READER EDITION

Prototyping at the Speed of AI

From a customer's pain point to a working prototype, in a single session.

Andrei Manea · Founder & CEO, CloudHero
AWS Advanced Tier Partner · 2024 AWS Global Partner of the Year, Application Modernization

CloudHero

A mockup asks your customer to imagine your product.

A prototype lets them use it.

AI can help you build a prototype quickly. You still need to decide what to test.

The scarce skill is knowing what to build, and describing it clearly.

Four ways to build with AI. Each one wins somewhere.

	Chat-to-app Lovable, Bolt, v0	**AI code editors** Cursor, GitHub Copilot	**Coding agents** Claude Code	**Spec-driven** Kiro
BEST FOR	A clickable first version in minutes	Engineers who want to move faster	Big changes across a whole codebase	**Founders who want to own the spec**
YOU STEER IT WITH	Chat messages	Code and prompts	A conversation, plus any plan you write	Requirements, design and tasks
YOU HAND OVER	An app and a chat history	Code	Code, plus your own notes	Code, plus the spec it was built from
WATCH OUT FOR	Decisions buried in the chat	You need to read code	You bring the discipline	Slower to the first screen. The spec has to be precise.

Approaches discussed on 8 October 2026. These are overlapping workflows; the tools can be used in several ways.

Why I chose Kiro for this session.

SESSION CONTEXT

How to Web, 8 October 2026

SPEC FILES

Requirements, design and tasks

DOCUMENTATION

kiro.dev/docs/specs/

PRICING AND CREDITS

Check current terms with Kiro and AWS.



You can follow it without reading code

The spec sits on screen as plain documents: requirements, design, tasks.



It works like a product owner

Requirements come before any code, and you review them before it builds.



The spec is the handover

You hand over the code and the spec it came from. Production starts there.



Check the terms for your startup

Kiro plans and AWS startup offers change. Review the current pricing and eligibility before committing.

From pain point to product, in six steps.

PREPARE

BUILD AND TEST

AFTER THE YES

1
Listen
Talk to customers until you can name the pain in one paragraph.

YOU AND THE CUSTOMER

2
Specify
Write a spec like a product owner: rules, requirements, tasks. Not a chat.

YOU, WITH AI'S HELP

3
Build
An AI agent builds the MVP from the spec.

THE AGENT

4
Watch
Put it in front of the customer, and watch them use it.

THE CUSTOMER

5
Update
Change the spec, rebuild, show it again.

YOU AND THE AGENT

6
Sell
Close the deal, then build the production version from the same spec.

ENGINEERS, WITH AGENTS

THE CUSTOMER PROBLEM

Supplier contracts and certificates are spread across spreadsheets and inboxes. Renewals get noticed late, and straightforward questions turn into report requests.

WHAT WE DECIDED TO PROTOTYPE

A screen where a buyer can see what needs attention, ask about expiring contracts, and open the records behind the answer.

Otto is a simulated assistant using fictional procurement records. The demo uses fixed rules, not a live language model or customer systems.

SPEC-DRIVEN DEVELOPMENT

Write a spec, not a chat.

Like a product owner would. The spec is the source of truth, and the code is built from it, today and in production.



The rules only you know
Business rules, how the assistant behaves, how it looks

4 files



Requirements
What it must do, in plain sentences you can review

12 requirements



Design
How it's built, with every rule copied in

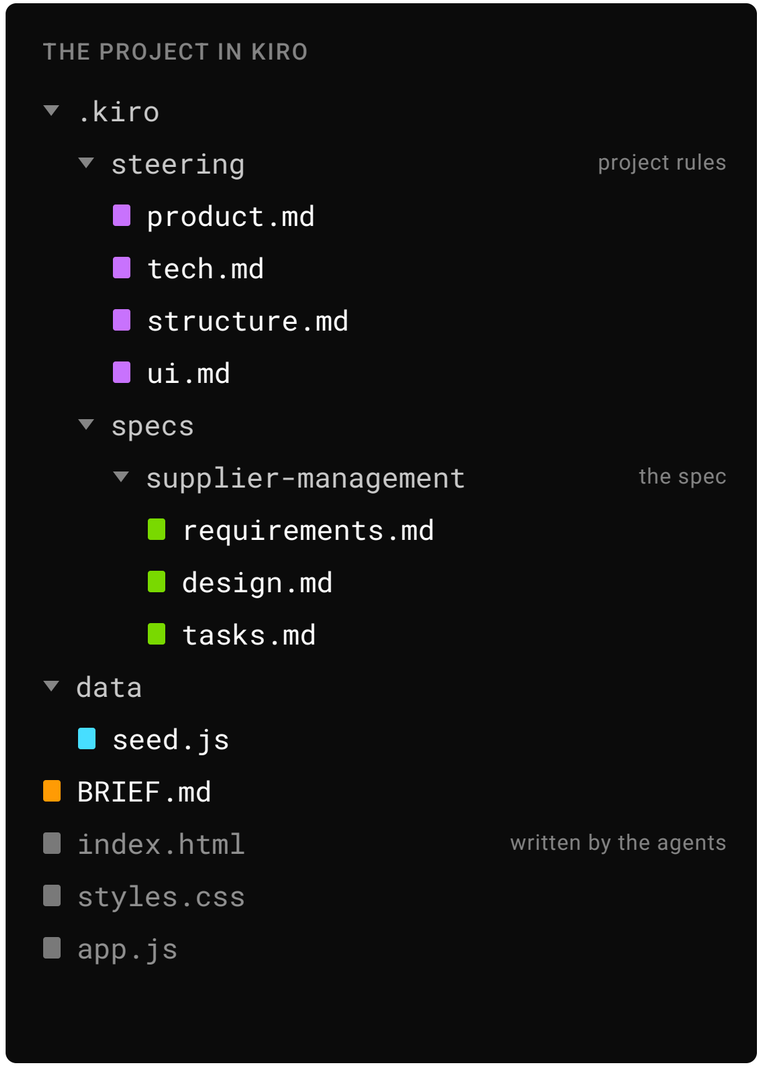
1 document



Tasks
Small steps, each linked to the requirements it covers

8 tasks

READ BY — You | Your customer's team | Your engineers | AI agents



What Kiro reads, file by file.

Everything the agents work from is a plain text file in the project. Two folders do the work.

STEERING

The project's rules

Who it's for, the business rules, the tech, the look. Four short files.

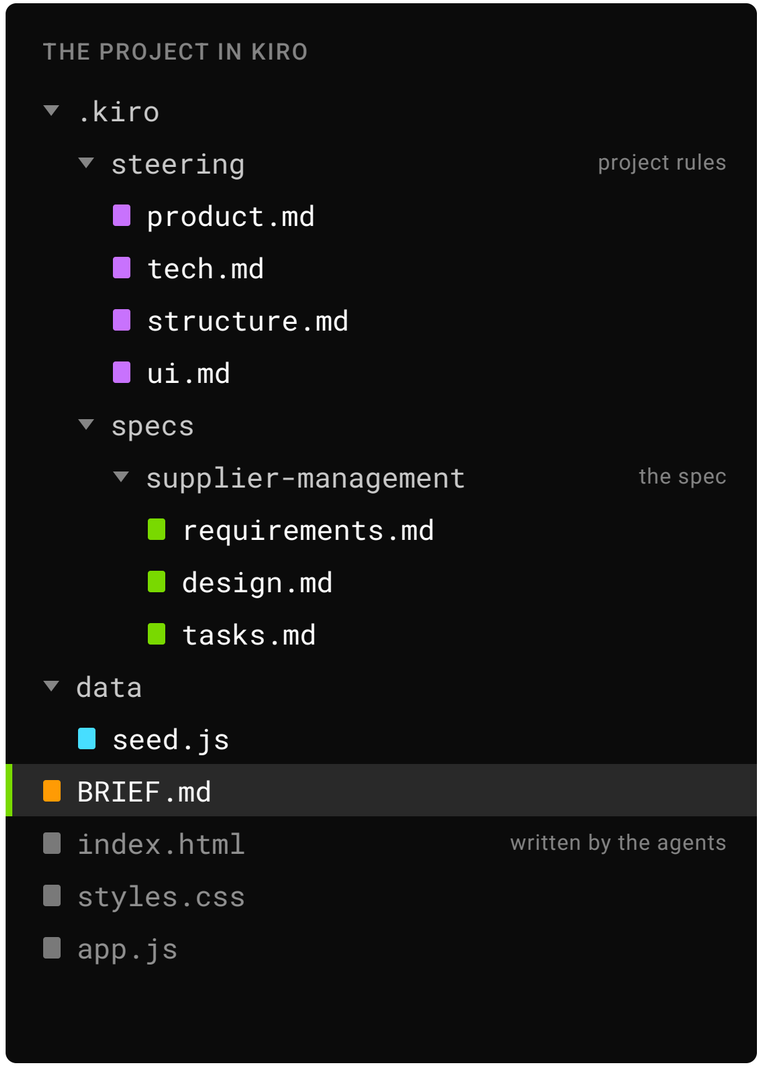
SPEC

What to build this time

Requirements, design and tasks. Kiro ticks the tasks off as it builds.

`BRIEF.md`	Where it started: the pain, in one paragraph
`data/seed.js`	The fictional data every answer is calculated from
`index.html` · `styles.css` · `app.js`	The app, written by the agents

The following pages explain each file.



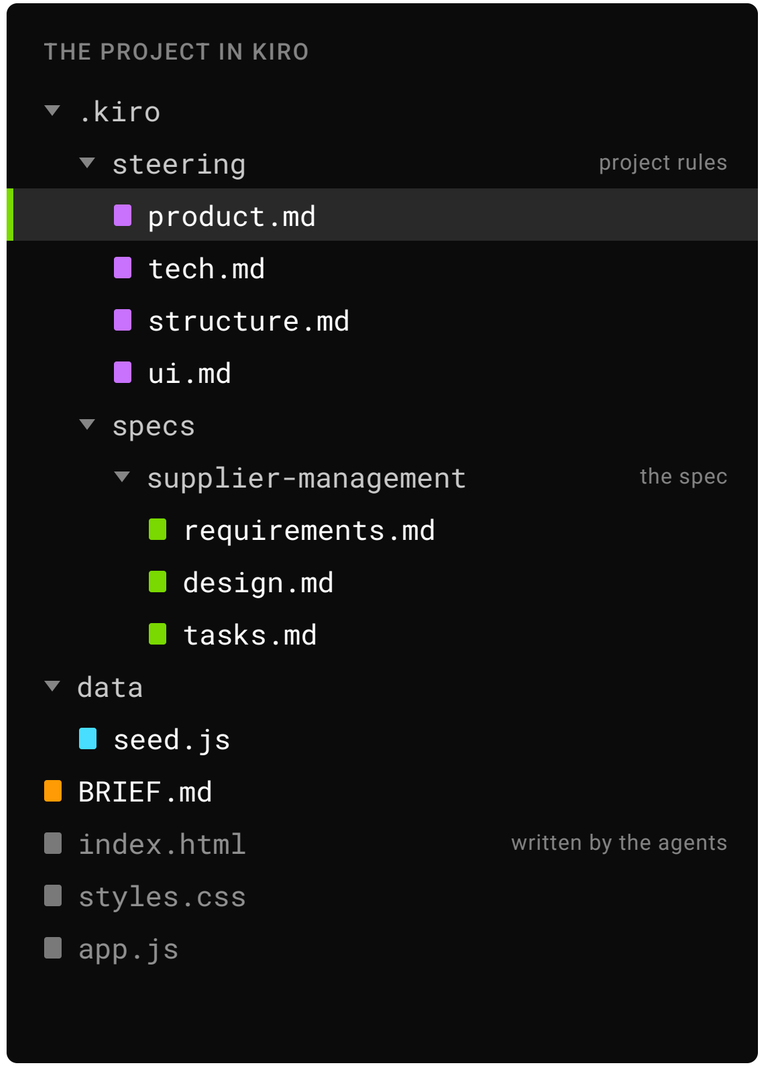
START HERE BRIEF.md

The pain, in one paragraph

Step 1, on one page. The rest of the spec was written from it.

```
1  # The brief
3  A supplier management assistant for procurement teams at mid-sized companies that
   work with 100+ suppliers. Instead of digging through spreadsheets, a buyer opens one
   screen, sees what needs attention today, and asks plain questions like "which contracts
   expire in the next 90 days?" or "what's going on with Carpatia Grid Works?". The answers
   come back as a short sentence with a table or a card, calculated from the same supplier,
   contract and certification records the buyer can open.
```

> **If you can't write this paragraph yet, you're not ready to build.**



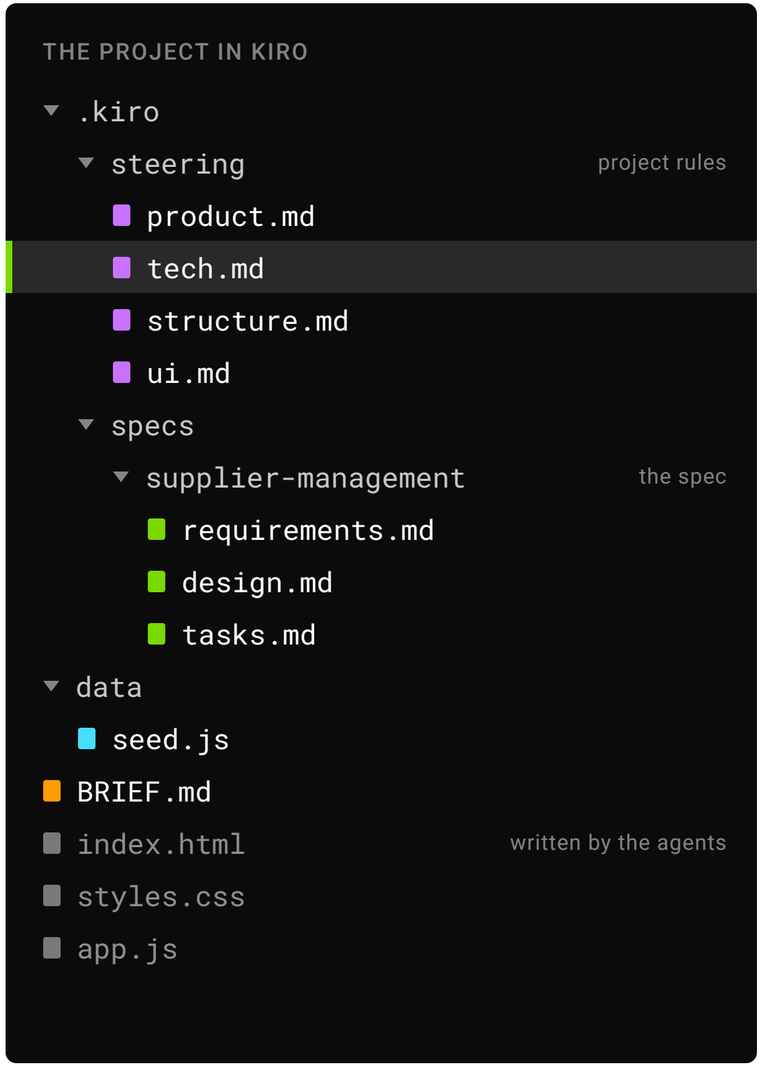
STEERING `.kiro/steering/product.md`

The rules only you know

Who it's for, what Otto answers, and the business rules. Written once, for Kiro and for people.

```
 2 inclusion: always
   ⋮
27 ## Domain rules (follow these exactly)
28 - A framework agreement sets a ceiling for a period. It is not money
   committed. Never add framework ceilings to contracted value.
31 - Today is SEED.today. Never use the computer clock. …
35 - Supplier score is 0 to 100: 40% quality, 40% delivery, 20% documents,
   rounded to the nearest whole number. Below 70 is flagged.
36 - A supplier is flagged when its status is not Active or its score is below
   70. Count each supplier once.
```

> **None of this is in any AI's training data. It's in your head, and your customer's.**



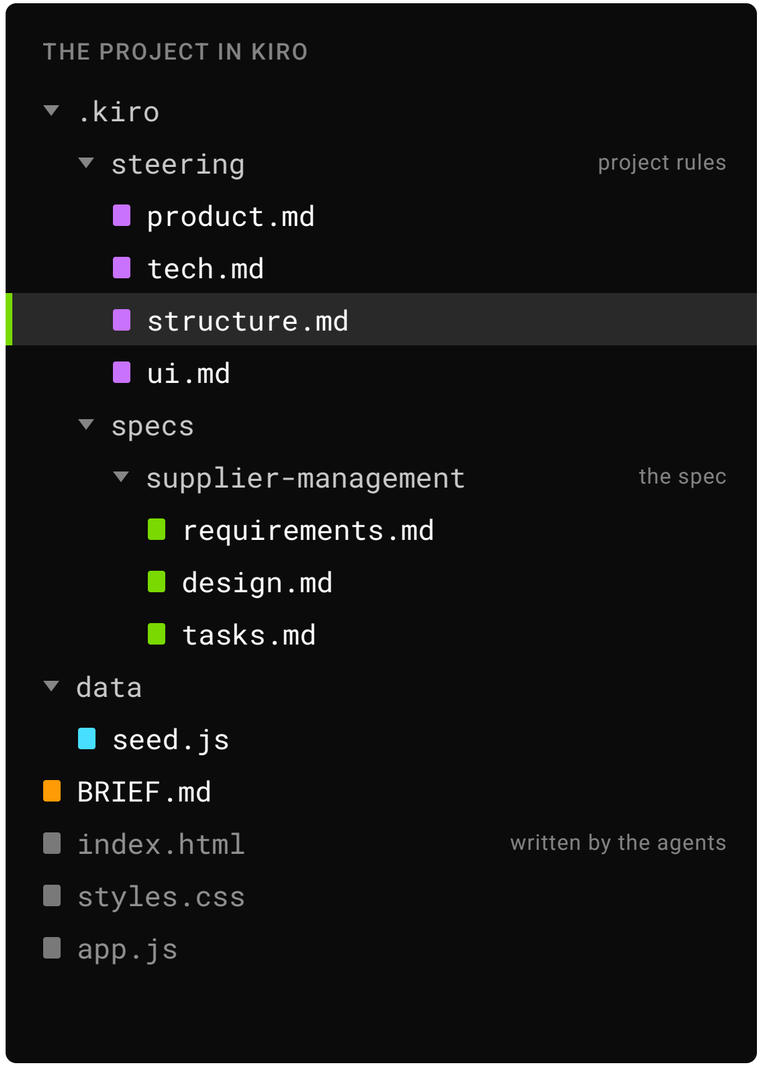
STEERING `.kiro/steering/tech.md`

How it's built and run

The technical guardrails. Here: no installs, no servers, it opens with a double-click.

```
7  - Plain HTML, CSS and JavaScript. No framework, no build step, no package
   manager, no dependencies, no network requests.
8  - It must open by double-clicking index.html (file://). …
   ⋮
17 - There is nothing to install, build or run. Do not run shell commands. …
18 - Work only inside this folder. Do not read or write files outside it.
```

> **Without guardrails, an agent builds a project. You wanted a prototype.**



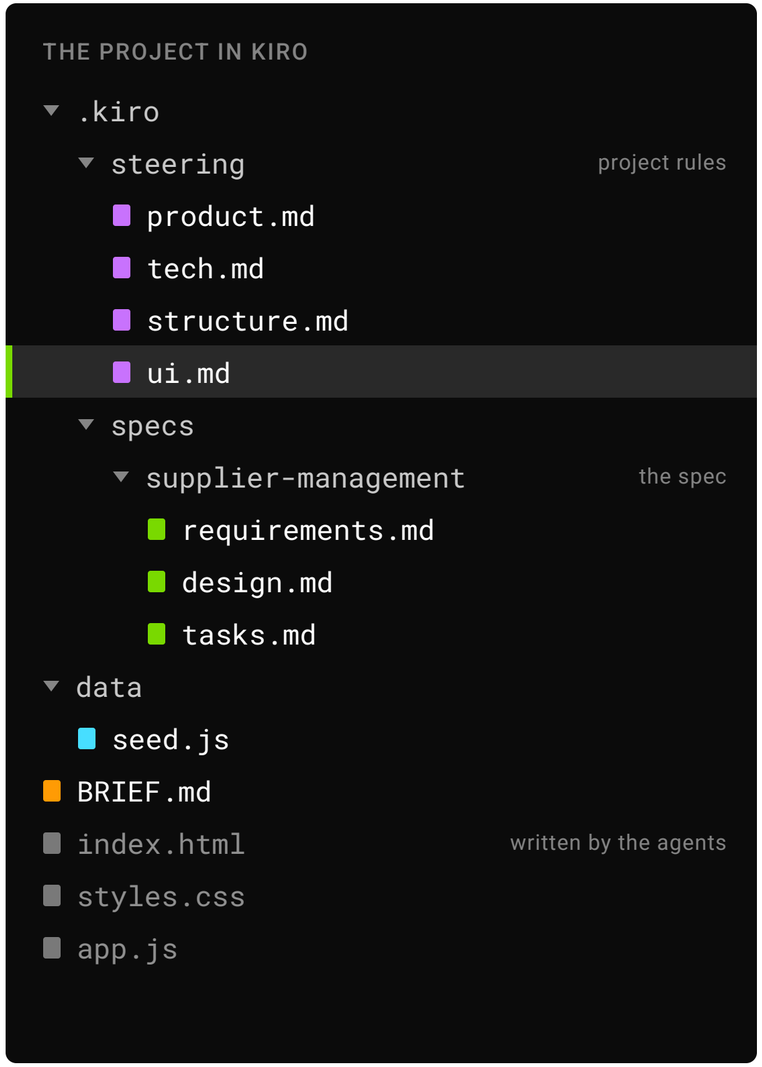
STEERING `.kiro/steering/structure.md`

Where things go

The file layout and the naming, so the code stays readable.

```
 8 index.html      shell: sidebar, top bar, <main id="root">, …
 9 styles.css      all styles
10 app.js          1. helpers  2. Otto (…)  3. views  4. router
11 data/seed.js    fictional data as window.SEED. …
   ⋮
19 - Keep it readable for a non-engineer scrolling through on a projector: short
   functions, plain names, a one-line comment above each function.
```

> **Your engineers will read this code later. Ask for plain names now.**



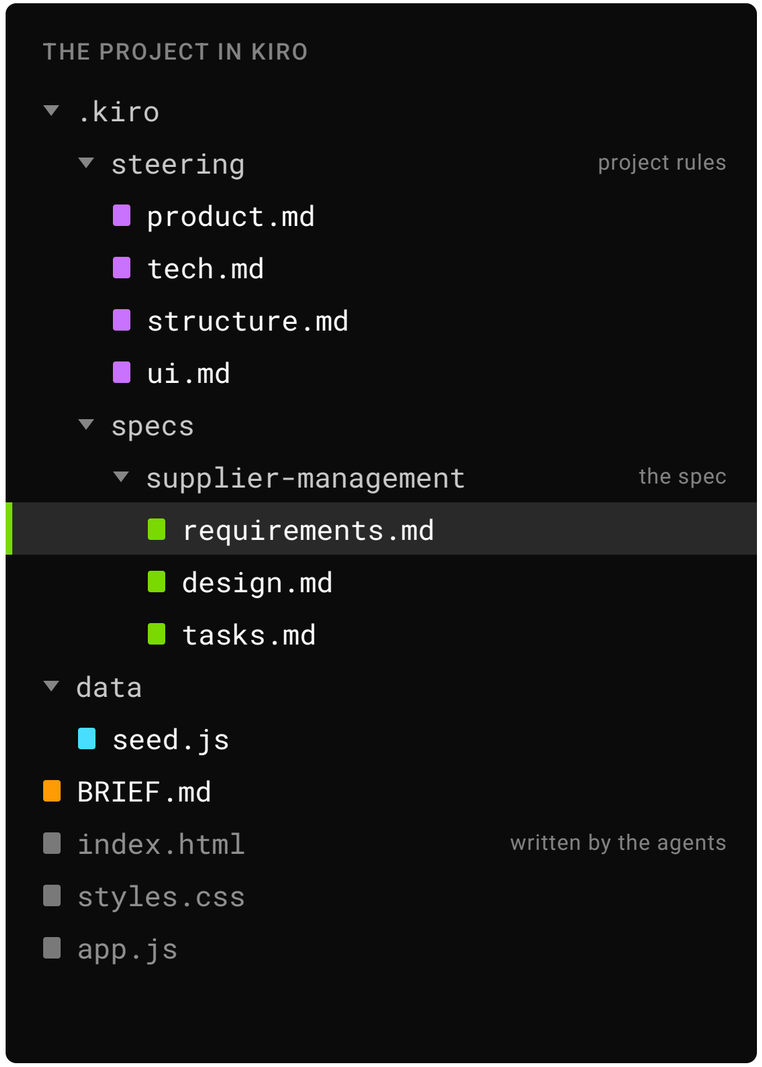
STEERING .kiro/steering/ui.md

How it looks

The look, in words: colours, type and layout.

```
 7 Calm, warm and editorial, like a well-made internal tool. Not a generic admin
   dashboard: no tile grids, no heavy borders, no gradients, no emoji.
   ⋮
10 - Page ground (sidebar and background): #FAF6EB, warm paper.
13 - Accent: #0E7C6B (links, active states, count badges, …).
   ⋮
21 - Home greeting 34 px, bold, letter-spacing -0.03em, centred.
```

> **Describe the look in words, or every AI-built app ends up looking the same.**



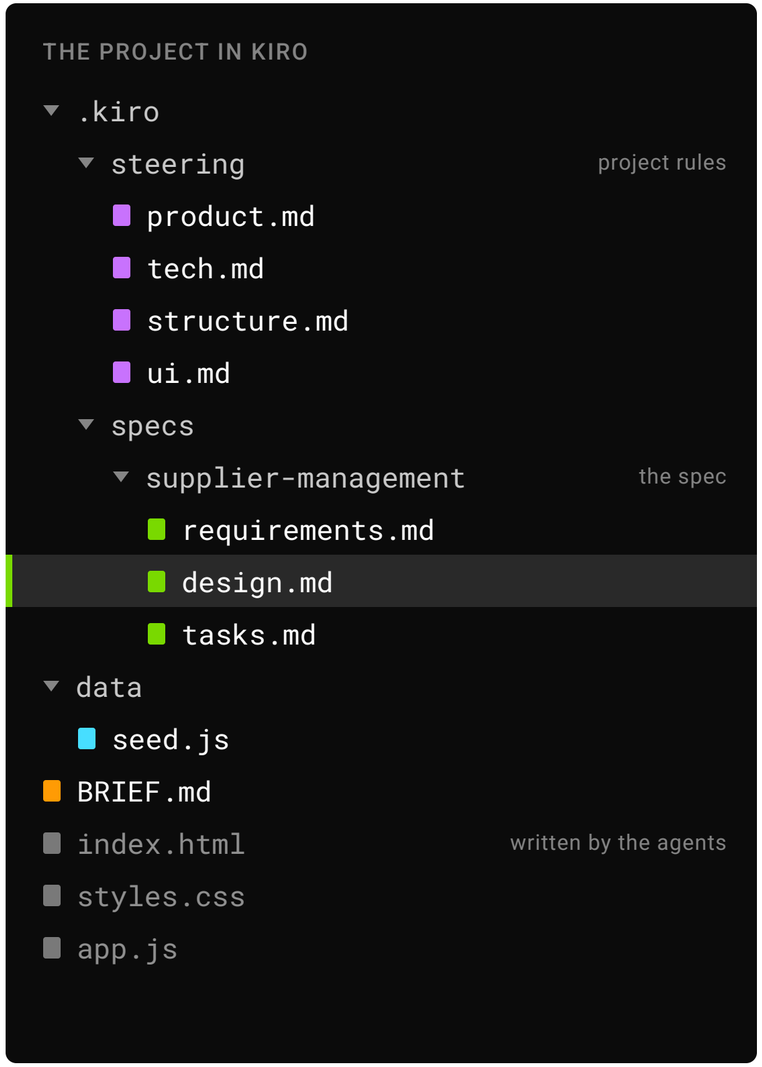
SPEC `.kiro/specs/supplier-management/requirements.md`

What it must do

12 requirements and 38 acceptance criteria, in plain sentences.

```
19 ### Requirement 2: Ask Otto
21 User Story: As a procurement manager, I want to type a question or pick a
   suggested one, so that I get an answer without building a filter.
   ⋮
27 3. WHEN the user clicks a suggested question THEN the system SHALL put its
   text in the question box without sending it.
```

> **Your customer can review this. So can your engineer. So can an AI agent.**



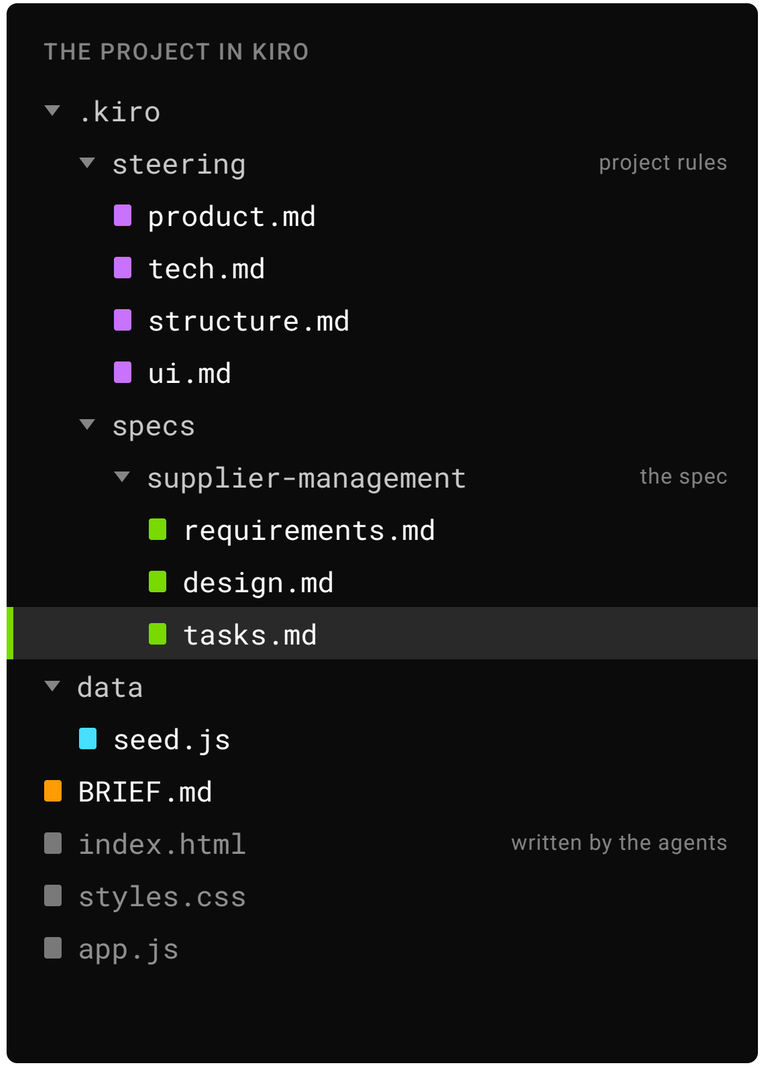
SPEC .kiro/specs/supplier-management/design.md

How it's built, with every rule copied in

The architecture, Otto's logic, the look, and the checks to run.

```
  7 In my first run, the task agents did not pick up the rules from the steering
    files. So every rule they need is repeated here.
    ⋮
107 3. Ending contracts: contains "expire", "expiring", "end", "ending" or "renew"
    → answerEndingContracts(days). Days = the first number in the text, else 90. …
    ⋮
152 ## Manual checks (run in the browser after each task)
154 - Attention line: 2 contracts and 1 framework agreement end within 30 days · 3
    expired documents · 4 suppliers flagged.
```

> **The right answers are written down before the build starts.**



SPEC `.kiro/specs/supplier-management/tasks.md`

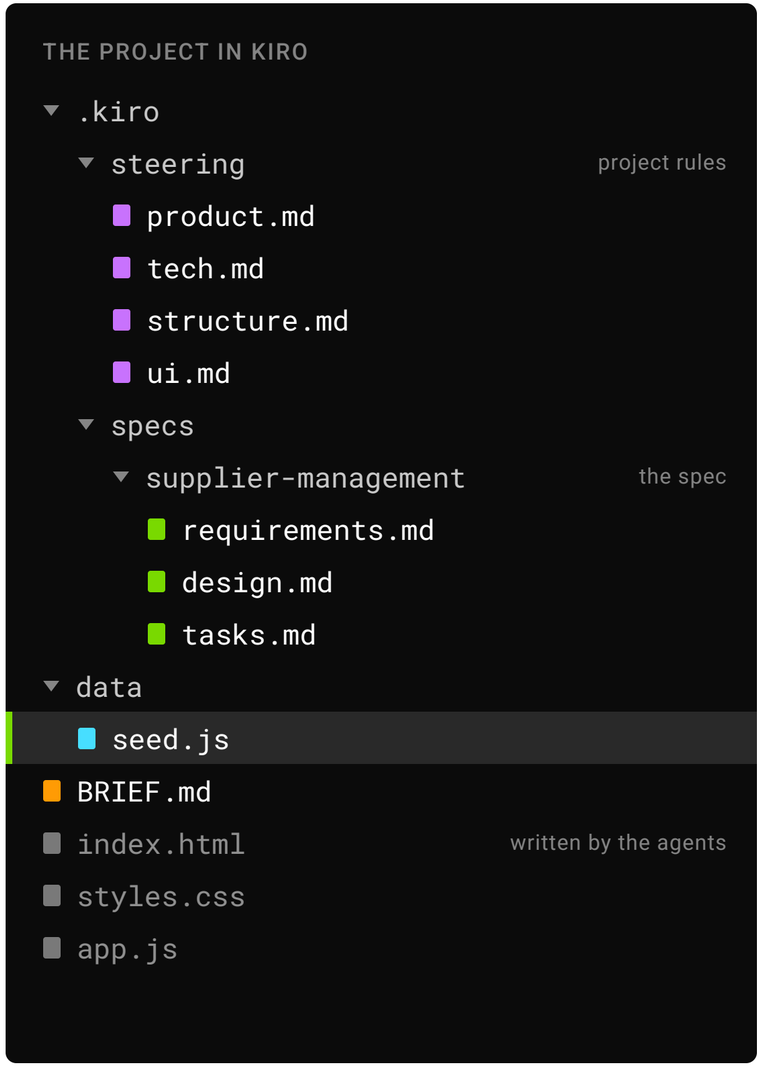
MID-BUILD SNAPSHOT

The build plan, ticked off as it builds

8 tasks, each linked to the requirements it covers.

```
 7 - [x] 1. Shell, styles and router
 8   - [x] 1.1 Create index.html with the sidebar, the top bar and …
12     - _Requirements: 12.1_
   ⋮
29 - [x] 3. Home and the conversation
40 - [ ] 4. Otto's four answers
41   - [ ] 4.1 Implement question matching in askOtto, in the order …
42     - _Requirements: 3.1, 4.1, 5.1, 6.1, 7.1, 7.2_
```

> **Every task points to a requirement. Nothing extra gets built.**



DATA data/seed.js

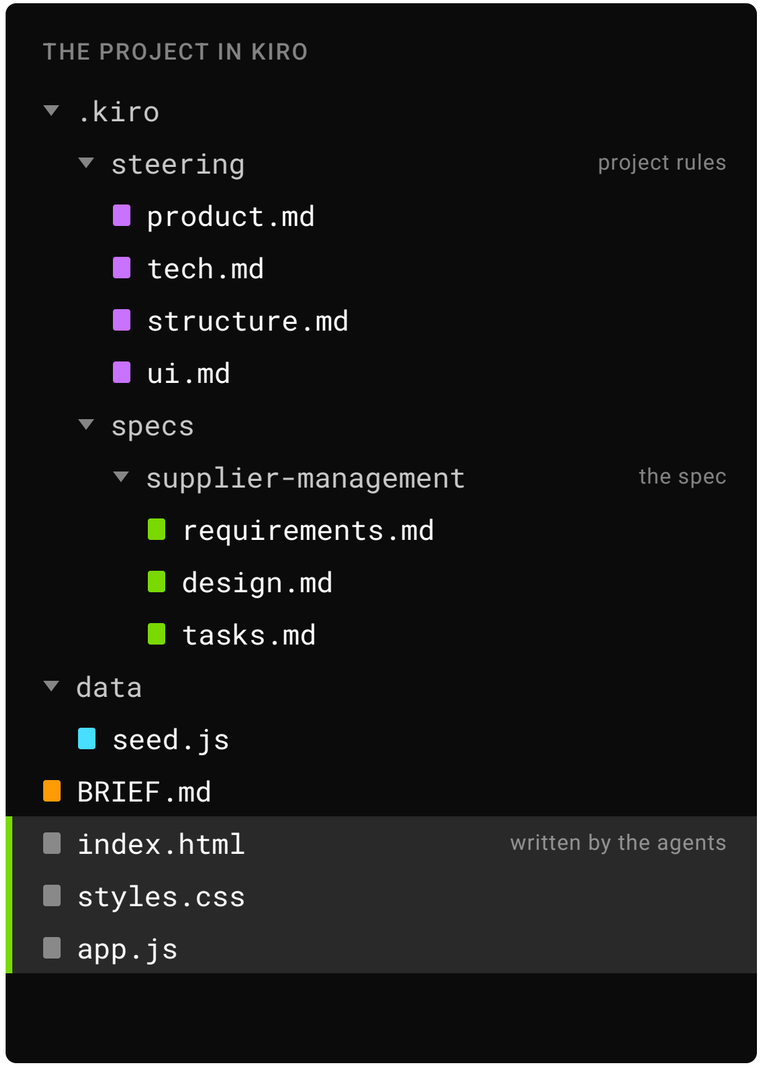
Fictional data, real rules

A fictional energy distributor with its suppliers, contracts and documents.

```
 8 window.SEED = Object.freeze({
 9   company: 'Arvena Energy',
10   today: '2026-10-08',
   ⋮
13     { id: 'S-101', name: 'Carpatia Grid Works SRL', …
16       status: 'Active', statusReason: '',
17       scores: { quality: 88, delivery: 82, documents: 90 },
```

> **Fictional, so I can show it on stage. A fixed date, so answers never change.**

THE PROJECT IN KIRO



WRITTEN BY THE AGENTS index.html · styles.css · app.js

FROM A REHEARSAL RUN

The app the agents write

Three files, nothing to install. The agents write them task by task.

```
101 // Calculate supplier score: round(0.4 * quality + 0.4 * delivery + 0.2 * documents)
102 function supplierScore(supplier) {
103   const scores = supplier.scores;
104   const score = 0.4 * scores.quality + 0.4 * scores.delivery + 0.2 * scores.documents;
105   return Math.round(score);
106 }
```

The rule from product.md, now in code, with a plain comment above it.

Build from the spec.

Once the spec is reviewed, an AI agent can build the prototype task by task.

1 paragraph

4 rule files

12 requirements

8 tasks

Kiro builds from the spec.



The green steps are the prepared spec. This session used task agents to implement it.

TASKS

Ticked off one by one as each agent finishes.

FILES

Appear in the project as they're written.

BROWSER

Reload to watch the app grow, screen by screen.

THE FULL BUILD

About an hour. The session also used builds prepared in advance.

The agents read the spec, not your notes.

MY FIRST RUN	Rules in a separate file	→	The agents skipped them	→	Blank page, wrong score
REVISED RUN	Every rule in the spec	→	Every agent reads it	→	**Every rule applied**

WHAT I LEARNED PREPARING THIS

Repeating the rules in the spec improved this demo.

Explicit rules help. Check the output against them after every build.

Show it. Then watch.

Hand the customer the keyboard, and stay quiet. Watch for three things:

The first question they ask

Where they hesitate

The answer they double-check

Ask a customer to try the prototype.
The stage session used a prepared build for this exercise.

Three right answers. One wrong order.

QUESTION 4

“Rank our framework agreements by value”

OTTO’S ANSWER

1	FA-2024-118	**18,500,000 RON**
2	FA-2023-207	**12,400,000 RON**
3	FA-2025-061	**9,800,000 RON**
4	FA-2025-034	**4,200,000 EUR**

4,200,000 EUR is 21,000,000 RON at 5 lei to the euro. It should be first.

THE SPEC SAID · DESIGN.MD, LINE 30

```
- Framework agreements are ranked by their value.
```

CHANGE THAT ONE LINE TO

```
- Framework agreements are ranked by their RON-equivalent value. For this demo, use a fixed demo rate of 1 EUR = 5.00 RON. Show the original amount alongside its RON equivalent.
```

THEN TELL KIRO

“Apply the revised ranking rule in design.md (Domain rules, line 30). Keep unrelated behaviour unchanged. Show me which files changed and how to check the result.”

Sell it. Then build it properly.

What the MVP proves to a buyer

- They'd open it every week
- It answers the questions they ask first
- Where it still needs work

What production adds

- Login and roles
- Real data from their systems
- An audit trail of who changed what
- Personal data handled properly
- A real language model, held to the same rule: answer only from the records

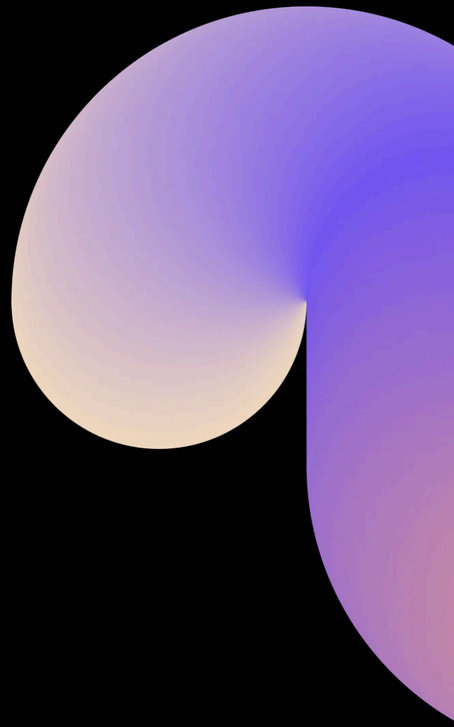
The spec carries over. Engineers and their agents start from the document the customer already said yes to.

What spec-driven development gives your company.

One source of truth	Your customer, your team and AI agents work from the same document.	Step 2 · Specify
Cheaper changes	Update the rule in the spec, rebuild the affected behavior, and check the result.	Step 5 · Update
Fewer surprises	The right answers are written down before the build, so you can check them.	design.md
A clean handover	Production starts from the spec the customer approved. The knowledge stays in it, not in one person's head.	Step 6 · Sell
Portable specification	The spec is plain text. Other spec-driven tools, and your engineers, can build from it.	The file tour

Try the first four steps with your own idea this month.

1. **Talk to customers. Write the pain in one paragraph.**
2. **Turn it into a spec, not a chat.**
3. **Build the MVP, then watch a customer use it.**

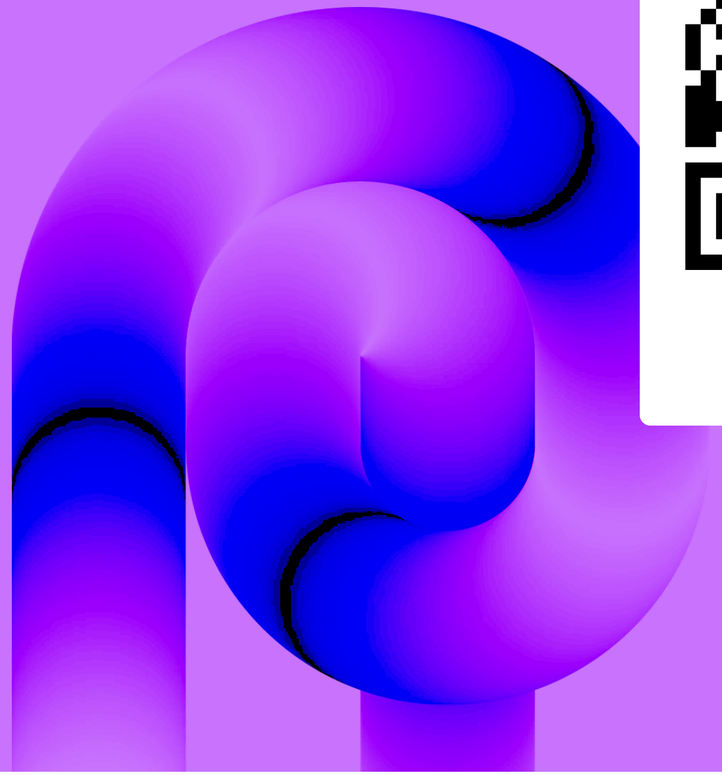


Continue the conversation.

Andrei Manea · Founder and CEO, CloudHero

cloudhero.io/contact/

Read the event recap at cloudhero.io/how-to-web-2026/



Connect on LinkedIn

Andrei · CloudHero