# Trang 5.7: Tự động hóa kiểm thử mã nguồn với /test.

## 1. Mục đích

`/test` là workflow để tự động hóa toàn bộ vòng đời kiểm thử — từ tạo test case, chạy bộ test hiện có, đến báo cáo độ bao phủ (coverage).

> **Triết lý:** Code không có test = code chưa hoàn chỉnh. `/test` giúp đội ngũ duy trì tiêu chuẩn kiểm thử mà không cần mất nhiều thời gian viết test thủ công.

---

## 2. Các chức năng của `/test`

| Cú pháp | Chức năng |
|---|---|
| `/test` | Chạy toàn bộ test suite hiện có |
| `/test [file hoặc thư mục]` | Tạo test cases cho file cụ thể |
| `/test coverage` | Báo cáo tỉ lệ bao phủ (coverage report) |
| `/test --watch` | Chế độ live — tự chạy lại khi code thay đổi |

---

## 3. Cách `/test` phân tích và tạo test

Khi nhận lệnh `/test [file]`, AI sẽ:

```
1. Đọc toàn bộ nội dung file được chỉ định
|
```

▼

2. Xác định tất cả functions và classes cần test

│

▼

3. Phân tích input/output, edge cases, và error paths

│

▼

4. Tạo test file tương ứng với bộ test đầy đủ:

- Happy path (trường hợp bình thường)
- Error cases (xử lý lỗi)
- Edge cases (giá trị biên)
- Null/undefined inputs

│

▼

5. Chạy test vừa tạo và báo cáo kết quả

---

# 4. Ví dụ phiên làm việc thực tế

## Ví dụ 1: Tạo test cho authentication service

Bạn: /test src/services/auth.ts

AI:

Đang phân tích auth.ts...

Tìm thấy 4 functions cần test:

- login(email, password): Promise<User>
- register(userData): Promise<User>
- verifyToken(token): boolean
- resetPassword(email): Promise<void>

Đang tạo test cases...

login():

- Trả về User khi credentials đúng
- Throw AuthError khi email không tồn tại
- Throw AuthError khi sai mật khẩu
- Throw ValidationError khi email không hợp lệ

register():

- Tạo User mới và trả về object
- Throw DuplicateError khi email đã tồn tại
- Hash password trước khi lưu (không lưu plaintext)
- Validate required fields

verifyToken():

- Trả về true với token hợp lệ
- Trả về false với token hết hạn
- Trả về false với token giả mạo
- Xử lý null/undefined input

resetPassword():

- Gửi email reset khi user tồn tại
- Không tiết lộ user không tồn tại (security)

Đã tạo: tests/auth.test.ts (12 test cases)

Đang chạy tests...

- 12/12 PASSED

Coverage: 95% (Statements), 91% (Branches)

# Ví dụ 2: Chạy toàn bộ test suite

Bạn: /test

AI:

Đang chạy toàn bộ test suite...

auth.test.ts — 12/12 passed (0.8s)

orders.test.ts — 8/8 passed (1.2s)

products.test.ts — 5/7 passed (FAILED)

FAIL: "calculateDiscount returns 0 for expired coupons"

Expected: 0

Received: 10

users.test.ts — 6/6 passed (0.5s)

Summary:

Tests: 31 passed, 2 failed

Coverage: 78%

2 tests failed. Gợi ý: chạy /debug để điều tra calculateDiscount.

# Ví dụ 3: Xem coverage report

Bạn: /test coverage

AI:

Coverage Report

File | Stmts | Branch | Funcs | Lines

----------|-------|--------|-------|------

src/services/auth.ts | 95% | 91% | 100% | 95%

src/services/orders.ts | 82% | 75% | 88% | 82%

src/utils/validators.ts | 60% | 45% | 70% | 60%

```
⚠ validators.ts có coverage thấp (60%).
Gợi ý: chạy /test src/utils/validators.ts để bổ sung test.
```

---

# 5. Các nguyên lý kiểm thử được áp dụng

## AAA Pattern (Arrange-Act-Assert)

Mọi test case đều theo cấu trúc:

```typescript
it('login returns User khi credentials đúng', async () => {
// Arrange — Chuẩn bị dữ liệu test
const mockUser = { email: 'test@example.com', password: 'hashed' };
mockDB.findUser.mockResolvedValue(mockUser);



// Act — Thực hiện hành động cần test
const result = await login('test@example.com', 'correct-password');



// Assert — Kiểm tra kết quả
expect(result).toEqual(expect.objectContaining({ email: 'test@example.com' }));
expect(result.password).toBeUndefined(); // Không trả về password
});
```

## Test hành vi, không test implementation

```typescript
// ❌ Test implementation (dễ vỡ khi refactor)
expect(service._privateMethod).toHaveBeenCalled();
```

```
// □ Test hành vi (bền vững)
expect(result.email).toBe('user@example.com');
```

# Kim tự tháp kiểm thử

```
      /\
     /E2E\ (ít nhất, chậm nhất, tốn kém nhất)
    /------\
   /Integration\ (vừa phải)
  /------------\
 / Unit Tests \ (nhiều nhất, nhanh nhất, rẻ nhất)
/______________\
```

---

# 6. Mẹo kiểm thử hiệu quả

## Luôn kiểm tra edge cases

Sau khi AI tạo test, yêu cầu thêm:

```
"Thêm test cases cho:
- Input rỗng (empty string, [], {})
- Giá trị giới hạn (0, -1, MAX_INT)
- Null và undefined
- Chuỗi unicode và ký tự đặc biệt"
```

## Tích hợp vào quy trình `/enhance`

```bash
# Sau mỗi lần thêm tính năng mới:
/test [file vừa tạo] # Tạo test cho code mới
/test # Chạy toàn bộ để kiểm tra regression
```

# Yêu cầu mock đúng cách

"/test src/services/payment.ts

Lưu ý: mock VNPay API, không gọi endpoint thực.

Và mock database, không dùng test database."

---

# 7. Frameworks được hỗ trợ

| Framework | File pattern | Ghi chú |
|---|---|---|
| **Jest** | `*.test.ts`, `*.spec.ts` | Phổ biến nhất với React/Node |
| **Vitest** | `*.test.ts`, `*.spec.ts` | Nhanh hơn Jest, dành cho Vite |
| **Playwright** | `e2e/*.spec.ts` | E2E testing, browser automation |
| **pytest** | `test_*.py`, `*_test.py` | Python projects |

AI tự phát hiện framework đang dùng từ package.json và tạo test theo đúng syntax.

---

Phiên bản #2
Được tạo 2026-03-04 06:12:30 UTC bởi Nam Đặng
Được cập nhật 2026-03-04 09:56:17 UTC bởi Nam Đặng