

Chương 5: Cẩm nang thực hành (Guides & Examples)

- [Trang 5.1: Lên ý tưởng và so sánh giải pháp với /brainstorm.](#)
- [Trang 5.2: Lập kế hoạch dự án chi tiết \(Blueprint\) với /plan.](#)
- [Trang 5.3: Xây dựng ứng dụng/tính năng mới từ đầu với /create.](#)
- [Trang 5.4: Nâng cấp và cải tiến mã nguồn hiện có với /enhance.](#)
- [Trang 5.5: Thiết kế giao diện Premium chuẩn UI-UX với /ui-ux-pro-max.](#)
- [Trang 5.6: Protocol 4 bước sửa lỗi có hệ thống với /debug.](#)
- [Trang 5.7: Tự động hóa kiểm thử mã nguồn với /test.](#)
- [Trang 5.8: Quản lý môi trường xem trước và theo dõi tiến độ \(/preview & /status\).](#)

Trang 5.1: Lên ý tưởng và so sánh giải pháp với /brainstorm.

1. Mục đích và triết lý

`/brainstorm` được thiết kế với triết lý rõ ràng: "**Suy nghĩ trước, code sau.**"

Nhiều quyết định kiến trúc sai lầm bắt nguồn từ việc bắt tay vào code ngay mà không xem xét đủ các lựa chọn. Workflow này buộc AI phải trình bày **ít nhất 3 phương án khác nhau** với đầy đủ ưu/nhược điểm và mức độ nỗ lực, giúp bạn đưa ra lựa chọn sáng suốt.

“**Nguyên tắc vàng:** Luôn chạy `/brainstorm` trước khi bắt đầu bất kỳ tính năng phức tạp nào.

2. Khi nào nên dùng `/brainstorm`?

☐ Nên dùng khi:

- Bắt đầu một tính năng mới và chưa chắc về hướng triển khai.
- Cần so sánh các công nghệ, thư viện hoặc kiến trúc khác nhau.
- Muốn có cái nhìn khách quan về trade-off.
- Cần thuyết phục đội hoặc cấp trên về một lựa chọn kỹ thuật.

☐ Không cần dùng khi:

- Tác vụ đơn giản, rõ ràng (fix typo, thêm field đơn giản...).
- Đã có quyết định rõ ràng và chỉ cần triển khai.

3. Quy trình thực hiện

Bước 1: Mô tả thách thức

```
bash  
/brainstorm [mô tả vấn đề hoặc nhiệm vụ cần giải quyết]
```

Lưu ý: Cung cấp càng nhiều ngữ cảnh (context) thì AI đưa ra phân tích càng chính xác.

Bước 2: Xem xét các phương án

AI sẽ trình bày **ít nhất 3 phương án** theo cấu trúc chuẩn:

Phương án 1: [Tên phương án]

- Ưu điểm: ...
- Nhược điểm: ...
- Nỗ lực: Thấp / Trung bình / Cao
- Phù hợp khi: ...

Phương án 2: [Tên phương án]

...

Phương án 3: [Tên phương án]

...

Khuyến nghị: Phương án X vì [lý do cụ thể dựa trên ngữ cảnh của bạn]

Bước 3: Trao đổi và quyết định

- Hỏi thêm về phương án bạn quan tâm.
- Yêu cầu AI phân tích sâu hơn về một khía cạnh cụ thể.
- Xác nhận phương án để tiến hành bước tiếp theo (`/create` hoặc `/enhance`).

4. Ví dụ phiên làm việc thực tế

Ví dụ 1: Chọn chiến lược quản lý state

Bạn: /brainstorm chiến lược quản lý state cho ứng dụng React có 50+ màn hình và cần đồng bộ real-time

AI: Tôi sẽ phân tích 3 phương án quản lý state phù hợp với yêu cầu của bạn:

Phương án 1: Zustand + React Query

- Ưu điểm: Nhẹ (2KB), không boilerplate, tích hợp tốt với server state
- Nhược điểm: Cần học 2 thư viện riêng biệt
- Nỗ lực: Thấp
- Phù hợp: Dự án muốn linh hoạt, tránh over-engineering

Phương án 2: Redux Toolkit + RTK Query

- Ưu điểm: Chuẩn ngành, devtools mạnh, dễ debug
- Nhược điểm: Nhiều boilerplate hơn, bundle size lớn hơn
- Nỗ lực: Trung bình
- Phù hợp: Team đã quen Redux, dự án lớn cần chuẩn hóa

Phương án 3: Jotai + SWR

- Ưu điểm: Atomic state, cực kỳ linh hoạt, tránh re-render thừa
- Nhược điểm: Ít tài liệu hơn, cộng đồng nhỏ hơn
- Nỗ lực: Trung bình
- Phù hợp: Dự án phức tạp với nhiều state độc lập

Khuyến nghị: Phương án 1 (Zustand + React Query) cho dự án mới, vì cân bằng tốt giữa đơn giản và hiệu năng, đặc biệt phù hợp với yêu cầu real-time của bạn.

Bạn muốn tôi phân tích sâu hơn về phương án nào?

☐ Bạn: Cho tôi biết thêm về Phương án 1, cách triển khai real-time như thế nào?

☐ AI: [Phân tích chi tiết hơn về Zustand + React Query + WebSocket pattern...]

Ví dụ 2: Chọn database cho hệ thống IoT

☐ Bạn: /brainstorm lựa chọn database cho hệ thống IoT cần lưu
1 triệu sự kiện/ngày, query theo khoảng thời gian

☐ AI: Phân tích 3 phương án phù hợp:

Phương án 1: TimescaleDB (PostgreSQL mở rộng)

Phương án 2: InfluxDB (Time-series database chuyên dụng)

Phương án 3: MongoDB với TTL collections

[Phân tích chi tiết từng phương án...]

☐ Khuyến nghị: TimescaleDB nếu đội quen SQL, InfluxDB nếu cần hiệu năng cao nhất cho time-series.

5. Mẹo để brainstorm hiệu quả

Cung cấp ngữ cảnh đầy đủ

Kém hiệu quả:

/brainstorm cách lưu dữ liệu

Hiệu quả hơn:

/brainstorm cách lưu dữ liệu người dùng cho ứng dụng SaaS B2B với 10,000 tenant, cần multi-tenancy, đang dùng PostgreSQL, team 3 backend engineer

Hỏi về trade-off cụ thể

Sau khi nhận phương án:

"Nếu dự án mở rộng lên 10 lần trong 1 năm, phương án nào sẽ scale tốt hơn?"

"Chi phí vận hành của phương án 2 và 3 khác nhau thế nào?"

Yêu cầu so sánh trực tiếp

"Tôi đang phân vân giữa Phương án 1 và 3.

Hãy so sánh trực tiếp 2 phương án này theo tiêu chí: hiệu năng, độ phức tạp, chi phí."

6. Bước tiếp theo sau khi có quyết định

Quyết định	Bước tiếp theo
Xây dựng tính năng từ đầu	<code>/create</code> với phương án đã chọn
Cải tiến code đang có	<code>/enhance</code> với phương án đã chọn
Cần kế hoạch chi tiết	<code>/plan</code> để phân rã công việc

Quyết định	Bước tiếp theo
Muốn thử nghiệm nhanh trước	Tạo prototype rồi <input type="button" value="/test"/>

Trang 5.2: Lập kế hoạch dự án chi tiết (Blueprint) với /plan.

1. Mục đích và triết lý

`/plan` là lệnh kích hoạt chế độ "**Lập kế hoạch thuần túy**" — AI trở thành `project-planner` và nhiệm vụ duy nhất là tạo ra một kế hoạch chi tiết, không viết một dòng code nào.

“**Quy tắc sắt:** `/plan` kết thúc = Chỉ có file kế hoạch `PLAN.md`. Không có code, không có implementation. Đây là ranh giới nghiêm ngặt.

2. Khi nào nên dùng `/plan`?

☐ Nên dùng khi:

- Bắt đầu một dự án hoặc tính năng lớn (hơn 2-3 ngày công).
- Cần trình bày kế hoạch cho đội hoặc cấp trên phê duyệt.
- Muốn phân rã công việc thành các task nhỏ có thể giao cho thành viên.
- Cần ước tính thời gian và phụ thuộc giữa các tác vụ.

3. Quy trình thực hiện (4 bước)

Bước 1: Khởi động lệnh

```
bash
/plan [mô tả ngắn gọn về dự án hoặc tính năng]
```

Ví dụ:

```
bash
/plan hệ thống quản lý nhân sự với đăng nhập SSO,
phân quyền RBAC và báo cáo Excel
```

Bước 2: Trả lời Câu hỏi Socratic

AI (`project-planner`) sẽ đặt một loạt câu hỏi làm rõ yêu cầu trước khi lên kế hoạch. Đây là **Cổng Socratic (Socratic Gate)** — bước bắt buộc để đảm bảo kế hoạch đúng ngay từ đầu.

Các câu hỏi thường xoay quanh:

- **Người dùng và quy mô:** Bao nhiêu người dùng? Loại người dùng nào?
- **Tech stack:** Đang dùng framework gì? Database nào?
- **Ràng buộc:** Deadline? Budget? Team size?
- **Tích hợp:** Có cần kết nối hệ thống nào khác không?
- **Ưu tiên:** MVP tối thiểu là gì? Tính năng nào có thể trì hoãn?

Bước 3: AI lên kế hoạch

Sau khi bạn trả lời đủ câu hỏi, AI tổng hợp thông tin và tạo ra file kế hoạch.

Bước 4: Xem xét và chỉnh sửa kế hoạch

File kế hoạch không phải là tài liệu cố định. Chỉnh sửa trực tiếp `PLAN.md` để:

- Điều chỉnh ưu tiên.
- Thêm/bớt tính năng.
- Phân công người phụ trách từng task.

4. Cấu trúc file kế hoạch đầu ra (`PLAN-{tên}.md`)

File kế hoạch thường được tạo tại docs/PLAN-tên-dự-án.md với nội dung đầy đủ:

markdown

Kế hoạch: Hệ thống Quản lý Nhân sự

Tổng quan

- Mô tả ngắn gọn về dự án
- Tech stack: Next.js 14, PostgreSQL, Prisma, NextAuth

Phân tích yêu cầu

- Người dùng: HR Admin, Manager, Employee (3 vai trò)
- Quy mô: ~500 nhân viên, 50 manager
- ...

Kiến trúc hệ thống

[Sơ đồ kiến trúc đơn giản]

Phân rã tác vụ

Giai đoạn 1: Nền tảng (Sprint 1-2)

- [] Thiết lập dự án Next.js
- [] Cấu hình database schema
- [] Triển khai SSO với NextAuth
- [] RBAC cơ bản (3 vai trò)

Giai đoạn 2: Tính năng cốt lõi (Sprint 3-4)

- [] Module quản lý nhân viên (CRUD)
- [] Module chấm công
- [] Module nghỉ phép

Giai đoạn 3: Báo cáo (Sprint 5)

- [] Dashboard tổng hợp

- [] Export Excel

Phụ thuộc tác vụ

- RBAC cần hoàn thành trước các module nghiệp vụ
- Dashboard cần dữ liệu từ tất cả module khác

Ước tính thời gian

- Tổng: ~5 sprint (10 tuần)
- MVP (Giai đoạn 1+2): 8 tuần

Agent phụ trách

- database-architect: Schema thiết kế
- backend-specialist: API + Business logic
- frontend-specialist: UI/UX
- security-auditor: Review xác thực và phân quyền

5. Ví dụ phiên làm việc thực tế

☐ Bạn: /plan tích hợp cổng thanh toán VNPAY vào ứng dụng Next.js của tôi

☐ AI (project-planner): Tôi cần làm rõ một số điểm để lên kế hoạch chính xác:

1. Ứng dụng của bạn hiện đang ở giai đoạn nào?
(MVP mới, đang hoạt động, hay đang scale)
2. Bạn cần hỗ trợ những loại giao dịch nào?
(Thanh toán 1 lần, đăng ký định kỳ, hay cả hai)
3. Tech stack backend hiện tại là gì?
(Next.js API Routes, Express, hay gì khác)
4. Bạn cần xử lý hoàn tiền (refund) không?

5. Timeline mong muốn?

Bạn:

1. Đang hoạt động, có ~500 đơn hàng/tháng
2. Thanh toán 1 lần cho đơn hàng
3. Next.js API Routes + PostgreSQL
4. Cần hoàn thiện một phần và toàn bộ
5. 2 tuần

AI: Tôi đã hiểu đủ thông tin. Đây là kế hoạch:

[Tạo file docs/PLAN-vnpay-integration.md với đầy đủ nội dung...]

Đã tạo kế hoạch tại docs/PLAN-vnpay-integration.md

Kế hoạch bao gồm 3 giai đoạn, ước tính 8 ngày công.

Bạn có muốn điều chỉnh gì không?

6. Mẹo lập kế hoạch hiệu quả

Trả lời đầy đủ câu hỏi Socratic

Câu hỏi càng chi tiết, kế hoạch càng chính xác. Đừng trả lời ngắn gọn như "bình thường" hay "tùy bạn".

Định nghĩa MVP rõ ràng

"MVP tối thiểu cho tôi là: có thể đăng nhập, xem danh sách nhân viên và thêm mới. Các tính năng còn lại có thể phát hành sau."

Chỉnh sửa kế hoạch ngay

Sau khi AI tạo kế hoạch:

- Mở file `PLAN.md` và đọc kỹ.
- Thêm người phụ trách (`@team-member`) vào từng task.
- Điều chỉnh độ ưu tiên nếu cần.

Commit kế hoạch vào Git

```
bash
git add docs/PLAN-tên-dự-án.md
git commit -m "plan: add feature X implementation plan"
```

Điều này tạo lịch sử quyết định cho cả đội có thể tham khảo sau này.

7. Sau khi có kế hoạch: Bước tiếp theo

```
Kế hoạch được phê duyệt ☐
|
▼
☐☐ Triển khai:
/create [task đầu tiên trong kế hoạch]
hoặc
/enhance [tính năng cụ thể cần thêm vào]
```

Trang 5.3: Xây dựng ứng dụng/tính năng mới từ đầu với `/create`.

1. Mục đích

`/create` đóng vai trò như một **trình thuật sĩ tạo ứng dụng (Application Creation Wizard)** — dẫn dắt bạn qua một cuộc hội thoại tương tác để hiểu yêu cầu, lên kế hoạch kiến trúc và điều phối các Agent chuyên gia cùng nhau xây dựng ứng dụng hoàn chỉnh.

“ Điểm khác biệt quan trọng:

- `/create` = Xây dựng từ **trang giấy trắng (greenfield)**.
- `/enhance` = Thêm vào hệ thống **đang hoạt động (brownfield)**.

2. Khi nào nên dùng `/create`?

□ Sử dụng `/create` khi:

- Bắt đầu một dự án hoàn toàn mới.
- Tạo một module/service độc lập mới trong dự án lớn.
- Cần tạo nhanh một prototype để kiểm tra ý tưởng.
- Muốn triển khai một tính năng phức tạp từ đầu mà không lo ảnh hưởng code cũ.

3. Quy trình thực hiện (3 bước)

Bước 1: Mô tả ứng dụng cần tạo

```
bash
/create [mô tả ứng dụng và các yêu cầu chính]
```

Cung cấp thông tin càng đầy đủ thì AI tạo ra code càng chính xác với nhu cầu của bạn.

Bước 2: Xem xét và phê duyệt kế hoạch

Agent `app-builder` sẽ tổng hợp yêu cầu và đề xuất:

- **Tech stack:** Framework, database, thư viện chính.
- **Cấu trúc thư mục:** Cách tổ chức code.
- **Database schema:** Các bảng và quan hệ chính.
- **Danh sách tính năng:** Những gì sẽ được xây dựng.

Bạn cần phê duyệt (Y) hoặc yêu cầu điều chỉnh trước khi AI bắt đầu viết code.

Bước 3: Các Agent cùng xây dựng

Sau khi phê duyệt, `orchestrator` sẽ điều phối các Agent chuyên gia:

```
orchestrator điều phối:
├─ database-architect → Tạo database schema và migrations
├─ backend-specialist → Viết API endpoints và business logic
├─ frontend-specialist → Xây dựng UI components và pages
└─ test-engineer → Tạo bộ test cơ bản
```

4. Ví dụ phân làm việc thực tế

Ví dụ: Tạo ứng dụng quản lý công việc (Todo)

```
□□ Bạn: /create ứng dụng todo có dark mode, xác thực người dùng,
phân loại task theo dự án, và có thể chia sẻ task với người khác
```

AI (app-builder):

Kế hoạch xây dựng:

Tech Stack:

- Framework: Next.js 14 (App Router)
- Database: PostgreSQL + Prisma ORM
- Auth: NextAuth.js (Email + Google OAuth)
- UI: Tailwind CSS + shadcn/ui
- State: Zustand

Database Schema:

- User (id, email, name, avatar)
- Project (id, name, color, owner_id)
- Task (id, title, status, due_date, project_id, assignee_id)
- ProjectMember (project_id, user_id, role)

Tính năng sẽ tạo:

- Đăng ký/đăng nhập (Email + Google)
- Dark mode toggle
- CRUD Projects
- CRUD Tasks với drag-and-drop
- Chia sẻ project và gán task
- Bộ test cơ bản

Bạn có muốn tiến hành không? (Y/N)

Bạn: Y (nhưng dùng MySQL thay vì PostgreSQL)

AI: Đã điều chỉnh sang MySQL. Đang bắt đầu xây dựng...

☐ database-architect: Đang tạo schema Prisma cho MySQL...

⚙️ backend-specialist: Đang tạo API routes...

☐ frontend-specialist: Đang xây dựng components...

☐ test-engineer: Đang viết test cases...

☐ Hoàn thành! Chạy lệnh sau để khởi động:

```
npm install && npx prisma migrate dev && npm run dev
```

5. Cấu trúc dự án được tạo ra (ví dụ Next.js)

```
my-todo-app/  
├─ app/  
│ ├─ (auth)/  
│ │ ├─ login/page.tsx  
│ │ └─ register/page.tsx  
│ └─ dashboard/  
│ │ ├─ layout.tsx  
│ │ └─ page.tsx  
│ └─ projects/[id]/page.tsx  
└─ api/  
    └─ auth/[...nextauth]/route.ts  
        └─ projects/route.ts  
            └─ tasks/route.ts  
├─ components/  
│ └─ TaskCard.tsx  
│ └─ ProjectList.tsx  
│ └─ ThemeToggle.tsx  
├─ lib/  
│ └─ db.ts (Prisma client)  
│ └─ auth.ts (NextAuth config)  
├─ prisma/  
│ └─ schema.prisma  
└─ tests/
```

```
| └─ api.test.ts
| └─ components.test.tsx
└─ package.json
```

6. Mẹo để `/create` tạo ra kết quả tốt nhất

Mô tả cụ thể và đầy đủ

Thiếu thông tin:

`/create` ứng dụng bán hàng

Đầy đủ thông tin:

`/create` ứng dụng thương mại điện tử bán quần áo thời trang.

Người dùng có thể duyệt sản phẩm, thêm vào giỏ hàng,

thanh toán qua VNPay. Admin quản lý sản phẩm và đơn hàng.

Tech stack: Next.js + PostgreSQL + Stripe (nếu cần quốc tế).

MVP không cần: review, wishlist, loyalty points.

Bắt đầu từ MVP

Tập trung vào tính năng cốt lõi. Sau khi có MVP chạy được, dùng `/enhance` để thêm dần.

1. `/create` [core features only] → MVP
2. `/enhance` [feature A] → Version 1.1
3. `/enhance` [feature B] → Version 1.2

Phê duyệt lần lượt từng thành phần

Nếu muốn kiểm soát chặt chẽ hơn, sau khi nhận kế hoạch:

"Hãy tạo database schema trước để tôi review,
sau đó mới tiếp tục với backend."

7. Sau khi `/create` hoàn thành

Bước tiếp theo	Lệnh
Kiểm tra ứng dụng	<code>/preview start</code>
Chạy tests	<code>/test</code>
Thêm tính năng mới	<code>/enhance [tính năng]</code>
Kiểm tra bảo mật	Hỏi <code>@security-auditor review code này</code>
Triển khai	<code>/deploy</code>

Trang 5.4: Nâng cấp và cải tiến mã nguồn hiện có với /enhance.

1. Mục đích và triết lý

`/enhance` là công cụ của vòng lặp phát triển hàng ngày — dùng để **thêm tính năng mới vào hệ thống đang chạy** mà không làm gián đoạn hoặc phá vỡ những gì đang hoạt động.

Triết lý cốt lõi của `/enhance` là "**phẫu thuật chính xác**" (**surgical precision**): chỉ thay đổi đúng những gì cần thiết, không viết lại, không refactor không liên quan.

“ Khi nào `/enhance` vs `/create` ?

- `/enhance`: "Dự án đang chạy, tôi muốn thêm tính năng X" → ENHANCE
- `/create`: "Tôi chưa có gì, tôi muốn xây dựng tính năng X từ đầu" → CREATE

2. Điều đặc biệt: Phân tích code trước khi thay đổi

Khác với `/create` (bắt đầu từ trang trắng), `/enhance` **bắt buộc phải đọc và hiểu codebase hiện tại** trước khi đề xuất bất kỳ thay đổi nào.

`/enhance` [yêu cầu]

|



AI đọc cấu trúc dự án hiện tại

|



☐ Phân tích các file liên quan đến yêu cầu

|



☐ Lập kế hoạch thay đổi tối thiểu cần thiết

|



☐ Đề xuất plan → Bạn phê duyệt → AI triển khai

|



☐ Chạy kiểm tra để xác nhận không có regression

3. Phương pháp "Vertical Slice" (Lát cắt dọc)

Khi thêm tính năng mới, `/enhance` áp dụng phương pháp **Vertical Slice** — triển khai đầy đủ một tính năng theo chiều dọc từ database đến UI, thay vì làm từng lớp riêng lẻ.

Ví dụ: Thêm tính năng "Bình luận bài viết"

Vertical Slice (☐ cách `/enhance` làm):

├─ Database: Bảng comments + migration

├─ Backend: API `/comments` (GET, POST, DELETE)

├─ Frontend: Component `CommentSection` + `CommentItem`

└─ Testing: Unit test + Integration test

Thay vì (☐ cách dễ gây lỗi):

├─ Tuần 1: Xong hết phần database

├─ Tuần 2: Xong hết phần backend

└─ Tuần 3: Xong hết phần frontend

Ưu điểm: Bạn có thể test và dùng tính năng ngay khi một Vertical Slice hoàn thành, không cần đợi toàn bộ.

4. Quy trình thực hiện

Bước 1: Mô tả tính năng cần thêm

```
bash
/enhance [mô tả tính năng + ràng buộc nếu có]
```

Bước 2: AI phân tích codebase

AI sẽ đọc:

- package.json để biết tech stack và thư viện.
- Cấu trúc thư mục `src/` để hiểu tổ chức code.
- Các file liên quan đến yêu cầu của bạn.

Bước 3: AI đề xuất kế hoạch thay đổi

☐ Kế hoạch thay đổi:

Tìm thấy:

- Framework: Next.js 14 với App Router
- Đang dùng Tailwind CSS
- Chưa có cấu hình dark mode

Thay đổi dự kiến:

1. Cập nhật tailwind.config.ts (thêm darkMode: 'class')
2. Tạo component ThemeProvider (dùng next-themes)
3. Tạo component ThemeToggle
4. Cập nhật app/layout.tsx để bọc trong ThemeProvider
5. Thêm class dark: vào các component chính

Thư viện cần cài: next-themes

Không ảnh hưởng đến: logic nghiệp vụ, database, API

Tiến hành? (Y/N)

Bước 4: Phê duyệt và triển khai

Sau khi bạn xác nhận, AI triển khai từng thay đổi và chạy kiểm tra.

5. Ví dụ phiên làm việc thực tế

Ví dụ 1: Thêm dark mode

Bạn: /enhance thêm dark mode toggle vào ứng dụng Next.js của tôi, đảm bảo lưu lại tùy chọn của người dùng

AI: Đang phân tích codebase...

Kết quả phân tích:

- Next.js 14 + App Router
- Tailwind CSS (chưa có darkMode config)
- localStorage có thể dùng (không có SSR conflict nếu dùng next-themes)

Kế hoạch (5 thay đổi):

1. npm install next-themes
2. Cập nhật tailwind.config.ts
3. Tạo ThemeProvider component
4. Tạo ThemeToggle button (sun/moon icon)
5. Cập nhật layout.tsx

Tiến hành? (Y/N)

Bạn: Y

AI:

- Đã cài next-themes
- Đã cập nhật tailwind.config.ts
- Đã tạo components/ThemeProvider.tsx
- Đã tạo components/ThemeToggle.tsx
- Đã cập nhật app/layout.tsx

Chạy `/preview start` để xem kết quả.

Tùy chọn dark/light mode được lưu vào localStorage tự động.

Ví dụ 2: Thêm tính năng dashboard analytics

Bạn: /enhance thêm dashboard analytics cho admin.

Cần: biểu đồ doanh thu 30 ngày, top 5 sản phẩm bán chạy, số lượng đơn hàng theo trạng thái

Requirements:

- Dùng Recharts cho biểu đồ
- API endpoint riêng để aggregate data
- Chỉ admin mới xem được

AI: Đang phân tích codebase...

Kết quả phân tích:

- Tìm thấy role-based auth trong middleware.ts
- Đang dùng Prisma + PostgreSQL
- Chưa có Recharts

☐ Kế hoạch (Vertical Slice):

Database: Không cần migration mới (aggregate từ bảng orders)

Backend: GET /api/admin/analytics (3 queries tổng hợp)

Frontend: Trang /admin/analytics với 3 chart components

Testing: Test API endpoint + mock data

Tiến hành? (Y/N)

6. Bảo vệ code hiện có — Các nguyên tắc an toàn

`/enhance` theo các nguyên tắc sau để không làm hỏng hệ thống:

Nguyên tắc	Diễn giải
Thay đổi tối thiểu	Chỉ tạo/sửa file thực sự cần thiết
Cô lập tính năng	Tính năng mới trong thư mục/module riêng
Không refactor ngoài phạm vi	Không sửa những gì không liên quan đến yêu cầu
Regression test	Chạy test hiện có sau mỗi thay đổi
Phê duyệt trước khi làm	Luôn trình bày kế hoạch để bạn xem xét trước

7. Mẹo để `/enhance` hoạt động hiệu quả

Kiểm tra dependencies trước

"Trước khi thêm thư viện XYZ, hãy kiểm tra xem nó có tương thích với phiên bản Node.js/React hiện tại không"

Yêu cầu xem kế hoạch trước khi làm

Luôn đọc kỹ kế hoạch thay đổi trước khi gõ Y. Đặc biệt chú ý:

- Số lượng file bị thay đổi có hợp lý không?
- Có file nào "không liên quan" cũng bị sửa không?

Yêu cầu test cụ thể

```
/enhance thêm tính năng export CSV,  
và viết test để đảm bảo CSV output đúng định dạng
```

Dùng với `/test` sau mỗi lần enhance

```
bash  
# Sau khi enhance xong  
/test # Chạy toàn bộ test suite để kiểm tra regression
```

Trang 5.5: Thiết kế giao diện Premium chuẩn UI-UX với /ui-ux-pro-max.

1. Mục đích

`/ui-ux-pro-max` là workflow chuyên biệt để tạo ra **giao diện web chất lượng cao, đẹp mắt và có tỉ lệ chuyển đổi cao** — đặc biệt là các trang Landing Page — trong thời gian tối thiểu.

Điểm khác biệt so với các cách thiết kế thông thường:

- Không phải output mẫu chung chung.
- AI áp dụng đúng **phong cách nghệ thuật** và **bảng màu** bạn chọn.
- Kết quả là code hoàn chỉnh: HTML semantic, components tái sử dụng, animations.

2. Thư viện phong cách NextLevelBuilder

Workflow này được tối ưu hóa để làm việc cùng với thư viện **50 phong cách thiết kế chuyên nghiệp** tại:

📄 nextlevelbuilder.io — Style Gallery

Thư viện cung cấp:

- **50 Style Prompts** (Glassmorphism, Claymorphism, Neubrutalism, Modern SaaS, Minimalist...)
 - **21 Bảng màu** đã được thiết kế và kiểm tra cẩn thận.
 - **50 Font Combinations** phù hợp từng phong cách.
-

3. Quy trình thực hiện (3 bước)

Bước 1: Chọn phong cách (tùy chọn nhưng khuyến nghị)

Truy cập nextlevelbuilder.io, duyệt qua các phong cách và chọn một cái phù hợp với thương hiệu của bạn.

Nếu không chọn cụ thể, AI sẽ tự đề xuất phong cách phù hợp dựa trên mô tả của bạn.

Bước 2: Chạy lệnh với mô tả đầy đủ

```
bash
/ui-ux-pro-max [mô tả trang + yêu cầu về phong cách, màu sắc, sections]
```

Bước 3: AI xây dựng và bàn giao

AI tạo ra đầy đủ:

- **Cấu trúc HTML semantic** (chuẩn SEO, đúng heading hierarchy).
- **Các UI component** có thể tái sử dụng.
- **Micro-animations** và hiệu ứng hover.
- **Responsive design** hoạt động tốt trên mobile.

4. Ví dụ phiên làm việc

Ví dụ 1: Landing page cho công ty dịch vụ thú cưng

```
📄 Bạn: /ui-ux-pro-max tạo landing page cho dịch vụ chăm sóc thú cưng cao cấp. Phong cách: Claymorphism với màu ấm áp và vui tươi.
```

Cần có: hero section, danh sách dịch vụ, gallery ảnh thú cưng, bảng giá 3 gói, testimonials từ chủ thú cưng, và form đặt lịch.

📄 AI:

📄 Phân tích phong cách:

Style: Claymorphism (3D soft shadows, rounded shapes)

Colors: Warm Amber #F59E0B + Rose #FB7185 + Cream #FEF3C7

Typography: Nunito (playful, rounded) + Inter (body)

Effects: Soft shadows, bouncy hover animations, gradient backgrounds

📄 Đang xây dựng...

- Tạo xong Hero Section với animated pet illustrations
- Tạo xong Services Grid (6 dịch vụ với icon claymorphism)
- Tạo xong Photo Gallery (masonry layout)
- Tạo xong Pricing Cards (3 gói với highlight card)
- Tạo xong Testimonial Carousel
- Tạo xong Booking Form với validation

📄 Tạo xong app/page.tsx

Chạy /preview start để xem kết quả.

Ví dụ 2: Landing page SaaS fintech

📄 Bạn: /ui-ux-pro-max landing page cho SaaS phân tích tài chính doanh nghiệp.

Phong cách: Glassmorphism dark mode với màu xanh navy và tím.

Sections: Hero với demo video, Features grid, Pricing 3 plans,

Dashboard preview screenshot, FAQ, CTA cuối trang.

📄 AI:

📄 Phân tích phong cách:

Style: Glassmorphism + Dark Mode

Colors: Navy #0F172A bg + Blue #3B82F6 primary + Violet #8B5CF6 accent

Typography: Inter (clean, professional)

Effects: Glass cards với backdrop-blur, gradient glow effects, subtle particle animations

[AI triển khai từng section...]

☑ Hoàn thành! High-conversion SaaS landing page ready.

5. Cách mô tả yêu cầu hiệu quả

Mô tả theo 4 yếu tố

/ui-ux-pro-max [Loại trang] cho [Đối tượng/Ngành]

[Phong cách + Màu sắc]

[Danh sách sections]

[Yêu cầu đặc biệt nếu có]

Ví dụ hoàn chỉnh:

/ui-ux-pro-max Landing page cho startup EdTech dạy lập trình online.

Phong cách: Modern SaaS với gradient tím-xanh, clean và chuyên nghiệp.

Sections: Hero (headline + CTA), Course Preview (3 khóa học),

Instructor bio, Student results/stats, Pricing (monthly/yearly),

FAQ, và footer với newsletter signup.

Đặc biệt: Thêm countdown timer cho giá khuyến mãi.

Các từ khóa phong cách phổ biến

Từ khóa	Giải thích
Glassmorphism	Card kính trong suốt, blur background, thường dùng với dark mode
Claymorphism	Hình khối 3D mềm mại, màu ấm, bóng đổ dày
Neubrutalism	Viền đen dày, màu sắc bold, không gradient, phong cách "thô"
Modern SaaS	Sạch, tối giản, tập trung vào product screenshot
Minimalist	Nhiều khoảng trống, typography nổi bật

Từ khóa	Giải thích
Gradient mesh	Màu gradient phức tạp, hiệu ứng ánh sáng

6. Kết quả bàn giao

Sau khi `/ui-ux-pro-max` hoàn thành, bạn nhận được:

Thành phần	Mô tả
<code>app/page.tsx</code>	Trang chính với tất cả sections
<code>components/</code>	Các UI components tái sử dụng (Hero, Pricing, Testimonial...)
<code>CSS/Tailwind</code>	Styles theo đúng phong cách đã chọn
<code>SEO meta tags</code>	Title, description, OG tags
<code>Accessibility</code>	ARIA labels, keyboard navigation

7. Bước tiếp theo sau khi tạo giao diện

```
bash
/preview start # Xem kết quả ngay tại localhost:3000
/enhance [thay đổi cụ thể] # Điều chỉnh màu sắc, nội dung...
/deploy # Đưa lên production khi hài lòng
```

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 ngưng hổng."

☐ 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(); ← LỖ!
```

☐ 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...

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

=== Phase 2: Hypothesis ===

GIẢI THUYẾT 1: Connection pool cạn kiệt dưới tải cao

GIẢI THUYẾT 2: Database connection timeout không được xử lý

GIẢI 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 Giải 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

<input type="checkbox"/> Không	<input type="checkbox"/> 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

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
<code>/test</code>	Chạy toàn bộ test suite hiện có
<code>/test [file hoặc thư mục]</code>	Tạo test cases cho file cụ thể
<code>/test coverage</code>	Báo cáo tỉ lệ bao phủ (coverage report)
<code>/test --watch</code>	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...

Tim 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	<code>*.test.ts</code> , <code>*.spec.ts</code>	Phổ biến nhất với React/Node
Vitest	<code>*.test.ts</code> , <code>*.spec.ts</code>	Nhanh hơn Jest, dành cho Vite
Playwright	<code>e2e/*.spec.ts</code>	E2E testing, browser automation
pytest	<code>test_*.py</code> , <code>*_test.py</code>	Python projects

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

Trang 5.8: Quản lý môi trường xem trước và theo dõi tiến độ (/preview & /status).

1. Tổng quan

Trang này hướng dẫn hai workflow đơn giản nhưng cực kỳ hữu ích trong công việc hàng ngày:

Workflow	Mục đích
<code>/preview</code>	Quản lý server phát triển cục bộ
<code>/status</code>	Xem tổng quan tiến độ và trạng thái dự án

Phần A: `/preview` — Quản lý server phát triển

A.1. Các lệnh `/preview`

Lệnh	Chức năng
<code>/preview start</code>	Khởi động server phát triển
<code>/preview stop</code>	Dừng server đang chạy
<code>/preview restart</code>	Khởi động lại server (áp dụng config mới)
<code>/preview</code>	Kiểm tra trạng thái hiện tại

A.2. Tính năng tự động nhận diện framework

Workflow này đủ thông minh để **tự nhận diện** loại dự án và chạy đúng lệnh:

Framework	Lệnh được chạy tự động
Next.js	<code>npm run dev</code> (port 3000)
Vite	<code>npm run dev</code> (port 5173)
Create React App	<code>npm start</code> (port 3000)
Express/Node.js	<code>node server.js</code> hoặc <code>npm start</code>
FastAPI (Python)	<code>uvicorn main:app --reload</code>

A.3. Xử lý xung đột Port

Nếu port mặc định (3000) đã bị chiếm bởi ứng dụng khác:

AI: Port 3000 đang bận.

Đề xuất alternatives:

- Port 3001 (available)

- Port 8080 (available)

Dùng port 3001? (Y/N)

A.4. Ví dụ phiên làm việc

Bạn: /preview start

AI:

Đang phát hiện framework...

Tìm thấy: Next.js 14 (App Router)

Đang khởi động...

Running: npm run dev

Preview sẵn sàng!

URL: http://localhost:3000

Server sẽ tiếp tục chạy. Dùng /preview stop để tắt.

Bạn: /preview

AI:

Preview Status

Status: Running

URL: http://localhost:3000

Framework: Next.js 14

Port: 3000

Uptime: 12 phút

Bạn: /preview stop

AI:

Đã dừng server preview.

Port 3000 đã được giải phóng.

A.5. Khi nào dùng /preview

Tình huống	Lệnh
Sau khi /create hoặc /enhance xong	/preview start
Cần xem UI thay đổi trên trình duyệt	/preview start
Thay đổi cấu hình (next.config.js...)	/preview restart
Trước khi tắt máy	/preview stop
Không nhớ server có đang chạy không	/preview

Phần B: `/status` — Bảng điều khiển tiến độ dự án

B.1. Mục đích

Lệnh `/status` cung cấp **bảng điều khiển tổng thể** (dashboard) để bạn nắm bắt nhanh tình trạng hiện tại của dự án, bao gồm: thông tin kỹ thuật, tiến độ các Agent đang làm việc, và trạng thái server.

B.2. Cú pháp

```
bash
/status
```

B.3. Thông tin được hiển thị

```
==== Project Status ====

📁 Project Information
Name: my-ecommerce-app
Type: Next.js + PostgreSQL
Version: 1.2.0
Files: 147 total (73 created, 12 modified, 62 unchanged)

==== Agent Status ====

📌 database-architect → Completed (Schema + Migrations)
📌 backend-specialist → Completed (All API Endpoints)
📌 frontend-specialist → In Progress (60% — Cart Page)
📌 test-engineer → Waiting (Pending frontend completion)
```

=== Test Results ===

- ☐ Unit Tests: 45/45 passed
- ☐ Integration Tests: 12/12 passed
- △ E2E Tests: 8/10 passed (2 failing)

=== Preview ===

- ☐ URL: http://localhost:3000
- ☐ Health: OK (responding in 124ms)

B.4. Ví dụ phiên làm việc

☐ Bạn: /status

☐ AI:

=== ☐ PROJECT STATUS ===

☐ my-saas-app (Next.js + Prisma + PostgreSQL)

Files: 89 tổng (34 mới tạo hôm nay)

Branch: feature/payment-integration

=== ☐ AGENT STATUS ===

- ☐ database-architect → DONE — Payment schema + Stripe webhook table
- ☐ backend-specialist → DONE — /api/payments (create, confirm, refund)
- ☐ security-auditor → DONE — Reviewed, không có high-severity issues

☐ frontend-specialist → 70% — Đang tạo CheckoutPage...

☐ test-engineer → Chưa bắt đầu

==== ☐ TEST STATUS ====

☐ 38 tests passed | ☐ 0 failed | ☐ Coverage: 82%

==== ☐ PREVIEW ====

Status: ☐ Running — http://localhost:3000

Response time: 89ms

==== ☐ NEXT STEPS ====

1. Chờ frontend-specialist hoàn thành CheckoutPage (~30 phút)
2. Chạy /test để tạo test cho payment flow
3. Chạy /deploy khi tất cả test pass

B.5. Khi nào dùng `/status`

Tình huống	Lý do
Đầu ngày làm việc	Nhắc lại tiến độ từ hôm qua
Sau khi <code>/orchestrate</code> phức tạp	Kiểm tra tất cả Agent đã xong chưa
Trước khi <code>/deploy</code>	Đảm bảo mọi thứ xanh trước khi đẩy lên production
Khi bàn giao cho đồng đội	Chia sẻ trạng thái hiện tại nhanh chóng
Khi không nhớ đang làm đến đâu	Lấy lại ngữ cảnh công việc

Phần C: Vòng lặp phát triển kết hợp

Sử dụng `/preview` và `/status` trong quy trình hàng ngày:

☐ Bắt đầu ngày làm việc:

`/status` → Xem lại tiến độ hôm qua

`/preview start` → Khởi động server để làm việc

☐ Trong giờ làm:

`/enhance [feature]` → Thêm tính năng

`/preview start` → Xem kết quả

`/test` → Kiểm tra không có regression

☐ Cuối ngày:

`/status` → Tổng kết tiến độ hôm nay

`/preview stop` → Tắt server

☐ Trước khi deploy:

`/status` → Xác nhận tất cả Agent xong việc

`/test` → Đảm bảo tất cả test pass

`/deploy` → Triển khai