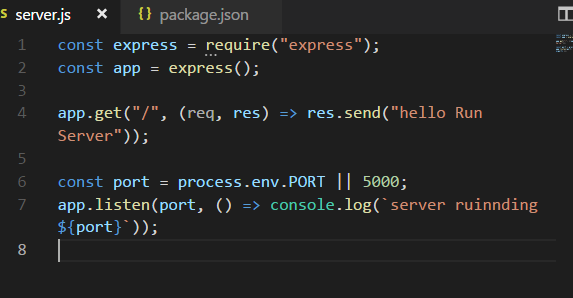
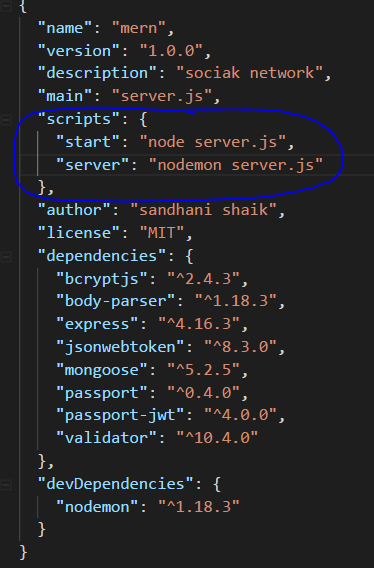
MERN DEVELOPER

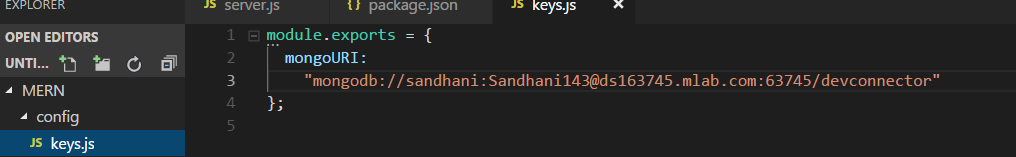
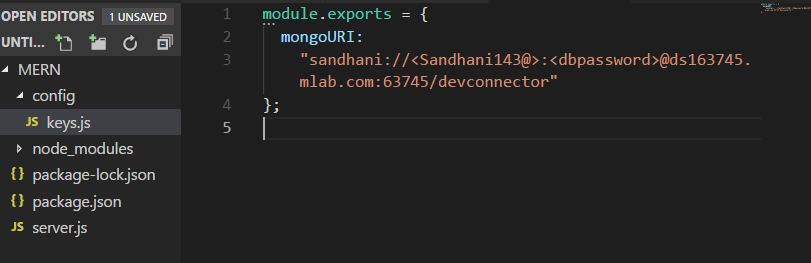
1st Videos

* Npm init
* Npm i express mongoose passport passport-jwt jsonwebtoken body-parser bcryptjs validator
* Npm I –D nodemon
* 
* Create server.js file npm run server



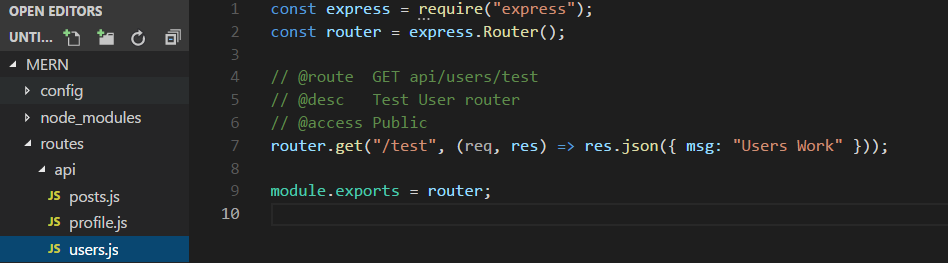
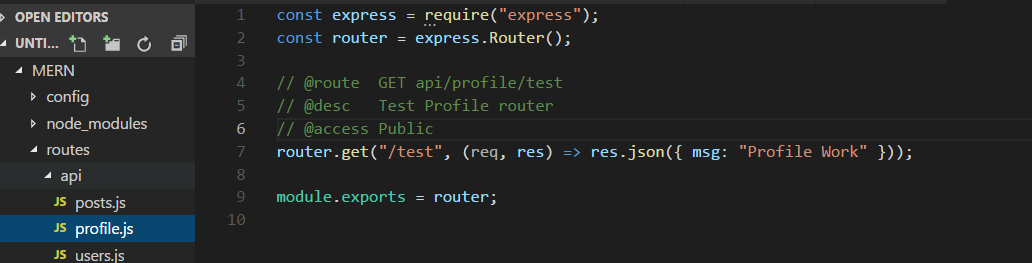
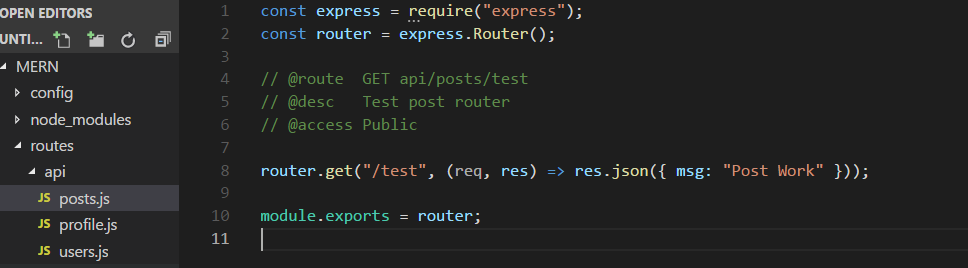
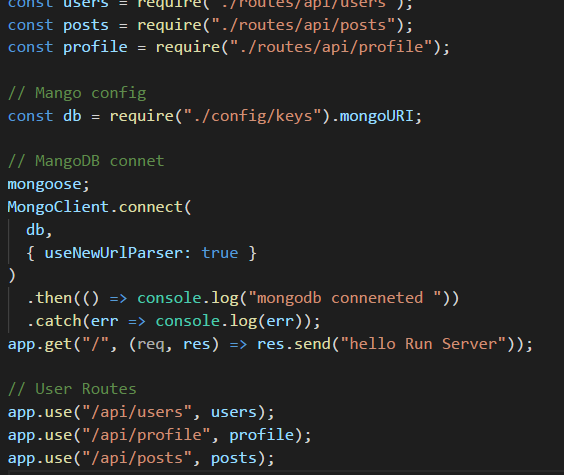
2nd video

* Create config folder for moongose
* Keys.js
* Get code from mlab mongodb://<dbuser>:<dbpassword>@ds163745.mlab.com:63745/devconnector



3rd videos

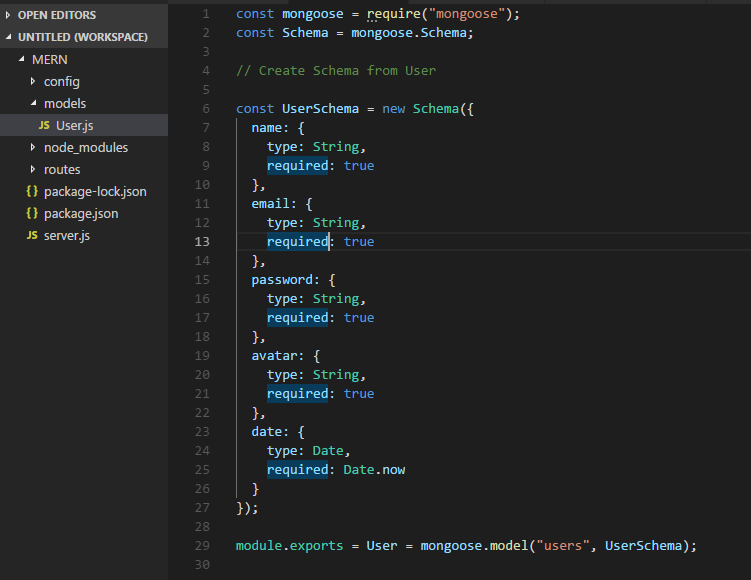
Create routes for user, post and profile



Route for testing

4th Video

Create model for User



User register Route

Server.js

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

// BOdy Parser middleware

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

app.use(bodyParser.json());

User router

const gravatar = require("gravatar");

const bcrypt = require("bcryptjs");

// @route GET api/users/register

// @desc Test UserRegister router

// @access Public

router.post("/register", (req, res) => {

User.findOne({ email: req.body.email }).then(user => {

if (user) {

return res.status(400).json({ email: "email alredy exits" });

} else {

const avatar = gravatar.url(req.body.email, {

s: "200", //size

r: "pg", //rating

d: "mm" //mm

});

const newUser = new User({

name: req.body.name,

email: req.body.email,

avatar,

password: req.body.password

});

bcrypt.genSalt(10, (err, salt) => {

bcrypt.hash(newUser.password, salt, (err, hash) => {

if (err) throw err;

newUser.password = hash;

newUser

.save()

.then(user => res.json(user))

.catch(err => console.log(err));

});

});

}

});

});

// @route GET api/users/login

// @desc Test Login router

// @access Public

router.post("/login", (req, res) => {

const email = req.body.email;

const password = req.body.password;

// Find the user

User.findOne({ email }).then(user => {

if (!user) {

return res.status(404).json({ email: "user email not found" });

}

// chack password

bcrypt.compare(password, user.password).then(isMatch => {

if (isMatch) {

res.json({ msg: "success", user });

} else {

return res.status(404).json({ password: "not mached" });

}

});

});

});

JWT Auth

User route

* const jwt = require("jsonwebtoken");
* const keys = require("../../config/keys");

config folder

module.exports = {

mongoURI:

"mongodb://sandhani:Sandhani143@ds163745.mlab.com:63745/devconnector",

secretOrKey: "secret"

};

User route

// @route GET api/users/login

// @desc Test Login router

// @access Public

router.post("/login", (req, res) => {

const email = req.body.email;

const password = req.body.password;

// Find the user

User.findOne({ email }).then(user => {

if (!user) {

return res.status(404).json({ email: "user email not found" });

}

// chack password

bcrypt.compare(password, user.password).then(isMatch => {

if (isMatch) {

//User matched

const payload = { id: user.id, name: user.name, avatar: user.avatar };

//sign token

jwt.sign(

payload,

keys.secretOrKey,

{ expiresIn: 3600 },

(err, token) => {

res.json({

success: true,

token: "Bearer " + token,

expiresIn: 3600,

payload: payload

});

}

);

} else {

return res.status(404).json({ password: "not mached" });

}

});

});

});

Current route Auth and Passport config

Server.js

const express = require("express");

const mongoose = require("mongoose");

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

const passport = require("passport");

const app = express();

// BOdy Parser middleware

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

app.use(bodyParser.json());

// Routes Refer

const users = require("./routes/api/users");

const posts = require("./routes/api/posts");

const profile = require("./routes/api/profile");

// Mango config

const db = require("./config/keys").mongoURI;

// MangoDB connet

mongoose

.connect(

db,

{ useNewUrlParser: true }

)

.then(() => console.log("mongodb conneneted "))

.catch(err => console.log(err));

// app.get("/", (req, res) => res.send("hello Run Server"));

// Passport Middleware

app.use(passport.initialize());

// Passport config

require("./config/passport")(passport);

// User Routes

app.use("/api/users", users);

app.use("/api/profile", profile);

app.use("/api/posts", posts);

const port = process.env.PORT || 5000;

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

.config/passport

const JwtStrategy = require("passport-jwt").Strategy;

const ExtractJwt = require("passport-jwt").ExtractJwt;

const mongoose = require("mongoose");

const User = mongoose.model("users");

const Keys = require("../config/keys");

const opts = {};

opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();

opts.secretOrKey = Keys.secretOrKey;

module.exports = passport => {

passport.use(

new JwtStrategy(opts, (jwt\_payload, done) => {

User.findById(jwt\_payload.id)

.then(user => {

if (user) {

return done(null, user);

}

return done(null, false);

})

.catch(err => console.log(err));

})

);

};

User route

// @route GET api/users/current

// @desc Test Login router

// @access Pravite

router.get(

"/current",

passport.authenticate("jwt", { session: false }),

(req, res) => {

res.json({

id: req.user.id,

email: req.user.email,

name: req.user.name

});

}

);

module.exports = router;

Validate the values

Link: <https://github.com/chriso/validator.js/>

Create validate folder

Register.js

Login.js

Validate/Is-empty.js

const isEmpty = value =>

value === undefined ||

value === null ||

(typeof value === "object" && Object.keys(value).length === 0) ||

(typeof value === "string" && value.trim().length === 0);

module.exports = isEmpty;

validate/register

const Validator = require("validator");

const isEmpty = require("./is-empty");

module.exports = function validateRegisterInput(data) {

let errors = {};

data.name = !isEmpty(data.name) ? data.name : "";

data.email = !isEmpty(data.email) ? data.email : "";

data.password = !isEmpty(data.password) ? data.password : "";

data.password2 = !isEmpty(data.password2) ? data.password2 : "";

if (!Validator.isLength(data.name, { min: 2, max: 30 })) {

errors.name = "name must be between 2 and 30 characters";

}

if (Validator.isEmpty(data.name)) {

errors.name = "name field required";

}

if (Validator.isEmpty(data.email)) {

errors.email = "email field required";

}

if (!Validator.isEmail(data.email)) {

errors.email = "email invalid";

}

if (Validator.isEmpty(data.password)) {

errors.password = "password field requried";

}

if (!Validator.isLength(data.password, { min: 6, max: 30 })) {

errors.password = "password should be 6 to 30 charecterers";

}

if (Validator.isEmpty(data.password2)) {

errors.password2 = "Confirm password field requried";

}

if (!Validator.equals(data.password, data.password2)) {

errors.password2 = " password must match";

}

return {

errors,

isValid: isEmpty(errors)

};

};

User Route

// load input validation

const validateRegisterInput = require("../../validation/register");

const validateLoginInput = require("../../validation/login");

const { errors, isValid } = validateRegisterInput(req.body);

// Check validate

if (!isValid) {

return res.status(400).json(errors);

}

Validate/login

const Validator = require("validator");

const isEmpty = require("./is-empty");

module.exports = function validateLoginInput(data) {

let errors = {};

data.email = !isEmpty(data.email) ? data.email : "";

data.password = !isEmpty(data.password) ? data.password : "";

if (Validator.isEmpty(data.email)) {

errors.email = "email field required";

}

if (!Validator.isEmail(data.email)) {

errors.email = "email invalid";

}

if (Validator.isEmpty(data.password)) {

errors.password = "password field requried";

}

return {

errors,

isValid: isEmpty(errors)

};

};

Router/User.js

// @route GET api/users/login

// @desc Test Login router

// @access Public

router.post("/login", (req, res) => {

const email = req.body.email;

const password = req.body.password;

const { errors, isValid } = validateLoginInput(req.body);

if (!isValid) {

return res.status(400).json(errors);

}

// Find the user

User.findOne({ email }).then(user => {

if (!user) {

errors.email = "user not found";

return res.status(404).json(errors);

}

// chack password

bcrypt.compare(password, user.password).then(isMatch => {

if (isMatch) {

//User matched

const payload = { id: user.id, name: user.name, avatar: user.avatar };

//sign token

jwt.sign(

payload,

keys.secretOrKey,

{ expiresIn: 3600 },

(err, token) => {

res.json({

success: true,

token: "Bearer " + token,

expiresIn: 3600,

payload: payload

});

}

);

} else {

errors.password = "not mached";

return res.status(404).json(errors);

}

});

});

});

Profile model

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

// Create Schema

const ProfileSchema = new Schema({

user: {

type: Schema.Types.ObjectId,

ref: 'users'

},

handle: {

type: String,

required: true,

max: 40

},

company: {

type: String

},

website: {

type: String

},

location: {

type: String

},

status: {

type: String,

required: true

},

skills: {

type: [String],

required: true

},

bio: {

type: String

},

githubusername: {

type: String

},

experience: [

{

title: {

type: String,

required: true

},

company: {

type: String,

required: true

},

location: {

type: String

},

from: {

type: Date,

required: true

},

to: {

type: Date

},

current: {

type: Boolean,

default: false

},

description: {

type: String

}

}

],

education: [

{

school: {

type: String,

required: true

},

degree: {

type: String,

required: true

},

fieldofstudy: {

type: String,

required: true

},

from: {

type: Date,

required: true

},

to: {

type: Date

},

current: {

type: Boolean,

default: false

},

description: {

type: String

}

}

],

social: {

youtube: {

type: String

},

twitter: {

type: String

},

facebook: {

type: String

},

linkedin: {

type: String

},

instagram: {

type: String

}

},

date: {

type: Date,

default: Date.now

}

});

module.exports = Profile = mongoose.model('profile', ProfileSchema);

api/profile router

const express = require("express");

const router = express.Router();

const mongoose = require("mongoose");

const passport = require("passport");

// Load Profile Model

const Profile = require("../../models/Profile");

//Load USer Moidel

const User = require("../../models/User");

// @route GET api/profile

// @desc get current Profile user

// @access private

router.get(

"/",

passport.authenticate("jwt", { session: false }),

(req, res) => {

const errors = {};

Profile.findOne({ user: req.user.id })

.then(profile => {

if (!profile) {

errors.noprofile = "there is no profile";

return res.status(404).json(errors);

}

res.json(profile);

})

.catch(err => res.status(404).json(err));

}

);

// @route Post api/profile

// @desc Create Profile user

// @access private

router.post(

"/",

passport.authenticate("jwt", { session: false }),

(req, res) => {

// Get fields

const profileFields = {};

profileFields.user = req.user.id;

if (req.body.handle) profileFields.handle = req.body.handle;

if (req.body.company) profileFields.company = req.body.company;

if (req.body.website) profileFields.website = req.body.website;

if (req.body.location) profileFields.location = req.body.location;

if (req.body.bio) profileFields.bio = req.body.bio;

if (req.body.status) profileFields.status = req.body.status;

if (req.body.githubusername)

profileFields.githubusername = req.body.githubusername;

// Skills - Spilt into array

if (typeof req.body.skills !== "undefined") {

profileFields.skills = req.body.skills.split(",");

}

// Social

profileFields.social = {};

if (req.body.youtube) profileFields.social.youtube = req.body.youtube;

if (req.body.twitter) profileFields.social.twitter = req.body.twitter;

if (req.body.facebook) profileFields.social.facebook = req.body.facebook;

if (req.body.linkedin) profileFields.social.linkedin = req.body.linkedin;

if (req.body.instagram) profileFields.social.instagram = req.body.instagram;

Profile.findOne({ user: req.user.id }).then(profile => {

if (profile) {

//Update

Profile.findByIdAndUpdate(

{

user: req.user.id

},

{ $set: profileFields },

{

new: true

}

).then(profile => res.json(profile));

} else {

//Create

// check handle

Profile.findOne({ handle: profileFields.handle }).then(profile => {

if (profile) {

errors.handle = "handle already exis";

res.status(404).json(errors);

}

//Save Profile

new Profile(profileFields).save().then(profile => res.json(profile));

});

}

});

}

);

// @route GET api/profile/test

// @desc Test Profile router

// @access Public

router.get("/test", (req, res) => res.json({ msg: "Profile Work" }));

module.exports = router;

validation part for profile user

validator/profile

const Validator = require("validator");

const isEmpty = require("./is-empty");

module.exports = function validateProfileInput(data) {

let errors = {};

data.handle = !isEmpty(data.handle) ? data.handle : "";

data.status = !isEmpty(data.status) ? data.status : "";

data.skills = !isEmpty(data.skills) ? data.skills : "";

if (!Validator.isLength(data.handle, { min: 2, max: 40 })) {

errors.handle = "Handle needs to between 2 and 4 characters";

}

if (Validator.isEmpty(data.handle)) {

errors.handle = "Profile handle is required";

}

if (Validator.isEmpty(data.status)) {

errors.status = "Status field is required";

}

if (Validator.isEmpty(data.skills)) {

errors.skills = "Skills field is required";

}

if (!isEmpty(data.website)) {

if (!Validator.isURL(data.website)) {

errors.website = "Not a valid URL";

}

}

if (!isEmpty(data.youtube)) {

if (!Validator.isURL(data.youtube)) {

errors.youtube = "Not a valid URL";

}

}

if (!isEmpty(data.twitter)) {

if (!Validator.isURL(data.twitter)) {

errors.twitter = "Not a valid URL";

}

}

if (!isEmpty(data.facebook)) {

if (!Validator.isURL(data.facebook)) {

errors.facebook = "Not a valid URL";

}

}

if (!isEmpty(data.linkedin)) {

if (!Validator.isURL(data.linkedin)) {

errors.linkedin = "Not a valid URL";

}

}

if (!isEmpty(data.instagram)) {

if (!Validator.isURL(data.instagram)) {

errors.instagram = "Not a valid URL";

}

}

return {

errors,

isValid: isEmpty(errors)

};

};

Api/profile

const express = require("express");

const router = express.Router();

const mongoose = require("mongoose");

const passport = require("passport");

// Load Validation

const validateProfileInput = require("../../validation/profile");

// Load Profile Model

const Profile = require("../../models/Profile");

// Load User Model

const User = require("../../models/User");

// @route GET api/profile/test

// @desc Tests profile route

// @access Public

router.get("/test", (req, res) => res.json({ msg: "Profile Works" }));

// @route GET api/profile

// @desc Get current users profile

// @access Private

router.get(

"/",

passport.authenticate("jwt", { session: false }),

(req, res) => {

const errors = {};

Profile.findOne({ user: req.user.id })

.populate("user", ["name", "avatar"])

.then(profile => {

if (!profile) {

errors.noprofile = "There is no profile for this user";

return res.status(404).json(errors);

}

res.json(profile);

})

.catch(err => res.status(404).json(err));

}

);

// @route GET api/profile/all

// @desc Get all profiles

// @access Public

router.get("/all", (req, res) => {

const errors = {};

Profile.find()

.populate("user", ["name", "avatar"])

.then(profiles => {

if (!profiles) {

errors.noprofile = "There are no profiles";

return res.status(404).json(errors);

}

res.json(profiles);

})

.catch(err => res.status(404).json({ profile: "There are no profiles" }));

});

// @route GET api/profile/handle/:handle

// @desc Get profile by handle

// @access Public

router.get("/handle/:handle", (req, res) => {

const errors = {};

Profile.findOne({ handle: req.params.handle })

.populate("user", ["name", "avatar"])

.then(profile => {

if (!profile) {

errors.noprofile = "There is no profile for this user";

res.status(404).json(errors);

}

res.json(profile);

})

.catch(err => res.status(404).json(err));

});

// @route GET api/profile/user/:user\_id

// @desc Get profile by user ID

// @access Public

router.get("/user/:user\_id", (req, res) => {

const errors = {};

Profile.findOne({ user: req.params.user\_id })

.populate("user", ["name", "avatar"])

.then(profile => {

if (!profile) {

errors.noprofile = "There is no profile for this user";

res.status(404).json(errors);

}

res.json(profile);

})

.catch(err =>

res.status(404).json({ profile: "There is no profile for this user" })

);

});

// @route POST api/profile

// @desc Create or edit user profile

// @access Private

router.post(

"/",

passport.authenticate("jwt", { session: false }),

(req, res) => {

const { errors, isValid } = validateProfileInput(req.body);

// Check Validation

if (!isValid) {

// Return any errors with 400 status

return res.status(400).json(errors);

}

// Get fields

const profileFields = {};

profileFields.user = req.user.id;

if (req.body.handle) profileFields.handle = req.body.handle;

if (req.body.company) profileFields.company = req.body.company;

if (req.body.website) profileFields.website = req.body.website;

if (req.body.location) profileFields.location = req.body.location;

if (req.body.bio) profileFields.bio = req.body.bio;

if (req.body.status) profileFields.status = req.body.status;

if (req.body.githubusername)

profileFields.githubusername = req.body.githubusername;

// Skills - Spilt into array

if (typeof req.body.skills !== "undefined") {

profileFields.skills = req.body.skills.split(",");

}

// Social

profileFields.social = {};

if (req.body.youtube) profileFields.social.youtube = req.body.youtube;

if (req.body.twitter) profileFields.social.twitter = req.body.twitter;

if (req.body.facebook) profileFields.social.facebook = req.body.facebook;

if (req.body.linkedin) profileFields.social.linkedin = req.body.linkedin;

if (req.body.instagram) profileFields.social.instagram = req.body.instagram;

Profile.findOne({ user: req.user.id }).then(profile => {

if (profile) {

// Update

Profile.findOneAndUpdate(

{ user: req.user.id },

{ $set: profileFields },

{ new: true }

).then(profile => res.json(profile));

} else {

// Create

// Check if handle exists

Profile.findOne({ handle: profileFields.handle }).then(profile => {

if (profile) {

errors.handle = "That handle already exists";

res.status(400).json(errors);

}

// Save Profile

new Profile(profileFields).save().then(profile => res.json(profile));

});

}

});

}

);

module.exports = router;

Validator/education

const Validator = require("validator");

const isEmpty = require("./is-empty");

module.exports = function validateExperienceInput(data) {

let errors = {};

data.school = !isEmpty(data.school) ? data.school : "";

data.degree = !isEmpty(data.degree) ? data.degree : "";

data.fieldofstudy = !isEmpty(data.fieldofstudy) ? data.fieldofstudy : "";

data.from = !isEmpty(data.from) ? data.from : "";

if (Validator.isEmpty(data.school)) {

errors.school = "School field is required";

}

if (Validator.isEmpty(data.degree)) {

errors.degree = "Degree field is required";

}

if (Validator.isEmpty(data.fieldofstudy)) {

errors.fieldofstudy = "Field of study field is required";

}

if (Validator.isEmpty(data.from)) {

errors.from = "From date field is required";

}

return {

errors,

isValid: isEmpty(errors)

};

};

Validator/experience

const Validator = require("validator");

const isEmpty = require("./is-empty");

module.exports = function validateExperienceInput(data) {

let errors = {};

data.title = !isEmpty(data.title) ? data.title : "";

data.company = !isEmpty(data.company) ? data.company : "";

data.from = !isEmpty(data.from) ? data.from : "";

if (Validator.isEmpty(data.title)) {

errors.title = "Job title field is required";

}

if (Validator.isEmpty(data.company)) {

errors.company = "Company field is required";

}

if (Validator.isEmpty(data.from)) {

errors.from = "From date field is required";

}

return {

errors,

isValid: isEmpty(errors)

};

};

Api/profile

const express = require("express");

const router = express.Router();

const mongoose = require("mongoose");

const passport = require("passport");

// Load Validation

const validateProfileInput = require("../../validation/profile");

const validateExperienceInput = require("../../validation/experience");

const validateEducationInput = require("../../validation/education");

// Load Profile Model

const Profile = require("../../models/Profile");

// Load User Model

const User = require("../../models/User");

// @route GET api/profile/test

// @desc Tests profile route

// @access Public

router.get("/test", (req, res) => res.json({ msg: "Profile Works" }));

// @route GET api/profile

// @desc Get current users profile

// @access Private

router.get(

"/",

passport.authenticate("jwt", { session: false }),

(req, res) => {

const errors = {};

Profile.findOne({ user: req.user.id })

.populate("user", ["name", "avatar"])

.then(profile => {

if (!profile) {

errors.noprofile = "There is no profile for this user";

return res.status(404).json(errors);

}

res.json(profile);

})

.catch(err => res.status(404).json(err));

}

);

// @route GET api/profile/all

// @desc Get all profiles

// @access Public

router.get("/all", (req, res) => {

const errors = {};

Profile.find()

.populate("user", ["name", "avatar"])

.then(profiles => {

if (!profiles) {

errors.noprofile = "There are no profiles";

return res.status(404).json(errors);

}

res.json(profiles);

})

.catch(err => res.status(404).json({ profile: "There are no profiles" }));

});

// @route GET api/profile/handle/:handle

// @desc Get profile by handle

// @access Public

router.get("/handle/:handle", (req, res) => {

const errors = {};

Profile.findOne({ handle: req.params.handle })

.populate("user", ["name", "avatar"])

.then(profile => {

if (!profile) {

errors.noprofile = "There is no profile for this user";

res.status(404).json(errors);

}

res.json(profile);

})

.catch(err => res.status(404).json(err));

});

// @route GET api/profile/user/:user\_id

// @desc Get profile by user ID

// @access Public

router.get("/user/:user\_id", (req, res) => {

const errors = {};

Profile.findOne({ user: req.params.user\_id })

.populate("user", ["name", "avatar"])

.then(profile => {

if (!profile) {

errors.noprofile = "There is no profile for this user";

res.status(404).json(errors);

}

res.json(profile);

})

.catch(err =>

res.status(404).json({ profile: "There is no profile for this user" })

);

});

// @route POST api/profile

// @desc Create or edit user profile

// @access Private

router.post(

"/",

passport.authenticate("jwt", { session: false }),

(req, res) => {

const { errors, isValid } = validateProfileInput(req.body);

// Check Validation

if (!isValid) {

// Return any errors with 400 status

return res.status(400).json(errors);

}

// Get fields

const profileFields = {};

profileFields.user = req.user.id;

if (req.body.handle) profileFields.handle = req.body.handle;

if (req.body.company) profileFields.company = req.body.company;

if (req.body.website) profileFields.website = req.body.website;

if (req.body.location) profileFields.location = req.body.location;

if (req.body.bio) profileFields.bio = req.body.bio;

if (req.body.status) profileFields.status = req.body.status;

if (req.body.githubusername)

profileFields.githubusername = req.body.githubusername;

// Skills - Spilt into array

if (typeof req.body.skills !== "undefined") {

profileFields.skills = req.body.skills.split(",");

}

// Social

profileFields.social = {};

if (req.body.youtube) profileFields.social.youtube = req.body.youtube;

if (req.body.twitter) profileFields.social.twitter = req.body.twitter;

if (req.body.facebook) profileFields.social.facebook = req.body.facebook;

if (req.body.linkedin) profileFields.social.linkedin = req.body.linkedin;

if (req.body.instagram) profileFields.social.instagram = req.body.instagram;

Profile.findOne({ user: req.user.id }).then(profile => {

if (profile) {

// Update

Profile.findOneAndUpdate(

{ user: req.user.id },

{ $set: profileFields },

{ new: true }

).then(profile => res.json(profile));

} else {

// Create

// Check if handle exists

Profile.findOne({ handle: profileFields.handle }).then(profile => {

if (profile) {

errors.handle = "That handle already exists";

res.status(400).json(errors);

}

// Save Profile

new Profile(profileFields).save().then(profile => res.json(profile));

});

}

});

}

);

// @route POST api/profile/experience

// @desc Add experience to profile

// @access Private

router.post(

"/experience",

passport.authenticate("jwt", { session: false }),

(req, res) => {

const { errors, isValid } = validateExperienceInput(req.body);

// Check Validation

if (!isValid) {

// Return any errors with 400 status

return res.status(400).json(errors);

}

Profile.findOne({ user: req.user.id }).then(profile => {

const newExp = {

title: req.body.title,

company: req.body.company,

location: req.body.location,

from: req.body.from,

to: req.body.to,

current: req.body.current,

description: req.body.description

};

// Add to exp array

profile.experience.unshift(newExp);

profile.save().then(profile => res.json(profile));

});

}

);

// @route POST api/profile/education

// @desc Add education to profile

// @access Private

router.post(

"/education",

passport.authenticate("jwt", { session: false }),

(req, res) => {

const { errors, isValid } = validateEducationInput(req.body);

// Check Validation

if (!isValid) {

// Return any errors with 400 status

return res.status(400).json(errors);

}

Profile.findOne({ user: req.user.id }).then(profile => {

const newEdu = {

school: req.body.school,

degree: req.body.degree,

fieldofstudy: req.body.fieldofstudy,

from: req.body.from,

to: req.body.to,

current: req.body.current,

description: req.body.description

};

// Add to exp array

profile.education.unshift(newEdu);

profile.save().then(profile => res.json(profile));

});

}

);

// @route DELETE api/profile/experience/:exp\_id

// @desc Delete experience from profile

// @access Private

router.delete(

"/experience/:exp\_id",

passport.authenticate("jwt", { session: false }),

(req, res) => {

Profile.findOne({ user: req.user.id })

.then(profile => {

// Get remove index

const removeIndex = profile.experience

.map(item => item.id)

.indexOf(req.params.exp\_id);

// Splice out of array

profile.experience.splice(removeIndex, 1);

// Save

profile.save().then(profile => res.json(profile));

})

.catch(err => res.status(404).json(err));

}

);

// @route DELETE api/profile/education/:edu\_id

// @desc Delete education from profile

// @access Private

router.delete(

"/education/:edu\_id",

passport.authenticate("jwt", { session: false }),

(req, res) => {

Profile.findOne({ user: req.user.id })

.then(profile => {

// Get remove index

const removeIndex = profile.education

.map(item => item.id)

.indexOf(req.params.edu\_id);

// Splice out of array

profile.education.splice(removeIndex, 1);

// Save

profile.save().then(profile => res.json(profile));

})

.catch(err => res.status(404).json(err));

}

);

// @route DELETE api/profile

// @desc Delete user and profile

// @access Private

router.delete(

"/",

passport.authenticate("jwt", { session: false }),

(req, res) => {

Profile.findOneAndRemove({ user: req.user.id }).then(() => {

User.findOneAndRemove({ \_id: req.user.id }).then(() =>

res.json({ success: true })

);

});

}

);

module.exports = router;

/model/post model

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

// Create Schema

const PostSchema = new Schema({

user: {

type: Schema.Types.ObjectId,

ref: 'users'

},

text: {

type: String,

required: true

},

name: {

type: String

},

avatar: {

type: String

},

likes: [

{

user: {

type: Schema.Types.ObjectId,

ref: 'users'

}

}

],

comments: [

{

user: {

type: Schema.Types.ObjectId,

ref: 'users'

},

text: {

type: String,

required: true

},

name: {

type: String

},

avatar: {

type: String

},

date: {

type: Date,

default: Date.now

}

}

],

date: {

type: Date,

default: Date.now

}

});

module.exports = Post = mongoose.model('post', PostSchema);

api/post

const express = require('express');

const router = express.Router();

const mongoose = require('mongoose');

const passport = require('passport');

// Post model

const Post = require('../../models/Post');

// Profile model

const Profile = require('../../models/Profile');

// Validation

const validatePostInput = require('../../validation/post');

// @route GET api/posts/test

// @desc Tests post route

// @access Public

router.get('/test', (req, res) => res.json({ msg: 'Posts Works' }));

// @route GET api/posts

// @desc Get posts

// @access Public

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

Post.find()

.sort({ date: -1 })

.then(posts => res.json(posts))

.catch(err => res.status(404).json({ nopostsfound: 'No posts found' }));

});

// @route GET api/posts/:id

// @desc Get post by id

// @access Public

router.get('/:id', (req, res) => {

Post.findById(req.params.id)

.then(post => res.json(post))

.catch(err =>

res.status(404).json({ nopostfound: 'No post found with that ID' })

);

});

// @route POST api/posts

// @desc Create post

// @access Private

router.post(

'/',

passport.authenticate('jwt', { session: false }),

(req, res) => {

const { errors, isValid } = validatePostInput(req.body);

// Check Validation

if (!isValid) {

// If any errors, send 400 with errors object

return res.status(400).json(errors);

}

const newPost = new Post({

text: req.body.text,

name: req.body.name,

avatar: req.body.avatar,

user: req.user.id

});

newPost.save().then(post => res.json(post));

}

);

// @route DELETE api/posts/:id

// @desc Delete post

// @access Private

router.delete(

'/:id',

passport.authenticate('jwt', { session: false }),

(req, res) => {

Profile.findOne({ user: req.user.id }).then(profile => {

Post.findById(req.params.id)

.then(post => {

// Check for post owner

if (post.user.toString() !== req.user.id) {

return res

.status(401)

.json({ notauthorized: 'User not authorized' });

}

// Delete

post.remove().then(() => res.json({ success: true }));

})

.catch(err => res.status(404).json({ postnotfound: 'No post found' }));

});

}

);

// @route POST api/posts/like/:id

// @desc Like post

// @access Private

router.post(

'/like/:id',

passport.authenticate('jwt', { session: false }),

(req, res) => {

Profile.findOne({ user: req.user.id }).then(profile => {

Post.findById(req.params.id)

.then(post => {

if (

post.likes.filter(like => like.user.toString() === req.user.id)

.length > 0

) {

return res

.status(400)

.json({ alreadyliked: 'User already liked this post' });

}

// Add user id to likes array

post.likes.unshift({ user: req.user.id });

post.save().then(post => res.json(post));

})

.catch(err => res.status(404).json({ postnotfound: 'No post found' }));

});

}

);

// @route POST api/posts/unlike/:id

// @desc Unlike post

// @access Private

router.post(

'/unlike/:id',

passport.authenticate('jwt', { session: false }),

(req, res) => {

Profile.findOne({ user: req.user.id }).then(profile => {

Post.findById(req.params.id)

.then(post => {

if (

post.likes.filter(like => like.user.toString() === req.user.id)

.length === 0

) {

return res

.status(400)

.json({ notliked: 'You have not yet liked this post' });

}

// Get remove index

const removeIndex = post.likes

.map(item => item.user.toString())

.indexOf(req.user.id);

// Splice out of array

post.likes.splice(removeIndex, 1);

// Save

post.save().then(post => res.json(post));

})

.catch(err => res.status(404).json({ postnotfound: 'No post found' }));

});

}

);

// @route POST api/posts/comment/:id

// @desc Add comment to post

// @access Private

router.post(

'/comment/:id',

passport.authenticate('jwt', { session: false }),

(req, res) => {

const { errors, isValid } = validatePostInput(req.body);

// Check Validation

if (!isValid) {

// If any errors, send 400 with errors object

return res.status(400).json(errors);

}

Post.findById(req.params.id)

.then(post => {

const newComment = {

text: req.body.text,

name: req.body.name,

avatar: req.body.avatar,

user: req.user.id

};

// Add to comments array

post.comments.unshift(newComment);

// Save

post.save().then(post => res.json(post));

})

.catch(err => res.status(404).json({ postnotfound: 'No post found' }));

}

);

// @route DELETE api/posts/comment/:id/:comment\_id

// @desc Remove comment from post

// @access Private

router.delete(

'/comment/:id/:comment\_id',

passport.authenticate('jwt', { session: false }),

(req, res) => {

Post.findById(req.params.id)

.then(post => {

// Check to see if comment exists

if (

post.comments.filter(

comment => comment.\_id.toString() === req.params.comment\_id

).length === 0

) {

return res

.status(404)

.json({ commentnotexists: 'Comment does not exist' });

}

// Get remove index

const removeIndex = post.comments

.map(item => item.\_id.toString())

.indexOf(req.params.comment\_id);

// Splice comment out of array

post.comments.splice(removeIndex, 1);

post.save().then(post => res.json(post));

})

.catch(err => res.status(404).json({ postnotfound: 'No post found' }));

}

);

module.exports = router;

validator/post

const Validator = require('validator');

const isEmpty = require('./is-empty');

module.exports = function validatePostInput(data) {

let errors = {};

data.text = !isEmpty(data.text) ? data.text : '';

if (!Validator.isLength(data.text, { min: 10, max: 300 })) {

errors.text = 'Post must be between 10 and 300 characters';

}

if (Validator.isEmpty(data.text)) {

errors.text = 'Text field is required';

}

return {

errors,

isValid: isEmpty(errors)

};

};

**REACT FRONTEND**

**Npm I –g create-react-app**

* **npm i –g npm for(update urn pm )**
* **axios for getting backend data**
* **package.js**
* "proxy": "http://localhost:5000"
* **For backend api**

**One commed run both backend and frontend**

* Npm I concurrently
* Server ->package.json
* "scripts": {
* "client-install": "npm install --prefix client",
* "start": "node server.js",
* "server": "nodemon server.js",
* "client": "npm start --prefix client",
* "dev": "concurrently \"npm run server\" \"npm run client\""
* },

Public/index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<meta name="theme-color" content="#000000">

<link rel="manifest" href="%PUBLIC\_URL%/manifest.json">

<link rel="shortcut icon" href="%PUBLIC\_URL%/favicon.ico">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.2/css/bootstrap.min.css" integrity="sha384-Smlep5jCw/wG7hdkwQ/Z5nLIefveQRIY9nfy6xoR1uRYBtpZgI6339F5dgvm/e9B"

crossorigin="anonymous">

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" integrity="sha384-hWVjflwFxL6sNzntih27bfxkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54bOGbiDQ"

crossorigin="anonymous">

<title>React Mern : A Social Network For Developers</title>

</head>

<body>

<noscript>

You need to enable JavaScript to run this app.

</noscript>

<div id="root"></div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js" integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js" integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"

crossorigin="anonymous"></script>

</body>

</html>

**Create component folder**

**Src/component/layout/Footer.js**

import React from "react";

export default () => {

return (

<footer className="bg-myclr text-center text-white p-4 mt-5">

Copyright &copy; {new Date().getFullYear()} Sandhani shaik

</footer>

);

};

**Header.js**

import React, { Component } from "react";

class Header extends Component {

render() {

return (

<nav className="navbar navbar-expand-sm navbar-dark bg-myclr mb-4">

<div className="container">

<a className="navbar-brand" href="landing.html">

DevContainers

</a>

<button

className="navbar-toggler"

type="button"

data-toggle="collapse"

data-target="#mobile-nav"

>

<span className="navbar-toggler-icon" />

</button>

<div className="collapse navbar-collapse" id="mobile-nav">

<ul className="navbar-nav mr-auto">

<li className="nav-item">

<a className="nav-link" href="profiles.html">

{" "}

Developers

</a>

</li>

</ul>

<ul className="navbar-nav ml-auto">

<li className="nav-item">

<a className="nav-link" href="register.html">

Sign Up

</a>

</li>

<li className="nav-item">

<a className="nav-link" href="login.html">

Login

</a>

</li>

</ul>

</div>

</div>

</nav>

);

}

}

export default Header;

**Landing.js**

import React, { Component } from "react";

class Landing extends Component {

render() {

return (

<div className="landing">

<div className="dark-overlay landing-inner text-light">

<div className="container">

<div className="row">

<div className="col-md-12 text-center">

<h1 className="display-3 mb-4">Developer Connector</h1>

<p className="lead">

{" "}

Create a developer profile/portfolio, share posts and get help

from other developers

</p>

<hr />

<a href="register.html" className="btn btn-lg btn-info mr-2">

Sign Up

</a>

<a href="login.html" className="btn btn-lg btn-light">

Login

</a>

</div>

</div>

</div>

</div>

</div>

);

}

}

export default Landing;

**Result App.js**

import React, { Component } from "react";

import Header from "./components/layout/Header";

import Footer from "./components/layout/Footer";

import Landing from "./components/layout/Landing";

import "./App.css";

class App extends Component {

render() {

return (

<div className="App">

<Header />

<Landing />

<Footer />

</div>

);

}

}

export default App;

**07 React Router Componet state**