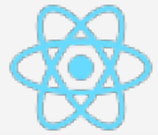


Tema 7

Debug & Testing

Debug & Testing

Testing consiste en verificar el comportamiento correcto de la aplicación. Se puede realizar en la interfaz de usuario, en el diseño de los algoritmos, en la implementación de componentes, etc.

Centrándonos en la implementación, esta puede ser evaluada utilizando pruebas de corrección y revisión de código. De esto último se encarga el ***Debugging***.

Debug & Testing

Debugging es el proceso de identificar y corregir errores en el código. Mediante esta técnica se puede realizar un seguimiento de la ejecución del programa, mostrando los valores de las variables y direcciones de memoria y ralentizando la salida de datos (modo de depuración).

De esta manera, *debugging* nos permite la ejecución controlada de un programa o código para seguir cada instrucción ejecutada y localizar fácilmente los errores.

Debug & Testing

En todos los lenguajes de programación suele haber más de una herramienta de *debug*. React no es una excepción.

En este tema veremos la herramienta **React Developer Tools** y cómo instalarla en nuestra máquina para poder facilitar la detección de errores.

Debug & Testing

Las principales diferencias entre **debugging** y **testing** son las siguientes:

Testing	Debugging
Busca y localiza un error en el código.	Arreglar el error encontrado en el código.
Se realiza mediante un equipo de testing.	Se realiza mediante un equipo de desarrollo.
El objetivo principal es encontrar tantos fallos como sean posibles.	El objetivo principal es eliminar todos esos fallos.

Debug & Testing

Una herramienta bastante completa es **React Developer Tools (RDT)**, ofrecida por Facebook. Esta herramienta es compatible tanto con Google Chrome como con Mozilla Firefox. Algunas de sus características son:

- Está desarrollado completamente con React.
- La instancia del componente seleccionado está disponible como $r desde la consola.
- Permite visualizar el código estructurado en árbol.
- Cuenta con total soporte para React Native.

Debug & Testing

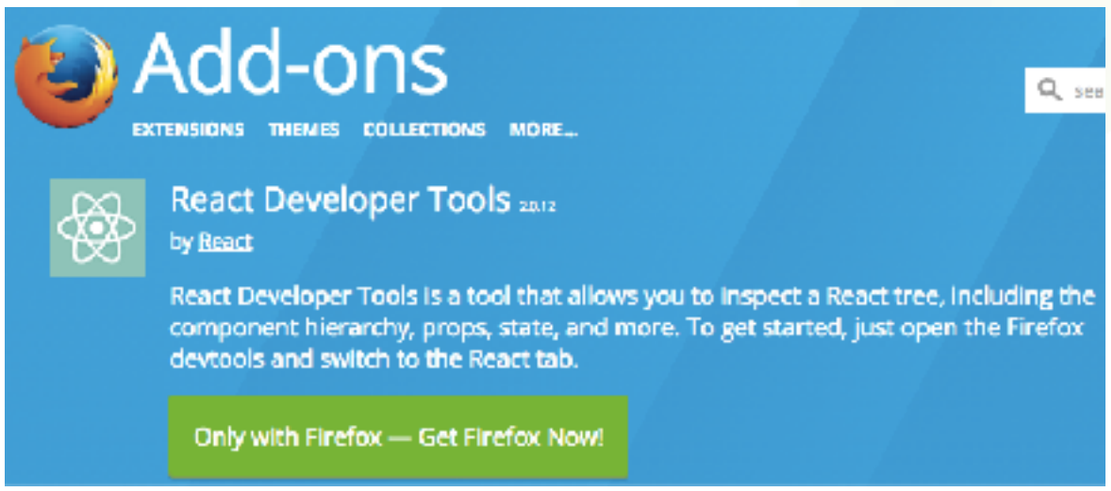
Esta herramienta te permite realizar cambios en el código mientras se está ejecutando en la página web. Se pueden hacer cambios en los valores de las variables en tiempo de ejecución.

RDT nos facilita una pestaña en el navegador en la cual se muestran los componentes React que han sido renderizados en la página. Asimismo, se puede seleccionar el componente deseado para obtener información y poder editarlo, así como todos los subcomponentes que tiene.

Debug & Testing

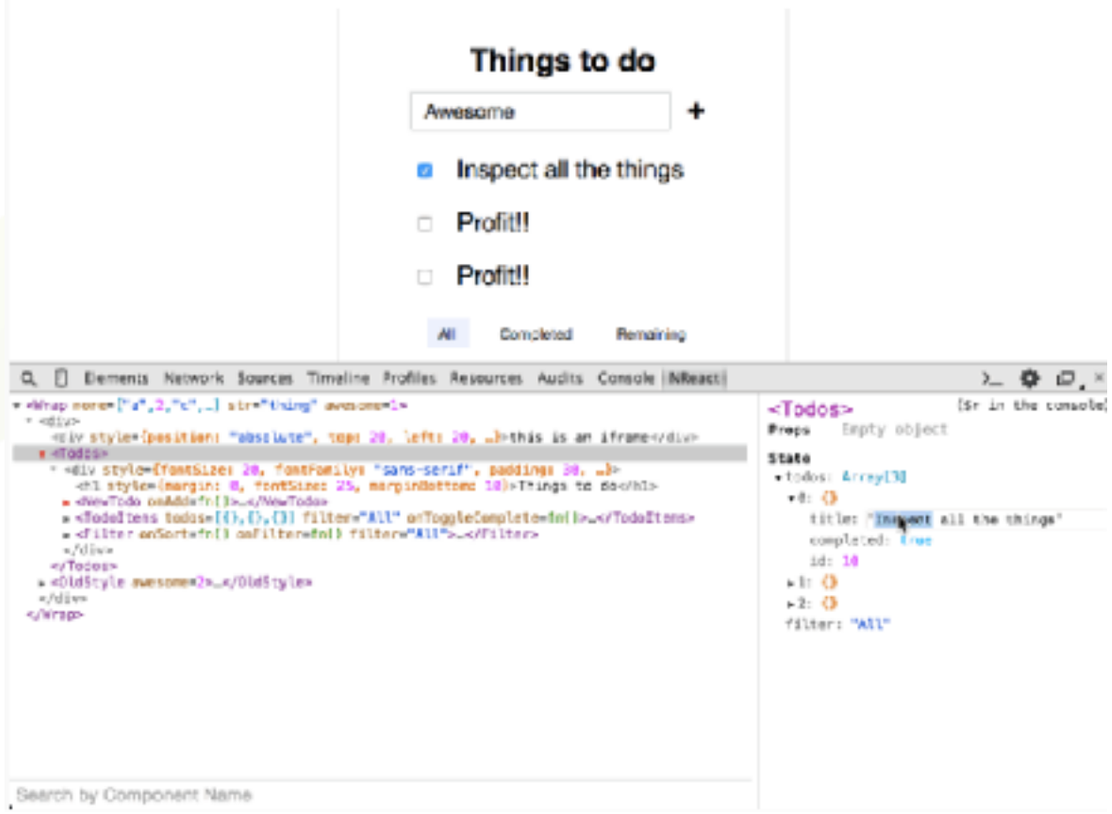
React Developer Tools está disponible en las tiendas de Firefox y Chrome. Para instalarlo simplemente deberemos añadirlo como extensión al navegador que deseemos.

- https://addons.mozilla.org/en-US/firefox/addon/react-devtools/
- https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi



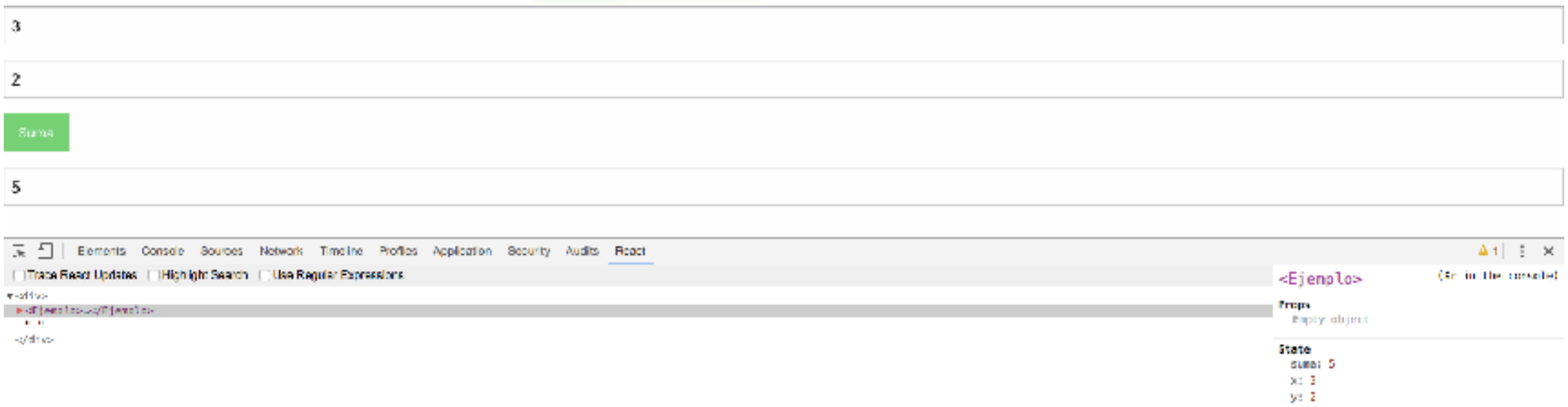
Debug & Testing

Una vez instalado, para utilizarlo en el navegador se hace click derecho sobre la página en cuestión y se pulsa en *Inspeccionar*. De esta manera se nos abrirá una pestaña en la parte inferior en la cual estará nuestra herramienta con el nombre React.



Debug & Testing

Si ejecutamos una aplicación React, veremos dicha pestaña de RDT.



Podemos cambiar los valores de las variables mediante el uso de esta herramienta. Simplemente hay que situarse en el elemento correspondiente y, en la parte derecha, hacer clic sobre la variable en cuestión y asignarle su nuevo valor.

Debug & Testing

Todos los elementos que no sean de lectura se podrán editar. Estos cambios se realizarán en tiempo de ejecución, lo cual resulta muy útil para comprobar la funcionalidad del programa.

En definitiva, esta herramienta resulta muy útil para detectar fallos y saber cómo corregirlos.

Jest

Jest es usado por Facebook para testear todos los archivos JavaScript incluyendo las aplicaciones React. Una de las filosofías de jest es la de proporcionar una integración que no requiera configuración. Con esta filosofía se consigue que se escriban muchas más pruebas de test y por tanto que se genere un código más estable y correcto.

Jest

Para instalar Jest tenemos que ejecutar el siguiente comando:

```
npm install -g jest
```

Un pequeño ejemplo podría verse creando un archivo sum.js en el que creamos un método que realice una suma de dos números que pasamos como parámetros:

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

module.exports = sum;
```

Jest

Podemos realizar distintas pruebas para comprobar el correcto funcionamiento del código, en nuestro ejemplo hemos utilizado expect y toBe para comprobar que el resultado es el esperado. En un archivo llamado sum.test.js añadimos la prueba usando test(nombreTest, ( ) =>…

```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Jest

Si queremos ejecutarlo podemos escribir el comando **jest** dentro de la carpeta del proyecto o modificar el archivo package.json para que al poner **npm test** se ejecute jest, esto último se realiza dentro del apartado scripts:

```
"scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "jest",
    "eject": "react-scripts eject"
  }
```

Jest

Independientemente de la forma en la que lo ejecutemos deberíamos ver en la consola algo como lo siguiente:

```
PASS src/sum.test.js
  ✓ adds 1 + 2 to equal 3 (3ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        9.573s, estimated 12s
```

También podemos ejecutar un único test o una carpeta donde se encuentren varios tests mediante el siguiente comando donde rutaArchivo es el lugar donde se encuentra la carpeta con los test o la ruta del test:

jest rutaArchivo

Jest

Al igual que en el ejemplo hemos usado expect y toBe para comprobar que el resultado de la suma es el esperado, también podemos usar expect junto con toEqual para realizar la misma función. En el siguiente ejemplo podemos ver como se comprueba que los elementos del array data coinciden con los insertados.

```
test('object assignment', () => {
  const data = {one: 1};
  data['two'] = 2;
  expect(data).toEqual({one: 1, two: 2});
});
```

Jest

Tenemos otros muchos Matchers que podemos usar para realizar pruebas:

- **.not:** comprueba la condición contraria de la que se indica. Si por ejemplo ponemos .not.toBeNull( ) comprueba que el resultado **no** tiene que ser Null.
- **.toBeNull:** comprueba que el resultado es Null.
- **.toBeUndefined:** comprueba que el resultado es un dato que no está definido.
- **.toBeFalsy:** comprueba que el resultado es False.

```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
  expect(n).not.toBeTruthy();
  expect(n).toBeFalsy();
});
```

Jest

Para testear números podemos hacer uso de los siguientes Matches:

- **.toBeGreaterThan:** si el valor comprobado es mayor que el indicado se pasa esta prueba.
- **.toBeGreaterThanOrEquals:** si el valor comprobado es mayor o igual que el indicado la prueba es correcta.
- **.toBeLessThan:** la prueba es correcta cuando el valor esperado es menor que el valor indicado.
- **.toBeLessThanOrEqual:** comprueba que el valor esperado es menor o igual al indicado.

```
test('two plus two', () => {
  const value = 2 + 2;
  expect(value).toBeGreaterThan(3);
  expect(value).toBeGreaterThanOrEqual(3.5);
  expect(value).toBeLessThan(5);
  expect(value).toBeLessThanOrEqual(4.5);
});
```

Jest

Para los Strings:

- **.toMatch:** comprueba que los dos strings son iguales.

Para los Arrays:

- **.toContain:** la prueba es correcta si el elemento indicado está dentro de la lista.

```
test('there is no I in team', () => {
  expect('team').not.toMatch(/I/);
});
```

```
const shoppingList = [
  'diapers',
  'kleenex',
  'trash bags',
  'paper towels',
  'beer',
];

test('the shopping list has beer on it', () => {
  expect(shoppingList).toContain('beer');
});
```

Jest

También podemos testear los errores:

- **.toThrow:** si llamamos a un método y lanza un error comprobamos que realmente se ha ejecutado el mismo.

```
function compileAndroidCode() {
  throw new ConfigError('you are using the wrong JDK');
}

test('compiling android goes as expected', () => {
  expect(compileAndroidCode).toThrow();
  expect(compileAndroidCode).toThrow(ConfigError);
});
```