✅ MỤC TIÊU CHÍNH CỦA TESTING

> Việc viết **unit test** là tốt, nhưng để thật sự **đảm bảo ứng dụng hoạt động đúng**, bạn nên ưu tiên **integration test** và **end-to-end (E2E) test**. Chúng kiểm tra hành vi thực tế gần với cách người dùng sử dụng ứng dụng.

🔍 1. Unit Tests – Kiểm thử đơn vị

💡 Là gì?

- Kiểm thử các đơn vị nhỏ nhất: **một hàm**, **một component**, một hook.
- Kiểm tra trong **cô lập** (isolation).
- Chạy rất nhanh và dễ viết.

📌 Khi nào dùng?

- Test hàm tính toán, xử lý logic.
- Component nhỏ, không phụ thuộc nhiều nơi khác.

🧠 Ví dụ:

ts

```
// 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?

- Kiểm tra **nhiều phần** trong hệ thống hoạt động cùng nhau (component + API + context + routing...).
- Giúp đảm bảo rằng **các phần giao tiếp với nhau đúng**.

📌 Khi nào dùng?

- Test một page/component tích hợp nhiều logic.
- Kiểm tra luồng đăng nhập, chuyển route, gọi API,...

🧠 Ví dụ:

tsx

```
// 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 thực tế của người dùng** từ đầu đến cuối.
- Tự động mở trình duyệt, click, nhập liệu, kiểm tra kết quả.

📌 Khi nào dùng?

- Đảm bảo **toàn bộ hệ thống frontend + backend** hoạt động như mong muốn.
- Kiểm tra các flow lớn như: đăng ký tài khoản, đặt hàng, thanh toán,...

✅ Dùng công cụ:

- **Playwright**

- **Cypress**
- Chạy trong trình duyệt thật (browser mode) hoặc headless (CI/CD).

🧠 **Ví dụ (Playwright):**

ts

```
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!');
});
```

🛠️ **Công cụ khuyên dùng**

Tên	Mô tả
Vitest	Khung test hiện đại, tương thích tốt với Vite/React. Tương tự Jest.
Testing Library	Test giống như người dùng thật (không kiểm tra state, chỉ kiểm tra output).
Playwright	Test E2E hiện đại, hỗ trợ đa trình duyệt, chế độ headless hoặc browser.
MSW (Mock Service Worker)	Giả lập API thực tế để test/fe dev mà không cần backend thật.

🧪 **MSW (Mock Service Worker) – Giả lập API để test và dev**

💡 **Là gì?**

- Tạo một **API giả lập chạy trong trình duyệt**, giúp frontend test hoặc làm việc mà **không cần backend**.

📌 **Khi nào dùng?**

- Backend chưa hoàn thiện.

- Bạn muốn test các luồng API như thật nhưng không có backend.
- Không cần "mock fetch" – request chạy như thật.

🧠 **Ví dụ:**

ts

```ts
// handlers.ts
import { rest } from 'msw';

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

ts

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

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

🧭 **Chiến lược tổng thể**

Mục tiêu	Ưu tiên
Test logic nhỏ	✅ Unit test
Test luồng component & logic	✅ Integration test
Test toàn bộ hành vi người dùng	✅ E2E test
Không có backend	✅ Dùng MSW để mock API
Viết test hiệu quả	✅ Dùng Testing Library