Coding & Development / Documentation

Turn technical discussions into structured ADRs — context, decision, consequences, alternatives.
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Architecture Docs, Technical RFCs, Design Reviews
Updated: May 2026
Why This Prompt Exists
Months after a decision, no one remembers why you chose PostgreSQL over DynamoDB.

You get:

- Slack threads with conflicting opinions and no resolution
- meeting notes that disappear into Notion graveyards
- decisions repeated because previous reasoning is lost
- new team members asking “why is this built this way?”
- architectural drift without documented intent

But good decisions leave traces:

- context: what problem were we solving?
- decision: what did we choose?
- consequences: what trade-offs did we accept?
- alternatives: what did we reject and why?

Without ADRs, institutional memory dies.

This prompt converts discussion fragments into professional Architecture Decision Records.

The Prompt

Assume the role of a software architect who documents technical decisions.

Your task is to convert discussion notes into a structured ADR.

Generate:

1. TITLE
   - Short, descriptive (e.g., "ADR-012: Use Redis for Rate Limiting")

2. STATUS
   - Proposed / Accepted / Deprecated / Superseded

3. CONTEXT
   - Problem statement
   - Constraints (time, budget, team, compliance)
   - Stakeholders affected

4. DECISION
   - What we chose
   - Who decided and when

5. CONSEQUENCES
   - Positive (what gets easier)
   - Negative (what gets harder)
   - Neutral (what stays the same)

6. ALTERNATIVES CONSIDERED

```
   - Option A (why rejected)
   - Option B (why rejected)

7. RELATED ADRs
   - Parent decisions or follow-ups

INPUTS:

Discussion Notes:
[PASTE SLACK THREADS, MEETING NOTES, PR COMMENTS, OR DECISION CONTEXT]

Technical Domain:
[DATABASE / CACHE / API DESIGN / DEPLOYMENT / AUTH / OTHER]

Decision Date (if known):
[DATE OR "RECENT"]

RULES:
- Extract decision even if notes are messy or contradictory
- Flag missing information explicitly ("Unknown: performance
benchmarks")
- If no explicit decision exists, summarize consensus or open
questions
- Keep tone neutral and factual
```

How To Use It

- Paste raw meeting notes, Slack threads, or PR comments — the prompt handles noise.
- Run this immediately after a decision is made, before memory fades.
- Store output in version control (docs/adrs/) alongside your code.

- Link ADRs from code comments when the decision affects specific modules.
- Update status when decisions are deprecated or revisited.

Example Input

Discussion Notes:
“Team: we need rate limiting before launch. Options: Redis (we already use it), nginx rate limiting (but we use Cloudflare), or in-memory (won’t work with multiple pods). Consensus leaning Redis because it’s shared across pods and we have the cluster. Performance seems fine in testing — about 0.5ms per check. Downside: another dependency? But we already depend on Redis for sessions. Let’s do it.”

Technical Domain:
RATE LIMITING

Decision Date (if known):
May 15, 2026

Why It Works
Most technical decisions are lost in ephemeral communication.

This framework improves outcomes by forcing:

- context documentation (why, not just what)
- consequences (trade-offs made explicit)
- alternatives (defensive reasoning)
- status tracking (decision lifecycle)
- relational links (connected decisions)

Great architecture documentation doesn’t describe systems — it explains why they exist.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also README First Responder