CLAUDE CODE

15 Prompt *Vibe Coding*

prompt chính xác cho từng giai đoạn của dự án

01 Viết một bản PRD đầy đủ

```
Viết một bản PRD hoàn chỉnh cho tính năng dưới đây.

Tính năng: [MÔ TẢ TÍNH NĂNG]
Người dùng: [AI SẼ DÙNG]
Stack: [STACH CỦA BẠN]

Bao gồm:

+ Vấn đề cần giải quyết + success metrics
+ User story kèm acceptance criteria
+ Scope: cái gì làm v1, cái gì thì không
+ Các thay đổi về data model
+ Edge case + trạng thái lỗi
+ Câu hỏi mở cần tôi trả lời

Giữ dưới 2 trang. Cụ thể, không lan man.
Lưu thành docs/prd-[feature].md để mọi prompt sau đều tham chiếu được.
```

02 Tạo file CLAUDE.md của bạn

```
Quét toàn bộ codebase này, rồi tạo một file CLAUDE.md.

Bao gồm:

+ Dự án này là gì, giải trong 2 dòng
+ Tech stack + những version quan trọng
+ Commands: [DEV / BUILD / TEST / LINT]
+ Kiến trúc: thư mục nằm ở đâu và vì sao
+ Code convention thực sự đọc được trong code
+ Luật cứng: [ĐIỀU KHÔNG ĐƯỢC PHÁ] - thứ không bao giờ được dụng nếu chưa hỏi
+ Các bẫy mà một engineer mới sẽ dính ngay tuần đầu

Viết luật dưới dạng câu mệnh lệnh ngắn. Không chung chung - chỉ những gì đúng với repo NÀY. Nếu không chắc về một luật, hãy hỏi tôi thay vì tự bịa ra.
```

03 Chế độ Ultra Plan

```
Vào plan mode. KHÔNG viết code vội.

Nhiệm vụ: [ĐẠI NHIỆM VỤ]
Ràng buộc: [DEADLINE / STACH / VÙNG CẤM]

1. Đọc mọi file mà nhiệm vụ này chạm tới, liệt kê vào 1 dòng mỗi file đang làm gì
2. Vẽ ra hành vi hiện tại so với hành vi mong muốn
3. Đề xuất 2-3 hướng làm kèm trade-off thật: độ phức tạp, rủi ro, phạm vi ảnh hưởng
4. Chọn một hướng và giải thích trong 3 dòng
5. Chia nhỏ thành các bước đủ nhỏ để kiểm chứng từng bước một, mỗi bước có cách check riêng
6. Liệt kê rủi ro + cách rollback chính xác cho từng cái
7. Đánh dấu mọi thứ động tới [AUTH / PAYMENTS / PROD DATA] để tôi duyệt riêng

Rồi dừng lại. Cho tôi xem kế hoạch và đợi tôi duyệt trước khi đụng vào bất kỳ file nào.
```

04 Phát triển theo Spec

```
Chúng ta viết spec trước. Viết spec cho: [TÍNH NĂNG]

Bối cảnh: [AI DÙNG + VÌ SAO LÀ LÚC NÀY]

Định dạng spec:

+ Hành vi: given / when / then, mọi trường hợp - happy path, edge case, trạng thái lỗi
+ API contract: input, output, hình dạng lỗi, status code
+ Dữ liệu: thay đổi schema + migration cần thiết
+ Trạng thái UI: loading, empty, error, success
+ Non-goal: những gì spec này cố tình bỏ qua
+ Acceptance checklist tôi soát được từng dòng

Sau khi tôi duyệt spec, hãy làm ĐÚNG như spec. Nếu thực tế bước phải lệch, hãy dừng, cập nhật spec, xin tôi đồng ý, rồi mới đi tiếp. Spec mới là nguồn chân lý, không phải code.
```

05 Design Brief UI & UX đầy đủ

```
Tạo một design brief UI/UX đầy đủ cho: [MÀN HÌNH HOẶC FLOW]

Đối tượng: [NGƯỜI DÙNG]
Thương hiệu: [MÀU / FONT / CẢM GIÁC]

Bàn giao:

+ Hành trình người dùng đi qua flow, từng bước một
+ Layout từng màn: phân cấp, spacing, breakpoint
+ Danh sách component kèm mọi trạng thái - hover, empty, error, loading
+ Typography + color token
+ Button: cái gì animate, thời lượng, easing
+ Ghi chú accessibility

Học pattern từ [3:5 SẢN PHẨM BẠN THÍCH] để lấy hướng đi. Đừng bao giờ copy chúng.
```

06 Kế hoạch triển khai

```
Tạo kế hoạch triển khai cho: [SPEC / PRD ĐÃ DUYỆT]

Quy tắc:

+ Bắp thứ tự các bước sao cho app vẫn build và chạy được sau MỖI bước
+ Mỗi bước: file nào bị chạm, thay đổi những gì, tôi kiểm chứng bằng cách nào
+ Đánh dấu bước nào cần migration hoặc dependency mới
+ Đặt phần rủi ro và ẩn nhiều nhất lên trước
+ Ước lượng mỗi bước: [S / M / L]

Xuất ra một checklist build được đánh số để tôi chạy từng bước một. Đợi tôi nói "go" giữa các bước.
```

07 Kết nối một MCP Server

```
Dựng một MCP server cho: [DỊCH VỤ / API]

Tôi cần nó làm được: [CÔNG VIỆC CẦN LÀM]

1. Kiểm tra xem đã có server chính thức hoặc được duy trì tốt chưa - nếu rồi nguồn
2. Nếu có: cấu hình cài chính xác + config .mcp.json, scope trong project này
3. Nếu chưa: dựng một cái bằng MCP SDK - tools, auth, xử lý lỗi, response có kiểu
4. Chỉ thêm những tool tôi thực sự dùng: [DANH SÁCH]
5. Truyền secret qua [ENV VARS], không bao giờ hardcode key trong file config
6. Kiểm tra kết nối và gọi thử một tool đầu-cuối, cho tôi xem output
7. Ghi chú mỗi tool trong 1 dòng để các session sau biết khi nào cần dùng tới
```

08 Kết nối Database của bạn

```
Kết nối app này với [POSTGRES / SUPABASE / DB CỦA BẠN].

+ Chọn client hợp với stack này, giải thích trong 1 dòng
+ Env var: đặt tên, thêm vào .env.example, không bao giờ commit giá trị thật
+ Schema: bảng cho [CÁC THỰC THỂ] kèm kiểu dữ liệu, quan hệ, và index chu [TRUY VẤN NÓNG]
+ Tạo + chạy migration, và cho tôi xem cách rollback từng cái
+ Mỗi bảng một query helper có kiểu - không rải raw SQL khắp component
+ Access rule / row-level security nếu đây là [MULTI-TENANT]
+ Connection pooling nếu chúng ta chạy serverless

Rồi chứng minh: seed một dòng, đọc lại, cho tôi xem output.
```

09 Tìm lỗ hổng bảo mật

```
Audit codebase này để tìm lỗ hổng bảo mật.
Soi nó như thể bạn đang tìm đường chui vào.

Vùng tập trung: [AUTH / PAYMENTS / DỮ LIỆU NGƯỜI DÙNG]

Kiểm tra:

+ Secret nằm trong code, config hay git history
+ Injection: SQL, XSS, command, path traversal
+ Auth: route thiếu kiểm tra, session yếu, redirect hỏng
+ IDOR: user A có đọc được dữ liệu của user B không?
+ File upload + validate input ở mọi form
+ CVE của dependency - chạy audit và đọc kỹ
+ Rate limit trên [ENDPOINT TỐN TÀI NGUYÊN]
+ Thông tin rò rỉ qua error message và log

Xếp hạng phát hiện theo mức nghiêm trọng kèm file:line chính xác, sửa ngay các lỗi critical, phần còn lại liệt kê thành ticket kèm ước lượng công sức.
```

10 Debug lỗi thật nhanh

```
Debug lỗi này. KHÔNG đoán mò.

Lỗi: [DÁN TOÀN BỘ LỖI + STACK TRACE]
Khi nào xảy ra: [CÁC BƯỚC TÁI HIỆN]

1. Đọc stack trace, mở đúng những file liên quan
2. Nêu hành vi mong đợi so với thực tế trong 1 dòng
3. Liệt kê 3 giả thuyết, xếp theo khả năng xảy ra
4. Chứng minh hoặc loại từng cái bằng log hay một test nhỏ - bằng chứng, không cảm tính
5. Sửa nguyên nhân gốc, không vá triệu chứng
6. Tìm cùng pattern đó trong repo - nếu hỏng được ở đây thì cũng hỏng ở chỗ khác
7. Thêm regression test mà nếu bỏ bản fix đi thì test phải fail
8. Nói cho tôi trong 2 dòng vì sao nó hỏng và vì sao nó không bao giờ hỏng kiểu này nữa
```

11 Viết E2E Test cho ứng dụng

```
Viết Playwright E2E test cho: [FLOW]

Stack: [STACH CỦA BẠN] CI: [GITHUB ACTIONS / KHÁC]

+ Ưu tiên luồng ra tiền trước: [SIGNUP / CHECKOUT / ANH HÙNG CỐT LÕI]
+ Test cái người dùng nhìn thấy, không test chi tiết cài đặt
+ Selector: dùng role và label, đừng bao giờ dùng chuỗi CSS dễ vỡ
+ Mỗi flow một unhappy path: input sai, hết mạng, session hết hạn
+ Test phải độc lập - chạy thứ tự nào cũng được, không dùng chung state, mỗi test tự seed dữ liệu
+ Headless trên CI, headed ở máy local để debug
+ Chụp screenshot + trace khi fail

Chạy cả bộ test, cho tôi xem kết quả, sửa những gì fail, và nói rõ bộ test vẫn CHƯA phủ được gì.
```

12 Dọn sạch Dead Code

```
Tìm và xóa dead code trong repo này.

Phạm vi: [TOÀN BỘ REPO / THƯ MỤC CỤ THỂ]

+ Export, component, hook, util không ai dùng
+ Nhánh không bao giờ chạy tới + block code bị comment
+ Dependency trong package.json không ai import
+ Feature flag đã bị xét luôn bật hoặc luôn tắt
+ Logic trùng lặp đáng lẽ nên gộp làm một
+ Class CSS chết và asset không dùng

Tìm kiếm xác minh trước MỖI lần xóa - dynamic import và tham chiếu dạng chuỗi cũng tính. Xóa theo từng commit nhỏ, chạy [BUILD + TESTS] sau mỗi commit, rồi báo tổng số dòng đã xóa cùng những chỗ bạn chưa chắc 100%.
```

13 Viết Git Commit gọn gàng

```
Commit các thay đổi đã stage một cách tử tế.

Quy ước: [CONVENTIONAL COMMITS / ĐỊNH DẠNG CỦA BẠN]

+ Tách các thay đổi không liên quan thành commit riêng
+ Định dạng: type(scope): đã đổi gì và vì sao
  feat / fix / refactor / chore / docs / test
+ Subject dưới 50 ký tự, dùng câu mệnh lệnh
+ Phần body giải thích VÌ SAO, wrap ở cột 72
+ Tham chiếu ticket: [TICKET-ID]
+ Đừng bao giờ trộn refactor với thay đổi hành vi trong cùng một commit
+ Đừng bao giờ commit [SECRETS / .ENV / FILE SINH TỰ ĐỘNG]

Cho tôi xem kế hoạch - file nào vào commit nào + nội dung message - trước khi commit bất cứ thứ gì. Rồi commit từng cái một để tôi có thể chặn giữa chừng.
```

14 Dùng Hooks làm Guardrails

```
Thiết lập Claude Code hooks làm guardrail ở đây.

Stack: [STACH + PACKAGE MANAGER CỦA BẠN]

+ PostToolUse: chạy [LINT + TYPECHECK] sau mỗi lần sửa file, đẩy lỗi ngược lại ngay
+ PreToolUse: chặn sửa [ĐƯỜNG DẪN ĐƯỢC BẢO VỆ: MIGRATIONS, .ENV, PROD CONFIG]
+ Stop: chạy [TEST SUITE] trước khi kết thúc session
+ Notification: báo tôi bằng [ÂM THANH / SLACK] khi cần tôi vào cuộc

Viết các script hook + phần khai báo trong settings.json. Giữ mỗi script dưới 20 dòng và exit khác 0 kèm thông báo rõ ràng để agent biết chính xác phải sửa gì. Mỗi kịch hoạt từng hook có chủ đích và cho tôi xem chúng chạy.
```

15 Biến một tác vụ thành Skill

```
Biến tác vụ lặp đi lặp lại này thành một Claude Code skill.

Tác vụ tôi làm hoài: [MÔ TẢ TÁC VỤ + CÁC BƯỚC]

+ Tạo .claude/skills/[TNAME]/SKILL.md
+ Frontmatter: name + description kèm đúng những cấu tiết hay nói để kích hoạt nó
+ Body: quy trình đánh số, các quy ước của tôi, edge case
+ Cái gì nên hỏi tôi, cái gì tự suy ra được
+ "Xong" thì trông như thế nào

Rồi chạy thử skill trên một ví dụ thật và tinh chỉnh cho tới khi output khớp với cách tôi vẫn làm tay.
```

01

Viết một bản PRD đầy đủ

Viết một bản PRD hoàn chỉnh cho tính năng dưới đây.

Tính năng: [MÔ TẢ TÍNH NĂNG]
Người dùng: [AI SẼ DÙNG]
Stack: [STACK CỦA BẠN]

Bao gồm:

* Vấn đề cần giải quyết + success metrics
* User story kèm acceptance criteria
* Scope: cái gì lên v1, cái gì thì không
* Các thay đổi về data model
* Edge case + trạng thái lỗi
* Câu hỏi mở cần tôi trả lời

Giữ dưới 2 trang. Cụ thể, không lan man.
Lưu thành docs/prd-[feature].md để mọi prompt sau đều tham chiếu được.

02

Tạo file CLAUDE.md của bạn

```
Quét toàn bộ codebase này, rồi tạo một file CLAUDE.md.

Bao gồm:

* Dự án này là gì, gói trong 2 dòng
* Tech stack + những version quan trọng
* Commands: [DEV / BUILD / TEST / LINT]
* Kiến trúc: thứ gì nằm ở đâu và vì sao
* Code convention thực sự đọc được trong code
* Luật cứng: [ĐIỀU KHÔNG ĐƯỢC PHÁ] - thứ không bao giờ được đụng nếu
  chưa hỏi
* Các bẫy mà một engineer mới sẽ dính ngay tuần đầu

Viết luật dưới dạng câu mệnh lệnh ngắn. Không chung chung - chỉ những
gì đúng với repo NÀY. Nếu không chắc về một luật, hãy hỏi tôi thay vì
tự bịa ra.
```

03

Chế độ Ultra Plan

```
Vào plan mode. KHÔNG viết code vội.

Nhiệm vụ: [DÁN NHIỆM VỤ]
Ràng buộc: [DEADLINE / STACK / VÙNG CẤM]

1. Đọc mọi file mà nhiệm vụ này chạm tới, liệt kê kèm 1 dòng mô tả
   chúng đang làm gì
2. Vẽ ra hành vi hiện tại so với hành vi mong muốn
3. Đề xuất 2-3 hướng làm kèm tradeoff thật: độ phức tạp, rủi ro, phạm
   vi ảnh hưởng
4. Chọn một hướng và giải thích trong 3 dòng
5. Chia nhỏ thành các bước đủ nhỏ để kiểm chứng từng bước một, mỗi bước
   có cách check riêng
6. Liệt kê rủi ro + cách rollback chính xác cho từng cái
7. Đánh dấu mọi thứ động tới [AUTH / PAYMENTS / PROD DATA] để tôi duyệt
   riêng

Rồi dừng lại. Cho tôi xem kế hoạch và đợi tôi duyệt trước khi đụng vào
bất kỳ file nào.
```

04

Phát triển theo Spec

```
Chúng ta viết spec trước. Viết spec cho: [TÍNH NĂNG]

Bối cảnh: [AI DÙNG + VÌ SAO LÀ LÚC NÀY]

Định dạng spec:

* Hành vi: given / when / then, mọi trường hợp - happy path, edge case,
  trạng thái lỗi
* API contract: input, output, hình dạng lỗi, status code
* Dữ liệu: thay đổi schema + migration cần thiết
* Trạng thái UI: loading, empty, error, success
* Non-goal: những gì spec này cố tình bỏ qua
* Acceptance checklist tôi soát được từng dòng

Sau khi tôi duyệt spec, hãy làm ĐÚNG như spec. Nếu thực tế buộc phải
lệch, hãy dừng, cập nhật spec, xin tôi đồng ý, rồi mới đi tiếp. Spec
mới là nguồn chân lý, không phải code.
```

05

Design Brief UI & UX đầy đủ

Tạo một design brief UI/UX đầy đủ cho: [MÀN HÌNH HOẶC FLOW]

Đối tượng: [NGƯỜI DÙNG]
Thương hiệu: [MÀU / FONT / CẢM GIÁC]

Bàn giao:

* Hành trình người dùng đi qua flow, từng bước một
* Layout từng màn: phân cấp, spacing, breakpoint
* Danh sách component kèm mọi trạng thái - hover, empty, error, loading
* Typography + color token
* Motion: cái gì animate, thời lượng, easing
* Ghi chú accessibility

Học pattern từ [2-3 SẢN PHẨM BẠN THÍCH] để lấy hướng đi. Đừng bao giờ copy chúng.

06

Kế hoạch triển khai

Tạo kế hoạch triển khai cho: [SPEC / PRD ĐÃ DUYỆT]

Quy tắc:

* Sắp thứ tự các bước sao cho app vẫn build và chạy được sau MỖI bước
* Mỗi bước: file nào bị chạm, thay đổi những gì, tôi kiểm chứng bằng cách nào
* Đánh dấu bước nào cần migration hoặc dependency mới
* Đặt phần rủi ro và mơ hồ nhất lên trước
* Ước lượng mỗi bước: [S / M / L]

Xuất ra một chuỗi build được đánh số để tôi chạy từng bước một. Đợi tôi nói "go" giữa các bước.

07

Kết nối một MCP Server

```
Dựng một MCP server cho: [DỊCH VỤ / API]

Tôi cần nó làm được: [CÔNG VIỆC CẦN LÀM]

1. Kiểm tra xem đã có server chính thức hoặc được duy trì tốt chưa -
   nêu rõ nguồn
2. Nếu có: câu lệnh cài chính xác + config .mcp.json, scope trong
   project này
3. Nếu chưa: dựng một cái bằng MCP SDK - tools, auth, xử lý lỗi,
   response có kiểu
4. CHỈ thêm những tool tôi thực sự dùng: [DANH SÁCH]
5. Truyền secret qua [ENV VARS], không bao giờ hardcode key trong file
   config
6. Kiểm tra kết nối và gọi thử một tool đầu-cuối, cho tôi xem output
7. Ghi chú mỗi tool trong 1 dòng để các session sau biết khi nào cần
   dùng tới
```

08

Kết nối Database của bạn

Kết nối app này với [POSTGRES / SUPABASE / DB CỦA BẠN].

* Chọn client hợp với stack này, giải thích trong 1 dòng
* Env var: đặt tên, thêm vào .env.example, không bao giờ commit giá trị thật
* Schema: bảng cho [CÁC THỰC THỂ] kèm kiểu dữ liệu, quan hệ, và index cho [TRUY VẤN NÓNG]
* Tạo + chạy migration, và cho tôi xem cách rollback từng cái
* Mỗi bảng một query helper có kiểu - không rải raw SQL khắp component
* Access rule / row-level security nếu đây là [MULTI-TENANT]
* Connection pooling nếu chúng ta chạy serverless

Rồi chứng minh: seed một dòng, đọc lại, cho tôi xem output.

09

Tìm lỗ hổng bảo mật

Audit codebase này để tìm lỗ hổng bảo mật.
Soi nó như thể bạn đang tìm đường chui vào.

Vùng tập trung: [AUTH / PAYMENTS / DỮ LIỆU NGƯỜI DÙNG]

Kiểm tra:

* Secret nằm trong code, config hay git history
* Injection: SQL, XSS, command, path traversal
* Auth: route thiếu kiểm tra, session yếu, redirect hỏng
* IDOR: user A có đọc được dữ liệu của user B không?
* File upload + validate input ở mọi form
* CVE của dependency - chạy audit và đọc kỹ
* Rate limit trên [ENDPOINT TỐN TÀI NGUYÊN]
* Thông tin rò rỉ qua error message và log

Xếp hạng phát hiện theo mức nghiêm trọng kèm file:line chính xác, sửa ngay các lỗi critical, phần còn lại liệt kê thành ticket kèm ước lượng công sức.

10

Debug lỗi thật nhanh

```
Debug lỗi này. KHÔNG đoán mò.

Lỗi: [DÁN TOÀN BỘ LỖI + STACK TRACE]
Khi nào xảy ra: [CÁC BƯỚC TÁI HIỆN]

1. Đọc stack trace, mở đúng những file liên quan
2. Nêu hành vi mong đợi so với thực tế trong 1 dòng
3. Liệt kê 3 giả thuyết, xếp theo khả năng xảy ra
4. Chứng minh hoặc loại từng cái bằng log hay một test nhỏ - bằng
   chứng, không cảm tính
5. Sửa nguyên nhân gốc, không sửa triệu chứng
6. Tìm cùng pattern đó trong repo - nếu hỏng được ở đây thì cũng hỏng ở
   chỗ khác
7. Thêm regression test mà nếu bỏ bản fix đi thì test phải fail
8. Nói cho tôi trong 2 dòng vì sao nó hỏng và vì sao nó không bao giờ
   hỏng kiểu này nữa
```

11

Viết E2E Test cho ứng dụng

```
Viết Playwright E2E test cho: [FLOW]

Stack: [STACK CỦA BẠN] CI: [GITHUB ACTIONS / KHÁC]

* Ưu tiên luồng ra tiền trước: [SIGNUP / CHECKOUT / HÀNH ĐỘNG CỐT LÕI]
* Test cái người dùng nhìn thấy, không test chi tiết cài đặt
* Selector: dùng role và label, đừng bao giờ dùng chuỗi CSS dễ vỡ
* Mỗi flow một unhappy path: input sai, mất mạng, session hết hạn
* Test phải độc lập - chạy thứ tự nào cũng được, không dùng chung
  state, mỗi test tự seed dữ liệu
* Headless trên CI, headed ở máy local để debug
* Chỉ chụp screenshot + trace khi fail

Chạy cả bộ test, cho tôi xem kết quả, sửa những gì fail, và nói rõ bộ
test vẫn CHƯA phủ được gì.
```

12

Dọn sạch Dead Code

Tìm và xóa dead code trong repo này.

Phạm vi: [TOÀN BỘ REPO / THƯ MỤC CỤ THỂ]

* Export, component, hook, util không ai dùng
* Nhánh không bao giờ chạy tới + block code bị comment
* Dependency trong package.json không ai import
* Feature flag cũ bị kẹt luôn bật hoặc luôn tắt
* Logic trùng lặp đáng lẽ nên gộp làm một
* Class CSS chết và asset không dùng

Tìm kiếm xác minh trước MỖI lần xóa - dynamic import và tham chiếu dạng chuỗi cũng tính. Xóa theo từng commit nhỏ, chạy [BUILD + TESTS] sau mỗi commit, rồi báo tổng số dòng đã xóa cùng những chỗ bạn chưa chắc 100%.

🧹

13

Viết Git Commit gọn gàng

```
Commit các thay đổi đã stage một cách tử tế.

Quy ước: [CONVENTIONAL COMMITS / ĐỊNH DẠNG CỦA BẠN]

* Tách các thay đổi không liên quan thành commit riêng
* Định dạng: type(scope): đã đổi gì và vì sao
  feat / fix / refactor / chore / docs / test
* Subject dưới 50 ký tự, dùng câu mệnh lệnh
* Phần body giải thích VÌ SAO, wrap ở cột 72
* Tham chiếu ticket: [TICKET-ID]
* Đừng bao giờ trộn refactor với thay đổi hành vi trong cùng một commit
* Đừng bao giờ commit [SECRETS / .ENV / FILE SINH TỰ ĐỘNG]

Cho tôi xem kế hoạch - file nào vào commit nào + nội dung message -
trước khi commit bất cứ thứ gì. Rồi commit từng cái một để tôi có thể
chặn giữa chừng.
```

14

Dùng Hooks làm Guardrails

Thiết lập Claude Code hooks làm guardrail ở đây.

Stack: [STACK + PACKAGE MANAGER CỦA BẠN]

* PostToolUse: chạy [LINT + TYPECHECK] sau mỗi lần sửa file, đẩy lỗi ngược lại ngay
* PreToolUse: chặn sửa [ĐƯỜNG DẪN ĐƯỢC BẢO VỆ: MIGRATIONS, .ENV, PROD CONFIG]
* Stop: chạy [TEST SUITE] trước khi kết thúc session
* Notification: báo tôi bằng [ÂM THANH / SLACK] khi cần tôi vào cuộc

Viết các script hook + phần khai báo trong settings.json. Giữ mỗi script dưới 20 dòng và exit khác 0 kèm thông báo rõ ràng để agent biết chính xác phải sửa gì. Rồi kích hoạt từng hook có chủ đích và cho tôi xem chúng chạy.

15

Biến một tác vụ thành Skill

```
Biến tác vụ lặp đi lặp lại này thành một Claude Code skill.

Tác vụ tôi làm hoài: [MÔ TẢ TÁC VỤ + CÁC BƯỚC]

* Tạo .claude/skills/[NAME]/SKILL.md
* Frontmatter: name + description kèm đúng những câu tôi hay nói để
  kích hoạt nó
* Body: quy trình đánh số, các quy ước của tôi, edge case
* Cái gì nên hỏi tôi, cái gì tự suy ra được
* "Xong" thì trông như thế nào

Rồi chạy thử skill trên một ví dụ thật và tinh chỉnh cho tới khi output
khớp với cách tôi vẫn làm tay.
```

⚡