Module 2: Architectural Planning and System Design, Lesson 1: Introduction to AI-Assisted System Design

Hands-On: From Prompt-and-Pray to Structured Design Workflow

Overview

This demo makes three claims from the lecture visible in real time:

- A vague one-line prompt produces hallucinated schemas and missing non-functionals — the “prompt-and-pray” anti-pattern
- The five-step workflow with explicit context injection produces a grounded, opinionated design draft
- AI is excellent at critiquing its own output when you ask it to — closing the AI-human handshake

You will sketch an initial architecture for a Team Workload Dashboard feature in a project management SaaS using exactly three prompts, all in the same chat window. Students will follow along and re-run each prompt with their own tool.

Before You Start

- Open your AI tool of choice: Claude.ai, ChatGPT, or Cursor Chat
- Have a blank scratchpad ready — Notes, a markdown file, or the back of a page — for the mini-ADR at the end
- Keep the five-step workflow visible on screen or a second monitor throughout
- Do NOT start a new chat between prompts — session continuity is part of the point

> **Remember**
> AI-assisted design is collaborative, not autonomous. AI brings the generative horsepower. You bring the real-world constraints, the domain knowledge, and the final judgment. We are going to prove both halves of that statement in the next twenty minutes.

During Hands-On Keep Visible if Possible

1. `DEFINE` — write constraints down before opening the chat
2. `INJECT` — business domain, personas, scale, stack, existing arch
3. `TEMPLATE` — structure the prompt with R-I-C-E
4. `CHALLENGE` — ask the AI to critique its own output
5. `VALIDATE` — record decisions in an ADR, compare to team reality

Prompt 1 — The Anti-Pattern (Prompt-and-Pray)

> **What to Learn**
> Here is what everyone does the first time they try AI design work. One line. No constraints. No stack. No users. Watch exactly which failure modes from the lecture show up in the response — hallucinated schemas, missing non-functionals, defaults to REST and Postgres even when we never asked for them.

Paste this into the AI chat:

```
Design me a backend for a team workload dashboard feature in a project
management SaaS.
```

> **What you should get**
> A generic REST API over Postgres. Invented tables like users, teams, tasks, assignments — none of which were grounded in anything you said. No latency targets. No auth model. No mention of compliance. Usually a microservices suggestion “because scalability.” This is the prompt-and-pray output: confident, plausible, and almost entirely disconnected from your reality.

You should ask:

“Where did these tables come from?”

Answer: they came from pattern-matching against every project-management tutorial the model has ever seen. Not from your domain. That is the difference between a generated draft and a grounded draft — and it is the whole reason the next prompt looks different.

Prompt 2 — The Structured Workflow (R-I-C-E + Context Injection)

> **Remember**
>
> Same chat window — do NOT start a new one. The AI has already seen what bad looks like. Now we show it what we actually need: role, instructions, context with all five elements from the lecture, and a concrete example of the output format. The prompt is longer. It is not more complicated — it is more specific.

Continue in the same chat session — do NOT start a new one:

```
ROLE: You are a senior staff engineer doing an initial architectural
sketch. Be opinionated. Flag tradeoffs. Do NOT produce final code.

INSTRUCTIONS: Produce an initial backend sketch for the feature below.
Return FOUR sections in this exact order:
  1. Entities        — 3 to 6 core entities with 2-line justification each
  2. API surface     — REST endpoints OR an alternative with reasoning
  3. Non-functionals — latency, throughput, auth, compliance hooks
  4. Open questions  — the 3 things you most need a human to confirm

CONTEXT:
  Business domain  — B2B project management SaaS. Feature: Team Workload
                     Dashboard. Shows per-person load, at-risk work,
                     re-balance suggestions.
  User personas    — Engineering managers (primary) reviewing weekly;
                     ICs (secondary) checking their own load.
  Scale            — ~2,000 tenants, largest tenant ~800 users, ~50k
                     tasks per tenant. Peak QPS ~150 read, ~20 write.
  Tech stack       — Existing monolith: Python/FastAPI, Postgres 15,
                     Redis, deployed on AWS ECS. We are NOT adopting
                     Kubernetes. We are NOT adopting microservices.
  Existing arch    — Tasks, Users, Teams already exist as Postgres
                     tables in the monolith. Auth is JWT via Cognito.
                     Assume you can add tables, not replace them.

EXAMPLE of the shape I want (not the content):
  Entities: WorkloadSnapshot — materialised per-user weekly rollup,
  refreshed nightly; chosen over live aggregation because...
```

> **What you should get**
> A sketch that reuses your existing tables instead of inventing new ones. Postgres and FastAPI, because you said so. No Kubernetes. Suggestions that look like extensions of a monolith, not a greenfield redesign. Three open questions at the end — that list is the AI telling you where it does not have enough information, which is exactly what a good senior engineer would do.

You should ask:

"Compare this output to Prompt 1. What changed and why?"

Answer: every field of the five-element context block shows up somewhere in the response. Scale numbers drove the read/write split. The stack constraint killed the microservices suggestion. The existing-architecture block kept the AI from re-inventing the task model. The output did not get better because the prompt got longer — it got better because the prompt got more specific.

Prompt 3 — Challenge the Output

> **What to Learn**
> Still the same session. This is the fourth step of the workflow — CHALLENGE. The AI has a draft. Now we ask it to find holes in that draft. You will be surprised at how honest it is when you give it permission to criticise its own work.

Third prompt — still the same session:

```
Critique the sketch above on three dimensions:

  1. Scalability     — what breaks at 10x the scale numbers I gave?
  2. Security        — what is the multi-tenant isolation risk?
  3. Ops complexity  — given a 3-engineer team, what is too expensive
                       to operate?

For each dimension, name the single highest-risk decision and propose
one cheaper alternative. Be blunt. Do not hedge.
```

What you should get

A pointed self-critique. The materialised snapshot probably falls over at 10x. The tenant_id filter probably needs a Postgres row-level security policy, not just a WHERE clause in application code. The nightly refresh job is probably more ops burden than the team can carry. Three real risks. Three cheaper alternatives. This is the AI-human handshake in action — it drafts, it critiques, you decide.

Tip: the highest-value critique is almost always the ops complexity one. Models consistently underestimate how much a small team pays to run a system, because training data overrepresents big-company architectures. Read that answer most carefully.

Wrap-Up — The Pattern, The Assignment Handoff

Remember

Three prompts. One session. We went from a generic hallucinated sketch, to a grounded draft, to a self-critique with alternatives. Nothing about that required a better model. It required defining constraints up front, injecting all five context elements, and closing the loop with a challenge step. That is the workflow. Use it every time.

Capture the decisions now — before you close the tab:

Open your scratchpad and write down, in under two minutes, three things:

- One decision the AI made that you are keeping (and why)
- One decision the AI made that you are overruling (and why)
- One open question the AI raised that needs a human before Lecture 2

That is the seed of your first Architecture Decision Record. The next hands-on builds a full PRD from the same case study — bring those three notes with you.

Key takeaways:

- Constraints before chat, always — define them before you open the window
- Five context elements: business domain, user personas, scale, tech stack, existing architecture
- Session continuity eliminates repetition — the AI carries your context forward
- A longer prompt is not a better prompt; a more specific prompt is
- The challenge step is free and under-used — make it part of every design session