# **CRUD App:Login and Registration using NodeJS With Express and Mongo**

### **1. Introduction**

we can create a simple Application in Express JS ( Node JS Framework ). We can create a CRUD (Create, Read, Update, Delete) Application.

### **2. CRUD, Express and MongoDB**.

[**Express**](https://expressjs.com/) is a framework for building web applications on top of [Node.js](https://nodejs.org/en/). It simplifies the server creation process that is already available in Node

[**MongoDB**](https://www.mongodb.com/) is a database. This is the place where you store information for your websites (or applications).

[**CRUD**](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) is an acronym for Create, Read, Update and Delete. It is a set of operations we get servers to execute (POST, GET, PUT and DELETE requests respectively). This is what each operation does:

* **Create (POST) -** Make something
* **Read (GET)-** Get something
* **Update (PUT) -** Change something
* **Delete (DELETE)-** Remove something

POST, GET, PUT, and DELETE requests let us construct [Rest APIs](https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/).

If we put CRUD, Express and MongoDB together into a single diagram, this is what it would look like:

### Diagram Description automatically generated

### **3. Prerequisites**

You should have prior knowledge of javascript basics, nodejs, Express, mongodb And, at last nodejs should be installed on your system.

### **4. Packages Required**

You will be needing these following 'npm' packages.

1. **Express**: Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications
2. **express-validator**: To Validate the body data on the server in the express framework, we will be using this library. It's a server-side data validation library. So, even if a malicious user bypasses the client-side verification, the server-side data validation will catch it and throw an error.
3. **body-parser**: It is nodejs middleware for parsing the body data.
4. **Mongoose:** Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment. Mongoose supports both promises and callbacks.

### **5. Project Structure**

Text

Description automatically generated with medium confidence

### **6. HTTP Server**

create the main entry point of our application. Create a new file named server.js in the root folder of the application with the following contents -

const express = require('express');

const dotenv = require('dotenv');

const morgan = require('morgan');

const bodyparser = require("body-parser");

const path = require('path');

const connectDB = require('./server/database/connection');

const app = express();

dotenv.config( { path : 'config.env'} )

const PORT = process.env.PORT || 8080

// log requests

app.use(morgan('tiny'));

// mongodb connection

connectDB();

// parse request to body-parser

app.use(bodyparser.urlencoded({ extended : true}))

// set view engine

app.set("view engine", "ejs")

//app.set("views", path.resolve(\_\_dirname, "views/ejs"))

// load assets

app.use('/css', express.static(path.resolve(\_\_dirname, "assets/css")))

app.use('/img', express.static(path.resolve(\_\_dirname, "assets/img")))

app.use('/js', express.static(path.resolve(\_\_dirname, "assets/js")))

// load routers

app.use('/', require('./server/routes/router'))

app.listen(PORT, ()=> { console.log(`Server is running on http://localhost:${PORT}`)});

**First**, We import express and body-parser modules. Then, We create an express app, and add two body-parser middlewares using express’s app.use() method.

### **7. Configuring and Connecting to the database**

Create a new file database.config.js inside config folder with the following contents -

const mongoose = require('mongoose');

const connectDB = async () => {

try{

// mongodb connection string

const con = await mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

useFindAndModify: false,

useCreateIndex: true

})

console.log(`MongoDB connected : ${con.connection.host}`);

}catch(err){

console.log(err);

process.exit(1);

}

}

module.exports = connectDB

After that import the above database configuration in server.js and connect to the database using mongoose.

### **8. Defining the model in Mongoose**

create a file called model.js inside server/model folder with the following contents –

const mongoose = require('mongoose');

var schema = new mongoose.Schema({

name : {

type : String,

required: true

},

email : {

type: String,

required: true,

unique: true

},

gender : String,

status : String

})

const Userdb = mongoose.model('userdb', schema);

module.exports = Userdb;

**9. Defining Routes using Express**

create a new file called router.js inside server/routes folder with the following contents –

const express = require('express');

const route = express.Router()

const services = require('../services/render');

const controller = require('../controller/controller');

/\*\*

\* @description Root Route

\* @method GET /

\*/

route.get('/', services.homeRoutes);

/\*\*

\* @description add users

\* @method GET /add-user

\*/

route.get('/add-user', services.add\_user)

/\*\*

\* @description for update user

\* @method GET /update-user

\*/

route.get('/update-user', services.update\_user)

// API

route.post('/api/users', controller.create);

route.get('/api/users', controller.find);

route.put('/api/users/:id', controller.update);

route.delete('/api/users/:id', controller.delete);

module.exports = route

**10. Writing the Controller functions**

Create a new folder called controllers inside the app folder, then create a new file called controller.js inside app/controllers folder with the following contents

var Userdb= require('../controller/controller);

// Create and Save a new Note

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

};

// Retrieve and return all notes from the database.

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

};

// Find a single note with a noteId

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

};

// Update a note identified by the noteId in the request

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

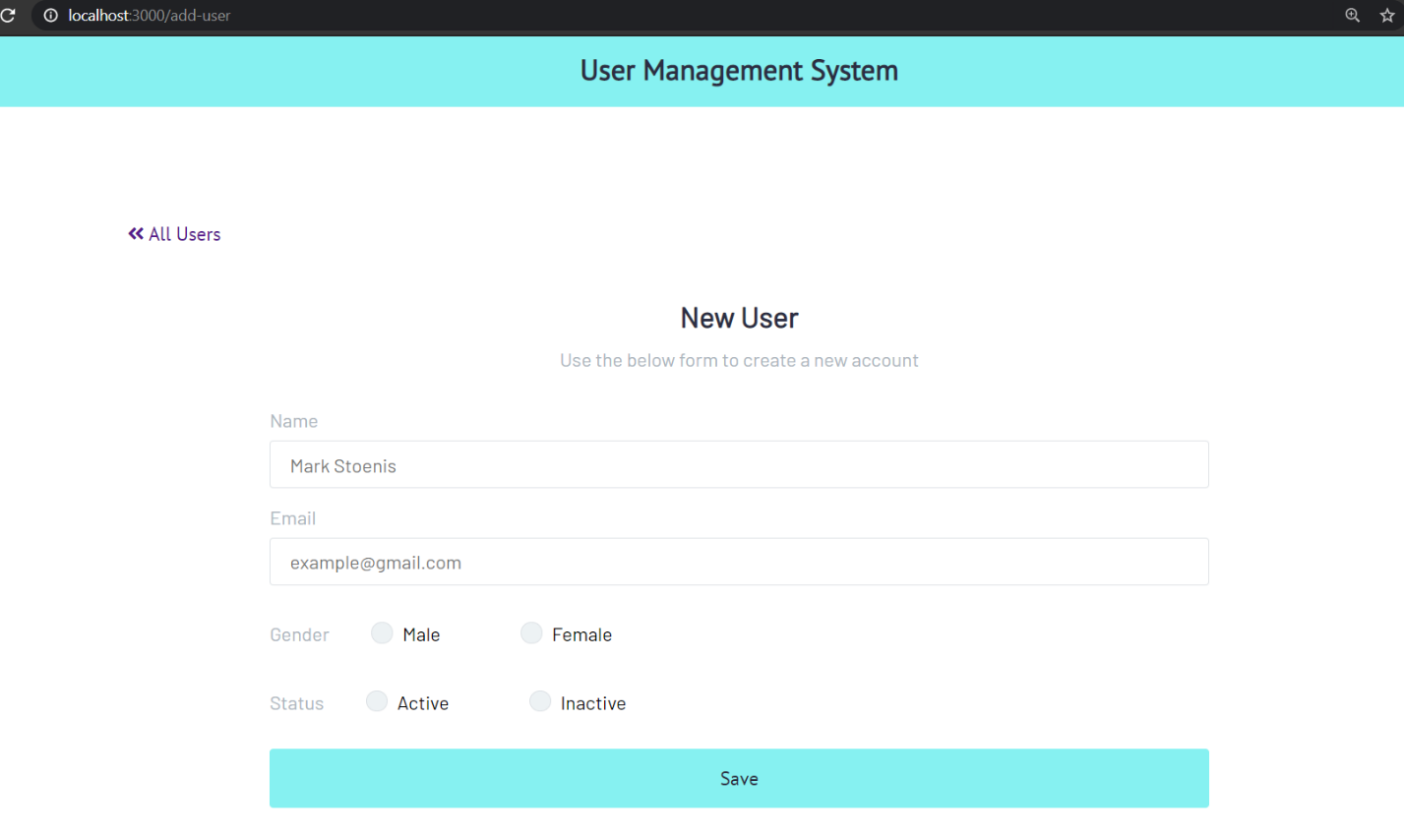
};

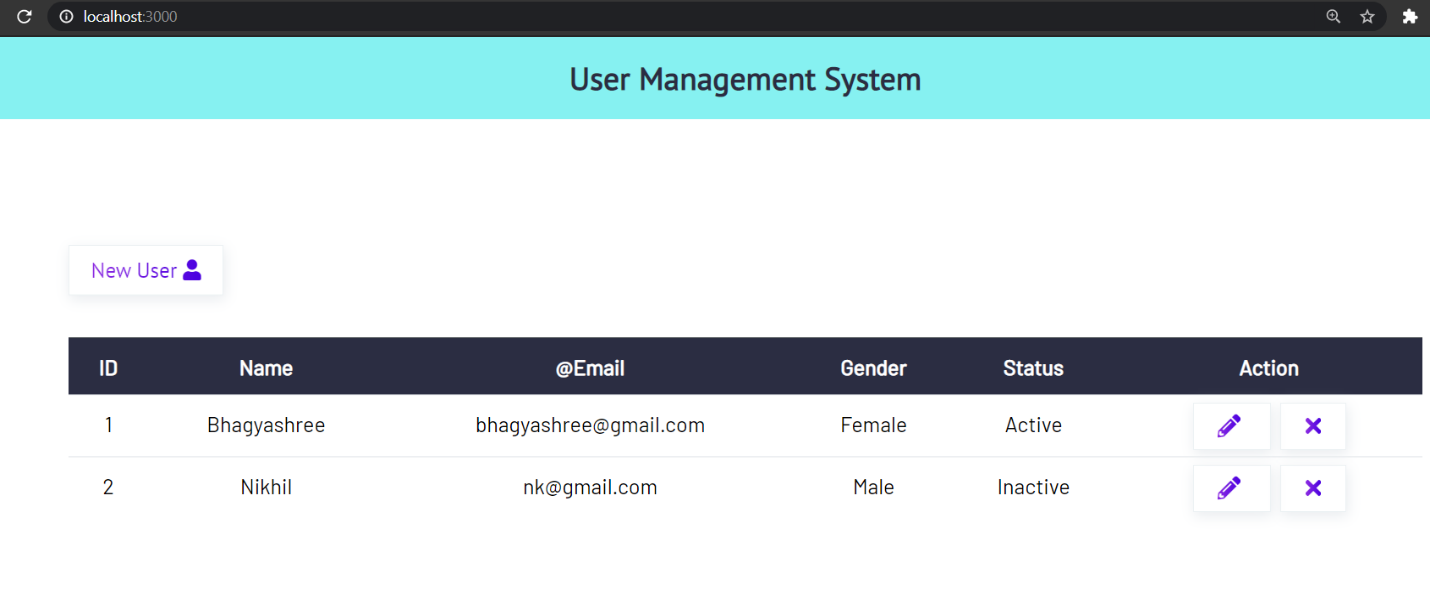
// Delete a note with the specified noteId in the request

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

};

**11. Screenshot of Login crud application**

****

****