RÉFÉRENCE RAPIDE JEST

Tests, matchers, mocking, async et snapshots

Configuration

Installation

```
npm install --save-dev jest
# package.json: "scripts": { "test": "jest" }
npx jest              # lancer tous les tests
npx jest --watch      # relancer à chaque modification
```

Nommage des fichiers

`*.test.js`	Fichiers de test (motif par défaut)
`*.spec.js`	Motif de test alternatif
`__tests__/`	Répertoire de tests (découverte automatique)

Lancer des tests spécifiques

```
npx jest path/to/file.test.js
npx jest --testNamePattern="adds"
npx jest --verbose     # sortie détaillée
```

Tests de base

Structure d'un test

```
describe("Calculator", () => {
  test("adds 1 + 2 to equal 3", () => {
    expect(add(1, 2)).toBe(3);
  });
});
```

test vs it

```
test("works correctly", () => { /* ... */ });
it("should work correctly", () => { /* ... */ });
// Les deux sont identiques ; "it" se lit comme de l'anglais
```

Ignorer et focaliser

`test.skip()`	Ignorer ce test
`test.only()`	Lancer uniquement ce test
`describe.skip()`	Ignorer toute la suite
`describe.only()`	Lancer uniquement cette suite

Matchers

Égalité

`.toBe(val)`	Égalité stricte (`===`)
`.toEqual(val)`	Égalité profonde (objets/tableaux)
`.toStrictEqual(val)`	Profond + type + propriétés undefined
`.not.toBe(val)`	Inverser tout matcher

Véracité

`.toBeTruthy()`	Valeur truthy
`.toBeFalsy()`	Valeur falsy
`.toBeNull()`	Exactement `null`
`.toBeUndefined()`	Exactement `undefined`
`.toBeDefined()`	Pas `undefined`

Nombres

`.toBeGreaterThan(n)`	Supérieur à n
`.toBeLessThanOrEqual(n)`	Inférieur ou égal
`.toBeCloseTo(0.3, 5)`	Comparaison de flottants (5 chiffres)

Chaînes, tableaux, objets

`.toMatch(/regex/)`	La chaîne correspond à la regex
`.toContain(item)`	Le tableau/itérable contient l'élément
`.toHaveLength(n)`	Longueur du tableau/chaîne
`.toHaveProperty(key, val)`	L'objet a la propriété
`.toMatchObject(obj)`	L'objet contient le sous-ensemble

Tests asynchrones

async / await

```
test("fetches data", async () => {
  const data = await fetchData();
  expect(data).toEqual({ id: 1 });
});
```

Promesses

```
test("resolves to data", () => {
  return expect(fetchData())
    .resolves.toEqual({ id: 1 });
});
```

Rejets

```
test("rejects with error", async () => {
  await expect(fetchBad())
    .rejects.toThrow("Not Found");
});
```

Exceptions

```
test("throws on invalid input", () => {
  expect(() => validate(null)).toThrow();
  expect(() => validate(null)).toThrow("invalid");
});
```

Mocking

Fonctions mock

```
const fn = jest.fn();
fn("hello");
expect(fn).toHaveBeenCalledWith("hello");
expect(fn).toHaveBeenCalledTimes(1);
```

Valeurs de retour mock

```
const fn = jest.fn()
  .mockReturnValue(42)
  .mockReturnValueOnce(99);
fn(); // 99 (premier appel)
fn(); // 42 (appels suivants)
```

Mocker des modules

```
jest.mock("./api");
const { fetchUser } = require("./api");
fetchUser.mockResolvedValue({ name: "Alice" });
```

Matchers mock

`.toHaveBeenCalled()`	Appelé au moins une fois
`.toHaveBeenCalledTimes(n)`	Appelé exactement n fois
`.toHaveBeenCalledWith(args)`	Appelé avec des arguments spécifiques
`.toHaveBeenLastCalledWith(args)`	Le dernier appel avait ces arguments

Espions

Espionner des méthodes

```
const spy = jest.spyOn(Math, "random")
  .mockReturnValue(0.5);
expect(Math.random()).toBe(0.5);
spy.mockRestore();  // restaurer l'original
```

Espionner des méthodes d'objets

```
const obj = { greet: (n) => `Hi ${n}` };
const spy = jest.spyOn(obj, "greet");
obj.greet("Alice");
expect(spy).toHaveBeenCalledWith("Alice");
```

Snapshots

Tests de snapshot

```
test("renders correctly", () => {
  const tree = renderer.create(<App />).toJSON();
  expect(tree).toMatchSnapshot();
});
```

Snapshots en ligne

```
test("formats name", () => {
  expect(formatName("alice"))
    .toMatchInlineSnapshot(`"Alice"`);
});
```

Mettre à jour les snapshots

```
npx jest --updateSnapshot   # mettre à jour tous
npx jest --updateSnapshot --testNamePattern="renders"
```

Setup et teardown

Hooks de cycle de vie

```
beforeAll(() => { /* une fois avant tous les tests */ });
afterAll(() => { /* une fois après tous les tests */ });
beforeEach(() => { /* avant chaque test */ });
afterEach(() => { /* après chaque test */ });
```

Portée

```
describe("Database", () => {
  beforeEach(() => db.connect());
  afterEach(() => db.disconnect());
  test("reads data", () => { /* ... */ });
});
```

Les hooks dans un describe ne s'appliquent qu'à ce bloc

Configuration

jest.config.js

```
module.exports = {
  testEnvironment: "node",
  coverageThreshold: {
    global: { branches: 80, lines: 80 }
  },
};
```

Options courantes

`testEnvironment`	`"node"` ou `"jsdom"` (DOM)
`roots`	Répertoires où chercher les tests
`collectCoverage`	Activer le rapport de couverture
`coverageDirectory`	Répertoire de sortie pour la couverture
`moduleNameMapper`	Alias de chemins (ex. préfixe `@/`)
`transform`	Transformations de fichiers (Babel, TS, etc.)
`setupFilesAfterFramework`	Lancer la configuration avant chaque suite

Couverture

```
npx jest --coverage
npx jest --collectCoverageFrom="src/**/*.js"
```

Modèles courants

Tester des appels API

```
jest.mock("./api");
test("loads users", async () => {
  api.getUsers.mockResolvedValue([{id: 1}]);
  const users = await loadUsers();
  expect(users).toHaveLength(1);
});
```

Mocks de timers

```
jest.useFakeTimers();
test("delays execution", () => {
  const cb = jest.fn();
  setTimeout(cb, 1000);
  jest.advanceTimersByTime(1000);
  expect(cb).toHaveBeenCalled();
});
```

Tests paramétrés

```
test.each([
  [1, 1, 2],
  [2, 3, 5],
])("add(%i, %i) = %i", (a, b, expected) => {
  expect(add(a, b)).toBe(expected);
});
```

Matchers personnalisés

```
expect.extend({
  toBeWithinRange(received, floor, ceil) {
    const pass = received >= floor
      && received <= ceil;
    return { pass, message: () =>
      `expected ${received} in [${floor},${ceil}]` };
}});
```