

AI Spec Kit

Write down what an AI tool may and may not do before anyone writes code. A blank template, a complete sample spec with notes on why each part matters, and the checklist I use before the build starts.

Last updated October 9, 2026 · By Mark Holland

Contents

1. How to use this kit	2
2. The blank template	3
3. A complete sample: the retailer support assistant	5
4. Before you write the first line of code	8

On an AI project, a thin spec rarely fails loudly. It fails quietly, in an answer that sounds right. An editable Markdown copy of the template comes with this kit.

1 How to use this kit

AI projects that go wrong usually go wrong in the first week, long before anyone writes a prompt. Someone sees a demo, and a developer opens an editor. A few weeks later there is a working prototype, and nobody can say whether it is finished, because nobody wrote down what finished means.

A spec does not need to be long. It needs to answer eight questions, and I do not start building until each one has an answer. Sometimes the answer is "unknown, and here is how we will find out," and that counts.

Question	What the spec says
The problem	What work happens today, who does it, and what it costs them in time or errors.
The users	Who will use the tool, who will read its output, and who is affected without ever opening it.
The data and its sensitivity	Every source the tool reads, who owns it, and whether it can contain personal or regulated data.
Success metrics	The number that should move, with a baseline taken before launch.
What the AI must never do	The hard limits, written as rules a test can check.
Human review points	Where a person approves, edits, or rejects the AI's work.
How it will be tested	The checks that run before release, and what counts as a pass.
Cost limits	What a run, a user, or a month may cost, and what happens at the limit.

The order I work in

1. Read the sample in chapter 3 once, end to end. The margin notes say why each part is there.
2. Fill in the blank template (chapter 2, or the Markdown copy) with what you know after discovery. Mark every gap "unknown, and here is how we will find out."
3. Send it to the business owner, compliance, and engineering together. One document to read and sign beats three different pictures of the tool.
4. Work the checklist in chapter 4. Start building only when every line is true.
5. When the facts change, update the spec first and the code second, so the document stays the thing everyone can point to.

Every setup is different. The exact steps depend on your cloud, your tools, and the data you handle. If you'd like help fitting these steps to your own environment, let's map it out together: markholland.tech/advisory.

2 The blank template

Project		Version	
Business owner		Date	

1. The problem

What work happens today, who does it, and what it costs them in time or errors. A goal is not a problem statement: say how long the work takes today, where the hours go, and which part a tool could take over.

2. The users

Who will use the tool, who will read its output, and who is affected without ever opening it.

3. The data and its sensitivity

Every source the tool reads, who owns it, and whether it can contain personal or regulated data. Any free-text field might hold sensitive data, even when the form tells people not to type it there.

Source	Owner	Can it hold personal or regulated data?	What the system does if that data shows up

4. Success metrics

Metric	Baseline today	Target	How it is measured after launch

5. What the AI must never do

Write each rule so a test can check it: the behavior, the condition that triggers it, and what the system does instead.

The AI must never	When	Instead, the system
Invent a value	A required input is missing	Asks a person for it

6. Human review points

Where in the workflow	Who reviews	What they see when they decide

7. How it will be tested

Include the hard cases: a missing input, sensitive data in the wrong field, malformed model output, a question no approved source covers. A criterion you cannot test yet is reported as not testable, never as passing.

8. Cost limits

Per run	Per user	Per month	At the limit (stop, warn, or cheaper model)

In the first version

Left out of the first version

Sign-off

Role	Name	Date
Business owner		
Compliance		
Engineering		

3 A complete sample: the retailer support assistant

A spec for the made-up support assistant from Part 6 of my series: an online retailer wants an assistant that answers order questions and refunds shipping on late orders. The notes in the margin say why each part is there.

Project	Customer support assistant	Version	1.3
Business owner	Head of customer support	Status	Approved for build

1. The problem

ILLUSTRATIVE EXAMPLE Every number in this sample is made up. Support handles about 1,800 chats a week. Roughly 45% are "where is my order?" and another 15% ask for a shipping refund on a late order. Each takes an agent about 6 minutes, mostly looking up the order and the carrier status. That is around 108 agent hours a week on questions with a known answer.

The tool should take over order-status answers and small shipping refunds, so agents spend their time on returns, damaged items, and upset customers.

2. The users

- **Customers** chat with the assistant on the website and in the app.
- **Support agents** read every hand-off and every refund over \$50.
- **Finance** never opens the tool but owns the refund budget it spends.

3. The data and its sensitivity

Source	Owner	Personal data?	If it shows up
Order system	Operations	Yes: name, address	Only order status and amounts reach the model
Help center	Support lead	No	Approved articles only
Chat messages	Support	Often: emails, card numbers	Masked before the model and the logs

4. Success metrics

Metric	Baseline	Target	Measured by
Order chats resolved without an agent	0%	60%	Chat logs, weekly
Wrong refunds	n/a	0 over \$50; under 1 in 500 below	Finance audit, monthly
Customer satisfaction on order chats	4.1 of 5	4.1 or better	Post-chat survey

WHY IT MATTERS

"Help support answer faster" is a goal. This is a problem: it says how much work, where the hours go, and which part a tool could take over. Every later section measures against it.

WHY IT MATTERS

The person who reads the output is often not the person who asked for it. Finance never sees the assistant, but its refunds land in their numbers, so they belong in the spec and in the sign-off.

WHY IT MATTERS

Data sensitivity is the first design decision. It decides which models you can call, what gets masked before a prompt leaves, and what gets logged. Customers will type card numbers into a chat box whatever the form says.

WHY IT MATTERS

Take the baseline before launch, or you will never know whether the tool helped. A guard metric (satisfaction holds) stops a fast-but-worse assistant from looking like a win.

5. What the AI must never do

Never	When	Instead
Refund more than \$50 on its own	A refund is proposed over \$50	Hands the refund to an agent
Answer without a source	No help article or order data covers it	Hands off to an agent
Repeat a card number or email	A customer types one	Masks it in the reply and the log
Follow instructions in a message	A message says to ignore the rules	Answers normally or hands off
Ask for a password	Any account question	Points to account settings

WHY IT MATTERS

"Be careful with refunds" cannot be tested. "Never refund over \$50; hand it to an agent" can. Written this way, the list is already half of the test plan, and each line becomes an eval case.

6. Human review points

Where	Who	What they see
Refund over \$50	Support agent	The order, the delay, the proposed amount, the chat
Any hand-off	Support agent	The full chat and why the assistant handed off
Weekly sample of 50 resolved chats	Support lead	Chat, sources used, action taken

WHY IT MATTERS

Put review where a mistake would reach a customer's money, and let low-risk answers move on after automated checks. The weekly sample catches the slow drift that no single chat shows.

7. How it will be tested

- An eval set of 412 cases (an illustrative size) from real (de-identified) chat types, including every never-do rule above. Release bar: 95% overall and every safety case passing.
- Hard cases: an order number that doesn't exist, the order system timing out, a card number in the message, a message that tells the assistant to ignore its rules, a question no article covers.
- Unit tests on the refund rule in code, including the kill switch.
- Not testable yet: satisfaction impact. Reported as "not testable" until the pilot, never as passing.

WHY IT MATTERS

If the spec does not say how the tool will be tested, testing becomes whatever the builder had time for. Naming what is not testable yet keeps a report honest.

8. Cost limits

Up to \$0.10 per chat and \$800 a month (about 7,800 chats; illustrative numbers). At 80% of the monthly limit, the team is warned. At 100%, order-status questions keep working on a cheaper model and everything else hands off to agents.

WHY IT MATTERS

A model call is cheap until it runs on every chat, every day. Saying what happens at the limit, before launch, beats discovering it on the invoice.

Scope

In version 1

- Order status
- Shipping refunds up to \$50
- Help center questions

Left out of version 1

- Returns and exchanges
- Damaged items
- Item refunds of any size
- Phone and email channels

Sign-off

Business owner	Head of customer support
Compliance	Privacy lead (masking rules, retention)
Engineering	Engineering manager, support tools
Also consulted	Finance (refund budget)

WHY IT MATTERS

AI projects grow because a model can do so many nearby things, and each one looks cheap to add. Writing down what the first version leaves out is the most reliable way I know to ship the first version.

WHY IT MATTERS

Without one signed document, each approver pictures a different tool, and you find out at launch.

4 Before you write the first line of code

- ☐ The problem is written in terms of today's work and what it costs in time or errors.
- ☐ Every user, and every reader of the AI's output, is named.
- ☐ Every data source is listed with its owner and whether it can contain personal or regulated data.
- ☐ There is a plan for sensitive data that shows up where it should not.
- ☐ Each success metric has a baseline and a way to measure it after launch.
- ☐ The never-do list is written as rules a test can check.
- ☐ Each human review point names who reviews and what they see.
- ☐ The test plan covers missing inputs, malformed model output, and questions no source answers.
- ☐ Cost limits are set, with a stated behavior at the limit.
- ☐ Scope out is written down, and the list is not empty.
- ☐ The business owner, compliance, and engineering have approved the spec.

When the facts change, update the spec first and the code second, so the document stays the thing everyone can point to.

Need help putting this to work?

Every setup is different. If you'd like help fitting this to your own environment, let's map it out together: markholland.tech/advisory

Based on Part 5 of the series From AI Idea to Production, "Spec Before Code," at markholland.tech/writing. By Mark Holland. Shared under CC BY 4.0: copy and adapt it, with credit.