Agentic Coding with Claude Code

Laravel edition

Peter Elmered

Vem är jag?

- Jobbat med PHP i över 20 år
- Full-stack med fokus på Backend, Arkitektur och DevOps
- Älskar Queues, Actions och DDD
- Har använt Claude Code sedan April

Dagens tema:

Agenter!

Dagens tema:

Kodagenter!

Vad är Claude Code och hur fungerar det?

- Agentisk kodassitent
- Körs via terminalen
- Utvecklad av Anthropic (Företaget bakom Claude-modellerna)
- MCP & Tools
- Slash commands
- Subagenter

Starta ett nytt projekt

- Använd ett starter-kit!
- Installera Laravel Boost och de paket som du tror att du kommer behöva.
- Kör `boost:install`
- Gör anpassningar av din `CLAUDE.md` genom att ändra i `.ai/guidelines`
- Jag brukar skriva över
  `.ai/guidelines/foundation.blade.php`
  `.ai/guidelines/laravel/core.blade.php`
  och lägga till
  `.ai/guidelines/architecture.blade.php`

Context is King

- Så lite som möjligt, men inte för lite
- För mycket kontext ger “Context rot”
- Försök hålla den relevant för vad du jobbar med

Context is King

Användbara slash commands:

- `/context`
- `/clear`
- `/resume`
- `/compact`
- Double esc

Context is King

MCP

Fantastiskt koncept, men lämnar lite att önska än så länge

Context is King

Tips:

Gör research och ställ frågor i en separat AI-chatt

Context is King

Claude Code är på gång med kontextfönster på 1M tokens

(nuvarande är 200k)

Subagenter

- Specialiserade agenter för specifika uppgifter
- Egen system prompt
- Helt eget kontextfönster
- Flera kan göras parallelt i bakgrunden
- Input (prompt)
  -> subagent
  -> output (text som stoppas in i huvudagentens kontext)

Varför Subagenter?

- För avgränsade flöden
- När du vill skicka med en egen system prompt
- Du vill inte förorena kontext i huvudagenten

Bra exempel:

- Code review
- Skapa dokument
- Köra tester

Hur får man Claude Code att göra som man vill?

Planning mode!

Tips: Använd “Opus Plan Mode” för att automatiskt byta till Opus när du planerar

Hur får man Claude Code att göra som man vill?

Product/Feature Requirements Documents

Product/Feature Requirements Document

Det här är ett dokument som i detalj beskriver en feature eller produkt. Innehåller saker som:

- Beskrivning
- Syfte
- Affärsvärde
- User stories
- Funktionella, tekniska och icke funktionella krav
- Success metrics
- Krav på testning
- Implementationsdetaljer

Product/Feature Requirements Document

- Skapa med slash kommando:
  `/create-feature-requirement-doc <prompt>`
- Vanligen använder jag en Linear issue ID som prompt:
  `/create-feature-requirement-doc id-35`

Product/Feature Requirements Document

- Starta implementation med slash kommando:
  `/implement-feature-requirement-doc docs/features/my-feature.md`

Checkpoints

Spara din kod ofta för att få återställningspunkter

- Skapa checkpoints genom att commita ofta
- Local history i PHPStom en live-saver som backup när AI gör bort sig

Git Worktrees

- Sepparat katalog, annan branch, samma projekt
- Jobba på flera olika saker parallellt
- Överanvänd inte

Hooks för automatiska kontroller

Exempelvis:

- Pint / Lint
- Rector
- Köra tester

Hooks för automatiska kontroller

Några möjliga hookar:

- PreToolUse / PostToolUse
- Notification
- UserPromptSubmit
- Stop
- PreCompact

GitHub workflows

- Automatisk code review
- Installera med: `/install-github-app`
- Custom slash commands för att jobba med PRs:
  - /create-pr , /update-pr
  - /fix-pr 54

Custom slash commands

- Slash commands är ungefär som återanvändbara promptar
- Några som jag använder mycket:
  - /linear-issue <id>
  - /review-code
  - /review-design
  - /fix-tests
  - /create-architecture-decision-record.md
  - /create-feature-requirement-doc.md

Custom slash commands

https://github.com/hesreallyhim/awesome-claude-code

Claude Code i telefonen!

https://happy.engineering/

Officiella best practices guiden från Anthropic

https://www.anthropic.com/engineering/claude-code-best-practices

Tack!