AGENTS.md

- In Markdown, image files must be placed in `images/` located in the same directory as the Markdown file.
- File names in `docs/` must use kebab-case (except for `AGENTS.md` and actual API endpoint names under `specs/api/`).
- `<root>/apps/[<group>/]<component>` refers to a "software component" (including applications, libraries, frameworks, data stores, and tools). Use "software component" when distinction is needed (e.g. from React UI components). "Component" may be used when the context clearly implies a software component.
- Documents in `docs/` (except for `AGENTS.md`) must include frontmatter with `name` (filename), `description` (1-line summary), `timestamp` (date in YYYY-MM-DD format), and `ai` (authorship level: `none` [human only], `ai-assisted` [human written, AI proofread], `ai-coauthored` [human designed, AI written], or `ai-generated` [AI created]).
- References (links) to `<workspace>` (`.local/workspace/`) are strictly forbidden in `docs/`.
- Files under `wiki/` (wiki files) must NOT be referenced (linked) from files outside `wiki/` (non-wiki files). References (links) from a wiki file to another wiki file are allowed.
- Files under `wiki/` are "AI documents" created by AI for AI, compiled from `docs/raw/`, source code, internal AI knowledge, or external resources. Non-wiki files must directly reference or use primary source data (`docs/raw/` or source code), rather than relying on wiki files.
- Diagrams must be created using Mermaid unless otherwise specified.
- For directory layout and available guides, refer to directory-structure.md.
- For the overall development process and guide map, refer to document-development-guide.md.

AI Agent Self-Checklist

AI エージェントがドキュメントや設計書を生成・更新する際は、以下のチェックリストを遵守してください。

- [ ] **用語定義**: コンポーネント識別子は `AppID`、タスクIDは `TaskID`、仕様書IDは `SpecID`、ユースケースIDは `USECASE-XXX` を使用しているか？
- [ ] **見出しパス記法**: 見出しに含まれるファイルパス・ディレクトリ名はインラインコード（`...`）で囲まれているか？
- [ ] **DR インライン相互リンク**: 意思決定記録（`DR-XXX.md`）へのリンクは、意思決定の関連記述箇所の直後に `([DR-001](...))` のインライン形式で記述し、`architecture.md`, `design.md`, `SPEC-YYY.md` 等の「関連」セクションにも相互リンクを記載しているか？
- [ ] **ネストコードブロック**: Markdown 内の Markdown コードブロック外側フェンスは内側より長く（4個 ```` 以上）記述されているか？

AGENTS.md

- 本ディレクトリ（`docs/guide/`）には、aidev-template の標準開発プロセス、ガイドライン、および各種ドキュメントテンプレート（`XXX-guide.md`, `XXX-template.md`）を配置します。
- ガイドラインおよびテンプレートの作成・編集にあたっては、フロントマター（`name`, `description`, `timestamp`, `ai`）を必ず記述してください。

ガイドライン・テンプレート インデックス

1. 全体プロセス & 設計規約

- document-development-guide.md ドキュメント駆動開発（DDD）および仕様駆動開発（SDD）の基本原則、設計・開発フェーズの全体プロセス、反復開発、縦切り実装スライス規約。
- document-status-guide.md ドキュメントステータス定義、ライフサイクル遷移、管理場所、フロントマター項目、レビュー連携ルール。
- document-design-guide.md 設計ドキュメント（全体設計および各ソフトウェアコンポーネント設計）の作成基準と記述フォーマット。
- document-implementation-guide.md 開発・実装ドキュメント（`backlog/`）の作成規約および運用ルール。
- directory-structure.md プロジェクトの標準ディレクトリ構成および各ディレクトリ・ファイルの役割と配置規則。
- commit-message.md コミットメッセージのフォーマット、ヘッダー部構文、prefix分類、参照規約。
- aidev-template-guide.md aidev-template の基本コンセプト、環境セットアップ、および提案書によるフィードバックワークフロー。
- mise-guide.md miseタスクの運用方針、タスク設定ファイル（永続・一時・テンプレート）の役割分担、およびmiseタスクの定義・実行ガイドライン。
- testing-guide.md テストの分類（単体・コンポーネント[統合]・パフォーマンス・セキュリティ・汎用検証）、配置規則、および検証チェックリスト（チェックリスト・テスト仕様書）の作成ガイドラインと記述テンプレート。
- local-guide.md AIエージェントの作業領域・記憶領域（.local/）の役割、内部構造（workspace, memory, worklog等）、および連携スキルの運用ガイドライン。

2. 設計ドキュメントテンプレート

- concept-guide.md コンセプト文書（Concept）作成・編集時に参照するテンプレート。
- architecture-guide.md 全体アーキテクチャ設計書（Architecture）作成・編集時に参照するテンプレート。

- design-guide.md コンポーネント詳細設計書（Design）作成・編集時に参照するテンプレート。
- specification-guide.md 機能仕様書（Specification / SPEC-XXX）作成・編集時に参照するテンプレート。
- usecase-guide.md ユースケース定義書（USECASE-XXX）作成・編集時に参照するテンプレート。
- decision-record-guide.md 意思決定記録（Decision Record / DR-XXX）作成・編集時に参照するテンプレート。
- tools-guide.md 開発ツールガイド（docs/tools.md）作成・編集時に参照するガイドラインおよびテンプレート。

3. 計画 & スライス

- backlog-slice-guide.md 実装スライス（Slice）作成・編集時に参照するテンプレート。
- backlog-implementation-guide.md 実装計画書（Implementation Plan）作成・編集時に参照するテンプレート。
- backlog-usecase-guide.md ユースケース進捗台帳（docs/backlog/usecase.md）作成・運用時に参照するテンプレート。

4. 実装タスク & 課題・管理ドキュメント

- backlog-task-guide.md 実装タスク詳細（Task）作成・編集時に参照するテンプレート。
- backlog-issue-guide.md 課題・不具合管理書（Issue）作成・編集時に参照するテンプレート。
- backlog-todo-guide.md TODO管理書（TODO）作成・編集時に参照するテンプレート。

AGENTS.md

- 本ディレクトリ（`docs/backlog/`）には、プロジェクト全体のバックログ、実装スライス（`slices/`）、実装計画（`implementations/`）、ユースケース進捗台帳（`usecase.md`）、フェーズ台帳（`phase.md`）等の管理ドキュメントを配置します。
- フェーズ別進捗ビューとして `phase.md` を管理します（詳細は backlog-phase-guide.md 参照）。
- ユースケース別進捗ビューとして `usecase.md` を管理します（詳細は backlog-usecase-guide.md 参照）。
- ドキュメントの作成・編集にあたっては、document-implementation-guide.md および本ディレクトリ配下に対応する各ガイド（backlog-slice-guide.md, backlog-implementation-guide.md 等）を参照し、定義されたセクション構成・フロントマター・記述ルールに従ってください。
- 同期スクリプトにより管理されるため、台帳ファイル（`usecase.md`, `phase.md`, `wbs.md` 等）の手動更新は不要です。

AGENTS.md

- 本ディレクトリ（`docs/decision-records/`）には、システムアーキテクチャや技術選定、設計方針に関する重要な意思決定記録（`DR-XXX.md`）を配置します。
- 意思決定記録の作成・編集にあたっては、decision-record-guide.md を参照し、定義されたセクション構成・フロントマター・記述ルールに従ってください。
- 同期スクリプトにより管理されるため、総合台帳（`index.md`）への手動更新は不要です。

AGENTS.md

- 本ディレクトリ（`docs/usecases/`）には、複数コンポーネントにまたがる機能群やユーザーシナリオを束ねるユースケース定義書（`USECASE-XXX.md`）を配置します。
- ユースケース定義書の作成・編集にあたっては、usecase-guide.md を参照し、定義されたセクション構成・フロントマター・記述ルールに従ってください。

Document Development Guide

本ガイドは、`aidev-template` における開発アプローチである **ドキュメント駆動開発（Document Driven Development: DDD）**、その中核となる **仕様駆動開発（Spec Driven Development: SDD）**、および **インクリメンタル開発（Incremental Development）** の原則、設計・開発フェーズの全体プロセス、縦切り開発（slice）と実装スライスの運用規約を定義するものです。

1. 開発基本方針（DDD × SDD × インクリメンタル開発）

`aidev-template` は、ドキュメント駆動開発（DDD）を基盤とし、仕様駆動開発（SDD）による「仕様ファースト」、およびインクリメンタル開発による「増分仕様主導の実装」を融合した3層構成を採用しています。

- **ドキュメント駆動開発 (DDD: Document Driven Development)**:
    - コード先行ではなく、要件・アーキテクチャ・設計・計画をドキュメントとして可視化し、ドキュメントをプロジェクトの正本（Single Source of Truth）として開発を推進します。
- **仕様駆動開発 (SDD: Spec Driven Development)**:
    - **仕様ファーストの徹底**: コードを書き始める前に、必ず対象機能の入出力、画面表示ロジック、ViewModel/Service仕様、受入条件などの「仕様」を明確に定義します。
    - **ユースケース・feature 主導**: ユーザー目的や価値シナリオをユースケース（`usecase`）として定義し、コンポーネントを縦切りした動作可能な最小機能単位（`feature`）ごとに仕様を導出します。
    - **詳細設計への取り込み**: 定義された仕様をコンポーネント詳細設計（`design.md`）に取り込み、実装の構造的基盤とします。
- **インクリメンタル開発 (Incremental Development)**:
    - **常に増分仕様で実装を推進**: 実装（コード作成・修正）を行う際は、**常に増分機能仕様書**（Incremental Specification: **`specs/incremental/<SPEC-I-ID>.md`**）**を起点**とします。新機能開発はもちろん、既存機能の拡張、仕様改修、バグ修正（ISSUE対応）であっても、必ず自己完結した増分仕様を作成（または既存仕様を直接更新）して実装に臨みます。
    - **1:1:1 の直結構造**: 増分仕様（`<SPEC-I-ID>`）、実装計画（`<component>/backlog/implementations/<SPEC-I-ID>.md`）、タスク詳細（`<component>/backlog/tasks/<SPEC-I-ID>.md`）を 1:1:1 で対応付け、小さく安全な反復サイクルで実装・検証を完結させます。

2. 全体開発プロセス (Overall Development Process)

開発プロセスは、上位の構造や方針を定義する「設計フェーズ」と、増分仕様を起点として反復実装を行う「インクリメンタル実装フェーズ」の2相で構成されます。これらを交互に繰り返しながらアジャイルに推進します。

設計フェーズ (Design Phase)

1. **最上位設計の策定**:
    - 最初に `concept.md`（概要・目的）および `architecture.md`（全体構成・技術選定・コンポーネント定義）を作成し、これらに基づいて独立した計画文書である `plan.md`（全体計画: マイルストーン・フェーズ）を策定します。
2. **ユースケース・feature定義**:
    - 全体設計に基づき、ユーザー価値やシナリオを定義するユースケース（`docs/usecases/USECASE-XXX.md`）および動作する最小機能単位（`feature`）を策定します。
3. **機能仕様・共通仕様・詳細設計の作成**:
    - ユースケースおよび全体設計に基づき、各ソフトウェアコンポーネントの機能仕様書（`specs/[<category>/]<SPEC-ID>.md`）、共通仕様書（`SPEC-C-<name>.md`）、および詳細設計書（`design.md`）を作成します。

インクリメンタル実装フェーズ (Incremental Implementation Phase)

1. **増分仕様の策定（仕様ファーストの徹底）**:
    - 実装（コード変更）を行う前に、必ず `specs/incremental/` 配下に自己完結した増分機能仕様書（`specs/incremental/<SPEC-I-ID>.md`）を作成します。
    - 対象の機能仕様書（`<SPEC-ID>.md`）に増分仕様をマウントし、仕様台帳（`specs/index.md`, `specs/index.link.tsv`）を同期します。
2. **実装計画・タスク詳細の策定 (1:1:1 原則)**:
    - **縦切り開発 (slice)**: 複数コンポーネントにまたがる feature の場合は、実装スライス（`docs/backlog/slices/<slice-name>.md`）を作成し、各コンポーネントの依存関係と実装順序を整理します。
    - **コンポーネント実装計画とタスク詳細**: 増分仕様（`<SPEC-I-ID>`）に対して、1つの実装計画（`<component>/backlog/implementations/<SPEC-I-ID>.md`）と1つのタスク詳細ファイル（`<component>/backlog/tasks/<SPEC-I-ID>.md`）を作成します（詳細は backlog-implementation-guide.md および backlog-task-guide.md を参照）。
3. **実装の推進と完了評価の連鎖**:
    - **実装タスクの遂行**: タスク詳細に沿って順次コード実装と自動テストを行い、実装計画の完了条件を評価・検証します。
    - **実装スライス（slice）の完了**: 関連コンポーネントの実装計画がすべて完了した時点で、feature 全体の完了条件を総合検証します。

- **ユースケースの実現確認**: 実装スライスの完了を受け、親ユースケースの受入条件・シナリオが充足されたかを最終評価します。

インクリメンタル開発のケース別フロー (Case-by-Case Development Flow)

インクリメンタル開発における作業着手時は、変更の性質および規模に応じて以下の3つのケースに分類して運用します。

1. **ケース 1: 新規機能の開発** (New Feature):
   - 新規に増分機能仕様書 (`specs/incremental/<SPEC-I-ID>.md`) を作成します。
   - 対応するコンポーネント実装計画 (`<component>/backlog/implementations/<SPEC-I-ID>.md`) および実装タスク詳細ファイル (`<component>/backlog/tasks/<SPEC-I-ID>.md`) を作成 (1:1:1 原則) し、通常フローで実装・テストを遂行します。
2. **ケース 2: 契約に変更のある仕様変更** (Breaking / Contract Spec Change):
   - API入出力、外部インターフェース、データモデル構造、公開振る舞い契約等に変更が及ぶ仕様変更の場合、**新規に増分機能仕様書 (`specs/incremental/<SPEC-I-ID>.md`) を作成** します。
   - 親機能仕様書 (`<SPEC-ID>.md`) にマウントし、ケース1と同様に実装計画および実装タスク詳細ファイルを新規作成して実装・検証を遂行します。
3. **ケース 3: 契約に変更のない仕様変更・不具合修正・脆弱性修正・リファクタリング・パフォーマンス改善** (Non-Contract Change / Bugfix / Refactor / Performance):
   - 外部契約に変更のない仕様修正 (誤字・表現の軽微な修正等)、不具合修正 (バグ修正)、脆弱性修正、内部リファクタリング、およびパフォーマンス改善の場合は、**既存の増分仕様書を直接更新** します。
   - その上で、コード変更対象の規模に応じて以下のいずれかで進めます:
     - **超軽量変更 (1ファイルかつ10行以下)**:
       - **実装計画は作成せず** に直接作業を実施します。
       - 作業完了後、対象の既存実装タスク詳細ファイル (`<component>/backlog/tasks/<SPEC-I-ID>.md`) の「実装メモ」セクションに作業内容・修正理由を記録します。
     - **通常変更 (複数ファイル、または11行以上)**:
       - 既存の実装タスク詳細ファイル (`<component>/backlog/tasks/<SPEC-I-ID>.md`) を `WIP` **ステータス** に戻します。
       - 変更内容・追加タスクを実装タスク詳細ファイルに反映して実装・検証を遂行します。
       - **上位ステータスの非変更ルール**: この際、**実装計画 (Implementation Plan) や実装スライス (Slice) など上位ドキュメントのステータスは変更しません。**

反復開発と仕様・設計の継続的更新 (Iterative Development & Continuous Updates)

- **スピード優先アプローチ**: 本プロセスでは開発速度を優先し、先行して最小限の増分仕様で実装を行い、後から仕様を追加・拡充していくアプローチをサポートします。
- **継続的同期原則**: 実装中の気づきや変更は、即座に増分仕様、機能仕様、および詳細設計書 (`design.md`) にフィードバックし、常にドキュメントとコードの完全な整合性を維持します。

3. ユースケースと縦切り開発 (Usecase & Slice Development)

複数のソフトウェアコンポーネントが連携する縦切り開発 (slice 単位での実装) を行います。

1. **ユースケース (`docs/usecases/USECASE-XXX.md`)**:
    - ユーザー目的やシナリオを定義し、配下の動作する機能単位 (`F-<no>: <feature-name>`) や実現条件 (論理式等)、仕様依存関係 (`<AppID>:<SpecID>`)、シーケンス図等を記録する永続文書です。
2. **実装単位 (`feature`) と slice**:
    - 動作する機能単位 (`F-<no>: <feature-name>`) ごとに縦切り実装の計画・進捗情報として実装スライス (`docs/backlog/slices/<slice-name>.md`) を作成します (**追跡対象**)。
    - 1つの `feature` (slice) の推奨粒度は「関連する各ソフトウェアコンポーネントの仕様 (SPEC) が1つずつ程度」とします。feature を分割して管理する場合は `track` を使用します。
    - feature の進捗・実装状況は `docs/backlog/usecase.md` で管理します。

4. 実装スライス (Implementation Slice)

実際の開発において複数のソフトウェアコンポーネントが相互に関連・連携して機能開発を進める場合は、各コンポーネントの機能仕様や実装計画、依存関係を整理した実装スライス (`docs/backlog/slices/<slice-name>.md`) を作成・運用します (**追跡対象**)。基本として追跡対象から非追跡対象へのリンクは禁止されていますが、実装スライスから実装計画 (非追跡対象) へのリンクは特別に許可されます (実装計画が完了した際は、リンクを `-` に更新します)。

具体的な配置場所、必須セクション構成、ステータス定義、およびMarkdownテンプレートについては、backlog-slice-guide.md を参照してください。

5. 設計ドキュメント体系(必須・任意)

設計ドキュメントは、プロジェクトおよびコンポーネント開発において必須となるコア設計ドキュメントと、必要に応じて導入する任意ドキュメントに分類されます (※ 詳細な作成基準・記述順序は document-design-guide.md を参照)。また、全体計画を管理する `plan.md` は設計ドキュメント群とは独立した計画文書として位置づけられます。

必須設計ドキュメント (Required Design Documents)

開発において必ず作成・維持する主要な 5 つの設計ドキュメントです。

1. `concept.md` **(コンセプト)**: 「何を作るのか（What）」と「なぜ作るのか（Why）」、開発背景やゴールを定義（concept-guide.md 参照）。
2. `architecture.md` **(基本設計・アーキテクチャ)**: システム全体の構成、ディレクトリ構造、コンポーネント役割、技術スタック、依存関係を定義（architecture-guide.md 参照）。
3. `usecases/` **(ユースケース・feature定義)**: 複数コンポーネントを横断するユーザー価値・目的シナリオ、動作する最小機能単位（`feature`）、実現条件（論理式等）、仕様依存関係を定義（※ 複数コンポーネント横断開発時は必須、単一コンポーネント完結時は任意・不要。usecase-guide.md 参照）。
4. `specs/` **(機能仕様書・増分仕様書・共通仕様書)**: 各機能ごとの機能仕様書（`<SPEC-ID>.md`）、実装の起点となる増分機能仕様書（`incremental/<SPEC-I-ID>.md`）、および共通仕様書（`SPEC-C-<name>.md`）を定義（specification-guide.md 参照）。
5. `design.md` **(詳細設計書)**: コンポーネント構造、モジュール設計、内部処理ロジック、全仕様書（`specs/`）の L1/L2マッピングを定義（design-guide.md 参照）。

計画ドキュメント (Planning Document)

- `plan.md` **(全体計画)**: マイルストーン（重要目標時点）と開発フェーズ（戦略的大日程）を定義する独立した文書です（plan-guide.md 参照）。`concept.md` および `architecture.md` を入力として作成されます。

任意設計ドキュメント (Optional Design Documents)

プロジェクトの規模や要件、データベースの有無に応じて追加・導入するドキュメントです。

1. `requirements.md` **(要件定義書)**: 機能一覧および画面一覧の整理。
2. `security.md` **(セキュリティ定義・ポリシー)**: セキュリティ基本方針、認証・認可、データ保護、脆弱性対策、シークレット管理。
3. `schema.md` **(データベース設計方針)**: データベース概要、命名規則、全体ER図、マイグレーション方針。
4. `models/` **(物理モデル定義)**: テーブルモデル概要、物理/論理カラム定義、インデックス、アソシエーション。

6. 文書レイヤーとリンク制限規則 (Document Layers & Link Rules)

ドキュメント間の疎結合性と抽象度の一貫性を保つため、`docs/` 配下の主要ドキュメントを 5 つのレイヤー（L1〜L5）で管理し、リンク可能範囲を自レイヤーおよび上下1層（±1層: $L_{n-1} \sim L_{n+1}$）までに制限します。

レイヤー	階層名・責務	対象ドキュメント・配置場所	リンク可能範囲
L1	方針・全体設計・全体計画	`concept.md`, `architecture.md`, `plan.md`, `security.md`	L1, L2
L2	要求・詳細設計・大分類計画	`design.md`, `usecases/` (`USECASE-XXX.md`), `backlog/phase.md`, `requirements.md`	L1, L2, L3
L3	仕様・進捗台帳・縦切りスライス	`<SPEC-ID>.md`, `backlog/usecase.md`, `backlog/slices/`	L2, L3, L4
L4	増分仕様・共通仕様・実装計画・課題・TODO・台帳	`specs/incremental/<SPEC-I-ID>.md`, `SPEC-C-*.md`, `backlog/implementations/`, `backlog/spec.md`, `backlog/issues/`, `backlog/todos/`, `backlog/wbs.md`	L3, L4, L5
L5	実装作業・タスク詳細	`backlog/task/` (`backlog/tasks/<SPEC-I-ID>.md`)	L4, L5

- **上下1層（±1層）制限**: 所属レイヤーの同一階層および上下1層（$L_{n-1} \sim L_{n+1}$）以外のレイヤーへのリンクは不適合（禁止）とします。
- **対象の限定**: レイヤーに定義された文書間のみを対象とし、ガイドライン（`docs/guide/`）、ツール定義（`docs/tools.md`）、意思決定記録（`docs/decision-records/`）などは本制約の対象外（自由参照可）とします。

7. ドキュメント関係図

必須ドキュメントを軸として、任意ドキュメントがどのように補完・連携されるかの構成図です。

> **Note**
>
> **線の意味と配色**:
>
> - 🟧 **実線（橙色）**: 必須 INPUTS
> - 🟧 **破線（橙色）**: 任意 INPUTS
> - 🟩 **実線（ライム色）**: 成果物
> - 🟩 **破線（ライム色）**: フィードバック
> - 🟦 **実線（シアン色）**: 参照（ビューから対象への参照）
> - 🟦 **破線（シアン色）**: 関連

```
flowchart TD
    subgraph CoreRequired ["必須設計ドキュメント (Required)"]
        CONCEPT["concept.md"] --> ARCHITECTURE["architecture.md"]
        ARCHITECTURE --> USECASES["usecases/"]
        USECASES --> SPECS["specs/<br/>(機能仕様: SPEC-XXX.md)"]
        ARCHITECTURE --> DESIGN["design.md"]
        SPECS --> DESIGN
    end

    PLAN_DOC["plan.md"]
    CONCEPT --> PLAN_DOC
    ARCHITECTURE --> PLAN_DOC

    subgraph OptionalDocs ["任意設計ドキュメント (Optional)"]
        REQ["requirements.md"] -.-> ARCHITECTURE
        SEC["security.md"] -.-> ARCHITECTURE
        SCHEMA["schema.md"] -.-> MODELS["models/"]
        MODELS -.-> DESIGN
    end

    subgraph IncrementalDev ["インクリメンタル開発・実装実行"]
        SPECS --> INCR_SPECS["specs/incremental/<br/>(増分仕様: SPEC-I-XXX.md)"]
        USECASES --> SLICE["docs/backlog/slices/"]
        SLICE -->|"構成"| IMPL_PLAN["backlog/implementations/<br/>(<SPEC-I-ID>.md)"]
        INCR_SPECS -->|"1:1:1"| IMPL_PLAN
        DESIGN --> IMPL_PLAN
        IMPL_PLAN -->|"1:1:1"| TASK["backlog/tasks/<br/>(<SPEC-I-ID>.md)"]
        TASK --> IMPL["実装・自動テスト実行"]
        ISSUES["backlog/issues/"]
        TODOS["backlog/todos/"]
    end

    subgraph BacklogViews ["バックログ・進捗ビュー (参照のみ)"]
        SPEC_LOG["backlog/spec.md"]
        USECASE_LOG["docs/backlog/usecase.md"]
        WBS["backlog/wbs.md"]
        PHASE_LOG["docs/backlog/phase.md"]
    end

    IMPL -.->|"フィードバック・仕様更新"| INCR_SPECS
    INCR_SPECS -.->|"マウント・反映"| SPECS
    IMPL -.->|"設計修正"| DESIGN
```

```
SPEC_LOG --> SPECS
USECASE_LOG --> USECASES
WBS --> TASK
WBS --> ISSUES
WBS --> TODOS
PHASE_LOG --> WBS
PHASE_LOG --> SPEC_LOG
PHASE_LOG --> USECASE_LOG
PHASE_LOG -->|"フェーズ定義参照"| PLAN_DOC

linkStyle 0,1,2,3,4,5,6,11,12,14,15,17 stroke:#f97316,stroke-width:2px;
linkStyle 7,8,9,10 stroke:#f97316,stroke-width:2px,stroke-dasharray:5 5;
linkStyle 13,16 stroke:#84cc16,stroke-width:2px;
linkStyle 18,19,20 stroke:#84cc16,stroke-width:2px,stroke-dasharray:5 5;
linkStyle 21,22,23,24,25,26,27,28,29 stroke:#06b6d4,stroke-width:2px;
```

ドキュメント別 入力・出力・成果物・参照一覧

ドキュメント / 要素	INPUTS	OUTPUTS	成果物
`concept.md`	-	`plan.md`、`architecture.md`	-
`plan.md`	`concept.md`、`architecture.md`	-	-
`architecture.md`	`concept.md`、`requirements.md`、`security.md`	`usecases/`、`design.md`、`plan.md`	-
`usecases/`	`architecture.md`	`specs/`、`docs/backlog/slices/`	-
`specs/` (機能仕様)	`usecases/`	`design.md`、`specs/incremental/`	-
`specs/incremental/` (増分仕様)	`specs/`	`backlog/implementations/`	-
`design.md`	`architecture.md`、`specs/`、`models/`	`backlog/implementations/`	-
`requirements.md`	-	`architecture.md`	-
`security.md`	-	`architecture.md`	-
`schema.md`	-	`models/`	-

ドキュメント / 要素	INPUTS	OUTPUTS	成果物
`models/`	`schema.md`	`design.md`	-
`docs/backlog/slices/`	`usecases/`	-	`backlog/implementations/`
`backlog/implementations/`	`specs/incremental/`、`design.md`	-	`backlog/tasks/`
`backlog/tasks/`	-	`実装・自動テスト実行`	-
`backlog/issues/`	-	-	-
`backlog/todos/`	-	-	-
`実装・自動テスト実行`	`backlog/tasks/`	-	`specs/incremental/`、`design.md`
`backlog/wbs.md`	-	-	-
`backlog/spec.md`	-	-	-
`docs/backlog/usecase.md`	-	-	-
`docs/backlog/phase.md`	-	-	-

8. ステータス管理・レビュー連携規約への委譲

各ドキュメント（ユースケース、機能仕様書、実装スライス、実装計画、タスク詳細、課題、TODO、意思決定記録等）のステータス定義、正本管理場所、ライフサイクル遷移、フロントマター規則、および `review-document-status` スキルとの連携ルールについては、すべて document-status-guide.md に定義されています。ステータス管理およびレビュー運用時は同ガイドを参照してください。

Document Design Guide

本ガイドは、プロジェクト全体および各ソフトウェアコンポーネントにおける設計ドキュメントの分類（必須・任意）、記述順序、および標準記述仕様を定義するものです。

---

1. 設計ドキュメントの分類と優先順序

設計ドキュメントは、以下の通り **必須ドキュメント (Required)** を前方、**任意ドキュメント (Optional)** を後方に配置して構成・整理します。

```
設計ドキュメント
├── 必須（Required）
│   ├── concept.md（コンセプト・概要）
│   ├── architecture.md（基本設計・システム構成）
│   ├── usecases/（ユースケース・feature定義）
│   ├── specs/（機能仕様書）
│   └── design.md（詳細設計書）
└── 任意（Optional）
    ├── requirements.md（要件定義書）
    ├── security.md（セキュリティ定義・ポリシー）
    ├── schema.md（DB設計方針・全体ER図）
    └── models/（物理モデル定義）
```

---

2. 必須設計ドキュメント (Required)

2.1 concept.md (コンセプト)

プロジェクト全体（`<root>/docs/concept.md`）および個別のソフトウェアコンポーネント（`<root>/docs/[<group>/]<component>/concept.md`）において、「何を作るのか（What）」と「なぜ作るのか（Why）」を明確化する最上位の必須ドキュメントです。（※ 詳細は concept-guide.md を参照）

- **概要**: What（何を作るのか）、機能全体像、主要コンポーネントを記述します。
- **目的**: Why（なぜ作るのか）、開発背景、解決すべき課題、ビジネス/技術的価値を明記します。

2.2 architecture.md (基本設計・システム構成)

システム全体の基本設計、ディレクトリ構造、システム構成、ソフトウェアコンポーネントの役割と技術スタック、依存関係およびデータフローを定義する必須ドキュメントです。（※ 詳細は architecture-guide.md を参照）

1. **ディレクトリ構造**: プロジェクト配置構造や主要ディレクトリの役割。
2. **システムアーキテクチャ**: 全体概要および構成図（Mermaid等）。
3. **ソフトウェアコンポーネント構成**: 各コンポーネント（アプリ、ライブラリ、ツール等）の役割と技術スタック。
4. **依存・データフロー**: コンポーネント間の依存関係・通信プロトコル・データフロー。
5. **関連**: 関連設計書へのリンク・参照。

2.3 usecases/（ユースケース・feature定義）

複数コンポーネントにまたがる機能群やシナリオを束ね、ユーザー価値・目的および最小機能単位（`feature`）を定義・追跡するディレクトリです（複数コンポーネント横断開発時は必須、単一コンポーネント完結時は不要・任意。※ 詳細は usecase-guide.md を参照）。

- **配置場所**: `<root>/docs/usecases/USECASE-XXX.md`
- **構成要素**: 概要、実現条件（`AND`/`OR`/`XOR` 論理式等）、feature 詳細（`F-<no>: <feature-name>`、関連仕様書 `<AppID>:<SpecID>`、完了条件、種類、シーケンス図）。
- **台帳管理**: feature 実装ステータス台帳（`docs/backlog/usecase.md`）と連携して管理。

2.4 specs/（機能仕様書）

各ソフトウェアコンポーネント配下に機能仕様書・入出力仕様書を格納する必須ディレクトリです。（※ 詳細は specification-guide.md を参照）

- **配置場所**: `<root>/docs/[<group>/]<component>/specs/` 配下
- **ステータス管理**: 機能仕様書自体のフロントマターには進捗ステータスを持たせず、各コンポーネントの台帳 `backlog/spec.md` で一括管理。

2.5 design.md（詳細設計書）

コンポーネント構造、モジュール設計、内部処理ロジック等を定義する詳細設計書です。（※ 詳細は design-guide.md を参照）

- **必須構成**:「設計方針」「ビルド・成果物」「レイヤー構造（L1/L2表）」「関係図（L1+L2 Mermaid図）」「機能仕様書一覧」「関連」。
- **仕様書マッピング**: `specs/` 配下の **すべての機能仕様書（`SPEC-XXX.md`）を L1/L2 分類して一覧表に掲載することを必須** とします。

---

3. 任意設計ドキュメント（Optional）

プロジェクトの規模や技術要求に応じて作成・追加する任意ドキュメントと構成項目です。

3.1 requirements.md (要件定義書)

各ソフトウェアコンポーネントにおける機能要件・画面要件を整理・明確化するドキュメントです。

- **構成項目**:
    - **機能一覧**: システムが提供する機能の洗い出し（機能ID、機能名、機能概要）。
    - **画面一覧**: アプリケーションの画面構成一覧（画面ID、画面名、役割・画面遷移概要）。

3.2 security.md (セキュリティ定義・ポリシー)

プロジェクト全体のセキュリティ方針、リスク対策、およびセキュリティ設計ガイドラインをまとめたドキュメントです。

- **構成項目**:
    - **セキュリティポリシー**: プロジェクト全体のセキュリティ基本方針。
    - **認証・認可**: 認証方式、ユーザー権限・アクセス制御方針。
    - **データの保護**: 通信・保管時の暗号化方式、個人情報・機密情報の取り扱い定義。
    - **脆弱性対策**: 入力値検証、各種攻撃（XSS, CSRF, SQLi等）への対策。
    - **シークレット管理**: APIキー、環境変数、パスワード等の安全な管理・運用の仕組み。

3.3 schema.md (データベース設計方針)

データベース全体の設計方針、ネーミングルール、マイグレーション方針、および全体ER図をまとめたドキュメントです。

- **構成項目**:
    - **データベース概要**: データベースの種類、バージョン、ホスティング環境方針。
    - **ネーミングルール**: テーブル、カラム、インデックス、外部キーの命名規則。
    - **全体ER図**: Mermaidを用いたエンティティ間の全体リレーション表現。
    - **マイグレーション方針**: テーブル変更管理ルール、マスタデータ投入方法。

3.4 models/ (物理モデル定義)

`schema.md` の設計方針に基づき、テーブルごとに個別に分割して作成する物理テーブル定義書群（`docs/models/<table-name>.md`）です。

- **構成項目**:
    - **モデル概要**: テーブル物理名、論理名、役割・責務。
    - **カラム定義**: 物理名、論理名、データ型、制約（PK/FK/Null）、デフォルト値、説明。
    - **インデックス定義**: プライマリキー、ユニークキー、外部キーインデックスなどの設計情報。
    - **アソシエーション（関連）**: 他テーブルとのリレーションシップ（1対1、1対多、多対多）。

Document Implementation Guide

開発・実装ドキュメント（`<root>/docs/[<group>/]<component>/backlog/`）の作成規約および運用ルールを定義します。

Overview

本ガイドは、機能仕様書（`SPEC-XXX.md`）に基づく実装計画の作成、実装タスクの分解方法、IDの命名規則、および実装タスク詳細ファイルの管理手順を定義するものです（※ 各ドキュメントのステータス定義・ライフサイクル遷移は document-status-guide.md を参照）。

docs/backlog/implementations/ (実装計画)

各増分機能仕様書（incremental specification）に対応した実装計画ファイルは、`<root>/docs/backlog/implementations/` 配下に集約して作成・保存します。複数コンポーネント間でのファイル名衝突を防ぐため、ファイル名先頭にソフトウェアコンポーネントの AppID（例: `webui-`）を付与します。実装計画は **Incremental Specification に対して必ず1つ（1:1:1）** 作成され、**Incremental型** と **Track型（計画分割）** があります。Track型は実装計画の中で Track 階層をつけて扱います。ファイル名は `<AppID>-<SPEC-I-ID>.md` となります。詳細な記述仕様、ステータス管理、セクション構成、およびテンプレートについては backlog-implementation-guide.md を参照してください。

tasks/ (実装タスク詳細)

実装計画に対応する実装タスク詳細ファイルは、増分仕様ごとに必ず1つ（1:1:1）作成し、`<root>/docs/[<group>/]<component>/backlog/tasks/<SPEC-I-ID>.md` としてフラットに配置・管理します。実装計画と同様に Incremental型 と Track型 があり、Track型はタスク詳細ファイル内に Track 階層を設けます。詳細な作成ルール、フロントマター属性、セクション構成、およびテンプレートについては backlog-task-guide.md を参照してください。

backlog/spec.md (仕様実装ステータス台帳・自動生成)

`backlog/spec.md` は、各ソフトウェアコンポーネント（`<root>/docs/[<group>/]<component>/backlog/spec.md`）におけるすべての機能仕様書（`SPEC-XXX.md`）の実装ステータスを一覧管理する台帳ビューです。本ファイルは `backlog/status.tsv` および `specs/` 配下の機能仕様書に基づき、Python スクリプト（`mise run aidev:backlog:sync`）により自動生成されます。

セクション構成と記述ルール

本台帳は機能仕様の一覧表のみで構成します。

SpecID	概要	ステータス	更新日
`SPEC-001`	サンプル機能仕様	WIP	2026-08-09

- SpecID: 該当する機能仕様書（`SPEC-XXX.md`）へのリンクを設定します。
- **概要**: 機能仕様書のフロントマターに記載された `description` を記述します。
- **ステータス**: `backlog/status.tsv` に記録された機能仕様書のステータス（`DRAFT` → `REVIEW:DRAFT` → `TODO` → `WIP` → `REVIEW:WIP` → `DONE` | `CLOSE`）を表示します。
- **更新日**: `backlog/status.tsv` の更新日（`YYYY-MM-DD`）を表示します。

backlog/status.tsv（仕様ステータス管理台帳）

`backlog/status.tsv` は、各ソフトウェアコンポーネントにおける機能仕様書（`SPEC-XXX`）および増分機能仕様書（`SPEC-I-XXX`）の実装ステータスを一元管理する正本台帳ファイルです。

- **形式**: KVTsv形式（`Key\tValue\tCreated\tUpdated`）
- **列構成**:
  - `Key`: 仕様ID（`<SPEC-ID>` または `<SPEC-I-ID>`、例: `SPEC-001`, `SPEC-I-001`）
  - `Value`: 最新ステータス（`DRAFT`, `REVIEW:DRAFT`, `TODO`, `WIP`, `REVIEW:WIP`, `DONE`, `CLOSE`）
  - `Created`: 初回登録日時（ISO 8601、自動設定）
  - `Updated`: 最終更新日時（ISO 8601、自動設定）
- **操作方法**:
  - ステータスの更新は `mise run aidev:kv-tsv -- set <path/to/status.tsv> <ID> <status>`（`AIDEV-TOOL-28`）を使用して行います。
  - ステータス変更後は `mise run aidev:backlog:sync`（`AIDEV-TOOL-29`）を実行して `spec.md` および `wbs.md` に反映します。

issues/（課題・不具合管理）

仕様として定まっていない課題、懸念事項、およびリリース後に発生した不具合報告などは、`<root>/docs/[<group>/]<component>/backlog/issues/` 配下に作成・管理します。詳細な命名規則、フロントマター属性、セクション構成、およびテンプレートについては backlog-issue-guide.md を参照してください。

todos/（TODO管理）

SPEC や ISSUE に該当しないが、将来的に検討・実施すべき作業メモや備忘録などは、`<root>/docs/[<group>/]<component>/backlog/todos/` 配下に作成・記録します。詳細なステータス管理、昇格（Promotion）ルール、およびテンプレートについては backlog-todo-guide.md を参照してください。

docs/backlog/phase.md (フェーズ別実装ステータス台帳ビュー)

`docs/backlog/phase.md` は、全体計画 (`docs/plan.md`) の各フェーズに帰属するユースケース、feature、および機能仕様書の実装状況を一覧管理する総合ビューです (詳細は backlog-phase-guide.md を参照)。

backlog/wbs.md (WBS・総合作業台帳・自動生成)

`backlog/wbs.md` は、各ソフトウェアコンポーネント (`<root>/docs/[<group>/]<component>/backlog/wbs.md`) における全タスク (`tasks/`)、課題 (`issues/`)、TODO (`todos/`) を一覧管理する総合台帳ビューです。本ファイルは `backlog/status.tsv`、`tasks/`、`issues/`、`todos/`、および `specs/incremental/` に基づき、Python スクリプト (`mise run aidev:backlog:sync`) により自動生成されます。

> Important
>
> **ステータス同期の原則**: `tasks/`, `issues/`, `todos/` や実装計画等のステータスを変更・更新する際は、必ず **`update-document-status` スキル** を使用してフロントマターおよび `backlog/status.tsv` を更新し、`mise run aidev:backlog:sync` により `wbs.md` と `spec.md` を自動同期してください (手作業による不整合の防止)。

セクション構成と記述順序

以下の順序でセクションを作成し、各カテゴリの台帳テーブルを記述します。

1. **タスク** (`tasks/` 配下の台帳)
2. **課題** (`issues/` 配下の台帳)
3. TODO (`todos/` 配下の台帳)

台帳テーブルフォーマット

1. タスク台帳

ID	概要	ステータス	更新日	SPEC-I	SPEC-I状態	SPEC-I更新日
SPEC-I-001	サンプル実装タスク	WIP	2026-08-09	SPEC-I-001	WIP	2026-08-09

- ID: 該当するタスク詳細ファイル (`tasks/<SPEC-I-ID>.md`) へのリンクを設定します。
- **概要**: タスク詳細ファイルのフロントマターの `description` を記述します。
- **ステータス**: タスク詳細ファイルのフロントマターの `status` を記述します。
- **更新日**: タスク詳細ファイルの更新日 (`YYYY-MM-DD`) を記述します。
- SPEC-I: 該当する増分仕様書 (`../specs/incremental/<SPEC-I-ID>.md`) へのリンクを設定します (存在しない場合は `-`)。

- **SPEC-I状態**: `backlog/status.tsv` に記録された該当増分仕様書のステータスを表示します。
- **SPEC-I更新日**: `backlog/status.tsv` に記録された該当増分仕様書の更新日（`YYYY-MM-DD`）を表示します。

2. 課題台帳

ID	概要	ステータス	更新日
`ISSUE-001`	サンプル課題	DRAFT	2026-08-09

- ID: 該当する課題管理ファイル（`issues/ISSUE-XXX.md`）へのリンクを設定します。
- **概要**: 課題管理ファイルのフロントマターの `description` を記述します。
- **ステータス**: 課題管理ファイルのフロントマターの `status` を記述します。
- **更新日**: 課題管理ファイルの更新日（`YYYY-MM-DD`）を記述します。

3. TODO台帳

ID	概要	ステータス	更新日
`TODO-001`	サンプルTODO	DRAFT	2026-08-09

- ID: 該当するTODO管理ファイル（`todos/TODO-XXX.md`）へのリンクを設定します。
- **概要**: TODO管理ファイルのフロントマターの `description` を記述します。
- **ステータス**: TODO管理ファイルのフロントマターの `status` を記述します。
- **更新日**: TODO管理ファイルの更新日（`YYYY-MM-DD`）を記述します。

Directory Structure Guide

プロジェクトの標準ディレクトリ構成および各ディレクトリ・ファイルの役割と配置規則を定義します。

Structure

```
<root>/
├── README.md                          # プロジェクト概要、セットアップ・使用方法
├── GEMINI.md                          # 【同期対象】AIエージェント共通指示・ルール定義
├── GEMINI-ja.md                       # 【同期対象】AIエージェント共通指示・ルール定義（日本語版）
├── GEMINI.local.md                    # 個人用AI指示書
├── GEMINI.project.md                  # プロジェクト固有のAI指示書
├── .agents/                           # AI開発用ハーネス・各種カスタムスキル群
│   ├── skills/                        # 【同期対象】review-document-status などの開発支援スキル
│   ├── agents/                        # 【同期対象】サブエージェント定義群
│   ├── hooks/                         # 【同期対象】フック処理スクリプト群
│   ├── hooks.gemini.json.sample       # 【同期対象】フック設定サンプル
│   ├── settings.gemini.json.sample    # 【同期対象】エージェント設定サンプル
│   └── statusline.gemini.ps1          # 【同期対象】ステータスライン表示スクリプト
├── .claude/                           # Claude Code 用設定・ハーネス
├── .local/                            # 【非追跡対象】ローカル専用作業・記憶領域
│   ├── artifacts/                     # 一時成果物（画像・変換出力等）
│   ├── memory/                        # 失敗・教訓・ナレッジ蓄積（write-memory）
│   ├── worklog/                       # 作業日報・残作業管理（write-worklog）
│   ├── workplan/                      # 作業計画ノート・手順管理（write-workplan）
│   └── workspace/                     # 一時作業領域・タスク・提案書（<workspace>）
├── .mise/                             # 【同期対象】mise 構成タスクおよびテンプレート用スクリプト群
│   ├── config.toml                    # テンプレート共通タスク定義
│   ├── scripts/                       # config.toml から呼び出される実行スクリプト群
│   └── tools/                         # config.toml / scripts から参照される内部ツール・モジュール群
├── docs/                              # 【同期対象】全体設計書・各コンポーネント設計書
│   ├── usecases/                      # ユースケース定義（USECASE-XXX.md）
│   ├── concept.md                     # 全体コンセプト概要
│   ├── architecture.md                # 全体基本設計・システム構成・技術選定
│   ├── security.md                    # 全体セキュリティ方針・ガイド
│   ├── plan.md                        # 全体計画書
│   ├── tools.md                       # 開発ツール利用ガイド（各プロジェクトで作成）
│   ├── documents.md                   # 文書インデックスおよび各ドキュメントへの誘導ガイド
│   ├── actions.md                     # アクションインデックスおよび状況別作業手順ガイド
│   ├── raw/                           # 不変ファイル・正本資料（人間管理）
│   ├── wiki/                          # AI文書・ナレッジ
│   │   └── reviews/                   # レビューレポート蓄積領域
│   ├── decision-records/              # 意思決定記録（DR-XXX.md）及び台帳（index.md）
│   ├── backlog/                       # 全体バックログ・スライス計画
│   │   ├── phase.md                   # フェーズ別実装ステータス台帳（ビュー）
│   │   ├── usecase.md                 # ユースケース・feature 実装ステータス台帳
```

```
│   │   ├── slices/                    # 実装スライス
│   │   └── implementations/           # 実装計画（各コンポーネントから集約、<AppID>-<ID>.md）
│   ├── development/                   # プロジェクト固有の開発資料・開発ガイド
│   ├── guide/                         # aidev-template 専用開発ガイド・テンプレート群
│   │   ├── directory-structure.md     # ディレクトリ構成と役割ガイド
│   │   ├── document-development-guide.md    # 開発プロセスガイド（DDD/SDD基本原則、設計・開発フェーズ反復、実装スライス等）
│   │   ├── document-design-guide.md   # 設計ガイド（concept, architecture, specs, design等）
│   │   ├── document-implementation-guide.md # バックログガイド（backlog以下のwbs, tasks, issues等）
│   │   ├── document-status-guide.md   # ドキュメントステータス・ライフサイクルガイド
│   │   ├── commit-message.md          # コミットメッセージ作成ガイド
│   │   ├── concept-guide.md           # concept.md 作成ガイド
│   │   ├── architecture-guide.md      # architecture.md 作成ガイド
│   │   ├── specification-guide.md     # 機能仕様書（SPEC-XXX.md）作成ガイド
│   │   ├── usecase-guide.md           # ユースケース定義書（USECASE-XXX.md）作成ガイド
│   │   ├── design-guide.md            # 詳細設計書（design.md）作成ガイド
│   │   ├── decision-record-guide.md   # 意思決定記録（DR-XXX.md）作成ガイド
│   │   ├── plan-guide.md              # 全体計画書（plan.md）作成ガイド
│   │   ├── backlog-phase-guide.md     # フェーズ台帳（phase.md）作成ガイド
│   │   ├── backlog-slice-guide.md     # 実装スライス作成ガイド
│   │   ├── backlog-implementation-guide.md # 実装計画作成ガイド
│   │   ├── backlog-task-guide.md      # 実装タスク詳細ファイル作成ガイド
│   │   ├── backlog-issue-guide.md     # 課題ファイル作成ガイド
│   │   ├── backlog-todo-guide.md      # TODOファイル作成ガイド
│   │   ├── tools-guide.md             # tools.md 作成ガイド
│   │   ├── local-guide.md             # .local/ 役割・運用ガイド
│   │   └── testing-guide.md           # テスト分類・配置規則および汎用検証チェックリスト作成ガイド
│   └── [<group>/]<component>/         # 各コンポーネント固有ドキュメント（apps/ に対応）
│       ├── requirements.md            # コンポーネント要件定義
│       ├── schema.md                  # DB・データモデル設計
│       ├── design.md                  # コンポーネント詳細設計
│       ├── specs/                     # 仕様書ディレクトリ（予約: index.md, index.link.tsv, id-seq-no.tsv, AGENTS.md, incremental/）
│       ├── models/                    # 個別物理モデル仕様
│       ├── raw/                       # コンポーネント不変ファイル・正本資料
│       ├── wiki/                      # コンポーネントAI文書・ナレッジ
│       └── backlog/                   # コンポーネントバックログ・進捗管理
│           ├── status.tsv             # 仕様ステータス台帳（KVTsv形式）
│           ├── wbs.md                 # WBS（実装タスク・TODO・issues 台帳・自動生成）
│           ├── spec.md                # 機能仕様実装ステータス台帳（自動生成）
│           ├── tasks/                 # 実装タスク詳細ファイル（<SPEC-I-ID>.md をフラットに配置）
│           ├── todos/                 # TODO管理ファイル
```

```
│           └── issues/                 # 課題管理ファイル
├── apps/                               # アプリケーションソースコード
│   └── [<group>/]<component>/          # 各コンポーネントの実装コード
├── data/                               # ドメインデータ・データセット・コンテンツ層（配下構造は任意）
├── deploy/                             # デプロイ構成・インフラ設定・環境定義（配下構造は任意）
├── dist/                               # 【非追跡対象】ビルド成果物・パブリッシュ用静的出力（配下構造は
任意）
├── tests/                              # テスト関連コード・検証用ファイル
│   ├── [<group>/]<component>/          # コンポーネントテスト（統合テスト・<test:appid>）
│   ├── performance/                    # パフォーマンス・負荷テスト
│   ├── security/                       # セキュリティ・脆弱性テスト
│   └── checklists/                     # 汎用検証チェックリスト（台帳およびテスト仕様書）
│       ├── checklist-<name>.md         # テスト仕様台帳
│       └── <name>/                     # 個別テスト仕様書ディレクトリ
│           └── <TESTSPEC-ID>.md        # テスト仕様書
├── scripts/                            # 各種処理のエントリーポイントスクリプト群（tools/ を使用）
└── tools/                              # scripts/ からのみ参照される内部ツール・ヘルパーモジュール群
```

Directory Details

Root Level Files

- `README.md`: プロジェクト概要、セットアップ・構築手順、主要コマンド等を記載します。
- `GEMINI.md`: AIエージェント（Gemini/Antigravity）共通の行動原則・ルールを定義します。
- `GEMINI.local.md`: 開発者個人用のローカルAI指示書です。
- `GEMINI.project.md`: プロジェクト固有の追加AIルール・ガイドラインを記載します。

`.local/`

AIエージェントおよび開発者のローカル専用作業領域および記憶領域です（**非追跡対象** / 詳細は local-guide.md 参照）。

- `.local/workspace/`: 一時作業領域（`<workspace>`）。一時タスク（`tasks.toml`）や改善提案書（`*-proposal.md`）の作成場所。
- `.local/memory/`: 会話や作業で得られた知見・失敗・教訓の蓄積領域（`write-memory` スキル連携）。
- `.local/worklog/`: セッションごとの作業日報および残作業（`todo.md`）の管理領域（`write-worklog` スキル連携）。

- `.local/workplan/`: 具体的な作業手順（`STEP-<ID>`）、作業記録、作業メモの一時管理領域（`write-workplan` スキル連携）。
- `.local/artifacts/`: 一時的な生成成果物（画像・変換出力等）の保管領域。
- `.local/docs/` / `.local/resources/`: ローカル検証専用の非追跡ドキュメントおよびリソースデータ。
- `.local/logs/`: ツールやスクリプトの実行ログ保管領域。

.mise/

`aidev-template` 基盤の共通タスク設定および実行用スクリプト群を管理します。

- `.mise/config.toml`: テンプレート共通のタスク定義ファイルです。
- `.mise/scripts/`: `.mise/config.toml` から呼び出される各種処理のエントリーポイントスクリプト群です。
- `.mise/tools/`: `.mise/scripts/` やテンプレートタスクからのみ参照される内部ツール・ライブラリ・ヘルパーモジュール群です。

docs/

プロジェクト全体の設計書およびコンポーネント別の仕様書・進捗管理ファイルを格納します。

Root Documents in docs/

- `docs/usecases/`: ユースケース定義文書（`USECASE-XXX.md`）を配置するディレクトリ。
- `docs/concept.md`: プロジェクト全体のコンセプト・ビジョン定義。
- `docs/architecture.md`: システム全体の基本設計・構成図・技術選定。
- `docs/security.md`: プロジェクトのセキュリティ方針・対策ガイドライン。
- `docs/plan.md`: プロジェクト全体の戦略・マイルストーン・フェーズ計画（plan-guide.md 参照）。
- `docs/tools.md`: プロジェクト固有の開発ツール（`tools/` および各種実行環境）の利用ガイド（各プロジェクト側で必要に応じて作成）。
- `docs/documents.md`: プロジェクト共通のドキュメント体系、設計書、ガイドライン、台帳の役割および各ドキュメントへの誘導手順をまとめた文書インデックス。
- `docs/actions.md`: 開発状況や作業目的に応じて実行すべきアクション、関連スキル、参照先ガイド、および具体的な実行フローをまとめたアクションインデックス。

- `docs/backlog/phase.md`: 全体計画に基づくフェーズ別ユースケース・仕様実装ステータス総合台帳（backlog-phase-guide.md 参照）。
- `docs/backlog/usecase.md`: 全ユースケース配下の feature 実装ステータス（`TODO` | `WIP` | `DONE`）を一括管理する台帳ファイル。
- `docs/backlog/slices/`: 実装スライス資料を配置するディレクトリ（ **追跡対象** ）。
- `docs/backlog/implementations/`: 各ソフトウェアコンポーネントの実装計画を集約配置するディレクトリ（ **非追跡対象** ）。ファイル名の衝突を防ぐため、ファイル名先頭に `<AppID>-` を付与します。
- `docs/decision-records/`: アーキテクチャや技術選定等の意思決定記録（`DR-XXX.md`）および総合台帳（`index.md`）を配置するディレクトリ。
- `docs/development/`: 各プロジェクト固有の開発資料・開発ガイドライン・環境構築手順・ノウハウ等を配置するディレクトリ。
- `docs/guide/`: aidev-template **専用** の開発運用ガイド・各種マニュアルおよびテンプレートを配置（`document-development-guide.md`、`document-design-guide.md`、`document-implementation-guide.md`、各ドキュメント作成用 `XXX-guide.md`、`commit-message.md` 等）。
- `raw/`: 人間が手動で管理する不変の正本資料・原稿ファイルを格納するディレクトリ（`<root>/docs/raw/` および `<root>/docs/[<group>/]<component>/raw/` に配置可能）。
- `wiki/`: AIによるAIのための「AI文書」を格納するディレクトリ（`<root>/docs/wiki/` および `<root>/docs/[<group>/]<component>/wiki/` に配置可能。※ 追跡対象にするかは任意）。`docs/raw/`、ソースコード、AIナレッジ、外部情報等を元に生成されます（`wiki/reviews/` には `review-document-status` スキルによるレビューレポートを蓄積）。非wikiファイルでデータを使用する場合は、wikiファイルではなく必ず `docs/raw/` 等の正本データを直接参照・使用してください。

Component Documents in `docs/[<group>/]<component>/`

`apps/[<group>/]<component>/` に対応する各ソフトウェアコンポーネントごとの詳細設計書を配置します（`<group>` は省略可）。

※ ここでの `<component>` は「ソフトウェアコンポーネント」（アプリケーション、ライブラリ、フレームワーク、データストア、ツール等）を意味します。React コンポーネント等との誤解を防ぐため原則「ソフトウェアコンポーネント」と表記しますが、コンテキストから明らかな場合は「コンポーネント」と省略可能です。

- `requirements.md`: コンポーネントごとの要件定義書。
- `schema.md`: DBスキーマ・テーブル設計・データモデル。
- `design.md`: コンポーネントの詳細設計書。

- `specs/`: 個別の機能仕様書・入出力仕様書を納めるディレクトリ。
- `models/`: 個別の物理モデル仕様書。
- `backlog/`: コンポーネントのバックログ管理ディレクトリ。
    - `status.tsv`: 機能仕様および増分仕様のステータス管理台帳（KVTsv形式）。
    - `wbs.md`: WBS（実装タスク・TODO・issues 台帳・自動生成ビュー）。
    - `spec.md`: 機能仕様実装ステータス台帳（自動生成ビュー）。
    - `tasks/`: 具体的な実装タスク詳細ファイル。
    - `todos/`: TODO管理項目。
    - `issues/`: 課題・障害管理ファイル。

apps/

実際のアプリケーションソースコードを配置します。

- `apps/[<group>/]<component>/`: コンポーネント単位の実装コードです（省略記法 `<src:<appid>>`）。`docs/[<group>/]<component>/`（`<doc:<appid>>`）および `tests/[<group>/]<component>/`（`<test:<appid>>`）と1対1で対応します。

> **Note**
>
> 各ソフトウェアコンポーネント（AppID）とディレクトリパス（`BaseDir`）のマッピング、および省略パス記法（`<src:<appid>>`, `<doc:<appid>>`, `<test:<appid>>`）は `GEMINI.project.md` にて一括定義されます。

data/

システムや各ソフトウェアコンポーネントが運用・利用するドメインデータ、データセット、マスターデータ、コンテンツ、静的リソースデータを格納します。アプリケーション実装コード（`apps/`）やプロジェクト設計書（`docs/`）から明確に分離します。

- 直下の内部構造は用途やプロジェクト構成に応じて任意とします。
- 特定のコンポーネント内部で閉じたデータは `apps/[<group>/]<component>/` 内で管理することも可能です。

deploy/

デプロイ用スクリプト、環境構成定義、インフラ構成マニフェスト（Docker, Kubernetes, Terraform, Wrangler等）を格納します。

- 直下の内部構造は用途や環境構成に応じて任意とします。

`dist/`

ソースコードやドキュメントからビルド・コンパイル・自動生成された配布物、パブリッシュ用静的エクスポート成果物（HTML/CSS/JS/バイナリ等）を出力します（ **非追跡対象** ）。

- 直下の内部構造は用途や出力構成に応じて任意とします。

`tests/`

テストコードおよび検証用チェックリストを配置します（詳細は testing-guide.md 参照）。単体テストは各ソフトウェアコンポーネントのソースコード内（`apps/` 配下等）に配置します。

- `tests/[<group>/]<component>/`: コンポーネントが別のコンポーネントを利用するコンポーネントテスト（統合テスト・`<test:appid>`）を配置します。
- `tests/performance/`: システムの負荷やパフォーマンスに関するテストを配置します。
- `tests/security/`: セキュリティ検証や脆弱性テストを配置します。
- `tests/checklists/`: パフォーマンスおよびセキュリティ以外の汎用的な検証チェックリストを配置します。
  - `checklist-<name>.md`: テスト対象機能群やサブシステム（`<name>`）に対応するテスト仕様台帳。
  - `<name>/<TESTSPEC-ID>.md`: 個別の検証シナリオやテスト項目を管理するテスト仕様書。

`scripts/` and `tools/`

自動化処理および実行環境です。

- `scripts/`: ビルド・デプロイ・データ処理等の各種実行エントリーポイントスクリプト群です。
- `tools/`: `scripts/` からのみ呼び出される内部ツール・ライブラリ・ヘルパーモジュールを格納します。

> Important
>
> **テンプレートタスクにおける配置規則**: `.mise/config.toml` から使用するツール・スクリプトは `.mise/scripts/` および `.mise/tools/` に配置します。`<root>/scripts/` および `<root>/tools/` は使用しません。

Commit Message Guide

本ガイドは、プロジェクトにおけるコミットメッセージの基本構造、ヘッダー部の構文フォーマット、プレフィックス（prefix）の適用ルール、および参照情報の記述規約を定義するものです。

Message Structure

コミットメッセージは以下の2部構造で構成されます。

1. **ヘッダー部 (Header)**: 主要な情報を集約して1行で記述する **必須** セクション。
2. **詳細部 (Details)**: 変更の動機・背景・実装詳細、補足情報等を記述する任意セクション（空行を挟んで記述）。

```
<prefix>[(<scope>)]: <summary> [Refs: <ref-id>]

[詳細部（Details） - 変更の動機や詳細な説明、補足情報]
```

Header Syntax & Format

基本として **ヘッダー部に主要な情報をすべて集約** して記述します。

基本構文

```
<prefix>[(<scope>)]: <summary> [Refs: <ref-id>]
```

要素	必須/任意	説明	例
`<prefix>`	原則必須	変更のカテゴリまたは増分仕様/課題ID	`SPEC-I-001`, `ISSUE-002`, `docs`
`(<scope>)`	任意	影響を受けるソフトウェアコンポーネント（AppID やモジュール）。ソフトウェアコンポーネント外では指定がない限り使用しない	`(webui)`, `(auth-service)`
`<summary>`	**必須**	変更内容の簡潔な1行要約。**破壊的変更** の場合は先頭に `BREAKING CHANGE:` を付与	`ログイン画面のバリデーションを追加`, `BREAKING CHANGE: 認証API構造を変更`
`[Refs: <ref-id>]`	任意	対象の増分仕様（`<SPEC-I-ID>`）を参照している親機能仕様（`<SPEC-ID>`）や、関連・影響を受ける他 SPEC / ISSUE ID	`[Refs: SPEC-001]`, `[Refs: SPEC-001, SPEC-I-002]`

Prefix Rules

コミット粒度の原則

- コミットは原則として **最小作業単位**（Work Item）で行います。
- 最小作業単位が不明な場合は、**最小の機能単位** または **最小の主題単位** に分割してコミットします。
- **ソースコードと関連文書の集約**: コミット単位には、実装したソースコードおよびそれに関連する文書（設計書・仕様書・テスト仕様書等）をまとめて含めます。

プレフィックス適用の判定基準

コミット単位の中に **ソフトウェアコンポーネントのソースコードが含まれているか否か** に応じて、適用する Prefix 規則を決定します。

- **ソースコードが含まれている場合**: 関連する文書が同時に含まれている場合も含め、**「1. ソフトウェアコンポーネントに対する変更」**の規則に従います。
- **ソースコードが含まれていない場合**: ドキュメント単体や環境設定等の変更であるため、**「2. ソフトウェアコンポーネント外に対する変更」**の規則に従います。

1. ソフトウェアコンポーネントに対する変更（apps/ 配下等）

コミット単位にソフトウェアコンポーネントのソースコード（`apps/` 配下等）が含まれる場合、`<prefix>` には原則として以下のいずれかを指定します。

- **`spec-i-id`**: 増分機能仕様書 ID（例: `SPEC-I-001`）
- **`issue-id`**: 課題・不具合 ID（例: `ISSUE-001`）
- **外部管理番号**: 外部プロジェクト管理ツール（Jira, Redmine 等）のチケット管理番号（例: `PROJ-123`）
- **`<scope>` の指定**: 変更が影響を与えるソフトウェアコンポーネント（AppID やモジュール）を括弧付きで指定します（例: `(webui)`）。

※ `<prefix>` なしのコミット表記もフォーマット上許可されますが、変更の追跡性を保つため **原則として非推奨** とし、適切な prefix を指定することを推奨します。

Refs (参照) の利用

対象の増分仕様（`<SPEC-I-ID>`）を参照している親機能仕様書（`<SPEC-ID>`）が存在する場合は、`Refs` にその `spec-id`（例: `SPEC-001`）を記述します。また、複数コンポーネント間で影響が広がる変更や関連する仕様がある場合は、関連する `spec-id`、`spec-i-id`、または `issue-id` をカンマ区切りで併記可能です（例: `[Refs: SPEC-001]`, `[Refs: SPEC-001, SPEC-I-002]`）。

2. ソフトウェアコンポーネント外に対する変更 (docs/, 全体設定等)

コミット単位にソフトウェアコンポーネントのソースコードが含まれない場合（ドキュメント単体や設定変更等）、基本プレフィックスとして `tools`、`docs` または `chore` を使用します。`docs` プレフィックスより他のプレフィックスを優先します。`docs` および `chore` プレフィックスを除いて、1つのコミットに複数のプレフィックスがある場合、コミットの分割を検討してください。

- **`tools`**: `scripts/` および `tools` 配下のファイル変更に適用します。
- **`docs`**: `docs/` 配下のファイル変更・ドキュメント更新に適用します。
- **`chore`**: ドキュメント以外の全体設定・ビルド構成・環境整備等に適用します。
- **任意の追加 prefix**: 必要に応じて `ci`, `refactor`, `test` などのプレフィックスを任意に追加・使用することを許可します。
- **`<scope>` の扱い**: `<scope>` はソフトウェアコンポーネントに対してのみ使用します。ソフトウェアコンポーネント外に対する変更では、指定がない限り `<scope>` は使用しません（例: `docs: ...`, `chore: ...`）。

3. 特例ルール (revert / merge)

- **Revert コミット**: コミットの打ち消し（revert）を行う場合、Git が自動生成するコミットメッセージ（例: `Revert "..."`）をそのまま使用します。
- **マージコミット**: ブランチ等のマージ（merge）を行う場合、Git や Pull Request 等が自動生成するコミットメッセージ（例: `Merge branch '...' into ...` や `Merge pull request #...`）をそのまま使用します。

Examples

- **増分機能仕様に基づく実装（親機能仕様を参照）**: `SPEC-I-001(webui): ログインフォームのバリデーション実装 [Refs: SPEC-001]`
- **複数 SPEC に影響する変更**: `SPEC-I-001(auth-service): 共通認証トークンの生成処理追加 [Refs: SPEC-001, SPEC-I-002]`
- **課題・不具合修正**: `ISSUE-005(webui): セッションタイムアウト時の再描画バグを修正`
- **ドキュメント更新**: `docs: コミットメッセージガイドラインを追加`
- **環境整備・設定更新**: `chore: CI設定の更新`
- **打ち消し (Revert)**: `Revert "SPEC-I-001(webui): ログインフォームのバリデーション実装"`
- **マージコミット (Merge)**: `Merge branch 'feature/login' into main`
- **破壊的変更 (Breaking Change)**: `SPEC-I-001(auth-service): BREAKING CHANGE: 認証APIレスポンスの構造を変更`

aidev-template Guide

本ガイドでは、`aidev-template` を活用したプロジェクト構築、開発環境のセットアップ、テンプレート同期スクリプト（`sync_template.py`）の使い方、および提案書を用いたテンプレートフィードバックの手順を総合的に解説します。

Overview

`aidev-template` は、AIエージェント（Antigravity, Claude Code, Codex等）と人間が協調して開発を進めるための基本システムおよび各種ガイドラインを提供するプロジェクトテンプレートです。

主な特徴

- **ハーネスエージェントの提供**: `.agents/` や `.claude/` にエージェント用カスタムスキル、ルール、フック設定を保持。
- **体系化された開発ドキュメント**: `docs/` 配下に設計、仕様、タスク、意思決定記録（Decision Records）などの標準フォーマットを用意。
- **一元化された環境・タスク管理**: `mise` を活用し、開発ツールの管理や実行エントリーポイントを共通化。
- **テンプレート同期機構**: `.mise/scripts/sync_template.py` を使用し、テンプレート本体の最新アップデートを各個別プロジェクトへ安全に同期。

Setup & Environment

`aidev-template` を使用するプロジェクトをセットアップするための手順と環境設定を解説します。

1. 動作前提条件

- PowerShell 7.x (`pwsh`): Windows 環境における標準シェル。
- Python 3.10 以上: スクリプト実行用。
- mise: タスクランナーおよび開発ツールバージョン管理ツール。
- Node.js: `.agents/hooks/` 配下のフックスクリプト（JavaScript）の実行用。
    - **フック用依存パッケージ**:
        - `ignore` パッケージ: `.agents/hooks/check-read-file.js` や `.agents/hooks/check-write-file.js` で `.devai.json`（`hook.read` / `hook.write`）のアクセス制御判定を行うために必要です。テンプレ

ート本体（`AIDEV_TEMPLATE_PATH`）側でインストール（`npm install ignore`）しておくことで、各派生プロジェクトから共有利用されます。

- **`markdownlint-cli2` パッケージ**: `.agents/hooks/lint-markdown.js` で Markdown ファイルの自動検証・修正を行うために必要です。テンプレート本体（`AIDEV_TEMPLATE_PATH`）側でインストール（`npm install markdownlint-cli2`）しておくことで、各派生プロジェクトから共有利用されます。

2. 環境変数 AIDEV_TEMPLATE_PATH の設定

テンプレートの同期・更新機能を利用するには、マスタとなる `aidev-template` リポジトリの絶対パスを環境変数 `AIDEV_TEMPLATE_PATH` に設定します。

3. 環境変数 MDTS_DOCS_PORT の設定

ドキュメントサーバー起動タスク（`mise run aidev:docs`）で使用するポート番号を環境変数 `MDTS_DOCS_PORT` に設定します。

.env ファイルでの設定例

プロジェクトルートの `.env` ファイルに記述することも可能です：

```
AIDEV_TEMPLATE_PATH=C:/dev/src/aidev-template
MDTS_DOCS_PORT=8521
```

3.1 環境診断とセットアップ支援（setup-aidev）

開発環境の前提ツールや設定が整っているかを一括で確認・診断するには、以下のタスクを実行します：

```
mise run aidev:check-env
```

また、AI エージェントに対して「セットアップを確認して」「環境を診断して」と指示することで、エージェントスキル `setup-aidev` が起動し、不足項目のガイドや初期設定を自動支援します。

4. ハーネス設定の初期セットアップ

`mise` タスクを実行して、エージェント設定ファイルおよびフックをローカル環境に配置します：

```
mise run aidev:setup-antigravity
```

このコマンドにより、`.agents/settings.gemini.json` および `.agents/statusline.gemini.ps1` がユーザープロファイル配下の設定ディレクトリに自動コピーされます。

また、フックで利用する Node.js パッケージ（`ignore`, `markdownlint-cli2` 等）はテンプレート本体（`AIDEV_TEMPLATE_PATH`）配下にインストールしておく必要があります（各個別プロジェクト側でのインストールは不要です）：

```
cd $env:AIDEV_TEMPLATE_PATH
npm install ignore markdownlint-cli2
```

Template Sync（sync_template.py）

テンプレート本体（`aidev-template`）が更新された際、個別プロジェクトへ最新のスキル、フック、ガイドドキュメント、環境設定を同期するための機能です。

1. 同期対象のディレクトリとファイル

テンプレート同期の対象となるディレクトリおよびファイルの一覧は、directory-structure.md の Overview ツリーにおいて `【同期対象】` と記載されている項目を参照してください。

2. 同期タスクの実行方法

`mise` に定義された以下のタスクを使用して同期を行います。

① 事前確認（ドライラン）

実際のファイルを変更せずに、変更対象（作成・更新・変更なし）のファイル一覧を確認します：

```
mise run "aidev:sync(dryrun)"
```

② 同期の実行

テンプレートから最新のファイルを上書き・追加作成します：

```
mise run aidev:sync
```

③ 同期スクリプト自体の更新（Self Update）

同期スクリプト本体（`.mise/scripts/sync_template.py`）も含めて同期を行う場合は、以下のコマンドを使用します：

```
# ドライラン
mise run "aidev:self-update(dryrun)"

# 同期実行
mise run aidev:self-update
```

同期処理完了後、同期元となったテンプレートリポジトリのコミットハッシュを含む推奨コミットメッセージが表示されます:

```
Recommended commit message:
  chore(aidev-template): pull 73dcee8
```

④ バックアップファイル (.bak) の整理とクリーンアップ

同期の際、既存ファイルが更新（`UPDATE`）されると、旧ファイルは自動的に `filename.bak` として同期先に一時退避されます。

内容を確認し、不要になったバックアップファイルは以下のタスクで安全に一括削除できます（同期対象外の無関係な `.bak` ファイルには影響しません）:

```
# 事前確認（ドライラン）
mise run "aidev:clean-bak(dryrun)"

# 対象の .bak ファイルを一括削除
mise run aidev:clean-bak
```

Proposal & Feedback Workflow

プロジェクト運用中に得られた知見やディレクトリ構造の拡張案などを、`aidev-template` 本体へ安全に取り込むための提案書（Proposal）によるフィードバック手順です。

※ 提案書はエージェントスキル `write-proposal`（または同スキル内のテンプレート構造）を利用して作成します。

```
flowchart TD
    A["1. 提案書の作成<br>(<workspace>/<topic>-proposal.md)"] --> B["2. AIエージェントへ取り込み指示<br>(topic: フィードバックを取り込みます...)"]
    B --> C["3. 変更案(Diff)の表示とユーザー確認"]
    C --> D["4. ドキュメント・設定への反映"]
    D --> E["5. テンプレート本体へコミット・同期"]
```

1. 提案書の作成場所と命名規則

- **作成領域**: ワークスペースディレクトリ内（`<workspace>`）
- **ファイル名**: `<topic>-proposal.md`（例: `aidev-template-proposal.md`, `directory-structure-proposal.md`）

2. 提案書の基本構造（テンプレート）

提案書は以下のフォーマットで作成します：

````
---
name: <topic>-proposal
title: "<提案のタイトル>"
description: "<提案内容の概要サマリー>"
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# <提案のタイトル>

## 1. 提案の背景と目的

現在の構成における課題や、拡張・変更を行う目的を記述します。

## 2. 拡張提案の詳細

具体的な変更仕様や新設するディレクトリ・機能のルールを定義します。

## 3. ドキュメントへの変更案（Diff）

修正対象ドキュメントに対する具体差分案を記述します。

```diff
--- docs/guide/target-document.md
+++ docs/guide/target-document.md
@@ -10,3 +10,5 @@
 現状の記述
+追加する記述
```

## 4. 期待される効果

本変更によって得られる保守性や開発効率向上のメリットを整理します。
````

3. AIエージェントへのフィードバック指示手順

提案書を作成後、AIエージェントに対して以下の指示を送ります：

```
topic: フィードバックを取り込みます。詳細は <workspace> にある提案書を確認してください
```

エージェントは提案書を読み込み、事前確認用の Diff を提示した上で、指定のドキュメントやスクリプトを自動更新します。

Related Documents

- directory-structure.md: プロジェクトの標準ディレクトリ構成および役割定義
- document-development-guide.md: 開発プロセスおよびドキュメント展開ガイド
- commit-message.md: Conventional Commits に準拠したコミットメッセージ規約

Concept Document Creation Guide

本ガイドは、プロジェクト全体（`<root>/docs/concept.md`）および個別のソフトウェアコンポーネント（`<root>/docs/[<group>/]<component>/concept.md`）において作成する `concept.md` の記述内容とMarkdownテンプレートを定義するものです。

Section Guidelines

`concept.md` は、「何を作るのか（What）」と「なぜ作るのか（Why）」を明確化する最上位ドキュメントです。以下の2つの必須セクションで構成します。

1. 概要 (What)

- 対象となるシステムまたはソフトウェアコンポーネントの全体像を記述します。
- 何を提供するのか、どのような主要機能や構成要素を持つのかを簡潔かつ明確にまとめます。

2. 目的 (Why)

- なぜこの開発を行うのか、背景や課題を明記します。
- 解決したい具体的なニーズ、達成すべきゴール、および期待されるビジネス/技術的価値を記述します。

Template

以下は `concept.md` を作成する際の標準Markdownテンプレートです。コピーしてご利用ください。

```
---
name: concept.md
description: <システムまたはコンポーネントの1行コンセプト概要>
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# Concept

## 概要

<対象となるシステムまたはソフトウェアコンポーネントのWhat（何を作るのか）を記述します。主要な機能や提供価値の全体像を明記してください。>

## 目的

<開発の背景や解決すべき課題、達成ゴールなどのWhy（なぜ作るのか）を記述します。>
```

Architecture Document Creation Guide

本ガイドは、システム全体の基本設計書である `<root>/docs/architecture.md` の記述内容とMarkdownテンプレートを定義するものです。

Section Guidelines

`architecture.md` はシステム全体のアーキテクチャ、構成図、コンポーネント定義、およびデータフローを明記するドキュメントです。以下の5つの必須セクションで構成します。

1. ディレクトリ構造

- プロジェクト全体のディレクトリ構成と、主要なディレクトリ・ファイルの役割をTree形式等で明記します。

2. システムアーキテクチャ

- システム全体の概要構造および全体構成図を記述します。
- 作図は原則として Mermaid 図 (`mermaid`) で作成してください。

3. ソフトウェアコンポーネント構成

- 各ソフトウェアコンポーネント (`<root>/apps/[<group>/]<component>`) の役割、技術スタック、責任範囲を記述します。

4. 依存・データフロー

- コンポーネント間の依存関係、通信プロトコル (REST, gRPC等)、データフローを記述します。

5. 関連

- 関連する上位設計書 (`concept.md`, `security.md`, `plan.md` 等) や外部参照資料へのリンクをまとめます。
- **意思決定記録 (DR) へのリンク規則**: 本基本設計の決定根拠となる意思決定記録 (`DR-XXX.md`) が存在する場合、**該当する技術選定・アーキテクチャ記述の直後に `([DR-001](./decision-records/DR-001.md))` の形式でインライン記述** し、「関連」セクションにも相互リンクを記載します。

Template

以下は `architecture.md` を作成する際の標準Markdownテンプレートです。

````
---
name: architecture.md
description: <システム全体の基本設計およびコンポーネント構成の定義>
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# System Architecture

## ディレクトリ構造

```txt
<root>/
├── docs/       # 設計ドキュメント
├── apps/       # ソフトウェアコンポーネントの実装コード
├── tests/      # テストコード
├── scripts/    # 実行用エントリーポイント
└── tools/      # スクリプト補助ツール
```

## システムアーキテクチャ

```mermaid
graph TD
    Client["Client (WebUI)"] --> API["API Gateway"]
    API --> ServiceA["Service A"]
    API --> ServiceB["Service B"]
```

## ソフトウェアコンポーネント構成

| コンポーネント名 | パス | 役割 | 主要技術スタック |
| :--- | :--- | :--- | :--- |
| `webui` | `apps/frontend/webui` | Web フロントエンド | TypeScript, React |
| `auth-service` | `apps/backend/auth` | 認証基盤 API | Python, FastAPI |

## 依存・データフロー

- `webui` → `auth-service`: HTTPS / REST API による認証リクエスト
- `auth-service` → DB: PostgreSQL 接続によるユーザーデータ参照 ([DR-001](./decision-records/DR-001.md))
````

```
## 関連

-   [concept.md](./concept.md)
-   [security.md](./security.md)
-   [plan.md](./plan.md)
-   [DR-001](./decision-records/DR-001.md)
```

Design Document Creation Guide

本ガイドは、各ソフトウェアコンポーネント配下（`<root>/docs/[<group>/]<component>/design.md`）に作成する詳細設計書である `design.md` の記述内容とMarkdownテンプレートを定義するものです。

Section & Rule Guidelines

1. 必須セクション構成と順序

1. **設計方針 (Design Policies)**: コンポーネント設計全体の概要およびアプローチ方針を記述。
2. **ビルド・成果物 (Build & Artifacts)**: 使用ツールごとに、プロジェクトファイル、成果物の種類、成果物の使い方（インポート方法、起動/デプロイ形態等、成果物自体の利用方法）を記述。
3. **レイヤー構造 (Layer Structure)**: 内部構成の L1（レイヤー）および L2（グループ）を一覧表（`| L1 | L2 | 役割・概要 |`）にまとめ。
4. **関係図 (Component & Layer Diagram)**: L1 + L2 による構造的関係図を Mermaid 図で作成。
5. **機能仕様書一覧 (Feature Specifications)**: 該当コンポーネント内のすべての Feature Spec（`specs/[<category>/]<SPEC-ID>.md`）を L1/L2 で分類したテーブル。※ `design.md` が参照する仕様は Feature Specification のみとし、増分仕様（Incremental Spec）や共通仕様（Common Spec）は参照しません。
6. **関連 (References)**: 上位設計書（`architecture.md`）、対応ユースケース（`USECASE-XXX.md`）、意思決定記録（`DR-XXX.md`）へのリンク。
    - **意思決定記録 (DR) へのリンク規則**: 本詳細設計の決定根拠となる意思決定記録（`DR-XXX.md`）が存在する場合、**該当する設計方針・レイヤー記述の直後に `([DR-001](../../decision-records/DR-001.md))` 形式でインライン記述** し、「関連」セクションにも相互リンクを記載します。

Template

以下は `design.md` を作成する際の標準Markdownテンプレートです。

````
---
name: design.md
description: <コンポーネント詳細設計の概要>
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# Component Detailed Design: <コンポーネント名>

## 設計方針

<コンポーネント全体の設計概要、基本方針、採用アーキテクチャの適用内容を記述します ([DR-001](../../decision-records/DR-001.md))。>

## ビルド・成果物

### <使用開発ツール名 1 (例: Node.js / Vite)>

- **プロジェクトファイル**: `package.json`, `tsconfig.json`
- **成果物の種類**: ライブラリバンドル (ESM/CommonJS)
- **成果物の使い方**: `import { authHandler } from '@myapp/auth-service'` として他モジュールからインポートして利用、またはスタンドアロンサービスプロセスとして起動

## レイヤー構造

| L1 | L2 | 役割・概要 |
| :--- | :--- | :--- |
| `Presentation` | `AuthHandler` | 認証リクエスト受入れ・応答変換 |
| `Domain` | `AuthService` | 認証・トークン制御コアロジック |
| `Infrastructure` | `UserRepository` | ユーザー情報の永続化アクセス |

## 関係図

```mermaid
flowchart TD
    subgraph Presentation["Presentation (L1)"]
        AuthHandler["AuthHandler (L2)"]
    end
    subgraph Domain["Domain (L1)"]
        AuthService["AuthService (L2)"]
    end
    subgraph Infrastructure["Infrastructure (L1)"]
````
````

````
        UserRepository["UserRepository (L2)"]
    end

    AuthHandler --> AuthService
    AuthService --> UserRepository
```

## 機能仕様書一覧

本コンポーネントがカバーするすべての機能仕様書をレイヤー（L1）およびグループ（L2）ごとに分類した一覧です。

| L1 | L2 | SpecID | 概要 |
| :--- | :--- | :--- | :--- |
| `Presentation` | `AuthHandler` | [`SPEC-001`](./specs/auth/SPEC-001.md) | ユーザー認証 API 仕様書 |
| `Domain` | `AuthService` | [`SPEC-002`](./specs/user/SPEC-002.md) | ユーザー情報変更仕様書 |

## 関連

- [architecture.md](../../architecture.md)
- [docs/usecases/USECASE-001.md](../../usecases/USECASE-001.md)
- [DR-001](../../decision-records/DR-001.md)
````
````

Specification Document Creation Guide

本ガイドは、各ソフトウェアコンポーネント配下の `specs/` ディレクトリに作成する機能仕様書（Feature Specification: `specs/[<category>/]<SPEC-ID>.md`）、増分機能仕様書（Incremental Specification: `specs/incremental/<SPEC-I-ID>.md`）、共通仕様書（Common Specification: `SPEC-C-<name>.md`）、仕様総合台帳（`specs/index.md`）、仕様リンク台帳（`specs/index.link.tsv`）、および仕様採番管理台帳（`specs/id-seq-no.tsv`）の構成規則、記述内容、およびMarkdownテンプレートを定義するものです。

基本方針

- **自己完結（Self-contained）原則**: Incremental Spec に差分（パッチ）記述を持ち込まず、最小機能単位の完全な仕様として記述します。
- **参照到達性（Reachability）によるライフサイクル管理**: Feature Spec から参照されている Incremental Spec を現役（Live）とし、参照されないものを過去ログ（Historical）として決定論的に判定します。
- **コード変更時の Incremental Spec 必須原則**: ソースコードに変更を加えるすべての作業は必ず Incremental Spec を経由するものとし、ISSUE から直接実装計画やタスク詳細を作成することを禁止します。

ディレクトリ構成と運用ルールの解決順序

各ソフトウェアコンポーネントの `specs/` ディレクトリ配下の構成（サブディレクトリ分類や配置ルール）は、以下の優先順位で解決・適用されます。

1. **予約要素**: `specs/index.md`、`specs/index.link.tsv`、`specs/id-seq-no.tsv`、`specs/AGENTS.md`、および `specs/incremental/` は予約されており、`specs/` 直下に配置します。
2. **機能仕様の判定**: `specs/` 配下のサブディレクトリを含め、`<SPEC-ID>.md` というファイル名を持つものが機能仕様です。
3. **増分仕様の判定**: `specs/incremental/` 配下にある `<SPEC-I-ID>.md` のみを増分仕様とします（例: `specs/archive/<SPEC-I-ID>.md` のように `specs/incremental/` 以外の場所にあるファイルは、増分仕様や機能仕様とはみなされず `index.md` 台帳には登録されません）。
4. **共通仕様の配置と命名**: `specs/` 配下に配置する共通仕様のファイル名は `SPEC-C-<name>.md` とします。既定では `specs/common/` や `specs/model/` への配置を推奨とします。
5. **構成定義（`specs/AGENTS.md`）**: `specs/AGENTS.md` に `## Structure` 見出しを作成し、3列テーブル（`パス` | `配置対象` | `命名規則`）でサブディレクトリ構成や共通仕様の配置先を定義します。定義がない場合の機能仕様は `specs/<SPEC-ID>.md`（直下フラット配置）とします。

仕様書の種別と責務

1. Feature Specification (`specs/[<category>/]<SPEC-ID>.md`)

- **役割**: 各機能の仕様を網羅する責任を持ちます。`specs/` 配下のサブディレクトリを含め `<SPEC-ID>.md` の形式とします。
- **記述内容**: 機能全体のフロー（Mermaidシーケンス図・必須）とインターフェース契約を定義し、構成する最新の Incremental Spec 群の台帳（BOM: 部品表）を保持します。

2. Incremental Specification (`specs/incremental/<SPEC-I-ID>.md`)

- **役割**: 実装スライスの最小単位となる自己完結した機能仕様です。`specs/incremental/` 直下にのみ配置されます。
- **記述ルール**: **差分記述は禁止**とし、単体で動作・検証可能な完全形として記述します。

3. Common Specification (`specs/[<category>/]SPEC-C-<name>.md`)

- **役割**: incremental specification から参照される共通仕様（データモデル、DTO、共通規約、エラー定義等）です。
- **ファイル名**: `SPEC-C-<name>.md`（例: `SPEC-C-auth-dto.md`, `SPEC-C-error-handling.md`）。
- **横断参照**: 他のソフトウェアコンポーネントから参照することも可能です。
- **配置場所**: 既定では `specs/common/` や `specs/model/` への配置を推奨とし、`specs/AGENTS.md` で変更可能です。

4. Specification Resource Index (`specs/index.md`)

- **役割**: コンポーネント内の全仕様リソースの相互参照（Reachability）を追跡・記録する総合台帳。
- **各表の列構成**: `識別子/ファイル名` | `概要` | `参照` | `更新日`
  - `概要`: 対象仕様ファイルのフロントマターにある `description` を記述。
  - `参照`: その仕様を参照している仕様の ID（リンク付き）を記述。参照元がない場合は `-` を記述。
  - `更新日`: 対象仕様ファイルのフロントマターにある `timestamp` を記述。

5. Specification Link Ledger (`specs/index.link.tsv`)

- **役割**: 増分仕様および共通仕様の参照関係（グラフ）を管理する TSV ファイル。
- **列構成**: `Path` | `Refs`（タブ区切り）
  - `Path`: 各仕様の `specs/` からの相対パス（例: `incremental/SPEC-I-001.md`, `common/SPEC-C-error-handling.md`）
  - `Refs`: その仕様を参照しているファイルの `specs/` からの相対パスリスト（`,` で連結したもの）。
    - 増分仕様の場合: その仕様を参照している機能仕様（`<SPEC-ID>.md`）のリスト
    - 共通仕様の場合: その仕様を参照している増分仕様（`<SPEC-I-ID>.md`）のリスト
    - 外部参照（他コンポーネントからの参照）は対象外

- **更新タイミングと整合性検証**:
  - `index.link.tsv` は `index.md` の作成・更新と同時に作成・更新します。
  - 更新時に、増分仕様から外部の共通仕様へのリンク検証を行い、参照対象が存在するか確認します。

6. Specification ID Sequence Ledger (`specs/id-seq-no.tsv`)

- **役割**: 機能仕様書（`<SPEC-ID>`）および増分機能仕様書（`<SPEC-I-ID>`）の最大連番（次に割り当てる番号）を一元管理する TSV ファイル。仕様のアーカイブや削除が発生した場合でも、ID の重複・衝突を確実に防止します。
- **形式**: KVTsv形式（`Key\tValue\tCreated\tUpdated`）
- **列構成**:
  - `Key`: ID 種別（`SPEC-ID`, `SPEC-I-ID`）
  - `Value`: 次回採番する連番（SeqNo）
  - `Created`: 初回作成日時（ISO 8601、自動設定）
  - `Updated`: 最終更新日時（ISO 8601、自動設定）
- **初期値**:

```
Key     Value   Created Updated
SPEC-ID 1       2026-09-04T00:00:00+09:00       2026-09-04T00:00:00+09:00
SPEC-I-ID       1       2026-09-04T00:00:00+09:00       2026-09-04T00:00:00+09:00
```

- **採番運用ルール**:
  - 新しく機能仕様または増分仕様を作成する際は、本ファイルを参照して該当 ID の `Value` を採番し、値をインクリメント（+1）して保存します。
  - `mise run aidev:kv-tsv`（`AIDEV-TOOL-28`）を利用して値の取得（`get`）やインクリメント（`inc`）を行うことができます。

参照到達性（Reachability）と仕様変更ルール

1. **ライフサイクル判定**:
   - いずれかの Feature Spec から参照されている Incremental Spec を **Active（現役）** と判定します。
   - すべての Feature Spec から参照されていない Incremental Spec を **Historical（過去・廃止）** と判定します。
2. **仕様変更フロー**:
   - **全体共通の改善・バグ修正**: 既存の `SPEC-I-xxx.md` を完全形のまま直接更新します。

- **特定機能都合・破壊的変更**: 新規 `SPEC-I-yyy.md` を作成し、対象 Feature Spec の台帳リンクを差し替えます（古い `SPEC-I` は自動的に Historical 化）。

仕様台帳ツール・検証ツールの利用方法 (CLI Tools)

仕様書の追加・変更を行った際は、以下の mise タスクを使用して台帳（`index.md`, `index.link.tsv`）の自動同期および外部リンクの検証を実行します。

1. 仕様台帳の一括同期・一括検証

```
# index.md, index.link.tsv の自動同期および外部共通仕様リンク検証を一括実行
mise run aidev:spec:sync

# 差分やリンク切れがないかを一括検証
mise run aidev:spec:check
```

2. 個別ツールの実行

- **仕様総合台帳（`specs/index.md`）の生成・同期**:

  ```
  # 自動生成・同期
  mise run aidev:spec:index:sync
  # 検証（差分チェック）
  mise run aidev:spec:index:check
  ```

- **仕様リンク台帳（`specs/index.link.tsv`）の生成・同期**:

  ```
  # 自動生成・同期
  mise run aidev:spec:link:sync
  # 検証（差分チェック）
  mise run aidev:spec:link:check
  ```

- **外部共通仕様参照のリンクチェック**:

  ```
  # 増分仕様から他コンポーネントの SPEC-C-*.md へのリンクが実在するか検証
  mise run aidev:spec:check-external-links
  ```

Template

1. Feature Specification テンプレート（specs/auth/SPEC-001.md）

````
---
name: SPEC-001.md
title: <機能名称>
description: <機能仕様の全体概要>
status: active
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# SPEC-001: <機能名称>

## 概要

<本機能が提供する価値およびセッション管理・処理全体の概要を記述します（[DR-001](../../decision-records/DR-001.md)）。>

## 機能フロー（必須）

```mermaid
sequenceDiagram
    autonumber
    Client->>API: 1. 要求送信
    API->>API: 2. 処理・検証
    API-->>Client: 3. 応答返却
```

## 機能契約・インターフェース要約

-   **公開インターフェース**: `/api/v1/...`
-   **適用規約**: `specs/common/SPEC-C-error-handling.md`

## 構成 Incremental 仕様台帳（Active Specifications）

| 分野 / 対象 | 参照 Incremental Spec | 概要 | 適用・更新日 |
| :--- | :--- | :--- | :--- |
| **基本処理** | [SPEC-I-001](../incremental/SPEC-I-001.md) | 基本ロジック | YYYY-MM-DD |
| **拡張処理** | [SPEC-I-002](../incremental/SPEC-I-002.md) | 拡張機能 | YYYY-MM-DD |
````

```
## 関連

-   [design.md](../../design.md)
-   [DR-001](../../decision-records/DR-001.md)
```

---

2. Incremental Specification テンプレート（specs/incremental/SPEC-I-001.md）

```
---
name: SPEC-I-001.md
title: <最小機能仕様名称>
status: active
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# SPEC-I-001: <最小機能仕様名称>

## 概要

<本増分仕様が定義する単独完結した処理内容を記述します。>

## 参照モデル・共通仕様

-   [specs/model/SPEC-C-auth-dto.md](../model/SPEC-C-auth-dto.md)
-   [specs/common/SPEC-C-error-handling.md](../common/SPEC-C-error-handling.md)

## 入力・出力仕様

### 入力パラメータ

| パラメータ名 | 型 | 必須 | 説明 |
| :--- | :--- | :--- | :--- |
| `user_id` | string | ○ | ユーザー識別ID |

### 出力仕様

| 項目名 | 型 | 説明 |
| :--- | :--- | :--- |
| `token` | string | 発行されたトークン |

## 機能・振る舞い仕様

```mermaid
sequenceDiagram
    Client->>API: 処理要求
    API-->>Client: 200 OK
```
```

```
1. 入力パラメータの妥当性を検証する。
2. 対象のデータを取得・加工する。
3. レスポンスオブジェクトを構築して返却する。

## エラー・例外処理

| エラーコード | 発生条件 | 処理内容 |
| :--- | :--- | :--- |
| `ERR_INVALID_PARAM` | パラメータの型不正 | 400 Bad Request を返却 |
```

---

3. 仕様総合台帳テンプレート (specs/index.md)

```
---
name: index.md
description: 仕様リソース総合台帳 (Incremental Spec, Common Spec 参照追跡)
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# Specification Resource Index

## 1. Incremental Specifications

| Incremental ID | 概要 | 参照 | 更新日 |
| :--- | :--- | :--- | :--- |
| [`SPEC-I-001`](./incremental/SPEC-I-001.md) | 基本認証ロジック仕様 | [`SPEC-001`](./auth/SPEC-001.md) | YYYY-MM-DD |

## 2. Common Specifications

| 共通仕様ファイル | 概要 | 参照 | 更新日 |
| :--- | :--- | :--- | :--- |
| [`SPEC-C-error-handling.md`](./common/SPEC-C-error-handling.md) | 共通エラーハンドリング規則 | [`SPEC-I-001`](./incremental/SPEC-I-001.md) | YYYY-MM-DD |
| [`SPEC-C-auth-dto.md`](./model/SPEC-C-auth-dto.md) | 認証リクエスト・レスポンスDTO定義 | [`SPEC-I-001`](./incremental/SPEC-I-001.md) | YYYY-MM-DD |
```

---

4. 仕様リンク台帳例（specs/index.link.tsv）

```
Path    Refs
incremental/SPEC-I-001.md       auth/SPEC-001.md
incremental/SPEC-I-002.md       auth/SPEC-001.md
common/SPEC-C-error-handling.md incremental/SPEC-I-001.md,incremental/SPEC-I-002.md
model/SPEC-C-auth-dto.md        incremental/SPEC-I-001.md
```

---

5. 仕様採番管理台帳例（specs/id-seq-no.tsv）

```
Key     Value   Created Updated
SPEC-ID 1       2026-09-04T00:00:00+09:00       2026-09-04T00:00:00+09:00
SPEC-I-ID       1       2026-09-04T00:00:00+09:00       2026-09-04T00:00:00+09:00
```

Usecase Creation Guide

本ガイドは、複数コンポーネントにまたがる機能群やシナリオを束ねる「ユースケース」を `<root>/docs/usecases/USECASE-XXX.md` 配下に作成する際の記述ルールと標準Markdownテンプレートを定義するものです。

概念定義（ユースケースと実装単位 feature）

- **ユースケース（`usecase`）**:
  - ユーザー視点またはシステム視点での目的・ユースケースを表すグループ構造。
  - 配置先: `<root>/docs/usecases/USECASE-XXX.md`
  - ID命名規則: `USECASE-XXX`（`XXX` は `001` から始まる 3桁ゼロ埋め数字。例: `USECASE-001.md`）。
  - 役割: 機能単位（`feature`）や実現条件、シーケンス図等を束ね、仕様・依存関係を永続的に記録・追跡します（※ feature の実装ステータス管理は `<root>/docs/backlog/usecase.md` で行います）。
- **機能単位（`feature`）**:
  - 動作する機能の最小単位。
  - 命名・ID規則: `F-<no>: <feature-name>`（`<feature-name>` は任意の言語。例: `F-1: ユーザーログイン`、`F-1: user-login`）。
  - 外部参照形式: 外部文書から特定の feature を参照する場合は `<UseCaseID>:<FeatureID>`（例: `USECASE-001:F-1`）と表記します。
  - ユースケース内に含まれる縦切りスライスの実体です。
- **実装スライス**:
  - 複数コンポーネントを横断して縦切りにした実装単位。
  - ファイル名: `docs/backlog/slices/<slice-name>.md`（ファイル名は `kebab-case`。**追跡対象** ）
  - 参照規則: 実装スライスから親ユースケースへの単方向参照を記述します。

運用手順 & 台帳への登録

1. **ユースケースファイルの起票**:
   - `<root>/docs/usecases/USECASE-XXX.md`（例: `USECASE-001.md`）として新規作成します。
2. **feature ステータス台帳への登録**:
   - ユースケース起票時、`docs/backlog/usecase.md` に該当ユースケースの節（`## USECASE-XXX: <ユースケース名>`）、ユースケースステータス（`- **ステータス**: DRAFT（YYYY-MM-DD）`）、実現条件（`- **実現条件`

**: ...`)、および feature 表を作成し、各 feature を `DRAFT` ステータスで登録します。

3. **実装とステータス更新**:
    - **`status` の遷移**: `DRAFT`（初期値）→ `REVIEW:DRAFT` → `TODO` → `WIP` → `REVIEW:WIP` → `DONE`
        - `DRAFT`: ユースケース・feature 定義中
        - `REVIEW:DRAFT`: 策定フェーズのレビュー中（`review-document-status` 実行）
        - `TODO`: レビュー完了・着手待ち
        - `WIP`: feature の実装スライス（`docs/backlog/slices/<slice-name>.md`）が着手・進行中の状態
        - `REVIEW:WIP`: feature 実装スライス完了後、ユースケース受入レビュー中（`review-document-status` 実行）
        - `DONE`: 実装スライスが完了し、実現条件を満たした状態
    - ステータス遷移に応じて `docs/backlog/usecase.md` の feature 行およびユースケース自体の `ステータス (Timestamp)` を更新します。
    - feature の完了によって実現条件を満たした際、ユースケース自体のステータスを `DONE` に更新します。
    - 実現条件が `OR` や `XOR` 等でユースケースが `DONE` となった場合、採用・実装されなかった選択 feature は `TODO` のまま残します（将来の拡張・代替候補として保持）。

セクション記述ルール

1. フロントマター

- 標準項目（`name`, `description`, `timestamp`, `ai`）を設定します（※ ユースケースおよび feature の実装進捗ステータスは、フロントマターではなく台帳 `docs/backlog/usecase.md` で一括管理します）。

2. 概要 (Overview)

- 本ユースケースが提供するユーザー価値またはシステム目的を記述します。

3. 実現条件 (Realization Conditions)

本ユースケースの目的を達成するための条件を記述します。

- **論理式による記述**:
    - `AND`, `OR`, `XOR` を使用した論理式で記述します。
        - `F-1 AND F-2`: F-1 と F-2 の両方の実装が必要
        - `F-1 OR F-2`: F-1 または F-2 の実装が必要
        - `F-1 XOR F-2`: F-1 か F-2 のいずれか一方の実装が必要
    - **条件付き feature の展開**:

    - F-2 が条件付き（種類: 条件付き）の場合、`F-1 AND F-2` は `F-1 AND (IF <F-2条件> THEN F-2 ELSE TRUE)` と展開されます。
    - 条件付き feature は `AND` のみ指定可能です。

- **文章による記述**:
    - 論理式では表現できない複雑な実現条件の場合は、文章で記述します。

4. feature 詳細 (Feature Details)

各 feature は見出し `### F-<no>: <feature-name>` として定義し、以下の項目を記述します。

1. **概要**: 機能の具体的仕様・振る舞いを記述。
2. **関連ソフトウェアコンポーネント・仕様書**: 関連するコンポーネントおよび機能仕様書（Feature Specification）（`<AppID>` または `<AppID>:<SpecID>` 形式。例: `webui:SPEC-001`, `auth-service:SPEC-001`）。
3. **完了条件**: 当該 feature が完了したとみなす条件。
4. **種類**: `必須` | `選択` | `条件付き` のいずれか。
5. **条件**: 種類が `条件付き` の場合に、適用される条件を記述（必須・選択の場合は不要または「なし」）。
6. **シーケンス図**: Mermaid 書式で処理の流れやコンポーネント間の連携を記述。

Template

以下はユースケース定義書を作成する際の標準Markdownテンプレートです。

```
---
name: USECASE-001.md
description: <ユースケースの1行概要>
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# USECASE-001: <ユースケース名>

## 概要

<ユースケースの目的と概要を記述します。>

## 実現条件

`F-1 AND F-2`

<論理式で表現できない複雑な条件がある場合は文章で記述します。>

## feature 詳細

### F-1: user-login

#### 概要

<ユーザーログイン機能の具体的仕様・振る舞いを記述します。>

#### 関連ソフトウェアコンポーネント・仕様書

- `webui:SPEC-001`（ログイン画面UI）
- `auth-service:SPEC-001`（認証API）

#### 完了条件

- ユーザーが認証情報を入力して正常にログインできること
- 認証失敗時に適切なエラーメッセージが表示されること

#### 種類

必須

#### 条件
```

なし

シーケンス図

```mermaid
sequenceDiagram
    autonumber
    actor User as ユーザー
    participant UI as webui
    participant Auth as auth-service

    User->>UI: ログイン情報入力 & 送信
    UI->>Auth: POST /api/login (credentials)
    Auth-->>UI: 200 OK (token)
    UI-->>User: ログイン完了画面表示
```

F-2: password-reset

概要

<パスワードリセット機能の具体的仕様・振る舞いを記述します。>

関連ソフトウェアコンポーネント・仕様書

- `webui:SPEC-002`（パスワード再設定画面UI）
- `auth-service:SPEC-002`（トークン検証・パスワード更新API）

完了条件

- パスワード再設定メールが送信され、リンクから新パスワードを設定できること

種類

条件付き

条件

メール配信サービスが利用可能な環境であること

シーケンス図

```mermaid
sequenceDiagram
    autonumber
    actor User as ユーザー
    participant UI as webui
    participant Auth as auth-service

    User->>UI: パスワード再設定要求
    UI->>Auth: POST /api/password-reset/request
    Auth-->>UI: 200 OK
    UI-->>User: 再設定メール送信完了表示
```

Decision Record Creation Guide

本ガイドは、システムアーキテクチャや技術選定、設計方針に関する重要な意思決定を記録・追跡するために `<root>/docs/decision-records/` 配下に作成する意思決定記録（`DR-XXX.md`）の記述内容とMarkdownテンプレートを定義するものです。

Rules & Section Guidelines

1. 命名規則・番号管理

- **ファイル名**: `DR-XXX.md` の形式で3桁以上の連番により管理します（例: `DR-001.md`, `DR-002.md`）。
- **台帳への登録**: ファイルを作成した際は、必ず `<root>/docs/decision-records/index.md` の一覧テーブルに登録・追記してください。

2. 改訂と新規作成の判断基準

- **新規DR作成（破壊的変更）**: 既存の決定内容を根本から覆す変更や非互換な方針転換（コミットにおける `BREAKING CHANGES` 相当）を行う場合は、既存DRを上書きせず新規にDRを発番・作成します。
    - 旧DRのステータスは `SUPERSEDED` に更新し、新DRへのリンクを明記します。
- **既存DRの改訂（軽微な変更・修正）**: 誤字脱字の修正、説明の補足・明確化、関連ドキュメントリンクの追加など、決定方針自体を覆さない軽微な変更・修正の場合は既存DRを改訂できます。
    - 改訂時は、末尾の「更新履歴」セクションに日付と変更内容を記録します。

3. フロントマターとステータス管理

- 標準項目（`name`, `description`, `timestamp`, `ai`）に加え、`status`, `review` キーを記述します。
- `status` **の値**:
    - `PROPOSED`: 提案中・議論中
    - `ACCEPTED`: 承認・採択済み
    - `REJECTED`: 却下・非採択
    - `DEPRECATED`: 非推奨・廃止
    - `SUPERSEDED`: 後続の意思決定により置換済み（`DR-YYY` により置換等）
- **フロントマター** `review`: `result`（`ACCEPTED` | `REJECTED`）, `timestamp`, `reason`（REJECTED時の1行理由サマリー。ACCEPTED時は `""`）
- **ステータス更新**: ステータス変更時は、`update-document-status` スキルを使用してフロントマターおよび台帳（`docs/decision-records/index.md`）を一括更新すること。

4. 必須セクション構成

1. **コンテキストと背景** (Context & Problem Statement)
    - 意思決定が必要となった背景、解決すべき課題、制約条件を記述します。
2. **決定事項** (Decision)
    - 採択した方針や技術選定結果を明確に記述します。
3. **検討した選択肢** (Options Considered)
    - 比較・検討した他の代替案とそのメリット・デメリットを記述します。
4. **帰結・影響** (Consequences)
    - この決定によって生じるプラス・マイナスの影響やトレードオフ、今後のアクションを記述します。
5. **関連ドキュメントとのインライン相互リンク必須化**
    - この意思決定によって影響を受ける・決定根拠となる設計書や仕様書（`architecture.md`, `design.md`, `SPEC-XXX.md` 等）が存在する場合、**対象記述の直後に `([DR-001](...))` 形式でインラインリンクを記述**するとともに、「帰結・影響」または「関連」セクション内に相互リンクを明記します。
6. **更新履歴** (Changelog)
    - 初版作成日および改訂時の日付・変更内容を時系列で記述します。

Template

以下は意思決定記録（`DR-001.md`）を作成する際の標準Markdownテンプレートです。

```
---
name: DR-001.md
description: <意思決定事項の1行概要>
timestamp: YYYY-MM-DD
ai: ai-coauthored
status: ACCEPTED # PROPOSED | ACCEPTED | REJECTED | DEPRECATED | SUPERSEDED
review:
  result: "" # ACCEPTED | REJECTED
  timestamp: "" # YYYY-MM-DD
  reason: "" # REJECTED時の1行理由サマリー
---

# DR-001: <意思決定タイトル・件名>

## コンテキストと背景

<意思決定が必要となった背景、発生していた問題、技術的/ビジネス的制約条件を記述します。>

## 決定事項

<最終的に採択・決定した方針や技術選定の内容を明確に記述します。>

## 検討した選択肢

### 選択肢 1: <案1の名称（採択案）>

- **メリット**: <メリットを記述>
- **デメリット**: <デメリットを記述>

### 選択肢 2: <案2の名称>

- **メリット**: <メリットを記述>
- **デメリット**: <デメリットを記述>

## 帰結・影響

- **プラスの影響**: <決定によって得られる効果やポジティブな結果>
- **マイナスの影響 / トレードオフ**: <決定に伴う受容すべきリスクや課題>
- **関連ドキュメント（必須）**:
    - [`architecture.md`](../architecture.md)
    - [`SPEC-001`](../auth/specs/SPEC-001.md)
```

```
## 更新履歴

-   YYYY-MM-DD: 初版作成
-   YYYY-MM-DD: <軽微な修正・更新内容を記述>
```

proposal-template.md

Tools Guide (docs/tools.md / docs/tools.project.md 作成ガイド)

本ガイドは、テンプレート共通および各プロジェクト固有の開発ツール・タスク（`.mise/` 設定や `.mise/scripts/`, `.mise/tools/`, `<root>/tools/` 配下、各種実行基盤）の使用方法やコマンド実行手順を定義する `docs/tools.md` および `docs/tools.project.md` を作成・編集する際の記述ルールおよびテンプレートを提供するものです。

概要と目的

開発ツール定義ファイルは、プロジェクトで利用する各種ツール（`mise` タスク、カスタムスクリプト、コンテナ実行環境等）の役割と具体的なコマンド実行手順をまとめ、AI エージェントおよび開発者が正しい手順でツールを実行できるように定義するためのドキュメントです。

記述ルール

1. **配置場所とファイル名の区分**:
   - `docs/tools.md`: テンプレート共通の開発ツール（アクション）を記述します（テンプレート同期対象）。`.mise/config.toml` から呼び出されるスクリプト・ツールは `.mise/scripts/` および `.mise/tools/` に配置し、`<root>/scripts/` と `<root>/tools/` は使用しません。
   - `docs/tools.project.md`: プロジェクト固有の開発ツール（アクション）を記述します（同期対象外）。
2. ID 形式の付与:
   - `docs/tools.md` では `<AIDEV-TOOL-ID(1)>` 形式（例: `AIDEV-TOOL-1`, `AIDEV-TOOL-2`）を使用します。
   - `docs/tools.project.md` では `<TOOL-ID(1)>` 形式（例: `TOOL-1`, `TOOL-2`）を使用します。
3. **章立てとセクション構成**:
   - ツール（アクション）ごとに `## <ID>: <action-name>` の見出し（章）を作成し、概要、実行コマンド、処理内容を記述します。
   - 一覧表（サマリーテーブル）は作成せず、各章に直接詳細を記述します。
4. **フロントマターの付与**:
   - `docs/AGENTS.md` に定める標準フロントマター（`name`, `description`, `timestamp`, `ai`）を付与します。

テンプレート

1. docs/tools.md テンプレート（共通ツール用）

````
---
name: tools.md
description: aidev-template 共通開発ツールおよび mise タスクの利用手順と仕様
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# Tools

本ドキュメントは、本テンプレート共通の開発ツールおよび `.mise/config.toml` に定義された mise タスクの役割、実行コマンド、処理内容をまとめたものです。

各ツール（アクション）には `<AIDEV-TOOL-ID(1)>` 形式の ID（`AIDEV-TOOL-1`, `AIDEV-TOOL-2`, ...）が付与されています。プロジェクト固有のツールについては `docs/tools.project.md`（ID 形式: `<TOOL-ID(1)>`）を参照してください。

## AIDEV-TOOL-1: <action-name>

<ツール・アクションの概要説明>

- **実行コマンド**:

  ```powershell
  mise run <action-name>
  ```

- **処理内容**:
  - <実行される処理の詳細・対象ファイル・副作用等>
````

2. docs/tools.project.md テンプレート（プロジェクト固有ツール用）

````
---
name: tools.project.md
description: 本プロジェクト固有の開発ツールおよび mise タスクの利用手順と仕様
timestamp: YYYY-MM-DD
ai: ai-coauthored
---

# Project Tools

本ドキュメントは、本プロジェクト固有の開発ツールおよびタスクの役割、実行コマンド、処理内容をまとめたものです。

各ツール（アクション）には `<TOOL-ID(1)>` 形式の ID（`TOOL-1`, `TOOL-2`, ...）が付与されています。テンプレート共通のツールについては `docs/tools.md`（ID 形式: `<AIDEV-TOOL-ID(1)>`）を参照してください。

## TOOL-1: <action-name>

<ツール・アクションの概要説明>

- **実行コマンド**:

  ```powershell
  mise run <action-name>
  ```

- **処理内容**:
  - <実行される処理の詳細・対象ファイル・副作用等>
````

backlog-roadmap-guide.md

Implementation Plan Creation Guide

本ガイドは、`<root>/docs/backlog/implementations/` 配下に集約して作成する実装計画の記述内容とMarkdownテンプレートを定義するものです（**非追跡対象**）。複数コンポーネント間でのファイル名衝突を防ぐため、ファイル名先頭にソフトウェアコンポーネントの AppID（例: `webui`）を付与します。

File Naming & Section Guidelines

1. 実装計画の種別とファイル名規則

すべての実装計画は Incremental Specification（`SPEC-I-ID`）**に対応して作成**され、Incremental Specification **と実装計画、タスク詳細ファイルは 1:1:1 の関係**となります。実装計画には以下の **2つの種別**がありますが、**いずれの種別でも実装計画ファイルは1つの Incremental Spec に対して必ず1つ**となります。

1. Incremental **型**:
    - 1つの `SPEC-I` に対し、計画分割を行わずに単一の計画手順で完結する場合。
2. Track **型（計画分割）**:
    - 1つの `SPEC-I` の作業規模が大きく、実装手順を段階化（Track）して進める場合。Track ごとにファイルを分割せず、**実装計画ファイルの中に Track 階層を設けて扱います。**

- **ファイル名規則**:
    - `<AppID>-<SPEC-I-ID>.md`（例: `webui-SPEC-I-001.md`）
- **作成省略と上位ステータス不変ルール**:
    - 契約に変更のない仕様変更・不具合修正・脆弱性修正・リファクタリング・パフォーマンス改善で、**変更対象が1ファイルかつ10行以下の場合は、実装計画を作成せずに作業**します（既存タスク詳細の実装メモに記録）。
    - 複数ファイルまたは11行以上で既存タスク詳細を `WIP` に戻して改修する場合も、**実装計画など上位ドキュメントのステータスは変更しません。**

2. フロントマターとステータス管理

- 標準項目（`name`, `description`, `timestamp`, `ai`）に加え、`status`, `review` キーを記述します。
- ステータス値（`status`）、レビューメタデータ（`review`）、ライフサイクル遷移、完了条件評価、および台帳同期ルールについては、document-status-guide.md を参照してください。

3. 必須セクション構成

1. **実装内容**: 機能仕様を実現するための技術的アプローチ、変更方針、影響範囲を記述します。

2. **完了条件**: 本実装計画が完了したとみなす具体的な受入・達成条件をチェックリスト形式（`- [ ]`）等で記述します。
3. **実装計画**: 実装の作業手順、進行フェーズ、依存関係を記述します。Track型の場合は本セクション内に Track 階層（`### Track 1: ...` 等）を設けて記述します。
4. **タスク一覧**: 対応するタスク詳細ファイル（`<component>/backlog/tasks/<SPEC-I-ID>.md`）へのリンクおよび進行状況を管理するテーブル。Track型の場合は Track 階層ごとのサブテーブルまたはセクションを設けて管理します。

4. 実装タスク一覧テーブルフォーマット

`TaskID` | `概要` | `SPEC` | `ステータス` | `依存` | `更新日時`

- **`TaskID` 列**: 各コンポーネントのタスク詳細ファイル（`<component>/backlog/tasks/<SPEC-I-ID>.md`）へのリンク。
- **`SPEC` 列**: 対応する増分機能仕様書（`specs/incremental/<SPEC-I-ID>.md`）へのリンク。
- **`ステータス` 列**: 参照先タスク等の最新進行ステータス（`DRAFT` | `REVIEW:DRAFT` | `TODO` | `WIP` | `REVIEW:WIP` | `DONE` | `CLOSE`）を記述。

Template

以下は実装計画（`webui-SPEC-I-001.md`）を作成する際の標準Markdownテンプレートです。

```
---
name: webui-SPEC-I-001.md
description: <増分機能仕様SPEC-I-001に基づく実装計画方針>
timestamp: YYYY-MM-DD
ai: ai-coauthored
status: DRAFT # DRAFT | REVIEW:DRAFT | TODO | WIP | REVIEW:WIP | DONE
review:
  result: "" # ACCEPTED | REJECTED
  timestamp: "" # YYYY-MM-DD
  reason: "" # REJECTED時の1行理由サマリー
---

# Implementation Plan: webui-SPEC-I-001

## 実装内容

<本機能仕様を実現するための実装アプローチ、モジュール追加・修正方針、影響範囲を記述します。>

## 完了条件

- [ ] <本コンポーネントにおける実装の完了条件1>
- [ ] <本コンポーネントにおける実装の完了条件2>

## 実装計画

1. データモデルおよびリポジトリ層の実装
2. サービスロジックおよびバリデーション処理の実装
3. API エンドポイントの実装と単体/結合テスト作成

## タスク一覧

| TaskID | 概要 | SPEC | ステータス | 依存 | 更新日時 |
| :--- | :--- | :--- | :--- | :--- | :--- |
| [`SPEC-I-001`](../../frontend/webui/backlog/tasks/SPEC-I-001.md) | 認証トークン生成・ログインAPI実装 |
[`SPEC-I-001`](../../frontend/webui/specs/incremental/SPEC-I-001.md) | WIP | NONE | 2026-09-03 |
```

Task Detail Document Creation Guide

本ガイドは、各ソフトウェアコンポーネント配下の `backlog/tasks/` ディレクトリにフラットに作成する実装タスク詳細ファイル（`<SPEC-I-ID>.md`）の記述内容とMarkdownテンプレートを定義するものです。

Rules & Section Guidelines

1. ディレクトリ配置・ID命名・作成ルール

- **1:1:1 の原則**: Incremental Specification と実装計画、タスク詳細ファイルは必ず 1:1:1 の関係 となります。タスク詳細ファイルは incremental specification に対して必ず 1 つ作成されます。
- **配置ディレクトリ**: `<component>/backlog/tasks/` 配下にフラットに配置・管理します（ディレクトリによる階層化は行いません）。
- **ファイル名 / ID**: 対応する増分仕様 ID と同一の **`<SPEC-I-ID>.md`**（例: `SPEC-I-001.md`）とします。
- **作成起点**: すべてのタスク詳細は Incremental Spec（`SPEC-I`）に紐づく実装計画（Plan）から作成されます（ISSUE から直接作成することは禁止）。
- **タスク詳細の種別**:
    - Incremental 型: 計画分割のない通常タスク詳細。単一の作業・受入条件・検証手順で完結します。
    - Track 型: 計画分割がある場合。**タスク詳細ファイルの中に Track 階層（`## Track 1: ...`, `## Track 2: ...` 等）を設けて管理** します。
- **ケース別運用ルール**:
    - **新規機能・契約変更**: 新規増分仕様に基づきタスク詳細を新規作成します。
    - **契約不変の軽微修正（1ファイル10行以下）**: 契約不変の仕様修正・不具合修正・脆弱性修正・リファクタリング・パフォーマンス改善で、1ファイル10行以下の場合は実装計画を作成せず直接作業し、完了後に本ファイルの「`## 実装メモ`」に作業内容・修正理由を追記・記録します。
    - **契約不変の通常修正（複数ファイルまたは11行以上）**: 本ファイルを **`WIP` ステータス** に戻し、作業内容を反映して実装します。**このとき実装計画など上位のステータスは変更しません。**

2. フロントマターとステータス管理

- 標準項目（`name`, `description`, `timestamp`, `ai`）に加え、`status`, `reason`, `review` キーを記述。
- ステータス値（`status`）、理由種別（`reason`）、レビューメタデータ（`review`）、ライフサイクル遷移、および台帳同期ルールについては、document-status-guide.md を参照してください。

3. 必須セクション構成

1. **概要**: タスク全体の概要。
2. **目的**: タスクの目的・達成ゴール。
3. **実装内容**: 作業項目全体に対する実装方針・内容概要。

4. **作業**: 個別の作業項目を `WI-<no>`（例: `WI-1`, `WI-2`）形式で定義。
5. **Acceptance Criteria**: 受入条件を `AC-<no>`（例: `AC-1`, `AC-2`）形式で定義し、チェックリスト項目（`- [ ]`）として記述。
6. **Verification**: 各 `AC-<no>` に対応する具体的な検証内容・手順・方法を記述。
7. **実装メモ**: 作業時のメモや、1ファイル10行以下の軽微な修正内容・履歴を記録（任意・追記用）。

Template

以下はタスク詳細ファイル（Incremental型: `SPEC-I-001.md`）を作成する際の標準Markdownテンプレートです。

```
---
name: SPEC-I-001.md
description: <タスクの1行概要>
timestamp: YYYY-MM-DD
ai: ai-coauthored
status: DRAFT # DRAFT | REVIEW:DRAFT | TODO | WIP | REVIEW:WIP | DONE | CLOSE
reason: feature # feature | defect | security | spec-change | investigation | performance | refactor |
ops
review:
  result: "" # ACCEPTED | REJECTED
  timestamp: "" # YYYY-MM-DD
  reason: "" # REJECTED時の1行理由サマリー
---

# Task: SPEC-I-001

## 概要

<タスク全体の概要を記述します。>

## 目的

<タスクを実施する目的や達成すべきゴールを記述します。>

## 実装内容

<すべての作業項目に対する全体的な実装方針・概要を記述します。>

## 作業

### WI-1: トークン生成ヘルパーモジュールの作成

- JWTトークンをエンコード・デコードするユーティリティ関数を実装する。

### WI-2: バリデーション処理の実装

- リクエストヘッダーのBearerトークンを検証する処理を追加する。

## Acceptance Criteria

### AC-1: トークン生成と検証の正常動作（WI-1, WI-2 に対応）
```

```
- [ ] 有効期限内のJWTトークンが正しく生成・検証されること。

### AC-2: 不正トークンのエラーハンドリング（WI-2 に対応）

- [ ] 期限切れまたは改ざんされたトークンで 401 Unauthorized が返却されること。

## Verification

### AC-1 の検証

-   単体テスト（`test_token_generate`）を実行し成功すること。
-   有効なトークンでのリクエスト検証が成功すること。

### AC-2 の検証

-   期限切れトークンによる検証テスト（`test_token_expired`）を実行し 401 が返ることを確認すること。

## 実装メモ

<!-- 1ファイル10行以下の軽微な修正内容や作業メモを必要に応じて記録 -->
```

Track型タスク詳細の構成例 (Track 階層を設ける場合)

Track型の場合、タスク詳細ファイル内に Track 階層を設けて作業・受入条件・検証をまとめます。

```
# Task: SPEC-I-001 (Track型)

## 概要・目的・実装内容
<全体概要を記述>

## Track 1: トークン生成基盤の実装

### 作業 (Track 1)
- WI-1: トークン生成ヘルパーモジュールの作成

### Acceptance Criteria (Track 1)
- [ ] AC-1: トークン生成の正常動作

### Verification (Track 1)
- AC-1 の検証: 単体テスト実行

## Track 2: 認証ミドルウェア連携

### 作業 (Track 2)
- WI-2: リクエストヘッダー検証処理の実装

### Acceptance Criteria (Track 2)
- [ ] AC-2: 不正トークンのエラーハンドリング

### Verification (Track 2)
- AC-2 の検証: 結合テスト実行
```

Issue Document Creation Guide

本ガイドは、各ソフトウェアコンポーネントの `backlog/issues/` 配下に作成する課題・不具合管理ファイル（`ISSUE-XXX.md`）の記述内容とMarkdownテンプレートを定義するものです。

Purpose & Section Guidelines

1. 目的と対象

- 仕様として定まっていない課題、懸念事項、およびリリース後に発生した不具合報告を記録・追跡管理します。

2. 命名規則・フロントマター

- **ファイル名**: `ISSUE-XXX.md` の形式で連番により管理（例: `ISSUE-001.md`）。
- ステータス値（`status`）、理由種別（`reason`）、レビューメタデータ（`review`）、ライフサイクル遷移、および台帳同期ルールについては、document-status-guide.md を参照してください。

3. セクション構成

1. **概要**: 課題・障害の1行～簡潔な要約。
2. **背景・障害内容 (Background & Symptoms)**: 発生した症状、再現手順、発生環境、または検討が必要な背景。
3. **原因分析・懸念事項 (Root Cause & Concerns)**: 発生原因の推測・特定結果、または懸念事項。
4. **対応方針・作業内容 (Action Plan)**: 解決のための修正内容や作業項目。
5. **検証結果 (Verification)**: 修正確認・解決の検証項目（`- [ ]`）。

Code Investigation & Resolution Workflow

課題や問題の調査・確認などでコードに変更が生じる場合の運用手順を以下のように定めます。

1. 作業ブランチの作成

- 調査や確認のためにコードを変更する必要がある場合、`<ISSUE-ID>` ブランチ（例: `ISSUE-001`）を作成して作業を行います。

2. 作業内容と結果の記録

- 作業ブランチで実施した作業内容とその結果は、該当する ISSUE ファイル（`ISSUE-XXX.md`）の「対応方針・作業内容」および「検証結果」セクションに詳細を記録します。

3. コード反映と実装フロー

- 調査・確認の結果を本番コードに反映させる場合は、**増分機能仕様書（SPEC-I）を起点として通常の実装フロー**に基づいて行います。
- **実装計画・詳細タスクの省略**: この際、実装計画（`backlog/implementations/`）や詳細タスク（`backlog/tasks/`）の作成は不要です。
- **マージとコミット規約**: 作業ブランチ（`<ISSUE-ID>`）からマージしても構いませんが、コミットメッセージは必ずプロジェクトのコミット規約（`docs/guide/commit-message.md`）に従ってください。

4. ブランチの削除

- コードへの反映が完了した、または調査完了により不要になった作業ブランチは速やかに削除します。

Template

以下は課題・不具合管理ファイル（`ISSUE-001.md`）を作成する際の標準Markdownテンプレートです。

```
---
name: ISSUE-001.md
description: <課題または障害の1行概要>
timestamp: YYYY-MM-DD
ai: ai-coauthored
status: DRAFT # DRAFT | REVIEW:DRAFT | TODO | WIP | REVIEW:WIP | DONE | CLOSE
reason: defect # feature | defect | security | spec-change | investigation | performance | refactor |
ops
review:
  result: "" # ACCEPTED | REJECTED
  timestamp: "" # YYYY-MM-DD
  reason: "" # REJECTED時の1行理由サマリー
---

# Issue: ISSUE-001 - <課題/不具合件名>

## 概要

<課題や不具合報告の全体概要を記述します。>

## 背景・障害内容

-   **現象**: <発生した現象や問題点を記述>
-   **再現手順**:
    1. ログイン画面にアクセスする
    2. 特殊文字を含むパスワードを入力して送信する
    3. 500 Internal Server Error が発生する

## 原因分析・懸念事項

<調査により判明した根本原因、または仕様未決定によるリスク・懸念事項を記述します。>

## 対応方針・作業内容

- [ ] パスワード入力値のサニタイズ処理を追加する。
- [ ] 例外発生時の適切なエラーメッセージ返却処理を実装する。
- [ ] 作業ブランチ: `ISSUE-001`

## 検証結果
```

- [ ] 特殊文字を含むパスワードで正常にログイン処理が完了することを確認。
- [ ] 反映完了後、作業ブランチ `ISSUE-001` を削除済みであることを確認。

TODO Document Creation Guide

本ガイドは、各ソフトウェアコンポーネントの `backlog/todos/` 配下に作成する TODO 管理ファイル（`TODO-XXX.md`）の記述内容とMarkdownテンプレートを定義するものです。

Purpose & Rule Guidelines

1. 目的と対象

- SPEC（機能仕様）や ISSUE（課題・不具合）に該当しないが、将来的に検討・実施すべき簡易な作業メモ、改善アイデア、備忘録等を記録します。

2. ステータス管理と昇格ルール

- TODO のステータス値（`status`）、レビューメタデータ（`review`）、ライフサイクル遷移、SPEC/ISSUE への昇格（Promotion）ルール、および台帳同期については、document-status-guide.md を参照してください。

Template

以下は TODO 管理ファイル（`TODO-001.md`）を作成する際の標準Markdownテンプレートです。

```
---
name: TODO-001.md
description: <TODO項目の1行概要>
timestamp: YYYY-MM-DD
ai: ai-coauthored
status: DRAFT # DRAFT | REVIEW:DRAFT | TODO | CLOSE
review:
  result: "" # ACCEPTED | REJECTED
  timestamp: "" # YYYY-MM-DD
  reason: "" # REJECTED時の1行理由サマリー
---

# TODO: TODO-001 - <TODO件名>

## 概要

<将来的に検討・実施したい作業メモやアイデアの概要を記述します。>

## 詳細・検討事項

<具体的なメモ内容、検討すべきポイント、関連コード箇所等を記述します。>

## 昇格ステータス (Promotion Status)

-   **昇格先**: 未定 / [`SPEC-003`](../specs/SPEC-003.md) / [`ISSUE-002`](../issues/ISSUE-002.md)
-   **メモ**: SPEC-003 策定時に本項目の要件を取り込むため昇格。昇格完了時に status を CLOSE に更新する。
```