

Die Claude-Code-Auftragsvorlagen

Begleitpaket zum Buch „Durchstarten mit Claude Code"

von Thorsten Behder

Zehn Auftragsvorlagen aus dem Buch als PDF: vom lesenden Erkunden eines Repositorys über den Bugfix mit Regressionstest bis zum unabhängigen Review.

PROMPT-SET

Zehn Auftragsvorlagen aus dem Buch, wörtlich übernommen. Sie folgen dem Weg des Lernprojekts vom lesenden Erkunden über den ersten Bugfix bis zum unabhängigen Review.

Erst lesen, dann beauftragen

Aus dem Buch — Kapitel „Zuerst lesen, nicht ändern" und „Gute Aufgaben statt magischer Prompts"

1. Codebase-Erkundung

```
Du arbeitest ausschliesslich lesend. Aendere keine Datei
und fuehre keine Kommandos aus, die etwas veraendern.

Ich bin neu in diesem Projekt. Beantworte mir nacheinander,
und zwar so, dass ich jede Aussage selbst nachpruefen kann:

1. Wie starte ich die Anwendung, wie fuehre ich die Tests aus?
2. Welche Datenmodelle gibt es, mit welchen Feldern und Beziehungen?
3. Welche API-Endpunkte gibt es und in welchen Dateien liegen sie?

Regeln für deine Antwort:
- Nenne zu jeder Aussage die Datei und die Funktion.
- Wenn du etwas vermutest statt weisst, schreibe VERMUTUNG davor.
- Fasse dich kurz. Keine Wiederholung meiner Frage.
```

2. Vierfelder-Schema

```
ZIEL
Nach einem falschen Passwort sieht die Nutzerin eine Fehlermeldung
auf der Login-Seite statt einer leeren Seite.

KONTEXT
Betroffen ist vermutlich der Auth-Flow zwischen
backend/app/auth.py und frontend/src/pages/Login.tsx.
Untersuche zuerst und erklaere mir die vermutete Ursache,
bevor du etwas aenderst.

GRENZEN
Aendere keine oeffentlichen API-Routen.
Das Response-Schema von POST /api/login bleibt unveraendert.

VERIFIKATION
1. Behebe die Ursache.
2. Ergaenze einen Regressionstest in backend/tests/test_auth.py.
3. Fuehre pytest backend/tests/test_auth.py aus.
4. Zeige mir den relevanten git diff und das Testergebnis.
```

Ändern mit Nachweis

Aus dem Buch — Kapitel „Ändern, testen, zurückrollen"

3. Bugfix mit Regressionstest

ZIEL
Nach einem falschen Passwort antwortet POST /api/login mit Status 401 und einem Fehlerobjekt, und Login.tsx zeigt diese Meldung über die bestehende Komponente ErrorBanner.tsx an.

KONTEXT
Ursache wie eben besprochen: auth.py wirft eine Exception, die niemand behandelt. ErrorBanner.tsx existiert bereits und wird nirgends verwendet. Benutze sie, schreibe keine neue Komponente.

GRENZEN
Der Erfolgsfall von POST /api/login bleibt unveraendert.
Keine anderen Routen anfassen, keine neue Abhaengigkeit, services.py nicht anfassen.
Wenn dir eine dieser Grenzen im Weg steht: nachfragen, nicht selbst entscheiden.

VERIFIKATION
1. Schreibe ZUERST den Regressionstest in backend/tests/test_auth.py und fuehre ihn gegen den unveraenderten Code aus. Zeige mir, dass er rot ist.
2. Behebe danach die Ursache.
3. Fuehre pytest backend/tests/test_auth.py erneut aus und zeige die vollstaendige Ausgabe.
4. Zeige mir git diff --stat.

4. Feature mit Definition of Done

ZIEL
GET /api/profile liefert für die angemeldete Nutzerin ein
JSON-Objekt mit id, email und created_at. Ohne gueltige
Session antwortet der Endpunkt mit Status 401.

KONTEXT
Route nach backend/app/routes/auth_routes.py, das Modell User
steht in backend/app/models.py, die Schemas in
backend/app/schemas.py. Die Session-Pruefung existiert bereits
für die Board-Routen, benutze denselben Mechanismus.

GRENZEN
Bestehende Routen bleiben unveraendert.
password_hash darf im Response nicht vorkommen.
Keine neue Abhaengigkeit, keine Aenderung an services.py,
keine Aenderung am Frontend.

VERIFIKATION
1. Schreibe zuerst drei Tests in backend/tests/test_auth.py:
   den Erfolgsfall, den Fall ohne Session unter dem Namen
   test_profile_requires_session, und einen Test, der prüft,
   dass password_hash nicht im Response steht.
2. Fuehre sie gegen den unveraenderten Code aus und zeige mir,
   dass alle drei rot sind.
3. Implementiere danach.
4. Fuehre pytest backend/tests/test_auth.py und
   ruff check backend aus, zeige beide Ausgaben.
5. Zeige mir git diff --stat.

Regeln, Skills und Hooks

Aus dem Buch — Kapitel „Kontext und Projektgedächtnis", „Permissions, Sandbox und Auto Mode" und „Skills, Hooks und MCP"

5. Projektregeln formulieren

```
Lies die vorhandene CLAUDE.md und pruefe sie Zeile für Zeile
gegen diese Frage:

  Wuerde das Weglassen dieser Zeile dazu führen, dass du bei
  einer typischen Aufgabe in diesem Projekt einen Fehler machst?

Gib mir eine Tabelle mit drei Spalten aus:
Zeile | behalten oder streichen | Begruendung in einem Satz.

Streiche vorschlagsweise alles, was
- du durch Lesen des Codes selbst herausfindest,
- eine Standardkonvention der Sprache wiederholt,
- eine Bewertung ohne pruefbaren Inhalt ist.

Aendere die Datei noch nicht. Ich entscheide danach.
```

6. Threat-Model-Übung

```
ZIEL
Eine Liste der Stellen, an denen dieses Repository Inhalte
in den Kontext eines Agenten bringt, die nicht von mir stammen.

KONTEXT
Untersuche das gesamte Projekt. Beruecksichtige mindestens:
Markdown-Dateien in docs/, Kommentare in Konfigurationsdateien,
Skripte, die externe Daten holen, sowie Ausgaben von Build- und
Testkommandos, die im Terminal landen.

GRENZEN
Aendere keine Datei. Fuehre keine Kommandos aus, die etwas
herunterladen. Erstelle keine Beispielangriffe.

VERIFIKATION
Gib eine Tabelle aus mit den Spalten:
Quelle, Wer kann sie beeinflussen, Was waere die Folge,
Welche deny- oder ask-Regel wuerde das begrenzen.
Sortiere nach Folgenschwere, schwerste zuerst.
Nenne am Ende die drei Eintraege, bei denen du dir unsicher bist.
```

7. Ein Skill aus vorhandenem Code destillieren

```
Lies backend/app/routes/ vollständig und
backend/app/schemas.py.

Leite daraus die Konventionen ab, die in diesem Projekt
tatsaechlich gelten, nicht die, die gelten sollten. Achte auf:
Routenbenennung, Statuscodes, Fehlerformat, Validierung,
Ablage der Antwortschemas.

Gib mir eine nummerierte Liste. Zu jeder Konvention:
- die Regel in einem Satz,
- zwei Fundstellen mit Datei und Zeilennummer,
- ob sie durchgehend gilt oder Ausnahmen hat.

Schreibe noch keine Datei. Ich streiche zuerst.
```

8. Hook-Skript schreiben lassen

```
Schreibe ein Bash-Skript nach .claude/hooks/lint.sh für einen
PostToolUse-Hook mit dem Matcher Edit|Write.

Verhalten:
1. Lies das Ereignis-JSON von stdin.
2. Zieh den Wert von .tool_input.file_path heraus. Wenn du dafuer jq
   benutzt, pruefe vorher mit command -v jq, ob es ueberhaupt da ist,
   und brich sonst mit exit 2 ab statt mit exit 0.
3. Endet der Pfad auf .py und liegt unter backend/, fuehre
   ruff check auf genau dieser Datei aus.
4. Bei Befunden: Ausgabe von ruff nach stderr, exit 2.
5. In allen anderen Faellen: exit 0, keine Ausgabe.

Randbedingungen:
- Kein Aufruf ohne Dateipfad darf das Skript zum Absturz bringen.
- Keine Ausgabe auf stdout.
- Windows-Backslashes im Pfad vorher zu Schraegstrichen normalisieren.

Zeige mir danach die Registrierung in .claude/settings.json.
```

Automatisieren und gegenprüfen

Aus dem Buch — Kapitel „Automation, Headless-Betrieb und CI/CD" und „Subagents, Agent Teams und die Grenzen der Vervielfachung"

9. Headless-Review

ZIEL
Pruefe den uebergebenen Diff auf genau drei Dinge:
fehlende Fehlerbehandlung, veraenderte oeffentliche
API-Routen, fehlende oder nur scheinbare Tests.

KONTEXT
Der Diff steht auf stdin. Das Projekt ist eine FastAPI-
Anwendung mit pytest. Aendere nichts, lies nichts ausserhalb
des Diffs.

GRENZEN
Keine Stilkritik, keine Umbenennungsvorschlaege, keine
Bewertung von Formatierung.

AUSGABE
Ein JSON-Objekt mit den Feldern findings (Liste aus
datei, zeile, kategorie, begruendung) und blocking
(true, wenn mindestens ein Fund eine oeffentliche Route
veraendert oder einen Test entfernt).

10. Unabhängiger Reviewer

AUFTRAG
Pruefe den angehaengten Diff gegen die folgende Anforderung.
Du kennst weder den Loesungsweg noch die Begruendung des Autors
und sollst beides auch nicht rekonstruieren.

ANFORDERUNG
Der CSV-Export eines Boards muss für beliebige Aufgabentitel
gueltiges CSV erzeugen, einschliesslich Titeln mit Komma,
Anfuehrungszeichen und Zeilenumbruch.

PRUEFKRITERIEN
1. Nenne jeden Eingabewert, für den die Aenderung noch immer
   ungueltiges CSV erzeugen wuerde.
2. Pruefe, ob der ergaenzte Test ohne die Aenderung fehlschlagen
   wuerde. Begruende das am Testcode, nicht an der Absicht.
3. Wenn du keinen Mangel findest, schreibe, welche drei Faelle du
   geprueft hast und welche du nicht prüfen konntest.

Die elf Labore mit dem Lernprojekt taskboard, das Permission- und Hook-Cookbook, die Checklisten zu Security, Datenschutz, EU AI Act und Kosten sowie das API-Cookbook in Python

> und TypeScript stehen im Buch.

Dieses Paket gehört zum Buch „Durchstarten mit Claude Code”

https://www.amazon.de/dp/B0HMLWR5MJ

tbai.cloud/buch-durchstarten-claude-code.html

Autor und inhaltlich verantwortlich

Thorsten Behder

Weilheimer Str. 21A

81373 München

thorsten@behder.de



Gestaltung und Layout: tbai.cloud