

Prompt Engineering

Prompt Library

List of prompting strategies and approaches.

Welcome to the prompt library! Here we've collected a set of reusable prompt patterns and examples for common scenarios when building with AI. Think of these as templates or inspiration that you can tailor to your own project. Each section covers a particular use case – from kicking off a new project to integrating payments – with guidance on when to use it and an example prompt.

Feel free to copy these, modify the details, and use them in Lovable or any AI builder. The tone is official yet casual – just like talking to a colleague – and each prompt provides enough context so the AI knows exactly what to do.

Starting Projects

When to use: At the very beginning of a project. This prompt helps the AI understand the high-level requirements and start building the foundation. Use it to **kick off a new app** by specifying what you're building, the tech stack, and core features. It's your project brief.

How to use: Outline the type of application, key technologies (frontend framework, backend, any services), and the primary features or pages. Then, direct the AI on where to start (often the main page or an important feature). This establishes the project scope and initial focus.

Example Prompt – Starting a New Project:

I need a **task management** application with:

- **Tech Stack:** Next.js frontend, Tailwind CSS for styling, Supabase for auth and database.
- **Core Features:** Project and task creation, assigning tasks to users, due date reminders, and a dashboard overview.

Start by building the **main dashboard page**, containing:

- A header with navigation,
- A list of projects with their status,
- and a button to create a new project.

Provide dummy data for now, and ensure the design is clean and responsive.

Create a new component called [ComponentName] with these features: [list features]. Make it responsive and accessible with proper

Explain how this function works in simple terms, highlighting its inputs, outputs, and any side effects: [paste function]

This prompt follows a proven structure for new projects. It first states the app type and tech stack, then lists core features, and finally tells the AI where to begin (the main dashboard page, with specifics). By doing this, you give Lovable a clear roadmap to initiate the project. *(Pro tip: It's often wise to start with an empty project and build up gradually, so the AI doesn't get overwhelmed.)*

UI/UX Design

When to use: Any time you want to **refine the look and feel** of your app without changing its functionality. This could be polishing the UI, adjusting layouts, or implementing a specific design style.

How to use: Clearly specify the scope of the design changes and emphasize that functionality should remain intact. The AI is quite good at styling, but you should guide it on what "look" you want (e.g. modern, minimalist, match a certain design system). If you have multiple changes, tackle them one at a time

>

Example Prompt – UI Only Changes:

The app UI should be improved, **without changing any functionality**.

- Keep all existing logic and state management as is.
- **Visual Enhancements:** Update the styling of the dashboard page: use a modern card design for each project listing, improve
- Ensure these changes do **not break any functionality or data flow**.

Goal: purely cosmetic improvements for a more polished look, with the app behaving exactly as before.

Enhance the visual appeal of this component: [paste component]. Add animations, improve spacing, create a polished look **while ma**

Create a comprehensive design system for my application with a color palette, typography scale, spacing system, and component **va**

Design a responsive dashboard layout with [describe key metrics/widgets]. It should work well on mobile, tablet, and desktop **wit**

Transform this desktop-only component into a mobile-first design with responsive breakpoints: [paste component]. Prioritize **cont**

Add subtle, performant animations to this component to enhance user experience: [paste component]. Include enter/exit animations **in**

Analyze and optimize the user flow for [describe task/goal]. Map out each step of the journey, identify friction points, and **sug**

Review these components for accessibility issues and suggest improvements: [paste components]. Check for proper keyboard **navigat**

In this prompt, we explicitly say to make solely visual enhancements and not affect how the app works. This is crucial – it tells the AI “don’t touch the logic.” We list specifics (card design, color contrast, spacing) so the AI knows what aspects of the UI to tweak. This kind of prompt is perfect after you’ve built features and want to beautify the interface.

Responsiveness

When to use: When your app’s layout needs to work across different screen sizes (mobile, tablet, desktop). If you notice things look good on desktop but break on mobile, it’s time for a responsiveness prompt. It’s also wise to do this as a final pass on any UI-heavy task.

How to use: Emphasize a **mobile-first approach** and ask the AI to ensure the design is responsive at all standard breakpoints. If using a CSS framework like Tailwind, mention to use its grid/flex and built-in breakpoints. You can also instruct the AI to avoid any fixed widths or anything that would prevent fluid resizing. Providing an example of what breaks on small screens (if you have one) can help, or simply say “make everything adapt to smaller screens gracefully.”

Example Prompt – Mobile Responsiveness:

>


- Ensure every page (especially the dashboard and project detail pages) reflows properly on a small screen: elements should stack
- **Do not change the core design or functionality**, just make sure it flexibly adapts to different screen sizes.


After making changes, please double-check the layout at iPhone 12 dimensions and a typical desktop width.

In this prompt, we explicitly instruct the AI to make all designs responsive at every breakpoint, focusing on mobile first. We even reference Tailwind's standard breakpoints to guide the implementation. We clarify that the design and functionality shouldn't fundamentally change; it should just work well on smaller screens. This sets a clear expectation: the outcome should look the same design-wise, but fluidly resize and re-stack for responsiveness.



 (Using Lovable's image upload? You could attach a screenshot of a broken mobile layout and ask: "Make it look like this on mobile." Visual prompts can reinforce what you describe.)

Refactoring

 **When to use:** Periodically during development, especially if the AI or you have added a lot of code and things are getting messy or slow. Refactoring means cleaning up the code without changing what it does – improving structure, readability, or performance. Lovable might even suggest refactoring if it detects a lot of repeated patterns or long functions.

 **How to use:** Identify the scope: is it a single file, a specific feature, or the whole codebase? For a single file or component, you can prompt something like "Refactor this file for clarity and efficiency, but **do not alter its functionality or output**." Emphasize that everything should behave the same after refactoring. If you want, specify what to focus on (e.g., reduce duplication, improve variable names, simplify logic). For larger-scale refactoring, it's wise to ask the AI to **plan the refactor in steps** (see the next section on Planning) or audit the code structure first.

Example Prompt – Safe File Refactor:

Refactor the **ProjectList** component file, but **keep its behavior and UI exactly the same**.  



Goals:

- Improve the code structure and readability (simplify complex functions, break into smaller ones if needed).
- Remove any unused variables or imports.
- Ensure the file follows best practices and is well-documented.

Do **not** introduce any new features or change how the component works for the user – this is purely a code cleanup for maintainability.

Review this code and suggest improvements for readability, performance, and maintainability: [paste code]. Focus on TypeScript and  

Suggest a folder structure for a [type] application with these features: [list features]. Include guidelines for organizing components and  

I'm getting this error: [paste error]. Here's the relevant code: [paste code]. Can you help me understand what's causing it and  

This prompt clearly states the component to refactor and the constraints (no functional changes allowed). It prioritizes structure and maintainability. The AI will go through the file, maybe reordering functions, renaming things for clarity, commenting tricky parts, etc., but the output of the app should remain identical. This helps prevent the dreaded scenario of a "refactor" accidentally breaking something.

For bigger refactoring efforts (like many files or an entire project), consider having the AI **analyze the codebase first**. You can use a prompt to get a report on what could be improved and where (see the Debugging section's *Full System Review* prompt for an idea). Then, apply changes incrementally. Refactor in small pieces and test as you go, rather than one massive overhaul.

>

Build a basic CMS for managing blog posts or articles with an admin dashboard. Include features for creating, editing, and publishing content.

Create a project management app with task boards, lists, and cards. Include features for task assignment, due dates, labels, and notifications.

Build a social media feed component with posts, comments, likes, and sharing functionality. Include user profiles, follow/unfollow, and notifications.

Create an analytics dashboard with multiple chart types (bar, line, pie), data filtering options, date range selection, and export functionality.

Build a SaaS application starter with user authentication, subscription management, a settings page, and a basic dashboard. Include a clean, modern UI.

Create a chat interface with an AI assistant that helps users with [describe task/purpose]. Include conversation history, typing indicators, and file uploads.

Build a tool that uses AI to generate [describe content type] based on user inputs and parameters. Include options to refine the output and save templates.

Implement a recommendation component for [describe items] based on user behavior and preferences. Include the ability to provide feedback on recommendations.

Enhance the search experience for [describe content] with AI-powered features like natural language understanding, semantic search, and filters.

Create a dashboard that uses AI to analyze [describe data] and present insights in an accessible way. Include visualizations, charts, and data tables.

Implement a system for personalizing the user experience based on behavior and preferences. Include customizable UI elements, content recommendations, and targeted notifications.

React Development

Create a custom React hook called use[Name] that handles [functionality]. It should handle proper state initialization, cleanup, and error handling.

Refactor this component to use React Context instead of prop drilling: [paste component]. Create a proper context provider and consumer.

Optimize this React component to prevent unnecessary re-renders: [paste component]. Use memo, useMemo, and useCallback where appropriate.

Create a form with validation for [describe form fields and validation rules]. Use react-hook-form with zod schema validation for type safety.

>

Create a smooth transition animation for [describe the element] when it [describe the action]. Use CSS transitions or Framework

Locking Files / Limiting Scope

- When to use:** Sometimes you want the AI to focus on specific parts of the project and leave everything else untouched – essentially “lock” certain files or areas so they are not modified. This is useful if you’ve manually written some code or have a stable component you don’t want altered while the AI works on something else. Since Lovable doesn’t have a literal file-lock feature yet, using the prompt to constrain scope is the next best thing.
- How to use:** In your prompt, **explicitly instruct the AI not to change** certain files or components. You might say, “Don’t edit the authentication files,” or “Keep the HomePage component unchanged.” Also, be clear about where the AI *should* focus changes. This directive should be included each time you prompt during that sensitive period, to remind the AI of the boundary.

Example Prompt – Limit Scope of Changes:

Please ****focus only on the Dashboard page**** for this change.

- Do ****not modify**** the `LoginPage.tsx` or `AuthProvider.tsx` files at all (authentication is working well, and we want to keep
- Concentrate your code edits on `Dashboard.tsx` and related dashboard components ****only****.

Task: Add a new section to the Dashboard that shows “Tasks due this week”. Make sure to fetch the relevant tasks from the databa

(Again, no changes to login or auth files – those are off-limits.)

Here we included a very direct constraint: “refrain from altering pages X or Y and focus changes solely on page Z.”. By repeating this in the prompt, we guide the AI’s attention. The task itself (adding a dashboard section) is given, but we wrapped it with instructions about scope. This greatly reduces the chance of Lovable tinkering with your login system while trying to add a dashboard feature.

Another scenario is when updating a very delicate feature. In such cases, you can combine scope limitation with a cautionary tone. For example: **“This update is sensitive; proceed very carefully and avoid touching anything unrelated”**. This was demonstrated in a prompt like: “This update is quite delicate... Steer clear of shortcuts or assumptions — take a moment to seek clarification if unsure. Precision is crucial.”. Including a line like that sets the AI’s “mindset” to be extra cautious.

Planning

- When to use:** Before diving into a complex or multi-step implementation, or when you have a big feature that could be broken into sub-tasks. Planning prompts are also useful if you want the AI to outline an approach before *writing* code, so you can verify the plan (and adjust it) without burning through code-generation credits on a wrong path. Essentially, use this when the *strategy* isn’t straightforward and you’d like the AI’s help to think it through.
- How to use:** Ask the AI to produce a plan or checklist. You can say, “Outline a step-by-step plan for X” or “Before coding, list the steps you will take to implement Y.” This can be done in Chat mode to ensure it doesn’t execute any code changes while planning. After getting the plan, you might even discuss it (maybe have the AI explain why each step is needed) and then proceed to implementation step by step. Planning prompts are meta – they don’t build the app directly, but they set the stage for a smoother build.

Example Prompt – Planning a Feature Implementation:

>

- Ensure the plan keeps the current functionality stable - we can't break anything existing.
- Provide the plan as an ordered list (1, 2, 3, ...), with a brief explanation of each step.

Once you outline the plan, pause for review. **Do not make any code changes yet.**

This prompt tells the AI to act as a planner. It asks for a sequenced plan to implement an “email notifications for overdue tasks” feature. We explicitly say not to code yet (so we'd run this in Chat mode or just trust that the AI will output a plan). The AI might respond with something like:

1. **Add a timestamp field** to tasks for due date (if not already present).
2. **Create a server-side function** (or scheduled job) to check for overdue tasks periodically.
3. **Integrate email sending** using an email service (e.g., Resend or SMTP) when an overdue task is found.
4. **Update the UI** to allow users to toggle notifications on/off for a task (optional setting).
5. **Test the flow** with a task that just passed its due time to ensure an email is sent.

By reviewing such a plan, you can catch any issues (maybe we realize we need a new DB table, or maybe step 4 is out-of-scope for now, etc.) *before* any coding happens. It's a lot easier to tweak the plan than to rewrite bad code. Planning prompts save time in complex features by getting the approach right from the start.

Stripe Setup

When to use: When you want to integrate payments into your app using Stripe. Lovable has integration points for Stripe, but it requires setting up keys, webhooks, and UI for checkout. A prompt can handle the boilerplate of connecting to Stripe's API. Use this when you need to add commerce (selling a product, subscription, etc.) in your project.

How to use: Provide the details Stripe needs: mode (test or live), product or pricing info, and redirect URLs after payment. Also, instruct how the UI should behave (e.g., a checkout form/modal). It's **crucial** to mention that sensitive keys will be provided securely (not hard-coded in the prompt) – you typically store those in environment variables or Lovable's secret storage. So you can say “assume I have set the API keys in the environment.” This way, the AI will know to call the keys, not include them literally. Additionally, specify *not* to alter unrelated code while setting up Stripe (to avoid accidental changes).

Example Prompt – Integrating Stripe Payments:

I want to **add Stripe payments** to the app.



- Use **Stripe in test mode** for now.
- We have a product in Stripe with ID `prod_12345` and a price ID `price_67890` (one-time purchase).
- Implement a checkout button on the **Pricing page** that starts a Stripe checkout for that product.
- After successful payment, redirect the user to `/payment-success`. If the payment is canceled, redirect to `/payment-cancelled`.

Important:

- Assume API keys and webhook secrets are configured securely (do **not** hard-code them).
- Do **not** modify any other pages or features unrelated to payments.

Once done, provide any webhook endpoint setup instructions I need (e.g., URL to add in Stripe dashboard for post-payment events)



This prompt gives all the key details for Stripe: test mode, product IDs, what happens on success/cancel, and where to put the checkout button. It explicitly says not to touch anything else. The AI (and Lovable's Stripe integration helper) will use this to scaffold the Stripe integration. Under the hood, Lovable might create a serverless function (if using Supabase) to handle webhooks, etc., but you don't have to prompt that separately – the instruction here is usually enough for a basic setup.



Note: We included a line about API keys being secure because we never want secret keys in the prompt. The docs remind us: “Use your Stripe Secret Key in the Supabase Edge Function secrets, and avoid including them in the prompt”. So by telling the AI “assume it's configured,” you ensure the code will reference an environment variable or config, not a plaintext key.

>

Supabase & Backend

Enhance the visual appeal of this component: [paste component]. Add animations, improve spacing, create a polished look while maintaining the core functionality.  



Design a database schema for [describe your application] with these entity relationships: [describe relationships]. Include foreign key constraints and indexes for performance.  



Create a service to fetch data from [API name] and implement caching, error retry logic, and request throttling. Set up proper TypeScript interfaces and guards.  

Create Row Level Security policies for a multi-tenant application with these tables: [list tables]. Implement proper user roles and permissions.  

Create a Supabase Edge Function to handle [describe functionality] with proper error handling, input validation, and security checks.  



Implement real-time data synchronization for [describe feature] using Supabase subscriptions. Handle connection management, graceful disconnection, and reconnection logic.  



Implement a robust search feature for [describe content type] with filtering, sorting, and highlighting of matched terms. Include pagination and caching.  

Create a data table/grid for [describe data] with sorting, filtering, pagination, column resizing, and row selection. Include feature flags for new columns.  

Build a system for importing and exporting [describe data] in various formats (CSV, JSON, etc.). Include validation, progress tracking, and error handling.  

Create a set of interactive charts for [describe data/metrics] using Recharts. Include different visualization types (bar, line, pie, etc.) and tooltips.  



Implement a strategy for synchronizing offline data changes with a backend when connectivity is restored. Handle conflict resolution and data consistency.  

Create a multi-step form wizard for collecting [describe data] with validation, progress tracking, the ability to save drafts, and a final summary screen.  

Workflow

Connect this Lovable project to GitHub and set up a good workflow for contributions. Include branch protection rules, PR templates, and CI/CD pipelines.  

Refactor this large component into smaller, more manageable components: [paste component]. Extract reusable parts, implement proper TypeScript interfaces, and add tests.  

Suggest a testing strategy for [component/feature] including what to test and how. Include unit tests for business logic, integration tests for API calls, and end-to-end tests for user flows.  

>

set up a deployment pipeline for this application that includes staging and production environments, automatic database migration

Analyze and optimize this user flow: [describe flow]. Suggest improvements for user experience, reduce friction points, implement

Using Chat Mode vs Default Mode

When to use: Lovable has two modes for prompting: **Default Mode** (which immediately applies changes to your project) and **Chat-Only Mode** (which is more like a conversation without altering code until you say so). Knowing when to use each can streamline your workflow. Use Default for straight coding tasks and use Chat mode for brainstorming, debugging, or when you want to discuss changes before executing them.

- How to use:**
- Default Mode** is great for when you have a well-defined feature to build or change to make. You give the instruction, and Lovable will do it in one go if possible.
 - Chat Mode**, on the other hand, is useful if you want to have a back-and-forth or analyze something (like asking “*Why is this not working?*” or “*What’s the best way to do X?*”) without immediately changing the codebase. In Chat Mode, the AI will respond with analysis or a plan, and you usually have to explicitly say “go ahead and implement” when ready.

Example use case – Default vs Chat:

Suppose you suspect there is outdated code in your project that needs cleaning up. In Default mode, you might directly prompt:

Review the app and tell me where there is outdated code.

Lovable could try to both identify and possibly start refactoring it in one go. But maybe you actually want to be careful and just get advice. In that case, you’d switch to Chat mode and ask something like:

I’m seeing some deprecated library warnings. What parts of the code might be outdated, and how should we update them?

Now the AI will *discuss* this with you, rather than immediately rewriting files.

Similarly, if you have an error and you want the AI to analyze it, Chat mode is safer. You can copy an error message and ask:

What does this error mean, and how can we fix it?

The AI will explain and propose a solution. Once you agree, you might switch to Default mode or explicitly instruct to apply the fix.

In summary, **use Default Mode for straightforward building** (“do X for me”), and **use Chat Mode for troubleshooting or design discussions** (“why or how should we do X?”). One concrete workflow is: *Brainstorm in Chat (get a plan or identify issues), then execute in Default*. This approach is even recommended: for example, *_use Chat Mode to have the AI analyze errors before making changes_*. It’s like using ChatGPT for advice and then applying the changes once you’re confident.

(Make sure Chat-Only Mode is enabled in your settings under Labs if you don’t see it. This mode ensures the AI won’t write code to your project until you say so, which is perfect for safe experimentation.)

Writing Knowledge Bases and PRDs

When to use: At the start of a project and whenever you have more context to give the AI than can fit in a single prompt. In Lovable, the **Knowledge Base** is a special place in your project settings where you can store background information, requirements, and guidelines that persist across prompts. A **Project Requirements Document (PRD)** is a comprehensive summary of your app’s objectives and specs – essentially, the blueprint of what you’re building. Use these tools to prevent misunderstanding and to anchor the AI with the big picture of your project.

>

- **Core Features & Scope:** What features are in scope (and maybe note any out of scope to avoid feature creep).
- **Tech Stack:** What technologies, APIs, and services will be used.
- **Design Guidelines:** Any specific UI/UX instructions – branding, color scheme, layouts, accessibility requirements.
- **Backend Details:** Outline data models, key database tables, integrations like Stripe or Supabase, etc...

This might sound like a lot, but writing it down pays off. **The AI will use this info on every prompt to stay aligned with your project’s needs.** It’s like giving the AI long-term memory about your app. You typically set this up once (and update as needed).

After populating the knowledge base, you can literally tell the AI to use it. For example, your very first prompt to Lovable might be: `_“Before writing any code, please review the Knowledge Base and confirm you understand the project.”_` This lets the AI summarize or acknowledge the context, and you can catch if it missed something.

Example Prompt – Using the Knowledge Base:

User: Before you write any code, please review the Knowledge Base and share your understanding of my project. 📄 ✨

AI (likely response): “Sure. Based on the Knowledge Base: You’re building a task management app for teams. The app flow is Users

This kind of summary from the AI confirms it has ingested your specifications correctly. If it misunderstood anything, you can correct it in Chat mode. Once confirmed, you proceed with building (the AI now “knows” the overall plan).

Writing a PRD and filling the Knowledge Base might feel like extra upfront work, but it **pays dividends throughout development.** It reduces errors and AI hallucinations because the AI always has a reference for what the app should do. It’s especially helpful for larger projects that can’t be built in one or two prompts. Think of it as seeding the AI with the same understanding you have in your head.

Lastly, maintain the Knowledge Base as your project evolves. If you add a major feature or change the scope, update the PRD/Knowledge Base document. This way, future prompts will consider the new information. It’s much easier than re-explaining context every time. In essence, the Knowledge Base + PRD is your AI project handbook – it keeps everyone (you and the AI) on the same page about what you’re building and how.

Was this page helpful?

👍 Yes

👎 No