CRUD MERN

By: Pratik Kumar

Folder structure:

Make 2 folders name client and server

Client :

- This is for the react part
- Steps:
  1. npx create-react-app ./
  2. packages: axios, moment, react-file-base64, redux, redux-thunk, react-redux
  3. clean the src and create a initial app and index setup.
  4. In package.json add “proxy”: “http://localhost:5000”

Server :

- This is for the express and mongodb
- Steps:
  1. npm init -y
  2. packages: express, mongoose, nodemon, cors, body-parser
  3. in package.json add “type” : “module”, “scripts” : {“start” : “nodemon index.js”}
  4. import express mongoose cors and body-parser in index.js

General server setup

- Initialize express: const app = express();
- app.use(bodyParser.json({limit: “30mb”, extended: true}));
- app.use(bodyParser.urlencoded({limit: “30mb”, extended: true}))
- app.use(cors())

General database setup

- Create a database in mongodb cloud
- const CONNECTION_URL = “url of the mongodb cluster”
- const PORT = 5000
- mongoose.connect(CONNECTION_URL, {newUrlParser:true, useUnifiedTopology:true})
  .then(()=>{app.listen(PORT, ()=>{console.log(`Server running on port ${PORT}`)})})
  .catch((error)=>{console.log(error.message)})

Creating Routes

- Create a routes folder for backend routes
- In routes folder create a file posts.js for posts routes
- In posts.js
  - Import express
  - Import Router: const router = express.Router();
  - Create a req:
  - Router.get(‘/’, (req, res) =>{res.send(“This works!”)})
  - export default router
- In index.js
  - import postRoutes from “…posts.js”
  - app.use(‘/posts’, postRoute)

Setting up Controllers

- Use to create handlers for the routes
- Create a controllers folder
- In the controllers folder create a file posts.js for the handlers of posts routes

- In controllers/posts.js
  - Create the handlers
  - export const getPosts = (req, res) =>{res.send("This works!")}
- In routes/posts.js
  - Import and add the handlers
  - Import getPosts from "…posts.js"
  - Router.get('/',getPosts)

Creating Models

- Create a models folder for backend models
- In models folder create a file postMessage.js for post schema
- In postMessage.js
  - Import mongoose
  - Create post schema
  - const postSchema = mongoose.Schema({

```
title:{
        type:String,
         required: ture
        }
 date:{
        type:Date,
         default: new Date()
        }
})
```

  - **define the model**
  - const PostMessage = mongoose.model('PostMessage', postSchema);
  - **export the model**
  - export default PostMessage

Creating Controllers

- import PostMessage from "…models/postMessage.js"
- **for getPosts**

```
export const getPosts = async () => {
        try{
                const postMessages = await postMessage.find()
                res.status(200).json(postMessages)
        } catch (error){
                res.status(404).json({message:error.message})
        }
}
```

- **for createPost**

```
export const createPost = async () => {
        const post = req.body
        const newPost = new PostMessage(post)
        try{
                await newPost.save()
                res.status(201).json(newPost)
        } catch (error){
                res.status(409).json({message:error.message})
        }
}
```

Client setup

- Create the components and screens setup.

Api client

- Create api folder in src of client
- Create file index.js in api
- import axios from 'axios'
- const url = https://localhost:5000/posts
- expost const fetchPosts = () => axios.get(url)

Redux setup

- Create 2 folders actions and reducers
- Create file named posts.js in both actions and reducers
- Create file index.js in reducers
- In src/index.js
  - **Import redux tools**
  - import {Provider} from “react-redux”
  - import {createStore, applyMiddleware, compose} from ‘redux’
  - import thunk from “redux-thunk”
  - import reducers from “./reducers”
  - **Now create the store**
  - const store = createStore(reducers, compose(applyMiddleware(thunk)))
  - **Finally** wrap <App/> in Provider and store = {store}
- Creating Reducers
  - **In reducers/index.js**
  - import {combineReducers} from “redux”
  - import posts from ‘./posts’
  - export default combineReducers({

    posts,

    })
  - **In reducers/posts.js create reducer for posts**
  - export default (posts = [], action) {

    switch(action.type){

    case ‘FETCH_ALL’:

    return posts;

```
        case 'CREATE':
            return posts;
        default:
            return posts;
    }
}
```

- **Create dispatch in src/app.js to dispatch actions**
- Import {useDispatch} from 'react-redux'
- Inside the App function
- Import {getPosts} from "./actions/posts"
- const dispatch = useDispatch()
- useEffect(() => {
  dispatch(getPosts())
  }, [dispatch])

➢ **Creating actions**

- In actions/posts
- Import * as api from '../api'
- export const getPosts = () async (dispatch) => {
  try (
  const {data} = await api.fetchPosts()
  dispatch({type: 'FETCH_ALL', payload: data})
)
  catch (error) {
  console.log(error.message)

```
        }
    }
```

- **Updating the Reducers**
  - In reducers/posts.js
  - In “FETCH_ALL” return action.payload

- **Retrieve the data in the component/posts/posts.js**
  - import {useSelector} from ‘react-redux’
  - Inside Post function
  - const posts = useSelector((state) => state.posts)

Creating the form

- Create a form with all the req. fields
- useState with the initial form values
- const [postData, setPostData] = useState({
  name: “”,
  email: “”,
  …..
  })
- onChange = {handleChange}
- const handleChange = (e) => {
  setPostData({…postData, [e.target.name]:e.target.value})
  }
- onSubmit = {handleSubmit}

Api and actions for the create post

- In the api/index.js
  - Create axios.get
  - export const createPost = (newPost) => axios.get(url, newPost)
- In actions/posts.js
  - Create action for post
  - export const createPost = (post) async (dispatch) => {

```
try (
        const {data} = await api.createPost(post)
        dispatch({type: 'CREATE', payload: data})
)
catch (error) {
        console.log(error.message)
}
}
```

Dispatch the create action

- import useDispatch in form.js
- const dispatch = useDispatch()
- dispatch the data on submit
  - import createPost from actions/posts
  - const handleSubmit = (e) => {

```
e.preventDefault();

dispatch(createPost(postData))

}
```

- in reducer/posts.js
  - case 'CREATE':

- return […posts, action.payload]

Viewing the posts in Frontend

- const posts = useSelector((state) => state.posts)
- {posts.map((post) => (

  <Post post = {post} />

  ))}
- In the posts/post/post.js
  - Create a Post function with all the props value

Updating the post

- In server/routes/posts.js add the patch route
  - router.patch('/:id', updatePost)

- In server/controllers/posts.js create the updatePost function
  - const updatePost = async (req, res) => {
    const {id : _id} = req.params
    const post = req.body
    if(!mongoose.Types.ObjectId.isValid(_id)) {
    return res.status(404).send("No posts with that id")
    }
    const updatePost =
    await PostMessage.findByIdAndUpdate(_id, post, {new: true})
    res.json(updatePost)
    }

- In client/App.js
  - const [currentId, setCurrentId] = useState(null)
  - <Form currentId={currentId} setCurrentId={setCurrentId} />
  - <Post setCurrentId={setCurrentId} />
  - Pass the values in the posts and forms
  - const Form = ({currentId, setCurrentId})
  - const Posts = ({setCurrentId})
  - <Post post = {post} setCurrentId={setCurrentId} />
  - Add currentId in the useEffect []

- In the edit Button of posts/post/post.js
  - onClick = {()=>{setCurrentId(props._id)}}

- In form/Form.js
  - If(currentId) {
    dispatch(updatePost(currentId, postData))
    } else {
    dispatch(createPost(postData))
    }

- In api/index.js
  - Add the updatePost api
  - export const updatePost = (id, updatedPost) =
    axios.patch(`${url}/${id}`, updatedPost)

- In actions/posts.js
  - Create the updatePost action creator
  - export const updatePost = (id, post) => async(dispatch) => {

```
try{
        const {data} = await api.updatePost(id,post)
        dispatch({type: 'UPDATE', payload: data})
} catch (error) {
        console.log(error)
}
}
```

- In reducers/posts.js
  - Creating reducer for update
  - case UPDATE :

```
return posts.map((post) => post._id === action.payload.id
        ?  action.payload : post)
```

- In from.js
  - import {useSelector}
  - const post = useSelector((state) => currentId

```
? state.posts.find((p) => p._id===currentId) : null )
```

- useEffect(() => {

  if(post) setPostData(post)

  }, [post])
- in clear function
- const clear = () => {

  setCurrentId(null);

  setPostData({initial})

  }
- call clear function in handle submit

Delete the post

- create the route
  - router.delete('/:id', deletePost)

- create the delete controller
  - const deletePost = async (req, res) => {

    const {id} = req.params

    if(!mongoose.Types.ObjectId.isValid(id)) {

    return res.status(404).send("No posts with that id")

    }

    await PostMessage.findByIdAndDelete(id)

    res.json(message: 'Post deleted successfully')

    }

- create api call for delete
  - export const deletePost = (id) => axios.delete(`${url}/${id}`)

- Create a action creator
  - export const deletePost = (id) => async(dispatch) => {

```
try{
        await api.deletePost(id)
        dispatch({type: 'DELETE', payload: true})
} catch (error) {
        console.log(error)
}
}
```

- Add the delete case in reducer
  - Case 'DELETE' :

```
return posts.filter((post) => post._id !== action.payload)
```

- Dispatch the delete
  - onClick = {() => dispatch(deletePost(props._id))}

Update likeCount the post

- create the route
  - router.patch('/:id/likePost', likePost)

- create the delete controller
  - const likePost = async (req, res) => {

    const {id} = req.params

    if(!mongoose.Types.ObjectId.isValid(id)) {

    return res.status(404).send("No posts with that id")

    }

    const post = await PostMessage.findById(id);

    const updatedPost = await PostMessage.findByIdAndUpdate

    (id, {likeCount: post.likeCount + 1}, {new: true})

    res.json(updatePost)

    }

- create api call for delete
  - export const likePost = (id) => axios.patch(`${url}/${id}/likePost`)

- Create a action creator
  - export const likePost = (id) => async(dispatch) => {

    try{

    const {data} = await api.likePost(id)

    dispatch({type: 'LIKE, payload: true})

    } catch (error) {

    console.log(error)

    }

    }

- Add the delete case in reducer
  - Case ‘LIKE :

    return posts.map((post) => post._id === action.payload._id

    ? action.payload : post)

- Dispatch the delete
  - onClick = {() => dispatch(likePost(props._id))}

Adding action constants

- create a folder named constant in the client/src
- In there create a file name actionType.js
- Then create the actions
  - export const FETCH_ALL = ‘FETCH_ALL’
  - export const CREATE = ‘CREATE’
  - export const UPDATE = ‘UPDATE’
  - export const DELETE = ‘DELETE’
  - export const LIKE = ‘LIKE’
- import these in actions/posts.js and reducers/posts.js and replace

Environmental variables

- npm install dotenv
- create a file named .env in server
- import dotenv from ‘dotenv’ and add dotenv.config() below express()

- In .env file
  - CONNECTION_URL = mongo_url
- In index.js
  - Import dotenv
  - Add dotenv.config()
  - Change CONNECTION_URL to process.env.CONNECTION_URL

Deployment

- For the client side deploy it on GitHub using gh-pages and change the url in api
- For the server
  - Add a '/' route
  - Create a file name Procfile in server.js
  - In there type web:npm run start
  - Install Heroku cli
  - Install global Heroku package
  - heroku login
  - heroku git:remote -a memories-project
  - git add .
  - git commit -am "Publishing"
  - git push heroku master