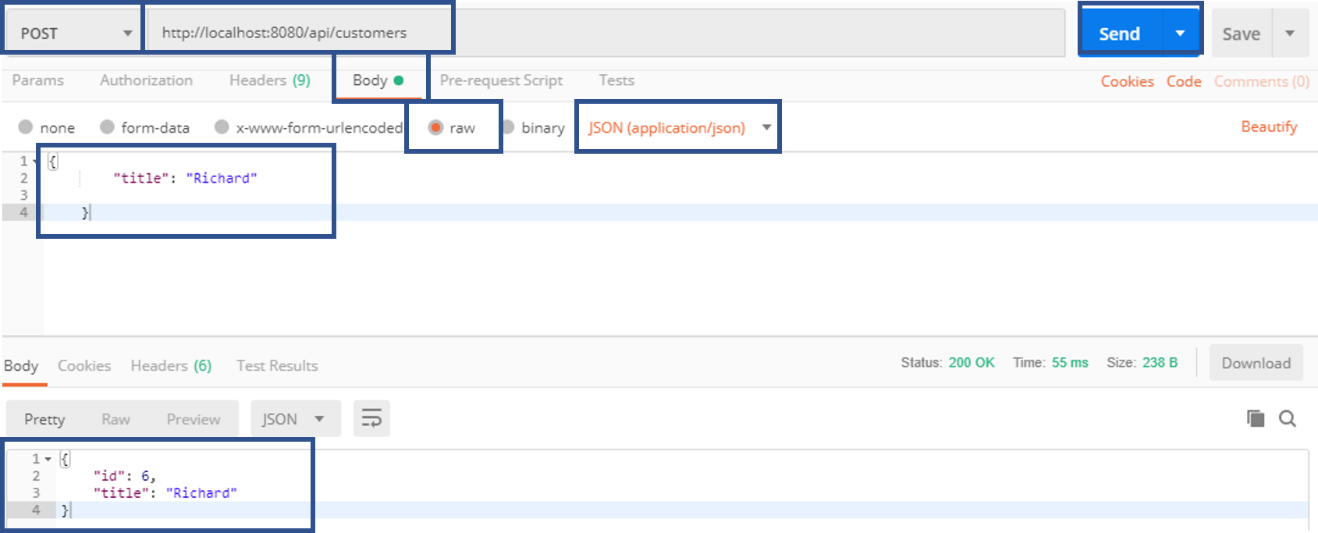
Let us start by testing the GET Method.

**Step 7:** In order to do that you need to select GET from the drop-down list, type in the defined URL and hit send.

If your code is working fine, then you will see the list of all the customers which we have added manually in our code. In the below picture, you can see how my result looks like. Here I have mentioned the URL to be localhost:8080/api/customers



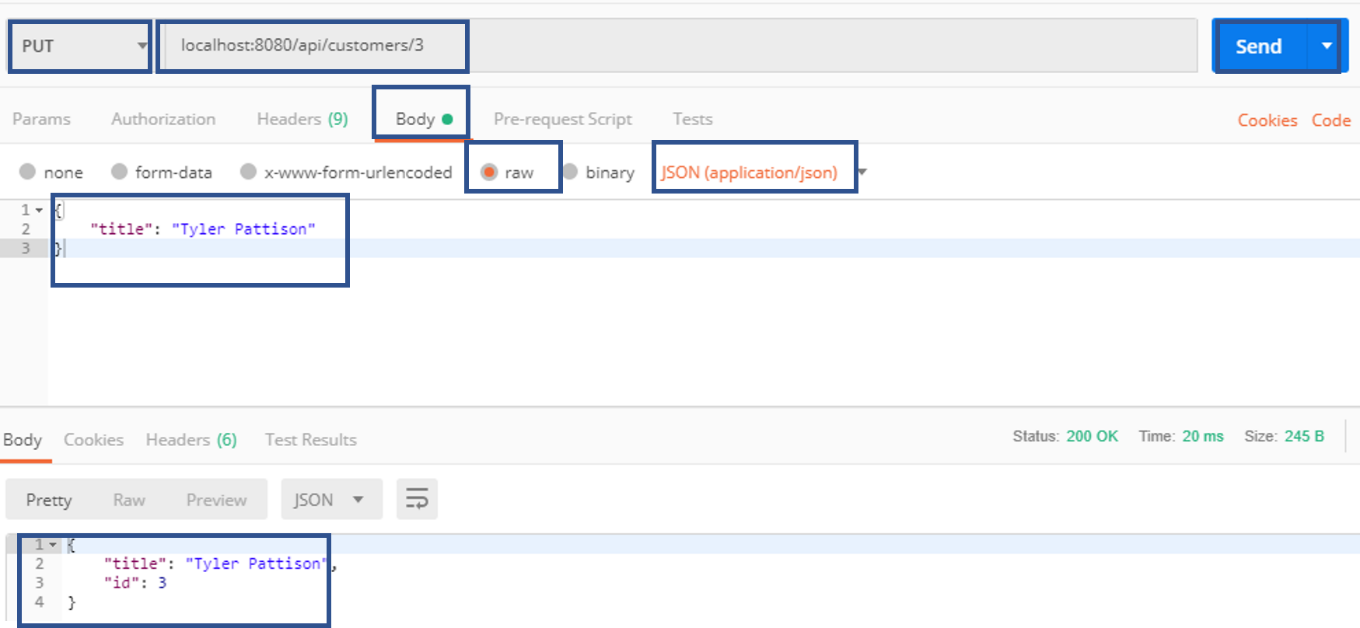
**Step 8:** Now, let’s try adding a new customer to our stack of customers. For that, select ‘POST’ from the drop-down list and type in the defined URL for the POST method. Then, click on ‘Body’, select ‘raw’ and move on to select ‘JSON’ from the drop-down list as depicted in the below image. Now, in the text area, type in the name of your customer as shown and hit send.



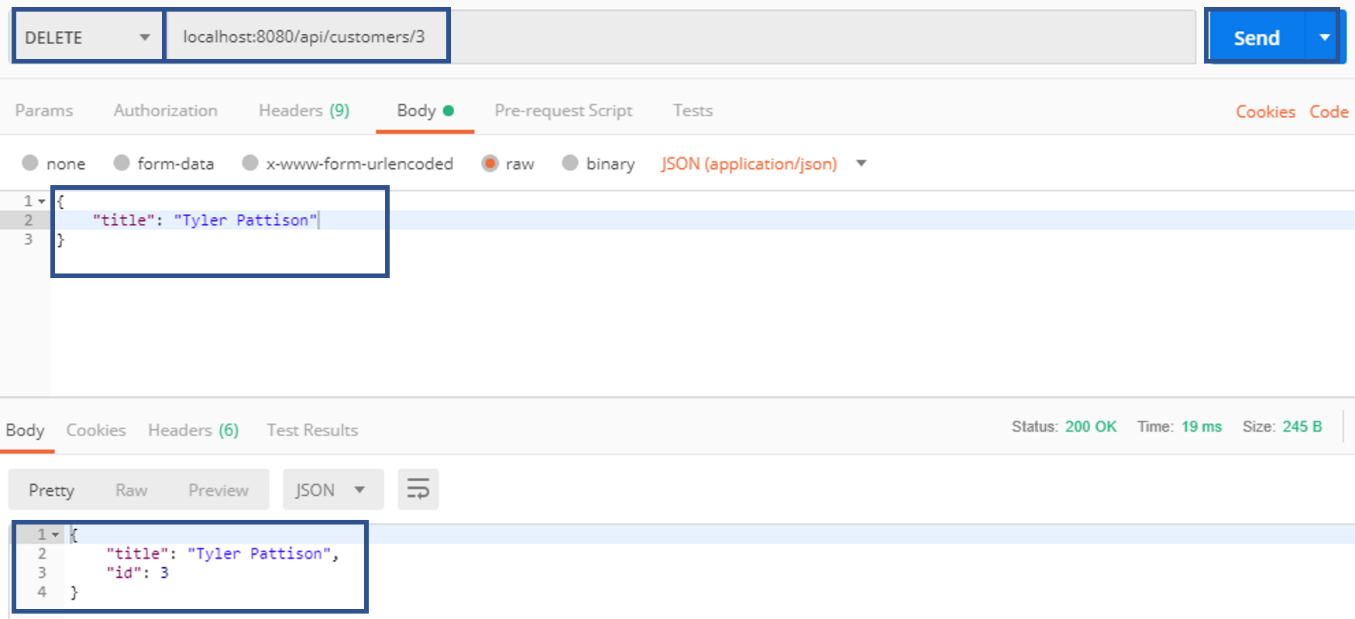
If your POST method is working fine, your response body will contain the new customer’s name along with the Customer ID. Here if you observe, we have only mentioned the name but we did not give the customer ID. This implies that the Customer ID is automatically incremented

**Step 9:** Now, let’s try to update a Customer name. Let us say we ant to update the name of the Customer ID = 3. So, to update the data, you need to first select ‘PUT’ from the drop-down table and enter the PUT request’s URL along with the customer id you wish to update. Next in the ‘Body’, type in the new customer name and hit enter.

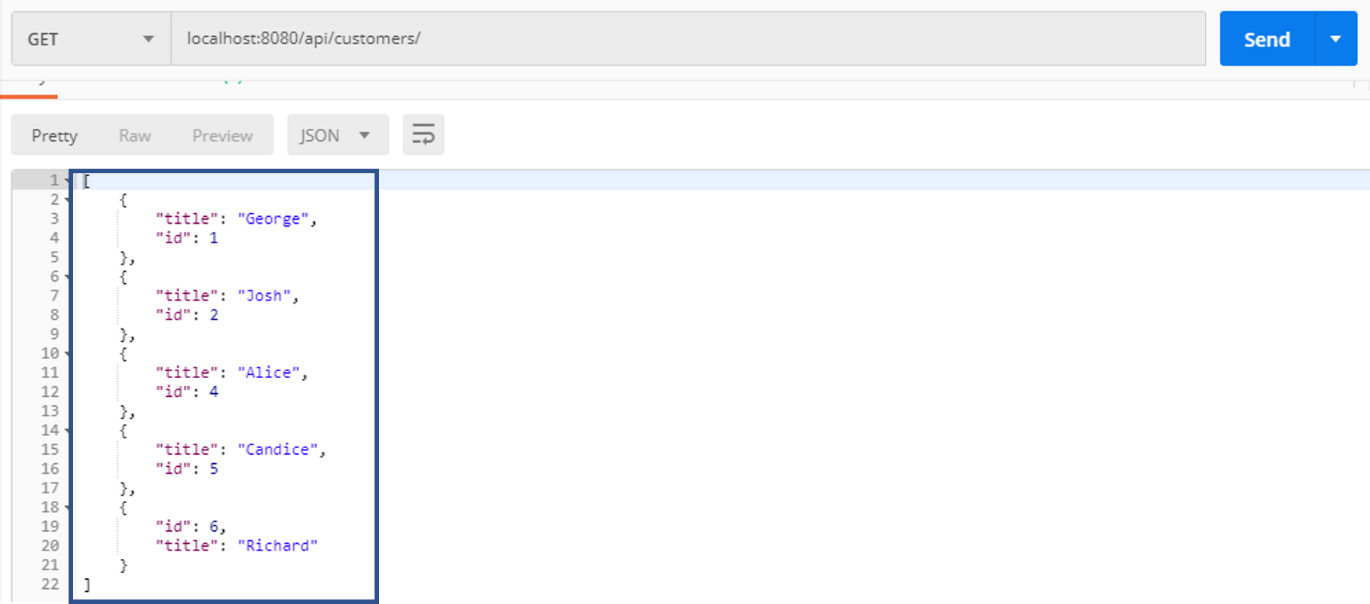
This will give you a response with the customer id and updated customer name.



**Step 10:** Finally, let’s send a ‘DELETE’ request to delete an existing record. For that select DELETE from the drop-down list and type in the URL of the delete request handler along with the customer’s details, you want to remove and hit enter. Let’s say, I want to delete the details of a customer with id = 3. If your transaction is successful, you will see the complete details of the entry you have removed in the response body.



Now, let’s send a GET request for our final list of customers.



As you can see from the above screenshot, the response body contains a total of five customers with the customer id 3 missing as we have already deleted that entry.

**References**:

1. <https://www.edureka.co/blog/what-is-rest-api/>

Activity

**Aim:** Web Application Integration Using Express+Angular+Mongo

**Learning outcome**: Able to develop the real time scenarios based on Node JS applications.

**Duration**: 2 hours

## List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Linux OS - Ubuntu 18.04 LTS
2. NodeJS, ExpressJS, AngularJS, MongoDB software installed

## Program:

**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: bezkoder**

**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**

**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(bodyParser.json());**

**// parse requests of content-type - application/x-www-form-urlencoded**

**app.use(bodyParser.urlencoded({ extended: true }));**

**// simple route**

**app.get("/", (req, res) => {**

**res.json({ message: "Welcome to bezkoder 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.

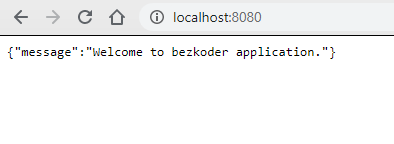
– create an Express app, then add body-parser (json and urlencoded) and cors middlewares using app.use() method. Notice that we set origin: http://localhost:8081.

– 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:



**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/bezkoder\_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;**

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"**

**}**

After finishing the steps above, we don’t need to write CRUD functions, Mongoose Model supports all of them:

create a new Tutorial: object.save()

find a Tutorial by id: findById(id)

retrieve all Tutorials: find()

update a Tutorial by id: findByIdAndUpdate(id, data)

remove a Tutorial: findByIdAndRemove(id)

remove all Tutorials: deleteMany()

find all Tutorials by title: find({ title: { $regex: new RegExp(title), $options: “i” } })

These functions will be used in our Controller.

**Create the Controller**

Inside app/controllers folder, let’s create tutorial.controller.js with these CRUD functions:

* create
* findAll
* findOne
* update
* delete
* deleteAll
* findAllPublished

**const db = require("../models");**

**const Tutorial = db.tutorials;**

**// Create and Save a new Tutorial**

**exports.create = (req, res) => {**

**};**

**// Retrieve all Tutorials from the database.**

**exports.findAll = (req, res) => {**

**};**

**// Find a single Tutorial with an id**

**exports.findOne = (req, res) => {**

**};**

**// Update a Tutorial by the id in the request**

**exports.update = (req, res) => {**

**};**

**// Delete a Tutorial with the specified id in the request**

**exports.delete = (req, res) => {**

**};**

**// Delete all Tutorials from the database.**

**exports.deleteAll = (req, res) => {**

**};**

**// Find all published Tutorials**

**exports.findAllPublished = (req, res) => {**

**};**

**Run the Node.js Express Server**

Run our Node.js application with command: **node server.js**

**Angular 10 Front-end**

* The App component is a container with router-outlet. It has navbar that links to routes paths via routerLink.
* TutorialsList component gets and displays Tutorials.
* Tutorial component has form for editing Tutorial’s details based on :id.
* AddTutorial component has form for submission new Tutorial.
* These Components call TutorialService methods which use Angular HTTPClient to make HTTP requests and receive responses.

There are 3 components: tutorials-list, tutorial-details, add-tutorial.

– tutorial.service has methods for sending HTTP requests to the Apis.

– app-routing.module.ts defines routes for each component.

– app component contains router view and navigation bar.

– app.module.ts declares Angular components and import necessary modules.

**Implementation**

**Setup Angular 10 Project**

Let’s open cmd and use Angular CLI to create a new Angular Project as following command:

**ng new Angular10Crud**

**? Would you like to add Angular routing? Yes**

**? Which stylesheet format would you like to use? CSS**

We also need to generate some Components and Services:

**ng g s services/tutorial**

**ng g c components/add-tutorial**

**ng g c components/tutorial-details**

**ng g c components/tutorials-list**

**Set up App Module**

Open app.module.ts and import FormsModule, HttpClientModule:

**...**

**import { FormsModule } from '@angular/forms';**

**import { HttpClientModule } from '@angular/common/http';**

**@NgModule({**

**declarations: [ ... ],**

**imports: [**

**...**

**FormsModule,**

**HttpClientModule**

**],**

**providers: [],**

**bootstrap: [AppComponent]**

**})**

**export class AppModule { }**

**Define Routes for Angular AppRoutingModule**

There are 3 main routes:

– /tutorials for tutorials-list component

– /tutorials/:id for tutorial-details component

– /add for add-tutorial component

app-routing.module.ts

**import { NgModule } from '@angular/core';**

**import { Routes, RouterModule } from '@angular/router';**

**import { TutorialsListComponent } from './components/tutorials-list/tutorials-list.component';**

**import { TutorialDetailsComponent } from './components/tutorial-details/tutorial-details.component';**

**import { AddTutorialComponent } from './components/add-tutorial/add-tutorial.component';**

**const routes: Routes = [**

**{ path: '', redirectTo: 'tutorials', pathMatch: 'full' },**

**{ path: 'tutorials', component: TutorialsListComponent },**

**{ path: 'tutorials/:id', component: TutorialDetailsComponent },**

**{ path: 'add', component: AddTutorialComponent }**

**];**

**@NgModule({**

**imports: [RouterModule.forRoot(routes)],**

**exports: [RouterModule]**

**})**

**export class AppRoutingModule { }**

**Create Data Service**

This service will use Angular HTTPClient to send HTTP requests.

You can see that its functions includes CRUD operations and finder method.

services/tutorial.service.ts

**import { Injectable } from '@angular/core';**

**import { HttpClient } from '@angular/common/http';**

**import { Observable } from 'rxjs';**

**const baseUrl = 'http://localhost:8080/api/tutorials';**

**@Injectable({**

**providedIn: 'root'**

**})**

**export class TutorialService {**

**constructor(private http: HttpClient) { }**

**getAll(): Observable<any> {**

**return this.http.get(baseUrl);**

**}**

**get(id): Observable<any> {**

**return this.http.get(`${baseUrl}/${id}`);**

**}**

**create(data): Observable<any> {**

**return this.http.post(baseUrl, data);**

**}**

**update(id, data): Observable<any> {**

**return this.http.put(`${baseUrl}/${id}`, data);**

**}**

**delete(id): Observable<any> {**

**return this.http.delete(`${baseUrl}/${id}`);**

**}**

**deleteAll(): Observable<any> {**

**return this.http.delete(baseUrl);**

**}**

**findByTitle(title): Observable<any> {**

**return this.http.get(`${baseUrl}?title=${title}`);**

**}**

**}**

**Create Angular Components**

As you’ve known before, there are 3 components corresponding to 3 routes defined in ***AppRoutingModule***.

* Add new Item Component
* List of items Component
* Item details Component

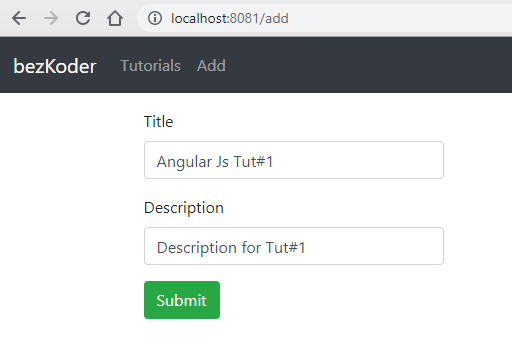
**Run the Angular App**

You can run this App with command: ng serve --port 8081.

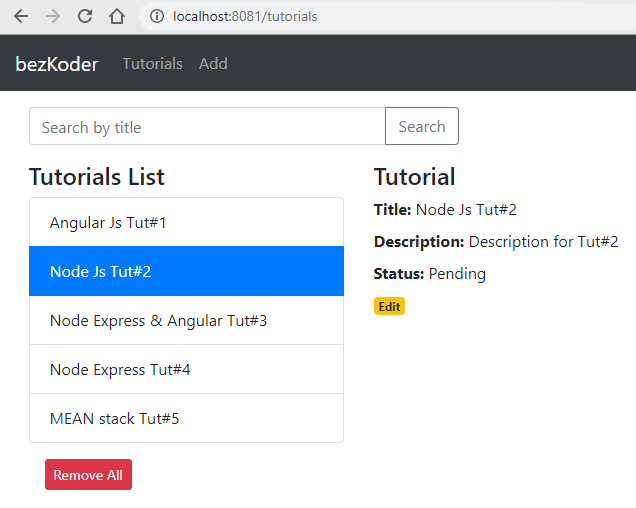
If the process is successful, open Browser with Url: http://localhost:8081/ and check it.

## Output/Results snippet:

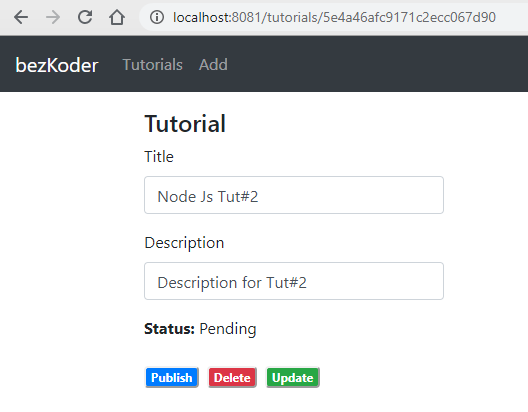
* Add an object:



* Retrieve all objects:



* Click on Edit button to update an object:



**References**:

1. <https://www.bezkoder.com/angular-mongodb-node-express/>

Activity

**Aim:** Create an Angular application using Node JS with complete templating system

**Learning outcome**: Able to develop the real time scenarios based on Node JS applications.

**Duration**: 2 hours

## List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Linux OS - Ubuntu 18.04 LTS
2. NodeJS, ExpressJS, VS Code installed

## Program:

### Install Angular CLI

Angular provides many libraries and packages for application development. You can install libraries required for your application using Angular CLI (Command Line Interface). Angular CLI is also used to generate, build, run, and deploy Angular application.

To install the Angular CLI globally using NPM, open a terminal/command window, and enter the following command:

**npm install -g @angular/cli@latest**

### Create Angular 2 Application

The following creates a new angular application named "FirstAngularApp" in the AngularApps folder.

**D:\AngularApps> ng new FirstAngularApp**

To open this project in VS Code, navigate to the project folder in the terminal/command window and type **code .** .

**D:\AngularApps\FirstAngularApp\>code .**

### Run Angular Application

Open the terminal in VS Code from menu Terminal -> New Terminal, and type ng serve -o command and press enter

### Angular 2 Components

Angular Component = HTML Template + Component Class + Component Metadata

### HTML Template

HTML template is nothing but a regular HTML code with additional Angular specific syntax to communicate with the component class.

### Generate Angular Component using Angular CLI

Use the following CLI command to generate a component.

**ng generate component greet**

Now, open greet.component.ts file in VS Code, and you will see the following code.

**import { Component, OnInit } from '@angular/core';**

**@Component({**

**selector: 'app-greet',**

**templateUrl: './greet.component.html',**

**styleUrls: ['./greet.component.css']**

**})**

**export class GreetComponent implements OnInit {**

**constructor() { }**

**ngOnInit(): void {**

**}**

**}**

Now, let's add a property and method in the component class.

**export class GreetComponent implements OnInit {**

**constructor() { }**

**ngOnInit(): void {**

**}**

**name: string = "Steve";**

**greet(): void {**

**alert("Hello " + this.name);**

**};**

**}**

Open greet.component.html file, remove existing code and add the following code.

**<div>**

**Enter Your Name: <input type="text" value={{name}} /> <br />**

**<button (click)="greet()">Greet Me!</button>**

**</div>**

In the above HTML template, we used name property in the {{ }} interpolation to display its value and greet() function as click event.

### Bootstrapping Component

Now, it's time to load our component, but before that, we need to host our application and load the root component. This process is called bootstrapping.

We will load our greet component into the root component in two steps.

1. Declare a component in the root module.

**import { BrowserModule } from '@angular/platform-browser';**

**import { NgModule } from '@angular/core';**

**import { AppRoutingModule } from './app-routing.module';**

**import { AppComponent } from './app.component';**

**import { GreetComponent } from './greet/greet.component'; //import GreetComponent**

**@NgModule({**

**declarations: [**

**AppComponent,**

**GreetComponent // <- include GreetComponent in declarations**

**],**

**imports: [**

**BrowserModule,**

**AppRoutingModule**

**],**

**providers: [],**

**bootstrap: [AppComponent]**

**})**

**export class AppModule { }**

1. Add component tag in the root HTML template.

**<div>**

**<app-greet></app-greet>**

**</div>**

Run the app by executing npm start command in the terminal of VS Code. After successful compilation, open the browser and enter <http://localhost:4200>.

We can also create a single component file greet.component.ts if the HTML code of a component is less. Use the template parameter in the @Component decorator to include HTML of the component. The following greet component gives the same result.

**import { Component } from '@angular/core';**

**@Component({**

**selector: "app-greet",**

**template: `<div>**

**Enter Your Name: <input type="text" value={{name}} /> <br/>**

**<button (click)="greet()">Greet Me!</button>**

**</div>`**

**})**

**export class GreetComponent {**

**name: string = "Steve";**

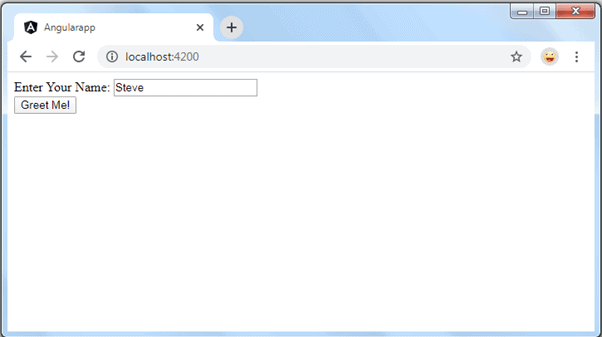
**greet(): void {**

**alert("Hello " + this.name);**

**};**

**}**

## Output/Results snippet:



**References**:

1. <https://www.tutorialsteacher.com/angular/create-angular-application>