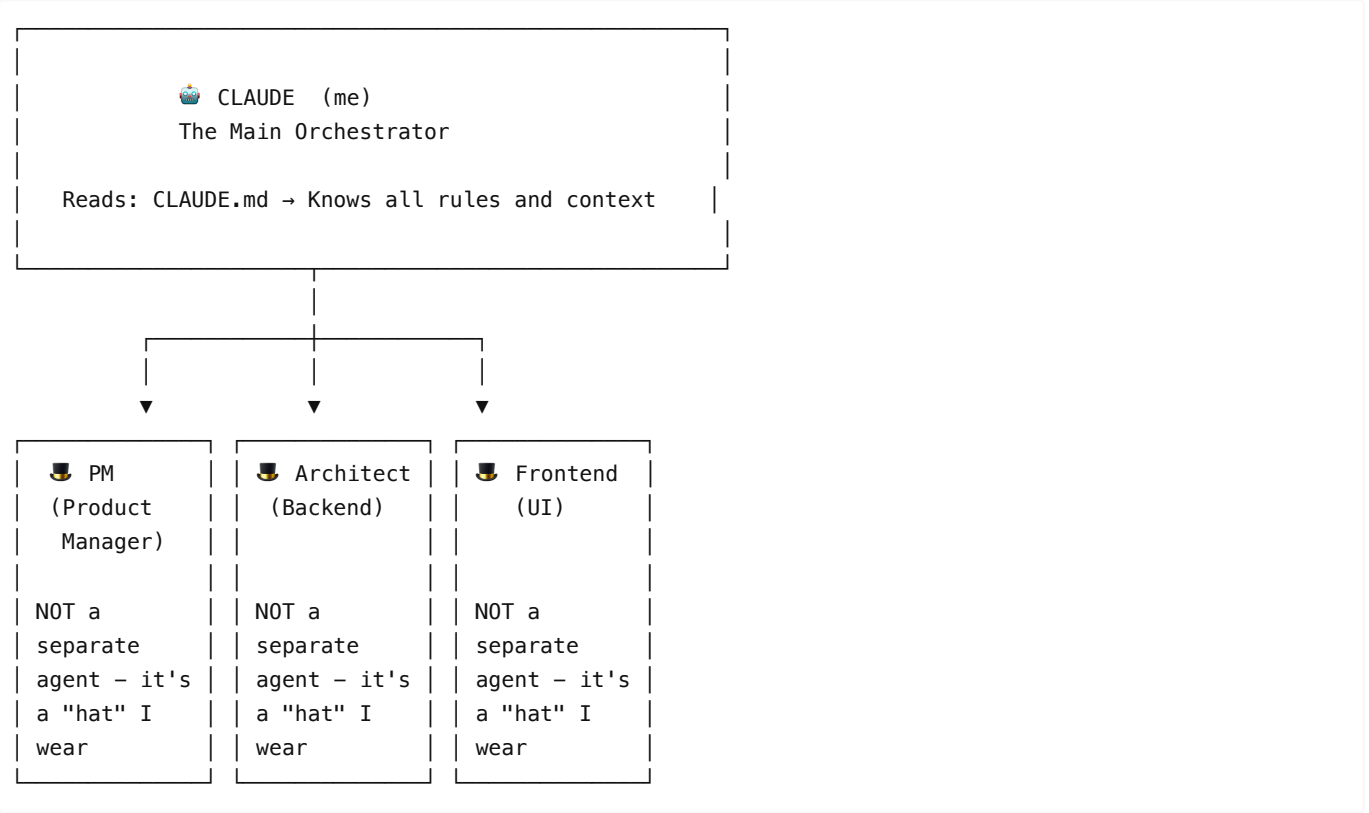
PRD-Engine System — Complete Architecture

> *Visual guide explaining how PRD-Engine works behind the scenes. Version: 2.1.0 | Date: February 2026*

🔗 How Everything Connects



The Concept: Claude is a single brain that switches "hats". When it's PM — it asks business questions. When it's Architect — technical questions. When it's Frontend — UI/UX questions. But always the same Claude.

🎩 PM / Architect / Frontend = "Hats", Not Separate Agents

Key Understanding: When I'm "PM", I'm not running another agent — I simply read the PM's SKILL.md and behave accordingly.

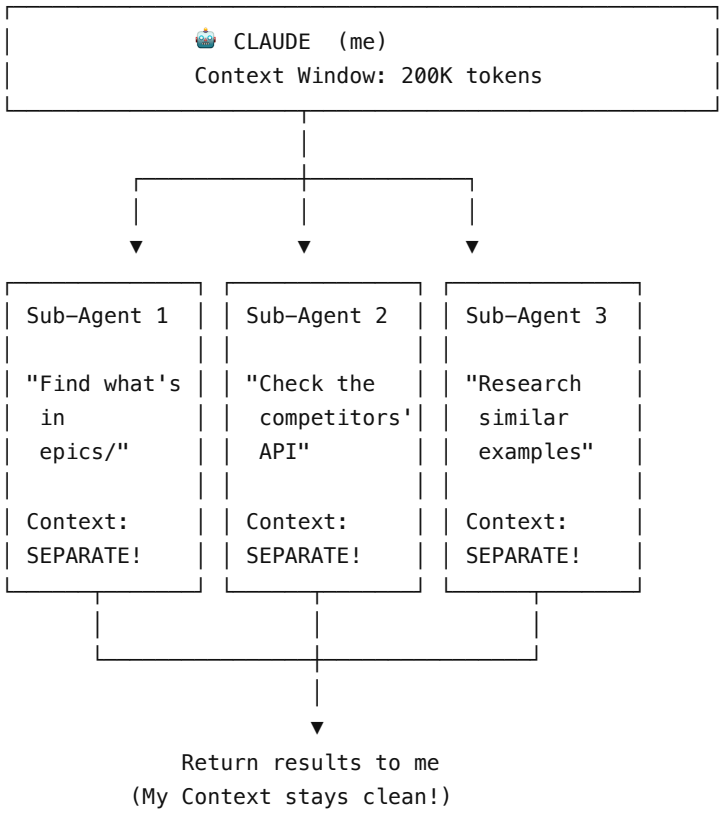
```
User: "Let's spec a Login system"
    │
    ▼
Me (Claude): reads product-manager/SKILL.md
    │
    ▼
Asks 9 business questions: "I'm now PM"
    │
    ▼
🎩 "Switching to Architect!"
Me (Claude): reads architect/SKILL.md
    │
    ▼
Asks 8 technical questions: "I'm now Architect"
    │
    ▼
🎩 "Switching to Frontend!"
```

```
Me (Claude): reads frontend/SKILL.md
    |
    ▼
Asks 11+1 UI questions: "I'm now Frontend"
    |
    ▼
And so on...
```

Why is this important? Because all the context stays with me. I remember what PM asked, and I can incorporate that when Architect asks. There's no "data transfer" between agents — everything is with me.

🔧 Sub-Agents (Task Tool) = Something Completely Different!

A Sub-Agent IS a separate agent that runs in parallel:



Why is this good?

- My main Context doesn't fill up with research "junk"
- Sub-Agents can run in parallel
- Each one gets its own clean Context

Iron Rule: Every sub-agent must have `model: "sonnet"` — never Haiku, never Opus!

4 Scenarios that require a Sub-Agent:

#	Scenario	Type	What it does
1	Session start / after compact	`Explore`	Reads all files + returns 60-line summary
2	Reading source document (DOC_SOURCE)	`Explore`	Reads large document + returns requirements summary

3	Checking links between epics	`Explore`	Checks existing epics + identifies overlaps
4	Cross-Agent Review	`general-purpose`	Analyzes contradictions between 3 Agents

---

📄 DOC_SOURCE = The Source Document

At the start of every new specification, the user provides a link to their requirements document (Google Doc, Notion, etc.).

```
┌─────────────────────────────────────────────────┐
│                                                 │
│  📄 DOC_SOURCE = READ-ONLY!                     │
│                                                 │
│  • This is the "source of truth" for requirements │
│  • Never modify — it's what the user provided   │
│  • Never read in main context — the document could │
│    be large (hundreds of KB) and destroy the context│
│                                                 │
│  📝 OUTPUT = epics/ files only!                 │
│  • .claude/memory/epics/XX-name.md              │
│  • Written only after Diff approval             │
│                                                 │
└─────────────────────────────────────────────────┘
```

The Flow:

```
DOC_SOURCE ──[Read via Sub-Agent ONLY!]──► Sub-Agent reads the requirements
                                                    │
                                                    ▼
                                          Returns requirements summary
                                                    │
                                                    ▼
                                       Claude asks detailed questions
                                                    │
                                                    ▼
epics/XX.md ◄──[Write after Diff approval]── Writes detailed specification
```

3 Options at the start (AskUserQuestionTool):

1. **"I have a link"** → Paste it and we'll start
2. **"I don't have one yet"** → Go create a document and come back
3. **"Spec from scratch"** → We'll ask more detailed questions without a document

---

📚 lessons.md = Long-term Memory

```
┌─────────────────────────────────────────────────┐
│  .claude/memory/lessons.md                      │
├─────────────────────────────────────────────────┤
│                                                 │
│  ## Lesson 1: Question Format                   │
│  • Mistake: Asked without options               │
│  • Fix: Always use AskUserQuestionTool          │
│    with numbers                                 │
│                                                 │
```

```
┌──────────────────────────────────────────────┐
│  ## Lesson 2: Google Doc                     │
│  • Mistake: Wrote to Doc before approval     │
│  • Fix: Always show Diff                     │
│    and wait for approval                     │
│                                              │
│  ## Lesson 3: User prefers...                │
│  • Hebrew in conversation, English for tech terms  │
│  • Tables with examples                      │
│                                              │
└──────────────────────────────────────────────┘
```

How it works:

1. You correct me → I identify it as a lesson
2. I add it to lessons.md
3. Every new Session → Sub-Agent reads lessons.md
4. Don't repeat the same mistake!

The Loop:

```
Mistake → Pattern identification → Write rule → Check → Improve
   ▲                                                    │
   └────────────────────────────────────────────────────┘
```

💾 checkpoint.json = Continuous Saving

The Problem: Claude Code may /compact at any moment, or the Session may close. Without saving — everything is lost!

The Solution: A small JSON file (~200 tokens) saved after every significant answer:

```
┌──────────────────────────────────────────────┐
│  .claude/memory/checkpoint.json              │
├──────────────────────────────────────────────┤
│                                              │
│  {                                           │
│    "timestamp": "2026-02-10T14:30:00",       │
│    "epic": "user-authentication",            │
│    "agent": "architect",                     │
│    "question_number": 5,                     │
│    "completed": ["Q1: Entities", "Q2: Relations",  │
│                  "Q3: APIs", "Q4: Validations"],   │
│    "pending": "Q5: Error Codes",             │
│    "doc_source": "https://docs.google.com/...",    │
│    "notes": "User wants JWT, not sessions"   │
│  }                                           │
│                                              │
└──────────────────────────────────────────────┘
```

When to save what?

Event	checkpoint	epic file	prd-index
Significant answer	✅	✅	❌
End of Agent phase	✅	✅	❌
Epic 100% complete	✅	✅	✅

50% Context	✅ + alert	✅	❌
PreCompact	✅	—	—

At 50% Context:

```
🛑 Stop! → 💾 Save everything → 🔄 Suggest /compact or new Session
```

📊 prd-index.json = Map of the Existing PRD

The Problem: Every Agent needs to know all the PRD that was already written to ask smart questions and identify links — but without burning all the Context!

The Solution: A compact JSON file (~500 tokens) containing the "map" of everything that exists:

```
┌──────────────────────────────────────────────────┐
│  .claude/memory/prd-index.json                   │
├──────────────────────────────────────────────────┤
│                                                  │
│  {                                               │
│    "epics_completed": 2,                         │
│    "epics": {                                    │
│      "user-auth": {                              │
│        "entities": ["User", "Role", "Session"],  │
│        "apis": ["/api/auth/login", "/api/users"],│
│        "relations": ["User->Role (N:N)"]         │
│      },                                          │
│      "product-catalog": {                        │
│        "entities": ["Product", "Category"],      │
│        "apis": ["/api/products"],                │
│        "relations": ["Product->Category (N:1)"]  │
│      }                                           │
│    },                                            │
│    "global_entities": ["User", "Role", "Product"],│
│    "cross_epic_relations": [                     │
│      "Product->User (created_by)"                │
│    ]                                             │
│  }                                               │
│                                                  │
└──────────────────────────────────────────────────┘
```

How an Agent uses this:

```
┌──────────────────────────────────────────────────┐
│  🎯 Session starts                               │
├──────────────────────────────────────────────────┤
│                                                  │
│  1. Reads prd-index.json                         │
│     "There are 2 epics: user-auth, product-catalog"│
│                                                  │
│  2. User: "Let's spec an order system"           │
│                                                  │
│  3. Architect asks a smart question:             │
│     "I see we have a Product and User Entity.    │
│      Is the order linked to a user and specific  │
│      products?"                                  │
```

```
│                                                          │
│  4. Automatic link detection:                            │
│     "⚠️ This epic will affect: user-auth,                │
│      product-catalog"                                    │
│                                                          │
└──────────────────────────────────────────────────────────┘
```

The Benefits:

- ~500 tokens instead of reading the entire PRD (thousands of tokens)
- Every Agent knows the system "map"
- Smart questions based on what exists
- Automatic link detection between epics

🔍 Cross-Review = Quality Control (7 Checks)

Before writing to an epic file, 7 mandatory checks are performed:

```
┌──────────────────────────────────────────────────────────┐
│                    🔍 Cross-Review                       │
├──────────────────────────────────────────────────────────┤
│                                                          │
│  1. ✅ PM Review                                         │
│     Is every User Story covered in the spec?             │
│                                                          │
│  2. ✅ Architect Review                                  │
│     Technical consistency — do Entities match APIs?      │
│                                                          │
│  3. ✅ Frontend Review                                   │
│     Every endpoint appears in UI? Every state            │
│     handled?                                             │
│                                                          │
│  4. ✅ Analytics Events (minimum 12!)                    │
│     Every significant action documented — page view,     │
│     form submit, click, error                            │
│                                                          │
│  5. ✅ SEO Metadata                                      │
│     Every public page — title, description, og:tags      │
│                                                          │
│  6. ✅ i18n Consistency                                  │
│     Every message in both languages                      │
│                                                          │
│  7. ✅ Deferred Documentation                            │
│     What was deferred — documented with reason           │
│                                                          │
├──────────────────────────────────────────────────────────┤
│  📊 Summary: X gaps found / Y closed / Z deferred        │
│                                                          │
│  ⚠️ Contradictions? → AskUserQuestionTool → Fix          │
│  ✅ All good? → Diff → Approval → Write to file          │
│                                                          │
└──────────────────────────────────────────────────────────┘
```

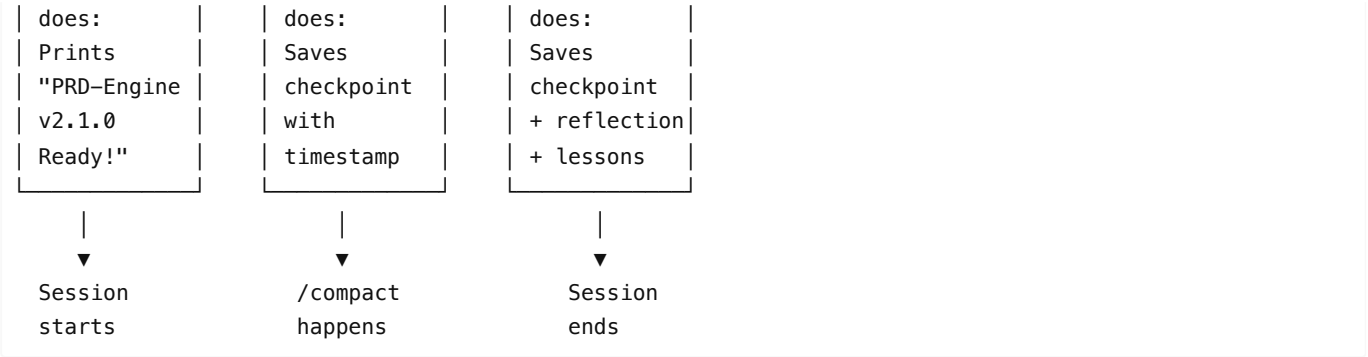
🏗️ File Structure

```
.claude/
├── CLAUDE.md                    ← 🧠 The "brain" — 18 Iron Rules
├── settings.json                ← 3 Hooks (SessionStart, PreCompact, Stop)
├── settings.local.json          ← WebFetch permissions
│
├── scripts/
│   └── statusline.sh            ← Context percentage in colors in CLI
│
├── memory/                      ← 💾 Persistent memory
│   ├── checkpoint.json          ← ~200 tokens — where we stopped
│   ├── prd-index.json           ← ~500 tokens — PRD map
│   ├── lessons.md               ← Lessons learned
│   ├── session-init.json        ← Sub-agent verification
│   └── epics/                   ← 📄 Dev-ready specifications
│       ├── 01-user-auth.md
│       └── 02-product-catalog.md
│
└── skills/prd-engine/           ← ⚙️ Skill Engine
    ├── SKILL.md                 ← Main Orchestrator
    ├── config/
    │   └── workflow.json        ← Workflow settings (v2.1.0)
    ├── agents/
    │   ├── product-manager/
    │   │   └── SKILL.md         ← 9 business questions
    │   ├── architect/
    │   │   └── SKILL.md         ← 8 technical questions
    │   └── frontend/
    │       └── SKILL.md         ← 11+1 UI/UX questions
    ├── rules/
    │   ├── INDEX.md
    │   ├── 01-questions-format.md
    │   ├── 02-review-crosscheck.md
    │   └── 03-reflection.md
    ├── templates/
    │   ├── epic-template.md     ← Epic template (Parts A-D)
    │   ├── checkpoint.json
    │   ├── prd-index.json
    │   ├── landing-page-guide.md  ← Landing Page guide (10 sections)
    │   └── landing-page-anatomy.jpg
    └── hooks/
        ├── startup.sh           ← SessionStart
        ├── pre-compact.sh       ← PreCompact
        └── auto-checkpoint.sh   ← Stop (Reflection)
```

⚡ Hooks = Automation

3 Hooks that run automatically — without the user needing to do anything:

```
┌──────────────┐    ┌──────────────┐    ┌──────────────┐
│ SessionStart │    │ PreCompact   │    │    Stop      │
│              │    │              │    │              │
│ startup.sh   │    │pre-compact   │    │auto-         │
│              │    │   .sh        │    │checkpoint.sh │
│              │    │              │    │              │
│ What it      │    │ What it      │    │ What it      │
```



Why are Hooks important?

- **SessionStart:** Announces that the engine is active
- **PreCompact:** Saves state before compact deletes history
- **Stop:** Saves everything remaining + writes reflection for future improvement

---

🔒 18 Iron Rules — Quick Overview

#	Rule	Summary
0	Sub-Agents	Sonnet only + DOC_SOURCE only via sub-agent
1	Structured Questions	AskUserQuestionTool + options + 🎯 implications
2	Modularity	500 lines maximum, each Agent in their domain
3	Continuous Saving	checkpoint after every answer + saving matrix
4	Zero Open Ends	Every detail defined, every error message written
5	Plan Mode	Required before significant tasks
6	Cross-Review	7 mandatory checks before writing to file
7	Sweet Spot	🟢 MVP / 🔵 Future / ❓ User Decides
8	Epics = Dev-Ready	Every epic file = ready for development
9	Diff Before Write	Show changes + explicit approval
10	Holistic Flexibility	SKILL questions = starting point, Agent goes deeper
11	Improvement Loop	lessons.md — learns from mistakes
12	Read DOC_SOURCE	Every Session via sub-agent
13	Load PRD Context	prd-index.json for smart questions
14	Hat Switching	🎩 announcement + read SKILL.md
15	Analytics	Minimum 12 events per epic
16	Design System	colors + typography + spacing required
17	Reflection	At end of every session — update lessons.md

🎯 Sweet Spot = MVP vs Future

In Architect, every technical question is separated into 3 levels:

```
┌──────────────────────────────────────────────┐
│                                              │
│  🟢 Required for MVP                         │
│  Without this the epic doesn't work          │
│  Example: "User Entity with email + password"│
│                                              │
├──────────────────────────────────────────────┤
│                                              │
│  🔵 Recommended for Future                   │
│  Will save refactoring later                 │
│  Example: "Add role_history field to document│
│   permission changes"                        │
│                                              │
├──────────────────────────────────────────────┤
│                                              │
│  ❓ User Decides                             │
│  There are 2 ways — you choose               │
│  Example: "JWT or Session-based? Both work"  │
│                                              │
└──────────────────────────────────────────────┘
```

📦 Epic = Finished Product

Every file in `epics/` is a standalone specification document with 4 parts + summary:

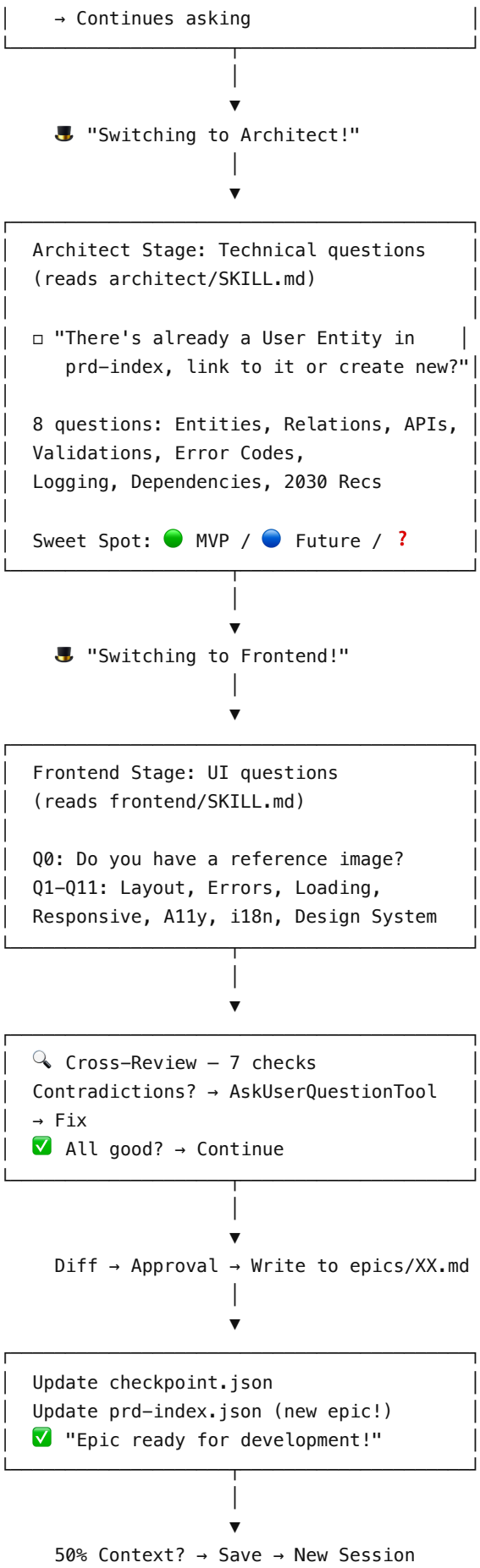
```
┌──────────────────────────────────────────────┐
│  📄 epics/03-order-system.md                 │
├──────────────────────────────────────────────┤
│                                              │
│  Part A — Business Requirements (PM)         │
│  ├─ User Stories (2-4)                       │
│  ├─ Acceptance Criteria (8-12, categorized)  │
│  ├─ User Roles Table                         │
│  ├─ Edge Cases / Funnel                      │
│  ├─ KPIs Tables                              │
│  ├─ 2030 Recommendations                     │
│  └─ Key Decisions                            │
│                                              │
│  Part B — Technical Architecture (Architect) │
│  ├─ Entities (fields, indexes, rules, edge cases)  │
│  ├─ Relations (with FK behavior)             │
│  ├─ API Endpoints (Auth + Rate Limit)        │
│  ├─ Validations (HE + EN)                    │
│  ├─ Error Codes (7 categories)               │
│  ├─ Logging & Monitoring                     │
│  ├─ Dependencies (3 categories)              │
│  ├─ 2030 Recommendations                     │
│  └─ Key Decisions                            │
│                                              │
│  Part C — Frontend Specification             │
│  ├─ ASCII Wireframes                         │
│  ├─ Error Display — 3 Levels (Inline/Banner/Toast)  │
│  ├─ Responsive Breakpoints                   │
│  ├─ Accessibility (WCAG AA)                  │
│  ├─ i18n System                              │
```

```
│  ├─ Design System                                    │
│  ├─ 2030 Recommendations                             │
│  └─ Key Decisions                                    │
│                                                      │
│  Part D — Cross-Review (7 checks)                    │
│  ├─ Analytics Events (12+)                           │
│  ├─ SEO Metadata                                     │
│  ├─ i18n Consistency                                 │
│  └─ Review Summary                                   │
│                                                      │
│  Key Decisions (All Agents) — Unified Table          │
│                                                      │
└──────────────────────────────────────────────────────┘
```

The Benefit: User takes the file → transfers to Cursor / Claude Code / Copilot / Windsurf / Bolt → Developer starts developing directly, without additional questions.

🔄 Full Workflow — End-to-End Scenario

```
Session starts
    │
    ▼
┌──────────────────────────────────────────┐
│  Reads CLAUDE.md (Iron Rules)            │
│  Reads checkpoint.json (if exists)       │
│  Sub-Agent reads: SKILL.md + rules/      │
│  + lessons.md + prd-index.json + epics/  │
│  + DOC_SOURCE (if exists) via WebFetch   │
└──────────────────┬───────────────────────┘
                   │
                   ▼
     📊 "Found 2 existing epics, continue?"
                   │
                   ▼
   Step 0 AskUserQuestionTool (source + target)
                   │
                   ▼
┌──────────────────────────────────────────┐
│  PM Stage: Business questions            │
│  (reads product-manager/SKILL.md)        │
│                                          │
│  9 questions:                            │
│  Q1: Who is the user?                    │
│  Q2: What problem does it solve?         │
│  Q3: User Stories                        │
│  Q4: Acceptance Criteria                 │
│  Q5: Priority                            │
│  Q6: Cancel/Error + Funnel               │
│  Q7: User Roles Table                    │
│  Q8: KPIs                                │
│  Q9: 2030 Recommendations                │
│                                          │
│  Need to research something?             │
│    → Task Tool = Sub-Agent               │
│    → Returns results                     │
```


→ Continues asking
"Switching to Architect!"
Architect Stage: Technical questions
(reads architect/SKILL.md)
□ "There's already a User Entity in prd-index, link to it or create new?"
8 questions: Entities, Relations, APIs, Validations, Error Codes, Logging, Dependencies, 2030 Recs
Sweet Spot: MVP / Future / ?
"Switching to Frontend!"
Frontend Stage: UI questions
(reads frontend/SKILL.md)
Q0: Do you have a reference image?
Q1–Q11: Layout, Errors, Loading, Responsive, A11y, i18n, Design System
Cross-Review — 7 checks
Contradictions? → AskUserQuestionTool → Fix
All good? → Continue
Diff → Approval → Write to epics/XX.md
Update checkpoint.json
Update prd-index.json (new epic!)
"Epic ready for development!"
50% Context? → Save → New Session

🧠 Holistic Flexibility = Smart Questions

The questions in SKILL.md are a starting point, not a closed list!

```
┌─────────────────────────────────────────────────────┐
│                                                     │
│  📋 SKILL.md = Required minimum + direction         │
│  🧠 The Agent = Goes deeper as needed               │
│                                                     │
│  Example:                                           │
│                                                     │
│  PM asks (from SKILL.md): "Who is the user?"        │
│  User answers: "Store manager"                      │
│                                                     │
│  PM continues (from its intelligence):              │
│  ┌                                                  │
│  │ "Can a store manager manage more than one store?"│
│  │ "Is there a difference between internal and      │
│  │  external manager?"                              │
│  │ "Does a store manager see all employees?"        │
│  └                                                  │
│                                                     │
│  ✅ Ask required questions from SKILL.md            │
│  ✅ Add questions as needed                         │
│  ✅ Go deeper when there's ambiguity                │
│  ❌ Don't ignore required questions                 │
│  ❌ Don't ask irrelevant questions                  │
│                                                     │
└─────────────────────────────────────────────────────┘
```

4️⃣ Zero Open Ends

Every detail must be defined — not "there will be something":

❌ Not enough	✅ Sufficient
"An error message will be shown"	"Display: 'An error occurred while saving. Please try again.'"
"The button will submit"	"Click: 1) spinner, 2) POST /api/x, 3) green toast / red message"
"There will be validation"	"Name — required, min 2 chars. Email — format. Phone — 10 digits."
"The user can delete"	"popup 'Delete this?' → red button → toast 'Deleted successfully'"
