THE 10 PROMPTS

Every Developer Should Have Saved

Copy-and-customize templates for the tasks you do every week. From the book Vibe Coding: The Architect's Guide to AI-Powered Software Development.

Josh Cooper

Software Architect | 15 Years in Enterprise Systems

FREE RESOURCE — Extracted from Appendix B of the full book

How to Use These Prompts

Each prompt follows the same framework: **CONTEXT** → **CONSTRAINTS** → **OUTPUT**. Replace the bracketed sections with your project's specifics. These aren't theoretical—they're the prompts I use on real projects every week. For a deeper dive into why each element matters, see Chapter 3 of the full book.

1. Project Scaffold

Use when starting a new project from scratch. Produces complete folder structure, config files, and boilerplate.

PROMPT

```
Create a new [web app / API / CLI tool] called [name].
Purpose: [what it does, who it serves].
Stack: [framework] with [language], [database], [CSS].
Structure: Separate [routes], [services], [models].
Include: README, .gitignore, .env.example, health check.
Do not include: [things to skip for now].
```

2. Code Review

Use when you want AI to review existing code for quality, security, or performance issues.

PROMPT

```
Review this [language] code for [security / performance /
maintainability]. Context: part of a [app type] that
[description]. For each issue: specific location, why it's
a problem, corrected code, severity (critical/warning/
suggestion). Note positive patterns worth keeping.
```

3. Debugging

Use when you're stuck on a bug and need systematic help finding the root cause.

PROMPT

```
Bug in my [language/framework] app.
Expected: [what should happen]
Actual: [what happens instead]
Error: [full error output]
Code: [relevant section]
Already tried: [attempt 1], [attempt 2]
Environment: [OS, versions, deps]
Identify root cause and explain why it's occurring.
```

4. Architecture Design

Use when planning a new system or major feature that requires architectural decisions.

PROMPT

```
Design architecture for [system/feature].
Requirements: [list key requirements]
Constraints: users/load, budget, timeline, compliance,
existing systems to integrate with.
Provide: 2-3 options with Mermaid diagrams, trade-offs,
recommended approach with reasoning, key risks, and
implementation roadmap.
```

5. Refactoring Existing Code

Use when improving code that already works but needs better structure. Most real development time is here.

PROMPT

```
I have an existing [framework] app. Current state:
[describe the mess honestly—file sizes, mixed concerns,
missing patterns]. I need you to:
1. [Specific refactoring goal]
2. [Second goal]
Keep [what must stay the same].
Maintain backward compatibility.
Start with new folder structure, then refactor [one area]
as an example I can follow for the rest.
```

6. Docker Setup

Use when containerizing an application for the first time.

PROMPT

```
Create production-ready Docker config for my [stack].
Runtime: [Node 20 / Python 3.12 / etc.]
Build: [npm run build / etc.] Entry: [npm start / etc.]
Port: [port]. Dependencies: [db, cache, queue].
Provide: Multi-stage Dockerfile, docker-compose.yml,
.dockerignore, health check. Run as non-root user,
pin base image versions, minimize final image size.
```

7. CI/CD Pipeline

Use when setting up automated testing and deployment with GitHub Actions.

PROMPT

```
Create GitHub Actions CI/CD for my [stack].
On PR: lint, type check, test, security scan.
On merge to main: full tests, build Docker image, push
to [registry], deploy to [staging].
On release tag: deploy to production with [strategy].
Include: dependency caching, parallel jobs, Slack
notifications on failure, environment secrets handling.
```

8. Security Audit

Use when you want a thorough security review before deployment.

PROMPT

```
Security audit of my [app type].
Stack: [frontend, backend, database, hosting].
Auth: [how users log in]. Data: [types of sensitive data].
External: [third-party APIs, payment processors].
[Paste code or describe architecture]
Evaluate against OWASP Top 10. Check: injection, auth,
data exposure, misconfig, dependencies, API security,
infra security. Severity + specific remediation for each.
```

9. Database Schema

Use when designing a new database or restructuring an existing one.

PROMPT

```
Design database schema for [app/feature].
Entities: [Entity 1: fields, purpose],
[Entity 2: fields, purpose].
Relationships: [how entities connect].
Database: [PostgreSQL / MySQL / etc.]
ORM: [Prisma / SQLAlchemy / none].
Provide: Complete schema with constraints, indexes for
common queries, migration files, seed data script,
notes on normalization decisions.
```

10. Tech Stack Evaluation

Use at the start of any new project to make informed technology choices.

PROMPT

```
I'm building [project description] for [audience].
Constraints: team knows [languages], budget is [range],
timeline is [weeks/months], expected scale [users].
Regulatory: [any compliance requirements].
Recommend a complete tech stack: frontend framework,
backend, database, ORM, hosting, and CI/CD.
Provide 2-3 options with trade-offs. Pick one and
explain why it's right for these specific constraints.
```

Want the Full Playbook?

These 10 prompts are extracted from **Appendix B** of the full book. The complete guide includes **10 chapters** covering everything from architectural thinking to cloud deployment to security—plus detailed walkthroughs showing exactly why each prompt element matters.

Get the full book:

stan.store/coopertech

"The more you understand the system, the more powerful your AI partnership becomes."

— Josh Cooper