FRONTEND TESTING - JEST

come realizzare **Unit test** in ambito javascript/typescript frontend

SCELTA DI JEST

- Ampiamente diffuso e documentato
- Standard di react
- Già presente di default

AGENDA

- Funzionamento di base
- Test di codice asincrono (callback)
- Tecniche di mock generali (funzioni mock, mock moduli, timer, ecc..)
- Test di componenti frontend (renderizzati)

BASE - UN PRIMO ESEMPIO / 1 - Sorgente

Funzione da testare

```
function somma(a: number, b: number): number {
    if (a === 1 && b === 1) {
        return 1.9999999987463487568
    } else {
        return a + b;
    }
}

export { somma }
```

BASE - UN PRIMO ESEMPIO / 2 - Test

Implementazione del test

```
import { sommma } from 'il/modulo';

describe("Test somma", () => {
    it("Esegue la somma", () => {
        expect(somma(1, 2)).toEqual(3);
    })
    // test <==> it (alias)
    test("Esegue 1 + 1", () => {
        expect(somma(1, 1)).toEqual(1.999999999874635);
    })
})
```

BASE - UN PRIMO ESEMPIO / 3 - Posizionamento test

`[filename].test.[js|ts|jsx|tsx]`

- Accanto al sorgente ()

```
mylib
    myfunctions.ts
    myfunctions.test.ts
```

- Cartella `__tests__`

```
src
    mylib/sub1/sub2
        myfunctions.ts
    __tests__/mylib/sub1/sub2
        myfunctions.test.ts
```

BASE - UN PRIMO ESEMPIO / 4 - Avvio test

```
# Tutti i test
npm test

# Uno specifico sorgente
npm test -- src/__tests__/lib/math.test.tsmath.test.tsx

# Con report di coverage
npm test -- src/__tests__ --coverage
```

BASE - UN PRIMO ESEMPIO / 5 - Esito test

```
 PASS  src/__tests__/lib/math.test.ts
  Test somma
    ✓ Esegue la somma (3 ms)
    ✓ Esegue 1 + 1

  ...

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        2.352 s
Ran all test suites matching /src\/__tests__\/lib\/math.test.ts/i.
```

BASE - UN PRIMO ESEMPIO / 6 - Esito test con errori

```
  ● Test somma › Esegue la somma

    expect(received).toEqual(expected) // deep equality

    Expected: 4
    Received: 3

      23 | describe("Test somma", () => {
      24 |     it("Esegue la somma", () => {
    > 25 |         expect(supersomma(1, 2)).toEqual(4);
         |                                  ^
      26 |     })
      27 |     it("Esegue 1 + 1", () => {
      28 |         expect(supersomma(1, 1)).toEqual(1.999999999874635);

      at Object.<anonymous> (src/__tests__/lib/math.test.ts:25:34)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 7 passed, 8 total
Snapshots:   0 total
Time:        3.308 s
```

BASE - UN PRIMO ESEMPIO / 7 - Avvio test in modalità CI

Jest rileva la variabile d'ambiente `CI`

Se impostata, non esegue in watch mode (il default)

```
export CI=true

npm test
```

TEST CALLBACK / 1 - Introduzione

Testare una funzione che riceve una callback come argomento che richiama in modo asincrono, volendo testare che la callback venga chiamata nel modo opportuno.

- La funzione callback riceve la callback `done()`
- Occorre richiamare done per indicare la fine del test

TEST CALLBACK / 2 - Codice

```
function total(samples: number[]) {
    return samples.reduce((acc, cur) => acc + cur, 0);
};

function totalizer(samples: number[], sumFunction: (samples: number[]) => number) {
    setTimeout(() => console.log(sumFunction(samples)), 0);
};
```

TEST CALLBACK / 3 - Test

```
describe("Test totalizer", () => {
    it("Richiama la somma", (done) => {
        totalizer([1, 2], (samples: number[]) => {
            try {
                expect(samples).toEqual([1, 2])
                return done();
            } catch (error) {
                return done(error)
            }
        })
    })
});
```

TEST DI PROMISE E CODICE ASINCRONO / 1 - Codice

Testare promise e codice asincrono in genere.

```
function willSum(samples: number[]) {
    return new Promise((resolve) => {
        resolve(total(samples));
    });
}
```

TEST DI PROMISE E CODICE ASINCRONO / 2 - Test

Si noti che i metodi di test sono *async*

```
// Attraverso l'attributo _resolves_
it("Richiama la somma con promise", async () => {
    await expect(willSum([1, 2])).resolves.toEqual(3);
})


// Via await
it("Richiama la somma async", async () => {
    const result = await willSum([1, 2]);
    expect(result).toEqual(3);
})
```

TEST DI PROMISE E CODICE ASINCRONO / 3 - Promise rejected

```
// Codice
function willSumSmallNumbers(samples: number[]) {
    return new Promise((resolve, reject) => {
        if (samples.some(item => item > 1000)) {
            reject("Some items are too big")
        }
        resolve(total(samples));
    });
}

// Test
describe("Test will sum small numbers", () => {
    it("Fallisce", async () => {
        await expect(willSumSmallNumbers([1, 2, 1001])).rejects.toEqual("Some items are too big");
    })
})
```

TECNICHE DI MOCK - MOCK DI FUNZIONI / 1

```
jest.fn()
jest.fn((...)=>...) // Con implementazione
```

```
function willSumViaCallback(samples: number[], sumFunction: (samples: number[]) => number) {
    return new Promise((resolve) => {
        resolve(sumFunction(samples));
    });
}
```

TECNICHE DI MOCK - MOCK DI FUNZIONI / 2 - Test

```
describe("Test totaliser mock", () => {
    it("La callback è richiamata correttamente", () => {
        const mockCallback = jest.fn()
        willSumViaCallback([1, 2], mockCallback);
        expect(mockCallback.mock.calls).toHaveLength(1);
        expect(mockCallback.mock.calls).toEqual([[[1, 2]]]);
    })
});
```

MOCK DI OGGETTI GLOBALI / 1

Es. window.location.href

Object.defineProperty

```
// La funzione usa internamente window.location.href

const Address = (props: AddressProps) => {

    const address: string = window.location.href;

    return <>
        <div>You are here: {address}</div>
    </>
}
```

MOCK DI OGGETTI GLOBALI / 2 - Test

```
describe('Test address', () => {
    let originalLocation: Location;

    beforeAll(() => {
        originalLocation = window.location;
    })

    afterAll(() => {
        Object.defineProperty(window, 'location', originalLocation)
    })

    it("Renders correctly", () => {

        Object.defineProperty(window, 'location', { value: { href: "https://my.web.address.xyz" } })

        const rendered = render(<Address />);

        const element = rendered.getByText("You are here: https://my.web.address.xyz");
        expect(element).toHaveTextContent("You are here")


    })
})
```

MOCK DI UN MODULO / 1 - Illustrazione

Modulo: data.ts

```
function fetchSomeData(...)
```

Modulo: application.ts

Usa il modulo data.ts

Vogliamo testare una funzione di application.ts moccando il risultato della funzione definita in data.ts

MOCK DI UN MODULO / 2 - Codice

```
// definizione: data.ts
function fetchOrder(orderNumber: number): Order {
    return DATABASE.filter(order => order.orderNumber === orderNumber)[0];
}
```

```
// definizione: application.ts
function totalOrderQty(orderNumber: number): number {
    const order = fetchOrder(orderNumber);
    return total(order.items.map(row => row.quantity))
}
```

MOCK DI UN MODULO / 3 - Test

```
// test
const mockOrder: Order = new Order(
    123, [new OrderItem("pere", 7), new OrderItem("banane", 4)]
    );

jest.mock('../../lib/data', () => {
    return {
        ...jest.requireActual('../../lib/data'),
        fetchOrder: (orderNumber: number) => mockOrder
    }
});

describe("Test totalOrderQty", () => {
    it("Esegue la somma delle quantità", () => {
        expect(totalOrderQty(1)).toEqual(11);
    })
});
```

MOCK DI TIMER / 1

```
// definizione
function remindMe(what: string, delay: number, action: (message: string) => void): void {
    const remindMessage = `REMINDER: ${what}`;

    setTimeout(() => { action(remindMessage) }, delay);
}

export { remindMe };
```

MOCK DI TIMER / 2.1 - Test setup

```
// test

import { remindMe } from '../../lib/reminder';

describe("Test reminder", () => {
    beforeEach(() => {
        jest.useFakeTimers();
    })

    afterEach(() => {
        jest.useRealTimers();
    })
```

MOCK DI TIMER / 2.2 - Test timer

```
it("Esempio di mock di Timeout", () => {

    jest.spyOn(global, 'setTimeout');

    remindMe("test", 1234, jest.fn());

    expect(setTimeout).toHaveBeenCalledTimes(1);
    expect(setTimeout).toHaveBeenLastCalledWith(expect.any(Function), 1234);
    expect(setTimeout).toHaveBeenLastCalledWith(expect.anything(), 1234);

})
```

MOCK DI TIMER / 2.3 - Avanzamento timer

```
    it("Esempio di simulazione di delay", () => {
        const callback = jest.fn();
        remindMe("test", 0, callback);
        expect(callback).not.toBeCalled();
        jest.runAllTimers();
        expect(callback).toBeCalled();
        expect(callback).toBeCalledTimes(1);
    })

});
```

LIBRERIA DI TEST REACT / Roadmap

Concetti chiave e tipico workflow di un test:

- Rendering di un componente
- Selezione di un componente renderizzato o parte di esso
- Eventuale trigger di eventi
- Asserzioni

TEST DI COMPONENTI FRONT END / Rendering

- Rendering di un componente o porzione di pagina

```
const rendered = render(<Selector title="Di Prova"></Selector>);
```

TEST DI COMPONENTI FRONT END / Selezione

- Selector: Selezionare una porzione

```
const mySelect = rendered.getByRole('combobox');
```

TEST DI COMPONENTI FRONT END / Trigger di eventi

```
fireEvent.change(mySelect, { target: { value: 'B' } });
```

TEST DI COMPONENTI FRONT END / Asserzioni

```
expect(mySelect).toHaveValue("B");
```

TEST DI COMPONENTI FRONT END / WaitFor - Motivazione

WaitFor è un'utility che permette di testare componenti front end che vedono multipli eventi e re-render al loro interno.

Il warning della libreria di testing che si ha in questi casi è

"When testing, code that causes React state updates should be wrapped into act(...)"

TEST DI COMPONENTI FRONT END / WaitFor - Utilizzo

```
test('Renders name holder', async () => {
    const userDescription = render(<HeaderComponent userDescription="example-usr-description" />)
    const nameHolder = userDescription.getByText(/example/)
    await waitFor(() => {
    expect(nameHolder).toContainHTML("<div class=\"user\">example-usr-description</div>");
    expect(nameHolder).toHaveTextContent("example-usr-description");
    expect(nameHolder).toHaveAttribute("class", "user");
    });
});
```

TEST FRONT END: MOCK USE STATE / 1

```
import { useState } from "react";

class SelectorProps {
    title: string | null = null;
}

const Selector = (props: SelectorProps) => {

    const [selected, setSelected] = useState("");
    const options: Array<number | string> = ["", "A", "B", "C"]
    return <>
        <div>{props.title}</div>
        <div>{selected ? selected : "..."}</div>
        <select onChange={(e) => setSelected(e.target.value)}>
            {options.map(opt => <option key={opt} value={opt} >Opzione {opt}</option>)}
        </select>
    </>
}


export { Selector };
```

TEST FRONT END: MOCK USE STATE / 3 - Test eventi

```
import { fireEvent, render } from "@testing-library/react";
import { Selector } from "../../../components/selector/selector";

describe("Test selector behaviour", () => {

    it("responds to change", () => {
        const rendered = render(<Selector title="Di Prova"></Selector>);

        const mySelect = rendered.getByRole('combobox');

        expect(mySelect).toHaveValue("")

        fireEvent.change(mySelect, { target: { value: 'B' } });

        expect(mySelect).toHaveValue("B");

        fireEvent.change(mySelect, { target: { value: 'C' } });

        expect(mySelect).toHaveValue("C");

    })

});
```

IDEE: COSA TESTARE

Idealmente "il più possibile", primo focus su:

- Componenti di base
- Funzionalità (es. wrapper di API)
- Tutto ciò che ha "Logica"

IDEE: PROSSIMI PASSI

- Utilizzo estensivo in un progetto
- Estensione ad altri framework

DOCUMENTAZIONE - GENERALE

Sito ufficiale Jest

https://jestjs.io/docs/getting-started

Convenzioni file test

https://create-react-app.dev/docs/running-tests/#filename-conventions

DOCUMENTAZIONE - TEST CALLBACK

Sezione documentazione di Jest

https://jestjs.io/docs/asynchronous

Cheat Sheet generico con focus sul test asincrono e mock:

https://www.codecademy.com/learn/learn-react-testing/modules/jest/cheatsheet

DOCUMENTAZIONE - MOCK

Mock di funzioni

https://jestjs.io/docs/mock-functions#using-a-mock-function

Mock di timer

https://jestjs.io/docs/timer-mocks#run-all-timers

Testing library: https://testing-library.com

Test react: https://jestjs.io/docs/tutorial-react

Aria Roles
https://www.w3.org/TR/html-aria/#docconformance

Reference sui tipi di asserzioni (jest-dom testing library): https://github.com/testing-library/jest-dom

Utilizzo di WaitFor: https://davidwcai.medium.com/react-testing-library-and-the-not-wrapped-in-act-errors-491a5629193b

Act (in trealtà non necessaria): https://it.legacy.reactjs.org/docs/test-utils.html#act

Mock fetch tutorial: https://medium.com/fernandodof/how-to-mock-fetch-calls-with-jest-a666ae1e7752
https://medium.com/swlh/how-to-mock-a-fetch-api-request-with-jest-and-typescript-bb6adf673a00

Mock Use State