

For people starting out in Salesforce

AI as your *technical mentor*

How to understand requirements, investigate the environment, deliver with quality and grow using AI, without outsourcing your own thinking.

CORE PRINCIPLE

AI expands your ability to think, investigate, learn and review. Reading the requirement, knowing the environment, validating the solution and owning the delivery are still your job.

BEFORE YOU START

A new request just landed. Now what?

This guide is for junior developers, people at the start of a tech career and anyone who is new to Salesforce, in a very common situation: a functional spec, a ticket, a user story, a requirement or a bug fix arrives, and you need to turn a business text into a safe technical change.

The goal goes beyond "asking AI for code". Here you will find a way of working where AI acts as a **mentor, reviewer, explainer and investigation partner**. The developer is still responsible for understanding, questioning, testing, deciding and explaining the solution.

This version also includes practices from complex Salesforce and OmniStudio work: separating the literal requirement from hypotheses, investigating the AS-IS scenario before designing the TO-BE, tracing dependencies end to end, turning questions into objective checks and feeding the AI with real evidence as the investigation moves forward.

DEFINITION OF FINISHED

If you cannot explain what the solution does, why it was chosen, what it impacts and how it was validated, the work is not finished yet, even if the code seems to work.

THE PATH

Follow the flow in chapter 03 every time a new request arrives. The other chapters go deeper into investigation, prompts, quality, security and career growth.

- | | |
|--|---|
| 01 The right role for AI | 08 Validating code, config and architecture |
| 02 Before you open the AI | 09 Tests, review and Definition of Done |
| 03 From request to delivery | 10 Security, data and limits |
| 3A Analysis script for a request | 11 When to ask a human for help |
| 04 Investigating a Salesforce org | 12 A complete example |
| 05 Talking to AI without letting it invent | 13 Pocket checklist |
| 06 Prompts for everyday work | 14 Junior growth path |
| 07 Learning while you build | 15 Official sources |

01

CHAPTER 01

The right role for AI in development

AI speeds up understanding a lot: it explains concepts, suggests investigation paths, writes queries, reviews hypotheses, spots risks, proposes tests and discusses alternatives. The problem starts when it becomes the developer's "outsourced brain".

HEALTHY USE OF AI	USE THAT BLOCKS YOUR GROWTH
Explaining a concept and then asking you to explain it back.	Generating a whole solution and copying it without understanding.
Helping you write questions to investigate the org.	Accepting object, field, class or process names that the AI invented.
Reviewing a hypothesis you created.	Asking "what is the solution?" before understanding the problem.
Writing a SOQL query or command and explaining each part.	Running DML scripts because "the AI said it is correct".
Comparing alternatives and showing risks.	Using the most convincing answer as proof.
Reviewing your code and pointing out problems.	Asking for code without giving context about the environment.
Helping you create test cases and negative scenarios.	Treating "it compiled" as "it is correct".

REQUIRED MINDSET

Doubting AI means treating it like any other technical source: asking where a statement came from, checking it against the real environment, reading the official documentation and testing before you conclude. The tool stays on the table, under your supervision.

Four roles AI can play

i

MENTOR

Explains concepts, asks questions and helps you build your reasoning.

ii

ASSISTED INVESTIGATOR

Helps you design queries, searches, commands and diagnostic paths.

iii

REVIEWER

Critiques the solution, code, architecture, tests and documentation.

iv

ACCELERATOR

Automates mechanical tasks once the problem is already understood.

02

CHAPTER 02

What to do before you open the AI

The first reading of the request, whether it is a ticket, a user story or a functional spec, must be human. If you hand the text to the AI before forming your own understanding, you skip exactly the step that builds analytical skill.

Manual reading in three passes

- 01 **First pass.** Read from start to end without trying to solve anything. The goal is to find out which business problem is being described.
- 02 **Second pass.** Mark users, inputs, outputs, rules, exceptions, screens, integrations, data, acceptance criteria and words you do not understand.
- 03 **Third pass.** Write in your own words what you believe needs to change, and list everything you still do not know.

BEFORE THE AI, WRITE THESE FIVE LINES

1. The problem is...
2. Today it seems to work like this...
3. The requested change seems to be...
4. I still do not understand...
5. To confirm it, I need to look at...

Minimum manual questions

- What is the current behavior?
- What is the expected behavior?
- Who runs this journey, and when?
- What data goes in, and what result should come out?
- Is there an explicit acceptance criterion?
- Are there exceptions, conditional rules or alternative scenarios?
- Does the request talk about a new component or a change to something that already exists?
- Which screen, object, field, product, integration or process names appear?
- What is a confirmed requirement, and what am I only imagining?
- What would I ask a functional analyst, tech lead or business user?

Classify the question before you try to answer it

Not every question should go to the AI. Classifying it stops you from asking the model for an answer that only the business or the environment can give.

TYPE	EXAMPLE	RIGHT SOURCE FOR THE ANSWER
Business	"Does the rule also apply to cancelled contracts?"	Functional analyst, PO, business user or the requirement owner.
Technical	"How does an Integration Procedure handle this input?"	Official documentation, existing code and configuration, tests, and AI as support.
Environment	"Is there already a Flow or Apex updating this field?"	The org, retrieved metadata, repository, logs and project searches.

SIMPLE RULE

Business questions should not be "solved" by AI. Environment questions should not be answered from memory or assumptions. Technical questions can use AI, but the answer must be checked against official documentation and the real patterns of the project.

03

CHAPTER 03

The complete flow: from request to delivery

This flow should become a habit. It works for bug fixes, improvements, new features, integrations, automation, LWC, Apex, OmniStudio, Flow or configuration.

- 01 Read.** Understand the problem manually and write down your questions.
- 02 Structure.** Separate facts, hypotheses, gaps and acceptance criteria.
- 03 Give the AI context.** Give the AI only what you have already observed, and require it not to invent.
- 04 Investigate.** Map what exists in the org and collect evidence.
- 05 Analyze impact.** Find dependencies, consumers, data, permissions, automations and integrations.
- 06 Design.** Compare options and choose the smallest safe change.
- 07 Build in small parts.** Build small increments, understanding each change.
- 08 Test.** Validate the happy path, negative cases, regression and security.
- 09 Review.** Use AI, Code Analyzer and human review when it applies.
- 10 Explain and record.** Be able to tell the story of the solution and keep what you learned.

The table that prevents hallucination

During the whole task, keep a simple table. It forces you and the AI to recognize how certain each piece of information is.

CATEGORY	WHAT IT MEANS	EXAMPLE	ACTION
FACT	It is explicit in the request or was confirmed by the person responsible.	"The field must be required when X."	Implement or validate directly.
EVIDENCE	It was observed in the environment, metadata, log, query or test.	"The OmniScript calls IP Y."	Keep it as proof of the current behavior.
HYPOTHESIS	A possible explanation, not confirmed yet.	"Maybe the value is calculated in the IP."	Investigate before deciding.

CATEGORY	WHAT IT MEANS	EXAMPLE	ACTION
QUESTION	Information you need that is still missing.	"Does this rule also apply to renewals?"	Ask or research.
DECISION	A technical choice made after the evidence.	"The change will live in the IP so the rule is not duplicated."	Record the reason and the impact.

Triage your open items: not everything becomes a question

A mature practice is to review your list of questions all the time and remove everything already answered by the requirement, by the source or by the investigation. The goal is to keep only what is really still unanswered.

STATUS	WHAT IT MEANS	NEXT ACTION
Answered by the requirement	The request already defines the rule literally.	Do not ask again. Turn it into an acceptance criterion or a test.
Answered by the source	Code, metadata, log or configuration already proved the current behavior.	Record the evidence and move on.
Search the org	The answer exists technically, but it has not been found yet.	Create an objective check: search, query, retrieve, log or inspection.
Functional question	The answer needs a business decision or clarification.	Take an objective question to the person responsible.
Technical decision	The requirement says what must happen, but not how to implement it.	Compare alternatives, existing patterns, risks and impact before choosing.

A GOOD QUESTION REMOVES UNCERTAINTY

Before you send a question to someone, ask yourself: is this already answered somewhere? Is it a business rule, environment information or a technical decision? What evidence would close this question?

Evidence notebook for the request

In big requests, keep a short record of what was confirmed. It stops you from going back to old hypotheses, helps the AI follow the investigation and makes code review, QA and handoff easier.

QUESTION	EVIDENCE COLLECTED	ALLOWED CONCLUSION	DOES NOT PROVE YET
Who calls this calculation?	Project search + retrieved metadata.	There is a direct reference in component X.	That it is the only path at runtime.

QUESTION	EVIDENCE COLLECTED	ALLOWED CONCLUSION	DOES NOT PROVE YET
Where is the value saved?	Payload before/after + queried record.	Field Y receives the value after step Z.	The behavior when the record is reprocessed.

3A

CHAPTER 3A

Operational script to analyze a new request

Read, form your own understanding, compare it with the document, turn questions into checkable questions, investigate the environment, go back to the open items with evidence, refine the wording, and only then close the technical scope. The key idea is to work in rounds, without trying to solve everything at once.

MAIN RULE

Your first list of questions is a draft, never the final list. It must be cleaned up as new evidence appears.

ROUND 0

Create the request notebook

Before your first conversation with the AI, open a single place to follow the analysis: a spreadsheet, a document or a notes page. The goal is to stop facts, hypotheses and decisions from getting mixed up over the days.

- Keep the original text of the request untouched. Do not rewrite the requirement and then treat your own interpretation as if it were the original.
- Create an area called **"My understanding"** and write what you believe will change.
- Create an area called **"My questions"**. Write, in first person, what really was not clear to you.
- Create an **evidence** area: query, metadata, code snippet, payload, log, screenshot or test that answered a question.
- Create a **decisions** area: functional, technical and architectural. Record who decided and why, when it matters.

Why it works: when you keep the original document separate from your reading, you can go back and ask "is this really written here, or did I conclude it?".

ROUND 1

Read on your own and write what you think you will do

After the general reading, turn the requirement into your own explanation. Do not look for a solution yet. The goal is to test whether you understood the problem.

- Write the current behavior you believe exists.
- Write the expected behavior described in the request.
- List what seems new, what seems to be a change to existing behavior, and what apparently must stay the same.
- List the components, objects, fields, screens and rules mentioned literally in the request.

- Mark with **HYPOTHESIS** any component you only imagine is involved.
- List your questions before you ask the AI anything.

A good test is being able to say out loud: **"today the problem is X; after this request we expect Y; I still do not know Z"**.

ROUND 2

Use the AI to challenge your reading

The first useful thing the AI can do is point out gaps in your understanding. Give it the request, your summary and your questions. Ask for a comparison, not an implementation.

PROMPT Challenge my reading of the request

I will give you the request, my understanding and my questions.
Do not propose a solution yet. Compare my understanding with the text and return:
(1) what is literally defined;
(2) what I interpreted correctly;
(3) what I am inferring without evidence;
(4) what is ambiguous or contradictory;
(5) what needs to be investigated in the environment;
(6) what needs a functional decision.
For each conclusion, tell me which evidence you are relying on.

ROUND 3

Clean up the open items before investigating

This is one of the most important habits. Before keeping any item on the open list, try to destroy it. An item only survives if you really could not find the answer.

- Look in the request itself first. Many questions come up because a detail is answered in another functional requirement, acceptance criterion, table, attachment or example.
- Look in the analysis history. Do not keep a question that was already answered in an earlier round.
- Check whether two items are, in practice, the same question written in different ways. If they are, merge them or remove the duplicate.
- If an item mixes two subjects, split it. Compound questions usually get incomplete answers.
- If the item does not need a functional answer but represents something that will need technical handling, reclassify it as a technical impact.
- If the request already made the decision but the text is badly written or incomplete, record it as "decision defined, needs to be made explicit in the requirement", not as an open decision.

Refinement examples: separate "precision of the discount percentage" from "technical limit of decimal places in the field"; separate "quantity entered by the user" from "quantity used in the calculation"; treat the rule for items already on promotion as a sub-item of the discount rule; remove a duplicate item when the concern was already covered by another question, and keep only the related technical impact.

Ask atomic questions

A good open item allows an objective answer. If the question has two or three "ands", it probably needs to be split.

QUESTION	
BAD	"What happens to the discount, the quantity tier and the total when the sales rep edits the quote?"
BETTER 1	"Which discount percentage must be saved on each quote line item?"
BETTER 2	"How is the discount tier defined from the quantity?"
BETTER 3	"Which rule applies when the item already has a promotional price?"

ROUND 4

Turn each environment question into a runnable investigation

"Needs to be validated in the org" is not a good enough answer. Every validation must become an objective procedure.

- › What exactly do I need to find out?
- › Where will I look first?
- › Which query, search, metadata, log, payload or test can answer it?
- › Which result would confirm hypothesis A?
- › Which result would rule out hypothesis A?
- › What would this evidence still not prove, even if I find it?

EXAMPLE

Instead of "validate whether the field is part of the calculation", write: search for the API Name in the project; retrieve the current metadata; find references in Flow, Apex and OmniStudio; run a controlled scenario and look at the payload or log to confirm it is used at runtime.

When the AI says "this needs validation", answer with four questions

- 01 Validate what, exactly?
- 02 Where can I validate it, in Salesforce or in the project?
- 03 What evidence should I bring back?
- 04 Which result closes the question, and which result opens a new investigation?

This way, "needs validation" stops being a parking lot for vague questions.

ROUND 5

Rebuild the AS-IS and follow the data end to end

Before designing the TO-BE, prove how the journey works today. In requests about calculation, price, discount, quantity, dates or configuration, follow the data as if you were tracking a package moving through the system.

- › Where does the user enter or change the data?
- › In which structure does it first appear: attribute, JSON, field, variable, table row?
- › Which component receives the value next?
- › Is there any transformation, aggregation, calculation or conversion?
- › Where is the value saved?
- › Who reads it later: calculation logic, Flow, Apex, Integration Procedure, Data Mapper, LWC, integration?
- › What happens on edit, recalculation, reopening, copy, renewal or reprocessing?
- › Is there a legacy path that must keep working when the new condition does not apply?

IMPORTANT

Do not invent the next link. If you know an OmniScript calls something, find the real name of that call. If you know an IP saves data, find which element and which object. Every link needs evidence.

ROUND 6

Go back to the open items with evidence and your own comments

After the technical discovery, do not jump straight into code. Do a second round of analysis: reread each open item, compare it with what you found, and write your updated question in your own words.

- If the evidence answered it, close the item and record the evidence.
- If the evidence answered only part of it, rewrite the question so it keeps only what is missing.
- If you found a contradiction between the requirement and the environment, record the difference explicitly.
- If you misunderstood the question, fix the text. Do not keep a bad item just because it already had a number.
- If an item became a technical impact, move it to that category.
- If the item depends on a human decision, write the question so that the analyst or PO can answer it objectively.

This moment is especially valuable for learning: first you see the AI's reading, then you see the environment, and then you write your own interpretation. The junior stops only consuming answers and starts building a technical point of view.

Your open-items sheet must be alive

The sheet works as the reasoning control of the request, much more than a list of questions. Update it every round. A useful structure:

COLUMN	WHAT TO RECORD
ID	A stable identifier while the item exists.
Topic	A short subject, for example "discount", "persistence", "calculation", "UI".
Request excerpt	The text that started the analysis.
My understanding	How you interpreted that excerpt.
My question	A question written by you, in direct language.
Classification	Requirement / environment / functional / technical decision / technical impact.
Current evidence	Query, code, metadata, payload, log, test or functional answer.
What the evidence proves	The allowed conclusion.
What it does not prove	The limit of the conclusion.
Next action	Exactly the next step.
Status	Open, investigating, answered by the requirement, answered by evidence, waiting for functional answer, waiting for technical decision, duplicate/removed.
Impact	Where this definition affects the implementation or the tests.

RULE

Never leave a cell with only "validate". Write validate what, where and with which evidence.

ROUND 7

Close and reclassify every open item

Before you say the analysis is done, every question must be in a state that makes sense:

STATUS	WHEN TO USE IT
CLOSED BY REQUIREMENT	The answer was already in the requirement.
CLOSED BY EVIDENCE	The org, code, metadata, log, payload or test answered it.
WAITING FOR FUNCTIONAL	It depends on a business rule that is not defined.

STATUS	WHEN TO USE IT
WAITING FOR TECH DECISION	There are valid alternatives and the team needs to choose.
OPEN INVESTIGATION	There is still a concrete discovery action to do.
CONFIRMED TECH IMPACT	Not a question: something the solution must take into account.
DUPLICATE / REMOVED	The item was absorbed by another one or no longer makes sense.

This avoids a huge list of "open items" where half are already solved and nobody knows what is really blocking development.

ROUND 8

Write the implementation brief before coding

When the critical items are closed or consciously accepted, write a one-page implementation brief. If you cannot write this summary, you do not understand the request well enough yet.

- ☐ Goal of the change in simple language.
- ☐ Confirmed AS-IS.
- ☐ TO-BE confirmed by the request and by the decisions made.
- ☐ Components that will change.
- ☐ Components that may be impacted without being changed directly.
- ☐ Incoming data, transformations and where it is saved.
- ☐ Rules, exceptions and legacy behavior that must stay the same.
- ☐ Remaining open items, and whether they block development or not.
- ☐ Implementation strategy in small, observable steps.
- ☐ Test scenarios: functional, regression, permissions, recalculation/re-entry and relevant limits.

The ritual to repeat for every new request

- 01 Read it on your own and write what you understood.
- 02 List your questions without asking for a solution.
- 03 Use the AI to challenge your reading.
- 04 Remove questions that are already answered or duplicated.
- 05 Split compound questions.
- 06 Classify what is requirement, environment, functional, technical decision or impact.
- 07 Turn each environment question into a checkable action.

- 08 Rebuild the AS-IS and trace the data end to end.
- 09 Bring the evidence back to the AI and update the hypotheses.
- 10 Rewrite the open items with your own comments.
- 11 Close or reclassify every item.
- 12 Write the implementation brief.
- 13 Build in small steps, understanding each change.
- 14 Test the requirement, regression and alternative paths.
- 15 Explain the solution without looking at the AI.

SIGN OF MATURITY

Using AI to speed up investigation and review without outsourcing your technical judgment.

04

CHAPTER 04

How to investigate a Salesforce org

The AI does not know your org. Even if it knows Salesforce, it does not know which customizations, metadata names, old rules, integrations, permissions, packages or architecture decisions exist in your company. That is why the investigation must produce real context.

Start with the journey, not the code

- Where does the user start the action?
- Which screen, OmniScript, Flow, LWC, Quick Action, FlexCard or page is involved?
- Which component receives the first input?
- Which calls happen next?
- Which objects and fields are read or written?
- Is there an external integration?
- Where do the business rules live?
- Who consumes the result?
- Are profiles, permission sets, sharing or FLS involved?
- Are there parallel automations that also update the same record?

AS-IS first, then TO-BE

Before discussing how to build it, rebuild how the journey works today. In complex projects, the requirement usually mentions only one visible piece, while the real behavior crosses several layers. If the AS-IS is wrong, the solution will be designed on a false premise.

KEY QUESTION

Instead of asking the AI "where is this rule in my org?", ask "how can I prove where this rule is?". The answer should become a search, a query, a metadata inspection, a log, a payload, a test or a functional question.

Trace the flow end to end

A good technical map follows the data and the execution. Not every request has all the layers below, but the investigation should continue until you know where the value is born, where it changes, where it is used and where it is saved.

01 User action Button, screen, edit, simulation, save, reprocessing.

02 Interface LWC, OmniScript, FlexCard, Screen Flow, Lightning page.

- 03 **Orchestration** Integration Procedure, Flow, Remote Action, action chain.
- 04 **Rule / calculation** Apex, Expression Set, Decision Matrix, formulas, Data Mapper transforms.
- 05 **Data** Objects, fields, relationships and the records involved (for example Opportunity, Quote, Order).
- 06 **Persistence** Where the final result or configuration is saved.
- 07 **Consumers** Who uses this value later: UI, next calculation, document, integration, approval.
- 08 **Reprocessing** What happens on edit, recalculation, reopening, cloning or with old records.
- 09 **Async / history** Jobs, batch/queueable, snapshots, versions, historical data and timing.

Investigation map by layer

LAYER	WHAT TO LOOK FOR	USEFUL EVIDENCE
Interface	Lightning page, LWC, FlexCard, OmniScript, Screen Flow.	Technical name, screenshot, XML/JSON, inputs and outputs.
Orchestration	Flow, Integration Procedure, OmniScript actions, Apex.	Execution order, conditions, payloads, logs.
Data	Objects, fields, relationships, record types, metadata/configuration.	SOQL, Object Manager, schema, anonymized data sample.
Business rule	Formulas, Apex, Decision Matrix, Calculation Procedure, Validation Rules, Flows.	Metadata excerpts, criteria and scenarios.
Integration	Named Credentials, APIs, events, callouts, middleware.	Contract, request/response, status, logs, error handling.
Security	Permission Sets, Profiles, FLS, sharing, running user.	Effective permissions and execution context.
Dependencies	Components that call or are called.	Project search, Where is this used, Tooling/Metadata APIs, references.
Quality	Tests, validations, static analysis, logs.	Apex tests, Jest, Code Analyzer, debug logs, QA evidence.

Practical investigation tools

- **Project search:** use "Find in Files" with the API Name of a field, object, class, LWC, Integration Procedure or OmniScript.
- **Object Manager and Setup:** confirm field type, relationships, formulas, rules, permissions and available features.
- **SOQL:** investigate real data and relationships. The AI can help you write the query, but you review the object, fields, filters and volume.
- **Tooling API and metadata:** useful for investigating classes, triggers and components when the question is about configuration and code, not business records.
- **Salesforce CLI:** queries data, retrieves metadata and lets you analyze the project locally.
- **Logs and payloads:** use execution evidence to understand the real path, especially in OmniStudio and integrations.
- **Salesforce Code Analyzer:** static analysis as one layer of review, never a replacement for tests.

IMPORTANT WARNING

"I did not find it" does not mean "it does not exist". It can be outside the local project, in a managed package, in metadata not yet retrieved, in an external integration, in dynamic configuration or in something you do not have access to. Record that limitation.

The assisted investigation cycle

- 01 **Write a checkable question:** "where does this value come from?", "who updates this field?", "what calls this class?".
- 02 **Ask the AI for the investigation path, not the conclusion:** search, query, command, log, payload or metadata to inspect.
- 03 **Run it yourself** in the right environment and read the result.
- 04 **Go back to the AI with the real evidence,** including what the result proves and what it does not prove yet.
- 05 **Update the flow map** and remove hypotheses that no longer make sense.
- 06 **Only then** write the next question.

INVESTIGATION IS ITERATIVE

The conversation context should grow with new facts. The AI does not "learn your org" by magic: you make it useful by bringing back query results, real metadata excerpts, payloads, logs and confirmed functional decisions.

When documentation, local code and runtime do not match

Treat a mismatch as a reason to investigate, never as permission to pick the version that seems most convenient. Confirm the branch/commit, the active version, the current org metadata and execution evidence. An old file in the repository may not represent runtime; a screen may show an active version different from the documentation; a requirement may describe the desired state, not what exists today.

- Record which source you checked and, when relevant, its version or date.
- Retrieve the current metadata when there is doubt about what is really deployed.
- Use a log, payload or test to confirm runtime behavior when configuration alone is not enough.
- Do not "fix" the mismatch by inference. First prove which state is real, then decide with the team.

Useful command examples

TERMINAL Query data with Salesforce CLI

```
sf data query --query "SELECT Id, Name FROM Account LIMIT 10" --target-org [my-org]
```

This command is only an example. Adjust the object, fields, filters and org. For Tooling API objects, the CLI has the `--use-tooling-api` option.

TERMINAL Retrieve specific metadata

```
sf project retrieve start --metadata ApexClass:[MyClass] --target-org [my-org]
```

Retrieve only what you need when possible, and use version control to compare changes.

05

CHAPTER 05

How to talk to AI without letting it invent

A good prompt is a way to give context and impose a working method, with no magic words. For development, it must make clear the AI's role, what is already known, what is still unknown, which task will be done and which limits cannot be crossed.

The AI answer rule

Ask the AI to classify what it says. This format works very well:

LABEL	MEANING
CONTEXT FACT	Only what you provided or showed.
GENERAL KNOWLEDGE	Documented platform behavior, ideally with an official source.
HYPOTHESIS	A possibility that depends on validation in the org.
NEEDS VALIDATION	Something the AI cannot confirm with the material available.
QUESTION	Information you need to find or ask before moving forward.

PROMPT BASE Turn the AI into your technical mentor

Act as my Salesforce technical mentor. I am a junior developer and I want to learn while I work on this request.

Mandatory rules:

1. Do not invent objects, fields, classes, components, behaviors of my org or requirements that I did not give you.
2. Clearly separate: CONTEXT FACT, GENERAL KNOWLEDGE, HYPOTHESIS, NEEDS VALIDATION and QUESTION.
3. If context is missing, ask questions before concluding.
4. Do not treat a hypothesis as a fact.
5. Before suggesting code, help me understand the current flow, the impact and the best way to investigate.
6. When you suggest a query, command or code, explain it line by line or block by block, and tell me what to expect from the result.
7. Ask questions to test my understanding. Do not do everything for me.
8. If there is more than one solution, compare alternatives, risks and trade-offs.
9. Always tell me what I need to validate in the real environment and in the official documentation.
10. If you do not know, say you do not know.

My goal is to be able to explain the solution on my own at the end.

Good context versus bad context

BAD	GOOD
"Write the code for this request."	"I read the request. I understood A, B and C. I have questions D and E. In the environment I found components X and Y. First help me validate the flow and build an investigation plan."
"Why is it not working?"	"The error happens on save. Input: X. Expected result: Y. Actual result: Z. Relevant log: [...]. I already checked A and B. Which hypotheses should I test, in order?"
"Write a SOQL."	"I need to confirm whether records of object X have Y when Z. These fields exist: A, B, C. Write a read-only SOQL query and explain the filter."
"Fix my Apex."	"This method should do X. The test fails in scenario Y. Here are the class, the test and the message. First explain the likely cause, then propose the smallest fix."

06

CHAPTER 06

Practical prompts for everyday work

The prompts below are templates to fill in with evidence from the environment. No template replaces the investigation. The parts in [brackets] are what you fill in.

PROMPT A After the manual reading of the request

I read the request manually and my first understanding is:

- Business problem: [write it]
- Current behavior: [write it or "I do not know yet"]
- Expected behavior: [write it]
- Acceptance criteria I found: [list them]
- Questions I already have: [list them]

I want you to:

1. Review my understanding without inventing information.
2. Point out ambiguities and requirements that seem to be missing.
3. Ask the questions I should answer before thinking about a solution.
4. Build a list of things I need to investigate in Salesforce.
5. Do not propose code yet.

PROMPT B Build the investigation plan

Context confirmed so far: [paste only facts and evidence].

Components I already found: [list them].

What I still need to find out: [list it].

Build an investigation plan in order, starting with the least invasive path. For each step, tell me:

- the goal;
- where to look;
- query/command only if needed;
- which evidence I should collect;
- how to interpret the result;
- what the next step would be depending on the result.

Do not assume that components I did not mention exist.

PROMPT C Write a SOQL query and learn from it

I need to investigate [goal].

Confirmed object: [API Name].

Confirmed fields: [list].

Business filter: [condition].

Write a read-only SOQL query.

Then explain each clause, possible limitations, expected volume and what the result proves or does not prove.

If a field you need is missing, do not invent it: ask me.

PROMPT D Analyze an OmniStudio payload

I am analyzing this OmniStudio journey.

Goal of the step: [describe it].

Current component: [OmniScript/IP/Data Mapper/FlexCard/LWC].

Real anonymized input: [paste it].

Real anonymized output: [paste it].

Expected behavior: [describe it].

Compare input and output.

Separate evidence from hypothesis.

Point out where the value may be created, transformed or lost, but do not invent element names.

Suggest the next evidence I should collect.

PROMPT E Review Apex or LWC

This code is mine and I need to understand and improve it.

Functional goal: [describe it].

Code: [paste it].

Test/error: [paste it].

Do the review in four steps:

1. Explain the code flow in simple language.
2. Point out problems in logic, security, limits, performance, readability and testability.
3. Separate confirmed errors from optional improvements.
4. Propose the smallest change needed and explain why.

Do not rewrite everything without a reason.

PROMPT F Create test cases

Confirmed requirement: [describe it].

Business rule: [describe it].

Impacted components: [list them].

Create a test matrix with:

- scenario;
- precondition;
- input;
- steps;
- expected result;
- risk covered.

Include the happy path, limits, null/empty values, permissions, regression and integration errors when they apply.

PROMPT G When you are stuck

I am stuck, but I do not want you to solve it for me.

What I already know: [list it].

What I already tried: [list it].

Results I got: [list them].

My current hypothesis: [write it].

Ask me at most 5 diagnostic questions, ordered by how many hypotheses each one can rule out.

After each answer, update the hypothesis.

Do not jump straight to code.

PROMPT H Close the request and learn from it

The request is done. I want to consolidate what I learned.

Problem: [summarize it].

Solution applied: [summarize it].

Changed components: [list them].

Tests: [list them].

Ask me 8 questions as if this were a code review or a technical interview, to confirm that I really understood.

Then write a learning summary with: concept, decision, mistake I avoided, reusable pattern and a point I still need to study.

PROMPT I Turn open items into objective investigation

These are my current open items: [list them].

These rules are already literally answered by the requirement: [list].

This is the evidence I already collected in the environment: [list].

Review the list and classify each item as:

- ALREADY ANSWERED BY THE REQUIREMENT;
- ALREADY ANSWERED BY EVIDENCE;
- SEARCH THE ORG;
- FUNCTIONAL QUESTION;
- TECHNICAL DECISION.

Remove redundant questions. For each "SEARCH THE ORG" item, turn the question into a concrete check and tell me which evidence would close it. Do not answer by inference.

PROMPT J Trace an OmniStudio/Salesforce chain

I want to trace a behavior end to end without assuming the architecture.

Confirmed starting point: [button/screen/OmniScript/LWC/etc.].

Value or rule I am following: [describe it].

Components already confirmed: [list them].

Current evidence: [list it].

Help me build the investigation chain:

UI → orchestration → rule/calculation → data → persistence → consumers → reprocessing.

For each unknown link, tell me exactly what to search for in VS Code/org/log/payload.

Do not invent the next component; write the search that can find it.

PROMPT K Write a safe investigation script

I need a script to investigate [goal] in a sandbox.

Confirmed objects/fields: [list them].

Known preconditions: [list them].

I want the script to:

1. be read-only whenever possible;
2. validate preconditions before continuing;
3. not invent API Names;
4. stop with a clear message if an ID, Record Type, permission or required field is missing;
5. print enough results for me to understand what happened;
6. explain each block and the risks.

If the investigation can be done with SOQL or metadata, without anonymous Apex, prefer the less invasive option.

PROMPT L Clean up the open-items list before investigating

I will give you the request, the current list of open items and the history of answers. For each item, classify it only as:
ANSWERED BY THE REQUIREMENT, ANSWERED BY EVIDENCE, SEARCH THE ORG, FUNCTIONAL QUESTION, TECHNICAL DECISION, TECHNICAL IMPACT or DUPLICATE.

Before keeping any item open, show me why it is not answered.

If two questions are about the same thing, tell me which one should stay and why. Do not create new items without pointing to the concrete gap that justifies them.

PROMPT M Turn "needs validation" into a concrete investigation

You said this needs to be validated. Do not stop there.

Tell me exactly:

- what to validate;
- where to look;
- which query/search/metadata/log/payload/test to use;
- which evidence I should bring back;
- which results would confirm or rule out the hypothesis.

If you need an API Name I did not give you, ask for it instead of inventing it.

PROMPT N Close the analysis before development

Based only on the request, the recorded decisions and the evidence I brought, build a technical brief with:
confirmed AS-IS, confirmed TO-BE, changed components, indirect impacts, data flow, rules/exceptions, preserved legacy behavior, remaining open items, risks and test matrix.

Do not present a new component as a fact if it is still only an implementation decision.

07

CHAPTER 07

How to use AI to learn while you build

Growth happens when you are forced to recall and organize knowledge, not just recognize a ready answer. That is why your use of AI should have moments when it explains and moments when it hands the responsibility back to you.

Learning cycle in five moves

- 01 **Try first:** write your hypothesis before asking.
- 02 **Ask for an explanation:** ask for the concept, an example and how it relates to your case.
- 03 **Do it yourself:** build or write the next step without copying automatically.
- 04 **Ask for critique:** show your solution and ask for problems, alternatives and risks.
- 05 **Teach it back:** explain it in your own words and ask the AI to point out the gaps.

THE COMMIT RULE

Never commit AI-generated code that you cannot explain. If there is a line you do not understand, stop at that line and study it before moving on.

Questions that build reasoning

- › Why is this layer the best place for this rule?
- › What alternative do I have, and what is the trade-off?
- › What could break if this field comes in null?
- › Which governor limit may matter here?
- › How does this code behave with 1 record and with 200 records?
- › How would I prove that this is really the cause of the bug?
- › Which test would fail if I removed this part of the solution?
- › Which evidence separates my hypothesis A from hypothesis B?

One conversation per request, fed with context

For a complex request, the AI conversation can work as a living technical notebook. The idea is to record what you discover step by step (without dumping the whole project into it) and correct the model when a hypothesis falls.

- Start with your manual understanding and your first questions.
- When you find an API Name, component, payload or real behavior, update the context.

- When the org rules out a hypothesis, say clearly that it is discarded.
- From time to time, ask for a summary of only the confirmed facts and the items still open.
- At the end, ask for a summary of what you learned and try to rebuild the flow in your head without looking at the conversation.

THE LEARNING IS IN THE CYCLE

You ask, investigate, come back with evidence, review the hypothesis and explain what you understood. Copying the first ready answer removes exactly this cycle.

08

CHAPTER 08

How to validate code, configuration and architecture

Validation happens in layers. No single tool proves that the solution is correct.

LAYER	VALIDATION QUESTION
Requirement	Does the solution do exactly what was asked, including exceptions?
Environment	Were names, dependencies and behaviors confirmed in the org?
Code / configuration	Does it compile, activate and follow the project patterns?
Security	Does it respect CRUD/FLS, sharing, execution context, secrets and sensitive data?
Performance	Does it consider bulkification, limits, volume, calls and queries?
Tests	Is there evidence for the happy path, negative cases, limits and regression?
Static analysis	Do Code Analyzer and linters point out relevant problems?
Human review	Were critical, architectural or unfamiliar changes reviewed by someone experienced?

Salesforce Code Analyzer can be part of this routine for static analysis of Apex, Visualforce, Flows and Lightning components, finding security, performance and best-practice problems. It complements tests and contextual review, without replacing them.

09

CHAPTER 09

Tests, review and Definition of Done

A junior tends to stop when "it worked in my test". Professional discipline starts when you learn to prove that the solution works, that it does not break nearby scenarios and that the known risks are covered.

Minimum checklist before saying "I am done"

- ☐ I can explain the requirement in my own words.
- ☐ I know which components changed and why.
- ☐ I know which components may be impacted even without a direct change.
- ☐ Every important hypothesis was confirmed or recorded as an open item.
- ☐ I did not use names, behaviors or rules invented by the AI.
- ☐ I tested the main scenario.
- ☐ I tested at least one relevant negative or limit scenario.
- ☐ I validated permissions and security when it applies.
- ☐ I reviewed logs, payloads or records to confirm the real effect.
- ☐ I ran the automated tests that apply.
- ☐ I ran static analysis when it applies.
- ☐ I compared the result with the acceptance criteria.
- ☐ I can explain every piece of code I added.
- ☐ I documented decisions and points that need attention in deploy/QA.
- ☐ If someone else takes over tomorrow, they can understand what was done.
- ☐ Before keeping an item open, I looked for the answer in the request, the history and the evidence already collected.
- ☐ I removed duplicates and split questions that mixed more than one subject.
- ☐ For every "needs validation", I know exactly what to validate, where and with which evidence.
- ☐ Every recorded piece of evidence also says what it does NOT prove.
- ☐ I have an implementation brief that I can explain without looking at the AI.

Test behavior, not only coverage

In Apex, line coverage is a required metric in the delivery process, but on its own it does not show that the rule is correct. A useful test prepares the scenario, runs the behavior and checks the expected result explicitly with relevant asserts.

- Test the happy path and at least the branches the rule really defines.
- Test errors, null/empty values, limits and permissions when they are real risks of the change.
- For a bug fix, first reproduce the defect, then prove that the same scenario is fixed.

- For a change in calculation or persistence, validate the value before, during and after the chain when that helps you find a regression.
- Also test the legacy behavior that should stay the same when the new condition does not apply.

AI review at the end

At the end, ask for an adversarial review. The goal is to try to break your own solution before someone else does.

PROMPT Adversarial review

Act as a critical technical reviewer. Try to find ways this solution can fail.

Requirement: [paste the confirmed summary]

Current architecture: [summarize it]

Changes: [list them or paste the diff]

Tests run: [list them]

Look for:

- requirements not covered;
- possible regressions;
- null values and unexpected data;
- permission and security problems;
- limits and volume;
- concurrency/execution order;
- dependencies not considered;
- missing tests.

Do not invent components of my org. Mark each point as CONFIRMED BY THE MATERIAL or HYPOTHESIS TO VALIDATE.

10

CHAPTER 10

Security, data and limits

Using AI in a company environment requires care with what you send. Convenience must never beat company policies, confidentiality and data protection.

Never send without need and permission

- ✗ passwords, tokens, API keys, session cookies and secrets;
- ✗ Named Credential secrets, client secrets and private certificates;
- ✗ personal data of customers, patients, employees or third parties;
- ✗ full production data when an anonymized example is enough;
- ✗ proprietary code or confidential documentation when company policy forbids it;
- ✗ internal URLs, credentials, IDs or payloads that can identify people without need.

Prefer to anonymize values, reduce payloads to the minimum needed and replace real information with consistent placeholders. If the AI needs to understand how records relate to each other, keep the structure and hide the identity of the people.

AI-GENERATED SCRIPTS

Read-only queries also need review. Scripts with INSERT, UPDATE, DELETE, UPSERT, anonymous Apex, metadata changes or production actions need a much higher level of review. Never run a destructive action just because the AI said it is safe.

11

CHAPTER 11

When to ask a human for help

Being independent does not mean staying stuck for hours. Knowing when to escalate is also a technical skill. The goal is to reach the senior with the investigation done, not just with "I could not do it".

Escalate when

- › the requirement is contradictory or depends on a business decision;
- › there is more than one possible architecture and the trade-off involves the project standards;
- › the change touches a critical integration, security, sensitive data or production;
- › you do not have enough access to confirm the hypothesis;
- › the behavior depends on a managed package or an undocumented external system;
- › the potential impact is large and there is no adequate test coverage;
- › you repeated experiments and could not produce new evidence;
- › the AI and your investigation point to different possibilities, and none of them can be proven.

How to ask for help well

MESSAGE Escalation structure

Problem: [1 to 3 lines]
Expected behavior: [summary]
Observed behavior: [summary]
What I already investigated: [list]
Evidence: [queries, logs, screenshots, components]
Remaining hypotheses: [list]
What I need your help to decide/confirm: [objective question]

12

CHAPTER 12

A complete example of a Salesforce request

Fictional example: a user story asks that sales reps can enter discounts by quantity tier before generating a quote, in a journey built with OmniStudio. The rule says when the feature appears and how the discounts should affect the total calculation, but it does not say which LWC, Integration Procedure, Apex or persistence structure should change.

FIRST CAUTION

"We need a new grid/LWC" can be a technical decision, not a functional requirement. The developer must keep literally what the request requires and investigate the AS-IS before turning their first implementation idea into a fact.

STEP 1

Manual reading

TYPE	ITEM
FACT	The feature only appears under the business conditions stated in the request.
FACT	The discounts entered must reach the calculation and change the quote total.
FACT	When the new option does not apply or is turned off, the legacy behavior must stay the same.
ENVIRONMENT	Where is the equivalent discount calculated and used today?
ENVIRONMENT	Where is the current configuration saved, and how does it enter recalculation and reopening of the quote?
OPEN DECISION	Using an existing component, a new LWC, Data Mapper, IP or Apex depends on the AS-IS and on the project patterns.

STEP 2

Questions before building

- › Which business conditions are already literally defined and do not need to become questions?
- › Which part is still functional and needs to be confirmed with the analyst?
- › Where does the equivalent configuration live today, and at what level of detail?
- › How does the calculation consume this value: per item, per tier, per quote or aggregated?
- › What should happen on edit, recalculation, reopening and with records created before the change?
- › Which legacy paths must stay exactly the same?

STEP 3

Investigation

- 01 Open the current journey and reproduce the AS-IS flow before changing anything.
- 02 Trace the data from the screen to orchestration, rule/calculation, persistence and later consumers.
- 03 Inspect the JSON/payload at key points to see when the value is born or changes.
- 04 Search the project for the real API Names you found, instead of imagining likely names.
- 05 Use SOQL, metadata retrieve, logs and tests to answer environment questions.
- 06 Review the question list and remove everything already answered by the requirement or by evidence.
- 07 Record the impact on recalculation, history, legacy paths, sync/async and consuming components.

STEP 4

Using the AI

Only now does the developer give the AI the observed facts and ask for help comparing options. For example: a visibility rule in the OmniScript itself versus a value calculated in an earlier layer; persistence through a Data Mapper versus existing Apex; impact on re-entering the journey; tests for null values and for the false condition.

STEP 5

Incremental implementation

- 01 Make the smallest change that lets you see the new data on the screen.
- 02 Validate input/output before adding persistence.
- 03 Add persistence and test it in isolation.
- 04 Test the full journey.
- 05 Test the false condition, missing data, re-entry and regression of the old flow.

STEP 6

Closing

The developer must be able to explain: where the condition comes from, where it is evaluated, why the rule lives in that layer, where the data is saved, which components depend on it, which tests were done and what is still a known risk.

The method in one line

Literal requirement → Classified questions → Proven AS-IS → Traced chain → Open items
become checks → Evidence goes back to the AI → Mapped impact → Smallest coherent solution
→ Rule and regression tests → **Final explanation by the developer**

13

CHAPTER 13

Pocket checklist

Four moments, four questions to ask yourself. Print it and stick it next to your monitor.

1

BEFORE THE AI

I read everything. I summarized it in my own words. I separated literal rules, business questions, technical questions and environment questions. I removed questions the request already answers.

2

WITH THE AI

I gave real context. I forbade inferences. I asked for questions. I required explanations. I used AI to investigate and learn, without outsourcing the solution.

3

BEFORE CODING

I proved the AS-IS. I traced the flow and the data end to end. I confirmed components, persistence, consumers and legacy paths. I know what is requirement and what is technical decision.

4

BEFORE DELIVERING

I tested. I reviewed. I ran the analysis that applies. I validated the acceptance criteria. I can explain everything I changed.

14

CHAPTER 14

Junior growth path

The goal is to slowly depend less on AI for simple decisions and use it more for review, advanced investigation and comparing alternatives.

PHASE	EXPECTED BEHAVIOR	USE OF AI	SIGN OF GROWTH
1. Guided	Needs a script to investigate.	AI asks questions, explains concepts and helps write queries.	Stops asking for the direct solution.
2. Investigator	Already arrives with hypotheses and evidence.	AI critiques the plan and points out gaps.	Can isolate the cause before coding.
3. Conscious builder	Designs and builds small parts.	AI reviews code, tests and trade-offs.	Explains every decision.
4. Independent on common requests	Runs the full flow and escalates only relevant decisions.	AI speeds up and challenges, without driving the basic reasoning.	Helps another junior investigate.

A 20-minute weekly ritual

- ☐ Pick a finished request and explain the architecture without looking at the code.
- ☐ Write down one thing the AI suggested that you proved was wrong or incomplete.
- ☐ Write down one new Salesforce concept you learned.
- ☐ Pick a piece of code/configuration and explain why it exists.
- ☐ Choose a topic to study next week, based on the real gaps you found.

FINAL GOAL

The best use of AI is to help the junior build, faster and more consciously, the habits of investigation, validation and reasoning that make them more and more independent.

15

CHAPTER 15

Official sources and recommended reading

The practices in this guide combine development experience with official guidance on critical use of AI, prompting, code quality and Salesforce architecture. Use the sources below to go deeper and to validate.

Salesforce Trailhead: Prompt Fundamentals

Official material on structuring prompts with role, context, task, instructions, format and constraints, and on testing and improving prompts.

<https://trailhead.salesforce.com/content/learn/modules/prompt-fundamentals>

Salesforce Trailhead: Generative AI Basics

Discusses the limitations of generative AI, including hallucinations and the need to check that generated content is factually correct.

<https://trailhead.salesforce.com/content/learn/modules/generative-ai-basics>

Salesforce Developers: Salesforce Code Analyzer

Official documentation of the static analysis tool. The current version uses the Code Analyzer v5 plugin and the code-analyzer commands in Salesforce CLI; v4 was retired in 2025.

<https://developer.salesforce.com/docs/platform/salesforce-code-analyzer/guide/code-analyzer.html>

Salesforce CLI: data query

Official reference for the command that runs SOQL queries through Salesforce CLI, including with the Tooling API.

https://developer.salesforce.com/docs/platform/salesforce-cli-reference/guide/cli_reference_data_query.html

Salesforce CLI: project retrieve start

Official reference for retrieving metadata from an org into the local project.

https://developer.salesforce.com/docs/platform/salesforce-cli-reference/guide/cli_reference_project_retrieve_start.html

Salesforce Well-Architected Framework

Official framework for evaluating the quality of Salesforce solutions from perspectives such as trust, reliability, operational excellence and other pillars.

<https://architect.salesforce.com/docs/architect/well-architected/guide/framework.html>



Read first. Think before. Investigate the environment. Use AI to learn and to challenge your reasoning. Validate everything that matters. Only deliver what you can explain.

MONIQUEDEV