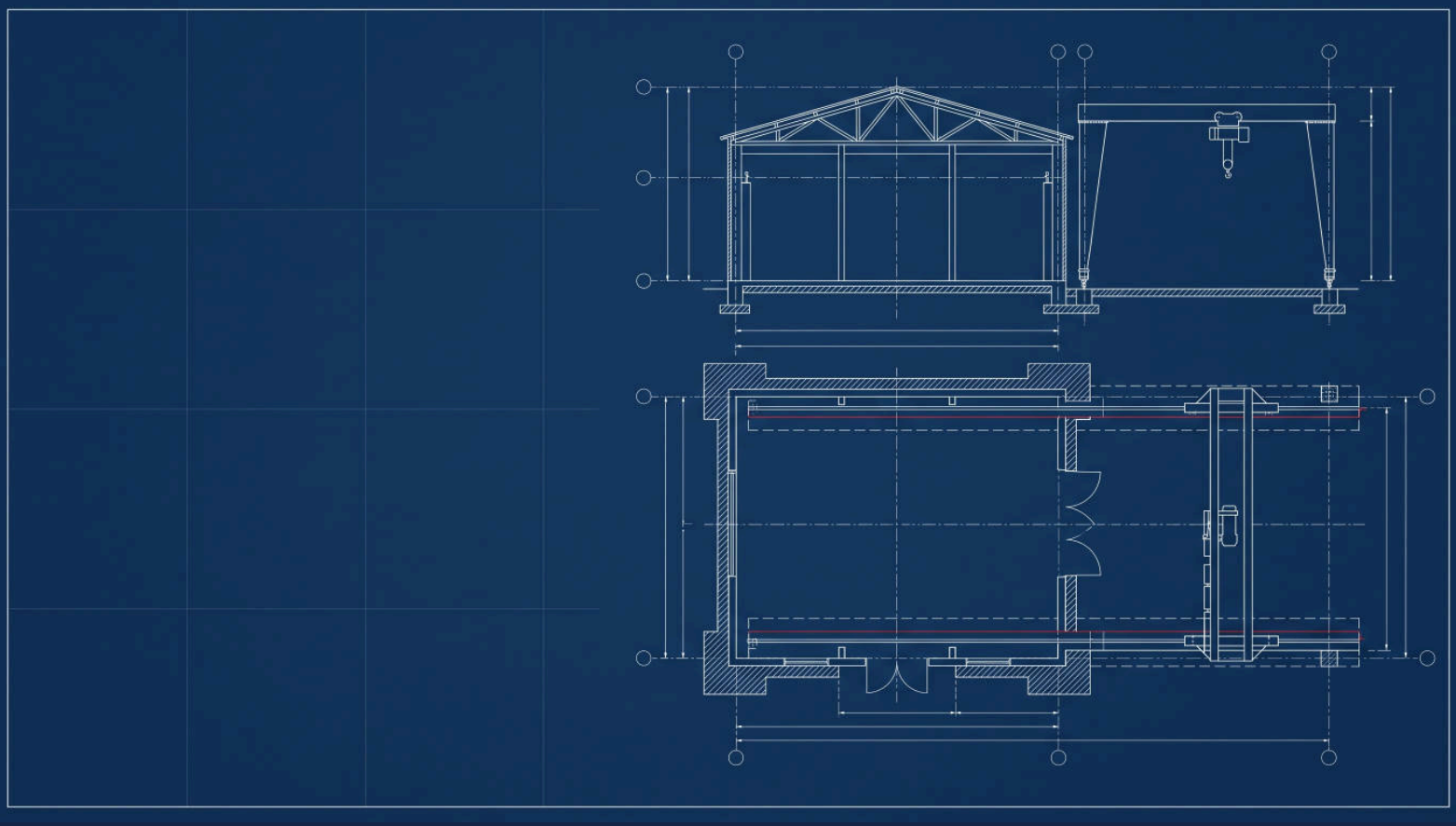

CHAPTER NINETEEN

Project Ideas and Real-World Problems

SECTIONS 19.1 TO 19.9

Exercises isolate one argument each. Projects remove the bounds, which is the point: the difficulty of real work is that the constraints arrive together and interact.

CHAPTER NINETEEN

Contents

PART V · PRACTICE

Exercises isolate one argument each so the measurement stays clean. Projects remove the bounds, and that is the point: the difficulty of real work is that the constraints arrive together and interact. This chapter is a set of projects at three sizes, five domains with genuinely different hard parts, the scoping questions that decide whether you finish, four agentic applications specified the way you would hand them to a team, and a ten-project ladder that isolates one capability per rung.

19.1 How to Choose a Project

Figure 19.1 The size that teaches you the most is usually one smaller than the one you want to build.

Four questions, and the third is the one that decides whether you finish.

Is there a real corpus? Not a scraped one, not a synthetic one: documents someone actually needs answers from. Without this you will build retrieval that works on your test set and nowhere else, because you have no distribution to be wrong about.

Are there real questions? Thirty of them, asked by a person, before you write any code. Section 14.2's argument applies with more force at project scale: invented questions produce a system that answers invented questions.

Does a wrong answer cost something? This is the one that matters most for learning. A project where being wrong is free teaches you nothing about abstention, grounding, approval or degradation, which is most of the interesting content in this book.

Can you finish the smallest version in the time you actually have? Multiply your estimate by three. The abandoned-repository failure mode is almost always scope, not difficulty.

Pick the smallest project that still scares you. Fear is a good signal here: it usually means an irreversible action, a real user, or a cost you have to defend. Those three are where the architecture in this book earns its keep, and a project with none of them is a tutorial with your name on it.

19.2 Weekend Projects

Two days, no users, one measurement. Each isolates a seam and produces a number you can show. These stay in the compact form because a weekend project *is* its number; the larger tiers below are specified the way you would hand a build to a team.

Project	What it teaches, and the number it produces
The honest chunker	Ingest one hundred PDFs three ways: fixed-size, structural, and structural with metadata. Report recall at 20 for each on thirty real questions. Teaches section 6.5, and the metadata column usually wins by more than expected.
Router economics	Classify a day of real queries by difficulty, route cheap ones to a small model, and report spend before and after with quality held. Section 3.6 got an 80 percent reduction; get your own figure.
The abstention switch	Add a relevance floor to an existing RAG demo. Report accuracy-on-answered and coverage at three thresholds. Teaches why section 6.9's trade is a product decision rather than a tuning one.
Trace-driven eval	Instrument one endpoint with the trace seam, run it for a day, and build twenty labelled cases from the traffic. Teaches why Chapter 1 built the seam before there was anything to trace.

19.3 One-Month Projects

Real corpus, real questions, one degraded path that genuinely works. These are portfolio pieces, and each is specified three ways: an objective, a specification table, and numbered functional requirements you can check off.

MP-1: A grounded assistant for a corpus you own

Objective. Answer questions over documents you actually care about: your own notes, a project archive, or a public collection someone needs answers from. The deliverable is not the demo, it is the eval report.

Item	Specification
Input	A real corpus and thirty real questions (section 19.1's first two gates)
Output	Cited answers with a working abstention path, plus the eval report
Human in the loop	Not required
Difficulty	Moderate
Acceptance numbers	Recall, accuracy-on-answered, coverage, and cost per successful task

Functional requirements

ID	Requirement
MP-1.1	Ingest with structural chunking and metadata (6.5).
MP-1.2	Retrieve with a hybrid dense and lexical leg fused by reciprocal rank fusion (7.5, 7.6).
MP-1.3	Rerank broad, then truncate narrow, before assembly (7.7).
MP-1.4	Emit citations that resolve to retrieved chunks (6.8).
MP-1.5	Abstain below a measured relevance floor (6.9).
MP-1.6	Produce the eval report from a labelled case set of real questions (14.2).

MP-2: A meeting-to-actions pipeline

Objective. Transcripts in, structured commitments out: who owes what to whom by when. The hard part is that a missed commitment is invisible and a fabricated one is embarrassing, which is section 6.9's asymmetry in a new setting.

Item	Specification
Input	Meeting transcripts (text)
Output	Structured commitment records: owner, action, counterparty, due date
Human in the loop	Not required
Difficulty	Moderate
Acceptance numbers	Precision and recall of extracted commitments against hand-labelled transcripts

Functional requirements

ID	Requirement
MP-2.1	Extract into a declared schema with an explicit insufficient-evidence escape (5.3).
MP-2.2	Ground every commitment in a transcript span it can point back to.
MP-2.3	Process idempotently: the same meeting processed twice must not create two tasks (12.5).
MP-2.4	Run ingestion off the request path on a queue (Chapter 8).

MP-3: A support-ticket triage service

Objective. Classify, route and draft, with a human approving anything sent. Volume economics bite immediately, so cost per successful task stops being theoretical.

Item	Specification
Input	Inbound support tickets
Output	A class, a route, and a drafted reply queued for review
Human in the loop	Required before any reply is sent
Difficulty	Moderate
Acceptance numbers	Cost per successfully resolved ticket, and the reviewer disagreement rate (15.5)

Functional requirements

ID	Requirement
MP-3.1	Classify tickets on a cheap routed tier (3.6).
MP-3.2	Route by class of service so a backlog cannot starve interactive work (8.9).
MP-3.3	Draft replies grounded in the ticket and the relevant policy documents.
MP-3.4	Queue every draft for human review; send only after approval (15.5).
MP-3.5	Attribute cost per ticket from the trace record (16.2).

MP-4: A code-aware documentation agent

Objective. Answer questions about a repository, citing files and line ranges. Teaches how badly generic chunking handles code, and why the lexical leg matters when every query is an identifier.

Item	Specification
Input	A repository and thirty real questions from someone who works in it
Output	Answers citing files and line ranges
Human in the loop	Not required
Difficulty	Moderate
Acceptance numbers	Accuracy-on-answered and coverage on the thirty questions

Functional requirements

ID	Requirement
MP-4.1	Chunk along code structure, not fixed sizes (6.5, 7.3).
MP-4.2	Carry a lexical leg with an identifier-preserving tokeniser (7.5).
MP-4.3	Emit citations that resolve to file and line range.
MP-4.4	Abstain when the answer is not in the repository (6.9).

MP-5: A research agent on a declared graph

Objective. Pick an operations task with real steps: gather evidence from three sources, reconcile it, draft a summary, file one ticket. Build it as Chapter 10's graph rather than a free loop.

Item	Specification
Input	A fixed set of thirty real tasks
Output	A reconciled summary and one filed ticket per task
Human in the loop	Required for the one irreversible action (10.7)
Difficulty	Moderate to high
Acceptance numbers	Compound success rate over the thirty tasks, reported alongside the per-step rate that produced it (9.1)

Functional requirements

ID	Requirement
MP-5.1	Declare the topology as a graph with per-node budgets (10.3, 10.4).
MP-5.2	Confine the open-ended part to one bounded agent node (10.11).
MP-5.3	Checkpoint so a run survives a deploy mid-flight (10.5).
MP-5.4	Hold the irreversible action behind an approval interrupt (10.7).

19.4 One-Quarter Projects

Someone else depends on it, cost has an owner, and an incident will happen.

QP-1: A multi-tenant document product

Objective. The whole book, essentially. The genuine difficulty is that isolation and deletion have to be right on day one, because retrofitting either means a migration under pressure.

Item	Specification
Input	At least two tenants' corpora and traffic
Output	A running product with an owner for its cost
Human in the loop	Sampled review of refusals and incidents (15.5)
Difficulty	High
Acceptance numbers	A release passing section 14.6's five gates, and one deletion demonstrated across all five surfaces (15.4)

Functional requirements

ID	Requirement
QP-1.1	Enforce tenancy at query time from authenticated identity (6.10).
QP-1.2	Delete across all five data surfaces, traces included (15.4).
QP-1.3	Attribute cost per tenant, feature and outcome (16.2).
QP-1.4	Gate every output through a single policy chokepoint (15.1).
QP-1.5	Release through the eval gate and a canary (14.6, 14.7).

QP-2: A voice agent for a narrow task

Objective. Booking, status lookup, or intake. Chapter 13's budget is unforgiving and will expose every slow decision in your pipeline. Choose a task where the vocabulary is small and the identifiers are spoken, so section 13.7's invisible failures are the main event rather than an afterthought.

Item	Specification
Input	Live calls for one narrow task with a small vocabulary
Output	The task completed in-call, or a handoff that carries state
Human in the loop	Escalation path with the conversation state handed over (15.5)
Difficulty	High
Acceptance numbers	Time to first audio inside section 13.1's budget on fifty test calls, and conversation state surviving section 13.4's barge-in truncation without corrupting a later turn

Functional requirements

ID	Requirement
QP-2.1	Stream every stage; never wait for a full transcript or a full answer (13.3).
QP-2.2	Tune endpointing on your own callers, not a global default (13.2).
QP-2.3	Implement barge-in as all four operations, including transcript truncation (13.4).
QP-2.4	Track the invisible failures as first-class metrics (13.7, 13.9).
QP-2.5	Fall back to keypad entry for high-risk identifiers on narrowband calls (13.8).

QP-3: An assistant whose memory is the product

Objective. A personal or team assistant used across weeks, where the hard part is Chapter 11 rather than retrieval.

Item	Specification
Input	Weeks of real use by you or a small team
Output	Answers that use remembered facts, each with provenance
Human in the loop	The user can inspect, correct and delete what is remembered (11.8)
Difficulty	High
Acceptance numbers	Recall on a planted set of facts stated across earlier sessions, and zero superseded values surfacing in answers, measured the way section 11.10 measures memory

Functional requirements

ID	Requirement
QP-3.1	Write memory only through the gated write path (11.5).
QP-3.2	Supersede rather than overwrite when facts change (11.7).
QP-3.3	Retrieve memory into a bounded budget region alongside evidence (11.6).
QP-3.4	Demonstrate the deletion path end to end (11.8).

QP-4: An internal knowledge system with real permissions

Objective. Different people may see different documents. The failure mode is a citation to a document the reader is not allowed to open, which is worse than no answer.

Item	Specification
Input	An organisation's documents with per-document access control
Output	Answers built only from documents the asking reader may open
Human in the loop	Not required
Difficulty	High
Acceptance numbers	Zero citations to unreadable documents on a permission-labelled eval set, with coverage reported beside it

Functional requirements

ID	Requirement
QP-4.1	Apply the per-document ACL as a query-time filter from authenticated identity, never from the model (6.10).
QP-4.2	Keep ingestion fresh enough that permissions changes propagate inside a stated window.
QP-4.3	Resolve citations against what the reader may open, not what the index contains.
QP-4.4	Abstain when only forbidden documents could answer (6.9).

19.5 Real-World Problem Patterns

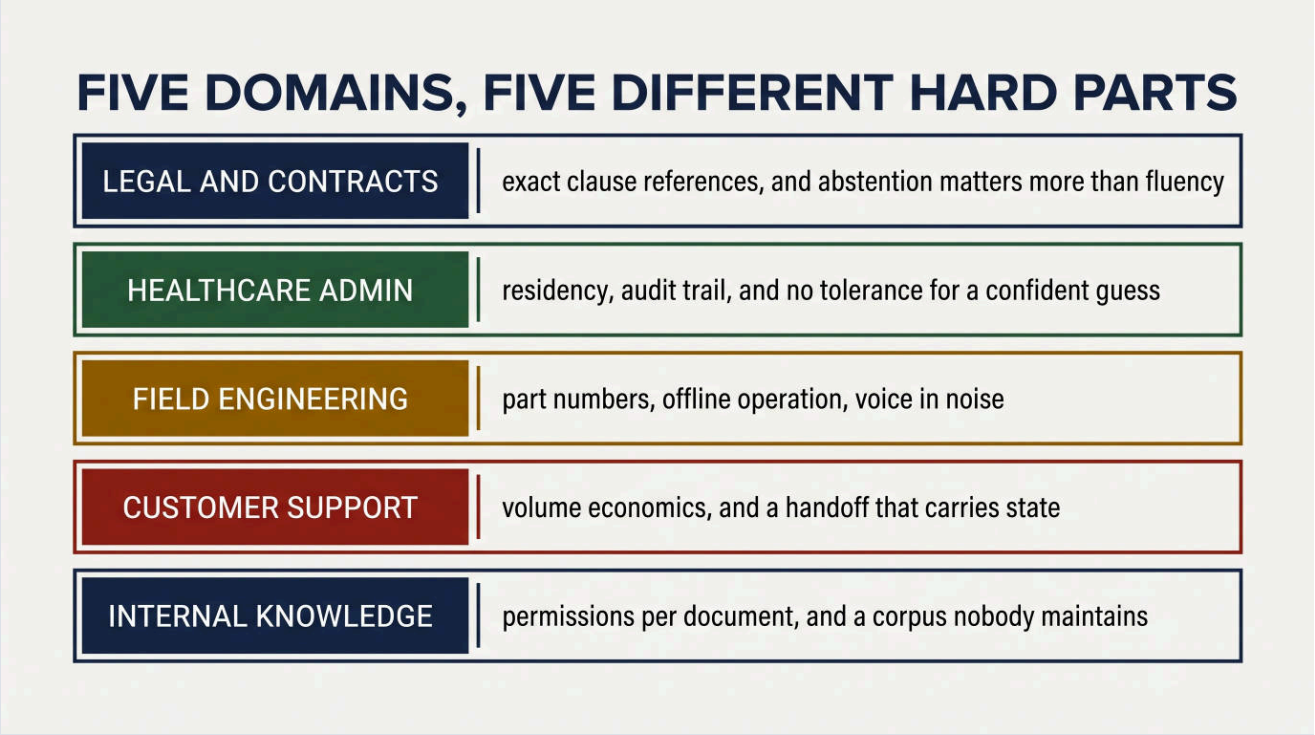

Figure 19.2 The same architecture, with the difficulty in a different place each time.

Domains differ less in their pipeline than in *which constraint dominates*, and picking a domain is mostly choosing which chapter you will live inside.

Domain	The dominant constraint, and the chapter it lands in
Legal and contracts	Exact clause references, and abstention valued over fluency. Lexical retrieval and citation resolution carry the product. *Chapters 6, 7*
Healthcare admin	Residency, audit trail, and no tolerance for a confident guess. The output policy and audit trail are the product. *Chapters 15, 16*
Field engineering	Part numbers, noisy audio, intermittent connectivity. Local models and voice constraints dominate. *Chapters 4, 13*
Customer support	Volume economics and a handoff that carries state. Routing, queue classes and cost attribution decide viability. *Chapters 3, 8, 16*
Internal knowledge	Per-document permissions and a corpus nobody maintains. Ingestion freshness and access control dominate. *Chapters 6, 12*

One constraint cuts across all five rather than choosing a domain: a corpus that is mostly scanned pages and tables is an extraction problem before it is anything else, and section 6.2 is where that fight is won or lost.

19.6 Scoping, Milestones and What to Measure

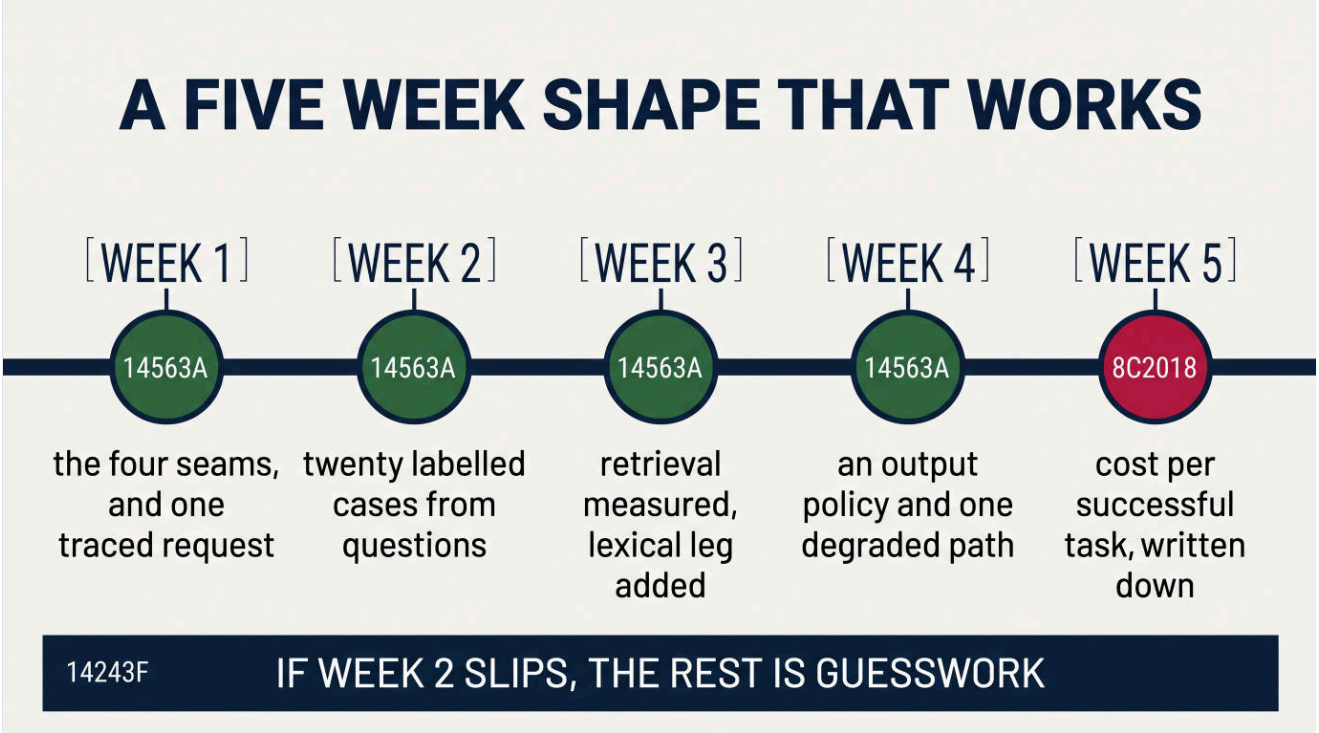

Figure 19.3 Week two is the load-bearing one. Everything after it is guesswork without it.

A shape that has worked repeatedly, and the reason it works is that the measurement arrives before the optimisation.

Week	Deliverable
One	The four seams, and one request traced end to end with versions recorded.
Two	Twenty labelled cases from real questions. *If this slips, stop and finish it.*
Three	Retrieval measured, failures labelled by stage, lexical leg added.
Four	An output policy and one degraded path that has been tested by breaking something.
Five	Cost per successful task, written down, with the success rate that produced it.

> THE FAILURE MODE IS ALWAYS WEEK TWO
>
> Labelling cases is unglamorous and produces nothing demonstrable, so it slips. Every week after it then produces numbers nobody can defend, and the project drifts into tuning prompts by feel, which is where abandoned repositories come from.
>
> If you have one week rather than five, spend it on week two.

The quarter tier does not need a new shape; it needs this one run three times with rising stakes. The first pass, weeks one to five as above, proves the loop on one tenant. The second adds the second tenant, and with it the isolation and deletion work that cannot be retrofitted. The third adds chapter 16's machinery: the canary, the runbook, and an on-call rotation that has rehearsed one incident before meeting a real one.

19.7 Writing It Up

A project nobody can read is a project you cannot be credited for. Four sections, one page:

The constraint. Which dimension was unbounded, and at what load. Section 1.1's question, answered for your system.

The measurement. What you measured, on how many cases, with what error bar. Section 14.1 applies to your write-up as much as to your gate.

The decision. One entry in section 17.2's format: decision, because, rejected, costs, revisit. This is the part that reads as engineering rather than enthusiasm.

The surprise. The thing that contradicted your expectation. If there isn't one, the write-up will read as a tutorial, because it probably is one.

The strongest thing you can show is a number that surprised you and the test you wrote to keep it true. Anyone can demonstrate a system that works on a good day. Very few people can tell you the rate at which theirs fails, why, and what would catch it regressing, and that gap is the whole subject of this book.

19.8 Four Agentic Applications, Specified

The projects so far are sized by calendar. This set is ordered by implementation difficulty instead: invoice processing, customer order inquiry, general customer service, and computer use. Each carries the same specification shape as sections 19.3 and 19.4, and together they trace the line this book has drawn from Chapter 9 onwards, from a workflow whose steps are known in advance to one the model must plan as it goes.

AR-1: Invoice processing

Objective. Replace manual review of invoices in a finance department by extracting the payment-critical fields and recording them, so payment is issued on time.

Item	Specification
Input	An invoice document (PDF)
Output	A database record with the extracted fields
Human in the loop	Not required
Difficulty	Low
Acceptance numbers	Field-level extraction accuracy on a labelled invoice set, and zero duplicate records from reprocessing

Functional requirements

ID	Requirement
AR-1.1	Accept a PDF document as input.
AR-1.2	Convert the PDF to model-readable structured text, for example markdown (6.2).
AR-1.3	Classify the document as invoice or non-invoice; non-invoice documents exit the workflow.
AR-1.4	Extract the required fields into a declared schema: biller name, biller address, amount due, due date (5.3).
AR-1.5	Persist the fields through a database tool, idempotently keyed so the same invoice never books twice (12.5).

Reference extraction, for the eval set: biller TechFlow Solutions, amount due $3,000, due date 20 August 2025.

AR-2: Customer order inquiry

Objective. Respond to basic customer inquiries about orders already placed.

Item	Specification
Input	A customer email (text)
Output	A draft response queued for human approval
Human in the loop	Required before sending (15.5)
Difficulty	Low to moderate
Acceptance numbers	Fraction of drafts approved without edits, and the reviewer disagreement rate

Functional requirements

ID	Requirement
AR-2.1	Ingest an inbound customer email.
AR-2.2	Extract the key information: customer name and order details.
AR-2.3	Verify the email actually concerns an order; otherwise exit the workflow.
AR-2.4	Query the orders database through a tool call to retrieve the relevant record.
AR-2.5	Draft a response grounded in the retrieved record, never in the model's guess.
AR-2.6	Place the draft in a human review queue; the wait releases the worker (10.7).
AR-2.7	Send only after review and approval.

AR-3: General customer service

Objective. Respond to an open-ended set of customer questions, not limited to existing orders. The required steps are not known in advance, so the agent must plan the sequence of actions per request. This is the boundary section 9.1 drew: reach for an agent only when the sequence genuinely cannot be known ahead of time.

Item	Specification
Input	An arbitrary customer query (text)
Output	A response, plus any required transactional side effects
Human in the loop	Recommended for transactional actions (9.7, 15.5)
Difficulty	High
Acceptance numbers	Compound success rate over a fixed task set (9.1), and zero unauthorised transactions on the adversarial cases

Functional requirements

ID	Requirement
AR-3.1	Plan a sequence of tool and database calls at runtime rather than following a fixed script, inside a bounded loop (9.2).
AR-3.2	Support multiple sequential inventory queries within a single request.
AR-3.3	Support purchase verification against customer history.
AR-3.4	Evaluate business policy rules before authorising an action (15.1).
AR-3.5	Execute approved transactional actions through tools, with irreversible ones behind approval (9.7).
AR-3.6	Update database state to reflect the action taken.

Worked scenarios. An inventory query, "do you have black jeans or blue jeans", requires a check of black jeans inventory, then a separate check of blue jeans inventory, then a consolidated response. A return request, "I would like to return the beach towel I bought", requires three planned steps: verify the customer purchased the towel, check the return policy (returns allowed within 30 days of purchase and only if unused), and if allowed, issue a return packing slip and set the record status to return pending.

AR-4: Computer use

Objective. Complete a task by operating a web browser, reading page content and acting on it.

Item	Specification
Input	A natural language task, plus browser access
Output	The task result, derived from live web pages
Human in the loop	Required; not reliable enough for mission-critical use
Difficulty	Very high
Acceptance numbers	Task completion rate over a fixed scenario set, with the failure modes labelled the way section 7.2 labels retrieval failures

Functional requirements

ID	Requirement
AR-4.1	Navigate websites autonomously, including clicking page elements and filling text fields, inside an isolation boundary with no ambient credentials (12.2).
AR-4.2	Reason over rendered page content to select the next action, treating page content as untrusted input (12.7).
AR-4.3	Recover from failure by switching to an alternative source when a site does not yield the needed result.
AR-4.4	Return the final answer to the user's original question, within the run's budget ceilings (9.6).

Reference scenario: check seat availability on two specific United Airlines flights from San Francisco to Washington DCA. The agent navigates the airline's site, encounters trouble, falls back to Google Flights to identify matching options, then returns to the airline's site and confirms availability. Two limitations to design around: a slow-loading page can leave the agent unable to interpret state, and many pages remain beyond its ability to parse accurately.

Triaging a candidate workflow

Use this to place any candidate on the ladder before committing to build it. It is Chapter 10's freedom argument in

triage form: every dimension that lands in the right-hand column is variance you will pay for.

Dimension	Easier	Harder
Process definition	Clear step-by-step process, or an existing standard operating procedure to codify	Steps not known ahead of time; the agent must plan or solve as it goes
Input modality	Text-only assets	Rich multimodal input: vision, audio, sound
Predictability	Deterministic, reliably repeatable	Unpredictable, lower reliability

Codifying an existing standard operating procedure into an agent is still substantial work, but it leads to an easier and more reliable implementation, because the procedure is the declared topology Chapter 10 argued for, written down by someone else.

Three requirements cut across all four applications rather than belonging to one. **CR-1, task decomposition:** the primary implementation skill is examining a complex workflow and identifying the individual discrete steps, so the workflow can be executed and evaluated one step at a time (14.5's per-node argument, applied at design time). **CR-2, tool interface inventory:** each application requires an explicit tool set, declared and scoped the way Chapter 12 scopes them; across the four examples these are a PDF-to-text conversion API, database read and write tools, a request-review tool, and browser control. **CR-3, guard classification:** AR-1 and AR-2 both include an early validity check, is this an invoice, is this about an order, that terminates the workflow when the input is out of scope; the guard is an abstention (6.9) placed at the cheapest possible point.

19.9 A Ten-Project Skills Ladder

The larger projects of sections 19.3 and 19.4 combine many capabilities at once, which is realistic and also why they are hard to start. This ladder runs the other way: ten projects, one capability per rung, ordered so each rung's machinery is assumed by the next. Several rungs are deliberately the seed of a larger project already specified, and the ladder says so, because climbing a rung and then growing it into the composite build is the natural path through this book. Every rung keeps the chapter's rule: it is not done when it runs, it is done when its number is measured.

SL-1: The structured output agent

Objective. Make one extraction task reliable rather than random: schema in, validated object out, every failure accounted for. This is the smallest project in the book that produces a defensible reliability number. *Difficulty: low. Acceptance: valid-output rate over two hundred real inputs, with the failure log as the second deliverable.*

Step	Implementation guideline
SL-1.1	Declare the schema with an explicit insufficient-evidence escape field (5.3).
SL-1.2	Request constrained decoding with the provider's structured-output mode, and validate the result anyway (5.3).
SL-1.3	Apply the same validation to tool results before they re-enter the prompt (12.7).
SL-1.4	On a parse or validation error, retry exactly once with the error shown to the model; one retry, not a loop (5.8, 9.8).
SL-1.5	Log every validation failure with its input; the log is next month's eval cases (14.2).

SL-2: The grounded RAG agent

Objective. Answers that carry their evidence or decline to exist. The seed of MP-1, at demo scale but with the honesty machinery intact. *Difficulty: low to moderate. Acceptance: accuracy-on-answered and coverage at your chosen relevance floor, reported together.*

Step	Implementation guideline
SL-2.1	Retrieve broad, show narrow (6.7), and assemble with resolved citations (6.8).
SL-2.2	Generate with sources named per claim, never as a bibliography at the end (6.8).
SL-2.3	Gate the answer on the relevance floor and citation resolution; below the floor, abstain with a route forward (6.9, 15.5).
SL-2.4	On weak retrieval, fall back to a second strategy, the lexical leg or a wider k, before falling back to abstention (7.5).
SL-2.5	Score a labelled case set with faithfulness beside accuracy, so a right answer with wrong evidence still fails (7.10).

SL-3: The bounded ReAct agent

Objective. An agent that cannot run away: every exit named, every ceiling enforced. The card version of chapter 9, and the rung everything after this stands on. *Difficulty: moderate. Acceptance: zero runs ending outside a named Stop reason over a fixed task set, with compound success rate reported per section 9.1.*

Step	Implementation guideline
SL-3.1	Build the observe, think, act, append loop with the six-exit Stop enum (9.2).
SL-3.2	Enforce all three ceilings, steps, cost and deadline, checked before the call, not after (9.6).
SL-3.3	Detect no-progress by call signature, and stop on repetition (9.2).
SL-3.4	Reflect only on mechanical failures where an external system supplied the error, once (9.8).
SL-3.5	On any budget stop, return the partial result and the reason, degraded rather than empty (9.6, 15.3).

SL-4: The multi-tool orchestrator

Objective. One agent holding many tools without holding all the permissions: a registry, scoped advertisement, and safe parallelism. *Difficulty: moderate. Acceptance: on an adversarial task set, zero calls to tools outside the advertised scope, with the denied-call log as evidence (9.10).*

Step	Implementation guideline
SL-4.1	Build the tool registry as an allowlist with pinned schema and description hashes (12.1, 12.9).
SL-4.2	Advertise per step only the intersection of node, policy and tenant tool sets (12.4).
SL-4.3	Route by capability through schemas that constrain, names stating scope, enums, bounds (9.3).
SL-4.4	Execute read-only calls concurrently; serialise anything with side effects through the authorise chokepoint (9.2, 9.7).
SL-4.5	Resolve conflicts by idempotency key, so a retried side effect lands once (12.5).

SL-5: The memory-enabled assistant

Objective. An assistant that remembers users across sessions with provenance, the seed of QP-3. *Difficulty: moderate. Acceptance: recall on planted multi-session facts, and zero superseded values surfacing (11.10).*

Step	Implementation guideline
SL-5.1	Run the rolling buffer for working memory, keeping the first user turn and whole turns (11.2).
SL-5.2	Extract durable facts through the two-gate write path, asynchronously, off the critical path (11.5).
SL-5.3	Prefer typed fact extraction to summarisation for compression; summaries forget silently (11.3, 11.4).
SL-5.4	Retrieve memory as recent plus relevant plus pinned into a bounded region that competes with evidence (11.6).
SL-5.5	Synchronise across sessions by supersession ordered on source turn time, never overwrite (11.7).

SL-6: The approval agent

Objective. An agent that knows what it is not entitled to decide: risk-tiered actions, an interrupt that releases the worker, and a record a non-engineer can read. The seed of AR-2's review queue. *Difficulty: moderate. Acceptance: one hundred percent of IRREVERSIBLE actions carrying an approval record, and a reviewer disagreement rate that is not zero (15.5).*

Step	Implementation guideline
SL-6.1	Tier every tool by what a wrong call costs, READ, WRITE_REVERSIBLE, IRREVERSIBLE (9.7).
SL-6.2	Detect uncertainty worth escalating: low grounding scores and low-confidence classifications route to a person rather than a guess (6.9, 13.7).
SL-6.3	Pause as an interrupt, not a held thread: persist state, release the worker (10.7).
SL-6.4	Resume from the checkpoint with the human's decision in state, repeating no paid work (10.5).
SL-6.5	Write the audit trail from the same chokepoints that produced the verdicts (15.8).

SL-7: The cost-aware router

Objective. Spend as an engineered quantity: routed tiers, early exits, and attribution per decision. The weekend router-economics project grown into a component. *Difficulty: moderate. Acceptance: your own before-and-after spend with quality held, measured, not the folklore range; section 3.6's report format is the deliverable.*

Step	Implementation guideline
SL-7.1	Budget tokens per task type, with max output set from the task's contract, not the default (2.2, 5.3).
SL-7.2	Route by a static task-type table first; earn a learned difficulty classifier later (3.6, 4.5).
SL-7.3	Exit early on confidence, with sampled verification against the strong tier to keep the exit honest (4.5).
SL-7.4	Attribute cost per decision from the trace record, including cache hit rate (16.2, 3.6).
SL-7.5	Enforce the ledger: downgrade before refusing when a budget is near (3.6).

SL-8: The event-triggered automation agent

Objective. Automation that survives redelivery, the queue discipline of chapter 8 wrapped around one real workflow. The seed of AR-1. *Difficulty: moderate to high. Acceptance: a chaos test, kill workers mid-run for an afternoon, ending with zero duplicated side effects and zero lost jobs.*

Step	Implementation guideline
SL-8.1	Accept triggers through the 202 pattern: enqueue with an idempotency key derived from the work, return immediately (8.6).
SL-8.2	Carry identifiers, not payloads, with prompt and index versions pinned in the message (8.4).
SL-8.3	Make every handler idempotent; the broker's dedup is not your correctness (8.10).
SL-8.4	Classify failures as retryable or deterministic; only the former re-enter the queue (8.10).
SL-8.5	Dead-letter what exhausts its retries, and mine the dead-letter queue for eval cases (8.10, 14.2).

SL-9: The debate panel

Objective. Multiple independent proposals, a critic, and a synthesis, built as a judge panel with declared boundaries rather than a swarm. Section 9.9's pattern-zoo warning applies in full: every seam here must name what it protects, and the answer is independence of the proposals and externality of the critic. *Difficulty: high. Acceptance: the panel beats your best single-agent baseline on the same eval set by a margin larger than section 14.1's error bar, or you ship the single agent.*

Step	Implementation guideline
SL-9.1	Fan out N proposers with genuinely different framings as parallel read-only legs with per-leg deadlines (10.8).
SL-9.2	Keep proposers context-isolated from each other; agreement is only evidence when it is independent (7.6, 9.9).
SL-9.3	Judge pairwise with position swapped, by a critic from outside the proposers' context (14.4, 9.8).
SL-9.4	Aggregate by declared rule, vote or score threshold, with disagreement carried as a confidence signal into the output (10.8).
SL-9.5	Cap the rounds; a debate is a cycle, and an unguarded cycle is a bill (10.9).

SL-10: The self-improving agent

Objective. The evaluator-optimizer pattern run honestly: execute, evaluate externally, regenerate under constraint, and prove the improvement is real. The top rung because it composes everything below it. *Difficulty: high. Acceptance: a quality-over-time curve on a fixed holdout set that rises across iterations and survives section 14.1's error bars; an improvement inside the noise is not an improvement.*

Step	Implementation guideline
SL-10.1	Execute against a fixed task set with the SL-3 loop; the case set is the deliverable (14.2).
SL-10.2	Evaluate with deterministic scorers first and a calibrated judge only where they cannot reach; the judge is from a different family and pinned in the fingerprint (14.3, 14.4).
SL-10.3	Critique reasoning only through the external verdicts; a model grading its own transcript is section 9.8's self-approval (9.8).
SL-10.4	Regenerate with the critique as explicit constraints in the prompt, versioned per section 5.8, one bounded revision cycle per iteration (5.8, 10.9).
SL-10.5	Log the improvement metrics per iteration on a holdout the optimiser never sees, and gate promotion on them (14.5, 14.6).

A closing suggestion

Build the smallest of these that involves a real person asking real questions of real documents where a wrong answer costs something. Instrument it before you optimise it. Label fifty failures before you change anything. Then re-read whichever chapter your failure distribution points at, because it will point at one, and it will not be the chapter you expected.