In this tutorial, we will learn how to create a form page by using React and save the data submitted from the form to Mongo DB by using a Node JS backend. We will also learn how to use Node JS to load data from Mongo DB, and display the data in a React page.

Create Node JS Express App

1.1 Environment setup

1. Setup Node JS

If you have not installed node yet, please go to <https://nodejs.org/en/> to download it. Then right click the downloaded file such as “node-v12.16.3-x64.msi” and select install. Accept the license terms, follow the instruction and complete the installation. After the installation completes, open a cmd line console and type in “node -v” then “npm -v”. If version displays, this means installation is successful.

1. Setup Mongo DB

If Mongo DB is not installed yet, please go to <https://www.mongodb.com/download-center> to download it. Then extract the downloaded compressed file into the location you want the Mongo DB to be installed and add the directory “…/bin” into system “Path” variable.

1. Setup Testing Environment

In this tutorial, I will be using VS Code as my preferred code editor. If you would like to install it, you can do so at <https://code.visualstudio.com/download>. Select whichever version is correct for your device and follow the instructions to complete the installation.

1.2 Create Application Folder

Open a cmd console, change directory to drive C, type in

mkdir org-form

to create the main project folder in C drive named "org-form".

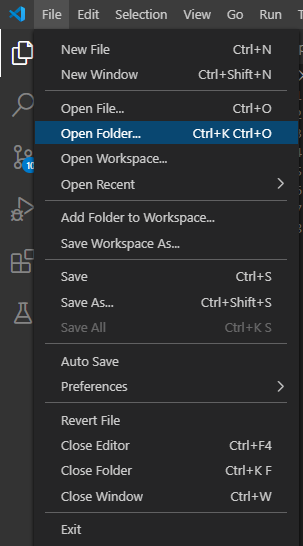
Change directory to the new directory just made, run the following commands:

npx express-generator api

cd api

npm install

This will create a new express app template in the folder api as well as install the default dependencies. Now, let’s open VS Code and open the folder. This can be done by the following:



1.3 Create config file

At this point, we are ready to start. Let’s first set up a location in our node JS app where we can store all of our configurations. To do that, make a new file named config.js in the api folder. Copy the following code into the config.js file:

module.exports = {

  'mongoUrl': 'mongodb://localhost:27017/organization'

};

1.4 Create Model, Service, Controller, and Route

Now it’s time to handle the routes. Change directory to “bin” within “api” and edit the “www” file. Find the line:

var port = normalizePort(process.env.PORT || '3000');

And change it to

var port = normalizePort(process.env.PORT || '9000');

this is because our front end will be running on port 3000.

We will be following the three tier architecture in this tutorial, meaning that we are spreading our code into three tiers: the model, controller, and service. This allows our code to be more organized and reusable. In order to create a three tier architecture, follow the steps below.

1. Create a new folder called “models” in the api folder. This folder will contain all of the models that we will be using. Now navigate to this folder on the command prompt and do the following:

npm install mongoose -- save

Let’s use the mongoose node module to handle our models

Now create a new file named “orgModel.js” in the newly created models folder. This file will define the model for our organization. Here is the “orgModel.js” file:

var mongoose = require('mongoose');

var orgModel = mongoose.Schema({

    name: {

        type: String

    },

    address: {

        type: String

    },

    type: {

        type: String

    }

}, {collection: 'organizations'});

module.exports = mongoose.model("orgs", orgModel);

1. Create a new folder called “services” in the api folder. This folder will contain all of the services that are involved with communicating with out server thorough the model. Next, create a file named “orgService.js” in the newly created services folder. Here is the orgService.js file.

var Organization = require('../models/orgModel');

const OrganizationService = {

    FindAll: (req) => {

        return Organization.find();

    },

    Create: (req) => {

        var org = new Organization({

            name: req.body.name,

            address: req.body.address,

            type: req.body.type

        });

        org.save();

    }

}

module.exports = OrganizationService;

1. Create a new folder called “controllers” in the api folder. This folder will contain all of the controllers that will be used in our node js program. Next, create a new file named “orgController.js” in the newly created controllers folder. Here is the “orgController.js” file:

var OrganizationService = require('../services/orgService');

exports.findAll = (req, res) => {

    OrganizationService.FindAll()

    .then(orgs => {

        res.send(orgs);

    })

};

exports.create = (req, res) => {

    OrganizationService.Create(req);

};

This controller defines two functions: findAll and create, which will find all organizations and create a new one respectively. These functions interact with orgModel to communicate with the database.

1. Create a new file named “orgRouter.js” in the routes folder. In this example, we are handling the submission and retrieval of a form so post and get requests are the only needed actions. Here is the “orgRouter.js” file:

module.exports = (app) => {

    const org = require('../controllers/orgController');

    app.post('/createorg', org.create);

    app.get('/listorg', org.findAll);

}

This code will create a new route for our express app. As seen from the code above, we will only be handling get and post requests, which will handle retrieving from the database and submitting to the database respectively.

1.5 Install CORS

To make the express app works, CORS is needed. Through CORS is not needed to run the express app on it’s own, but it is needed to connect the backend app to the frontend react app.

On the command console, change directory to “/org-form/api”, run the following command:

npm install cors -- save

1.6 Update app.js

Update the app.js file in /org-form/api to look like the following

var createError = require('http-errors');

var express = require('express');

var path = require('path');

var cookieParser = require('cookie-parser');

var logger = require('morgan');

var cors = require('cors');

var mongoose = require('mongoose');

mongoose.Promise = global.Promise;

var indexRouter = require('./routes/index');

var usersRouter = require('./routes/users');

var orgRouter = require('./routes/orgRouter');

var app = express();

// view engine setup

app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'jade');

app.use(cors());

app.use(logger('dev'));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(express.static(path.join(\_\_dirname, 'public')));

module.exports = (app) => {

  var org = require('./controllers/orgController');

  app.post('/createorg', org.create);

  app.get('/listorg', org.findAll);

}

var config = require('./config');

var url = config.mongoUrl;

var connect = mongoose.connect(url, {

  useNewUrlParser: true

});

connect.then((db) => {

  console.log("Connected correctly to server");

}, (err) => { console.log(err); });

require('./routes/orgRouter')(app);

// catch 404 and forward to error handler

app.use(function (req, res, next) {

  next(createError(404));

});

// error handler

app.use(function (err, req, res, next) {

  // set locals, only providing error in development

  res.locals.message = err.message;

  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page

  res.status(err.status || 500);

  res.render('error');

});

module.exports = app;

This update version of the app.js file will make use of all of the new flies and dependencies that we have implemented in 2.3 and 2.4

Create React App

2.1 Install react

The next step is to create our frontend form. First go back to the command console, change directory to the project root “org-form”. Type in the following command:

npx create-react-app client

This will create a new React template under the directory “client”.

Now navigate to the newly created client folder and run the following two commands:

npm install axios --save

npm install react-router-dom -- save

The “axios” library is needed in communication between Node JS backend and React frontend.

The “react-router-dom” library is needed in order to route between various pages that we will have in our app.

2.2 Get data from DB and display it in UI

Let’s get started on making our react app. We will start by making a components folder in client/src. This folder will contain all of the components that we will be using in our react app.

After creating the folder, let’s make a new file named ListOrg.js in the components folder. This component will be used in order to display all of the organizations that are currently in our database. Below is our completed ListOrg.js file.

import React, { Component } from 'react';

import Table from './Table';

class ListOrg extends Component {

    constructor(props) {

      super(props);

      this.state = {

        response: [{}]

      };

    }

    componentDidMount() {

      fetch("http://localhost:9000/listorg")

        .then(res => res.json())

        .then(response => this.setState({ response }));

    }

    render() {

      return (

        <div>

          <br />

          <div className="container">

            <header>

              <h1>All Orgs</h1>

            </header>

            <Table data={this.state.response} />

          </div>

        </div>

      );

    }

  }

  export default ListOrg;

In our constructor, we are storing a json array named response. This is because when we perform a get request to the node js app, we will be sent a json array of all current organizations. This array can be stored in the state to be used by our react app.

We also have the componentDidMount method from the object life-cycle. This method will perform a fetch request to our node js app and store the received array to our state. Here I use fetch, but you can use axios like we did for submitting a form.

Now it’s time to display our data. A simple way to display the data would be to simply write what is currently stored in the response state in a paragraph, but that would just display the entire json array. Instead, let’s make a table to display the data. In client/src/components, make a new folder called Table.js. Then, in the new file, copy the following code:

import React, { Component } from 'react';

class Table extends React.Component {

constructor(props) {

super(props);

this.getHeader = this.getHeader.bind(this);

this.getRowsData = this.getRowsData.bind(this);

this.getKeys = this.getKeys.bind(this);

}

getKeys = function () {

return Object.keys(this.props.data[0]);

}

getHeader = function () {

var keys = this.getKeys();

return keys.map((key, index) => {

return <th key={key}>{key.toUpperCase()}</th>

})

}

getRowsData = function () {

var items = this.props.data;

var keys = this.getKeys();

return items.map((row, index) => {

return <tr key={index}><RenderRow key={index} data={row} keys={keys} /></tr>

})

}

render() {

return (

<div>

<table>

<thead>

<tr>{this.getHeader()}</tr>

</thead>

<tbody>

{this.getRowsData()}

</tbody>

</table>

</div>

);

}

}

const RenderRow = (props) => {

return props.keys.map((key, index) => {

return <td key={props.data[key]}>{props.data[key]}</td>

})

}

export default Table;

This code will take in a json array and use the getHeader and getRowsData methods to create a table with the given data.

2.3 Home page and Navigator

With our first component created, let’s add a simple home page and a way to navigate between pages. In the components folder, create two new files named Home.js and Navigation.js.

First, let’s make our homepage. In the Home.js file, copy the following code:

import React from 'react';

const home = () => {

    return (

       <div>

          <h1>Home</h1>

           <p>Welcome! Please use the navigator above to start.</p>

       </div>

    );

}

export default home;

This creates a page with a simple header and message.

We can now move on to the more important Navigation.js file. Before we type any code, we will need to install a new node-module. In the command prompt, navigate to org-form/client and type the following command

We will be using NavLink to do our routing. Within the Navigation.js file, copy the following code:

import React from 'react';

import { NavLink } from 'react-router-dom';

const Navigation = () => {

    return (

       <div>

          <NavLink to="/">Home</NavLink>

          <br/>

          <NavLink to="/listOrg">Get All</NavLink>

       </div>

    );

}

export default Navigation;

Here, we are using NavLink to navigate to different end points. For example, if “Get All” is selected, it will go to the /listOrg end point. We will implement what these end points do in the next step.

2.4 Edit App.js

We will now edit the template App.js file. In order to do this, we will first have to import all of the components and libraries that we have made and will need. First, delete everything in the already existing App.js file and add the imports:

import React, { Component } from 'react';

import { BrowserRouter, Route, Switch } from 'react-router-dom';

import logo from './logo.svg';

import './App.css';

import Home from './components/Home';

import ListOrg from './components/ListOrg';

import Navigation from './components/Navigation';

With the imports installed, the next step is to make a class component. Since all of the actions are done in the components, the only thing that App.js needs to handle is routing. Below is the class components in App.js:

class App extends Component {

  render() {

    return (

       <BrowserRouter>

        <div>

          <Navigation />

            <Switch>

             <Route path="/" component={Home} exact/>

             <Route path="/listOrg" component={ListOrg}/>

           </Switch>

        </div>

      </BrowserRouter>

    );

  }

}

export default App;

As seen above, App.js is using BrowserRouter, Switch, and Route from the react-router-dom node module. We are using Switch in order to assign the end points that were defined in Navigation.js to other components.

Now we can try running our application and see if it works. First, make sure that your mongo db server is running. When you have your mongo db server running, the next step is to start the node js back end. Open a new command prompt, and navigate to org-form/api. Now run the npm start command on the line. The message “Connected correctly to server” should be displayed.

Finally, we can run our actual react application. Like we did for the node js app, open a new command prompt and navigate to org-form/client and run npm start. We can find our app at <http://localhost:3000/>.

You may see that for now /listOrg has nothing in it except for the header, but this is expected as we have not added anything yet.

2.5 Post form data to database

The Last thing that needs to be added is a form submission system.

1. Create SubmitOrg component for organization creation

Now let’s start making our form. In VS Code, navigate to “client/src/components”, and create a new file called createOrg.js. Here it is the outline.

import React, { Component } from 'react';

class CreateOrg extends Component {

render() {

return (

<div></div>

);

}

}

export default CreateOrg;

1. Create a constructor

The constructor that will initialize the default form values. Right under the class declaration insert the following lines of code:

constructor(props) {

super(props);

this.state = {

name: '',

address: '',

type: 'Gov'

};

}

This is setting the default state of our form to the specified values. “type: Gov” sets the default value of organization type into “Gov” in the drop-down menu.

1. Add form in the render

We make a very basic form without any css or extra classes. Within our render function, let’s change the return statement to the following:

render() {

return (

<div>

<br />

<div className="container">

<form onSubmit={this.handleSubmit}>

<div style={{ width: '30%' }} className="form-group">

<label>Name: </label>

<input

type="text"

className="form-control"

name="name"

placeholder="Your Name"

onChange={this.handleInputChange}

/>

</div>

<br />

<div style={{ width: '30%' }} className="form-group">

<label>Address: </label>

<input

type="text"

className="form-control"

name="address"

placeholder="Your Address"

onChange={this.handleInputChange}

/>

</div>

<br />

<div style={{ width: '30%' }} className="form-group">

<label>Type: </label>

<select

type="text"

className="form-control"

name="type"

placeholder="Gov"

onChange={this.handleInputChange}>

<option value="Gov">Gov</option>

<option value="Hospital">Hospital</option>

<option value="SuperMarket">SuperMarket</option>

</select>

</div>

<br />

<div style={{ width: '30%' }}>

<button className="btn btn-success" type="submit">

Submit

</button>

</div>

</form>

</div>

</div>

);

}

This will create a Form with three fields and a button to submit the form. The first two are text fields while the third is a drop-down menu.

As seen from the code above, the functions, handleSubmit and handleInputChange need to be implemented before the code will work. Let’s do that next.

1. create handleInputChange

Now let’s create handleInputChange functionality. Copy this code under the previously defined constructor.

handleInputChange = e => {

console.log("setting " + e.target.name + " to " + e.target.value);

this.setState({

[e.target.name]: e.target.value,

});

};

This method will change the current state as the form fields change. If we monitor the JavaScript console while typing value in the fields, we can see the console.log statement prints out the changed states.

1. create handleSubmit

First, let’s import the axios node-module that was previously installed. Copy this line of code under the other import statements:

import axios from 'axios';

Copy the code following code just after the handleInputChange that was just created.

handleSubmit = e => {

e.preventDefault();

const { name, address, type } = this.state;

const org = {

name,

address,

type

};

axios

.post('http://localhost:9000/createorg', org)

.then(() => console.log('Organization Created'))

.catch(err => {

console.error(err);

});

};

This piece of code uses the axios node module to post the form data to the backend Node application at port 9000 with the form data org as a parameter in Json.

With these changes, let’s try running the application again. But first, we must edit our App.js and Navigation.js files to include CreateOrg.js.

Here is what our updated App.js file should look like:

import React, { Component } from 'react';

import { BrowserRouter, Route, Switch } from 'react-router-dom';

import logo from './logo.svg';

import './App.css';

import Home from './components/Home';

import ListOrg from './components/ListOrg';

import SubmitOrg from './components/SubmitOrg';

import Navigation from './components/Navigation';

class App extends Component {

  render() {

    return (

       <BrowserRouter>

        <div>

          <Navigation />

            <Switch>

             <Route path="/" component={Home} exact/>

             <Route path="/submit" component={SubmitOrg}/>

             <Route path="/listOrg" component={ListOrg}/>

           </Switch>

        </div>

      </BrowserRouter>

    );

  }

}

export default App;

And here is the updated Navigation.js file:

import React from 'react';

import { NavLink } from 'react-router-dom';

const Navigation = () => {

    return (

       <div>

          <NavLink to="/">Home</NavLink>

          <br/>

          <NavLink to="/submit">Submit New</NavLink>

          <br/>

          <NavLink to="/listOrg">List All</NavLink>

       </div>

    );

}

export default Navigation;

Now we have a new end point that is added to out Navigator. Start the servers like we did before and look at our new app. Try submitting a new organization and see if it get’s added to our list.

1. Validation

Let’s implement some simple validation rule before a form can be submitted.

First updating the constructor so that it can take in errors.

constructor(props) {

super(props);

this.state = {

name: '',

address: '',

type: 'Gov',

errors: {

name: '',

address: '',

type: '',

}

};

}

Now the state contains a list of errors, one for each of our form fields. In this tutorial, we will only cover the name form field. Let’s say that any organization which has the letter “a” in the name is invalid.

Editing handleInputChange method. Before the state is set, insert the following code snippet:

const { name, value } = e.target;

let errors = this.state.errors;

switch (name) {

case 'name':

errors.name = !validNameRegex.test(value) ? 'Name can not have an A or a' : '';

break;

}

this.setState({ errors, [name]: value }, () => {

console.log(errors)

})

This will check the name field and set the state of the error to a new error message depending on whether or not some conditions are true. In this case, it’s validNameRegex. Since we don’t have this implemented, this is the next part that is needed to be implemented

Under the import statements, let’s will define our condition:

const validNameRegex = /^((?!a|A)[\s\S])\*$/;

We are going to use regular expressions to determine if your string contains an “a” or “A” in it.

Now let’s display the error on the form page if there’s an error.

At the top of our render function, copy the following line of code:

const { errors } = this.state;

This will store the current list of errors under the constant errors. Now, under the div where we defined our name form field, copy the following:

{errors.name.length > 0 &&

<span style={{color: "red"}} className='error'>{errors.name}</span>}

This essentially states that if there currently exists an error associated with the name field, then display it in the color red.

Our final step is to check if our form is valid before submitting. Let’s edit the handleSubmit method to the following:

handleSubmit = e => {

    console.log('submitting...');

    e.preventDefault();

    if (validateForm(this.state.errors)) {

      console.info('Valid Form');

      const { name, address, type } = this.state;

      const org = {

        name,

        address,

        type,

      };

      axios

        .post('http://localhost:9000/createorg', org)

        .then(() =>

          console.log('Organization Created')

        )

        .catch(err => {

          console.error(err);

        });

    } else {

      console.error('Invalid Form')

    }

  };

Right before the data is send to the Node JS backend, we are checking if the form is valid or not. If it is, then log it and continue the submission. However, if the form is not valid, then an error message is displayed.

The last step is the implement validateForm. Right before the class declaration, copy the following:

const validateForm = (errors) => {

let valid = true;

Object.values(errors).forEach(

(val) => val.length > 0 && (valid = false)

);

return valid;

}

This will check the errors that are currently in our state and return false if any exist, else it will return true. With this, we complete the SubmitOrg.js file. Here is the final code for the SubmitOrg.js file:

import React, { Component } from 'react';

import axios from 'axios';

const validNameRegex = /^((?!a|A)[\s\S])\*$/;

const validateForm = (errors) => {

  let valid = true;

  Object.values(errors).forEach(

    (val) => val.length > 0 && (valid = false)

  );

  return valid;

}

class SubmitOrg extends Component {

  constructor(props) {

    super(props);

    this.state = {

      name: '',

      address: '',

      type: 'Gov',

      errors: {

        name: '',

        address: '',

        type: '',

      }

    };

  }

  handleInputChange = e => {

    const { name, value } = e.target;

    let errors = this.state.errors;

    switch (name) {

      case 'name':

        errors.name = !validNameRegex.test(value) ? 'Name can not have an A or a' : '';

        break;

    }

    this.setState({ errors, [name]: value }, () => {

      console.log(errors)

    })

    console.log("setting " + e.target.name + " to " + e.target.value);

    this.setState({

      [e.target.name]: e.target.value,

    });

  };

  handleSubmit = e => {

    console.log('submitting...');

    e.preventDefault();

    if (validateForm(this.state.errors)) {

      console.info('Valid Form');

      const { name, address, type } = this.state;

      const org = {

        name,

        address,

        type,

      };

      axios

        .post('http://localhost:9000/createorg', org)

        .then(() =>

          console.log('Organization Created')

        )

        .catch(err => {

          console.error(err);

        });

    } else {

      console.error('Invalid Form')

    }

  };

  render() {

    const { errors } = this.state;

    return (

      <div>

        <br />

        <div className="container">

          <form onSubmit={this.handleSubmit}>

            <div style={{ width: '30%' }} className="form-group">

              <label>Name: </label>

              <input

                type="text"

                className="form-control"

                name="name"

                placeholder="Your Name"

                onChange={this.handleInputChange}

              />

            </div>

            {errors.name.length > 0 &&

              <span style={{ color: "red" }} className='error'>{errors.name}</span>}

            <br />

            <div style={{ width: '30%' }} className="form-group">

              <label>Address: </label>

              <input

                type="text"

                className="form-control"

                name="address"

                placeholder="Your Address"

                onChange={this.handleInputChange}

              />

            </div>

            <br />

            <div style={{ width: '30%' }} className="form-group">

              <label>Type: </label>

              <select

                type="text"

                className="form-control"

                name="type"

                placeholder="Gov"

                onChange={this.handleInputChange}>

                <option value="Gov">Gov</option>

                <option value="Hospital">Hospital</option>

                <option value="SuperMarket">SuperMarket</option>

              </select>

            </div>

            <br />

            <div style={{ width: '30%' }}>

              <button className="btn btn-success" type="submit">

                Submit

              </button>

            </div>

          </form>

        </div>

      </div>

    );

  }

}

export default SubmitOrg;

When the form is submitted, we want to be redirected to the /listOrg end point. In order to do this, edit the handleSubmit method to look like this:

handleSubmit = e => {

    console.log('submitting...');

    e.preventDefault();

    if (validateForm(this.state.errors)) {

      console.info('Valid Form');

      const { name, address, type } = this.state;

      const org = {

        name,

        address,

        type,

      };

      axios

        .post('http://localhost:9000/createorg', org)

        .then(() =>

          console.log('Organization Created'),

          this.props.history.push('/listOrg')

        )

        .catch(err => {

          console.error(err);

        });

    } else {

      console.error('Invalid Form')

    }

  };

Conclusion

With this, we have a fully functioning form submission system with react and node js. To test if the code is working, we can do the following.

1. Ensure that mongo db is running in the background.
2. Navigate the the api and client folders on two separate command prompts and run npm start on both
3. Go to <http://localhost:3000/>
4. Upon submitting the form, you can see in the table below all organizations that have been added. Make sure to refresh the page to see new submssions.

The full repository can be found at this link: <https://github.com/hubiuts/React-NodeJS-form>. Remember to run npm install in both the api and client folders before running if you clone the repository.