FREE SAMPLE · BUILDING SOFTWARE

The requirements interview

Before any design or code, describe the problem, not the solution, and let the assistant interview you. Then turn the answers into a short requirements document you can test against later.

Copy a prompt into a chat and replace the highlighted [placeholders]. "I don't know" is a fine answer: ask for it to be noted as an open question.

PROMPT 1 · THE INTERVIEW

I want to build some software, and I'd like you to interview me before anything is designed or written. Here's the situation: [describe the problem, who has it and what happens today].
Ask me questions one at a time. Cover: who the users are and what each needs to do; what data is involved and how sensitive it is; how many users and how much data, now and in a year; where it must run (phone, computer, web, offline?); what already exists that it must work with; accounts and who can see or change what; what happens when something goes wrong; budget, deadline and who will maintain it; and anything legal or regulatory I should consider.
After each answer, decide what to ask next. When you have enough, say so and write a requirements draft. Don't propose technology yet.

PROMPT 2 · FROM ANSWERS TO REQUIREMENTS

Turn our interview into a requirements document. Use these sections:
1. Goal (two sentences) and users (each role, one line).
2. Must have, should have, could have, won't have this time. Each item one line, written as "[role] can [action] so that [reason]".
3. For each must-have item, acceptance criteria: concrete checks a non-programmer could perform, including at least one failure case.
4. Data: each kind of data stored, why it's needed, how sensitive it is, who can see it.
5. Constraints: platform, budget, deadline, maintenance, legal.
6. Open questions.
Keep it under two pages. Don't invent requirements I didn't state; list anything you think is missing under open questions instead.

MAKE SURE THE INTERVIEW COVERS THESE

Who can see and change what? Even a small app has roles: in the tool library, members, volunteers and an administrator.

What personal data is involved, and do you really need it? Every field you collect is one you must protect.

What happens on a bad day? A wrong entry, a double booking, a lost phone, the server down on a Saturday morning.

Who maintains it after launch? If the answer is "nobody", choose the simplest possible build.

What is not included? Writing down non-goals keeps the project from growing.

CHECK BEFORE YOU DESIGN

Challenge the requirements

Use a fresh chat that did not write the requirements. Decide on each point yourself.

PROMPT 3 · CHALLENGE THE REQUIREMENTS

Here are the requirements for a small software project. Act as an experienced product manager who has seen many small projects fail. Find: contradictions; requirements that are vague or can't be tested; missing failure cases; data we collect but may not need; features that could be dropped or simplified; and questions a real user would ask that this doesn't answer. Rank your points by how much trouble they'd cause if ignored.

[requirements]

What a few lines of the result can look like, from the book's running example, a lending tracker for a volunteer-run tool library:

EXAMPLE · REQUIREMENTS EXCERPT (TOOL LIBRARY)

MUST
- A volunteer can record a loan (member, tool, due date) so that we know where every tool is.
  Accept: recording a loan marks the tool "out"; the tool no longer shows as available. Fails: a tool already out cannot be lent again; the volunteer sees a clear message saying who has it.
- A member can see which tools are available without signing in.
  Accept: the list shows name, photo, status. It shows no member names.

WON'T (this time)
- Online payments, late fees, reservations in advance.

DATA
- Member name, email, phone: needed to chase overdue tools. Sensitive (personal). Visible to volunteers and admin only.

REQUIREMENTS ARE READY WHEN

- [ ] Every user role is listed with what it needs to do.
- [ ] Every must-have has acceptance criteria you can check by trying it, including a failure case.
- [ ] Every piece of personal data has a reason, a sensitivity and a list of who sees it.
- [ ] Won't-have items are written down.
- [ ] Open questions are listed, and none of them blocks the design.
- [ ] A fresh chat has challenged the requirements, and you have decided on each point.

PROJECT FILES

BRIEF and HANDOFF

Keep these as plain text files in your project folder. Paste the rules block at the start of every chat; write a new HANDOFF at the end of every chat.

TEMPLATE · BRIEF

PROJECT: [name]. One sentence: what it is and for whom.
PROBLEM: [two or three sentences: what happens today, why it matters]
USERS: [role]: [what they need]. [role]: [what they need].
OUT OF SCOPE: [list]
PROJECT RULES (paste at the start of every chat):
- Stack and versions: [ ]. No new technology or libraries without asking.
- Work only on the step given. Don't change other files or behavior.
- Complete files or exactly marked changes.
- No real secrets or personal data in code, logs or examples.
- Validate input; check permissions on every data action.
- Say when unsure whether something exists in our versions.
- After each step: how to run, how to test, what you assumed.

PROMPT 4 · THE HANDOFF NOTE

We're ending this chat. Write a handoff note for a new assistant that has no access to this conversation. Include: the last completed step and what now works; the next step; decisions made in this chat and why; known problems, loose ends and approaches we tried that didn't work; a file map (each main file, one line); and warnings about mistakes the next assistant is likely to make. Under one page. Be specific: names of files, functions and settings, not general descriptions. Don't include code; the files have it.

TEMPLATE · HANDOFF

WHERE WE ARE: [last completed step; what works]
NEXT: [next step from PLAN]
DECISIONS SINCE DESIGN: [decision and reason]
KNOWN ISSUES / TRIED AND FAILED: [ ]
FILE MAP: [file]: [one line]
WARNINGS: [what the next chat is likely to get wrong]

What the full handbook adds. “Prompt the Prompt: Building Software” is a 71-page PDF (70 in A4) in US Letter and A4: 5 parts, 19 chapters and 4 appendices, with 29 copy-ready prompts, checklists, and 7 project-file templates (also as a plain-text file). It covers choosing how to build, design and a first threat model, a build plan in small tested steps, debugging, existing code, testing, fresh-chat code review, audits, launch and maintenance. At mothleaf.com/product/prompt-the-prompt-building-software.

Set for US Letter (8.5 × 11 in); print at 100% or read on screen. Free for personal use: copy and adapt the prompts and templates for your own projects. Please share the link rather than the file. Works with any chat-based AI assistant. Educational only, not professional or security advice: for software that handles payments, personal data or money, have it reviewed by a qualified professional.