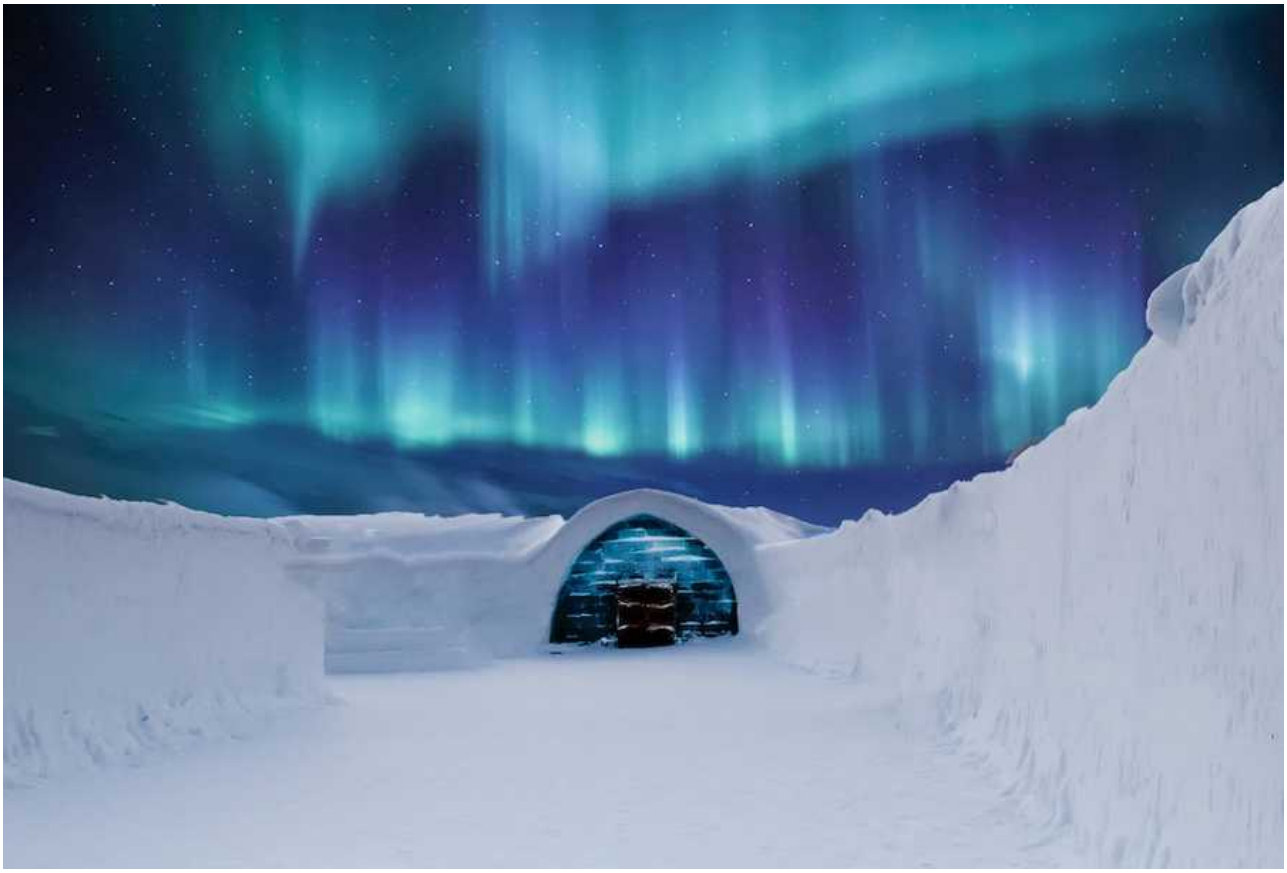
Mediator Component in React

NOVEMBER 22, 2020 BY ROBIN WIERUCH - EDIT THIS POST



When I work with clients on their React applications, I often encounter convoluted React components. Such overly complex components happen, because they have too many responsibilities -- whereas one responsibility translates to one feature and one feature translates to multiple React Hooks and event handlers. Thus, when having more than one responsibility, the component becomes convoluted with too many hooks and handlers.

Let's take for example the following React component:

```
import * as React from 'react';

import useFetch from 'some-fetch-library';

import Table, { Pagination } from 'some-table-library';
```

```
import Search from './Search';


const applyFilter = (search) => (users) => ...

const Users = () => {
  const {
    data: users,
    isLoading: isLoadingUsers,
    isError: isErrorUsers,
  } = useFetch('https://api.mydomain/users');

  const [search, setSearch] = React.useState('');

  const handleSearch = (event) => {
    setSearch(event.target.search);
  };

  const tableOptions = {
    ...
    // some giant configuration object
    ...
  };

  const filteredUsers = users.filter(user => {
    ...
    // do something with user and search
    ...
  });

  if (isLoadingUsers) {
    return <div>Loading ...</div>;
  }

  if (isErrorUsers) {
    return <div>Something went wrong ...</div>;
  }

  return (
    <div>
      <Search
        search={search}
        onSearch={handleSearch}
      />

      <Table
        users={filteredUsers}
        options={tableOptions}
      />
    </div>
  );
};
```

How many responsibilities/features do you count? The Users component takes care of 4 things:

- fetching users
- conditional rendering of loading and error states
- setting up Search component and wiring it to the lifted state
- setting up Table component and using the lifted state

Everything that ends up between the component definition and the final return statement -- e.g. handlers and hooks -- convolutes the component and makes it more complex. This is just a minimal example to illustrate the issue, however, I have encountered components with more than 1000 lines of code. It's a nightmare for every developer who has to step through these components.

So how do we fix this issue? I call it extracting functionality with so called **mediator components**. Taking the previous example, one could say that the Users component is tightly coupled to a specific domain (e.g. user domain) whereas the Search and Table components are reusable components which are most likely used by other domain components too.

Now, a mediator component would fit between the domain-driven and reusable components. For example, a domain specific table component would be the best approach to simplify the previous component:

```
import * as React from 'react';

import Table, { Pagination } from 'some-table-library';

import Search from './Search';

const applyFilter = (search) => (users) => ...

const UserTable = ({ users, tableOptions }) => {
  const [search, setSearch] = React.useState('');

  const handleSearch = (event) => {
    setSearch(event.target.search);
  };

  const defaultOptions = { ... };

  const mergedOptions = {
    ...defaultOptions,
    ...tableOptions,
  };

  return (
```

```
    <div>
      <Search

        search={search}
        onSearch={handleSearch}
      />

      <Table
        users={filteredUsers}
        options={mergedOptions}
      />
    </div>
  );
};
```

Afterward, the Users component could use this new domain specific (specialized) table component:

```
import * as React from 'react';

import useFetch from 'some-fetch-library';

import UserTable from './UserTable';

const Users = () => {
  const {
    data: users,
    isLoading: isLoadingUsers,
    isError: isErrorUsers,
  } = useFetch('https://api.mydomain/users');

  if (isLoadingUsers) {
    return <div>Loading ...</div>;
  }

  if (isErrorUsers) {
    return <div>Something went wrong ...</div>;
  }

  const userTableOptions = { ... };

  return (
    <div>
      <UserTable
        users={users}
        tableOptions={userTableOptions}
      />
    </div>
  );
};
```

Finally, the top-level domain specific Users component got simplified: it only cares about 2 out of 4 things -- data fetching and conditional rendering -- while the UserTable takes care of the rest. The Table component, while being a reusable component, receives all of its domain specific props from the UserTable component as mediator.

Show Comments

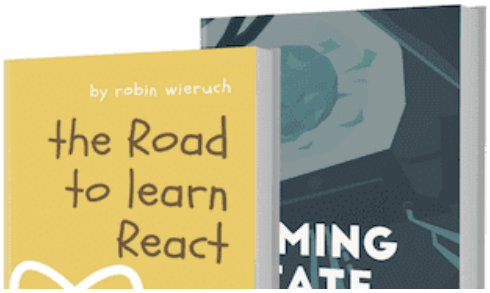
KEEP READING ABOUT REACT

REACT EVENT HANDLERS: ONCLICK, ONCHANGE ...

In this React tutorial, we will get to know event handlers in React for HTML elements such as button and input elements. You will learn how to use a button with its onClick event and how to define and...

REACT HOOKS MIGRATION

React Hooks were introduced to React to make state and side-effects available in React Function Components. Before it was only possible to have these in React Class Components; but since React's way...



THE ROAD TO REACT

Learn React by building real world applications. No setup configuration. No tooling. Plain React in 200+ pages of learning material. Learn React like **50.000+ readers**.

GET THE BOOK >

Get it on Amazon.

TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.

- ✔ Join 50.000+ Developers
- ✔ Learn Web Development with JavaScript
- ✔ Tips and Tricks
- ✔ Access Tutorials, eBooks and Courses
- ✔ Personal Development as a Software Engineer

Your email address SUBSCRIBE