Trang 5.6: Protocol 4 bước sửa lỗi có hệ thống với /debug.

1. Vấn đề với cách debug thông thường

Cách debug phổ biến nhất: **thấy lỗi → thử sửa ngay → không được → thử sửa khác → vẫn không được → frustration**.

Đây là cách tiếp cận theo kiểu "đoán mò" (guess-driven debugging). Nó tốn thời gian và dễ gây thêm lỗi mới.

`/debug` giải quyết vấn đề này bằng cách bắt buộc AI tuân thủ một **giao thức điều tra 4 giai đoạn dựa trên bằng chứng**. Không được phép sửa code cho đến khi có bằng chứng xác định nguyên nhân gốc rễ.

---

2. Giao thức 4 giai đoạn

Giai đoạn 1: Discovery (Khám phá)

Mục tiêu: Thu thập toàn bộ sự kiện liên quan.

AI sẽ:

- Đọc error logs và stack trace (nếu bạn cung cấp).
- Xác định điều kiện tái hiện lỗi (repro steps).
- Đọc các file code liên quan đến khu vực bị lỗi.

- Thu thập thông tin môi trường (OS, version, config).

Giai đoạn 2: Hypothesis (Giả thuyết)

Mục tiêu: Liệt kê các nguyên nhân tiềm năng theo thứ tự xác suất.

AI tạo danh sách giả thuyết, ví dụ:

```
Giả thuyết 1 (Xác suất cao): emailRef.current là null khi submit
Giả thuyết 2 (Xác suất trung bình): Event handler bị bind sai context
Giả thuyết 3 (Xác suất thấp): Race condition trong async operation
```

Giai đoạn 3: Verification (Xác minh)

Mục tiêu: Kiểm tra từng giả thuyết bằng bằng chứng cụ thể, **không suy đoán**.

AI kiểm tra bằng:

- Đọc code liên quan đến giả thuyết.
- Thêm logging tạm thời để quan sát giá trị.
- Tham chiếu đến tài liệu chính thức hoặc lịch sử thay đổi (git log).

Giai đoạn 4: Resolution (Giải quyết)

Mục tiêu: Sửa đúng nguyên nhân gốc rễ + ngăn lỗi tái phát.

AI thực hiện:

1. Áp dụng bản sửa lỗi tối thiểu (minimal fix).
2. Giải thích tại sao cách sửa này đúng.
3. **Viết regression test** — test case để đảm bảo lỗi không quay lại.
4. Gợi ý cải thiện thêm nếu có.

---

3. Khi nào nên dùng `/debug`?

 Lý tưởng cho:

- "Nút submit không phản hồi gì, không có lỗi console."
- "API trả về 500 một cách ngẫu nhiên (intermittent)."

- "Ứng dụng crash chỉ trên một thiết bị/browser cụ thể."
- "Sau khi deploy, một tính năng vốn hoạt động tốt bỗng dưng hỏng."

 Không cần dùng cho:

- Lỗi typo rõ ràng, lỗi cú pháp đơn giản.
- Lỗi mà bạn đã biết nguyên nhân và chỉ cần sửa.

---

4. Ví dụ phiên làm việc thực tế

Lỗi: Form submit im lặng

□□ Bạn: /debug Form checkout của tôi bị lỗi. Khi click nút "Đặt hàng"
thì không có gì xảy ra cả, không có lỗi nào trong console.
Đây là stack trace khi tôi thêm try-catch:
TypeError: Cannot read properties of undefined (reading 'email')
at handleSubmit (CheckoutForm.tsx:47)

□□ AI (debugger):
=== Phase 1: Discovery ===

Đang đọc CheckoutForm.tsx...

Tìm thấy hàm handleSubmit tại dòng 45-62.
Quan sát: formData.email được truy cập tại dòng 47,
nhưng formData được khởi tạo là undefined.

=== Phase 2: Hypothesis ===

GIẢ THUYẾT 1 (Xác suất 85%):
formData chưa được khởi tạo đúng — state ban đầu là undefined thay vì {}

GIẢ THUYẾT 2 (Xác suất 10%):
useForm() hook chưa hoàn tất khởi tạo khi component mount

GIẢ THUYẾT 3 (Xác suất 5%):
Event không được preventDefault(), gây page reload trước khi chạy code

=== Phase 3: Verification ===

Kiểm tra state khởi tạo tại dòng 12:
const [formData, setFormData] = useState(); ← LỖI!

 Xác nhận Giả thuyết 1: useState() không có giá trị mặc định

=== Phase 4: Resolution ===

Sửa dòng 12:
Cũ: const [formData, setFormData] = useState();
Mới: const [formData, setFormData] = useState({ email: '', name: '', address: '' });

 Đã áp dụng bản sửa lỗi
 Đã thêm regression test: "Form không submit khi formData undefined"

 Gợi ý thêm: Cân nhắc dùng TypeScript interface để
compiler phát hiện lỗi này sớm hơn.

Lỗi: API 500 ngẫu nhiên

 Bạn: /debug API /api/orders trả về 500 khoảng 1/5 lần gọi.
Không tái hiện được một cách nhất quán.
Server log:

Error: Connection terminated unexpectedly

at Connection.parseE (pg.js:234)

AI (debugger):

=== Phase 1: Discovery ===

Kiểm tra database connection pooling...

Tìm thấy: Pool size = 1, query timeout = 30s

=== Phase 2: Hypothesis ===

GIẢ THUYẾT 1: Connection pool cạn kiệt dưới tải cao

GIẢ THUYẾT 2: Database connection timeout không được xử lý

GIẢ THUYẾT 3: Race condition giữa các concurrent requests

=== Phase 3: Verification ===

Đọc db.ts: Pool maxConnections = 1 ← BẤT THƯỜNG

Error message "terminated unexpectedly" = timeout issue (không phải query lỗi)

Xác nhận Giả thuyết 1: Pool size quá nhỏ cho concurrent requests

=== Phase 4: Resolution ===

[Gợi ý cấu hình pool size phù hợp + retry logic...]

---

5. Cách cung cấp thông tin để debug nhanh hơn

Cung cấp đầy đủ context ngay từ đầu

/debug [Mô tả lỗi]

Thông tin thêm:

- Stack trace: [paste toàn bộ error message]
- Bước tái hiện: [1. Làm A, 2. Làm B, 3. Lỗi xuất hiện]
- Môi trường: [Node 20, Next.js 14, macOS]
- Khi nào bắt đầu xảy ra: [Sau khi update thư viện X / deploy hôm qua]

Yêu cầu xác nhận sau khi sửa

"Sau khi sửa xong, hãy tạo test case để xác nhận lỗi không tái phát,
và chạy lại test suite để đảm bảo không có regression."

---

6. Điều Agent `debugger` tuyệt đối không làm

❌ Không	✅ Thay vào đó
Đoán nguyên nhân và sửa ngay	Thu thập bằng chứng trước
Thay đổi nhiều thứ cùng lúc	Thay đổi từng thứ một và verify
"Thử cái này xem sao"	Giải thích tại sao cách sửa này đúng
Bỏ qua viết regression test	Luôn thêm test sau khi sửa

Phiên bản #2
Được tạo 2026-03-04 06:12:21 UTC bởi Nam Đặng
Được cập nhật 2026-03-04 09:51:53 UTC bởi Nam Đặng