State Management

Context API

Context provides a way to pass data through the component tree without having to pass props down manually at every level. Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language. You can read more about it [here](https://reactjs.org/docs/context.html).

We used it to manage the app settings. Once the project is mounted we extract the stored settings from localStorage, pass the data to context provider and then use it when needed.

Redux

Redux is more robust alternative and currently offers more helpful features. Here is a [link](https://redux.js.org/basics/usage-with-react) that can help you understand the entire concept behind it.

The project uses Redux Toolkit with Hooks (Thunk method, not Sagas) to help you manage the applications chat, kanban, mail, etc. The clear project structure allows you to replace it whenever you need to with ease. There are 2 main folders src/slices, where the reducer logic is being implemented, and src/store that creates the store and combines the reducers in one root reducer. If you’re new please read the official documentation from [Redux Toolkit](https://redux-toolkit.js.org/usage/usage-guide) to understand the basics.

Creating a new slice

For this example let’s say that you have a blog and you want to load the articles in Redux and then render the data somewhere in your application.

First, open src/slices folder and create a new file, blog.js. Now open this new file and import createSlice from @reduxjs/toolkit package.

import { createSlice } from '@reduxjs/toolkit';

And then use this helper to create our blog slice and export it as a default module.

const initialState = {

articles: []

};

const slice = createSlice({

name: 'blog',

initialState,

reducers: {}

});

export default slice;

Now in the src/store/rootReducer.js import the reducer from our new slice.

import { reducer as blogReducer } from 'src/slices/blog';

const rootReducer = combineReducers({

// rest of reducers

blog: blogReducer

});

At this point we have a slice, where it’s initial state is an object with a key articles. In this key we are going to store our data.

Using the slice state

Since our state at the moment as new articles, nothing will be rendered.

// BlogView.js

import { useSelector } from 'react-redux';

const BlogView = () => {

const { articles } = useSelector((state) => state.blog);

return (

<div>

<div>

{articles.map(article) => (

<div>

Article ID: {article.id}

</div>

)}

</div>

</div>

);

};

Create a reducer action

Lets get back to our slice and add a setArticles action.

const slice = createSlice({

name: 'blog',

initialState,

reducers: {

setArticles(state, payload) {

// We should get articles from the action payload but for the moment lets use a static data

state.articles = [

{

id: '1',

title: 'My first article'

}

];

}

}

});

When executing this action, we set the current state to have the articles key an array of objects.

**IMPORTANT**

The state is a Proxy object, you cannot update it directly and you have to update its keys instead. For example state = { name: 'Alex' } will break the state, but state.user = { name: 'Alex' } works as expected.

Creating a thunk method

This allows you to dispatch multiple actions, extremely useful for async requests.

In our slice file, create a getArticles method that dispatches the slice setArticles action.

export const getArticles = () => async (dispatch) => {

// Here make an async request to your sever and extract the date from the server response

// const response = await axios.get('/api/blog/articles');

// const { data } = reponse;

const data = [

{

id: '1',

title: 'My first article'

}

];

dispatch(slice.actions.setArticles(data));

};

Now that we have a method that dispatches an action, we can update the setArticles action to use the data from the payload.

reducers: {

setArticles(state, payload) {

state.articles = payload.articles;

}

}

Dispatching an action

This is the last step of the process, executing the method getArticles to dispatch our slice action and update the state data. Once the action sets the new state, the app is being re-rendered and we display our articles.

// BlogView.js

import { useDispatch, useSelector } from 'react-redux';

import { getArticles } from 'src/slices/blog';

const BlogView = () => {

const dispatch = useDispatch();

const { articles } = useSelector((state) => state.blog);

const handleClick = () => {

dispatch(getArticles());

};

return (

<div>

<div>

{articles.map(article) => (

<div>

Article ID: {article.id}

</div>

)}

</div>

<Button onClick={handleClick}>

Load articles

</Button>

</div>

);

};