Agent Journey Map

Verified 2026-11-22

Design for agents before you build — seven ordered steps from discovery to hand-back, what each needs, how it breaks, and the question to ask first.

Contents

Mental model 2

Map, not measurement 2

1. Discover 4

2. Prove it has permission 5

3. Understand the offer 6

4. Attempt 7

5. Interpret the result 9

6. Recover 10

7. Hand back 11

Using the map 13

Further reading 14

Mental model

A user journey map assumes someone who can see the screen, read the hints, and work out what you meant. An agent has none of that, so the same app takes a different shape: seven steps, strictly ordered, where an agent cannot get through a step until the one before it works. Most teams design only step four — the callable API — and then nothing uses it, because no agent could find it or prove it had permission.

Map, not measurement

Tool	When	Answers
User journey map	Design	What a human sees and does
Agent journey map	Design	What an agent needs at each step
Agent Analyzer	After launch	What an agent really did on a page

The map is a framework for the design stage: the steps, what each one needs, and the question to answer while changing your answer is still cheap. The seven steps, in order:

1. Discover
2. Prove it has permission
3. Understand the offer
4. Attempt
5. Interpret the result
6. Recover
7. Hand back

> **Gotcha:** Starting at step four feels like engineering, so teams do. An API no agent can discover or authenticate to is unused, not early.

1. Discover

Needs	Not
Machine-readable description of the app	Marketing copy
Published list of the actions offered	A sales page
A fixed address for that list	A URL that moves per release

The agent must learn that your software exists and what it can do without a human reading your marketing. When the action list lives in a PDF, a Notion page, or a developer site behind a login, the agent never learns the action exists — and falls back to clicking through your screens the way a human would.

```
Stable:   /actions        (same every release)
Hidden:   /docs/v3.pdf    (unreadable)
Gated:    dev portal      (login first)
```

Ask before building: if an agent had never heard of us, which single web address would tell it what we can do?

2. Prove it has permission

Needs	Instead of
Credentials an agent can hold	The user's own password
Narrow, grantable permissions	All-or-nothing access
Tell a sent agent from a scraper	Blocking both

The agent has to show it acts for a specific person, with their permission, for a limited purpose. Scope permissions so handing over one task does not hand over the whole account.

```
grant: bookings:reschedule   ← the task
not:   bookings:*            ← everything
not:   account:admin         ← far worse
```

> **Gotcha:** If the only way in is a human login with a one-time code, the agent either stops there or the user hands over their password. Both are worse than designing for agents up front.

Ask before building: what is the smallest permission that lets an agent do the everyday task, and can someone grant only that?

3. Understand the offer

Needs	Why
Actions named for the outcome	Internal terms mean nothing
Required inputs, formats, units	No guessing `10` = £10 or 10p
Prerequisites stated explicitly	Order is not implied

The agent works out which action matches what it was sent to do, and what that action needs. A form's screen

order silently encodes dependencies; an agent looking at actions one by one sees none of it.

```
Bad:  submitStageB(payload)
Good: reschedule_booking
        booking_id: string (required)
        new_start: ISO 8601 datetime
        requires: booking status "confirmed"
```

> **Gotcha:** Steps that only make sense in order because the screens forced it get tried in the wrong order — and fail without the agent knowing why.

Ask before building: reading only our action names and inputs, could a newcomer pick the right one first time?

4. Attempt

Needs	Breaks on
Completable without a gesture	Drag-to-reorder, sliders

Needs	Breaks on
Idempotency (safe retries)	A second order on retry
No human-only friction	CAPTCHAs, visual-only confirms

This is where the task is carried out — and the only step most teams design for. Idempotency means a retried request has the same effect as the first, so a timeout followed by “try again” never creates a duplicate.

```
POST /orders  key=7f3a  → 201 order #1042
POST /orders  key=7f3a  → 201 order #1042
                          (same order, no copy)
```

> **Gotcha:** One step mid-flow that needs a mouse or a human eye — a slider, a CAPTCHA, a confirmation shown only as a visual cue — lets the agent get most of the way there and then stop.

Ask before building: can the whole task be done without seeing the layout?

5. Interpret the result

Kind	Agent should
Wrong input	Fix the named field, then retry
Try again later	Wait, then retry the same request
Will never work	Stop retrying; escalate

The agent has to learn whether it worked and, if not, whether to try again. Every error must say which of the three kinds it is, name the exact field or value that was wrong, and describe what happened rather than give a bare error number.

```
{
  "kind": "wrong_input",
  "field": "new_start",
  "detail": "Must be after 2026-11-22T09:00Z"
}
```

> **Gotcha:** One generic failure for everything means the agent can't tell a typo from an outage: it retries what will never work and gives up on what would have worked a second later.

Ask before building: does every error say which of those three kinds it is?

6. Recover

Needs	So the agent can
A way to check current state	Look before acting again
Reversible actions, or an undo	Back out of a bad step
A retry-versus-human line	Know when to stop and ask

The agent gets back on track without a human, or decides a human is needed. Recovery must exist in the

same channel the agent is using: an API caller cannot follow “go to settings and check”.

```
GET  /orders/1042      → status: "pending"
                           (half-finished: resume)
POST /orders/1042/undo → cancelled
retry myself:  "try again later" errors
hand to human: refunds, anything irreversible
```

> **Gotcha:** When the only recovery path runs through the screens, an agent on your API has no route back — it stalls or repeats the thing that failed.

Ask before building: after a half-finished task, can an agent find out where things stand and carry on?

7. Hand back

Needs	Gives the user
A human-readable record	A way to audit later
Decisions raised beforehand	A choice, not a report

Needs	Gives the user
A reference, receipt or link	Proof it happened

The agent returns to the user with a result they can check, or a decision only they can make. Raise decisions before they are made, not afterwards.

```
Done: rescheduled booking B-2291
From: Tue 10:00  To: Thu 14:00
Ref:  CHG-58120  (link to booking)
Ask:  new slot costs £5 more — approve?
```

> **Warning:** If the agent says it succeeded and the user can't check, the first wrong "done" ends their trust — and you lose every future task that would have come through that agent, not just this one.

Ask before building: when the agent says it's done, what can the user look at to be sure?

Using the map

Pass	Task	Reveals
1	One a customer would delegate	Where agents get stuck
2	One you would not want delegated	Where access must stop
Diff	Where the two paths split	Your agent-access policy

Pick one task a customer might well hand to an agent — rescheduling a booking, reordering something they keep running out of, cancelling a plan. Walk it through all seven steps, answering each question honestly for that task alone. If you can't answer the first three, start there. Then repeat with a task you would not want an agent doing, and write down where the paths split now rather than finding out from live customers.

> **Tip:** Run each pass for one task only. Answering for "the app in general" hides the step where a specific task actually breaks.

Further reading

- The agent journey map — the source framework this sheet follows
- RFC 9457 — Problem Details for HTTP APIs — a standard shape for step 5's structured errors
- RFC 9110 — HTTP Semantics — idempotent methods and `Retry-After`
- OpenAPI Specification — one machine-readable format for step 1's action list