🎯 MỤC TIÊU CHÍNH CỦA TESTING

- ✅ **Test giống như người dùng thật sự sử dụng ứng dụng**
- ✅ **Tránh test implementation details** (nội dung bên trong component)
- ✅ **Test tốt là test ít thay đổi khi refactor nhưng vẫn đảm bảo behavior**
- ✅ Không cần 100% coverage, khoảng **70% là đủ** cho frontend
- ✅ Test giúp bạn tự tin **refactor mà không sợ hỏng** – nếu không làm được, test đó thất bại

---

🔍 1. Unit Tests – Kiểm thử đơn vị

💡 Là gì?

- Kiểm tra **từng phần nhỏ** như function, hook, hoặc component đơn lẻ.
- Chạy nhanh, viết đơn giản, không phụ thuộc nhiều phần khác.

📌 Khi nào dùng?

- Kiểm tra logic xử lý (validate, tính toán, convert...).
- Component hoặc hook **độc lập**.

🧠 Ví dụ:

ts
Sao chépChỉnh sửa

```
// utils/math.ts
export function sum(a: number, b: number) {
  return a + b;
}

// math.test.ts
import { sum } from './math';
test('sum works correctly', () => {
  expect(sum(2, 3)).toBe(5);
```

});

🔗 2. Integration Tests – Kiểm thử tích hợp

💡 Là gì?

- Test một luồng hoạt động bao gồm **nhiều phần hoạt động cùng nhau**.
- Có thể bao gồm component, API call, context, router...

📌 Khi nào dùng?

- Khi component/phần đó phức tạp: gọi API, thay đổi route, cập nhật context.

🧠 Ví dụ:

tsx
Sao chépChỉnh sửa

```
// LoginPage.test.tsx
render(<LoginPage />);
fireEvent.change(screen.getByLabelText('Email'), { target: { value: 'a@b.com' } });
fireEvent.click(screen.getByRole('button', { name: /submit/i }));
expect(await screen.findByText(/Welcome/i)).toBeInTheDocument();
```

🌐 3. E2E (End-to-End) Tests – Kiểm thử đầu cuối

💡 Là gì?

- Mô phỏng **hành vi người dùng thật** – click, nhập liệu, chuyển trang...
- Chạy bằng trình duyệt thật (hoặc headless trong CI/CD)

📌 Khi nào dùng?

- Test các flow lớn như: đăng ký, thanh toán, gửi form,...

✅ Dùng công cụ:

- ✅ Playwright
- ✅ Cypress

🧠 **Ví dụ với Playwright:**

ts

Sao chépChỉnh sửa

```
test('user can login', async ({ page }) => {
  await page.goto('/login');
  await page.fill('#email', 'test@example.com');
  await page.click('text=Submit');
  await expect(page).toHaveText('Welcome back!');
});
```

🧪 **MSW – Mock Service Worker**

💡 **Là gì?**

- **Giả lập API** thật bằng cách intercept fetch / XMLHttpRequest – hữu ích khi backend chưa xong.

📌 **Khi nào dùng?**

- Backend chưa sẵn.
- Cần test frontend **với request chạy như thật**.

🧠 **Ví dụ:**

ts

```
// handlers.ts
import { rest } from 'msw';
export const handlers = [
  rest.get('/api/user', (req, res, ctx) => {
    return res(ctx.json({ name: 'John Doe' }));
  }),
];

// setupTests.ts
import { setupServer } from 'msw/node';
import { handlers } from './handlers';
```

```
const server = setupServer(...handlers);
beforeAll(() => server.listen());
afterEach(() => server.resetHandlers());
afterAll(() => server.close());
```

🛠️ Công cụ khuyên dùng

Công cụ	Công dụng chính
Vitest	Khung test nhanh, thân thiện với Vite và React
Jest	Khung test truyền thống, ổn định, hỗ trợ mock tốt
Testing Library	Test giống người dùng thật (không kiểm tra state)
MSW	Mock API bằng service worker, không cần mock fetch bằng tay
Playwright / Cypress	E2E testing hiệu quả, hỗ trợ đa trình duyệt

---

🧭 Chiến lược tổng thể

Mục tiêu	Ưu tiên
Test logic nhỏ	✅ Unit test
Test component & nhiều logic kết hợp	✅ Integration test
Test toàn bộ hành vi người dùng	✅ E2E test
Test khi không có backend	✅ Dùng MSW
Viết test giống người dùng sử dụng app	✅ Testing Library

✅ Mẹo thực tế

- **Test quan trọng nhất là test behavior – không phải test chi tiết code**
- **Test càng ít gắn với implementation càng dễ bảo trì khi refactor**
- **Bạn có thể dùng TDD (test-driven development) cho logic tính toán**

- Không cần 100% coverage, chỉ cần test đủ để bạn **tự tin refactor**