Wednesday, May 24, 2023 11:10 AM

Client Server Index.js Index.js · createStore to hold the complete state tree of the app and pass in the reducers and middleWare We're using 'thunk' middleWare to intercept dispatched actions and write action creators that return functions instead of plain action objects Sets up a server Connects to ./api/index.js Render app in the root element: createPost = (newPost) => axios.post(url, newPost) Axios makes requests to the server side to carry out the functions on the backend. · Basically we declare a bunch of functions that are used by ./actions/posts.js. This file just sends a request to the backend and tells it to make mongoDB some changes and the backend responds back with some data. res.json(newPost) Server listening on 127.0.0.1:3000 App.is useState returns a variable item and an updater function. set/tem which is used to update the value of the item. await newPost.save() . dispatch sends your action to the store which is then sent to all data router.post('/', createPost); the reducers in the application. So it basically triggers the state api.createPost(post) //received update process. new data • useEffect is used for responding to the changes in the component lifecycle ./controllers/posts.js > Header bar Posts Grid // do something with the postMessage ./routes/posts.js > Form Grid // the main logic behind all the functions ./actions/posts.js '/' getPost • setCurrentId={setCurrentId} in both <Posts /> and <Form /> Const getPost = async (req, res) => { ... } '/' createPost to ensure that the changes happen to the lowest child in the Const createPost = async (req, res) => { ... } • Here we declare the core dispatching functions which are used by the components. '/:id' updatePost object tree from here. • We import all the functions from ../api so that we can make calls the to backend '/:id' deletePost /:id/likePost' likePost and to get the data from the backend and then dispatch it to the app. • We specify the type for each dispatch so that the reducers update the state tree based on the type of the action. So when you call lets say updatePost() in Forms.js, it calls that function with (currentld, postData) and dispatches it to the Redux store. Dispatching can be a bit confusing to understand at first but you get the hang of it over time. Form.js /models/postMessage.js Posts.js Takes input from the user Sets up a <Grid /> to This is basically the format or structure for the data Uses createPost and display all the individual object that will be stored in the database. dispatch({ type: CREATE, payload: data }) undatePost from Jactions on posts button click. createPost(postData), updatePost(id, postData) ./reducers/posts.js • Reducers in are responsible for specifying how the application's state Post.js should be updated in response to dispatched actions. return [ ...posts, payload ] . It returns the next state tree given the current state tree and action. deletePost(id), likePost(id) · Initializes the post object with Here it returns responses for the actions that we sent all it's attributes from the from ./actions/post.js When the three dots are clicked, currentld is set to post.\_id. This triggers the Form to be populated for calling updatePost()