# Prompt Engineering for a Complete SDLC Workflow

Crafting a sequence of well-defined prompts can guide an AI through each phase of the Software Development Life Cycle (SDLC), from initial idea to a finished project. Below, we break down the SDLC into stages and explain how to design hyper-focused prompts for each area, using advanced prompt engineering techniques. Each prompt is tailored to gather the right information or produce the desired output while preventing common issues like AI hallucinations, scope creep, or deviations from best practices.

## Requirements Gathering

In the **requirements phase**, the goal is to elicit a clear understanding of the project’s objectives, features, and constraints. To achieve this, design prompts that adopt the role of a skilled business analyst or product manager and ask comprehensive questions about the project. The prompt should encourage the AI to systematically query the user (or stakeholder) for all relevant information:

* **Role and Context:** Begin by instructing the AI to act as a requirements analyst. For example: *“You are a business analyst helping to define a software project’s requirements.”* This sets the stage and focuses the AI on relevant questioning techniques[[1]](https://www.byteplus.com/en/topic/407919#:~:text=To%20get%20the%20most%20out,from%20a%20particular%20expert%20perspective)[[2]](https://www.byteplus.com/en/topic/407919#:~:text=purpose%20of%20the%20code%2C%20and,from%20a%20particular%20expert%20perspective).
* **Ask Clarifying Questions:** Have the AI ask about the project’s purpose, target users, and key features. A good prompt might include: *“Ask me any questions needed to clarify the project’s goals, target audience, required features, constraints (like budget or deadline), and success criteria.”* This ensures no crucial aspect is missed. For instance, ChatGPT in planning can ask, *“What are the potential risks and challenges associated with this project?”* or *“What are the best tools and technologies to use for this project?”*[[3]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=,to%20use%20for%20this%20project%3F%E2%80%9D). By soliciting such details, the AI gathers functional requirements, non-functional requirements, and any domain-specific considerations.
* **Iterative Refinement:** Encourage a back-and-forth. The AI should present a summary of understood requirements and ask if anything is missing. This iterative prompting aligns with the principle *“the output’s quality depends on the input’s quality”*[[4]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=Simply%20put%2C%20the%20output%E2%80%99s%20quality,depends%20on%20the%20input%E2%80%99s%20quality), refining the requirements with each exchange.

*Example Prompt (Requirements):*

“**Act as a software requirements analyst.** I have an idea for a software project. Please **ask me a series of questions** to gather all necessary requirements and constraints. Start by asking about the project’s overall goal and target users, then ask about specific features, performance needs, security, and any other important requirements. **Ensure each question is clear and addresses a different aspect** (functional needs, non-functional criteria, timeline, etc.). Once I answer, use my answers to ask follow-up questions until you have a comprehensive understanding. Finally, **summarize the requirements** you’ve gathered in a concise list.”

This approach yields a thorough requirements specification through multi-turn dialogue. It leverages natural language and detailed context to improve answer quality[[5]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=1,For%20example)[[6]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=for%20loop%20in%20Python%20to,a%20colleague%20you%E2%80%99re%20having%20a). By the end, we should have a clear description of what the project must accomplish.

## Technology Stack Research

With requirements in hand, the next step is to determine the appropriate technology stack and high-level design approach. Here, we **prompt the AI to research and recommend** languages, frameworks, or tools that fit the project’s needs. The prompt should remain **platform-agnostic** at first (since we haven’t assumed any specific tech), and gather reasoning for each option to guide a manual decision. Key techniques include asking for pros/cons and referencing known solutions:

* **State the Project Context:** Begin the prompt by summarizing the project (you can feed in the requirements summary from the previous step as context). This ensures the AI’s suggestions are relevant[[6]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=for%20loop%20in%20Python%20to,a%20colleague%20you%E2%80%99re%20having%20a). For example, *“We are developing a mobile healthcare appointment scheduling app with 5 developers over 6 months*[*[7]*](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=%E2%80%9CGenerate%20a%20project%20charter%20document,%E2%80%9D)*.”*
* **Ask for Recommendations with Justifications:** Request the AI to suggest a tech stack or architecture. For instance: *“Given the requirements above, what are the best tools, programming languages, and frameworks to use for this project, and why?”* This mirrors a planning-stage question like *“What are the best tools and technologies to use for this project?”*[[3]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=,to%20use%20for%20this%20project%3F%E2%80%9D). The AI might weigh options (e.g. **React Native** vs **Flutter** for a mobile app, or different backend frameworks) and mention how each meets the requirements (scalability, security, team expertise, etc.).
* **Ensure Comparative Analysis:** A well-engineered prompt can ask for pros and cons of each option to avoid one-sided or random answers. For example: *“Provide 2–3 possible tech stack choices and discuss their advantages and drawbacks in the context of our project.”* This pushes the AI to draw on factual knowledge of technologies rather than guessing, reducing the chance of hallucination. If available, ask for references or known industry examples for credibility (e.g., *“According to known best practices, which stack suits a healthcare mobile app and why?”*).

The output should be a researched recommendation, not a final decision. As the user, you would review these suggestions (verifying any factual claims) and choose a stack. It’s important to **verify the AI’s claims**, since LLMs can occasionally sound confident with incorrect info. Always *“check your work”*, as one prompt engineering guide reminds us – *ChatGPT can hallucinate, so verify outputs yourself*[[8]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=and%20refine%20as%20necessary,your%20judgment%20to%20the%20tool). In practice, that means double-checking any critical technology advice against documentation or trusted sources.

## Creating Memory and Context Artifacts

Throughout a multi-turn project, it’s crucial to retain important details so they can inform later stages. Large language models have a limited conversation memory, so we create **“memory” prompts or files** to store context and decisions that the AI (and user) can refer back to. Two strategies help here: **summarization prompts** to condense information, and **injection of context** into subsequent prompts:

* **Summarize Key Decisions:** After the requirements and tech stack discussions, prompt the AI to **summarize the project context**. For example: *“Summarize the agreed-upon requirements and chosen tech stack in a bullet list, including key constraints and success criteria.”* The result might be a brief specification or project charter. This artifact serves as a single source of truth for what will be built.
* **Use the Summary in Future Prompts:** At the start of each new phase (design, coding, testing, etc.), you can prepend or reference this summary. For instance, *“Refer to the project summary: {insert summary}. Now, based on this, [ask next task].”* By doing so, we keep the AI grounded in the established context and **prevent it from forgetting** earlier details. OpenAI’s guidelines suggest establishing context early in the conversation to frame answers[[9]](https://community.openai.com/t/chatgpt-memory-and-chat-history-usage-practicalities/1229848#:~:text=ChatGPT%20Memory%20and%20Chat%20History,%E2%80%9CFor%20this%20whole), and this technique achieves that by carrying forward a memory of prior steps.
* **Memory File Approach:** In a longer workflow, you might maintain an external “memory file” (in a tool or just a document) where all important outputs (requirements list, design decisions, coding guidelines, etc.) are saved. When needed, you copy the relevant parts into the prompt. This manual practice mimics how some advanced AI systems use vector databases for memory, but here it’s done by the user. It ensures that at any point, the AI can be reminded: *“As previously decided, the project will use X and must meet Y criteria.”*

By continually integrating these memory artifacts, we **protect against hallucinations** and inconsistencies. The AI is less likely to introduce contradictory details if we keep reminding it of the facts it should stick to. Additionally, labeling and organizing these memory snippets (e.g., *“Project Goals”, “Tech Stack Chosen”, “Coding Standards”*) helps maintain clarity. This approach is reflected in community strategies like the C.L.E.A.R. method (Collect, Label, Erase, Archive, Refresh), which emphasize organizing and refreshing the AI’s memory context[[10]](https://www.reddit.com/r/ChatGPTPromptGenius/comments/1jqzpi9/finally_i_found_a_way_to_keep_chatgpt_remember/#:~:text=%E2%86%92%20C,ChatGPT%E2%80%99s%20memory)[[11]](https://www.reddit.com/r/ChatGPTPromptGenius/comments/1jqzpi9/finally_i_found_a_way_to_keep_chatgpt_remember/#:~:text=The%20prompt%20%E2%86%92). While we may not explicitly run such a loop here, the essence is to keep the conversation’s important points always within the AI’s attention.

## Task Breakdown and Dependency Tracking

With a solid vision and stack in place, the development work can be organized. Use the AI to **break the project into manageable tasks and sub-tasks**, complete with dependencies and priorities. A focused prompt can transform a broad project into a structured work breakdown structure (WBS):

* **Decompose the Project:** Prompt the AI to list major components or milestones first. For example: *“Break down the project into its main components or phases (e.g., front-end UI, backend API, database design, etc.).”* Then instruct it to break each component into specific tasks. A great template is: *“Deconstruct the project into smaller tasks. Identify major phases, then break each phase into actionable tasks with details.”* An example from a prompt library demonstrates this approach, guiding the AI step-by-step to identify phases, tasks, and even time estimates[[12]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=I%20have%20a%20project%20where,logical%20sequence%20and%20assign%20deadlines).
* **Include Dependencies and Sequencing:** The prompt should explicitly ask for any task prerequisites or order. For instance: *“For each task, note if it depends on completion of another task, and arrange tasks in logical sequence.”* This way, the AI will flag things like “Database schema must be designed before API implementation” or “UI design should be finalized before front-end coding begins” – classic dependency awareness in project planning[[13]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=smaller%2C%20more%20manageable%20tasks,logical%20sequence%20and%20assign%20deadlines). The result might be a numbered list of tasks with sub-bullets for sub-tasks, each tagged with dependencies (e.g., Task 2 cannot start until Task 1 is done).
* **Output as a Structured List or Table:** To ensure clarity, you can request the output in a structured format. For example: *“Present the tasks as a markdown list, where each task includes an estimate (if possible) and dependencies.”* This turns the AI’s answer into a pseudo-project plan that is easy to follow. The **Daily Prompt** example shows an ideal output structure: a clear task list with durations and dependencies, organized for effective planning[[14]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=sequence%20and%20assign%20deadlines)[[15]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=,produces%20focused%20and%20practical%20output).

By having ChatGPT generate this WBS, we create a roadmap that will help keep development focused. It’s also a checkpoint against **feature creep** – tasks not in this list are by definition outside the initial scope unless intentionally added later with good reason. If the AI introduces tasks that seem off-scope, that’s an opportunity to catch and trim them now (e.g., “do we really need a machine learning component for this simple app?”). In sum, this prompt uses the AI as a project manager, ensuring we have **phases, tasks, and dependencies clearly laid out** before we start coding.

## Preventing AI Hallucinations and Ensuring Accuracy

As we progress into design and development prompts, controlling for **AI hallucinations** (confident but incorrect statements) is paramount. Several prompt engineering techniques help minimize hallucinations and keep the AI’s output factual and relevant:

* **Provide High-Quality Context:** Always feed the AI the relevant details from previous steps (requirements summary, tech decisions, etc.) when asking for new output. This reduces the chance it will fill gaps with invented details. As noted earlier, *thorough context leads to more accurate outputs*[[6]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=for%20loop%20in%20Python%20to,a%20colleague%20you%E2%80%99re%20having%20a).
* **Ask for Reasoning or Sources:** For critical answers (especially in research or design justification), prompt the model to explain *why* it suggests something. For example: *“Propose a database design and* *explain how it meets* *the scalability requirement.”* By forcing an explanation, you can catch errors in its logic. In fact, asking for sources or references is a known method to check the AI’s honesty – *“If ChatGPT can’t provide real sources, don’t trust the answer.”*[[16]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=Want%20more%20accurate%20answers%3F%20Follow,these%2010%20simple%20tips). In our context, while designing software, we might not get academic citations, but we can ask for things like, “Is this approach based on known design patterns or frameworks?” to gauge validity.
* **Keep Prompts Specific and Clear:** Broad or vague questions encourage the model to roam and possibly fabricate. Instead of *“Design the system,”* ask *“Design the system’s* *high-level architecture* *in terms of components (UI, API, database) and describe the interactions. Do not assume any extra features beyond the requirements.”* This clarity limits the scope of the answer[[17]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=3,or%20wrong%29%20answers). Also, breaking complex queries into smaller parts (design database schema *separately* from UI workflow, for instance) helps reduce errors[[17]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=3,or%20wrong%29%20answers).
* **Validation Checks:** We can incorporate a “verification” step prompt. After the AI produces an output (say a piece of code or a design), use a follow-up prompt like: *“Double-check the above for any assumptions or details not grounded in the provided context. If any exist, point them out.”* Essentially, ask the AI to critique its own answer. Another strategy is re-asking the same question in a rephrased way later – if the answers diverge greatly, that’s a red flag (as the **God of Prompt** blog suggests: inconsistent answers mean uncertainty[[18]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=Not%20sure%20if%20ChatGPT%20is,up%3F%20Here%E2%80%99s%20how%20to%20tell)). While we can’t fully eliminate hallucinations, these measures **greatly reduce their occurrence** or catch them before they cause issues.

Finally, **human oversight remains crucial**: Always double-check critical outputs (design decisions, code logic, etc.) using your own knowledge or external references. Prompt engineering can prompt the AI to be careful, but it’s not a substitute for expert review[[8]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=and%20refine%20as%20necessary,your%20judgment%20to%20the%20tool). Treat the AI as a junior assistant or brainstorming partner, not an infallible authority. This mindset ensures any hallucinated detail gets corrected in subsequent turns.

## Avoiding Overengineering and Feature Creep

Feature creep – the tendency to keep adding “one more feature” – can derail a project. Our prompts should actively guard against this by keeping the AI focused on the **Minimal Viable Product (MVP)** and the core requirements defined earlier. Strategies to enforce this include:

* **Reiterate the Core Vision:** Many prompts (especially in design and development) should begin by restating the project’s primary goal and scope. For instance, *“Recall: The product’s core purpose is to allow patients to schedule appointments and message doctors. All features must support this goal.”* By reminding the AI of the *“crystal-clear product vision”* at each step, we align every suggestion with the core goals[[19]](https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused#:~:text=Establishing%20a%20Clear%20Product%20Vision).
* **Explicit Scope Constraints:** Phrase prompts to explicitly disallow extra functionality. For example: *“Design the module for user login* *using only the requirements given. Do not introduce new features beyond what’s specified (no social login unless requested, no extra profile fields, etc.).”* This acts as a safeguard so the AI doesn’t get creative beyond the brief. If the user themselves attempts to enlarge scope mid-process, the AI (when following instructions) can gently flag that as scope creep: *“This feature was not in the original requirements. Are we expanding scope?”* – allowing a conscious decision rather than accidental bloat.
* **Prioritize and Say No:** Another prompt engineering trick is to have the AI act as a *gatekeeper*. For instance, *“List any feature ideas that have come up. For each, mark whether it’s in-scope or out-of-scope based on the project requirements. If out-of-scope, suggest deferring it to a ‘Phase 2’.”* This aligns with project management best practices of using a roadmap to schedule future enhancements rather than expanding the current project[[20]](https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused#:~:text=%E2%80%93%20Steve%20Johnson)[[21]](https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused#:~:text=,product%E2%80%99s%20direction%20and%20expected%20outcomes). By structuring the conversation this way, any “nice-to-have” that sneaks in can be evaluated and potentially set aside for later, keeping the current development lean and focused.
* **Review Against Requirements:** After a design or implementation prompt, you can ask the AI to compare the result with the requirements list: *“Ensure that the design above addresses all functional requirements and none beyond. Confirm each requirement is met and nothing extra was added.”* This final check uses the AI to cross-verify that we haven’t inadvertently gold-plated the solution.

By systematically reinforcing scope boundaries, we prevent the AI from overengineering the solution. The result is a product that stays true to its intended purpose, avoiding the trap of a bloated tool with lots of unused or unnecessary features. As product experts note, *feature creep leads to a bloated product that loses its core purpose*[[22]](https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused#:~:text=Feature%20creep%20happens%20when%20a,was%20originally%20meant%20to%20do), so our prompts act as the countermeasure, keeping the development on a strict diet of **must-haves** only.

## Enforcing Coding Guidelines and Best Practices

Maintaining high code quality is vital, so we incorporate the project’s coding standards and industry best practices into our prompts. This way, all code generated by the AI will be aligned with expected style and quality from the start:

* **Establish Style Guidelines in Memory:** Early in the process (possibly during tech stack selection or just before coding), ask the AI to **recite relevant coding standards**. For example, *“List the key coding style rules for Python (PEP8) and any project-specific conventions we should follow.”* If the company has an internal style guide, you could summarize it and provide it to the AI. We saw earlier that *many companies have internal coding standards, and AI-generated code should match these as closely as possible*[[23]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=%2C%20you%20may%20need%20to,other%20users%20and%20OpenAI)[[24]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=2,standards%20as%20closely%20as%20possible). By explicitly sharing these rules in the prompt, the AI will format and structure its code accordingly.
* **Include Guidelines in Code Prompts:** Whenever you prompt the AI to write code, remind it of these guidelines. For instance: *“Generate the Java class for the User entity, following our coding standards (use CamelCase for class names, include Javadoc comments, etc.) and best practices.”* This ensures the output isn’t just functionally correct but also stylistically consistent. If security or performance best practices are relevant (which they usually are), mention them too: *“...follow best practices for input validation and error handling.”*
* **Role Prompt as a Senior Developer:** A useful technique is to have the AI *“act as a senior {language} developer”*. For example: *“You are a senior Python developer. Write the function to do X, ensuring it’s efficient and follows good coding practices (DRY, clear naming, etc.).”* Role-playing primes the AI to not just churn out code, but to do so thoughtfully as an expert would[[1]](https://www.byteplus.com/en/topic/407919#:~:text=To%20get%20the%20most%20out,from%20a%20particular%20expert%20perspective)[[2]](https://www.byteplus.com/en/topic/407919#:~:text=purpose%20of%20the%20code%2C%20and,from%20a%20particular%20expert%20perspective). It might do things like include comments or choose clearer logic structures by itself under this guidance.
* **Ask for Best Practice Checks:** After code is generated, a follow-up prompt can be: *“Review the above code for adherence to best practices and our style guide. If any deviations, correct them.”* This double-check uses the AI’s knowledge to polish the code. It’s akin to running a linter or code review automatically. In fact, AI can be very effective at enforcing style consistency and catching simple issues, as it can apply rules systematically. *“ChatGPT can analyze thousands of lines for style and consistency in seconds,”* freeing human reviewers to focus on deeper issues[[25]](https://www.byteplus.com/en/topic/407919#:~:text=Integrating%20AI%20into%20the%20code,quality%20and%20maintainability%20across%20projects)[[26]](https://www.byteplus.com/en/topic/407919#:~:text=subtle%20bugs%2C%20performance%20bottlenecks%2C%20and,quality%20development%20workflow).

By building these guidelines into the prompts, we significantly reduce the amount of cleanup needed later. The AI is less likely to produce, say, non-compliant naming or formatting if we’ve reminded it at every turn what the expectations are[[27]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=2,standards%20as%20closely%20as%20possible). This prompt-engineering-driven discipline results in code that not only works but is clean, readable, and maintainable, aligning with professional standards from the outset.

## Development: Prompting for Implementation of Each Task

Now comes the core **development stage**, where the AI actually generates code for each task in our plan. We handle this in an iterative, task-by-task manner. The prompt design for coding tasks should supply necessary context and specify the output needed (usually code snippets or modules). Key considerations:

* **Focus on One Task at a Time:** It’s best to tackle tasks individually or in logical groupings. For each task (from our WBS), start a prompt that clearly states what needs to be developed. *“Implement the user login API endpoint that validates credentials and returns a session token,”* for example. Include relevant details: *“Use the chosen tech stack (Express.js with MongoDB), follow the input/output format defined in the spec, and include error handling for wrong password.”* By being this specific, we ensure the AI doesn’t wander off-topic or merge multiple concerns.
* **Provide Context Inputs:** Whenever possible, provide the AI with additional context such as data models or previous code. For instance, if an earlier task created a database schema, include a summary of the schema in the prompt when generating the API code that interacts with it. This aligns with the prompt engineering principle of *adding context to guide the model*[[28]](https://maximebeauchemin.medium.com/mastering-ai-powered-product-development-introducing-promptimize-for-test-driven-prompt-bffbbca91535#:~:text=,to%20present%20a%20balanced%20view). An example prompt might be:

*“Below is the database model for User and Appointment collections: {schema details}.* *Now, act as a backend developer* *and write the Express.js route handler for POST /login. It should: check user credentials against the database, create a JWT token using our secret key, and return the token and user info.* *Follow best practices* *(don’t store plaintext passwords – use the existing password hash, etc.) and our coding standards.”*

This prompt packs context (schema, endpoint spec), a role (backend developer), and instructions (best practices, standards) into one – resulting in a focused, high-quality output. - **Encourage Step-by-Step Logic (if complex):** For complex algorithmic tasks, you might use a two-step prompt: first ask the AI to outline a solution approach, then ask for code. *“First, outline how you will implement feature X step by step. Once I approve, provide the code.”* This is a form of **chain-of-thought prompting** that can help ensure the AI’s plan is correct before coding. It leverages the idea that breaking down the reasoning or coding into steps can improve accuracy[[17]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=3,or%20wrong%29%20answers). Once the plan is sound, a subsequent prompt can be *“Great. Now implement that in code, in a single cohesive snippet.”* - **Constrain Output Format:** Always specify how you want the answer. E.g., *“Provide the code only, no explanation, enclosed in markdown triple backticks for formatting.”* Or if you *do* want an explanation for educational purposes or clarity, ask for a brief one after the code. Specifying format is a recommended technique to focus the output[[29]](https://maximebeauchemin.medium.com/mastering-ai-powered-product-development-introducing-promptimize-for-test-driven-prompt-bffbbca91535#:~:text=%3E%20%3E%20,or%20sources%20for%20its%20claims). It prevents scenarios where the AI might otherwise give unnecessary commentary or omit code due to confusion about response style.

By developing each task in isolation and providing all needed context, we **protect against mistakes propagating**. If one task’s output doesn’t look right, we can correct it (with a refined prompt or additional info) before moving on. This is the manual, iterative approach as opposed to asking for a full codebase in one go (which would almost certainly go off the rails). It mirrors how a human developer tackles a project piece by piece, using each completed part to inform the next.

Crucially, this stage is where the earlier groundwork pays off: because we have a solid requirements memory, a chosen tech stack, a task list, and coding standards all fed into the model, the code it generates should align well with our needs on the **first attempt** more often than not. We also continuously verify that the code meets the task description (and tests, next section) to adjust if needed. This methodical prompt-by-prompt development is essentially pair-programming with the AI, ensuring we stay on track.

## UX and UI Design Assistance

If the project involves a user interface or any UX considerations, we can dedicate prompts to **designing the UI/UX**. This can be split into two parts: researching design best practices for our context, and then having the AI produce actual design artifacts (like descriptions or code for the UI).

### Researching UX Best Practices

Before designing the UI, gather guidance on what a good UI should entail for our particular application. We prompt the AI to *act as a UX designer or researcher* to get relevant insights:

* **Ask about Design Principles:** For example: *“What are the best practices for creating a user-friendly interface for a healthcare scheduling app?”* or *“List some UX design principles I should keep in mind (e.g. Nielsen’s heuristics) for this type of application.”* In the design stage, questions like *“How can we create a user-friendly interface for this application?”* and *“What are the best practices for responsive design?”* are exactly the kind of prompts recommended[[30]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=,best%20practices%20for%20responsive%20design%3F%E2%80%9D). The AI might respond with guidelines about clarity, accessibility (important for healthcare), mobile responsiveness, etc., which we can then use in the actual design.
* **Inquire about Visual Design Patterns:** If our app has common features (like a login, dashboard, forms), ask the AI for typical UI patterns or examples. *“What are common design patterns for a scheduling calendar UI?”* might yield useful suggestions (like using a familiar calendar widget, highlighting available slots, etc.). This helps ensure our UI isn’t reinventing the wheel and follows user expectations.
* **User Flow and Experience Questions:** We can even prompt the AI to think from the user’s perspective: *“Describe the ideal user journey for scheduling an appointment through our app, step by step.”* This might surface any UX considerations (like confirmation screens, notifications) that should be included. It’s another way to verify we’re not missing a piece of the interface that users would need.

By doing this research with the AI, we effectively compile a mini UX brief. All this information should then inform the next step, where we design the UI.

### UI Design and Prototyping Prompts

Now we translate those principles into a concrete UI design. Depending on the needs, this could be done as a written specification or even generating code (like HTML/CSS or frontend framework code). Key ways to prompt for UI design:

* **High-Level Design Document:** One approach is to ask for a **UI design document or description**. For instance: *“Generate a UI design specification for the main screens of the app (login, appointment list, booking form, profile). Describe the layout and key elements of each screen, and explain how they follow UX best practices.”* This could result in a text description of each screen (wireframe in words). In fact, the AI can create fairly detailed design docs. A sample prompt from earlier stages shows how to request this: *“Generate a detailed design document... The document should include ... user interface design ... with any diagrams or flowcharts necessary.”*[[31]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=%E2%80%9CGenerate%20a%20detailed%20design%20document,design%2C%20and%20database%20design%2C%20along). While the AI can’t draw images in plain ChatGPT, it can describe what a wireframe would contain or suggest a layout.
* **Front-end Code Generation:** If we know our front-end tech (say we chose React or simple HTML/CSS), we can ask the AI to produce starter code for the UI. For example: *“Using React and Material-UI, code the appointment booking page interface with a calendar view and form. Include proper component structure and styling per Material Design guidelines.”* Here we rely on the AI’s ability to output code given design instructions. We must ensure we’ve provided enough detail (like what fields are in the form, any specific design system, etc.). The AI earlier suggested using a *“modern framework”* for web UI and adhering to separation of concerns[[32]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=interface%20should%20be%20easy%20to,%E2%80%9D), so we echo such best practices in our prompt.
* **Iterate and Refine the UI:** Once initial UI code or design is provided, we review it and refine via prompts. Maybe the layout needs adjustments – we can instruct: *“Adjust the previous UI code to make the submit button more prominent and add validation messages under each field.”* This iterative improvement parallels what a front-end developer would do, and the AI can handle these tweaks quickly.

Throughout UI design, we also remember accessibility and responsiveness if applicable. Prompts can include: *“ensure the design is responsive for mobile and desktop”* or *“follow accessibility standards (WCAG) for color contrast and form labels.”* The AI, when prompted with these specifics, will incorporate them or at least keep them in mind (e.g., adding alt text to images, using proper HTML labels).

By treating UI design as a collaborative process with the AI – first **learning** what good design entails, then **applying** it – we maximize the chances of ending up with a user interface that is both attractive and user-friendly. It’s important that our prompts in this area tie back to the **user needs defined in requirements**, thereby ensuring the UI serves the functionality without unnecessary complexity (tying back to avoiding feature creep as well).

## Test Creation and Quality Assurance

No project is complete without thorough testing. We will use prompts to have the AI generate tests (and possibly even execute or simulate them) and to perform final code reviews for quality. This stage ensures the code meets the specifications and is robust.

### Generating Test Cases and Test Code

We start by asking the AI to produce test scenarios and/or test scripts for our features:

* **Unit and Integration Test Cases:** A prompt might be: *“Generate a set of test cases for the appointment scheduling feature. Include both typical cases (successful booking) and edge cases (double-booking, invalid dates, etc.).”* We should be specific on format: maybe we want them as a list of scenarios, or even as actual test code (like in a given testing framework). For example, *“Provide the test cases as Jasmine unit tests in JavaScript for the booking function.”* ChatGPT can indeed create such tests. In earlier guidance, it’s suggested: *“Can you help me write test cases for this feature?”*[[33]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=Software%20engineers%20should%20consider%20questions,like), and even a sample prompt: *“Generate a set of test cases and test data for a web-based e-commerce application... cover different scenarios... ensure the test cases are easy to follow and include test scripts.”*[[34]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=Sample%20Pompt%3A). We can model our prompts similarly, adjusting for our project context.
* **Automated Testing Scripts:** If our project is large, we might have the AI generate a **test plan** or automated test scripts (e.g., Selenium scripts for UI, or API tests). Prompt example: *“Write a pytest test function to verify the login API returns 200 OK for valid credentials and 401 for invalid ones.”* Including a bit of context (like the API endpoint details) helps accuracy. The AI should output test code that we can then run.
* **Test Data and Mocking:** In cases where we need realistic test data or mocks, prompt the AI accordingly: *“Also provide some example test data (sample user accounts, appointments) to use in these tests.”* The AI can generate dummy data which saves time.

After generating tests, we (the human developer) would run them in our development environment. Since our scenario is a manual process, we assume tests are run outside of ChatGPT (e.g., by copy-pasting code into an IDE or using a tool). Any failures or issues discovered can then be fed back to ChatGPT for debugging help.

### Running Tests and Debugging with AI

If a test fails or a bug is found, we can loop back with a prompt including the error message or a description of the issue. For example: *“One test case failed: the system allowed double-booking an appointment, which it shouldn’t. Here is the problematic code snippet. Help identify the bug and suggest a fix.”* This directs the AI’s attention to a specific problem. The AI, acting as a debugger, can analyze the code and explain what’s wrong, then propose a corrected code block. In a sense, the AI becomes a rubber-duck debugging partner or a junior QA engineer at this stage.

Even if tests pass, we should do a **final code review** for cleanliness, performance, and security. A dedicated prompt for code review could be:

* *“Act as a senior code reviewer. Review the combined code of the project for any potential issues: bugs, security vulnerabilities (like SQL injection, XSS), performance bottlenecks, or style inconsistencies. Provide a list of any problems found, and suggest improvements.”*

We have evidence that such targeted prompts yield valuable feedback: for instance, asking an AI as a cybersecurity expert to review code can uncover vulnerabilities with explanations[[35]](https://www.byteplus.com/en/topic/407919#:~:text=%2A%20For%20a%20Security,concerned%20about%20memory%20usage%20and). Similarly, asking for performance and maintainability improvements (e.g., adhering to SOLID principles) can result in insightful suggestions[[36]](https://www.byteplus.com/en/topic/407919#:~:text=,and%20explain%20the%20reasoning%20behind). By splitting our review prompt into focused areas (security, performance, readability), we apply the earlier principle of breaking down tasks for more thorough results[[37]](https://www.byteplus.com/en/topic/407919#:~:text=Another%20crucial%20principle%20is%20to,more%20relevant%20and%20precise%20feedback).

The AI might respond with a bullet list of issues like “Potential SQL injection in function X – sanitize inputs” or “Inefficient loop in module Y – could use hashing for faster lookup,” along with fixes. We then incorporate those fixes manually or via additional prompts.

Finally, after all tests are green and code is reviewed, we can have confidence the code is **clean and solid**. At this point, the project is essentially complete from a development and QA standpoint. (If deployment is in scope, we could similarly ask for deployment scripts or instructions, but since our focus was up to finished code, we conclude here.)

## Multi-Turn Workflow and Retrospective of Choices

Throughout this SDLC workflow, we have employed a **multi-turn, manual prompting approach**. Each stage’s prompts build on the outputs of previous stages in a conversational manner, which is crucial for managing complexity. Let’s review how our design choices align with the initial criteria:

* **Front-end and Back-end (Both):** We addressed both aspects of the project. Requirements gathering and design prompts considered the system as a whole (e.g. UI and server components). Our task breakdown separated front-end UI tasks and backend API tasks, and we created prompts for each (UI design, UX research for front-end; API implementation, database design for back-end). This ensures neither side was neglected.
* **Manual Guidance:** The process was driven manually, meaning at each phase a human (prompt engineer) decided the next prompt based on the previous answer. We did not let the AI run autonomously; instead, we carefully crafted each prompt and checked the results, in true **human-in-the-loop** fashion. This allowed us to inject corrections or additional context as needed, yielding better outcomes and preventing errors from compounding.
* **Inclusion of Coding and Execution:** We answered “Yes” to whether the AI should actually produce code and other deliverables. Indeed, our prompts went beyond planning – they generated real artifacts: requirement documents, design descriptions, code snippets, test cases, etc. This provides a tangible finished project, not just theoretical advice.
* **Technology Agnostic Start:** We intentionally kept the process platform/language agnostic until the AI (via the tech stack research prompt) helped determine an appropriate stack. By not fixing a framework upfront, we allowed the AI to consider multiple options[[3]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=,to%20use%20for%20this%20project%3F%E2%80%9D) and choose what best fits the requirements. This makes our approach adaptable to virtually any project domain.
* **Multi-Turn and Iterative:** Yes, our workflow is multi-turn. It’s essentially a conversation that walks through the SDLC. Each prompt’s output informed the subsequent prompts, demonstrating the power of iterative refinement. We treated ChatGPT as a collaborator that one can brainstorm with, refine answers, and progressively home in on the final solution[[38]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=4,your%20judgment%20to%20the%20tool). The benefit of this multi-turn setup is evident in how we handled complex tasks (breaking them down) and verified outputs (through follow-up checks), all impossible in a single-turn prompt.

In conclusion, using an AI assistant with carefully engineered prompts for each SDLC phase can significantly streamline software development. From idea to implementation to testing, we’ve shown prompts that elicit requirements, suggest optimal designs, write code following standards, and verify the results. By combining **prompt engineering techniques** (clear context, role specification, iterative refinement, constraint enforcement, self-checking) with software engineering best practices, we ensure the AI remains a helpful partner rather than a source of confusion. The outcome is a coherent, end-to-end development process that yields a finished project aligned with the original vision – all achieved through the artful design of prompts.

**Sources:**

* KMS Technology – *“30 Best ChatGPT Prompts for Software Engineers”*[[4]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=Simply%20put%2C%20the%20output%E2%80%99s%20quality,depends%20on%20the%20input%E2%80%99s%20quality)[[23]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=%2C%20you%20may%20need%20to,other%20users%20and%20OpenAI)[[39]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=To%20refine%20your%20understanding%20of,design%2C%20start%20with%20questions%20like) (demonstrating prompting principles and examples for each SDLC stage)
* *The Daily Prompt* – “Break Down Your Project into Manageable Tasks”[[12]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=I%20have%20a%20project%20where,logical%20sequence%20and%20assign%20deadlines)[[15]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=,produces%20focused%20and%20practical%20output) (example prompt for task breakdown with dependencies)
* GodofPrompt.ai – *“How to Stop ChatGPT Hallucinations”*[[16]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=Want%20more%20accurate%20answers%3F%20Follow,these%2010%20simple%20tips)[[17]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=3,or%20wrong%29%20answers) (tips to reduce AI misinformation, like asking for sources and breaking queries into steps)
* Product School – *“Avoiding Feature Creep: Tips to Keep Your Product Focused”*[[19]](https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused#:~:text=Establishing%20a%20Clear%20Product%20Vision) (importance of a clear vision to prevent scope creep)
* BytePlus – *“ChatGPT Code Review Prompt: Optimize Your AI Reviews”*[[1]](https://www.byteplus.com/en/topic/407919#:~:text=To%20get%20the%20most%20out,from%20a%20particular%20expert%20perspective)[[37]](https://www.byteplus.com/en/topic/407919#:~:text=Another%20crucial%20principle%20is%20to,more%20relevant%20and%20precise%20feedback)[[35]](https://www.byteplus.com/en/topic/407919#:~:text=%2A%20For%20a%20Security,concerned%20about%20memory%20usage%20and) (advice on crafting prompts with specific context, role, and focus for thorough code analysis)
* KMS Technology – Prompting considerations on style guides[[23]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=%2C%20you%20may%20need%20to,other%20users%20and%20OpenAI) and verification of outputs[[8]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=and%20refine%20as%20necessary,your%20judgment%20to%20the%20tool), stressing the need to enforce coding standards and to double-check AI results.

[[1]](https://www.byteplus.com/en/topic/407919#:~:text=To%20get%20the%20most%20out,from%20a%20particular%20expert%20perspective) [[2]](https://www.byteplus.com/en/topic/407919#:~:text=purpose%20of%20the%20code%2C%20and,from%20a%20particular%20expert%20perspective) [[25]](https://www.byteplus.com/en/topic/407919#:~:text=Integrating%20AI%20into%20the%20code,quality%20and%20maintainability%20across%20projects) [[26]](https://www.byteplus.com/en/topic/407919#:~:text=subtle%20bugs%2C%20performance%20bottlenecks%2C%20and,quality%20development%20workflow) [[35]](https://www.byteplus.com/en/topic/407919#:~:text=%2A%20For%20a%20Security,concerned%20about%20memory%20usage%20and) [[36]](https://www.byteplus.com/en/topic/407919#:~:text=,and%20explain%20the%20reasoning%20behind) [[37]](https://www.byteplus.com/en/topic/407919#:~:text=Another%20crucial%20principle%20is%20to,more%20relevant%20and%20precise%20feedback) ChatGPT Code Review Prompt: Optimize Your AI Reviews

<https://www.byteplus.com/en/topic/407919>

[[3]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=,to%20use%20for%20this%20project%3F%E2%80%9D) [[4]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=Simply%20put%2C%20the%20output%E2%80%99s%20quality,depends%20on%20the%20input%E2%80%99s%20quality) [[5]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=1,For%20example) [[6]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=for%20loop%20in%20Python%20to,a%20colleague%20you%E2%80%99re%20having%20a) [[7]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=%E2%80%9CGenerate%20a%20project%20charter%20document,%E2%80%9D) [[8]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=and%20refine%20as%20necessary,your%20judgment%20to%20the%20tool) [[23]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=%2C%20you%20may%20need%20to,other%20users%20and%20OpenAI) [[24]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=2,standards%20as%20closely%20as%20possible) [[27]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=2,standards%20as%20closely%20as%20possible) [[30]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=,best%20practices%20for%20responsive%20design%3F%E2%80%9D) [[31]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=%E2%80%9CGenerate%20a%20detailed%20design%20document,design%2C%20and%20database%20design%2C%20along) [[32]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=interface%20should%20be%20easy%20to,%E2%80%9D) [[33]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=Software%20engineers%20should%20consider%20questions,like) [[34]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=Sample%20Pompt%3A) [[38]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=4,your%20judgment%20to%20the%20tool) [[39]](https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html#:~:text=To%20refine%20your%20understanding%20of,design%2C%20start%20with%20questions%20like) 30 ChatGPT Prompts for Software Development Engineers - KMS

<https://kms-technology.com/software-development/30-best-chatgpt-prompts-for-software-engineers.html>

[[9]](https://community.openai.com/t/chatgpt-memory-and-chat-history-usage-practicalities/1229848#:~:text=ChatGPT%20Memory%20and%20Chat%20History,%E2%80%9CFor%20this%20whole) ChatGPT Memory and Chat History Usage Practicalities

<https://community.openai.com/t/chatgpt-memory-and-chat-history-usage-practicalities/1229848>

[[10]](https://www.reddit.com/r/ChatGPTPromptGenius/comments/1jqzpi9/finally_i_found_a_way_to_keep_chatgpt_remember/#:~:text=%E2%86%92%20C,ChatGPT%E2%80%99s%20memory) [[11]](https://www.reddit.com/r/ChatGPTPromptGenius/comments/1jqzpi9/finally_i_found_a_way_to_keep_chatgpt_remember/#:~:text=The%20prompt%20%E2%86%92) Finally, I found a way to keep ChatGPT remember everything about Me daily: : r/ChatGPTPromptGenius

<https://www.reddit.com/r/ChatGPTPromptGenius/comments/1jqzpi9/finally_i_found_a_way_to_keep_chatgpt_remember/>

[[12]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=I%20have%20a%20project%20where,logical%20sequence%20and%20assign%20deadlines) [[13]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=smaller%2C%20more%20manageable%20tasks,logical%20sequence%20and%20assign%20deadlines) [[14]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=sequence%20and%20assign%20deadlines) [[15]](https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt#:~:text=,produces%20focused%20and%20practical%20output) Break Down Your Project into Manageable Tasks with this Prompt

<https://daily.promptperfect.xyz/p/break-down-project-into-manageable-tasks-with-chatgpt>

[[16]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=Want%20more%20accurate%20answers%3F%20Follow,these%2010%20simple%20tips) [[17]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=3,or%20wrong%29%20answers) [[18]](https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD#:~:text=Not%20sure%20if%20ChatGPT%20is,up%3F%20Here%E2%80%99s%20how%20to%20tell) How To Stop ChatGPT Hallucinations: Here’s How - Workflows

<https://www.godofprompt.ai/blog/stop-chatgpt-hallucinations?srsltid=AfmBOooLgHoL8n8BWzBGCS3_gWCWoRyxrZB4Vc3hU7HHNw7OoymQBZxD>

[[19]](https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused#:~:text=Establishing%20a%20Clear%20Product%20Vision) [[20]](https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused#:~:text=%E2%80%93%20Steve%20Johnson) [[21]](https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused#:~:text=,product%E2%80%99s%20direction%20and%20expected%20outcomes) [[22]](https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused#:~:text=Feature%20creep%20happens%20when%20a,was%20originally%20meant%20to%20do) Avoiding Feature Creep: Tips to Keep Your Product Focused

<https://productschool.com/blog/product-strategy/avoiding-feature-creep-tips-to-keep-your-product-focused>

[[28]](https://maximebeauchemin.medium.com/mastering-ai-powered-product-development-introducing-promptimize-for-test-driven-prompt-bffbbca91535#:~:text=,to%20present%20a%20balanced%20view) [[29]](https://maximebeauchemin.medium.com/mastering-ai-powered-product-development-introducing-promptimize-for-test-driven-prompt-bffbbca91535#:~:text=%3E%20%3E%20,or%20sources%20for%20its%20claims) Mastering AI-Powered Product Development: Introducing Promptimize for Test-Driven Prompt Engineering | by Maxime Beauchemin | Medium

<https://maximebeauchemin.medium.com/mastering-ai-powered-product-development-introducing-promptimize-for-test-driven-prompt-bffbbca91535>