TEST

Summary

During this chapter we will see

- what is the purpose of testing?
- how can we test a frontend?
- Exercise
- time: 2-3h

Tests

Tests

Tests are necessarily when creating and application or an API. As you and the team implement new solution you have to - each time - try that this is working

Let's take a minutes to think why we have to test a solution

Tests

- Be sure that the solution is working
- Be sure that we did not add bugs on working features
- Test that be didn't forgot to handle edge case (like errors or specials case - think specials chars in some cases)

Tests

Because when we write code we cannot guarantee that we do not add bugs to a project we want to be sure that everything is working before pushing it to production

But when an application growth how can we test all the features ? Think of an app or API like facebook or twitter. An human or a team cannot test all the available features each time manually

Tests

That is why we write automatics tests

- to be run each time it is needed
- to tests all the cases (normal and edge)
- to easily test an app with multiples browsers

Tests

We also have different kind of tests depending on what you want to verify and control. The main cases are

- unitary: testing one isolated function
- functional: test a use case like an API endpoint or a render of a react component
- end-to-end (e2e): test the render of a full page across multiple pages

Tests

In this class about React we will try to see each cases

First of all you have to understand that tests are here to test and validate both

- the normal use case
- others edge case

So testing correctly a frontend can be long and tedious - some time even longer than writing the page

Tests

As a global estimation it is often considered that writing good tests take a long time

- at least a 30% increase of the time of the feature
- but you spend less time after to fix bugs
- it is a "pay now enjoy later" kind of deal

Tests

With React if you created the project using CRA you will have a basic setup and some default tests being implemented

- `npm test` will exist in the `package.json`
- a file `app.test.js` will be created
- multiples libraries will be installed

Tests

In a frontend project with React you will almost always see the the same libraries being used

- jest
- react-testing-library

These two are the most used testing solution with this ecosystem.

Tests

First off all let's see the default tests when creating a project

- one file `app.test.js`

Tests

```
import { render, screen } from '@testing-
library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn
react/i);
  expect(linkElement).toBeInTheDocument();
});
```

Tests

The first import allow the project to get both the functions `render` and `screen`. This will allow us to easily get the result of the build solution and what is displayed on the screen

On the next line we import the component we want to test, in that case the `App` component

Tests

After that we have our test function which is the function allowing us to test our component. If you want to have multiples tests for this component you only need to define multiple `test` function

The next three line are almost trivial. We render our React component, we read the screen (rendered) and try to match a Regex.

In the next step we `expect` that we have at least one element which matched the previous Regex. If that is true the test is valid if not it fails

Exercises

Add some tests in your React project and try to validate the basics components you have defined

Tests

We just saw how to write some functional tests, now we also want to write some unitary one

Tests

Let's take a default `add` function

```
function sum(a, b) {
  return a + b;
}
```

Tests

One basic test could be

```
import sum from './sum';

test('pure function returns the same output for the
same input', () => {
  expect(sum(2, 2)).toBe(4);
});
```

Tests

Of course we need to test and validate more case tha that

- what is we send letters
- if the total is above a BigInt
- Others issues

But at least you have the main idea behind testing a unitary function

Tests

For some more advanced testing solution with React, Jest allows us to take snapshot of the code

This snapshot will then be compared with the value / content of the generation component. If everything matches then the tests is good if not it fails

Tests

One main issue in a lot of project with this snapshot solution is that people does not look at the snapshot. And each time the full tests suite is running it will update the snapshot.

So people basically have a snapshot to compare their code to but always refresh it without validation !

Tests

Now try to handles all use cases on your projects to have a full testing solution

You will mostly need to check about `mocking` to be able to have a coverage of the event functions.

`mocking` is part of jest and can be easily used