Givery, Inc.

HANDS-ON GUIDE / DAY 1 ・ 2026 年 5 月版

ハンズオンガイド Day 1

機能を 1 つずつ投入しながら
α（案件ヘルスチェックダッシュボード）を作る

提供	Givery 株式会社 / 講師 安田 光喜
配布先	PwC コンサルティング合同会社 御中
所要	9:00 - 17:00（実働 400 分、昼休憩 60 分、午前午後の休憩各 10 分）
題材	マツカゼ製薬 海外子会社向け統合 CRM 刷新（MTKZ-CRM-2026）
版数	v2.0（2026 年 5 月 3 日）

本書は受講者の手元で開く資料です。タイムテーブル、各演習の詳細手順、API 仕様、トラブルシューティング、コマンドリファレンスを 1 冊に集約しています。

SECTION 01 ・ OVERVIEW

1. 本日のゴール

AI の理論解説はほぼ行いません。Claude Code をターミナルから起動し、案件ヘルスチェックダッシュボードを 4 段階で育てます。素朴に作る、CLAUDE.md で固める、Skills / Subagents / Commands で再利用可能な形に切り出す、verification loop（code-reviewer / security-auditor）で品質ゲートを通す。最終 120 分は自律開発で持ち帰り設計案を作ります。

形式	オンサイト ／ 7 時間 ／ 実働 400 分
環境	Windows + WSL2 Ubuntu 22.04 / macOS、VSCode + Claude Code（CLI または拡張）
主要ツール	Claude Code CLI、VSCode（Remote - WSL）、FastAPI、SQLite、React + TypeScript
配布素材	`pwc-cc-handson_受講者用.zip`（マツカゼ CRM 案件ヘルスチェックダッシュボードのスタブ）

1.1 Day 1 の到達点

1. 雛形リポジトリに対し、CLAUDE.md / Skills / Subagents を投入しながら新機能を 1 つ追加できている
2. ヒアリング起こし 3 名分を Risk テーブルに自動登録できている
3. ステージ済み差分を `code-reviewer` Subagent でレビューできる
4. ハードコード秘密を `security-auditor` Subagent で検出できる
5. Haiku / Sonnet / Opus の 3 モデルで同じレビューを実行し、コスト差を実測している
6. 自律開発（120 分）で、起こしを再起点に新機能を 1 つ自分で要件定義 → 設計 → 実装している

> **本研修のスタンス**
> Claude Code の生成結果は常に疑って読みます。同じ依頼でも CLAUDE.md / Skills / Custom Commands の整備状況で品質は変わります。各演習で素朴な状態と整備した状態を比較し、設定差分を確認してから次へ進みます。

SECTION 02 • AGENDA

2. タイムテーブル

各時間枠は目安です。詰まる箇所が出たら講師が現場判断で延長します。S02 ベストプラクティス講義は座学編 PDF を投影しながら進みます。

開始	終了	区分	内容
09:00	09:30	S01 オープニング	環境確認、配布物チェック、自己紹介
09:30	11:00	S02 ベストプラクティス講義	座学編 PDF を投影（オリエンテーション）
11:00	11:10	休憩	10 分
11:10	11:35	S03-1	Exercise 1：CLAUDE.md（25 分）
11:35	12:00	S03-2	Exercise 2：Skills（25 分）
12:00	12:25	S03-3	Exercise 3：Subagents + Command（25 分）
12:25	12:30	振り返り（午前）	5 分
12:30	13:30	昼休憩	60 分
13:30	13:55	S03-4	Exercise 4：code-reviewer（25 分）
13:55	14:20	S03-5	Exercise 5：security-auditor（25 分）
14:20	14:40	S03-6	Exercise 6：モデル選択とコスト管理（20 分）
14:40	14:50	休憩	10 分
14:50	16:50	S04	Exercise 7：自律開発（120 分）
16:50	17:00	クロージング	提出物確認、Day 2 予告

SECTION 03 · PROJECT

3. 題材：案件ヘルスチェックダッシュボード（FastAPI + SQLite + React）

Project（案件）・Stakeholder（関係者）・StatusReport（週次レポート）・Risk（リスク）の 4 エンティティ。架空クライアント マツカゼ製薬の海外子会社向け統合 CRM 刷新（MTKZ-CRM-2026）を題材に、ヒアリング起こしや週次レポートを起点にリスクを構造化し、ダッシュボードで横断把握する Web アプリを育てます。

3.1 配布フォルダ構成

```
pwc-cc-handson_受講者用/
├── README.md                       配布物一覧と起動手順
├── CLAUDE.md                       Day 1 で育てていく最小版（5 行程度）
├── AGENTS.md                       他 AI エージェント向け共通ガイド（完成版）
├── DESIGN.md                       UI/UX 方針（Day 2 で育てる）
├── docs/
│   ├── api-spec.md                  REST API 全仕様
│   ├── source-materials/
│   │   ├── client-brief.md          クライアント概要
│   │   ├── legacy-system.md         旧システム情報
│   │   └── interview-transcripts/
│   │       ├── 01_tanaka.md         70 分・PMO 田中真司氏
│   │       ├── 02_sato.md           65 分・要件責任 佐藤美咲氏
│   │       └── 03_yamada.md         55 分・現場担当 山田直人氏
│   └── reference/                  公式抜粋集
├── exercises/
│   ├── day1/                        exercise1〜7 の md + answers/
│   └── day2/
├── .claude/
│   ├── settings.json               モデル設定 + Hooks（Day 2 で育てる）
│   ├── skills/
│   │   ├── risk-classifier/         Day1 Ex2 で受講者が記入
│   │   ├── transcript-extractor/    Day1 Ex2 で受講者が記入
│   │   └── api-spec-checker/        Day1 Ex5 で受講者が記入
│   ├── commands/                   Day1〜2 で受講者が育てる
│   ├── agents/                     code-reviewer / security-auditor / risk-extractor
│   └── hooks/                      Day 2 Ex3 で受講者が育てる
├── .mcp.json                       Day 2 Ex5 で MCP サーバー定義
├── frontend/                       React 18 + TypeScript 5 + Vite + Recharts
└── backend/                        FastAPI 0.110 + SQLAlchemy 2.x + SQLite + Anthropic SDK
```

3.2 レイヤードアーキテクチャ

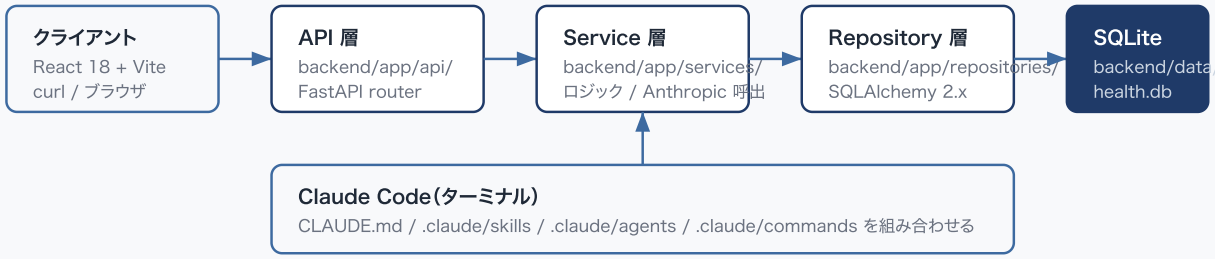

図 1：レイヤードアーキテクチャ。api → service → repository → model の順序を守るのが CLAUDE.md の規約

3.3 主要 API 一覧

メソッド	パス	用途
GET	`/api/projects`	案件一覧(status / phase / owner でフィルタ)
GET	`/api/projects/{id}`	詳細 + 関連 Stakeholder / StatusReport / Risk
GET	`/api/risks`	Risk 一覧(severity / category でフィルタ)
GET	`/api/risks/by-severity/{severity}`	severity 別(CRITICAL / MAJOR / MINOR / INFO)
GET	`/api/risks/by-category/{category}`	category 別(SCHEDULE / SCOPE / QUALITY / COST / TEAM / EXTERNAL)
POST	`/api/projects/{id}/analyze`	起こし(transcript)を取り込み Risk を抽出(Day1 Ex3 で実装)
GET	`/api/metrics/dashboard`	全体ダッシュボード集計
GET	`/api/metrics/by-pm`	PM 別集計

完全な API 仕様は配布フォルダの `docs/api-spec.md` を参照してください。本研修中は VS Code でこのファイルを開きながら進めます。

3.4 起動手順

```
# バックエンド
cd backend
python -m venv .venv
. .venv/bin/activate            # Windows は .venv\Scripts\activate
pip install -r requirements.txt
python -m app.seeds              # 初期データ投入（Project 3 件 / Risk 5 件）
uvicorn app.main:app --reload --port 8000

# Claude Code（リポジトリのルートで起動）
claude
# Welcome to Claude Code と出たら起動成功
# 初回は Pro / Max サブスクリプションのブラウザログイン、
# または社内 LLM 中継基盤の認証情報読み込みが走る
```

> **補足:フロントエンドの起動**
>
> フロントエンド(`frontend/`)の起動は S03 のハンズオン中に Claude Code に依頼します。Node.js が手元にない受講者でも、Claude Code に「frontend の開発サーバーを起動できる状態にして」と指示すれば、必要な環境構築から起動までを案内してもらえます。

> **VS Code(Remote - WSL)を使う理由**
>
> Claude Code は CLI ですが、生成コードの確認と差分レビューは VS Code 側のほうが視認性が高くなります。Remote - WSL 拡張で WSL2 の作業ディレクトリを直接開けるため、Windows ↔ WSL 間のファイルコピーも不要です。本研修では Eclipse ユーザーも VS Code + Remote - WSL に環境を揃えます。

SECTION 04

4. 開始前のセットアップチェック

S01 オープニング前に以下が全部 OK な状態にしてください。NG があれば S01 中に講師に申告してください。

確認項目	確認コマンド	OK 判定
Claude Code 起動	`claude`	"Welcome to Claude Code" 表示
受講者用 ZIP 解凍	`ls pwc-cc-handson_受講者用/`	README.md / CLAUDE.md / backend/ / frontend/ が見える
バックエンド起動	`uvicorn app.main:app --reload --port 8000`	"Application startup complete." 表示
health チェック	`curl localhost:8000/health`	`{"status":"ok"}` が返る
seeds 投入確認	`curl -s localhost:8000/api/projects | jq length`	`3` が返る

SECTION 05

5. Day 1 のデータフロー

Day 1 は同じデータを段階的に処理します。各演習の冒頭の「前提条件」で、下の図のどこから始めるかを確認できます。

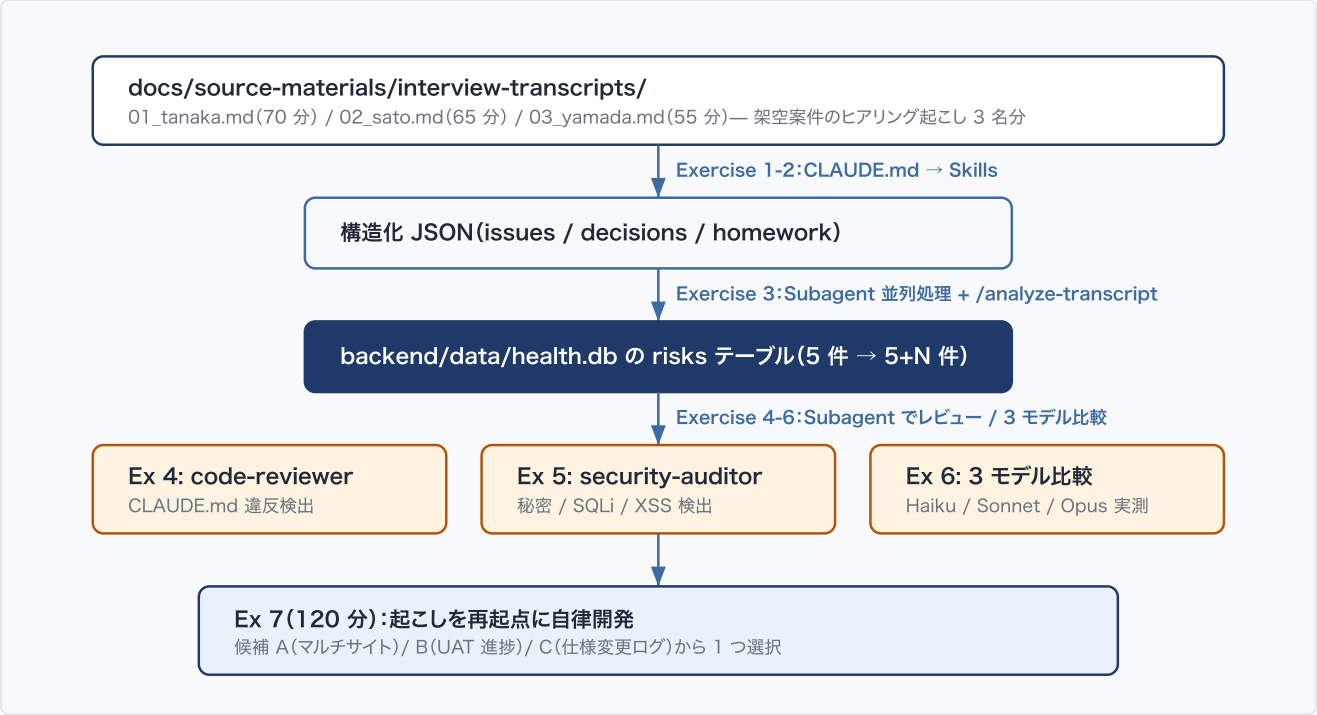

図 2:Day 1 のデータフロー。Ex1 で品質基盤、Ex2-3 で起こしを Risk 化、Ex4-6 で品質ゲート、Ex7 で自走

DAY 1 ・ EXERCISES

6. 演習詳細

各演習の詳細手順は `exercises/day1/exerciseN-*.md` に書かれています。本書には「やること」「対象ファイル」「AI への指示例」「期待される結果」「観察ポイント」を抜粋しています。詳細手順は VS Code でハンズオンガイド側を見ながら、実行しながら進めてください。

EXERCISE 1 ／ 25 分　CLAUDE.md でプロジェクト規約を Claude に渡す

対象ファイル　`CLAUDE.md`（編集対象）／ `backend/app/api/risks.py` ほか Service / Repository

詳細手順　`exercises/day1/exercise1-claude-md.md`

答え合わせ　`exercises/day1/answers/exercise1-claude-md_例.md`

やること

最小版 CLAUDE.md の状態で「素朴な依頼」を投げ、生成物の粗さを観察。次にプロジェクト規約・レイヤー方針・禁止事項を CLAUDE.md に追記し、同じ依頼で品質がどう変わるかを比較します。CLAUDE.md は出力品質を底上げする最初の道具という感覚を体で掴むのが目標です。

AI への指示例（プロンプト 1 / プロンプト 2 共通）

```
backend に GET /api/risks/by-pm/{pm_name} を追加してほしい。
指定された PM が担当する案件に紐づく Risk を全部返す。
```

意図的に抽象的に書きます。レイヤー方針・命名・DTO の有無は伝えません。

CLAUDE.md 追記（5 項目以内）

```
## レイヤー構成と依存方向
api → service → repository → model の順序を守る。
Controller から SQLAlchemy セッションを直接触らない。

## バックエンド規約（5 項目以内）
- 依存性注入は FastAPI の Depends を使う
- DB アクセスは app/repositories/ 経由のみ
- エラーは app/exceptions.py のカスタム例外を投げる
- ロギングは logging.getLogger(__name__)。print は使わない
- 型ヒント必須。mypy --strict で通る状態を維持

## 禁止事項
- API キー・パスワードのハードコード
- exercises/**/answers/ 配下の読み込み
```

期待される結果（プロンプト 2＝CLAUDE.md 追記後）

- `risk_repository.py` に `list_by_pm(session, pm_name)` 等が追加され、Service 経由で呼ばれる
- `RiskRead` DTO で返される
- pm_name が空文字や存在しない場合のエラーパスが実装される
- `print()` ではなく `logger.info()` が使われる
- 既存 URL パターン（`/api/risks/...`）と命名が揃う

確認方法

```
curl -s "http://localhost:8000/api/risks/by-pm/田中" | jq
# 期待：田中真司が担当する MTKZ-CRM-2026 の Risk が複数返る

curl -s -w "\nHTTP %{http_code}\n" "http://localhost:8000/api/risks/by-pm/存在しない人"
# 期待：200 で空配列、または 404
```

観察ポイント

CLAUDE.md に書いた規約が全部反映されたわけではないはずです。反映されなかった項目は、書き方が抽象的だったか、別の優先指示と競合したか。**Claude が読みやすい CLAUDE.md と、人間が読みやすい CLAUDE.md は別物**です。Good / Bad の具体例コードを貼ると反映率が大きく変わります。

早く終わった方向け

CLAUDE.md の規約セクションを 10 項目に増やしてみて、Claude が守らなくなることを実測してください。`@AGENTS.md` の取り込み行を CLAUDE.md から外してみて、生成物がどう変わるかも観察できます。

EXERCISE 2 ／ 25 分　Skills でロジックを再利用可能な形に切り出す

対象ファイル　`.claude/skills/transcript-extractor/SKILL.md` ／ `.claude/skills/risk-classifier/SKILL.md`

詳細手順　`exercises/day1/exercise2-skills.md`

答え合わせ　`exercises/day1/answers/exercise2-skills_例.md`

やること

Exercise 1 で観察した「CLAUDE.md に書けば品質が上がる」事実を、毎回のプロンプトに書かずに済ませます。`transcript-extractor`（起こしから論点を抽出）と `risk-classifier`（severity / category を判定）の 2 つの Skill を作成し、Exercise 3 / 5 で繰り返し呼び出す前段にします。

SKILL.md の frontmatter（例）

```
---
name: transcript-extractor
description: ヒアリング起こしの長文から、論点・決定事項・宿題・固有名詞を構造化抽出する。risk-classifier に渡す前段として使う。
---
```

本文に必ず書く 5 つ

1. 入力フォーマット（`transcript_path`、`project_id` 等）
2. 出力フォーマット（`session` / `issues[]` / `decisions[]` / `homework[]` / `stakeholders_mentioned[]`）
3. 抽出方針（issues は事実だけ、severity 判定は risk-classifier に委ねる）
4. 禁則（推測で要約しない、原文の語尾を保つ）
5. 具体例 1 つ以上（入力 → 出力の対）

AI への指示例

```
docs/source-materials/interview-transcripts/01_tanaka.md を transcript-extractor Skill で処理して、
構造化 JSON を返してください。
```

description 設計の勘所

Skills の発火率は `description` の書き方で決まります。抽象的な description は呼ばれず、**具体的なトリガー語を 3〜5 個並べる**と安定します。チーム運用では description の書き方の例集を CLAUDE.md に明記し、標準化します。

早く終わった方向け

SKILL.md に自社のセキュリティポリシー（例：ログ出力で個人情報マスク必須）を追記し、新しい Finding を分類させます。Skill が自社の判断基準を AI に内蔵させる装置である事実が確認できます。

EXERCISE 3 ／ 25 分　Subagent で並列分析、Command で 1 ターン化

対象ファイル　`.claude/agents/risk-extractor.md` ／ `.claude/commands/analyze-transcript.md` ／ `backend/app/services/risk_extractor.py`

詳細手順　`exercises/day1/exercise3-subagents-command.md`

答え合わせ　`exercises/day1/answers/exercise3-subagents-command_例.md`

やること

3 名のヒアリング起こしを 1 つずつ親コンテキストで処理するとトークン消費もコンテキストも膨らみます。`risk-extractor` Subagent で 3 ファイルを並列処理し、`/analyze-transcript` Custom Command で「抽出 → DB 登録」を 1 行で通します。`backend/data/health.db` の `risks` テーブルが seeds 5 件から増加します。

risk-extractor Subagent の frontmatter

```
---
name: risk-extractor
description: Use this agent in parallel for analyzing multiple interview transcripts and producing structured ris
k JSON per file. Keeps parent context clean. Use when running on 2 or more files at once.
tools: Read, Glob
model: claude-sonnet-4-6
---
```

AI への指示例

```
3 つの Subagent を並列で起動し、各 Subagent に 1 件ずつ起こしを割り当ててください。
- Subagent 1: 01_tanaka.md
- Subagent 2: 02_sato.md
- Subagent 3: 03_yamada.md

各 Subagent は transcript-extractor Skill で構造化し、risk-classifier Skill で severity を付与した
Risk JSON 配列を 1 行で返してください。親エージェントは結果をマージして
backend/app/services/risk_extractor.py 経由で DB に登録します。
```

期待される結果

- 3 件の Subagent が並列起動し、起こし 3 本を独立コンテキストで処理する
- 処理時間が逐次実行と比べて短縮される（`/cost` でトークン消費も比較）
- 親エージェントのコンテキストには要約のみ残り、起こし本文は残らない
- `risks` テーブルが 5 件 → 12〜18 件に増える（`curl -s localhost:8000/api/risks | jq length` で確認）

> **Subagent の使いどころ**
> 複数ファイル / PR を読むが結果だけ親に渡したい、並列化で時間短縮したい、親コンテキストを汚したくない、のいずれかが当てはまる場合に Subagent を選択。対話を継続するタスクは親エージェント側のほうが安定します。

EXERCISE 4 ／ 25 分　code-reviewer Subagent でコミット前自動レビュー

対象ファイル　`.claude/agents/code-reviewer.md` ／ `.claude/commands/review-changes.md`
詳細手順　`exercises/day1/exercise4-code-reviewer.md`
答え合わせ　`exercises/day1/answers/exercise4-code-reviewer_例.md`

やること

人間が時間を取りにくい「コードレビュー」を Subagent に肩代わりさせます。意図的に CLAUDE.md 違反を含む差分を作り、code-reviewer Subagent が CLAUDE.md と既存実装に照らして自動検出するか確認します。Day 1 提出物の品質ゲートにします。

意図的なステージング（NG 実装の例）

```
# backend/app/api/risks.py の末尾に追記
@router.get("/by-pm-bad/{pm_name}")
def list_risks_by_pm_bad(pm_name: str, session: Session = Depends(get_session)) -> list[dict]:
    # NG: Repository を経由せず直接 SQL を組み立て、DTO も使わない
    rows = session.execute(
        text(f"SELECT * FROM risks WHERE owner LIKE '%{pm_name}%'")
    ).fetchall()
    return [dict(r._mapping) for r in rows]
```

AI への指示例

```
git add backend/app/api/risks.py
claude
> /review-changes
```

期待される結果（CRITICAL / MAJOR で複数件）

- **CRITICAL**：SQL 文字列フォーマット（インジェクション可能性）
- **MAJOR**：Repository 層を経由せず Controller から直接 SQL を発行（CLAUDE.md 違反）
- **MAJOR**：DTO（Pydantic）を使わず生 dict で返却
- **MINOR**：URL パスが既存の命名規約と揃っていない（`by-pm-bad`）

> **Custom Command が効く理由**
> 素プロンプトで毎回「PR 差分をレビューして severity 付きで返して」と書くと、文章のブレで結論が変わります。`/review-changes` に固定すれば、誰が実行しても同じフォーマットで結論が返ります。これが PR 運用に組み込める形です。

EXERCISE 5 ／ 25 分　security-auditor Subagent でセキュリティスキャン

対象ファイル　`.claude/agents/security-auditor.md` ／ `.claude/commands/security-scan.md`
詳細手順　`exercises/day1/exercise5-security-auditor.md`

答え合わせ `exercises/day1/answers/exercise5-security-auditor_例.md`

やること

セキュリティ観点を Subagent 化することで、人間レビュアの取りこぼしを防ぎます。3 種類の違反（API キーハードコード、SQL 文字列フォーマット、XSS リスク）を意図的に仕込み、security-auditor が検出するか確認します。**演習後に必ず違反コードを削除**してください。

仕込む 3 違反

- 違反 A：`backend/app/services/llm_client.py` に `api_key = "sk-ant-fakekey-..."` をハードコード
- 違反 B：`backend/app/repositories/risk_repository.py` に `list_risks_unsafe`（SQL 文字列フォーマット）を追加
- 違反 C：`frontend/src/components/RiskDetail.tsx` に `dangerouslySetInnerHTML` を入れる

AI への指示例

```
claude
> /security-scan
```

期待される結果（severity 別件数）

- **CRITICAL**：API キーハードコード（違反 A）／ SQL インジェクション可能性（違反 B）
- **MAJOR**：DOMPurify 不使用の `dangerouslySetInnerHTML`（違反 C）
- **MINOR / INFO**：依存古さ、ログ出力での個人情報、Cookie の SameSite 未指定 等

> **注意：違反コードを必ず削除**
>
> 演習後、3 違反は必ず削除してください。`git restore` で元の状態に戻すのが安全です。本番リポジトリで同じ作業を行う場合、絶対に commit しないこと。

EXERCISE 6 ／ 20 分　モデル選択とコスト管理

対象ファイル `.claude/agents/code-reviewer.md`（`model` フィールド変更）／ `CLAUDE.md`（モデル選択ルール追記）
詳細手順 `exercises/day1/exercise6-model-cost.md`
答え合わせ `exercises/day1/answers/exercise6-model-cost_例.md`

やること

Exercise 4 と同じステージング差分に対し、code-reviewer の `model` を Haiku 4.5 → Sonnet 4.6 → Opus 4.7 に切り替えて 3 回実行。検出件数とトークン消費を計測します。最後に CLAUDE.md にモデル選択ルールを追記します。

3 モデルでの実行

```
# 1) Haiku 4.5 に切り替えて実行
sed -i 's/^model: .*/model: claude-haiku-4-5/' .claude/agents/code-reviewer.md
claude
> /review-changes
> /cost     # トークン消費とコスト確認、結果メモ

# 2) Sonnet 4.6 で再実行
sed -i 's/^model: .*/model: claude-sonnet-4-6/' .claude/agents/code-reviewer.md
> /clear
> /review-changes
> /cost

# 3) Opus 4.7 で再実行
sed -i 's/^model: .*/model: claude-opus-4-7/' .claude/agents/code-reviewer.md
> /clear
> /review-changes
> /cost
```

期待される結果

モデル	検出件数(目安)	1 回あたりコスト	使い分け
Haiku 4.5	4-6 件(取りこぼしあり)	$0.005 前後	CI 前段ふるい、軽量レビュー
Sonnet 4.6	7-9 件	$0.03 前後	常用
Opus 4.7	9-11 件(業務文脈の指摘あり)	$0.10 前後	重要 PR、複雑なリファクタ

CLAUDE.md に追記するモデル選択ルール(例)

```
## モデル選択ルール
- Haiku 4.5 : CI の前段、フォーマッタ的な単純判定、要約
- Sonnet 4.6 : 日次レビュー、コーディング全般、デバッグ
- Opus 4.7 : 重要 PR レビュー、複雑なリファクタ、複数ファイル横断
.claude/agents/ 配下では各 agent の frontmatter に model を固定する。
```

> **早く終わった方向け**
>
> `/security-scan` も同じ 3 モデル比較を行います。code-reviewer と security-auditor で「適切なモデルが違う」かどうかを観察してください。一般に security-auditor は Opus が真価を発揮しやすい領域です。

EXERCISE 7 ／ 120 分　自律開発:起こしを再起点に新機能を作る

対象　下記 A / B / C から 1 つ選択。Day 1 提出物のメイン

詳細手順　`exercises/day1/exercise7-autonomous.md`

答え合わせ　`exercises/day1/answers/exercise7-autonomous_例.md`(候補 A の参考解のみ収録)

やること

ここまでで揃った CLAUDE.md / Skills / Subagents / Commands を全部使って、新規機能を「自分で設計して、自分で Claude に投げる」体験をします。最初の 1 行の指示から完成までを自走するパートです。**詰まっても講師は「ヒントのみ」を返します。**

候補 A:マルチサイト対応の準備

シンガポール現地保管要請(田中ヒアリング)を踏まえ、Project に `data_locality`(`HQ_ONLY` / `MULTI_SITE`)を追加し、ダッシュボードで data_locality 別の集計ができる状態にする。

候補 B：UAT 進捗トラッキング

UAT 計画書未着手リスク（田中ヒアリング）を踏まえ、UAT 計画・実施・完了の 3 ステータスを Project に紐づけ、ダッシュボードで UAT 進捗が可視化できる状態にする。

候補 C：仕様変更ログ

3 月以降に発生した仕様変更 27 件（佐藤ヒアリング）を構造化保管する `SpecChange` モデルを追加し、変更カテゴリ・発生日・担当者で集計できる状態にする。

120 分の進め方（目安）

- 0:00-0:10　要件メモ：ゴールを 3 行で書き出す（書かずに走り出すと散漫になる）
- 0:10-0:30　設計：エンドポイント・テーブル・画面の差分を箇条書き
- 0:30-1:40　実装：Skills / Subagents / Commands を活用して進める
- 1:40-1:55　`/review-changes` + `/security-scan` で品質ゲート、修正
- 1:55-2:00　提出物整理（README に追記、スクリーンショット）

> **研修の最終成果物は設計図**
> 3 日間で作る成果は「動くコード」よりも「自社チームに持ち帰れる設計判断」です。Exercise 7 の進行ログを記録に残し、Day 2 のリフレクションと自社展開計画の素材にします。

SECTION 07

7. Day 1 到達チェックと提出物

Day 1 到達チェック

- 素朴プロンプトで生成されたコードと CLAUDE.md 追記後のコードの差分を 5 つ以上書き出した
- `.claude/skills/transcript-extractor/SKILL.md` と `risk-classifier/SKILL.md` が動作する
- 3 名分のヒアリング起こしから抽出された Risk が `backend/data/health.db` に登録された
- `/review-changes` で意図的な NG 差分が CRITICAL / MAJOR で検出される
- `/security-scan` で 3 違反のうち少なくとも 2 つが CRITICAL で検出される
- 3 モデル比較の結果が表として手元にある
- Exercise 7 の自律開発で機能 1 つが画面で動く状態になっている

7.1 提出物（Day 1 終了時に各自）

提出方法は当日アナウンス（社内チャットで講師宛 DM、またはメール）。

- 動く成果物：バックエンド + フロントエンドが起動し、Exercise 7 で追加した機能が画面で確認できる状態
- README.md に追記：何を作ったか / どんな指示を使ったか（5 行以内）
- 印象に残ったプロンプト 3 つ（一番効いたもの / 一番苦戦したもの / Exercise 1 と比べて明らかに進化したもの）
- レビュー結果のスクリーンショットまたはコピペ：`/review-changes` と `/security-scan` の CRITICAL / MAJOR 件数
- 3 モデル比較の表（Haiku / Sonnet / Opus の検出件数とコスト）

SECTION 08 ・ FAQ

8. よくある質問

Q1：Exercise 1 で生成されるコードが、答え（完成形）と違う

問題ありません。CLAUDE.md の書き方によって出力は変わります。完成形は一例で、受講者本人の書いた CLAUDE.md と整合していれば OK です。Day 2 のリフレクションで自分の CLAUDE.md と完成形を並べて比較します。

Q2：Skill が呼び出されない

`description` フィールドが抽象的すぎる可能性があります。「いつ呼び出すか」の判断基準として機能する具体的な名詞（例：`severity` / `Risk` / `transcript`）を 2 つ以上含めてください。発火率は description の精度で決まります。

Q3：Subagent の出力が JSON にならない

Subagent の本文に「JSON 以外の説明文を付けない」と明記し、出力例を 1 つ以上載せてください。それでも崩れる場合は Sonnet 以上のモデルに切り替えます（Haiku は厳密フォーマット出力が苦手）。

Q4：Skill と Custom Command の違いは？

Skill は「特定の処理ロジック」、Custom Command は「ワンショットで呼び出すワークフロー」。同じロジックを複数 Command から呼ぶなら Skill にします。Command は通常 1 ターンで完結、Skill は複数ターンで再利用される想定です。本研修では `risk-classifier` Skill を `/analyze-transcript` と `/review-changes` の両方が呼びます。

Q5：Subagent を Skill で代用できるか

部分的には可能ですが、並列実行とコンテキスト保護は Subagent の専売です。複数ファイル横断で時間がかかる作業は Subagent に委ねます。Day 1 Exercise 3 で 3 起こしを並列処理する体験はこの差を確認するためのものです。

Q6：CLAUDE.md と AGENTS.md の使い分けは？

CLAUDE.md は Claude Code 専用、AGENTS.md は GitHub Copilot / Cursor / Codex などの他エージェントとの共通ガイド。`@AGENTS.md` で CLAUDE.md から取り込む形が一般的です。本研修では AGENTS.md は配布時点で完成済み、CLAUDE.md だけを Day 1 で育てます。

Q7：トークン消費が予想より多い

ヒアリング起こしのフルテキストを毎ターン渡している可能性があります。Exercise 3 で導入する `/analyze-transcript` を経由すれば、1 ファイルあたり input 4,000〜7,000 / output 1,500〜2,500 tokens に収まります。`/cost` で都度確認してください。

Q8：トークン消費の月次予算管理は？

社内中継基盤経由なら基盤側の請求機能で月次上限を管理します。Pro / Max サブスクリプション利用ならアカウントの使用量モニタで把握します。Sonnet 常用、定期実行のみ Haiku、重要レビューのみ Opus、で月数千円〜数万円のレンジに収まります。

Q9：詰まったときに講師に何と言えばいいか

「何が」「どこで」「どう詰まっているか」を 1 行で言語化してください。

- 悪い例：「動かない」「うまくいかない」
- 良い例：「Exercise 3 の risk-extractor が JSON ではなく自然文を返してきていて、`/analyze-transcript` の DB 登録処理でパース失敗している」

SECTION 09 · TROUBLESHOOTING

9. 詰まったときの即応マニュアル

症状	確認	対処
`claude` が起動しない	`which claude` で PATH	ターミナル再起動。それでもダメなら講師に申告。WSL2 側で実行されているか確認（PowerShell や cmd.exe では動作しない）
`uvicorn` 起動失敗	`python -m app.seeds` 実行済みか	seeds 再投入。`backend/data/health.db` を削除して再作成
フロントエンド起動失敗	Node.js が手元にあるか	Claude Code に「frontend の開発サーバーを起動できる状態にして」と指示すれば、必要な環境構築から起動までを案内してもらえる
ポート競合（8000 / 5173）	`lsof -i :8000` / `:5173`	`--port 8001` 等で別ポート起動。`vite.config.ts` の proxy を合わせる
認証エラー	Pro / Max サブスクリプションのログイン状態、または `ANTHROPIC_BASE_URL` の値	サブスク利用なら `claude /logout` 後に再ログイン。中継基盤経由なら基盤側のログを確認
`/compact` で会話を短くしたい	長い会話で応答が遅い	`/compact` で会話履歴を要約。`/clear` で完全リセット。CLAUDE.md は自動再読込されるためプロジェクト知識は引き継がれる
Allow の確認が毎回出る	常用コマンドの権限	"Always allow" を選択するか、`.claude/settings.json` にホワイトリストを記載すると抑制できる
モデル選択を切替えたい	現在のモデル	`/model` で切替。Haiku は単純判定、Sonnet は常用、Opus は複雑な設計判断

> **詰まったら 5 分ルール**
> 5 分以上自力で抜けられなければ、上の表を確認 → それでもダメなら講師にエスカレ。受講者同士で相談するのも歓迎します。質問は「何が」「どこで」「どう詰まっているか」を 1 行で言語化して。

参考リンク

1. Claude Code 公式：claude.com/product/claude-code
2. Claude Code Docs：code.claude.com/docs
3. Claude Code Best Practices：code.claude.com/docs/en/best-practices
4. Sub-agents（公式 Docs）：code.claude.com/docs/en/sub-agents
5. Anthropic Cookbook：github.com/anthropics/anthropic-cookbook
6. Models Overview（Haiku 4.5 / Sonnet 4.6 / Opus 4.7）：platform.claude.com/docs/en/about-claude/models/overview

7. Claude Code 認証（Pro / Max サブスクリプション利用時のサインイン）：claude.ai/login

---