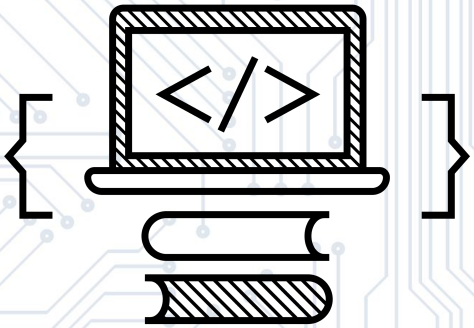
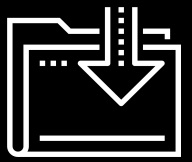


State Management with the Store

Web Development Boot Camp

Lesson 10.5



Let’s Talk About **Pure Functions**

And why they’re better than impure functions.

Pure Functions

- Pure functions are straightforward with singular purposes.
- No side effects inside the function.
- Whenever data is modified, it is stored in a new variable.

Impure Functions

- Include side effects like database and network calls.
- Often modify the values that are passed in.

Here's an Impure Function

```
const tripleInput = (inp) => {
  // Modifies existing parameter
  inp = inp * 3
  // Contains a call to the database
  writeToDB(inp);

  return inp;
}
```

Here's a Pure Function

```
const tripleInput = (inp) => {
    const tripledResult = inp * 3;
    return tripledResult;
}
```

Creating pure functions is an effective technique to apply to all of your JavaScript, but it is especially useful when creating React components.

Redux

Redux is a state management library that helps organize the state of complex applications. Redux requires a lot of boilerplate code, but its utility skyrockets as applications get more complex.

Instead of learning about the Redux library, we will focus on the concepts and design patterns that Redux uses. After learning how to create each function from scratch, we will implement a Redux-like state management system in a small CMS app.

Actions, Reducers, and Store

Today, we will cover three core Redux topics

01

Action

A declarative object that defines what the store should do to the state.

02

Reducer

Handles all actions by taking in the previous state and returning a new state.

03

Store

Grants the entire application access to the reducer function and the global state.

The Principles of Redux

Principle 1: State Tree

The first premise of Redux is that every stateful aspect of your application can be represented by a single JavaScript object known as the state tree.

```
console.log(store.getState())

/* Prints
{
  visibilityFilter: 'SHOW_ALL',
  todos: [
    {
      text: 'Consider using Redux',
      completed: true,
    },
    {
      text: 'Keep all state in a single tree',
      completed: false
    }
  ]
}
*/
```

Principle 2: Read Only

The second principle of Redux is that the state tree is read-only. This means that in order to change the state tree, you need to dispatch an action. The action describes the change that the state will undergo in a declarative manner.

```
store.dispatch({
  type: 'COMPLETE_TODO',
  index: 1
})

store.dispatch({
  type: 'SET_VISIBILITY_FILTER',
  filter: 'SHOW_COMPLETED'
})
```

Principle 3: Changes are made with pure functions

Try to make your reducers pure functions. It is considered bad practice to use side effects inside reducer functions.

```
function visibilityFilter(state =
'SHOW_ALL', action) {
  switch (action.type) {
    case 'SET_VISIBILITY_FILTER':
      return action.filter
    default:
      return state
  }
}
```

```
case 'COMPLETE_TODO':
      return state.map((todo, index) => {
        if (index === action.index) {
          return Object.assign({}, todo, {
            completed: true
          })
        }
        return todo
      })
    default:
      return state
  }
}
```

Let’s Get Started!