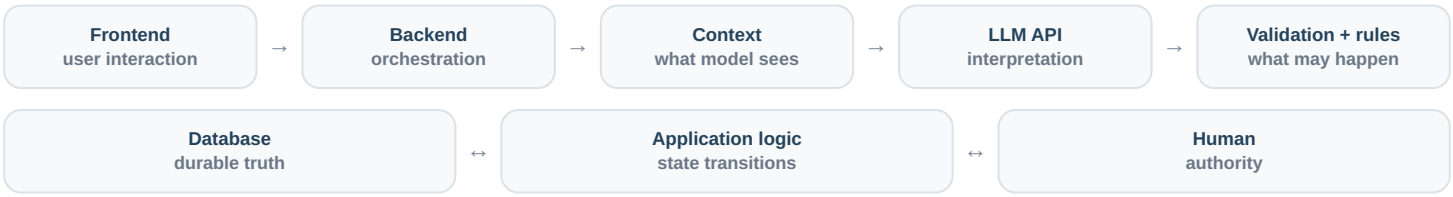
AI SYSTEMS CHEAT SHEET

Anatomy of an LLM Application

The model is one component. The surrounding application controls context, data, permissions, validation, persistence, and consequences.

Core idea: Treat the LLM as a probabilistic interpreter inside a larger software system. Reliability comes from what the application does around the model, not from the model alone.

The system at a glance



What each part owns

FRONTEND
Shows the experience, captures user intent, and renders authoritative application state.

BACKEND
Coordinates protected logic, data access, model calls, validation, tools, and server-side security.

DATABASE
Persists information across sessions: evidence, state, history, reviews, configuration, provenance.

LLM API
Makes semantic judgments: summarize, classify, interpret, draft, choose among ambiguous possibilities.

APPLICATION LOGIC
Enforces deterministic rules, permissions, allowed transitions, retries, and consequences.

HUMAN
Provides judgment and authority where consequential decisions or approvals must remain explicit.

Backend is an umbrella term. Request handling, database access, model calls, context construction, validation, and business rules are separate responsibilities even when they live in one small codebase.

AI SYSTEMS CHEAT SHEET

Anatomy of an LLM Application

Three concepts matter especially for product work: context construction, structured outputs, and validation.

One evidence flow through State

Save source evidence → **Build trusted context** → **LLM interprets** → **Validate output** → **Human approves if needed** → **Software applies transition**

Context construction

Decide what information the model receives for this task. Send the smallest trustworthy governing context that is sufficient - not the entire history by default.

Structured output

Ask the model to return a predefined shape rather than free-form prose when downstream software needs predictable fields.

Validation

Check required fields, allowed values, references, and combinations before the application acts.

Valid output is not necessarily correct output. Schema validation proves the response is well-formed enough to process. It does not prove the model interpreted the evidence correctly.

Design for model mistakes

- ✓ Preserve original source evidence unchanged.
- ✓ Keep AI interpretation separate from source material.
- ✓ Validate before application actions.
- ✓ Use deterministic rules for mechanical facts.
- ✓ Require human authority for consequential transitions.
- ✓ Show provenance beside AI interpretation.
- ✓ Keep history so prior state is inspectable.
- ✓ Fail safely when the model is unavailable or unusable.

Current State vs. repeated re-reasoning

For ordinary current-state questions, maintained State should govern. Do not repeatedly ask the model to reconstruct truth from the raw archive when the product already has an authoritative current representation.

If an unresolved review could make that state stale, surface the qualification rather than silently changing the answer.

The model interprets ambiguity. The surrounding application owns persistence, permissions, validation, state transitions, and consequences.