How to use AI for Web Development

Document version: v1.0
Document last updated: 14 Jan 2026
Document Owner: Tamas Karakai

This guide is a practical map for using generative AI in web engineering. It's organized into three sections:

- **Foundations:** core concepts, safety guardrails, and shared vocabulary.
- **AI-Assisted Development:** using AI to build software faster and safer.
- **Shipping AI Features:** shipping embedded AI features with production-grade reliability.

If you're unsure where to start: complete **Foundations**, then do **AI-Assisted Development**. Add **Shipping AI Features** when you're responsible for shipping AI-backed product capabilities.

Scope note: This document focuses on generative AI for text and voice (LLMs, STT/TTS, tool-using agents). It excludes Computer Vision and non-generative ML unless explicitly noted.

1. Foundations

1.1. Orientation

Topics

- What AI changes in web engineering (SDLC + product outcomes)
- Typical failure modes and how to recognize them early
- Policies: data handling, approved tools, review expectations

Enables you to

- Decide when to use AI vs. avoid it (risk-based)
- Recognize hallucinations, drift, and "confidently wrong" outputs
- Apply internal policy constraints consistently

1.2. Bias, Harms, and Transparency (baseline)

Topics

- Common bias and harm patterns in generated outputs
- Human impact assessment for user-facing AI features
- User transparency: disclosures, citations, and limitations
- Safeguards for sensitive or high-stakes domains

Enables you to

- Spot and mitigate biased or harmful outputs early
- Set expectations and disclosures appropriate to risk level
- Decide when not to use AI for a given user workflow

1.3. LLM Mechanics

Topics

- Tokens / cost / latency / truncation
- Context windows and sampling controls
- Determinism, reproducibility, and why "close enough" is dangerous in prod
- **Open-source vs. proprietary LLMs**
    - Trade-offs: quality, cost, control, privacy, and operational overhead
    - Running models locally:
        - Model size vs. RAM/VRAM requirements
        - CPU vs GPU inference
        - CUDA (NVIDIA) vs non-CUDA (Metal/macOS, ROCm/Linux, CPU-only) considerations
    - Practical ecosystem:
        - Hugging Face (model discovery/weights)
        - GGUF + llama.cpp (local inference with quantization)

Enables you to

- Choose a model/serving approach based on latency, cost, and privacy constraints
- Understand why prompt length and context strategy dominate cost/perf
- Predict and mitigate truncation and nondeterminism issues

1.4. Prompting and Interaction Patterns

Topics

- Instruction hierarchy, constraints, examples, output formats
- Prompt debugging and iteration discipline
- "Trust but verify" habits: tests, repros, diffs
- **Agent loops & agentic workflows (intro)**
    - When to use single-shot prompting vs plan→act→observe loops
    - Stop conditions: step limits, budget limits, timeouts
    - Tool-using prompting patterns (e.g., ReAct-style separation of reasoning/action)

Enables you to

- Write prompts that produce structured, reviewable outputs
- Iteratively debug prompts by controlling variables and tightening constraints
- Design "safe prompts" that include verification steps (tests, validators)

1.5. Context Management

Topics

- How to provide the right repo/code/doc context
- Summarization and drift control for long threads
- Watching context length to avoid overload and confusion
- Context compaction strategies (when supported by tools)
- Avoiding accidental leakage of secrets/PII in prompts
- Data lifecycle basics: logging, retention, redaction, training opt-out
- **LLM-friendly documentation strategies (intro)**

  - Write docs that work as model context: explicit contracts, examples, edge cases, “do/don’t”
  - Handling “docs newer than model training data”:
    - convert web docs to curated Markdown snapshots
    - controlled browsing (only in allowed environments/tools)
    - retrieval tools and doc servers (e.g., Context7)

Enables you to

- Create “context bundles” that maximize accuracy and minimize leakage
- Keep long-running threads coherent using summaries and checkpoints
- Make internal docs usable as reliable context for agents

1.6. Operational Usage Guardrails

Topics

- What can/can’t go into prompts (data classification)
- Secure handling of secrets and customer data
- When to avoid AI (high-risk changes, unclear requirements, missing tests)
- Guardrails for local models too (privacy ≠ correctness)
- Rules for browsing-enabled tools (if permitted): source trust + citations

Enables you to

- Avoid accidental policy violations and data leaks
- Know when AI increases risk more than it helps
- Create team-level “safe defaults” for AI usage

1.7. MCP Protocol Overview

Topics

- What MCP is: a standard protocol for tool and context interfaces
- Core primitives: servers, tools, resources, prompts
- Local vs remote servers; transport and auth patterns
- Capability scoping, auditability, and least privilege
- Versioning and schema stability for tool contracts

Enables you to

- Explain when to use MCP vs direct API integrations
- Scope MCP servers safely and assess risk
- Map MCP usage to developer tooling (AI-Assisted Development) and product architecture (Shipping AI Features)

1.8. Evals Basics

Topics

- Define success criteria: what “good” means for the task
- Build small eval sets (golden cases, edge cases, regressions)
- Baselines and diffs: compare prompts/models/configs over time
- Human review gates and when to require them

Enables you to

- Create a minimal eval harness that catches obvious regressions
- Compare changes objectively before shipping or adopting new prompts
- Decide when human review is required vs optional

1.9. Legal, IP, and Compliance Basics

Topics

- Provider terms and data usage restrictions (what may be trained on, stored, or shared)
- IP/copyright considerations for generated outputs
- Attribution expectations and third-party license constraints
- Data residency and export/compliance requirements (as applicable)

Enables you to

- Avoid accidental policy or licensing violations
- Know when to consult legal/compliance before use
- Apply “safe defaults” for external sharing and publishing

2. AI-Assisted Development (Using AI to Build Software Faster)

2.1. Tooling Ecosystem and Setup

Topics

- IDE copilots vs editor chat vs CLI agents
- Repo indexing/search tools; local vs hosted models
- Standard setups for full-stack roles (frontend/backend/platform)
- **Local open models deep dive**
    - Model selection: size, context length, quantization, quality/perf trade-offs
    - CPU-only vs GPU workflows; CUDA vs non-CUDA guidance
    - llama.cpp + GGUF basics; Hugging Face model discovery

- **MCP for developer tooling**
    - Using MCP servers locally for:
        - docs retrieval,
        - repo search,
        - ticketing/runbooks/internal APIs (where permitted)

    - Security considerations:
        - treat MCP servers as privileged; add auth/ACL if not strictly localhost
        - scope tools by least privilege and audit usage

Enables you to

- Build a personal “AI toolbelt” that’s safe and consistent
- Configure local model fallback for privacy or offline use
- Use retrieval/indexing to supply better context than copy/paste

2.2. Day-to-Day Workflows

Topics

- Scaffolding with architectural constraints
- Refactoring at scale: slicing strategy, safe checkpoints
- Debugging: minimal repro, hypothesis loops, targeted instrumentation
- Reading unfamiliar repos: architecture extraction, dependency maps
- Migration assistance (framework upgrades, API versioning, DB migrations)
- **Vibe coding**
  - When it's useful (rapid prototyping, exploring solution space)
  - When it's risky (security-critical changes, ambiguous requirements)
  - How to "graduate" to disciplined engineering: spec→checkpoints→tests→review

- **Spec-driven development (SDD)**
  - Approaches/tools: Kiro, Spec-Kit, BMAD, OpenSpec
  - Best practices:
    - acceptance criteria as executable intent
    - freeze constraints early (APIs, invariants, performance budgets)
    - plan/task breakdown before code
    - verify with tests and diffs at each checkpoint

Enables you to

- Run safe, incremental AI-assisted refactors with rollback options
- Use SDD to reduce "AI thrash" and improve consistency
- Convert vague ideas into testable specs and implementation plans

2.3. Testing and Quality with AI

Topics

- Generating tests responsibly (assertion-first prompting)
- Property-based tests, fuzzing assistance, contract tests
- Using AI to strengthen reviews (risk surfacing, invariants, edge cases)
- Spec→tests workflows (from acceptance criteria to test scaffolds)
- Mutation-inspired prompting: ask AI to propose changes that tests should catch

Enables you to

- Increase coverage *without* false confidence
- Generate tests that fail meaningfully when behavior is wrong
- Use AI to uncover edge cases you didn't think of

2.4. AI-Assisted Technical Writing

Topics

- RFCs/ADRs and tradeoff capture
- API docs, runbooks, release notes
- Spec → checklist → tests workflows
- Operational constraints and "safe usage" guidance in docs

Enables you to

- Produce clear docs faster without losing accuracy

- Convert design intent into actionable checklists and tests
- Improve operational readiness (runbooks, incident response steps)

2.5. Code Review and Governance Practices

Topics

- Rubric-driven review prompts (security/correctness/perf/maintainability)
- Standards for accepting AI-generated changes
- Prompt packs and workflow templates as shared team assets
- **Agent steering via repo conventions**
  - Adopt AGENTS.md as the repo-standard “agent instruction file”
  - Use tool-specific files (e.g., .claude.md) as shims/redirects if needed
  - What goes in AGENTS.md:
    - repo map and architecture notes
    - build/test commands
    - code style rules
    - “safe areas” vs “danger zones”
    - review checklist + invariants

Enables you to

- Make AI usage repeatable across the team (not heroics)
- Reduce review burden by standardizing constraints and verification
- Ensure AI-generated changes are held to the same quality bar

3. Shipping AI Features (System Design + Platform)

3.1. Product Patterns and UX for AI Features

Topics

- Chat/copilot UX, citations/provenance, revision workflows
- Failure-state UX: “I don’t know”, partial answers, escalations
- Agent workflows: approvals, audit logs, recoverability
- **Voice modality (intro)**
  - voice UX basics: turn-taking, barge-in, confirmations for risky actions
  - latency targets and why voice is unforgiving

Enables you to

- Design internal AI UX that handles uncertainty gracefully
- Make agent actions auditable and reversible
- Decide when voice adds value vs complexity

3.2. Message Design and Application State

Topics

- System/developer/user message separation in production
- Template versioning and change control
- State management: session memory vs durable memory vs user preferences

- Preventing instruction drift and injection via state
- Memory hygiene: what to store, how to summarize, when to forget

Enables you to

- Prevent prompt injection via state contamination
- Version prompts and state transitions like APIs
- Build predictable multi-turn behaviors

3.3. Output Control and Reliability (production-grade)

Topics

- Structured outputs with schemas + validators + repair loops
- Tool/function calling contracts and invariants
- Streaming + partial parsing + UI implications
- Reproducibility strategies and behavioral regression testing

Enables you to

- Make AI outputs parseable and safe to consume
- Reduce flakiness with validation and repair loops
- Test behavior changes like code changes

3.4. API Integration Patterns and Architecture

Topics

- Chat vs agentic APIs; tool-based designs
- Latency/cost tradeoffs, caching layers, backpressure
- Idempotency, retries, dedupe; side-effect control
- Sandboxing and constrained tool environments
- **MCP in product architecture**
    - using MCP servers as standardized tool/context backends
    - stable schemas, scoped capabilities, safe defaults
    - security: authn/authz, tool scoping, audit logging

Enables you to

- Build safe tool-using agents that handle retries and side effects
- Integrate tools without turning the model into a privileged backdoor
- Use MCP where it improves maintainability and interoperability

3.5. Moderation, Rate Limits, User Reporting, and Policy Enforcement

Topics

- Moderation models and policy layers (pre/post filters, allow/deny lists)
- Rate limits and abuse throttling (per-user, per-tenant, adaptive limits)
- User reporting workflows and feedback loops
- Policy enforcement in UI and backend (refusals, safe alternatives, escalation paths)
- Logging for safety review without exposing sensitive content

Enables you to

- Prevent abuse and policy violations at the product boundary
- Build clear user-facing escalation and reporting paths
- Balance safety controls with usable UX

3.6. Security: Prompt Injection, Tool Abuse, Exfiltration

Topics

- Direct/indirect injection; confused deputy problems
- Defenses: scoped tools, least privilege, content sanitization
- Logging/telemetry exfil pathways and mitigations
- Red-teaming and continuous adversarial testing
- MCP-specific security: treat server/tool surface as privileged

Enables you to

- Threat model AI systems like any other input-driven system
- Build layered defenses and continuous adversarial tests
- Prevent tool misuse and data leakage

3.7. Observability and Monitoring for LLM Systems

Topics

- End-to-end tracing: request → retrieval → model → tools → response
- Cost monitoring by feature/tenant; budget enforcement
- Quality monitoring: deflection, escalation, feedback taxonomy
- Drift detection (model/corpus/prompt/config)

Enables you to

- Debug performance, cost, and quality issues quickly
- Detect drift before users do
- Create operational playbooks for AI incidents

3.8. Repeatable Evals and CI/CD for AI Behavior

Topics

- Offline eval harnesses and golden conversations
- Online evals: A/B, canary, shadow traffic
- Rubrics + graders; pitfalls of LLM-as-judge
- CI gating: prompts/templates/corpora as release artifacts

Enables you to

- Ship behavior changes safely
- Use eval gates to prevent regressions
- Compare models/prompts objectively

3.9. Retrieval-Augmented Generation (RAG) Systems

Topics

- Chunking, indexing, hybrid search, reranking
- Freshness/versioning of corpora; staleness controls
- Authorization-aware retrieval (multi-tenant + per-doc ACLs)
- Groundedness, citation quality, and “unknown” handling
- Corpus structure for retrieval quality:
  - chunkable structure, stable headings, canonical IDs
  - versioned pages and clear deprecation markers

Enables you to

- Build grounded systems that cite sources and admit unknowns
- Prevent stale or unauthorized knowledge leakage
- Improve retrieval quality via structure and evaluation

3.10. Fine-Tuning and Customization Strategy

Topics

- Decision framework: prompt/templates vs RAG vs fine-tune
- Data pipelines, leakage risks, ongoing maintenance burden
- Post-change evaluation and rollback planning
- Open-source fine-tuning considerations vs proprietary constraints

Enables you to

- Pick the simplest effective customization method
- Avoid “fine-tune by default”
- Plan for lifecycle: data, evals, monitoring, rollback

3.11. Model Routing and Cost/Latency Engineering

Topics

- Task-based routing, dynamic routing signals
- Caching: semantic, retrieval, response
- Fallbacks, degraded modes, timeouts
- Batch/async patterns for expensive operations
- Local-first vs hosted escalation routing:
  - privacy/cost first locally,
  - escalate hard tasks to stronger hosted models

Enables you to

- Meet performance targets while controlling spend
- Design robust degraded modes
- Use routing to optimize quality where it matters

3.12. Deployment, Versioning, and Change Management

Topics

- Version prompts/templates/tool schemas/eval sets/corpora
- Release playbooks for behavior changes
- Rollback strategies and compatibility management

Enables you to

- Treat AI artifacts as first-class release components
- Roll back safely when behavior changes regress
- Maintain compatibility as tools/schemas evolve

3.13. Voice Interfaces (STT/TTS + Voice-specific challenges)

Topics

- Voice pipeline: STT → LLM → TTS
- STT basics: accuracy, latency, noise handling, multi-speaker issues
- TTS basics: streaming vs non-streaming, SSML/prosody control
- Voice UX: confirmations, barge-in, error recovery, safe actions

Enables you to

- Decide where voice is worth it for internal tools
- Prototype a safe voice assistant workflow
- Design around recognition errors and latency