SAMIR GENAI ACADEMY

PROFESSIONAL STUDY GUIDE SERIES

Phase 05 · Capstone & Deployment

23

AI Product Architecture

Design AI-native SaaS — APIs, streaming, caching & costs

Module 23 of 25 · **1 hour**

Instructor: **Samir Kumar** · Edition 2026

STUDY GUIDE

01 Overview

> Design AI-native SaaS — APIs, streaming, caching & costs. This guide gives you the core theory, the tools that matter, a hands-on build, working code, and the pitfalls to avoid — everything you need to learn the topic to a professional standard and apply it in real projects.

1 hour	4 + 4	4	23/25
LESSON LENGTH	CONCEPTS + BUILD STEPS	FRAMEWORKS & TOOLS	PHASE 05

02 Learning Objectives

- Architect an AI-native backend with streaming APIs.
- Apply caching strategies to cut cost and latency.
- Design for rate limits, retries, and graceful degradation.
- Build a FastAPI backend with WebSocket streaming.

03 Core Concepts

AI-native backend

AI apps differ from CRUD apps: responses stream, calls are slow and costly, and outputs are non-deterministic. Architect around async I/O, streaming transports (SSE/WebSocket), queues for long jobs, and clear separation between the model layer and business logic.

Caching strategies

Cache aggressively: exact-match response cache, semantic cache (reuse answers for near-identical queries), prompt-prefix caching (provider-side) for long static system prompts, and embedding caches. Caching is often the single biggest cost lever.

Resilience

Design for provider rate limits and outages: retries with backoff, model fallbacks, timeouts, and graceful degradation (return a cached or simpler answer rather than an error). Queue long tasks and stream progress.

Cost engineering

Track cost per request and per feature. Right-size models per call, trim prompts, cap output tokens, and batch where possible. Make cost a first-class metric in design reviews.

04 Visual Diagram

Figure 23 — Stream responses, cache aggressively, and degrade gracefully under rate limits.

05 Formula Spotlight

Cache cost saving

$$\text{saving} = \text{hit\_rate} \times \text{cost\_per\_call}$$

`hit_rate`	fraction of requests served from cache
`cost_per_call`	average price of a full model call
`saving`	expected cost avoided per request

Caching is usually the single biggest cost lever: a 40% hit rate cuts ~40% of model spend on those requests for near-zero cost. This is why exact, semantic, and prompt-prefix caching belong in every AI backend.

06 Key Frameworks & Tools

FastAPI · WebSocket / SSE · Redis (cache/queue) · Async clients

07 Hands-On Build

🛠 FastAPI AI Backend with WebSocket Streaming

1. Build a FastAPI service with an async model client.
2. Stream responses over WebSocket/SSE.
3. Add a Redis response + semantic cache.
4. Add retries, timeouts, and a fallback model.

08 Code Walkthrough

Streaming endpoint with FastAPI WebSocket — PYTHON

```python
from fastapi import FastAPI, WebSocket

app = FastAPI()

@app.websocket("/chat")
async def chat(ws: WebSocket):
    await ws.accept()
    query = await ws.receive_text()
    async with client.messages.stream(
        model="claude-opus-4-8", max_tokens=500,
        messages=[{"role": "user", "content": query}],
    ) as stream:
        async for text in stream.text_stream:
            await ws.send_text(text)
    await ws.close()
```

09 🧪 Lab Assistant

Lab 23 · Streaming FastAPI backend

Prerequisites:pip install fastapi uvicorn redis · An async model client

1. **Expose a WebSocket endpoint that streams model tokens**
   > Expected: Tokens arrive live in the client
2. **Add a Redis response cache**
   > Expected: Repeat queries return instantly
3. **Add retries + a fallback model**
   > Expected: Graceful behaviour under rate limits

Troubleshooting

- Workers exhausted → use async, not blocking calls.
- Errors to users → degrade to cached/simpler answers.

10 Pitfalls & Key Takeaways

⚠ COMMON PITFALLS

- Synchronous blocking calls that exhaust workers under load.
- No caching, so identical queries pay full price every time.
- Returning hard errors instead of degrading gracefully.

✓ KEY TAKEAWAYS

- AI-native backends are async, streaming, and queue-based.
- Caching (exact/semantic/prefix) is the biggest cost lever.
- Design for rate limits with retries, fallbacks, and degradation.

📺 **Companion video:** Watch the module 23 lesson in your Student Dashboard, then use this guide to revise, copy the code, and complete the build. Download both for offline study.

11 Further Resources

FastAPI docs • Provider prompt-caching guides • Redis caching patterns