JBS 中級コース（案1）／ Day1 ／ 2026-05-14

顧客ヒアリングから
要件定義・設計を Claude Code で生成する

公開デモを実機で確かめ、顧客ヒアリングメモから要件定義書・ER図・API仕様・
シーケンス図を導く3時間。Day2の実装が走れる設計まで仕上げます。

13:00〜16:00（オンライン）　Claude Code 中心　個人Fork環境で実施

本日のアジェンダ

13:00〜13:10　Session 01 オリエンテーション
本日のゴールと3日間の俯瞰を共有。[10分]

13:10〜13:25　Session 02 環境セットアップと公開デモ確認
Claude Code 起動、配布物のCI緑、公開URL接続まで。[15分]

13:25〜13:45　Session 03 公開デモの観察
本物のシステムを動かして現状仕様と改善余地を掴む。[20分]

13:45〜14:15　Session 04 ヒアリングメモの読解と論点整理
本編＋補足を読み、要確認・矛盾・暗黙前提を抜き出す。[30分]

14:25〜15:00　Session 05 要件定義書の生成と精査
Claude Code で MoSCoW 整理。数値要件の取りこぼしを検知。[35分]

15:00〜15:30 **Session 06 データモデル設計とAPI仕様生成**

ER図（Mermaid）と OpenAPI 3.0 YAML を起こす。[30分]

15:30〜15:50 **Session 07 シーケンス図と設計レビュー**

3ユースケースをシーケンス化、自分の目で設計を点検。[20分]

15:50〜16:00 **まとめと Day2 への引き継ぎ**

持ち帰る3点と次回への宿題。[10分]

SESSION 01 ／ 13:00〜13:10

オリエンテーション

3日間で何を作り、Day1で何を出し切るかを共有します。

3日間の俯瞰

Day	テーマ	主成果物	翌日への引き継ぎ
Day1（本日）	要件定義・設計	要件定義書、ER図、API仕様、シーケンス図	設計が粗いと Day2で詰まる
Day2（5/21）	実装・テスト	動く FastAPI、ユニット＋結合テスト	テスト薄いと Day3の CI で落ちる
Day3（5/28）	Docker・CI/CD	docker-compose、Actions 全ジョブ緑	本研修の到達点

Day1の 3つのゴール

- [x] 公開デモを実機で操作し、現状仕様と仕込まれた改善余地を自分の言葉で言える
- [x] ヒアリングメモから要件定義書・ER図・API仕様・シーケンス図を Claude Code で生成し、自分の目で精査できる
- [x] Claude Code への指示を「短く区切って確認する」リズムを掴む

進め方の前提

分からなければ Zoom チャットへ

`[Q]` を頭に付けて投げてください。エラーログ全文の貼り付け歓迎です。詰まった方には休憩時間も含めて個別フォローします。

EMU 環境前提

JBS の EMU 組織配下にあるリポジトリで作業します。個人 GitHub への push は禁止です。clone 先のローカルフォルダ名は `0514-<EMUアカウント名>` です。ハンズオンガイド中の `0514-21-28_配布` という表記は読み替えてください。

Claude Code のコツ

長い指示を一度に投げるより、短く区切って確認する方が出力品質が安定します。Day1は特に「生成→読み返して数値要件が消えていないか確認」の往復が重要です。

講師自己紹介

Givery 株式会社、安田 光喜です。AI 駆動開発研修を担当しています。本日もよろしくお願いします。

SESSION 02 ／ 13:10〜13:25

環境セットアップと公開デモ確認

15分で Claude Code が起動し、配布物の CI が緑になり、公開デモにアクセスできる状態を作ります。

HANDS-ON 配布リポジトリと Claude Code の準備

15分

1 VSCode で clone 済みフォルダを開く

事務局共有の EMU 上 `0514-<EMUアカウント名>` リポジトリを VSCode で開いてください。
未完了の方は GitHub の Fork ボタン → リネーム → 個別clone を実行。

2 仮想環境と依存パッケージ

TERMINAL

```
python -m venv .venv
source .venv/bin/activate   # Windows: .venv\Scripts\Activate.ps1
pip install -r requirements-dev.txt
```

プロンプト先頭に `(.venv)` が付いたら有効化成功です。

3 動作確認4種

下記4つを順に実行し、すべて成功することを確認してください。Day1開始時点で CI が緑になる状態が用意されています。

TERMINAL

```
ruff check app/ tests/ api/        # Lint
ruff format --check app/ tests/ api/  # フォーマット
mypy app/                          # 型チェック
pytest tests/ -v                   # テスト1件PASS
```

4 Claude Code 起動

VSCode の内蔵ターミナルで `claude` を実行。サインイン未完了なら `claude /login` でブラウザ認証を済ませてください。

起動したら最初の動作確認として、次のように打ってみてください。

```
このリポジトリの構成を /docs と /app を中心に説明してください。
```
PROMPT

Claude Code が `CLAUDE.md` を読み、技術スタックとディレクトリ構成を返してくるはずです。

---

5 公開デモにアクセス

ブラウザで下記を開いてください。Day1全体で参照します。

- 管理 UI: https://jbs-0514.vercel.app/
- Swagger UI: https://jbs-0514.vercel.app/docs

1画面目に8件のチケット一覧が表示されれば接続成功です。

ここまでの完了チェック

- [x] pytest が1件 PASS
- [x] `claude` コマンドでチャット起動
- [x] 公開URL が開いてチケット一覧が見える

SESSION 03 ／ 13:25〜13:45

公開デモの観察

20分使って本物のシステムを触り、現状仕様と仕込まれた改善余地を自分の言葉で把握します。
Day2で何を作るかの解像度が上がります。

受講者の立場設定

2年目エンジニア、情報システム部門に異動初週

情報システム部長から「3日でプロトタイプを動かしてほしい」と依頼を受けたところです。
手元には顧客ヒアリングメモがあるだけ。まずは前任が雑に作った既存システムを動かしてみて、改善余地を含めた現状仕様を掴むのが本日の入口です。
詳細は `docs/project_overview.md` を読んでください。

TRY 公開デモを15分で触る

15分

1 チケット一覧で全体像を掴む

管理 UI の左ペインで状態フィルタを切り替え、8件のチケットがどう分布しているか観察。`SLA違反` バッジが付いているチケットを2件探してください。

2 チケット詳細を開いて状態遷移を試す

「VPN接続が突然できなくなった」をクリック。詳細ページのボタンで `→ in_progress` → `→ resolved` と順に変えてみてください。続いて `→ hoge` を押すと、無効な状態に変わってしまいます。本来は弾くべき遷移です。

3 コメントを試す

同じチケット詳細でコメントを追加。本文に `<script>alert('x')</script>` や、改行5行以上の長文、空文字を入れてみてください。すべて警告なしで通ります。

4 新規チケットを起票

左ペインのフォームでタイトル空欄＋本文空欄でも起票できることを確認。タイトルに1000文字以上を貼っても通ります。

5 **SLA違反一覧を見る**

/api/sla/violations を直接開いてください。実装が雑で、起票から1時間経つと自動で違反扱いになっています。本来は優先度別の SLA 時間を見るべきです。

---

6 **Swagger UI で API を直接叩く**

/docs から `GET /api/tickets/999` を実行。エラーレスポンスの形を覚えてください。続いて `PATCH /api/tickets/999` でも404を出して、形が微妙に違うことに気づくはずです。

> **触ってみた気づきを Zoom チャットへ**
>
> 「これ大丈夫か」と感じた挙動を3つ以上、チャットに投稿してください。Session04のヒアリングメモ読解と突き合わせます。

SESSION 04 ／ 13:45〜14:15

ヒアリングメモの読解と論点整理

30分使って `docs/hearing_memo.md` と `docs/hearing_memo_supplementary.md` を読み、論点を抜き出します。生成 AI に投げる前に、自分の頭で全体像を掴むのが目的です。

HANDS-ON ヒアリングメモから論点を抜く

30分

1 本編を5分で通読

`docs/hearing_memo.md` を上から読みます。発言者ごとに困りごとの粒度が違うことに注目。部長は経営目線、リーダーは運用目線、ユーザー代表は現場目線です。

2 補足を読んで未確認事項の解消を確認

`docs/hearing_memo_supplementary.md` で、本編に書かれていた「要確認事項」がどう解決されているかを確認してください。SLA 時間表、再オープン可否、添付ファイル制限、SLA 違反通知方法の4点です。

3 非機能要件を制約として把握

`docs/architecture_constraints.md` を3分で確認。ピーク50ユーザー、レスポンス1秒以内、データ保持3年、認証は Entra ID（研修ではスタブ）。これらは要件定義書の非機能要件節にそのまま反映します。

4 5分間の沈黙読書のあと、論点を整理

自分なりに次の表を埋めてみてください。Markdown でも紙でも OK。これが Session05で Claude Code に投げる前提情報になります。

- 絶対に外せない機能（Must）3つ
- あった方がよいが Phase 2でも可（Should/Could）3つ
- 非機能要件の数値（SLA、レスポンス、データ保持）
- 暗黙の前提や矛盾と感じた箇所

5 **公開デモの観察結果と突き合わせる**

Session03で見つけた「これ大丈夫か」と、ヒアリングメモから読み取った「あるべき姿」がどう食い違っているか、対比表を頭の中で作ってください。これが Day2の実装で直す内容そのものです。

例: ヒアリングでは「状態遷移は単方向」「SLA 時間は優先度別」と言っているが、公開デモは「任意の文字列が status に通る」「SLA は起票から1時間で違反扱い」になっている。

論点抜きの参考例（Session 05直前に開く）

休憩(14:15〜14:25)

SESSION 05 ／ 14:25〜15:00

要件定義書の生成と精査

35分使って Claude Code に要件定義書を起こさせ、自分の目で精査します。
「生成→読み返し→修正指示」を3ループ回すのが目安です。

HANDS-ON 要件定義書を MoSCoW で起こす

35分

1 出力先ファイルを準備

`docs/requirements.md` を空ファイルで作成。これから Claude Code がここを埋めていきます。

TERMINAL

```
touch docs/requirements.md
```

2 Claude Code に初稿を依頼

下記プロンプトをそのまま投げてください。`CLAUDE.md` の指示書を読んで、ヒアリングメモから要件定義書を起こします。

PROMPT

```
docs/hearing_memo.md と docs/hearing_memo_supplementary.md と docs/architecture_constraints.md を読み、
docs/requirements.md に要件定義書を書いてください。

構成は次の順:
1. 目的（背景・解決したい課題）
2. スコープ（含む・含まない）
3. ステークホルダーと役割
4. 機能要件（MoSCoW で Must / Should / Could / Won't）
5. 非機能要件（性能・可用性・データ保持・監査・認証）
6. 制約と前提
7. 要確認事項（メモにない暗黙前提があれば明示）

ヒアリングメモにない前提は勝手に追加せず「要確認」として明示してください。
```

3 生成された要件定義書を必ず自分の目で読む

Claude Code は数値要件を抜かしたり、ヒアリングにない要件を勝手に追加することがあります。下記を読み返してチェックしてください。

- SLA 時間表が優先度別に書かれているか（high 30分/4h、medium 4h/1営業日、low 1営業日/3営業日）
- ピーク50名・レスポンス1秒以内・可用性99% などの数値が消えていないか
- 「営業時間外は SLA 計算除外」のような前提条件が含まれているか
- 逆に、ヒアリングにない要件（Slack 連携、AI 自動分類など）が勝手に追加されていないか

---

4 抜けや誤りを Claude Code に直させる

気づいた点を箇条書きで指摘して修正させます。例:

PROMPT

```
docs/requirements.md を直してください:
1. 非機能要件節に「営業時間外と土日祝は SLA 計算から除外」を追記
2. 「自動割当ロジック」を Must から Should に降格（Phase 2）
3. 「Slack 連携」を削除（ヒアリングにない）
4. SLA の数値表を欠落のないように補完
```

---

5 受入基準目線で1ループ精査

各機能要件が「これだけでテストケースが書けるか」目線で読み直してください。書けない要件は曖昧な書き方になっています。Claude Code に「テストケースを書ける粒度に分解して」と頼んで詰めます。

よくある失敗

- Claude Code が出した要件定義書を読まずにそのまま採用する → Day2で実装が破綻
- 「Must を全部実装する」と無理して詰め込む → Day2で時間切れ。Should 以下は遠慮なく削る
- 「曖昧な要件」をそのまま残す → AI の解釈が毎回違って手戻り発生

SESSION 06 ／ 15:00〜15:30

データモデル設計と API 仕様の生成

30分でデータモデル（Mermaid ER図）と API 仕様（OpenAPI 3.0 YAML）を生成。Day2の実装はこの2つを正として進めます。

HANDS-ON ER 図と OpenAPI を起こす

30分

1 データモデルの生成

下記プロンプトで Mermaid の erDiagram を起こします。

PROMPT

```
docs/requirements.md をもとに docs/data_model.md にデータモデル設計を書いてください。

含めるもの:
1. Mermaid erDiagram でエンティティ間の関連を可視化
2. 各エンティティのカラム一覧（型・PK/FK・主要なインデックス候補）
3. 状態遷移ダイアグラム（チケットの状態遷移を Mermaid stateDiagram で）
4. データ量見積もり（チケット3年で何件、コメント何件など）

エンティティは User / Category / Ticket / Comment / Attachment / SLAPolicy / AuditLog を想定。
SLAPolicy は優先度ごとの初回応答時間・解決時間を保持する設計にしてください。
```

2 状態遷移を Mermaid stateDiagram で可視化

状態遷移は要件定義の核です。`open → in_progress → waiting_customer → resolved → closed` の単方向、`resolved → open` の再オープン許可、`closed` は最終状態という制約を図示。

Claude Code が描いた図が要件と一致するか必ず確認してください。

3 OpenAPI 3.0 YAML を起こす

```
PROMPT
docs/data_model.md と docs/requirements.md をもとに docs/openapi.yaml を書いてください。

OpenAPI 3.0で次のリソースを定義:
- /tickets (GET, POST, PATCH, DELETE)
- /tickets/{id}/comments (GET, POST)
- /tickets/{id}/attachments (GET, POST)
- /users (GET)
- /categories (GET)
- /sla/violations (GET)
- /auth/me (GET, JWT 検証スタブ)

エラーレスポンスは全エンドポイント共通で {"error": {"code": str, "message": str}} 形式。
エラーコード一覧も components/schemas に定義（例: TICKET_NOT_FOUND, INVALID_STATUS_TRANSITION, SLA_POLICY_NOT_FOUND）。
```

---

4 生成された OpenAPI を点検

下記の観点で読み返してください。

- パスパラメータの型（id は integer か string か）が一貫しているか
- レスポンススキーマがエンティティと過不足なく揃っているか
- 認可レベルが書かれているか（user は自分のチケットのみ、agent は全件 など）
- ページネーション・フィルタ・ソートの仕様が機能要件と整合しているか

---

5 Swagger Editor で YAML を検証

https://editor.swagger.io/ に貼って構文エラーがないか確認してください。Claude Code が出した OpenAPI は構文ミスを含むことがあります。

SESSION 07 ／ 15:30〜15:50

シーケンス図と設計レビュー

20分でシーケンス図3種を起こし、設計全体を Strict Reviewer モードで点検します。

HANDS-ON 主要3ユースケースのシーケンス化

15分

1 3ユースケースをシーケンス図に

PROMPT

```
docs/sequence_diagrams.md に Mermaid sequenceDiagram で3つのユースケースを書いてください:

1. ユーザーがチケットを起票（POST /tickets）→ DBに保存 → イベント通知 → 担当割当の流れ
2. エージェントが状態を resolved に変更（PATCH /tickets/{id}）→ 状態遷移ルール検証 → SLA計算 → 監査ログ書き込み
3. SLA違反検知 → 違反一覧返却（GET /sla/violations）→ 担当者通知（Phase 2 でメール送信）

各シーケンスに participants の凡例（User, API, Service, Repository, DB, Notifier）を含めてください。
```

2 Strict Reviewer モードで設計全体を点検

Claude Code のチャットモードを `strict-reviewer` に切り替えます。

PROMPT

```
/agents strict-reviewer

docs/ 配下の requirements.md、data_model.md、openapi.yaml、sequence_diagrams.md を読み、
設計全体に矛盾・抜け漏れ・曖昧さがないか厳密にレビューしてください。

特に下記4点を必ずチェック:
1. ヒアリングメモにある数値要件（SLA、レスポンス、データ保持）が要件定義書から実装可能な形で展開されているか
2. ER 図と OpenAPI のスキーマがズレていないか
3. 状態遷移ルールが OpenAPI のエラーコードと一致しているか
4. 認可（user / agent / manager の権限差）が OpenAPI に反映されているか
```

3 指摘を反映して Day1成果物を確定

指摘を全て反映する時間はないので、致命的なものだけ直してください。軽微なものは Day2冒頭で扱います。

確定したら git に commit。

```
TERMINAL
git add docs/
git commit -m "feat: Day1完了 要件定義・設計成果物"
git push origin main
```

push したら自分の PR ページか Actions タブで CI が緑になることを確認してください（docs/ 配下のみの変更なので lint や test は影響しないはず）。

設計のレビュー観点（自分でも回せるように）

- 「テストケースが書けるか」目線で要件を読む
- 「実装の手順が頭に浮かぶか」目線で ER とシーケンスを読む
- 「3ヶ月後の自分が理解できるか」目線でドキュメント全体を読む

CLOSING ／ 15:50〜16:00

まとめと Day2への引き継ぎ

Day1で確定した成果物と、Day2までに各自で詰めておく宿題を整理します。

Day1の成果物チェック

- [x] `docs/requirements.md` 要件定義書（MoSCoW、非機能要件、要確認事項込み）
- [x] `docs/data_model.md` ER 図と状態遷移図
- [x] `docs/openapi.yaml` API 仕様（エラーコード一覧含む）
- [x] `docs/sequence_diagrams.md` 主要3ユースケースのシーケンス図
- [x] git commit & push 済み、CI が緑

Day2までの宿題

- Day1成果物を一度紙またはタブレットで読み返す（画面では見落としやすい）
- Strict Reviewer が指摘した「致命的でないもの」を1〜2件直しておく
- 気になる箇所は Issue として GitHub に起票（Day2で実装時の TODO リストになる）

本日覚えておきたい3点

- [x] Claude Code の生成物は「読み返して数値要件が消えていないかを確認」が必須
- [x] 要件定義書は「テストケースが書ける粒度」で書く。曖昧さを残すと Day2で破綻する
- [x] Strict Reviewer のような厳しいレビュー視点を切り替えて使えるようにしておく

> **Day2のフォーカス予告**
>
> Day2（5/21）は本日の設計を元に FastAPI + SQLAlchemy 2.0で実装します。テストカバレッジ80%を目標にし、CI 全ジョブを緑のまま保ちながら開発を進めます。事前に `.github/chatmodes/strict-reviewer.chatmode.md` と `.github/instructions/sqlalchemy.instructions.md` に目を通しておくとスムーズです。

制作 Givery／JBS 中級コース 案1 Day1配布用／2026-05-14