Prompts:

Q&A

I need to build a user authentication system for my web application. Before providing a solution, please ask me relevant questions about my specific requirements and constraints so you can give me the most appropriate implementation advice.

Great question. What you're asking for is **how to think and communicate like a senior/lead engineer when using AI**, not just how to "ask for code." Below is a **battle-tested prompt framework + a library of copy-paste prompts** you can reuse. These are designed to:

- Enforce **good architecture**
- Reduce rework and refactors
- Make AI act like a **senior reviewer + implementer**
- Help you **understand** what's being built, not blindly paste code

You'll notice these prompts slow things down *at the right places* and speed things up everywhere else.

---

1. The Golden Rule Prompt (Always Start Here)

Use this **before any serious coding task**.

Prompt

Act as a senior software engineer and tech lead.
Before writing any code:
1. Ask clarifying questions if requirements are ambiguous.

2. Propose a clean architecture and folder structure.
3. Explain trade-offs and assumptions.
4. Identify edge cases and failure modes.
Only after alignment, produce production-ready code.

Why this works:

- Forces **thinking before typing**
- Prevents “half-solutions”
- Mimics how a lead engineer actually works

---

2. Architecture-First Prompt (Prevents Messy Codebases)

Use this when starting a feature, service, or app.

Prompt

Design this feature using industry best practices.
Focus on:
- Separation of concerns
- Scalability and maintainability
- Clear boundaries between layers
- Future extensibility

Provide:
1. High-level architecture diagram (described in text)
2. Folder structure
3. Data flow explanation
4. Key abstractions and interfaces
Do NOT write implementation code yet.

This makes AI think like:

- Backend architect
- Systems designer
- Principal engineer

---

3. “No Rewrites” Prompt (Minimize Back-and-Forth)

This is extremely important.

> **Prompt**

I want a solution that does not require refactoring later.
Before coding:
- Validate assumptions
- List risks and unknowns
- Suggest improvements to the requirements
- Flag anything that could cause technical debt

Only proceed when the solution is robust.

This simulates a senior dev saying:

> “Let’s do this right the first time.”

---

4. Implementation With Guardrails Prompt

Use **after architecture is approved**.

> **Prompt**

Implement the solution using clean, readable, production-quality code.
Requirements:
- Follow SOLID principles where applicable

- Include meaningful comments only where needed
- Avoid over-engineering
- Handle errors explicitly
- Include types/interfaces where relevant

Assume this code will be maintained by another engineer.

This prevents:

- Overly clever code
- Fragile shortcuts
- AI-generated spaghetti

---

5. “Explain It Like I’m a Mid-Level Engineer” Prompt

This is key for **learning while building**.

Prompt

After writing the code:
- Walk me through the implementation step by step
- Explain WHY each major decision was made
- Point out where juniors often make mistakes
- Explain how this would evolve as the system scales

This turns AI into:

- Mentor
- Code reviewer
- Senior teammate

---

6. Code Review Prompt (Catch Bugs Before You Ship)

Use this **even if the code "looks fine."**

Prompt

Review this code as a senior engineer.
Check for:
- Logical bugs
- Edge cases
- Performance issues
- Security risks
- Poor abstractions
- Violations of best practices

Suggest concrete improvements with reasoning.

This alone will save you **hours of debugging later**.

---

7. "Production Readiness" Prompt

Use before deploying or handing off.

Prompt

Assess this feature for production readiness.
Evaluate:
- Logging and observability
- Error handling
- Configuration management
- Testing strategy
- Deployment considerations

List what is missing and how to fix it.

This is how leads think before saying “ship it.”

---

8. Testing Strategy Prompt (Most Devs Skip This)

> **Prompt**

Design a testing strategy for this feature.
Include:
- Unit tests (what to mock, what not to)
- Integration tests
- Edge cases
- Failure scenarios

Provide example test cases and structure.

This prevents:

- Untestable code
- Last-minute test pain

---

9. “If This Were My Codebase” Prompt (High Signal)

This is a powerful mindset shift.

> **Prompt**

If this were your own production codebase:
- What would you change?
- What would you simplify?
- What would you future-proof?
- What would you intentionally NOT optimize yet?

Be honest and opinionated.

AI responds much more intelligently when framed this way.

---

10. Refactoring Without Breaking Things Prompt

Use when touching existing code.

Prompt

Refactor this code safely.
Constraints:
- No breaking changes
- Preserve existing behavior
- Improve readability and structure
- Reduce complexity

Explain each refactor step and why it’s safe.

This mirrors real-world senior dev refactoring.

---

11. “Decision Log” Prompt (Think Like a Lead)

Prompt

Create a short technical decision record (TDR) for this solution.
Include:
- Problem
- Options considered
- Decision made
- Trade-offs
- Future considerations

This builds **engineering maturity** fast.

---

12. Debugging Like a Pro Prompt

Instead of “why doesn’t this work?”

Prompt

Debug this issue systematically.
Steps:
1. List possible root causes
2. Rank them by likelihood
3. Show how to validate each hypothesis
4. Propose the fix with minimal risk

Do not guess.

This trains *engineering thinking*, not trial-and-error.

How to Use This Daily (Simple Workflow)

1. **Golden Rule Prompt**
2. **Architecture-First Prompt**
3. **No Rewrites Prompt**
4. **Implementation Prompt**
5. **Code Review Prompt**
6. **Production Readiness Prompt**

That’s it. This is how senior engineers work.

Final Mindset Shift (Very Important)

AI is not:
❌ “Write code for me”

AI **is**:
✅ “Be my senior engineer and force me to do this correctly”

If you want, next we can:

- Build a **custom reusable prompt template** just for *your stack*
- Create a **Lead Engineer System Prompt** you can reuse everywhere
- Apply this to a real feature you’re working on right now

Just tell me how deep you want to go.