Example

During this skill, we are going to explore an example starting from scratch. To see the big picture.

First, let’s start by creating our react application

```
$ npx create-react-app blog
```

Now let’s set the component tree, we are going to create a folder component that contains the component we are using

under src/component creat these files PostList.js and CreatePost.js

PostList will receive the list of articles from the store and CreatePost.js will add new posts into the store.

```
//PostList.js
import React from 'react'

const PostList = (props) => {
    return (
        <div>
            {props.posts.map((post) => <div id={post.id}>
                <h1>{post.title}</h1>
                <p>{post.content}</p>
            </div>)}
        </div>
    )
}

export default PostList

//CreatePost.js
import React, { useState } from 'react'
```

```
const CreatePost = () => {
    const [title, setTitle] = useState('')
    const [content, setContent] = useState('');
    const handleSubmit = (e) => {
        e.preventDefault()
    }
    return (
        <form onSubmit={handleSubmit}>
            <div>

                <label htmlFor="Title">Title</label>
                <input type="text" name="title" id="title" onChange={e => setTitle(e
            </div>
            <div>
                <label htmlFor="Content">Content:</label>
                <textarea name="content" id="content" cols="30" rows="10" onChange={
            </div>
            <div>
                <input type="submit" value="Add" />
            </div>



        </form>
    )
}

export default CreatePost

//App.js
import CreatePost from './Components/CreatePost';
import PostList from './Components/PostList';

function App() {
  return (
    <div className="App">
      <CreatePost/>
      <PostList/>
    </div>
  );
}

export default App;
```

Setup Redux

To start working with Redux, first, we start by installing react and react-redux library

```
$ npm i redux react-redux
```

Then we are going to follow the architecture that we talked about in the previous slides.

```
//src/JS/store.js
import { createStore } from 'redux'
import rootReducer from '../Reducers/rootReducer'

const store = createStore(rootReducer)
export default store;

//src/JS/Reducers/rootReducers.js
import { ADD_ARTICLE } from "../Constants/actions-types";

const initialState = {
    posts: [
        {
            id: 1,
            title: 'my first post',
            content: 'my first content'
        }
    ]

}

const rootReducer = (state = initialState, action) => {

    switch (action.type) {

        case ADD_ARTICLE:

            return {

                posts: [...state.posts, action.payload]

            }

        default:

            return state

    }

}


export default rootReducer


// src/JS/Constants/actions-types.js

export const ADD_ARTICLE = 'ADD_ARTICLE'


// src/JS/Actions/actions.js

import { ADD_ARTICLE } from "../Constants/actions-types";


export const addPost = newPost => {

    return {

        type: ADD_ARTICLE,

        payload: newPost

    }

}
```

Connect the store React application

To connect the store to your react application, you are going to need a component from the react-redux library named Provider.

The provider component is used to wrap the react application and it will give it access to the store.

To be sure that the application has access to the store, go ahead and change the index.js of your react application like the following example:

```
//src/index.js
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { Provider } from 'react-redux';
import store from './JS/store';

ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}> {/* the component Provider needs a props store  */}
      <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById('root')
);
```

As you can see from the code above, the component Provider needs a props named store, through this `props` we pass the redux `store` to the wrapped hierarchy (which in our case the whole App)

Connect the component to get the state

In the previous slide, we saw how to make the store accessible for the hierarchy. Every component in that hierarchy gets the right to connect and use the data that exists in the Store. But this is not enough. The component itself needs to be connected to the store.

The react-redux library provides us with another method `connect`. It provides its connected component with the pieces of the data it needs from the store, and the functions it can use to dispatch actions to the store.

It does not modify the component passed to it; instead, it returns a new, connected component class that wraps the component you passed in.

The syntax of the connect method might seem a little weird but let's take a look at it.

```
connect(mapStateToProps, mapDispatchToProps)(Component)
```

Let's explain it now:

- `mapStateToProps`: it's a function that returns the state (or only a portion of the state) and passes it to the Component as a props. this function is not provided by the library it needs to be implemented by you.
- `mapDispatchToProps`: it does something similar, but for actions. mapDispatchToProps connects Redux actions to React props. This way a connected React component will be able to send messages to the store.
- `Component`: The component that wish to connect to the store.

Let's take a look at the file PostList.js to better understand.

```
import React from 'react'
import { connect } from 'react-redux'

const mapStateToProps = state => {
    return {
        posts: state.posts
    }
}
const PostList = (props) => {
    return (
        <div>
            {props.posts.map((post) => <div id={post.id}>
                <h1>{post.title}</h1>
                <p>{post.content}</p>
            </div>)}
        </div>
    )
}

export default connect(mapStateToProps)(PostList)
```

Handle Actions

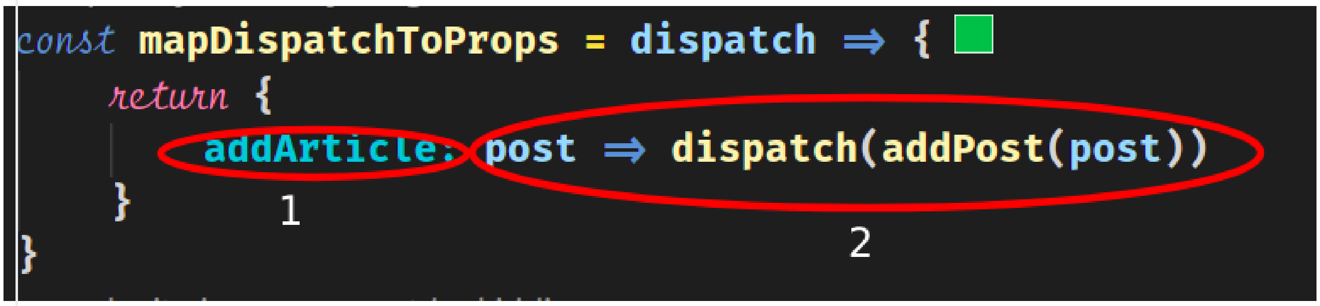
The next step in our project is to make the component CreatePost work and add the new posts to the store. To do so, we are going to use the mapDispatchToProps function, like shown in the example below

```
import React, { useState } from 'react'
import { connect } from 'react-redux';
import { addPost } from '../JS/Actions/actions';


const mapDispatchToProps = dispatch => {
    return {
        addArticle: post => dispatch(addPost(post))
    }
}
const CreatePost = (props) => {
    const [title, setTitle] = useState('')
    const [content, setContent] = useState('');
    const handleSubmit = (e) => {
        e.preventDefault()
        props.addArticle({
            id: Date.now(),
            title,
            content
        })
    }
    return (
        <form onSubmit={handleSubmit}>
            <div>


                <label htmlFor="Title">Title</label>
                <input type="text" name="title" id="title" onChange={e => setTitle(e
            </div>
            <div>
                <label htmlFor="Content">Content:</label>
                <textarea name="content" id="content" cols="30" rows="10" onChange={
```

```
            </div>

            <div>

                <input type="submit" value="Add" />

            </div>




        </form>

    )

}


export default connect(null, mapDispatchToProps)(CreatePost)
```

The mapDispatchToProps is the second argument that the connect method need so that's why we passed the null instead of mapSateToProps but both methods can be passed if needed.

The function `mapDispatchToProps` takes as a parameter `dispatch` and returns an object.



```
const mapDispatchToProps = dispatch => {
    return {
        addArticle: post => dispatch(addPost(post))
    }
}
```

1. `addAtricle` : The key of the first element in the returned object. It's the one that will be used in the component (props.AddArticle)
2. function: That takes a post as an argument (the new post to be added) and dispatch the action of adding that new post.

Go ahead and try this example for your own. You can find the whole project at this link

‹ Previous next ›