Chương 3: Khám phá các khái niệm cốt lõi (Core Concepts)

Trang 3.1: AI Agents - Các đặc vụ chuyên gia và cơ chế định tuyến thông minh (Intelligent Routing).

1. Agent là gì?

Agent là một "nhân cách AI chuyên gia" được cấu hình sẵn với kiến thức sâu, công cụ và hành vi riêng biệt cho một lĩnh vực cụ thể trong phát triển phần mềm.

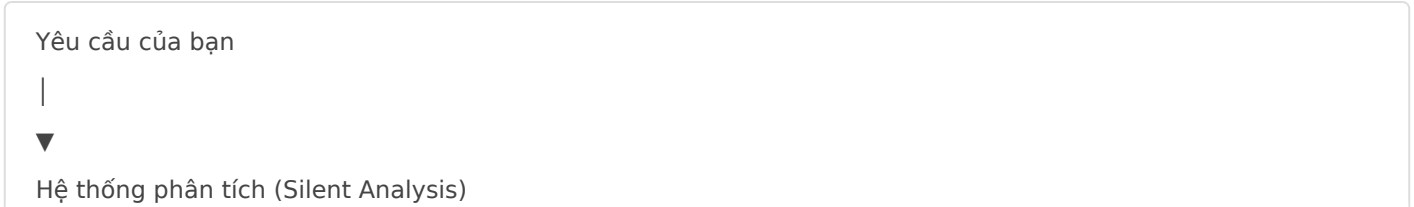
Hãy hình dung Agent như một chuyên viên tư vấn: thay vì hỏi một người biết tất cả một cách chung chung, bạn đang nói chuyện trực tiếp với **kiến trúc sư Frontend**, **chuyên gia bảo mật**, hay **kỹ sư kiểm thử** tùy theo từng vấn đề.

---

2. Hệ thống Định tuyến thông minh (Intelligent Routing)

Cơ chế hoạt động

Điểm đặc biệt quan trọng nhất: **bạn không cần phải gọi tên Agent thủ công**. Hệ thống sẽ tự động phân tích yêu cầu của bạn và kích hoạt (các) Agent phù hợp nhất.



|

▼

Phát hiện lĩnh vực (Frontend? Backend? Security?...)

|

▼

Chọn Agent chuyên gia tương ứng

|

▼

Thông báo cho bạn Agent đang được dùng:

□□ Đang áp dụng kiến thức của @[agent-name]...

|

▼

Trả lời ở cấp độ chuyên gia

Ví dụ thực tế

Yêu cầu của bạn	Agent được kích hoạt
"Thêm xác thực JWT vào API"	`@security-auditor` + `@backend-specialist`
"Sửa lỗi nút dark mode bị lệch màu"	`@frontend-specialist`
"Login trả về lỗi 500 ngẫu nhiên"	`@debugger`
"Thiết kế schema cho bảng users và orders"	`@database-architect`
"Viết unit test cho service xác thực"	`@test-engineer`
"Tối ưu tốc độ tải trang"	`@performance-optimizer`
"Lập kế hoạch tính năng thanh toán"	`@project-planner`

Ghi đè thủ công (Override)

Nếu bạn muốn buộc sử dụng một Agent cụ thể, chỉ cần đề cập tên trong yêu cầu:

"Dùng security-auditor để review toàn bộ phần auth của tôi"
"Nhờ debugger phân tích lỗi này theo quy trình 4 bước"

---

3. Checklist bắt buộc trước khi AI viết code

Trước khi thực hiện bất kỳ tác vụ code hoặc thiết kế nào, AI **bắt buộc** phải hoàn thành checklist sau:

Bước	Kiểm tra	Nếu chưa làm
1	Đã xác định đúng Agent cho lĩnh vực này chưa?	→ Dừng lại. Phân tích lại lĩnh vực yêu cầu.
2	Đã đọc file cấu hình của Agent chưa?	→ Dừng lại. Mở và đọc `.agent/agents/{agent}.md`
3	Đã thông báo Agent đang dùng cho người dùng chưa?	→ Dừng lại. Thêm thông báo `◫ Đang áp dụng @[agent]...`
4	Đã tải các Skills cần thiết từ frontmatter chưa?	→ Dừng lại. Kiểm tra trường `skills:` và đọc chúng.

---

4. Danh sách đầy đủ 20 Agents

Nhóm Quản lý & Điều phối

Agent	Chuyên môn	Skills sử dụng
`orchestrator`	Điều phối nhiều Agent làm việc song song cho tác vụ phức tạp	`parallel-agents`, `behavioral-modes`
`project-planner`	Khám phá yêu cầu, lập kế hoạch và phân rã công việc	`brainstorming`, `plan-writing`, `architecture`
`product-manager`	Yêu cầu nghiệp vụ, user stories	`plan-writing`, `brainstorming`
`product-owner`	Chiến lược sản phẩm, quản lý backlog, định nghĩa MVP	`plan-writing`, `brainstorming`

Nhóm Phát triển

Agent	Chuyên môn	Skills sử dụng
`frontend-specialist`	Giao diện Web (React, Next.js, Tailwind CSS)	`react-best-practices`, `frontend-design`, `tailwind-patterns`, `web-design-guidelines`

Agent	Chuyên môn	Skills sử dụng
backend-specialist	API, Business Logic, máy chủ	api-patterns, nodejs-best-practices, database-design
database-architect	Thiết kế schema, SQL, tối ưu truy vấn	database-design, prisma-expert
mobile-developer	iOS, Android, React Native, Flutter	mobile-design
game-developer	Logic game, cơ học trò chơi	game-development
devops-engineer	CI/CD, Docker, hạ tầng cloud	deployment-procedures, docker-expert

Nhóm Chất lượng & Bảo mật

Agent	Chuyên môn	Skills sử dụng
security-auditor	Kiểm tra bảo mật, tuân thủ OWASP	vulnerability-scanner, red-team-tactics
penetration-tester	Bảo mật tấn công (Offensive Security)	red-team-tactics
test-engineer	Chiến lược kiểm thử toàn diện	testing-patterns, tdd-workflow, webapp-testing
qa-automation-engineer	Kiểm thử E2E và CI Pipeline	webapp-testing, testing-patterns
debugger	Phân tích nguyên nhân gốc rễ của lỗi	systematic-debugging
performance-optimizer	Tối ưu hiệu năng và Core Web Vitals	performance-profiling

Nhóm Chuyên biệt

Agent	Chuyên môn	Skills sử dụng
seo-specialist	Xếp hạng tìm kiếm, khả năng hiển thị (SEO + GEO)	seo-fundamentals, geo-fundamentals
documentation-writer	Hướng dẫn sử dụng, tài liệu kỹ thuật	documentation-templates
code-archaeologist	Phân tích và tái cấu trúc code cũ (Legacy code)	clean-code, code-review-checklist
explorer-agent	Khám phá và phân tích codebase hiện có	*(không có skill cố định, dùng ngữ cảnh)*

---

5. Khi nào nên sử dụng Agent nào?

Tình huống	Agent phù hợp
Bắt đầu dự án mới, chưa biết thiết kế thế nào	project-planner → sau đó orchestrator

Tình huống	Agent phù hợp
Xây dựng trang web, component UI	`frontend-specialist`
Tạo API endpoint, kết nối database	`backend-specialist`
Thiết kế bảng DB, model dữ liệu	`database-architect`
Phát hiện lỗ hổng bảo mật	`security-auditor`
Ứng dụng bị lỗi không rõ nguyên nhân	`debugger`
Muốn viết test cho code hiện có	`test-engineer`
Trang web tải chậm, điểm Lighthouse thấp	`performance-optimizer`
Cần lên kế hoạch triển khai sản phẩm	`project-planner` + `devops-engineer`
Cần phân tích một codebase lạ	`explorer-agent`

---

6. Agents làm việc cùng nhau

Các Agent có thể **phối hợp** để giải quyết các vấn đề phức tạp đa lĩnh vực. Sử dụng Agent `orchestrator` để điều phối:

Ví dụ: Xây dựng hệ thống đăng nhập hoàn chỉnh

```
orchestrator điều phối:
├── database-architect → Thiết kế bảng users
├── backend-specialist → Viết API authentication
├── security-auditor → Review bảo mật, kiểm tra JWT
├── frontend-specialist → Xây dựng form login
└── test-engineer → Viết test cho toàn bộ flow
```

> **Lưu ý thực tế:** AI thực sự xử lý tuần tự (không song song thực sự), nhưng `orchestrator` quản lý ngữ cảnh và đảm bảo mỗi phần được thực hiện đúng bởi "góc nhìn" của Agent chuyên gia tương ứng.

Trang 3.2: Skills - Hệ thống kỹ năng và kiến thức chuyên sâu cho AI.

1. Skill là gì?

Skill là một gói kiến thức chuyên biệt chứa các nguyên lý, mẫu thiết kế (patterns) và khung ra quyết định (decision-making framework) cho một lĩnh vực cụ thể.

Skills khác gì so với template code thông thường?

Đặc điểm	Template Code Thông thường	Skill trong Antigravity Kit
Bản chất	Code mẫu cứng nhắc để copy	Nguyên lý để suy nghĩ và quyết định
Tính linh hoạt	Cố định, áp sát một pattern	Thích nghi theo ngữ cảnh dự án
Kết quả	Code giống nhau cho mọi trường hợp	Code phù hợp với yêu cầu cụ thể

> **Ví dụ:** Thay vì đưa cho bạn một template API cố định, Skill `api-patterns` dạy AI *cách quyết định* khi nào dùng REST, khi nào dùng GraphQL, và khi nào dùng tRPC dựa trên ngữ cảnh dự án của bạn.

---

2. Cách Skills được tải (Load on-demand)

Skills **không** được tải toàn bộ một lúc. Chúng được tải **theo yêu cầu** theo 3 cơ chế:

1. **Cấu hình Agent:** Mỗi Agent trong YAML frontmatter liệt kê các Skills nó được phép truy cập (trường `skills:`).
2. **Ngữ cảnh tác vụ:** Khi nhận yêu cầu, AI đọc mô tả của các Skills có liên quan và chỉ tải những Skill thực sự cần thiết.
3. **Đọc có chọn lọc:** Chỉ đọc các section cần thiết trong **SKILL.md**, không đọc toàn bộ để tối ưu hiệu suất.

```
Ví dụ:
Yêu cầu: "Tạo API endpoint cho giỏ hàng"
|
▼
backend-specialist được kích hoạt
|
▼
Tải: api-patterns + nodejs-best-practices + database-design
|
▼
Không tải: mobile-design, seo-fundamentals, red-team-tactics...
```

---

3. Cấu trúc một Skill

```
.agent/skills/react-best-practices/
├── SKILL.md # Bắt buộc — Metadata, nguyên lý chính, khung quyết định
├── sections/ # Tùy chọn — Hướng dẫn chi tiết theo từng chủ đề nhỏ
├── examples/ # Tùy chọn — Ví dụ triển khai tham khảo thực tế
├── scripts/ # Tùy chọn — Script Python/Bash có thể chạy tự động
└── assets/ # Tùy chọn — Hình ảnh, assets tham khảo
```

Phần đầu của

SKILL.md (Frontmatter YAML):

```
yaml
---
name: react-best-practices
description: React & Next.js performance optimization from Vercel Engineering.
```

```
allowed-tools: Read, Write, Edit, Bash
---
```

---

4. Danh sách đầy đủ 36 Skills theo nhóm

□□ Frontend & UI (5 Skills)

Skill	Mô tả
`react-best-practices`	Tối ưu React & Next.js — 57 quy tắc từ Vercel Engineering
`web-design-guidelines`	Kiểm toán UI web — 100+ quy tắc về Accessibility, UX, Performance
`tailwind-patterns`	Tailwind CSS v4, CSS-first config, container queries
`frontend-design`	Hệ thống thiết kế, bảng màu, typography, micro-animations
`ui-ux-pro-max`	**50 phong cách** thiết kế, **21 bảng màu**, **50 font chữ** chuyên nghiệp

⚙ Backend & API (4 Skills)

Skill	Mô tả
`api-patterns`	Thiết kế API: REST vs GraphQL vs tRPC, versioning, pagination
`nestjs-expert`	NestJS modules, DI container, decorators, Guards
`nodejs-best-practices`	Node.js async patterns, bảo mật, kiến trúc module
`python-patterns`	FastAPI, Django, async vs sync, type hints, Pydantic v2

□□ Database (2 Skills)

Skill	Mô tả
database-design	Thiết kế schema, chiến lược indexing, tránh N+1 query
prisma-expert	Prisma ORM, migrations, relations, query optimization

□□ TypeScript / JavaScript (1 Skill)

Skill	Mô tả
typescript-expert	Lập trình kiểu nâng cao (generics, conditional types), hiệu năng biên dịch

☁ Cloud & Infrastructure (3 Skills)

Skill	Mô tả
docker-expert	Containerization, Docker Compose, multi-stage builds
deployment-procedures	CI/CD, chiến lược deploy 5 giai đoạn, rollback
server-management	Quản lý process, monitoring, scaling

□□ Testing & Quality (5 Skills)

Skill	Mô tả
testing-patterns	Pyramid kiểm thử, Jest/Vitest, AAA pattern, mocking
webapp-testing	E2E testing, Playwright, audit chiến lược sâu
tdd-workflow	Vòng lặp RED-GREEN-REFACTOR, test-first development
code-review-checklist	Tiêu chuẩn review code: chất lượng, bảo mật, best practice
lint-and-validate	ESLint, Prettier, TypeScript strict mode

□□ Security (2 Skills)

Skill	Mô tả
vulnerability-scanner	OWASP 2025, Supply Chain Security, phân tích lỗ hổng

Skill	Mô tả
red-team-tactics	Chiến thuật tấn công theo MITRE ATT&CK, báo cáo penetration test

□□ Architecture & Planning (4 Skills)

Skill	Mô tả
app-builder	Tạo khung ứng dụng full-stack, chọn tech stack, phối hợp Agent
architecture	ADR (Architecture Decision Records), đánh đổi trade-off
plan-writing	Phân rã công việc, định nghĩa phụ thuộc, tiêu chí xác minh
brainstorming	Giao thức Socratic: đặt câu hỏi làm rõ trước khi triển khai

□□ Mobile (1 Skill)

Skill	Mô tả
mobile-design	Touch interaction, Mobile-first, iOS/Android conventions

□□ Game Development (1 Skill)

Skill	Mô tả
game-development	Logic game, cơ học vật lý, vòng lặp game loop

□□ SEO & Growth (2 Skills)

Skill	Mô tả
seo-fundamentals	E-E-A-T, Core Web Vitals, Google algorithm principles
geo-fundamentals	Generative Engine Optimization (tối ưu cho AI search: ChatGPT, Perplexity)

□□ Shell / CLI (2 Skills)

Skill	Mô tả
bash-linux	Linux commands, piping, error handling, scripting
powershell-windows	PowerShell operators, quản lý lỗi, pitfalls

□□ Other (6 Skills)

Skill	Mô tả
clean-code	Tiêu chuẩn code sạch toàn cục, không over-engineering
behavioral-modes	Các chế độ hoạt động AI (brainstorm, implement, debug, ship...)
parallel-agents	Mẫu điều phối đa Agent, phân chia công việc độc lập
mcp-builder	Xây dựng MCP Server, thiết kế tool & resource patterns
documentation-templates	Cấu trúc README, API docs, comment code
i18n-localization	Phát hiện hardcoded string, quản lý translations, RTL
performance-profiling	Core Web Vitals, đo lường, bundle analysis
systematic-debugging	Quy trình 4 giai đoạn: Discovery → Hypothesis → Verification → Resolution
rust-pro	Rust 1.75+, async/await, Tokio, axum, systems programming

---

5. Skills đặc biệt kèm theo công cụ thực thi

Các Skills sau không chỉ cung cấp kiến thức mà còn có **scripts Python** để Agent có thể tự động kiểm tra:

Skill	Script	Tác dụng
vulnerability-scanner	security_scan.py	Quét lỗ hổng bảo mật trong dependencies
lint-and-validate	lint_runner.py	Chạy ESLint, kiểm tra type errors

Skill	Script	Tác dụng
database-design	schema_validator.py	Kiểm tra tính nhất quán schema DB
testing-patterns	test_runner.py	Chạy bộ test tự động (Jest/Vitest)
frontend-design	ux_audit.py, accessibility_checker.py	Kiểm tra UX Laws và WCAG Accessibility
seo-fundamentals	seo_checker.py	Kiểm tra meta tags, cấu trúc heading
performance-profiling	lighthouse_audit.py, bundle_analyzer.py	Đo lường Lighthouse score và bundle size
webapp-testing	playwright_runner.py	Chạy E2E test với Playwright
mobile-design	mobile_audit.py	Kiểm tra responsive và touch targets

---

6. Cách Skills được tích hợp với Agents (ví dụ thực tế)

frontend-specialist.md:

---

skills: react-best-practices, frontend-design, tailwind-patterns, web-design-guidelines

---

Khi yêu cầu: "Tạo component card sản phẩm đẹp và dễ tiếp cận"

|

Agent tải:

├── react-best-practices → Áp dụng Server Components, tránh re-render thừa

├── frontend-design → Chọn bảng màu, spacing hợp lý

├── tailwind-patterns → Viết class Tailwind đúng chuẩn v4

└── web-design-guidelines → Đảm bảo ARIA labels, contrast ratio

Trang 3.3: Workflows - Quy trình làm việc tự động thông qua các lệnh gạch chéo (/).

1. Workflow là gì?

Workflow là các quy trình được định nghĩa sẵn theo từng bước để hoàn thành một tác vụ lập trình phổ biến. Mỗi Workflow đảm bảo tính **nhất quán** và **đúng chuẩn** cho dù bạn yêu cầu bao nhiêu lần.

Workflow giải quyết vấn đề gì?

Không có Workflow, kết quả AI trả về cho cùng một yêu cầu có thể khác nhau mỗi lần. Với Workflow, AI luôn đi theo đúng quy trình đã được định nghĩa — đảm bảo không bỏ sót bước nào.

---

2. Cách kích hoạt và sử dụng

Cú pháp cơ bản

```
/tên-workflow [mô tả yêu cầu]
```

Ví dụ thực tế

```
bash
/brainstorm hệ thống thanh toán cho SaaS app
/plan trang e-commerce với giỏ hàng và checkout
/create ứng dụng todo có dark mode và đồng bộ real-time
```

```
/debug form submit không hoạt động, không báo lỗi gì
/test src/services/auth.ts
/ui-ux-pro-max landing page cho startup fintech, tông màu xanh navy
```

---

3. Tính năng Turbo (`// turbo`)

Một số bước trong Workflow được đánh dấu `// turbo`. Khi gặp dấu này:

- AI có thể **tự động thực thi** lệnh terminal an toàn mà không cần dừng lại hỏi bạn.
- Nếu toàn bộ Workflow có nhãn `// turbo-all`: mọi lệnh terminal trong Workflow đều tự chạy.

> **Lợi ích:** Tăng tốc độ đáng kể cho các tác vụ lặp đi lặp lại như chạy test, lint check, hay preview server.

---

4. Danh sách 11 Workflows và hướng dẫn chi tiết

`/brainstorm` — Lên ý tưởng có cấu trúc

Mục đích: Khám phá ít nhất 3 phương án tiếp cận khác nhau cho một vấn đề trước khi viết code. AI sẽ phân tích ưu/nhược điểm và mức độ nỗ lực của từng phương án, sau đó đưa ra khuyến nghị.

Khi nào dùng:

- Bắt đầu một tính năng mới và chưa chắc về hướng triển khai.
- Cần so sánh các công nghệ hoặc kiến trúc khác nhau.
- Muốn đảm bảo đã cân nhắc đủ các lựa chọn trước khi commit.

Ví dụ:

```
/brainstorm chiến lược quản lý state cho ứng dụng có 50+ màn hình
/brainstorm lựa chọn database: PostgreSQL vs MongoDB cho hệ thống IoT
```

Lưu ý quan trọng: Chỉ đưa ra ý tưởng, **chưa viết code** ở bước này. Sau khi chọn phương án xong mới dùng `/create` hoặc `/enhance` để triển khai.

`/plan` — Lập kế hoạch dự án

Mục đích: Kích hoạt Agent `project-planner` để tạo ra một file kế hoạch chi tiết (`PLAN-{tên}.md`). **Không viết code** trong quá trình này.

Quy trình:

1. Bạn mô tả dự án hoặc tính năng.
2. AI đặt câu hỏi làm rõ (Socratic Questions) về yêu cầu, ràng buộc, tech stack.
3. Bạn trả lời các câu hỏi.
4. AI tạo file `docs/PLAN-tên-dự-án.md` với danh sách tác vụ và phân công Agent.

Ví dụ:

```
/plan hệ thống quản lý nhân sự với phân quyền RBAC
/plan tích hợp cổng thanh toán VNPay vào ứng dụng Next.js
```

`/create` — Tạo ứng dụng/tính năng mới

Mục đích: Trình thuật sĩ tạo ứng dụng hoàn chỉnh. Từ mô tả yêu cầu đến code chạy được.

Quy trình:

1. Bạn mô tả ứng dụng cần tạo.
2. AI phân tích và đề xuất tech stack.
3. Bạn xác nhận (Y) để tiến hành.
4. Các Agent chuyên gia triển khai từng phần (DB → Backend → Frontend → Tests).

Ví dụ:

```
/create blog cá nhân với Next.js, MDX và hệ thống comments
/create REST API cho ứng dụng quản lý thư viện sách
```

`/enhance` — Cải tiến code hiện có

Mục đích: Thêm tính năng mới vào codebase hiện tại một cách "phẫu thuật" — không làm hỏng code đang chạy.

Khác `/create` ở điểm nào?

- `/create` → Xây từ đầu (greenfield).
- `/enhance` → Sửa chữa và bổ sung vào hệ thống đang chạy (brownfield).

Quy trình:

1. AI phân tích codebase hiện có (đọc các file liên quan).
2. Lập kế hoạch thay đổi tối thiểu cần thiết.
3. Triển khai tính năng mới cùng với bài test tương ứng.
4. Chạy kiểm tra để đảm bảo không có regression.

Ví dụ:

```
/enhance thêm tính năng dark mode toggle vào ứng dụng
/enhance tích hợp real-time notifications dùng WebSocket
```

`/debug` — Sửa lỗi có hệ thống

Mục đích: Kích hoạt Agent `debugger` để điều tra lỗi theo giao thức 4 giai đoạn nghiêm ngặt.

4 giai đoạn:

1. **Discovery (Khám phá):** Thu thập log, báo cáo lỗi, các bước tái hiện.
2. **Hypothesis (Giả thuyết):** Đưa ra danh sách nguyên nhân tiềm năng, sắp xếp theo khả năng xảy ra.
3. **Verification (Xác minh):** Kiểm tra từng giả thuyết bằng bằng chứng cụ thể (không đoán mò).
4. **Resolution (Giải quyết):** Áp dụng bản sửa lỗi và thêm regression test.

Ví dụ:

```
/debug API trả về 403 dù đã đăng nhập đúng
/debug animation bị giật trên mobile Safari
```

`/test` — Tạo và chạy kiểm thử

Mục đích: Tự động hóa toàn bộ quy trình kiểm thử.

Các chức năng:

```
bash
/test # Chạy toàn bộ test suite hiện có
/test src/auth.ts # Tạo test cases cho file cụ thể (ở đây là auth.ts)
/test coverage # Báo cáo độ bao phủ (coverage report)
```

Kết quả trả về: File test mới (`tests/auth.test.ts`), số lượng test cases, tỉ lệ coverage.

`/ui-ux-pro-max` — Thiết kế UI nâng cao

Mục đích: Tạo landing page hoặc giao diện chất lượng cao theo phong cách cụ thể trong thời gian ngắn.

Tích hợp với NextLevelBuilder: Bạn có thể chọn phong cách từ thư viện tại nextlevelbuilder.io và dán vào lệnh.

Kết quả trả về:

- HTML semantic chuẩn SEO.
- Components có thể tái sử dụng.
- Micro-animations và responsive design.

Ví dụ:

```
/ui-ux-pro-max tạo landing page SaaS với glassmorphism cards, pricing 3 tới, testimonials
/ui-ux-pro-max portfolio cá nhân cho designer, phong cách minimalist với dark mode
```

`/deploy` — Triển khai ứng dụng

Mục đích: Quy trình 5 giai đoạn để triển khai an toàn lên môi trường production.

5 giai đoạn: Pre-flight checks → Build → Test → Deploy → Verify.

Ví dụ:

```
/deploy lên Vercel
/deploy lên VPS với Nginx và PM2
```

`/preview` — Quản lý server xem trước

Mục đích: Khởi động, dừng hoặc kiểm tra trạng thái server phát triển cục bộ.

```
bash
/preview start # Khởi động server (tự nhận diện Next.js, Vite...)
/preview stop # Dừng server
/preview # Kiểm tra trạng thái hiện tại
```

`/status` — Xem tổng quan dự án

Mục đích: Hiển thị bảng điều khiển (dashboard) tổng thể: thông tin dự án, tech stack, tiến độ các Agent, trạng thái server.

```
bash
/status
```

`/orchestrate` — Điều phối đa Agent

Mục đích: Kích hoạt `orchestrator` để phối hợp nhiều Agent chuyên gia cùng giải quyết một vấn đề lớn, phức tạp đòi hỏi nhiều góc nhìn chuyên môn.

Khi nào dùng: Tác vụ yêu cầu đồng thời Frontend + Backend + Security + QA.

Ví dụ:

```
/orchestrate build toàn bộ hệ thống authentication (DB + API + UI + Tests + Security review)
```

---

5. Tạo Workflow tùy chỉnh của riêng bạn

Bất kỳ ai cũng có thể thêm Workflow riêng phù hợp với quy trình nội bộ của đội:

Bước 1: Tạo file trong `.agent/workflows/`:

```
.agent/workflows/code-review.md
```

Bước 2: Viết nội dung theo format:

```
markdown
---
description: Tiến hành code review theo tiêu chuẩn nội bộ của đội
---



# Code Review Workflow



1. Đọc toàn bộ diff thay đổi
// turbo
2. Kiểm tra security issues
3. Kiểm tra performance implications
4. Kiểm tra code style theo team conventions
5. Tạo báo cáo review với mức ưu tiên (critical/major/minor)
```

Bước 3: Gọi bằng lệnh:

```
/code-review
```

---

6. Tóm tắt nhanh — Chọn Workflow nào?

Tình huống	Workflow

Chưa biết nên làm theo hướng nào	`/brainstorm`
Cần kế hoạch trước khi code	`/plan`
Xây dự án/tính năng mới từ đầu	`/create`
Thêm tính năng vào dự án hiện có	`/enhance`
Cần giao diện đẹp nhanh	`/ui-ux-pro-max`
Ứng dụng bị lỗi không rõ nguyên nhân	`/debug`
Muốn kiểm thử code	`/test`
Sẵn sàng đưa lên production	`/deploy`
Muốn xem app đang chạy	`/preview start`
Muốn biết dự án đang ở đâu	`/status`
Tác vụ cần nhiều chuyên gia cùng lúc	`/orchestrate`