Escribiendo tests unitarios

React Redux – Unit tests

- Introducción
- Acciones síncronas
- Acciones asíncronas
- Reducers
- Selectores
- Componentes

Introducción

- Escribiremos tests **unitarios**
- Usaremos **jest** para ejecutar nuestros tests: https://jestjs.io/
- Con create-react-app ya viene configurado
  - `npm run test`
- Excepto en las acciones asíncronas, no tendremos necesidad de mocks de funciones

Acciones síncronas

- Cómo testear un action creator síncrono:
  - Creamos el objeto con la acción esperada
  - Ejecutamos el action creator
  - Comprobamos que el resultado del action creator coincide con la acción esperada

```
import * as actions from './actions';
import * as types from './types';

it('should create a REMOVE_FROM_CART action', () => {
  const bikeId = '1';
  const quantity = 5;
  const expectedAction = {
    type: types.REMOVE_FROM_CART,
    bikeId,
    quantity,
  };
  expect(actions.removeFromCart(bikeId, quantity)).toEqual(expectedAction);
});
```

Acciones asíncronas

- **Opción 1:** mockear dispatch, getState, servicios…

```
import * as actions from './actions';
import * as types from './types';
import BikesService from '../services/Bikes';
jest.mock('../services/Bikes');

describe('fetchBikes', () => {
  const bikes = [];
  const dispatch = jest.fn();
  BikesService.getAllBikes.mockResolvedValueOnce(bikes);

  it('should dispatch a FETCH_BIKES_SUCCESS action', async () => {
    await actions.fetchBikes()(dispatch, undefined, {
      services: { BikesService },
    });
    expect(dispatch).toHaveBeenNthCalledWith(1, {
      type: types.FETCH_BIKES_REQUEST,
    });
    expect(dispatch).toHaveBeenNthCalledWith(2, {
      type: types.FETCH_BIKES_SUCCESS,
      bikes,
    });
  });
});
```

Acciones asíncronas

- Opción 2: redux-mock-store y mockear servicios

```
import configureStore from 'redux-mock-store';
import thunk from 'redux-thunk';
import * as actions from './actions';
import * as types from './types';
import BikesService from '../services/Bikes';
jest.mock('../services/Bikes');

const middlewares = [thunk.withExtraArgument({ services: { BikesService } })];
const mockStore = configureStore(middlewares);
const store = mockStore({});
const bikes = [];
BikesService.getAllBikes.mockResolvedValueOnce(bikes);

it('should dispatch a FETCH_BIKES_SUCCESS actions', async () => {
  const expectedActions = [{
    type: types.FETCH_BIKES_REQUEST,
  }, {
    type: types.FETCH_BIKES_SUCCESS,
    bikes,
  }];
  await store.dispatch(actions.fetchBikes());
  expect(store.getActions()).toEqual(expectedActions);
});
```

Reducers

- Cómo testear un reducer:
  - Establecemos el estado inicial, la acción y el estado esperado
  - Ejecutamos el reducer pasando estado inicial y acción
  - Comprobamos que el resultado del reducer coincide con el estado esperado

```
import * as types from './types';
import * as reducers from './reducers';

it('should handle a ADD_TO_CART_SUCCESS action', () => {
  const initialState = [{ id: '1', stock: 10 }];
  const action = {
    type: types.ADD_TO_CART_SUCCESS,
    bikeId: '1',
    quantity: 3,
  };
  const expectedState = [{ id: '1', stock: 7 }];
  expect(reducers.bikes(initialState, action)).toEqual(expectedState);
});
```

Selectores

- Cómo testear un selector:
  - Como cualquier función javascript, pasamos parametros y comprobamos resultados
  - Intentaremos probar el mayor número de casos

```
import * as selectors from './selectors';

it('should filter bikes', () => {
  const bikes = [{ id: '1', type: 'road' }, { id: '2', type: 'mountain' }];
  const filter = 'road';
  expect(selectors.getVisibleBikes(bikes, filter)).toHaveLength(1);
});
```

Componentes

- Usaremos **enzyme** como librería de utilidades para renderizar nuestros componentes https://airbnb.io/enzyme/
  - `npm i –save-dev enzyme`
  - `npm I –save-dev enzyme-adapter-react-16`
- Podremos renderizar nuestros componentes de dos modos:
  - Shallow rendering (**shallow**)-> equivale al render()
  - Full DOM rendering (**mount**)-> renderiza todo el árbol DOM
- Con las utilidades de enzyme podemos comprobar nodos, props, state, simular eventos, etc…

Componentes no conectados a Redux

- Probando un componente no conectado a Redux

```
import React from 'react';
import { shallow } from 'enzyme';
import BikesList from './BikesList';

const props = {
  addToCart: jest.fn(),
  bikes: [
    {
      id: 'bike1',
      name: 'mountain cf 7.0',
      image: 'url/to/image',
      price: 3999.99,
      hasStock: true,
    },
  ],
};
let wrapper;

beforeEach(() => {
  wrapper = shallow(<BikesList {...props} />);
});
```

```
it('should render a List of bikes', () => {
  expect(wrapper.find('List').props().items).toHaveLength(1);
});

it('should add a bike to cart', () => {
  const [bike] = props.bikes;
  const item = shallow(
    wrapper
      .find('List')
      .props()
      .renderItem(bike),
  );
  item.find('.actions button').simulate('click');
  expect(props.addToCart).toHaveBeenCalledWith(bike.id);
});
```

Componentes conectados a Redux

Probando un componente conectado a Redux

```
import React from 'react';
import { Provider } from 'react-redux';
import { mount } from 'enzyme';
import BikesList from '../BikesList';

const store = {
  dispatch: () => {},
  getState: () => ({
    bikes: [
      {
        id: '1',
        image: 'full_exceed-cf-sl-7_c1189.png',
        name: 'Mountain CF SL 7.0',
        price: 2600,
        stock: 10,
        type: 'mountain',
      },
    ],
  }),
  subscribe: () => {},
};
```

```
it('should render a list of BikeCard', () => {
  const wrapper = mount(
    <Provider store={store}>
      <BikesList filter="all" />
    </Provider>,
  );
  expect(wrapper.find('BikeCard')).toHaveLength(1);
});
```

Componentes – Snapshot testing

- Tests muy útiles para comprobar que nuestros componentes no sufren regresiones por cambios en el código
- La primera vez que se ejecuta un test con **toMatchSnapshot()**, se genera un fichero (snapshot) con el contenido serializado del renderizado
- Las siguientes veces que se ejecute el test, **jest** comprueba que el resultado coincide con lo guardado la vez anterior, si no coincide el test **falla**
- Si el test falla, **actualizamos** el snapshot o **modificamos el código**
- Los snapshots se guardan en el repositorio de código
- Con enzyme, necesitamos un serializador: enzyme-to-json

Componentes – Snapshot testing (II)

```
import React from 'react';
import { shallow } from 'enzyme';
import BikesList from './BikesList';

describe('BikesList', () => {
  it('should render BikesList component', () => {
    const props = {
      addToCart: () => {},
      bikes: [{
        id: 'bike1',
        name: 'mountain cf 7.0',
        image: 'url/to/image',
        price: 3999.99,
        hasStock: true,
      }],
    };
    const wrapper = shallow(<BikesList {...props} />);
    expect(wrapper).toMatchSnapshot();
  });
});
```

```
// Jest Snapshot v1, https://goo.gl/fbAQLP
exports[`BikesList should render BikesList component 1`] = `
<div
  className="bikes-list"
>
  Bikes List
  <List
    className="list"
    items={
      Array [
        Object {
          "hasStock": true,
          "id": "bike1",
          "image": "url/to/image",
          "name": "mountain cf 7.0",
          "price": 3999.99,
        },
      ]
    }
    renderItem={[Function]}
  />
</div>
`;
```