config/DB.js

```
module.exports = {
DB: "mongodb://localhost:27017/reactcrud",
"jwtSecret": "sl_myJwtSecret"
};
```

servers.js

```
const express = require('express');
const mongoose = require('mongoose');
const path = require('path');
const config = require('config');
const app = express();
// Bodyparser Middleware
app.use(express.json());
// DB Config
const configs = require('./config/DB');
// Connect to Mongo
mongoose.Promise = global.Promise;
mongoose.connect(configs.DB, { useNewUrlParser: true }).then(
() => {console.log('Database is connected') },
err => { console.log('Can not connect to the database'+ err)}
);
// Use Routes
app.use('/api/items', require('./routes/api/items'));
app.use('/api/users', require('./routes/api/users'));
app.use('/api/auth', require('./routes/api/auth'));
// Serve static assets if in production
if (process.env.NODE_ENV === 'production') {
// Set static folder
app.use(express.static('client/build'));
app.get('*', (req, res) => {
res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'));
});
}

const port = process.env.PORT || 4000;
app.listen(port, () => console.log(Server started on port {port}));
```

middleware/auth

```
const config = require('config');
const jwt = require('jsonwebtoken');
function auth(req, res, next) {
const token = req.header('x-auth-token');
// Check for token
if (!token)
return res.status(401).json({ msg: 'No token, authorizaton denied' });
try {
// Verify token
const decoded = jwt.verify(token, config.get('jwtSecret'));
// Add user from payload
req.user = decoded;
next();
}
catch (e) {
res.status(400).json({ msg: 'Token is not valid' });
}
}
```

```
module.exports = auth;
```

modelsItem.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
// Create Schema
const ItemSchema = new Schema({
name: {
type: String,
required: true
},
date: {
type: Date,
default: Date.now
}
});

module.exports = Item = mongoose.model('item', ItemSchema);
```

Users.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
// Create Schema
const UserSchema = new Schema({
name: {
type: String,
required: true
},
email: {
type: String,
required: true,
unique: true
},
password: {
type: String,
required: true
},
register_date: {
type: Date,
default: Date.now
}
});

module.exports = User = mongoose.model('user', UserSchema);
```

devConnector*models/postDevConnector.js*

```
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);
```

ProfilesDevConnector.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const User = mongoose.model('users');
// Create Schema
const ProfileSchema = new Schema({
user: {
type: Schema.Types.ObjectId,
ref: 'users' //model->User ->user
},
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);
```

usersDevConnector.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
//Create Schema
const UserSchema = new Schema ({
name: {
type:String,
required:true
},
email: {
type:String,
required:true
},
password: {
type:String,
required:true
},
avatar: {
type:String,
},
date: {
type:Date,
default:Date.now
}
});

module.exports = User = mongoose.model('users', UserSchema);
```

routes*routes/api/auth*

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const config = require('config');
const jwt = require('jsonwebtoken');
const auth = require('../../middleware/auth');
const User = require('../../models/User');

router.post('/', (req, res) => {
const { email, password } = req.body;
// Simple validation
if(!email || !password) {
return res.status(400).json({ msg: 'Please enter all fields' });
}
// Check for existing user
User.findOne({ email })
.then(user => {
if(!user) return res.status(400).json({ msg: 'User Does not exist' });
// Validate password
bcrypt.compare(password, user.password)
.then(isMatch => {
```

```
if(!isMatch) return res.status(400).json({ msg: 'Invalid credentials' });
jwt.sign(
{ id: user.id },
config.get('jwtSecret'),
{ expiresIn: 3600 },
(err, token) => {
if(err) throw err;
res.json({
token,
user: {
id: user.id,
name: user.name,
email: user.email
}
});
}
)
})
})
});

router.get('/user', auth, (req, res) => {
User.findById(req.user.id)
.select('-password')
.then(user => res.json(user));
});

module.exports = router;
```

routes/api/items

```
const express = require('express');
const router = express.Router();
const auth = require('../../middleware/auth');
// Item Model
const Item = require('../../models/Item');
router.get('/', (req, res) => {
Item.find()
.sort({ date: -1 })
.then(items => res.json(items));
});

router.post('/', auth, (req, res) => {
const newItem = new Item({
name: req.body.name
});

newItem.save().then(item => res.json(item));
});
router.delete('/:id', auth, (req, res) => {
Item.findById(req.params.id)
.then(item => item.remove().then(() => res.json({ success: true })))
.catch(err => res.status(404).json({ success: false }));
});

module.exports = router;
```

routes/api/users

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const config = require('config');
const jwt = require('jsonwebtoken');

// User Model
const User = require('../../models/User');
router.post('/', (req, res) => {
const { name, email, password } = req.body;
// Simple validation
if(!name || !email || !password) {
return res.status(400).json({ msg: 'Please enter all fields' });
}
// Check for existing user
User.findOne({ email })
.then(user => {
if(user) return res.status(400).json({ msg: 'User already exists' });
const newUser = new User({
name,
email,
password
});
// Create salt & hash
bcrypt.genSalt(10, (err, salt) => {
bcrypt.hash(newUser.password, salt, (err, hash) => {
if(err) throw err;
newUser.password = hash;
newUser.save()
.then(user => {
jwt.sign(
{ id: user.id },
config.get('jwtSecret'),
{ expiresIn: 3600 },
(err, token) => {
if(err) throw err;
res.json({
token,
user: {
id: user.id,
name: user.name,
email: user.email
}
});
}
)
});
})
})
})
});

module.exports = router;
```

routes/DevConnector/posts

```
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');
const passport = require('passport');
// Load Post model
const Post = require('../../models/Post');
// Load 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/post/: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 a 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 a 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._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;
```

routes/DevConnector/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 Profile';
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 edit users 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 - split into array
if (typeof req.body.skills !== 'undefined') {
profileFields.skills = req.body.skills.split(',');
}
//Social meida
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 experience array
profile.experience.unshift(newExp);
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 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 education array
profile.education.unshift(newEdu);
profile.save()
.then(profile =>res.json(profile));
})
});
//@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 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;
```

routes/DevConnector/uses

```
const express = require('express');
const router = express.Router();
const gravatar = require('gravatar');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const keys = require('../../config/keys');
const passport = require('passport');
//Load input Validation
const validateRegisterInput = require('../../validation/register');
const validateLoginInput = require('../../validation/login');
//Load User model
const User = require('../../models/User');
//@route GET api/users/test
//@desc Tests users route
//@access Public
router.get('/test', (req, res) => {
res.json({
msg: 'User Works'
})
});
```

```
//@route POST api/users/register
//@desc Register user
//@access Public
router.post('/register', (req, res) => {
const { errors, isValid } = validateRegisterInput(req.body);
//Check Validation
if(!isValid){
return res.status(400).json(errors);
}
User.findOne({
email: req.body.email
})
.then(user => {
if (user) {
errors.email = 'Email already exists';
return res.status(400).json(errors);
} else {
const avatar = gravatar.url(req.body.email, {
s: '200', //Size
r: 'pg', //Rating
d: 'mm' //Default
});
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 POST api/users/login
//@desc Login User / Returnin JWT Token
//@access Public
router.post('/login', (req,res) =>{
const { errors, isValid } = validateLoginInput(req.body);
//Check Validation
if(!isValid){
return res.status(400).json(errors);
}
const email = req.body.email;
const password = req.body.password;
//Find user by email
User.findOne({email})
.then(user => {
//Check for user
if(!user){
errors.email ='User not found';
return res.status(404).json(errors);
}
//Check 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
});
});
}else {
errors.password='Password incorrect';
return res.status(400).json(errors);
}
});
});
});
//@route GET api/users/current
//@desc Return current user
//@access Privte
router.get('/current', passport.authenticate('jwt', { session:false}), (req, res)=>{
res.json({
id:req.user.id,
name:req.user.name,
email:req.user.email
});
});

module.exports = router;
```

authActions.js

```
import axios from 'axios';
import { returnErrors } from './errorActions';
import { USER_LOADED, USER_LOADING, AUTH_ERROR, LOGIN_SUCCESS, LOGIN_FAIL, LOGOUT_SUCCESS,
REGISTER_SUCCESS, REGISTER_FAIL } from './types';
// Check token & load user
export const loadUser = () => (dispatch, getState) => {
// User loading
dispatch({ type: USER_LOADING });
axios .get('/api/auth/user', tokenConfig(getState))
.then(res =>
dispatch({
type: USER_LOADED,
payload: res.data
})
)
.catch(err => {
dispatch(returnErrors(err.response.data, err.response.status));
dispatch({
type: AUTH_ERROR
});
});
};

// Register User
export const register = ({ name, email, password }) => dispatch => {
// Headers
const config = {
headers: {
'Content-Type': 'application/json'
}
};
// Request body
const body = JSON.stringify({ name, email, password });
axios .post('/api/users', body, config)
.then(res =>
dispatch({
type: REGISTER_SUCCESS,
payload: res.data
})
)
.catch(err => {
dispatch(
returnErrors(err.response.data, err.response.status, 'REGISTER_FAIL')
);
dispatch({
type: REGISTER_FAIL
});
});
};

// Login User
export const login = ({ email, password }) => dispatch => {
// Headers
const config = {
headers: {
'Content-Type': 'application/json'
}
};

// Request body
const body = JSON.stringify({ email, password });
axios .post('/api/auth', body, config)
.then(res =>
dispatch({
type: LOGIN_SUCCESS,
payload: res.data
})
)
.catch(err => {
dispatch(
returnErrors(err.response.data, err.response.status, 'LOGIN_FAIL')
);
dispatch({
type: LOGIN_FAIL
});
});
};

// Logout User
export const logout = () => {
return {
type: LOGOUT_SUCCESS
};
};

// Setup config/headers and token
export const tokenConfig = getState => {
// Get token from localstorage
const token = getState().auth.token;

// Headers
const config = {
headers: {
'Content-type': 'application/json'
}
};

// If token, add to headers
if (token) {
config.headers['x-auth-token'] = token;
}
return config;
};
```

errorActions.js

```
import { GET_ERRORS, CLEAR_ERRORS } from './types';

// RETURN ERRORS
export const returnErrors = (msg, status, id = null) => {
return {
type: GET_ERRORS,
payload: { msg, status, id }
};
};

// CLEAR ERRORS
export const clearErrors = () => {
return {
type: CLEAR_ERRORS
};
};
```

itemActions.js

```
import axios from 'axios';
import { GET_ITEMS, ADD_ITEM, DELETE_ITEM, ITEMS_LOADING } from './types';
import { tokenConfig } from './authActions';
import { returnErrors } from './errorActions';

export const getItems = () => dispatch => {
dispatch(setItemsLoading());
axios .get('/api/items')
.then(res =>
dispatch({
type: GET_ITEMS,
payload: res.data
})
)
.catch(err =>
dispatch(returnErrors(err.response.data, err.response.status))
);
};

export const addItem = item => (dispatch, getState) => {
axios .post('/api/items', item, tokenConfig(getState))
.then(res =>
dispatch({
type: ADD_ITEM,
payload: res.data
})
)
.catch(err =>
dispatch(returnErrors(err.response.data, err.response.status))
);
};

export const deleteItem = id => (dispatch, getState) => {
axios .delete(/api/items/id, tokenConfig(getState))
.then(res =>
dispatch({
type: DELETE_ITEM,
payload: id
})
)
.catch(err =>
dispatch(returnErrors(err.response.data, err.response.status))
);
};

export const setItemsLoading = () => {
return {
type: ITEMS_LOADING
};
};
```

types.js

```
export const GET_ITEMS = 'GET_ITEMS';
export const ADD_ITEM = 'ADD_ITEM';
export const DELETE_ITEM = 'DELETE_ITEM';
export const ITEMS_LOADING = 'ITEMS_LOADING';
export const USER_LOADING = "USER_LOADING";
export const USER_LOADED = "USER_LOADED";
export const AUTH_ERROR = "AUTH_ERROR";
export const LOGIN_SUCCESS = "LOGIN_SUCCESS";
export const LOGIN_FAIL = "LOGIN_FAIL";
export const LOGOUT_SUCCESS = "LOGOUT_SUCCESS";
export const REGISTER_SUCCESS = "REGISTER_SUCCESS";
export const REGISTER_FAIL = "REGISTER_FAIL";
export const GET_ERRORS = 'GET_ERRORS';
export const CLEAR_ERRORS = 'CLEAR_ERRORS';
```

authReducers.js

```
import { USER_LOADED, USER_LOADING, AUTH_ERROR, LOGIN_SUCCESS, LOGIN_FAIL, LOGOUT_SUCCESS,
REGISTER_SUCCESS, REGISTER_FAIL } from '../actions/types';

const initialState = {
token: localStorage.getItem('token'),
isAuthenticated: null,
isLoading: false,
user: null
};

export default function(state = initialState, action) {
switch (action.type) {
case USER_LOADING:
return {
...state,
isLoading: true
};
case USER_LOADED:
return {
...state,
isAuthenticated: true,
isLoading: false,
user: action.payload
};
case LOGIN_SUCCESS:
case REGISTER_SUCCESS:
localStorage.setItem('token', action.payload.token);
return {
...state,
...action.payload,
isAuthenticated: true,
isLoading: false
};
case AUTH_ERROR:
case LOGIN_FAIL:
case LOGOUT_SUCCESS:
case REGISTER_FAIL:
localStorage.removeItem('token');
return {
...state,
token: null,
user: null,
isAuthenticated: false,
isLoading: false
};

default:
return state;
}
}
```

errrorReducers.js

```
import { GET_ERRORS, CLEAR_ERRORS } from '../actions/types';

const initialState = {
msg: {},
status: null,
id: null
}

export default function(state = initialState, action) {
switch(action.type) {
case GET_ERRORS:
return {
msg: action.payload.msg,
status: action.payload.status,
id: action.payload.id
};
case CLEAR_ERRORS:
return {
msg: {},
status: null,
id: null
};
default:
return state;
}
}
```

itemReducers.js

```
import { GET_ITEMS, ADD_ITEM, DELETE_ITEM, ITEMS_LOADING } from '../actions/types';

const initialState = {
items: [],
loading: false
};

export default function(state = initialState, action) {
switch (action.type) {
case GET_ITEMS:
return {
...state,
items: action.payload,
loading: false
};
case DELETE_ITEM:
return {
...state,
items: state.items.filter(item => item._id !== action.payload)
};
case ADD_ITEM:
return {
...state,
items: [action.payload, ...state.items]
};
case ITEMS_LOADING:
return {
...state,
loading: true
};
default:
return state;
}
}
```

indexReducers.js

```
import { combineReducers } from 'redux';
import itemReducer from './itemReducer';
import errorReducer from './errorReducer';
import authReducer from './authReducer';

export default combineReducers({
item: itemReducer,
error: errorReducer,
auth: authReducer
});
```

store.js

```
import { createStore, applyMiddleware, compose } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const initialState = {};
const middleWare = [thunk];
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
const store = createStore(
rootReducer,
initialState,
composeEnhancers(applyMiddleware(...middleWare))
);
export default store;
```

authLoginModal.jsx

```
import React, { Component } from 'react';
import { Button, Modal, ModalHeader, ModalBody, Form, FormGroup, Label, Input, NavLink, Alert } from 'reactstrap';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';
import { login } from '../../actions/authActions';
import { clearErrors } from '../../actions/errorActions';

class LoginModal extends Component {
state = {
modal: false,
email: '',
password: '',
msg: null
};
static propTypes = {
isAuthenticated: PropTypes.bool,
error: PropTypes.object.isRequired,
login: PropTypes.func.isRequired,
clearErrors: PropTypes.func.isRequired
};

componentDidUpdate(prevProps) {
const { error, isAuthenticated } = this.props;
if (error !== prevProps.error) {
// Check for register error
if (error.id === 'LOGIN_FAIL') {
this.setState({ msg: error.msg.msg });
}
else {
this.setState({ msg: null });
}
}

// If authenticated, close modal
if (this.state.modal) {
if (isAuthenticated) {
this.toggle();
}
}
}

toggle = () => {
// Clear errors
this.props.clearErrors();
this.setState({
modal: !this.state.modal
});
};
onChange = e => {
this.setState({ [e.target.name]: e.target.value });
};

onSubmit = e => {
e.preventDefault();
const { email, password } = this.state;

const user = {
email,
password
};
// Attempt to login
this.props.login(user);
};
render() {
return (
'<'div>
'<'NavLink onClick={this.toggle} href='#''><'br/> Login '<'/NavLink>
'<'Modal isOpen={this.state.modal}
toggle={this.toggle}'>'
'<'ModalHeader toggle={this.toggle}'>'Login'<'/ModalHeader>
'<'ModalBody>
{this.state.msg ? (
'<'Alert color='danger'>{this.state.msg}'<'/Alert>
) : null}
'<'Form onSubmit={this.onSubmit}>
'<'FormGroup>
'<'Label for='email'>Email'<'/Label>
'<'Input
type='email'
name='email'
id='email'
placeholder='Email'
className='mb-3'
onChange={this.onChange}
/'>'
'<'Label for='password'>Password'<'/Label>
'<'Input
type='password'
name='password'
id='password'
placeholder='Password'
className='mb-3'
onChange={this.onChange}
/> '<'Button color='dark' style={{ marginTop: '2rem' }} block> Login '<'/Button>
'<'/FormGroup>
'<'/Form>
'<'/ModalBody>
'<'/Modal>
'<'/div>
);
}
}

const mapStateToProps = state => ({
isAuthenticated: state.auth.isAuthenticated,
error: state.error
});

export default connect(
mapStateToProps,
{ login, clearErrors }
)(LoginModal);
```

authLogout.jsx

```
import React, { Component, Fragment } from 'react';
import { NavLink } from 'reactstrap';
import { connect } from 'react-redux';
import { logout } from '../../actions/authActions';
import PropTypes from 'prop-types';

export class Logout extends Component {
static propTypes = {
logout: PropTypes.func.isRequired
};
render() {
return (
'<'Fragment>
'<'NavLink onClick={this.props.logout} href='#'> Logout '<'/NavLink>
'<'/Fragment>
);
}
}

export default connect(
null,
{ logout }
)(Logout);
```

authRegister.jsx

```
import React, { Component } from 'react';
import { Button, Modal, ModalHeader, ModalBody, Form, FormGroup, Label, Input, NavLink, Alert } from 'reactstrap';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';
import { register } from '../../actions/authActions';
import { clearErrors } from '../../actions/errorActions';

class RegisterModal extends Component {
state = {
modal: false,
name: '',
email: '',
password: '',
msg: null
};

static propTypes = {
isAuthenticated: PropTypes.bool,
error: PropTypes.object.isRequired,
register: PropTypes.func.isRequired,
clearErrors: PropTypes.func.isRequired
};

componentDidUpdate(prevProps) {
const { error, isAuthenticated } = this.props;
if (error !== prevProps.error) {
// Check for register error
if (error.id === 'REGISTER_FAIL') {
this.setState({ msg: error.msg.msg });
}
else {
this.setState({ msg: null });
}
}

// If authenticated, close modal
if (this.state.modal) {
if (isAuthenticated) {
this.toggle();
}
}
}

toggle = () => {
// Clear errors
this.props.clearErrors();
this.setState({
```

```
modal: !this.state.modal
});
};

onChange = e => {
this.setState({ [e.target.name]: e.target.value });
};

onSubmit = e => {
e.preventDefault();
const { name, email, password } = this.state;
// Create user object
const newUser = {
name,
email,
password
};
// Attempt to register
this.props.register(newUser);
};
render() {
return (
'<'div>
'<'NavLink onClick={this.toggle} href='#'>
Register '<'/NavLink>
'<'Modal isOpen={this.state.modal} toggle={this.toggle}>
'<'ModalHeader toggle={this.toggle}>Register'<'/ModalHeader>
'<'ModalBody>
{this.state.msg ? (
'<'Alert color='danger'>{this.state.msg}'<'/Alert>
) : null}
'<'Form onSubmit={this.onSubmit}>
'<'FormGroup>
'<'Label for='name'>Name'<'/Label>
'<'Input
type='text'
name='name'
id='name'
placeholder='Name'
className='mb-3'
onChange={this.onChange}
/>
'<'Label for='email'>Email'<'/Label>
'<'Input
type='email'
name='email'
id='email'
placeholder='Email'
className='mb-3'
onChange={this.onChange}
/>
'<'Label for='password'>Password'<'/Label>
'<'Input
type='password'
name='password'
id='password'
placeholder='Password'
className='mb-3'
onChange={this.onChange}
/>
'<'Button color='dark' style={{ marginTop: '2rem' }} block>
Register '<'/Button>
'<'/FormGroup>
'<'/Form>
'<'/ModalBody>
'<'/Modal>
'<'/div>
);
}
}
```

```
const mapStateToProps = state => ({
isAuthenticated: state.auth.isAuthenticated,
error: state.error
});

export default connect(
mapStateToProps,
{ register, clearErrors }
)(RegisterModal);
```

appNavBar.jsx

```
import React, { Component, Fragment } from 'react';
import { Collapse, Navbar, NavbarToggler, NavbarBrand, Nav, NavItem, Container } from 'reactstrap';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';
import RegisterModal from './auth/RegisterModal';
import LoginModal from './auth/LoginModal';
import Logout from './auth/Logout';

class AppNavbar extends Component {
state = {
isOpen: false
};

static propTypes = {
auth: PropTypes.object.isRequired
};

toggle = () => {
this.setState({
isOpen: !this.state.isOpen
});
};

render() {
const { isAuthenticated, user } = this.props.auth;
const authLinks = (
'<'Fragment>
'<'NavItem>
'<'span className='navbar-text mr-3'>
'<'strong>{user ? Welcome user.name : ''}'<'/strong>
'<'/span>
'<'/NavItem>
'<'NavItem>
'<'Logout />
'<'/NavItem>
'<'/Fragment>
);

const guestLinks = (
'<'Fragment>
'<'NavItem>
'<'RegisterModal />
'<'/NavItem>
'<'NavItem>
'<'LoginModal />
'<'/NavItem>
'<'/Fragment>
);

return (
'<'div>
'<'Navbar color='dark' dark expand='sm' className='mb-5'>
'<'Container>
'<'NavbarBrand href='/'>ShoppingList'<'/NavbarBrand>
'<'NavbarToggler onClick={this.toggle} />
'<'Collapse isOpen={this.state.isOpen} navbar>
'<'Nav className='ml-auto' navbar>
{isAuthenticated ? authLinks : guestLinks}
'<'/Nav>
'<'/Collapse>
'<'/Container>
'<'/Navbar>
'<'/div>
);
}
}

const mapStateToProps = state => ({
auth: state.auth
});

export default connect(
mapStateToProps,
null
)(AppNavbar);
```

itemModal.jsx

```
import React, { Component } from 'react';
import { Button, Modal, ModalHeader, ModalBody, Form, FormGroup, Label, Input } from 'reactstrap';
import { connect } from 'react-redux';
import { addItem } from '../actions/itemActions';
import PropTypes from 'prop-types';

class ItemModal extends Component {
state = {
modal: false,
name: ''
};

static propTypes = {
isAuthenticated: PropTypes.bool
};

toggle = () => {
this.setState({
modal: !this.state.modal
});
};

onChange = e => {
this.setState({ [e.target.name]: e.target.value });
};

onSubmit = e => {
e.preventDefault();

const newItem = {
name: this.state.name
};

// Add item via addItem action
this.props.addItem(newItem);


// Close modal
this.toggle();
};
render() {
return (
'<'div>
{this.props.isAuthenticated ? (
'<'Button
color='dark'
style={{ marginBottom: '2rem' }}
onClick={this.toggle}
>
Add Item '<'/Button>
) : (
'<'h4 className='mb-3 ml-4'>Please log in to manage items'<'/h4>
)}
'<'Modal isOpen={this.state.modal} toggle={this.toggle}>
'<'ModalHeader toggle={this.toggle}>Add To Shopping List'<'/ModalHeader>
'<'ModalBody>
'<'Form onSubmit={this.onSubmit}>
'<'FormGroup>
'<'Label for='item'>Item'<'/Label>
'<'Input
type='text' name='name' id='item' placeholder='Add shopping item'
onChange={this.onChange} />
'<'Button color='dark' style={{ marginTop: '2rem' }} block>
Add Item '<'/Button>
'<'/FormGroup>
'<'/Form>
'<'/ModalBody>
'<'/Modal>
'<'/div>
);
}
}

const mapStateToProps = state => ({
item: state.item,
isAuthenticated: state.auth.isAuthenticated
});

export default connect(
mapStateToProps,
{ addItem }
)(ItemModal);
```

shoppingList.jsx

```
import React, { Component } from 'react';
import { Container, ListGroup, ListGroupItem, Button } from 'reactstrap';
import { CSSTransition, TransitionGroup } from 'react-transition-group';
import { connect } from 'react-redux';
import { getItems, deleteItem } from '../actions/itemActions';
import PropTypes from 'prop-types';

class ShoppingList extends Component {
static propTypes = {
getItems: PropTypes.func.isRequired,
item: PropTypes.object.isRequired,
isAuthenticated: PropTypes.bool
};

componentDidMount() {
this.props.getItems();
}

onDeleteClick = id => {
this.props.deleteItem(id);
};

render() {
const { items } = this.props.item;
return (
'<'Container>
'<'ListGroup>
'<'TransitionGroup className='shopping-list'>
{items.map(({ _id, name }) => (
'<'CSSTransition key={_id} timeout={500} classNames='fade'>
'<'ListGroupItem>
{this.props.isAuthenticated ? (
'<'Button
className='remove-btn' color='danger' size='sm'
onClick={this.onDeleteClick.bind(this, _id)} >
× '<'/Button>
) : null}
{name}
'<'/ListGroupItem>
'<'/CSSTransition>
))}
'<'/TransitionGroup>
'<'/ListGroup>
'<'/Container>
);
}
}

const mapStateToProps = state => ({
item: state.item,
isAuthenticated: state.auth.isAuthenticated
});

export default connect(
mapStateToProps,
{ getItems, deleteItem }
)(ShoppingList);
```

merndoc
ShoppingList
teamMaster

"start": "node server.js",**server.js**

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
var mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1/image-store');

var conn = mongoose.connection;
var multer = require('multer');
var GridFsStorage = require('multer-gridfs-storage');
var Grid = require('gridfs-stream');
Grid.mongo = mongoose.mongo;
var gfs = Grid(conn.db);

/** Seting up server to accept cross-origin browser requests */
app.use(function(req, res, next) { //allow cross origin requests
res.setHeader("Access-Control-Allow-Methods", "POST, PUT, OPTIONS, DELETE, GET");
res.header("Access-Control-Allow-Origin", "http://localhost:3000");
res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
res.header("Access-Control-Allow-Credentials", true);
next();
});

app.use(bodyParser.json());

/** Setting up storage using multer-gridfs-storage */
var storage = GridFsStorage({
gfs : gfs,
filename: function (req, file, cb) {
var datetimestamp = Date.now();
cb(null, file.fieldname + '-' + datetimestamp + '.' + file.originalname.split('.')[file.originalname.split('.').length -1]);
},
/** With gridfs we can store aditional meta-data along with the file */

metadata: function(req, file, cb) {
cb(null, { originalname: file.originalname });
},
root: 'ctFiles' //root name for collection to store files into
});

var upload = multer({ //multer settings for single upload
storage: storage
}).single('file');
```

```
/** API path that will upload the files */
app.post('/upload', function(req, res) {
upload(req,res,function(err){
if(err){
res.json({error_code:1,err_desc:err});
return;
}
res.json({error_code:0,err_desc:null});
});
});

app.get('/file/:filename', function(req, res){
gfs.collection('ctFiles'); //set collection name to lookup into
/** First check if file exists */
gfs.files.find({filename: req.params.filename}).toArray(function(err, files){
if(!files || files.length === 0){
return res.status(404).json({
responseCode: 1,
responseMessage: "error"
});
}

/** create read stream */
var readstream = gfs.createReadStream({
filename: files[0].filename,
root: "ctFiles"
});

/** set the proper content type */
res.set('Content-Type', files[0].contentType)
/** return response */
return readstream.pipe(res);
});
});

app.get('/file', function(req, res){
gfs.collection('ctFiles'); //set collection name to lookup into
gfs.files.find().toArray(function(err, files){
if(!files || files.length === 0){
return res.status(404).json({
responseCode: 1,
responseMessage: "error"
});
}
res.send(JSON.stringify(files));
});
});

app.listen('3002', function(){
console.log('Running on 3002...');
});
```

imageCard.jsx

```
import React from 'react';

class ImageCard extends React.Component {
render() {
return (
'<'div className="column is-3">
'<'div className="card" >
'<'div className="card-image">
'<'figure className="image is-4by3">
'<'img alt={this.props.alt} src={this.props.src} />
'<'/figure>
'<'/div>
'<'div className="card-content">
'<'div className="media">
'<'div className="media-content">
'<'p className="title is-4">{new Date(this.props.date).toLocaleDateString("en-GB", { weekday: 'long', year: 'numeric',
month: 'long', day: 'numeric' })}'<'/p>
'<'/div>
'<'/div>
'<'/div>
'<'/div>
'<'/div>
);
} }

export default ImageCard;
```

fileUpload.jsx

```
import React from 'react';
import ImageCard from './ImageCard';

class Uploader extends React.Component {
constructor() {
super();
this.state ={
file: {name: null},
images: []
};
this.onFormSubmit = this.onFormSubmit.bind(this);
this.onChange = this.onChange.bind(this);
}

componentWillMount() {
this.getImageData();
}

onFormSubmit(e) {
e.preventDefault();
this.fileUpload(this.state.file);
}

onChange(e) {
this.setState({file:e.target.files[0]})
}

getImageData() {
const url = 'http://localhost:3002/file';
fetch(url)
.then(response => {
if(response.ok) return response.json();
// throw new Error('Request failed.');
return [];
})
.then(data => { this.setState({images: data}); });
}

fileUpload(file){
const url = 'http://localhost:3002/upload';
const formData = new FormData();
formData.append('file',file)
fetch(url, {
mode: 'no-cors',
method: "POST",
body: formData
})
.then(res => { this.getImageData(); });
}
render() {
let images;
if(this.state.images.length > 0) {
images = this.state.images.map( i => {
return (
'<'ImageCard key={i._id} alt={i.metadata.originalname}
src={'http://localhost:3002/file/'+i.filename} date={i.uploadDate} />
);
});
}
else {
images = '<'h2 className="subtitle">No images :('<'/h2>;
}

return (
'<'section className="section">
'<'div className="container is-fluid">
'<'h1 className="title">Photo Gallery'<'/h1>
'<'div className="file is-info has-name is-fullwidth">
'<'label className="file-label">
'<'input className="file-input" type="file" name="resume" onChange={this.onChange} />
'<'span className="file-cta">
'<'span className="file-icon">
'<'i className="fas fa-upload">'<'/i>
'<'/span>
'<'span className="file-label">
Choose a file… '<'/span>
'<'/span>
'<'span className="file-name">
{this.state.file.name}
'<'/span>
'<'/label>
'<'/div>
'<'button className="button is-primary" onClick={this.onFormSubmit} type="submit">Upload'<'/button>
'<'/div>
'<'hr/>
'<'div className="container is-fluid">
'<'div className="columns is-multiline">
{images}
'<'/div>
'<'/div>
'<'/section>
);
} }
export default Uploader;
```

merndoc
ShoppingList
fileUpload

server.js

const express = require("express"); const mongoose = require("mongoose"); const bodyParser = require("body-parser"); const passport = require("passport"); const users = require("./routes/api/users"); const projects = require("./routes/api/projects"); const tasks = require("./routes/api/tasks"); const app = express(); // Bodyparser middleware app.use(bodyParser.urlencoded({ extended: false })); app.use(bodyParser.json()); // DB Config //const db = require("./config/keys"); const config = require('./config/DB.js'); // Connect to MongoDB // mongoose // .connect(// db, // { useNewUrlParser: true } //) // .then(() => console.log("MongoDB successfully connected")) // .catch(err => console.log(err)); mongoose.Promise = global.Promise; mongoose.connect(config.DB, { useNewUrlParser: true }).then(() => {console.log('Database is connected') }, err => { console.log('Can not connect to the database'+ err)}); // Passport middleware app.use(passport.initialize()); // Passport config require("./config/passport")(passport); // Routes app.use("/api/users", users); app.use("/api/projects", projects); app.use("/api/tasks", tasks); const port = process.env.PORT || 5000; app.listen(port, () => console.log(Server up and running on port port !));

modalsProject.js

const mongoose = require("mongoose"); const Schema = mongoose.Schema; // Create Schema const ProjectSchema = new Schema({ name: { type: String, required: true }, owner: { type: Object, required: true }, teamMembers: [{ email: { type: String }, name: { type: String } }], date: { type: Date, default: Date.now } }); module.exports = Project = mongoose.model("projects", ProjectSchema);

modalsTask.js

const mongoose = require("mongoose"); const Schema = mongoose.Schema; // Create Schema const TaskSchema = new Schema({ project: { type: Schema.Types.ObjectId, ref: "projects", required: true }, taskName: { type: String, required: true }, dateDue: { type: String }, assignee: { type: String }, dateCreated: { type: Date, default: Date.now } }); module.exports = Task = mongoose.model("tasks", TaskSchema);

modelsUser.js

```
const mongoose = require("mongoose"); const Schema = mongoose.Schema; // Create Schema const UserSchema = new
Schema({ name: { type: String, required: true }, email: { type: String, required: true }, password: { type: String, required: true
}, date: { type: Date, default: Date.now } }); module.exports = User = mongoose.model("users", UserSchema);
```

routesApiProjects.js

```
const express = require("express"); const router = express.Router(); const passport = require("passport"); const Project =
require("../../models/Project"); // @route GET api/projects // @desc Get all projects for a specific user // @access Private
router.get( "/", passport.authenticate("jwt", { session: false }), async (req, res) => { let projectsArr = []; // Member projects
await Project.find({}) .then(projects => { projects.map(project => { project.teamMembers.map(member => { if
(member.email == req.user.email) { projectsArr.push(project); } }); }); }) .catch(err => console.log(err)); const OWNER = {
id: req.user.id, name: req.user.name, email: req.user.email }; // Combine with owner projects await Project.find({ owner:
OWNER }) .then(projects => { let finalArr = [...projects, ...projectsArr]; res.json(finalArr); }) .catch(err => console.log(err));
} ); // @route GET api/projects/:id // @desc Get specific project by id // @access Private router.get( "/:id",
passport.authenticate("jwt", { session: false }), (req, res) => { let id = req.params.id; Project.findById(id).then(project =>
res.json(project)); } ); // @route POST api/projects/create // @desc Create a new project // @access Private router.post(
"/create", passport.authenticate("jwt", { session: false }), async (req, res) => { const OWNER = { id: req.user.id, name:
req.user.name, email: req.user.email }; const NEW_PROJECT = await new Project({ owner: OWNER, name:
req.body.projectName, teamMembers: req.body.members }); NEW_PROJECT.save().then(project => res.json(project)); } ); //
@route PATCH api/projects/update // @desc Update an existing project // @access Private router.patch( "/update",
passport.authenticate("jwt", { session: false }), (req, res) => { let projectFields = {}; projectFields.name =
req.body.projectName; projectFields.teamMembers = req.body.members; Project.findOneAndUpdate( { _id: req.body.id }, {
$set: projectFields }, { new: true } ) .then(project => { res.json(project); }) .catch(err => console.log(err)); } ); // @route
DELETE api/projects/delete/:id // @desc Delete an existing project // @access Private router.delete( "/delete/:id",
passport.authenticate("jwt", { session: false }), (req, res) => { Project.findById(req.params.id).then(project => {
project.remove().then(() => res.json({ success: true })); }); } ); module.exports = router;
```

routesApiTasks.js

```
const express = require("express"); const router = express.Router(); const passport = require("passport"); const Task =
require("../../models/Task"); // @route GET api/tasks/:id // @desc Get tasks for specific project // @access Private router.get(
"/:id", passport.authenticate("jwt", { session: false }), (req, res) => { let id = req.params.id; Task.find({ project: id
}).then(tasks => res.json(tasks)); } ); // @route POST api/tasks/create // @desc Create a new task // @access Private
router.post( "/create", passport.authenticate("jwt", { session: false }), (req, res) => { const NEW_TASK = new Task({ project:
req.body.project, taskName: req.body.taskName, dateDue: req.body.dateDue, assignee: req.body.assignee });
NEW_TASK.save() .then(task => res.json(task)) .catch(err => console.log(err)); } ); module.exports = router;
```

routesApiUsers.js

```
const express = require("express"); const router = express.Router(); const bcrypt = require("bcryptjs"); const jwt =
require("jsonwebtoken"); const keys = require("../../config/DB"); const passport = require("passport"); // Load input validation
const validateRegisterInput = require("../../validation/register"); const validateLoginInput = require("../../validation/login"); //
Load User model const User = require("../../models/User"); // @route POST api/users/register // @desc Register user //
@access Public router.post("/register", (req, res) => { // Form validation const { errors, isValid } =
validateRegisterInput(req.body); // Check validation if (!isValid) { return res.status(400).json(errors); } User.findOne({ email:
req.body.email }).then(user => { if (user) { return res.status(400).json({ email: "Email already exists" }); } else { const
newUser = new User({ name: req.body.name, email: req.body.email, password: req.body.password }); // Hash password
before saving in database 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 POST api/users/login // @desc Login user and return JWT token // @access Public router.post("/login", (req, res) =>
{ // Form validation const { errors, isValid } = validateLoginInput(req.body); // Check validation if (!isValid) { return
res.status(400).json(errors); } const email = req.body.email; const password = req.body.password; // Find user by email
User.findOne({ email }).then(user => { // Check if user exists if (!user) { return res.status(404).json({ emailnotfound: "Email
not found" }); } // Check password bcrypt.compare(password, user.password).then(isMatch => { if (isMatch) { // User
matched // Create JWT Payload const payload = { id: user.id, name: user.name, email: user.email }; // Sign token jwt.sign(
payload, keys.secretOrKey, { expiresIn: 31556926 // 1 year in seconds }, (err, token) => { res.json({ success: true, token:
"Bearer " + token }); } ); } else { return res .status(400) .json({ passwordincorrect: "Password incorrect" }); } }); }); });
module.exports = router;
```

validationRegister.js

```
const Validator = require("validator"); const isEmpty = require("is-empty"); module.exports = function
validateRegisterInput(data) { let errors = {}; // Convert empty fields to an empty string so we can use validator functions
data.name = !isEmpty(data.name) ? data.name : ""; data.email = !isEmpty(data.email) ? data.email : ""; data.password =
!isEmpty(data.password) ? data.password : ""; // Name checks if (Validator.isEmpty(data.name)) { errors.name = "Name field
is required"; } // Email checks if (Validator.isEmpty(data.email)) { errors.email = "Email field is required"; } else if
(!Validator.isEmail(data.email)) { errors.email = "Email is invalid"; } // Password checks if
(Validator.isEmpty(data.password)) { errors.password = "Password field is required"; } if (!Validator.isLength(data.password,
{ min: 6, max: 30 })) { errors.password = "Password must be at least 6 characters"; } return { errors, isValid: isEmpty(errors)
}; };
```

validationLogin.js

```
const Validator = require("validator"); const isEmpty = require("is-empty"); module.exports = function validateRegisterInput(data) { let errors = {}; // Convert empty fields to an empty string so we can use validator functions data.email = !isEmpty(data.email) ? data.email : ""; data.password = !isEmpty(data.password) ? data.password : ""; // Email checks if (Validator.isEmpty(data.email)) { errors.email = "Email field is required"; } else if (!Validator.isEmail(data.email)) { errors.email = "Email is invalid"; } // Password checks if (Validator.isEmpty(data.password)) { errors.password = "Password field is required"; } return { errors, isValid: isEmpty(errors) }; };
```

authActions.js

```
import axios from "axios"; import setAuthToken from "../utils/setAuthToken"; import jwt_decode from "jwt-decode"; import { GET_ERRORS, SET_CURRENT_USER, USER_LOADING } from "./types"; // Register User export const registerUser = (userData, history) => dispatch => { axios .post("/api/users/register", userData) .then(res => history.push("/")) .catch(err => dispatch({ type: GET_ERRORS, payload: err.response.data }) ); }; // Login - get user token export const loginUser = userData => dispatch => { axios .post("/api/users/login", userData) .then(res => { // Save to localStorage // Set token to localStorage const { token } = res.data; localStorage.setItem("jwtTokenTeams", token); // Set token to Auth header setAuthToken(token); // Decode token to get user data const decoded = jwt_decode(token); // Set current user dispatch(setCurrentUser(decoded)); }) .catch(err => dispatch({ type: GET_ERRORS, payload: err.response.data }) ); }; // Set logged in user export const setCurrentUser = decoded => { return { type: SET_CURRENT_USER, payload: decoded }; }; // User loading export const setUserLoading = () => { return { type: USER_LOADING }; }; // Log user out export const logoutUser = () => dispatch => { // Remove token from local storage localStorage.removeItem("jwtTokenTeams"); // Remove auth header for future requests setAuthToken(false); // Set current user to empty object {} which will set isAuthenticated to false dispatch(setCurrentUser({})); };
```

projectsActions.js

```
import axios from "axios"; import { CREATE_PROJECT, UPDATE_PROJECT, DELETE_PROJECT, GET_PROJECT, PROJECT_LOADING, GET_PROJECTS, PROJECTS_LOADING } from "./types"; // Create Project export const createProject = projectData => dispatch => { axios .post("/api/projects/create", projectData) .then(res => dispatch({ type: CREATE_PROJECT, payload: res.data }) ) .catch(err => console.log(err)); }; // Update Project export const updateProject = projectData => dispatch => { axios .patch("/api/projects/update", projectData) .then(res => dispatch({ type: UPDATE_PROJECT, payload: res.data }) ) .catch(err => console.log(err)); }; // Delete Project export const deleteProject = id => dispatch => { axios .delete(/api/projects/delete/id) .then(res => dispatch({ type: DELETE_PROJECT, payload: id }) ) .catch(err => console.log(err)); }; // Get specific project by id export const getProject = id => dispatch => { dispatch(setProjectLoading()); axios .get(/api/projects/id) .then(res => dispatch({ type: GET_PROJECT, payload: res.data }) ) .catch(err => dispatch({ type: GET_PROJECT, payload: null }) ); }; // Get all projects for specific user export const getProjects = () => dispatch => { dispatch(setProjectsLoading()); axios .get("/api/projects") .then(res => dispatch({ type: GET_PROJECTS, payload: res.data }) ) .catch(err => dispatch({ type: GET_PROJECTS, payload: null }) ); }; // Project loading export const setProjectLoading = () => { return { type: PROJECT_LOADING }; }; // Projects loading export const setProjectsLoading = () => { return { type: PROJECTS_LOADING }; };
```

taskActions.js

```
import axios from "axios"; import { CREATE_TASK, // UPDATE_TASK, // DELETE_TASK, GET_TASKS, TASKS_LOADING } from "./types"; // Create Task export const createTask = taskData => dispatch => { axios .post("/api/tasks/create", taskData) .then(res => dispatch({ type: CREATE_TASK, payload: res.data }) ) .catch(err => console.log(err)); }; // Get tasks by project id export const getTasks = id => dispatch => { dispatch(setTasksLoading()); axios .get(/api/tasks/id) .then(res => dispatch({ type: GET_TASKS, payload: res.data }) ) .catch(err => dispatch({ type: GET_TASKS, payload: null }) ); }; // Tasks loading export const setTasksLoading = () => { return { type: TASKS_LOADING }; };
```

types.js

```
// Authentication export const GET_ERRORS = "GET_ERRORS"; export const USER_LOADING = "USER_LOADING"; export const SET_CURRENT_USER = "SET_CURRENT_USER"; // Projects export const CREATE_PROJECT = "CREATE_PROJECT"; export const UPDATE_PROJECT = "UPDATE_PROJECT"; export const DELETE_PROJECT = "DELETE_PROJECT"; export const GET_PROJECT = "GET_PROJECT"; export const PROJECT_LOADING = "PROJECT_LOADING"; export const GET_PROJECTS = "GET_PROJECTS"; export const PROJECTS_LOADING = "PROJECTS_LOADING"; // Tasks export const CREATE_TASK = "CREATE_TASK"; export const UPDATE_TASK = "UPDATE_TASK"; export const DELETE_TASK = "DELETE_TASK"; export const GET_TASKS = "GET_TASKS"; export const TASKS_LOADING = "TASKS_LOADING";
```

authReducer.js

```
import { SET_CURRENT_USER, USER_LOADING } from "../actions/types"; const isEmpty = require("is-empty"); const initialState = { isAuthenticated: false, user: {}, loading: false }; export default function(state = initialState, action) { switch (action.type) { case SET_CURRENT_USER: return { ...state, isAuthenticated: !isEmpty(action.payload), user: action.payload }; case USER_LOADING: return { ...state, loading: true }; default: return state; } }
```

errorReducer.js

```
import { GET_ERRORS } from "../actions/types"; const initialState = {}; export default function(state = initialState, action) { switch (action.type) { case GET_ERRORS: return action.payload; default: return state; } }
```

projectsReducer.js

import { CREATE_PROJECT, UPDATE_PROJECT, DELETE_PROJECT, GET_PROJECT, PROJECT_LOADING, GET_PROJECTS, PROJECTS_LOADING } from "../actions/types"; const initialState = { projects: [], project: [], projectLoading: false, projectsLoading: false }; export default function(state = initialState, action) { switch (action.type) { case CREATE_PROJECT: return { ...state, projects: [action.payload, ...state.projects] }; case UPDATE_PROJECT: let index = state.projects.findIndex(project => project._id === action.payload._id); state.projects.splice(index, 1); return { ...state, projects: [action.payload, ...state.projects] }; case DELETE_PROJECT: return { ...state, projects: state.projects.filter(project => project._id !== action.payload) }; case GET_PROJECT: return { ...state, project: action.payload, projectLoading: false }; case GET_PROJECTS: return { ...state, projects: action.payload, projectsLoading: false }; case PROJECT_LOADING: return { ...state, projectLoading: true }; case PROJECTS_LOADING: return { ...state, projectsLoading: true }; default: return state; } }

tasksReducer.js

import { CREATE_TASK, // UPDATE_TASK, // DELETE_TASK, GET_TASKS, TASKS_LOADING } from "../actions/types"; const initialState = { tasks: [], tasksLoading: false }; export default function(state = initialState, action) { switch (action.type) { case CREATE_TASK: return { ...state, tasks: [action.payload, ...state.tasks] }; case GET_TASKS: return { ...state, tasks: action.payload, tasksLoading: false }; case TASKS_LOADING: return { ...state, tasksLoading: true }; default: return state; } }

indexReducers.js

import { combineReducers } from "redux"; import authReducer from "./authReducer"; import errorReducer from "./errorReducer"; import projectsReducer from "./projectsReducer"; import tasksReducer from "./tasksReducer"; export default combineReducers({ auth: authReducer, errors: errorReducer, projects: projectsReducer, tasks: tasksReducer });

setAuthToken.js

import axios from "axios"; const setAuthToken = token => { if (token) { // Apply authorization token to every request if logged in axios.defaults.headers.common["Authorization"] = token; } else { // Delete auth header delete axios.defaults.headers.common["Authorization"]; } }; export default setAuthToken;

store.js

import { createStore, applyMiddleware, compose } from "redux"; import thunk from "redux-thunk"; import rootReducer from "./reducers"; const initialState = {}; const middleware = [thunk]; const store = createStore(rootReducer, initialState, compose(applyMiddleware(...middleware), (window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()) || compose)); export default store;

404.js

import React from "react"; import "./404.scss"; const NotFound = props => { return (
404

The requested page was not found on our server.

Either you the url you typed in is incorrect, you do not have access privileges to the page, or the page you are looking for has been removed.

); }; export default NotFound;

Register.js

import React, { Component } from "react"; import { Link, withRouter } from "react-router-dom"; import PropTypes from "prop-types"; import { connect } from "react-redux"; import { registerUser } from "../../actions/authActions"; import "./Auth.scss"; class Register extends Component { constructor() { super(); this.state = { name: "", email: "", password: "", errors: {} }; } componentDidMount() { // If logged in and user navigates to Register page, should redirect them to dashboard if (this.props.auth.isAuthenticated) { this.props.history.push("/dashboard"); } } componentWillReceiveProps(nextProps) { if (nextProps.errors) { this.setState({ errors: nextProps.errors }); } } onChange = e => { this.setState({ [e.target.id]: e.target.value }); }; onSubmit = e => { e.preventDefault(); const newUser = { name: this.state.name, email: this.state.email, password: this.state.password }; this.props.registerUser(newUser, this.props.history); }; render() { const { errors } = this.state; return (
Register below
Name
{this.state.name}
{errors.name}
Email address
{this.state.email}
{errors.email}
Password
•••••••••••••••••••
{errors.password}

Sign up

Sign in

); } } Register.propTypes = { registerUser: PropTypes.func.isRequired, auth: PropTypes.object.isRequired, errors: PropTypes.object.isRequired }; const mapStateToProps = state => ({ auth: state.auth, errors: state.errors }); export default connect(mapStateToProps, { registerUser })(withRouter(Register));

Login.js

import React, { Component } from "react"; import { Link } from "react-router-dom"; import PropTypes from "prop-types"; import { connect } from "react-redux"; import { loginUser } from "../../actions/authActions"; import "./Auth.scss"; class Login extends Component { constructor() { super(); this.state = { email: "", password: "", errors: {} }; } componentDidMount() { // If logged in and user navigates to Login page, should redirect them to dashboard if (this.props.auth.isAuthenticated) { this.props.history.push("/dashboard"); } } componentWillReceiveProps(nextProps) { if (nextProps.auth.isAuthenticated) { this.props.history.push("/dashboard"); } if (nextProps.errors) { this.setState({ errors: nextProps.errors }); } } onChange = e => { this.setState({ [e.target.id]: e.target.value }); }; onSubmit = e => { e.preventDefault(); const userData = { email: this.state.email, password: this.state.password }; this.props.loginUser(userData); }; fillDemoEmail = () => { this.setState({ email: "test@test.com" }); }; fillDemoPassword = () => { this.setState({ password: "test123" }); }; render() { const { errors } = this.state; return (
Sign in

Just here to demo? Use
u: test@test.com {" "} and{" "} p: test123

Email address
{this.state.email}
{errors.email} {errors.emailnotfound}
Password
••••••••••••••••••••
{errors.password} {errors.passwordincorrect}
Login
Sign up
); } } Login.propTypes = { loginUser: PropTypes.func.isRequired, auth: PropTypes.object.isRequired, errors: PropTypes.object.isRequired }; const mapStateToProps = state => ({ auth: state.auth, errors: state.errors }); export default connect(mapStateToProps, { loginUser })(Login);

PrivateRoute

import React from "react"; import { Route, Redirect } from "react-router-dom"; import { connect } from "react-redux"; import PropTypes from "prop-types"; const PrivateRoute = ({ component: Component, auth, ...rest }) => (auth.isAuthenticated === true ? ( ) : ( ) } />); PrivateRoute.propTypes = { auth: PropTypes.object.isRequired }; const mapStateToProps = state => ({ auth: state.auth }); export default connect(mapStateToProps)(PrivateRoute);

ModalComponents.js

import React, { Component } from "react"; import { connect } from "react-redux"; import { createProject, updateProject, deleteProject } from "../../../../actions/projectsActions"; import { createTask } from "../../../../actions/taskActions"; import moment from "moment"; import "./Modal.scss"; class Modal extends Component { state = { projectName: "", members: [{ name: "", email: "" }], taskName: "", assignee: "", monthDue: "", dayDue: "" }; componentWillReceiveProps(nextProps) { if (nextProps.edit) { this.setState({ projectName: nextProps.name, members: nextProps.members }); } } onChange = e => { if (["name", "email"].includes(e.target.name)) { let members = [...this.state.members]; members[e.target.dataset.id][e.target.name] = e.target.value; this.setState({ members }); } else { this.setState({ [e.target.id]: e.target.value }); } }; addMember = e => { this.setState(prevState => ({ members: [...prevState.members, { name: "", email: "" }] })); }; deleteMember = index => { let array = [...this.state.members]; array.splice(index, 1); this.setState({ members: array }); }; createProject = () => { let project = { projectName: this.state.projectName, members: this.state.members }; this.props.createProject(project); this.onClose(); }; updateProject = async id => { let project = { id: this.props.id, projectName: this.state.projectName, members: this.state.members }; await this.props.updateProject(project); this.onClose(); }; deleteProject = id => { this.props.deleteProject(id); this.onClose(); }; onClose = e => { this.props.onClose && this.props.onClose(e); this.setState({ projectName: "", taskName: "", assignee: "", monthDue: "", dayDue: "", members: [{ name: "", email: "" }] }); }; onSelectChange = e => { this.setState({ [e.target.id]: e.target.value }); }; createTask = e => { e.preventDefault(); let fullDate = this.state.monthDue + "-" + this.state.dayDue + "-" + Date().split(" ")[3]; let momentDate = moment(fullDate, "MM-DD-YYYY") ._d.toString() .split(" "); let finalDate = momentDate[1] + " " + momentDate[2]; const data = { project: this.props.projects.project._id, taskName: this.state.taskName, assignee: this.state.assignee, dateDue: finalDate }; this.props.createTask(data); this.onClose(); }; render() { if (!this.props.modal) { return null; } document.onkeyup = e => { if (e.keyCode === 27 && this.props.modal) { this.onClose(); } }; let { members } = this.state; // Create task modal if (this.props.task) { const { teamMembers } = this.props.projects.project; const { name, email } = this.props.auth.user; // Assignee dropdown in Modal let membersOptions = teamMembers.map((member, index) => (
{member.name}
)); // Due date dropdown in Modal const MONTHS = new Array(12).fill(1); const DAYS = new Array(31).fill(1); let monthsOptions = MONTHS.map((month, i) => (
{i < 9 && "0"} {i + 1}
)); let daysOptions = DAYS.map((day, i) => (

{i < 9 && "0"} {i + 1}
)); return (
×

Create task

Task Name (required)
{this.state.taskName}
Assignee
{name + " (You)"} ▾
Due Date
▾ ▾
Create Task
); } else if (this.props.editTask) { const { teamMembers } = this.props.projects.project; const { name, email } = this.props.auth.user; // Assignee dropdown in Modal let membersOptions = teamMembers.map((member, index) => (
{member.name}
)); // Due date dropdown in Modal const MONTHS = new Array(12).fill(1); const DAYS = new Array(31).fill(1); let monthsOptions = MONTHS.map((month, i) => (
{i < 9 && "0"} {i + 1}
)); let daysOptions = DAYS.map((day, i) => (
{i < 9 && "0"} {i + 1}
)); return (
×

Edit task

Task Name (required)
{this.state.taskName}
Assignee
{name + " (You)"} ▾
Due Date
▾ ▾
Create Task
); } // Edit project modal else if (this.props.edit) { return (
×

Edit Project Info

Created by {this.props.owner.name} ({this.props.owner.email})

Project Name (required)
{this.state.projectName}
Add team members (optional)
Add another member
{members.map((val, id) => { let memberId = member-id, emailId = email-id; return (
Name (required for teams) {members[id].name} Email (required for teams) {members[id].email} REMOVE
); })}
Update Project {this.props.owner.id === this.props.auth.user.id ? (Delete Project) : null}
); } // Create project modal else return (
×

Create a project

Project Name (required)
{this.state.projectName}
Add team members (optional)
Add another member
{members.map((val, id) => { let memberId = member-id, emailId = email-id; return (
Name (required for teams) {members[id].name} Email (required for teams) {members[id].email} REMOVE
); })}
Create Project
); } } const mapStateToProps = state => ({ auth: state.auth, projects: state.projects, tasks: state.tasks }); export default connect(mapStateToProps, { createProject, updateProject, deleteProject, createTask })(Modal);

ProjectComponents.js

import React, { Component } from "react"; import { connect } from "react-redux"; import { getProject } from "../../../../actions/projectsActions"; import { getTasks } from "../../../../actions/taskActions"; import Spinner from "../../../common/Spinner"; import Modal from "../Modal/Modal"; import "../MainContent.scss"; import "./Project.scss"; class Project extends Component { state = { modal: false, edit: false, editTask: false, task: false, name: "", members: [], id: "", owner: {}, tasks: [], date: "" }; toggleModal = e => { this.setState({ modal: !this.state.modal, edit: false, task: false }); }; toggleEditModal = (name, members, id, owner, e) => { this.setState({ modal: !this.state.modal, edit: !this.state.edit, name: name, members: members, id: id, owner: owner }); }; toggleTaskModal = e => { this.setState({ modal: !this.state.modal, task: !this.state.task }); }; toggleEditTaskModal = e => { this.setState({ modal: !this.state.modal, editTask: !this.state.editTask }); }; componentDidMount() { this.props.getProject(this.props.match.params.project); this.props.getTasks(this.props.match.params.project); } componentDidUpdate(prevProps) { if (this.props.match.params.project !== prevProps.match.params.project) { this.props.getProject(this.props.match.params.project); this.props.getTasks(this.props.match.params.project); } } onChange = async e => { await this.setState({ tasks: this.props.tasks.tasks }); let tasks = await [...this.state.tasks]; await alert(tasks[e.target.id].taskName); tasks[e.target.id].taskName = await e.target.value; await this.setState({ tasks }); }; render() { const { tasks } = this.props.tasks; let tasksList = tasks.map((task, index) => (
alert("TODO")}> check_circle {task.taskName} {task.assignee === this.props.auth.user.email ? "You" : task.assignee || "Unassigned"} {task.dateDue === "Date undefined" ? "Not Set" : task.dateDue}
)); if (this.props.project && this.props.project.teamMembers && !this.props.projects.projectLoading && !this.props.tasks.tasksLoading) { const { project } = this.props; return (

{project.name}

Edit Project Info
Add task

Assignee

Due

{tasksList}
); } return (
); } } const mapStateToProps = state => ({ auth: state.auth, project: state.projects.project, projects: state.projects, tasks: state.tasks }); export default connect(mapStateToProps, { getProject, getTasks })(Project);

DashboardComponents.js

import React, { Component } from "react"; import "./MainContent.scss"; import "./Dashboard.scss"; import { connect } from "react-redux"; import Modal from "./Modal/Modal"; class Dashboard extends Component { state = { modal: false, edit: false, name: "", members: [], id: "", owner: {} }; toggleModal = e => { this.setState({ modal: !this.state.modal, edit: false }); }; toggleEditModal = (name, members, id, owner, e) => { e.stopPropagation(); this.setState({ modal: !this.state.modal, edit: !this.state.edit, name: name, members: members, id: id, owner: owner }); }; render() { const { projects } = this.props.projects; let content; let projectData = projects.sort().map(project => (
this.props.history.push(/projects/project._id)} >
{project.name}
Edit project
Go to project
)); if (projects.length > 0) { // At least one project content = (<> Create another project
{projectData}
); } else { // No projects content = (<>

You have no projects

Create your first project
); } return (

Your Projects

{content}
); } } const mapStateToProps = state => ({ projects: state.projects }); export default connect(mapStateToProps, {})(Dashboard);

TasksComponents.js

import React, { Component } from "react"; import "./MainContent.scss"; import { connect } from "react-redux"; import Modal from "./Modal/Modal"; class Tasks extends Component { state = { modal: false }; toggleModal = e => { this.setState({

modal: !this.state.modal }); }; render() { const { projects } = this.props.projects; return (

Your Tasks

You have no tasks

{projects.length > 0 ? (

Visit a project to create your first task

) : (Create your first project)}
); } } const mapStateToProps = state => ({ projects: state.projects }); export default connect(mapStateToProps, {})(Tasks);

SideNavComponents.js

import React, { Component } from "react"; import { NavLink, Link, withRouter } from "react-router-dom"; import { connect } from "react-redux"; import { logoutUser } from "../../../actions/authActions"; import "./SideNav.scss"; class SideNav extends Component { onLogoutClick = e => { this.props.logoutUser(); }; // Hide Side Nav toggleMenu = e => { let sideNav = document.querySelector(".side"); sideNav.classList.add("invisibile"); let hamburger = document.querySelector(".hamburger-top-menu"); hamburger.classList.add("hamburger-visible"); let rightSide = document.querySelector(".right"); rightSide.classList.add("no-side"); let rightSideRight = document.querySelector(".right-top"); rightSideRight.classList.add("right-top-visibile"); }; render() { const { projects } = this.props.projects; let projectData = projects.sort().map(project => (

- {project.name}

)); return (

- *menu*
- *home*Home
- *check_circle*My Tasks
- *arrow_back*Sign Out

{projects.length > 0 && (

- **Projects**

  {projectData}

)}
); } } const mapStateToProps = state => ({ projects: state.projects }); export default withRouter(connect(mapStateToProps, { logoutUser })(SideNav));

TopNavComponents.js

import React, { Component } from "react"; import { connect } from "react-redux"; import { logoutUser } from "../../../actions/authActions"; import { Link } from "react-router-dom"; import "./TopNav.scss"; class TopNav extends Component { state = { dropdown: false }; componentDidMount() { document.addEventListener("mousedown", this.handleClick, false); } componentWillUnmount() { document.removeEventListener("mousedown", this.handleClick, false); } // Close dropdown when click outside handleClick = e => { if (this.state.dropdown && !this.node.contains(e.target)) { this.setState({ dropdown: !this.state.dropdown }); } }; onLogoutClick = e => { e.preventDefault(); this.props.logoutUser(); }; handleProfileClick = e => { this.setState({ dropdown: !this.state.dropdown }); }; // Show Side Nav toggleMenu = e => { let sideNav = document.querySelector(".side"); sideNav.classList.remove("invisibile"); let hamburger = document.querySelector(".hamburger-top-menu"); hamburger.classList.remove("hamburger-visible"); let rightSide = document.querySelector(".right"); rightSide.classList.remove("no-side"); let rightSideRight = document.querySelector(".right-top"); rightSideRight.classList.remove("right-top-visibile"); }; render() { const { name, email } = this.props.auth.user; return (
(this.node = node)}>
menu

Teams

- Signed in as {email}

  Not you?{" "} Sign Out

- {name !== undefined && name.split("")[0]}
  {this.state.dropdown ? (

    Hello, {name !== undefined && name.split(" ")[0]}

- Home
- My Tasks
- Sign Out

) : null}

); } } const mapStateToProps = state => ({ auth: state.auth }); export default connect(mapStateToProps, { logoutUser }) (TopNav);

LayoutComponents.js

import React, { Component } from "react"; import PropTypes from "prop-types"; import { connect } from "react-redux"; import { getProjects } from "../../actions/projectsActions"; import { BrowserRouter as Router, Route, Switch, withRouter } from "react-router-dom"; import Spinner from "../common/Spinner"; import SideNav from "./SideNav/SideNav"; import TopNav from "./TopNav/TopNav"; import Dashboard from "./MainContent/Dashboard"; import Tasks from "./MainContent/Tasks"; import Project from "./MainContent/Project/Project"; import NotFound from "../404/404"; import "./Layout.scss"; class Layout extends Component { componentDidMount() { this.props.getProjects(); } render() { const { projects, projectsLoading } = this.props.projects; let dashboardContent; if (projects === null || projectsLoading) { dashboardContent = ; } else if (projects.length > 0) { dashboardContent = (<>
); } else { dashboardContent = (<>
); } return (
{dashboardContent}
); } } Layout.propTypes = { auth: PropTypes.object.isRequired }; const mapStateToProps = state => ({ auth: state.auth, projects: state.projects }); export default withRouter(connect(mapStateToProps, { getProjects })(Layout));

AppComponents.js

// React import React, { Component } from "react"; import { BrowserRouter as Router, Route, Switch } from "react-router-dom"; // Utils import jwt_decode from "jwt-decode"; import setAuthToken from "./utils/setAuthToken"; // Redux import { Provider } from "react-redux"; import store from "./store"; import { setCurrentUser, logoutUser } from "./actions/authActions"; // Components import Register from "./components/auth/Register"; import Login from "./components/auth/Login"; import PrivateRoute from "./components/private-route/PrivateRoute"; import Layout from "./components/dashboard/Layout"; import NotFound from "./components/404/404"; // Style import "./App.scss"; // Check for token to keep user logged in if (localStorage.jwtTokenTeams) { // Set auth token header auth const token = localStorage.jwtTokenTeams; setAuthToken(token); // Decode token and get user info and exp const decoded = jwt_decode(token); // Set user and isAuthenticated store.dispatch(setCurrentUser(decoded)); // Check for expired token const currentTime = Date.now() / 1000; // to get in milliseconds if (decoded.exp < currentTime) { // Logout user store.dispatch(logoutUser()); // Redirect to login window.location.href = "./"; } } class App extends Component { render() { return (
); } } export default App;