React

Grupo 4

Natalia Barra - Luis Chodiman - Mauricio Ortiz

Pero primero... la demo!

Puntos principales

- Uso de **Hooks** y componentes como **clases**
- Manejo de estado con **Redux**
- Manejo de router con **React Router**
- Uso de API
- Bonus: **Tests** con enzyme y jest

Componentes en React

Componente stateless

```
import React, { Fragment } from 'react';
import Favorite from '../layout/Favorite';

const Favorites = () => (
  <Fragment>
    <h1 className="title">My favorites</h1>
    <Favorite />
  </Fragment>
);

export default Favorites;
```

Componentes en React

Componente de clase

Importaciones necesarias

```
import React from 'react';
import { connect } from 'react-redux';
import { selectAnimal, selectBreed } from '../../modules/game';
// Otros imports...

const mapDispatchToProps = { selectAnimal, selectBreed };
const mapStateToProps = state => state.game;
```

Componentes en React

Componente de clase

Constructor

```
class SelectAnimal extends React.Component {
  constructor(props) {
    super(props);
    this.state = { selected: "dog" }; // Estado inicial del componente
    this.handleChange = this.handleChange.bind(this);
  }
```

Método de la clase

```
  handleChange(event) {
    this.setState({ selected: event.target.value }); // Cambia estado del componente
    this.props.selectAnimal(event.target.value);
    this.props.selectBreed('random');
  }
```

Componentes en React

Componente de clase

Mostrar el componente

```
  render() {
    const { selected } = this.state;
    return (
      <FormControl disabled={this.props.playing} className="form">
        // Componentes del form...
      </FormControl>
    );
  }
```

Export con uso de redux

```
export default connect(
  mapStateToProps,
  mapDispatchToProps
)(SelectAnimal);
```

Manejo de estado con Redux

Para explicar su funcionamiento, nos enfocaremos en el funcionamiento del juego, con `game.js`.

Store

Define el "*state*" de TODA la aplicación.

`modules/index.js`

```
import { combineReducers } from 'redux';
import dogs from './dogs';
import cats from './cats';
import favorites from './favorites';
import game from './game';
export default combineReducers({
  dogs,
  cats,
  favorites,
  game
});
```

Cada módulo maneja su propio *state* interno.

Reducers

- "Modifican" el state asignado, según el action que reciben.
- Modificar = Copia el state, modifica esa copia y lo retorna, resultando en el **nuevo** state.

Initial State (`game.js`)

```
const initialState = {
  animals: [],
  selections: [],
  breeds: {
    cats: [],
    dogs: [],
  },
  breedsLoaded: false,
  animalSelected: 'dog',
  breedSelected: 'random',
  playing: false,
  submittedAnswer: false,
};
```

Reducer en `game.js`

```
export default (state = initialState, action) => {
  const { payload, type } = action;
  switch (type) {
    case GET_BREEDS:
      return {
        ...state,
        breeds: {
          cats: [...payload.cats],
          dogs: [...payload.dogs],
        },
        breedsLoaded: true,
      };

  // More cases

    default:
      return {
        ...state
      };
  }
};
```

Actions

- Señales que le indican al store que proceso realizar.
  - type: Describe la acción. Es lo que reciben los reducers para saber cual es el siguiente estado del store.
  - payload (opcional): Data que utiliza la store.

```
{
  type: GET_BREEDS,
  payload: {cats, dogs},
}
```

- Es recomendable que cada action sea "activada" por una función (*action creator*), que será llamada fuera de la store según lo requiera (por ejemplo, un componente).

```
export const getBreeds = () => async dispatch => {
  const cats = await fetch(cat_url+'breeds')
    .then(res => res.json())
    .then(data => data.map(cat => ({id: cat.id, name: cat.name})));
  const dogs = await fetch(dog_url+'breeds/list/all')
    .then(res => res.json())
    .then(data => Object.keys(data.message).map(dog => (
      {id: dog, name: dog}))
    );
  return dispatch({
    type: GET_BREEDS,
    payload: {cats, dogs},
  });
}
```

- Con `dispatch`, "despachamos" la action al store. El store escucha a las acciones que son llamadas con esta función.

- Los componentes pueden llamar a este *action creator*, al "conectarlos" al store (`connect`), con la función `mapDispatchToProps` .

```
import { connect } from 'react-redux';
import { getBreeds } from '../../modules/game';

// Other imports and configurations...

const mapDispatchToProps = { getBreeds };

class Game extends React.Component {
  constructor(props) {
    super(props);
    this.props.getBreeds(); //
    this.start = this.start.bind(this);
  }
  // More code...
}

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(Game);
```

- Los componentes también pueden conocer el state de la store y obtener sus valores, mediante `mapStateToProps`.

```
const mapStateToProps = state => state.game;

class Game extends React.Component {
  // constructor and others...

  async start() {

    const { animalSelected, breeds, breedSelected, startGame } = this.props;
    // destructuramos los objetos/funciones que necesitamos
    startGame();
    const selBreeds = breeds[animalSelected + 's'];
    // more code...
  }
};

export default connect(
  mapStateToProps, // Don't forget this!
  mapDispatchToProps
)(Game);
```

- Por último, hay conectar la store con toda la aplicación. Para eso, utilizamos `Provider`.

`/index.js`

```
import { Provider } from 'react-redux';
import store, { history } from './store';
import App from './App';

// ...

render(
  <Provider store={store}>
    <ConnectedRouter history={history}>
      <div>
        <App />
      </div>
    </ConnectedRouter>
  </Provider>,
  target
);
```

React Router

App.js

```
import { Route } from 'react-router-dom';
// Import de componentes

const App = () => (
  <Fragment>
    <Header />

    <main>
      <Route exact path="/" component={Home} />
      <Route exact path="/favorites" component={Favorites} />
      <Route exact path="/game" component={Game} />
    </main>
  </Fragment>
);

export default App;
```

React Router

Barra de navegación

```
import React from 'react';
import { Link } from 'react-router-dom';

const Header = () => {
  return (
    <div className="navbar">
      <div className="link">
        <Link to="/favorites">Favorites</Link>
      </div>
      Otros links...
    </div>
  );
};

export default Header;
```

Uso de API

- Uso de The Dog API y The Cat API
- Obtener imagenes aleatorias, listado de razas e imagen de raza específica

Uso de API

Ejemplo: Obtener imagenes aleatorias de perros

modules/dogs.js

```
const response = await fetch(
    'https://dog.ceo/api/breeds/image/random/18'
  ).then(res => res.json());

  const dogs = response.message;
```

Luego se usa la respuesta en la aplicación

components/images/Images.js

```
  const dogImages = dogs.map((dog, index) => {
    return <Image animal="dog" key={index} url={dog} id={index} />;
  });
```

Bonus: Testing

Tests en Redux

Importaciones necesarias:

```
import reducer, {
  ADD_FAVORITE,
  REMOVE_FAVORITE
} from '../../modules/favorites';
```

Y a testear!

```
describe('favorite reducer', () => {
  it('should return the initial state', () => {
    expect(reducer(undefined, {})).toEqual({
      favorites: []
    });
  });
```

```
it('should add favorite after ADD_FAVORITE', () => {
  expect(
    reducer(
      {
        favorites: []
      },
      {
        type: ADD_FAVORITE,
        payload: {
          url: 'some-url',
          animal: 'dog',
          id: 1
        }
      }
    )
  ).toEqual({
    favorites: [
      {
        url: 'some-url',
        animal: 'dog',
        id: 1
      }
    ]
  });
});
```

¿Así de fácil?

Sí!

- `Reducers` son funciones **puras**
- Son síncronas
- Los `input` y `output` están bien definidos

Test de componentes

Requerimientos:

```
yarn add enzyme enzyme-adapter-react-16 react-test-renderer
```

¿Para qué?

Simular `render` de componentes de `React`.

Más info: Enzyme

Un poco de configuración inicial de `Enzyme`

```
import React from 'react';
import { configure, shallow } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

import { Favorite } from '../../components/layout/Favorite';
import Image from '../../components/images/Image';

configure({ adapter: new Adapter() });

describe('<Favorite />', () => {
```

Conveniente `beforeEach` para no repetir configuración inicial.

```
let wrapper;
let favorites = [];

beforeEach(() => {
  wrapper = shallow(<Favorite favorites={favorites} />);
});
```

Ahora sí, vienen los tests!

```
it('should render three Image components if there are three favorites', () => {
  favorites = [
    {
      id: 1,
      animal: 'dog',
      url: 'some-url'
    },
    {
      id: 2,
      animal: 'cat',
      url: 'some-url'
    },
    {
      id: 3,
      animal: 'dog',
      url: 'some-url'
    }
  ];

  wrapper.setProps({
    favorites
  });

  expect(wrapper.find(Image)).toHaveLength(3);
});
```

Pero, *wait*... ¿Cómo corro los tests?

El clásico:

```
yarn test
```

Furterest