Vision Ark

Vision Ark System Design

要件定義 (Requirement Definition)

はじめに (Introduction)

システム名称と目的 (System Name and Purpose)

システム名称

Vision Ark

背景 (Background)

開発者（ユーザー）は、研究・開発・自己研鑽・生活事務など性質の異なる複数プロジェクトを並列で遂行している。
このとき発生する主なコストは、(1) 状態の保持、(2) 文脈切替、(3) 情報検索、(4) 外部ツールとの整合維持である。これらを人間のワーキングメモリに依存して運用すると、認知資源が圧迫され、実行品質と継続性が低下する。

目的 (Purpose)

本システムは、ユーザーの活動における「状態（State）」「事実（Facts）」「経験ログ（Episodes）」および「タスク／負荷（LBS）」を永続化し、Hub/Spoke型の対話エージェントがそれらを参照・更新できるインターフェースを提供する。
単なるToDo管理ではなく、ユーザーの意思決定と実行を支える **Exocortex（外部化された思考基盤）**として機能することを目的とする。

開発哲学 (Development Philosophy)

本システムは、以下の4つの哲学に基づいて設計・実装される。

1. **Explicit Control (明示的制御)**

重要な状態更新（タスク生成、期日変更、負荷スコア変更、Fact/State確定、外部同期の書き込み）は、必ずユーザー承認または事前に許可されたポリシーに従って実行される。
例外として、ログ保存・インデックス更新・重複除去等の「安全な自動処理」は無承認で実行してよい。

2. Decentralized Execution (自律分散実行)

中央集権的な巨大な脳（Single Huge Context）を作らない。各プロジェクト（Spoke）は、それぞれ独立したファイルシステム、参照資料、Personaを持つ「閉じた宇宙」として動作する。サービスは疎結合とし、相互依存を最小化する。

3. tate over Memory (記憶より状態)

チャットログは揮発的であり正本ではない。タスク・負荷情報は **LBS Microservice**、ユーザー文脈は **Knowledge Core Service** という独立したSSOT（Single Source of Truth）サービスに分離し、API経由でアクセスする。

4. Replaceability（差し替え可能性）

RAG基盤、LLMベンダ、UI、同期先（ToDo/Calendar）は差し替え可能とし、コアデータ（LBS/Knowledge Core）を中心に周辺を交換できる構造を維持する。

想定ユーザーとユースケース (Target User & Use Cases)

想定ユーザー

- 属性：エンジニア／研究者（複数プロジェクト並行、CLI/スクリプト志向）
- 課題：状態管理と文脈切替のコスト、タスク・ログ・資料の散逸

主要ユースケース (Key Use Cases)

1. **R&D / Learning**：論文・仕様書・コードを参照しながら、意思決定ログと成果物を蓄積
2. **Project / Task Orchestration**：LBSに基づく負荷分散、計画修正、外部ツールへの同期
3. **Life Admin / Personal Ops**：生活事務の即時入力、後でInboxでトリアージし、SSOTに反映
4. **Long-term Strategy**：LifeVision等の参照資産と、Knowledge CoreのFacts/Statesに基づく継続運用

機能要件

長期コンテキスト管理 (Long-term Context Management)

- ユーザーとLLMの間の対話ログは、一定期間または一定量を超えた場合に自動的にローテーションされ、重要な情報のみを抽出した要約として保存されること。
- この要約プロセスは、対話の主題やゴールを維持するように最適化されていること。

外部知識参照（RAG: Retrieval-Augmented Generation）

- 過去の要約コンテキストやプロジェクト関連ドキュメントは、ベクトルデータベースに格納されること。
- LLMへのプロンプト生成時、現在の対話内容に基づき、このデータベースから関連性の高い情報を自動的に検索・取得し、プロンプトに含めること。
- 検索される情報の関連性スコアを調整可能とし、不必要な情報付加を抑制できること。

記憶の可視化と編集

- システムが長期記憶として保持している要約や参照ドキュメントの一覧をユーザーが確認できること。
- ユーザーは、誤った、または不要となった記憶を手動で編集・削除できるインターフェースを提供すること。

アーキテクチャ連携 (Functional Requirements: Architectural Integration)

API-First Contract (APIファースト契約) 各マイクロサービス（Core, LBS, Knowledge Core）は、独立してデプロイ・スケール可能であると同時に、明確に定義されたインターフェース契約に基づいて協調する。

- **Explicit API Definition**
  - 各サービスは、HubおよびSpokeエージェントがSSOT（Single Source of Truth）にアクセスするためのRESTful API（またはgRPC）を公開し、OpenAPI Specification (Swagger) 等で仕様を厳格に定義する。
- **Version Control Strategy**
  - エージェントの自律動作を妨げないため、バックエンドAPIはバージョン管理（例: `/v1/tasks`）を行い、破壊的変更（Breaking Changes）を回避する。
- **Health & Status Disclosure**
  - 各サービスは自身の稼働状況（Health Check）および同期ラグ（Sync Status）をCore Systemへ通知するエンドポイントを持ち、UI上のステータスバーでユーザーが「現在のシステムの状態」を把握できるようにする。

外部連携と整合性 (External Integration and Consistency)

本システムは、独自のLBSデータベース（Master）を持ちつつ、ユーザーが日常的に使用するMicrosoft

ToDoやOutlook Calendarを「表示・操作端末（Client）」として利用する。データの整合性を保つため、厳格な同期ロジックと方向性を定義する。両サービスとも Microsoft Graph API を介して制御するため、認証基盤を統一できる利点がある。

Master-Slave Architecture

- **Master (SSOT - Single Source of Truth):** LBS Service。タスクの定義、負荷スコア、コンテキストの管理主体。競合解決は LBS Service の仕様として定義し、Hubは方針提案に留める。
- **Client:** Microsoft To Do / Calendar。ユーザーが日常的にタスクを確認・完了・簡易登録するためのインターフェース。

Sync Scope

- **Microsoft To Do:** 「生活事務（Life Admin）」や「雑務」の管理。LBS上のタスクリストと双方向同期する。
- **Outlook Calendar:** 「Time Blocking」。LBS上で負荷が高いタスクや時間が固定されたイベントをカレンダーにブロックとして確保する。

Authentication

- Microsoft Graph APIを使用し、OAuth 2.0フローを通じてユーザーのデータにアクセスする。アクセストークンおよびリフレッシュトークンは暗号化して厳格に管理される。

Command Line Interface (CLI) & Shortcuts

システムは、GUI操作を補完し、Power Userの操作効率を最大化するためのコマンドラインインターフェースを提供する。これは必須の操作手段ではなく、**「ショートカット（Alias）」**として位置づけられる。

Hybrid Processing (ハイブリッド処理)

- コマンドは、**Client-Side Commands**（画面遷移など即時性を要するもの）と、**Server-Side Commands**（データの変更・計算を要するもの）に分類され、適切なレイヤーで処理される。

Syntax (構文)

- すべてのコマンドは `/` (スラッシュ) で開始される。
- 形式: `/{command} [arguments]`

Safety (安全性)

- データの削除や強制リセットを伴うコマンド（`/reset` 等）は、実行前に確認ダイアログを要求する。
- コマンド入力中はLLMへの送信をブロックし、意図しないトークン消費を防ぐ。

UI/UXと操作性 (User Interface & Operability)

Chat-Centric IDE体験:

- 本システムは、対話型AIの柔軟性（Chat）と、プロジェクト管理ツールの堅牢性（GUI）を融合させた、「Chat-Centric IDE (Integrated Development Environment)」としてデザインされること。
- ユーザーはコンテキストスイッチを最小限に抑えつつ、マウスとキーボードを行き来しながら高速に意思決定を行うことができること。

重要UI要素の集約:

- システムの主要な操作と承認は、以下の3つの要素に集約され、ユーザーの認知負荷を最小限に抑えること。
    - **Chat（Hub/Spoke）:** エージェントとの対話、記録、指示の実行。
    - **Inbox（承認ゲート）:** Spokeからの報告や提案をレビュー・承認する、唯一の情報流入制御点。
    - **Dashboard（LBS可視化・編集）:** LBSの数値（負荷スコア、トレンド）を可視化し、計画を直接操作（修正）する場。

Hybrid Interaction Model (ハイブリッド操作モデル)

本システムは、マウス操作を主とするGUIと、キーボード操作を主とするCLI（Command Line Interface）をシームレスに統合し、ユーザーの習熟度に応じた最高速度の操作性を提供する。

- **Command Line Interface (CLI)**
    - Power Userの認知負荷と操作手数を削減するため、チャット入力欄において `/` (スラッシュ) で始まるコマンド操作をサポートすることは必須要件とする。
    - 特に、コンテキスト切り替え（Navigation）や強制ログローテーション（Archive）などの頻出操作は、キーボードのみで完結しなければならない。
- **Keyboard Shortcuts**
    - コマンド以外にも、一般的なIDE（VS Code等）に準拠したショートカットキー（例: `Ctrl+K` で検索、`Ctrl+Tab` でタブ切り替え）を実装し、開発者が違和感なく操作できる環境を提供する。

非機能要件 (Non-Functional Requirements)

システムの品質と運用性を保証するため、以下の非機能要件を定義する。

パフォーマンス (Performance)

- 応答時間
    - **Hub/Spoke** チャット応答
        - 通常応答：2〜5 秒以内

        - RAG 参照を含む応答：5〜10 秒以内
    - **Dashboard**（**LBS** 可視化）
        - 初回ロード：2 秒以内
        - 再描画：1 秒以内
    - **Inbox** 処理
        - 承認操作後の反映：1 秒以内（UI レベル）
        - LBS への反映：5 秒以内（非同期）
- バックグラウンド同期（**Sync Worker**）
    - **ToDo / Calendar** 同期周期
        - 通常：5 分間隔
        - 手動トリガ時：即時実行
    - 外部差分取得（**Delta Query**）
        - 1 回の同期で最大 200 件まで処理
        - 1 件あたりの処理時間：100ms 以下

スケーラビリティ **(Scalability)**

- データ量
    - プロジェクトドキュメント、ログ、タスクデータが増加しても、システムのパフォーマンスが維持されるよう、データベースおよびベクトルストアの設計を考慮すること。特に長期的なログ保存に耐えうるストレージ戦略を確立する。

信頼性・可用性 **(Reliability & Availability)**

- 稼働率
    - 個人向けシステムとして 99% 稼働率を目標とする
    - （計画停止を除く）
- フェイルセーフ
    - LBS / Knowledge Core が一時的にダウンしても
    - Hub/Spoke のチャットは degraded mode で継続
    - 外部同期（ToDo/Calendar）は
    - 再試行キュー（Retry Queue）に蓄積し、復旧後に処理
- 冗長性
    - LBS ServiceとKnowledge Core Serviceは、データの永続性と可用性を確保するため、適切なバックアップおよび冗長化戦略を持つこと。
- エラーハンドリング
    - 外部連携（Microsoft Graph API）を含む全ての外部依存サービスとの通信エラー発生時、ユーザーに分かりやすい形でエラーを通知し、適切なリトライロジックを実装すること。
- 再試行ポリシー
    - Sync Worker は以下の指数バックオフを採用
        - 1 回目：5 秒
        - 2 回目：30 秒
        - 3 回目：5 分
        - 4 回目：30 分
    - 24 時間失敗が続いた場合は Inbox に警告を送信

運用・保守性 (Operability & Maintainability)

- ロギング
  - システムの動作、特にデータの変更や外部との同期に関するログを詳細に記録し、問題発生時のトレーサビリティを確保すること。
  - Hub / LBS / KC / Sync Worker は 構造化ログ（**JSON**）を出力
  - ログレベル
    - INFO：通常動作
    - WARN：外部同期の遅延
    - ERROR：API 失敗、DB エラー
    - CRITICAL：データ破損の可能性
- デプロイメント
  - コンテナ技術（例：Docker）を活用し、環境構築およびデプロイメントを自動化・簡素化すること。
- 設定管理
  - LLMのモデル名、APIキー、同期設定などの構成情報は、環境変数または集中管理された設定ファイルを通じて外部から容易に変更できること。
  - すべての設定値は `.env, config.yaml` に集約
  - 設定変更は再デプロイなしで反映可能（ホットリロード）

セキュリティ (Security)

- 認証
  - 認証は、Microsoft Graph API連携のためのOAuth 2.0フローを基本とし、他の機密情報へのアクセスにも適切な認可メカニズムを使用すること。
- データ保護
  - ユーザーの機密データ（特にKnowledge CoreのFacts/States）は、保存時および転送時に業界標準の暗号化プロトコル（例：AES-256、TLS 1.3）を使用して保護すること。
- アクセス制御
  - 各Spoke（プロジェクト）のデータへのアクセスは、ユーザー本人およびエージェントの処理コンテキストに厳格に限定されること。

Authentication & Authorization (認証と認可)

本システムはマルチユーザー・マルチテナント環境で動作するため、ゼロトラストを前提とした厳格なアクセス制御を実装する。

- **End-User Authentication (OAuth 2.0)**
  - ユーザー認証には独自のID/Pass管理を行わず、標準的な **OAuth 2.0 Authorization Code Flow** (Google Account または Microsoft Entra ID) を採用する。
  - これにより、MFA（多要素認証）の実装コストを外部委譲し、セキュリティリスクを最小化する。
- **Service-to-Service Authentication**
  - マイクロサービス間（Core ⇔ LBS ⇔ Knowledge Core）の通信は、プライベートネットワー

ク内であっても暗黙的に信頼しない。
    - 全てのリクエストヘッダーに **`X-Service-Key`** (または署名付きJWT) の付与を必須とし、各サービス側で呼び出し元の正当性を検証する。
- **Role-Based Access Control (RBAC)**
    - Hubエージェント（Write権限）とSpokeエージェント（Read-Only権限）の権限分離は、アプリケーションロジックだけでなく、APIレベルまたはDB接続ユーザーレベルでも強制されるべきである。
    - ユーザーインターフェース上の「Home (参照用)」と「Hub (対話用)」の分離に伴い、権限も明確に分離する。
    - Hubエージェントからのリクエストのみが `WRITE` 権限を持ち、Dashboard表示などの参照系リクエストは `READ-ONLY` トークンで実行されること。

基本設計 (Basic Design)

システムアーキテクチャ (System Architecture)

全体構成図 (Hub-Spoke Model)

本システムは、中央のHubが全体最適（Orchestration）を担いつつ、各Spokeが自律的に実行（Execution）と相互連携を行う「階層的連邦モデル」を採用する。ユーザーはUIを通じて任意のAgent（Node）に直接アクセスし、Agent間はツール呼び出しを通じて連携する。

```
graph TD
  subgraph UI_Layer [Antigravity Shell (UI)]
    Chat[Chat Interface]
    Dash[LBS Dashboard]
    InboxUI[Inbox View]
  end

  subgraph Agent_Runtime [Agent Network]
    Hub[👑 Hub Agent (PM)]
    SpokeA[💎 Spoke: Research]
    SpokeB[💰 Spoke: Finance]

    %% Spoke-to-Spoke (Federated)
    SpokeA <-->|Direct Dialogue| SpokeB

    %% Hub-Spoke (Hierarchical)
    Hub <-->|Directive / Report| SpokeA
    Hub <-->|Directive / Report| SpokeB
  end

  subgraph Coordination [Coordination Buffer]
    Inbox[(Inbox / Request Queue)]
  end

  subgraph OS_Sovereignty [Core System Storage (SSOT)]
    CoreDB[(Core DB / Internal RAG)]
  end

  subgraph Microservices [External Services]
    LBS[LBS Service (Calculator)]
    KC[Knowledge Core (Personalization)]
  end
```

```
%% UI Interaction (Manual Selection)
Chat -->|Select Node| Hub
Chat -->|Select Node| SpokeA
Chat -->|Select Node| SpokeB

InboxUI <-->|Review & Approve| Inbox
Dash <-->|Visualize| LBS

%% Inbox Flow (Asynchronous)
SpokeA -- Tool Call: Push Request --> Inbox
SpokeB -- Tool Call: Push Request --> Inbox
Inbox -- Batch Process --> Hub

%% Service Access: LBS
Hub <-->|Read / Write (Exclusive)| LBS
SpokeA -.->|Read Only| LBS
SpokeB -.->|Read Only| LBS

%% Service Access: Knowledge Core (Personalization)
Hub & SpokeA & SpokeB -.->|Fetch User Context| KC

%% Data Access: Internal RAG (OS managed files)
Hub & SpokeA & SpokeB <-->|Reference Documents| CoreDB
```

サービス責務定義 (Service Responsibilities)

サービス名称	Layer	責務
Core System (Hub)	UI / Orchestration	ユーザー認証、エージェント管理、UI（Chat/Dashboard/Inbox）、サービス間認証（**署名付き認証**）を担当。LBS/KC ServiceへのAPIゲートウェイとして機能し、**LBS/KCへの書き込み権限を持つ唯一のクライアント**となる。
LBS Microservice	SSOT / Core Logic	負荷計算、タスク正本（マスターデータ）、スケジューリングロジックを担当。Hubからの承認済みActionのみを受け付け、DBへの書き込み権限を持つ（**独立サービス**）。

Knowledge Core (KC)	SSOT / Core Logic	ユーザー文脈（Facts/States/Episodes）の長期記憶と管理を担当。ユーザーとHub/Spokeのやり取りから必要な情報を記憶し，Hub/Spokeに最適な追加情報を提供する（Phase 3で統合）。
Spoke Agents	Execution	各プロジェクト固有の判断・分解・生成を担当。LBS/KCへのデータ変更提案を **Inbox** に送信する。

データ管理とマルチテナント (DB-Centric Design & Multi-Tenancy)

項目	旧管理方法（Old）	新管理方法（New: v2.2）
データ基盤	**JSON/Markdown**ファイル（ファイルシステム）	**PostgreSQL** による **DB**中心設計
プロジェクト管理 **(Spoke)**	`spokes/` ディレクトリ	``` **NODES** ``` テーブルのレコードとして管理
チャットログ	`chat.log` **(**テキストファイル**)**	``` **CHAT_SESSIONS** ``` テーブルで構造化管理（**JSONB**など）
ユーザー管理	シングルユーザー	マルチユーザー・マルチテナント対応。全ての主要テーブルに **`user_id`** を付与し、論理的なデータ隔離を担保。
サービス間認証	なし	署名付き認証 **(**``` **X-Service-Key** ```**)** を導入し、サービス間のセキュアな連携を実現。

レイヤー定義 (Layer Definitions)

各レイヤーは「責任の単一性（Single Responsibility Principle）」に基づき設計されており、上位レイヤーは下位レイヤーの詳細に関知せず、下位レイヤーは上位レイヤーの決定に従う。

Layer 1: Global Reference (Global Assets)

- 役割: システム全体の「憲法」および「羅針盤」。AIが判断に迷った際の最終的な拠り所となる「静的な真実（Ground Truth）」を提供する。
- 構成要素:
    - LifeVision.pdf, NTTvision.pdf: ユーザーの長期的目標や価値観を定義したドキュメント。
    - Global_System_Prompt.md: 全エージェントに共通する振る舞い（トーン＆マナー、禁止事項、出力形式）の定義ファイル。
    - Shared_Glossary.json: プロジェクト横断で使用される専門用語や略語の定義集。
- 特性: 完全な読み取り専用 **(Read-only)**。
    - HubおよびすべてのSpokeは、起動時（コンテキスト生成時）にこのレイヤーをシステムプロンプトの一部として読み込む。
    - AI自身がこのレイヤーを変更することは許可されない。変更にはユーザーによる明示的なファイル更新（Gitコミットやファイル上書き）が必要であり、これによりAIによる「目標の勝手な書き換え」や「ルールの自己都合解釈」を防ぐ。

Layer 2: The HUB (Orchestration Layer)

- 役割: リソース（時間・認知・労力）の最適配分と、プロジェクト間の競合調停を行う「管制塔」。
- 責任:
    - **LBS (Load Balancing System)** 管理: 全タスクの負荷スコア（Load Score）を監視・集計する。特定の日に負荷が集中した場合、優先順位の低いタスクを別日へ移動させるなどの調整案を提示し、システム全体のオーバーフローを防ぐ。
    - **Inbox**処理 **(Information Traffic Control):** Spokeから非同期に送信されてくる報告（Inbox Buffer）を処理する。ノイズを除去し、重要な更新のみをコンテキストに取り込み、LBSデータベースへ反映させる。
    - **Spoke**管理 **(Lifecycle Management):** 新規プロジェクト発生時のSpoke生成（ディレクトリ作成、初期設定）や、プロジェクト完了時のSpokeアーカイブ（ログ保存、コンテキスト破棄）を行う。
- 非干渉の原則 **(Non-Interference):**
    - Hubは、Spoke内部で行われている「具体的な作業内容」（例：Pythonコードのデバッグ、論文のパラグラフ推敲、個別のメール文面作成）には一切干渉しない。
    - Hubが扱うのは「メタ情報」（進捗率、障害の有無、完了予定日、次のマイルストーン）のみであり、これによりHubのコンテキスト消費を最小限に抑え、長期記憶の維持を可能にする。

Layer 3: The SPOKE (Execution Layer)

- 役割: 具体的タスクの実行と記録を行う「現場」。
- 構成要素:

- **Custom Persona:** プロジェクトの性質に応じた特化型プロンプト。
  - 例（Research）:「論理的整合性を最重視し、批判的思考を持つ研究パートナー」
  - 例（Life Admin）:「事務手続きを効率的に処理する、簡潔な秘書」
- **Local Refs:** そのプロジェクト固有の参考文献。Hubには共有されない膨大なPDF（論文）、データセット、コードベースなどが含まれる。RAG（Retrieval-Augmented Generation）の検索対象となる。
- **Artifacts:** 生成された成果物（ドラフト、コード、図表）。

データの流れと制御フロー (Data Flow & Control Logic)

制御フロー（最小要件）

1. **Spoke → Inbox**：各判断（タスク案、優先度案、期日案）を投入
2. **Inbox → Hub**：集計単位でまとめて送信（例：手動トリガ、一定件数到達、一定時間経過）
3. **Hub**（全体最適）：全体負荷・優先度・競合を評価し「調整案」を生成
4. **Hub → Spoke**：調整結果を通知し、Spoke側が必要に応じて再分解・再調整

外部連携アーキテクチャ (External Integration Architecture)

Synchronization Architecture

同期処理は、Core System (Hub) ではなく、データの所有者である **LBS Service** 内部のバックグラウンドワーカー（Sync Worker）によって非同期に実行される。

```
graph TD
  subgraph LBS_Microservice
    DB[(LBS Master DB)]
    API[LBS API]
    Worker[🔄 Sync Worker]
  end

  subgraph External_Cloud
    GraphAPI[Microsoft Graph API]
  end

  subgraph Clients
    ToDo[MS To Do App]
```

```
    Outlook[Outlook Calendar]
  end

  %% Flow
  API -->|Write Task| DB
  Worker -->|Polling / Trigger| DB
  Worker <-->|HTTPS / JSON| GraphAPI
  GraphAPI <-->|Sync| ToDo
  GraphAPI <-->|Sync| Outlook
```

- Sync Worker: 定期的（例: 5分ごと）またはイベントトリガー（Hubからの変更通知）により起動し、LBS DBと外部クラウドの差分を解消する。

Data Flow (Sync Cycle)

同期エージェント（Sync Agent）は、以下のサイクルで動作する。

Cycle 1: Outbound Sync (LBS -> External) LBSでタスクが作成・更新された場合、外部サービスへ反映する。

```
sequenceDiagram
  participant Spoke as Spoke/User
  participant DB as LBS Master DB
  participant Agent as Sync Agent
  participant Ext as MS ToDo / Outlook

  Spoke->>DB: タスク作成 / 更新
  Agent->>DB: 変更検知 (Polling/Trigger)
  Agent->>Ext: Create / Update Task/Event (Graph API)
  Ext-->>Agent: Return External ID
  Agent->>DB: Save External ID
```

Cycle 2: Inbound Sync (External -> LBS) 外部アプリでタスクが完了または作成された場合、LBSへ取り込む。

```
sequenceDiagram
  participant Ext as MS ToDo / Outlook
  participant Agent as Sync Agent
  participant DB as LBS Master DB
  participant Inbox as Core Inbox

  Ext->>Ext: スマホでタスク完了
  Agent->>Ext: Delta Query (差分取得)
```

```
Ext-->>Agent: Completed Task List
Agent->>DB: Update Status = DONE
DB->>Inbox: (Optional) Notify Hub via Inbox
```

Command Routing Architecture

チャット入力欄に入力されたテキストが / で始まる場合、Frontendのパーサーがこれを検知し、以下のフローで処理を分岐させる。

```
graph TD
  Input[User Input] -->|Check Prefix| IsCommand{Starts with '/'?}

  IsCommand -- No --> LLM[Send to LLM API]
  IsCommand -- Yes --> Parser[Frontend Command Parser]

  Parser -->|Identify Type| TypeCheck{Command Type}

  TypeCheck -- Client Action --> ClientLogic[Execute Frontend Logic]
  TypeCheck -- Server Action --> ServerAPI[POST /api/commands]

  subgraph Client_Side
    ClientLogic --> SwitchTab[Switch Node Tab]
    ClientLogic --> ClearUI[Clear Chat Window]
  end

  subgraph Server_Side
    ServerAPI --> Archive[Log Rotation]
    ServerAPI --> Report[Inbox Push]
    ServerAPI --> Task[LBS Quick Add]
  end
```

- Frontend Parser: 正規表現を用いてコマンド名と引数を抽出する。定義済みリストにないコマンドは、ローカルでヘルプを表示する。
- Backend Endpoint: POST /api/commands/execute
    - サーバーサイドコマンドの場合、ここで認証と権限チェック（Hubのみ実行可能など）を行った上で、各サービス（LBS, DB）を操作する。
    -

UI設計と画面レイアウト (UI Design & Screen Layout)

- 全体原則:
    - UIは「操作と承認」の場であり、SSOT（LBS Microservice / Knowledge Core）は保持しない。

    - 画面全体は「常駐するナビゲーション」と「可変のワークスペース」に明確に分離される。
- サイドバー構成 (Navigation Menu)
    - 常時表示される左端のナビゲーションメニューにより、ユーザーは以下の主要モードへ直感的にモードを切り替えることができる。
        1. 🏠 **Home / Hub Chat:** 統合司令塔（Hub）とのメインチャット画面。
        2. 📊 **LBS Dashboard:** LBS専用のアナリティクスビュー。
        3. 📥 **Inbox (Notification Center):** Spokeからの報告や外部同期タスクが集まる承認ゲート。
        4. 💎 **Projects (Spokes Explorer):** プロジェクト一覧（ツリー形式）と、各Spokeへの遷移。
- ワークスペース構成 (Tabbed Interface)
    - チャット画面やダッシュボードをマルチタブ形式で複数開くことができる。
    - タブをドラッグして画面を左右（または上下）に**分割表示（Split View）**可能とし、並行作業を支援する。

詳細設計 (Detailed Design)

データ・ディレクトリ構造 (Data & Directory Structure)

ルートディレクトリ構成 (Root Directory Structure)

本システムは、機能（Service）と責任（Responsibility）に基づき、マイクロサービス構成としてディレクトリを分割する。各サービスは独立したDockerコンテナとして動作し、データはファイルシステムではなくデータベース（PostgreSQL）に永続化される。

```
Vision Ark/
├── global_assets/         # [Shared Config] 全サービス共通の定義ファイル
│   │   # システム全体の振る舞いを規定する静的な設定。
│   │   # DB管理するまでもない定数や、初期セットアップ用のシードデータを含む。
│   ├── system_prompt_global.md # 全エージェント共通のベースプロンプト
│   └── glossary.json      # ドメイン用語定義
│
├── core/                  # [Core System] Hub & Agent Orchestrator
│   │   # ユーザー対話、エージェント実行、全体統括を行うメインシステム。
│   ├── backend/           # Python (FastAPI + LangGraph)
│   │   ├── app/
│   │   │   ├── agents/    # HubおよびSpokeエージェントの思考ロジック
│   │   │   ├── api/       # Frontend向けREST APIエンドポイント
│   │   │   ├── core/      # 設定、認証、共通ユーティリティ
│   │   │   ├── models/    # Core DBスキーマ (Nodes, ChatSession, Inbox)
│   │   │   └── services/  # 外部サービス連携 (LBS Client) やRAGロジック
│   │   └── migrations/    # AlembicによるDBマイグレーションスクリプト
│   │
│   └── frontend/          # TypeScript (Next.js App Router)
│       ├── components/    # UI部品 (Chat, Inbox, Dashboard)
│       └── app/           # ページルーティング
│
├── infra/                 # [Infrastructure] 構成管理
│   ├── docker-compose.yml  # ローカル開発用の一括起動定義
│   ├── postgres/          # DB初期化スクリプト (init.sql)
│   └── env_samples/       # 環境変数テンプレート (.env.example)
│
└── docs/                  # ドキュメント
    ├── requirements/      # 要件定義書
    └── architecture/      # 設計図、シーケンス図
```

データベース設計 (Database Design)

Entity Relationship Diagram (ERD)

```
erDiagram
    %% ========================================
    %% 1. Identity & Config
    %% ========================================
    USERS ||--o{ SERVICE_CONNECTIONS : "configures"
    USERS ||--o{ NODES : "owns"

    USERS {
        uuid id PK
        string email UK
        boolean is_active
        datetime created_at
    }

    SERVICE_CONNECTIONS {
        uuid id PK
        uuid user_id FK
        string service_type "LBS, KNOWLEDGE_CORE"
        string endpoint_url
        string api_key_encrypted
    }

    %% ========================================
    %% 2. Agent Nodes (Contexts)
    %% ========================================
    NODES ||--o{ AGENT_PROFILES : "defined_by"
    NODES ||--o{ CHAT_SESSIONS : "has_history"
    NODES ||--o{ INBOX_ITEMS : "generates_request"
    NODES ||--o{ UPLOADED_FILES : "owns_ref_data"

    NODES {
        uuid id PK "Context ID"
        uuid user_id FK
        string name "Slug"
        string display_name
        string node_type "HUB, SPOKE"
        string lbs_access_level "READ_ONLY, WRITE"
```

```
    boolean is_archived
}

AGENT_PROFILES {
    uuid id PK
    uuid node_id FK
    int version
    text system_prompt
    boolean is_active
    datetime created_at
}

%% ========================================
%% 3. Communication
%% ========================================
CHAT_SESSIONS ||--o{ CHAT_MESSAGES : "contains"
CHAT_SESSIONS ||--o{ CHAT_SESSIONS : "continues_from"

CHAT_SESSIONS {
    uuid id PK
    uuid node_id FK
    uuid parent_session_id FK "Previous session (Linked List)"
    string title
    text summary "Summary of THIS session (to be carried over)"
    boolean is_archived "True = Rotated/Closed"
    datetime created_at
    datetime updated_at
}

CHAT_MESSAGES {
    uuid id PK
    uuid session_id FK
    string role
    text content
    jsonb meta_payload "Action data"
    boolean is_excluded "True = Hide from Context (Soft delete/Filter)"
    int token_count
    datetime created_at
}

%% ========================================
%% 4. Orchestration (Inbox)
%% ========================================
```

```
INBOX_ITEMS {
    uuid id PK
    uuid source_node_id FK
    string action_type "TASK_CREATE, etc."
    jsonb payload
    string status "PENDING, APPROVED, REJECTED"
    text resolution_note
    datetime created_at
}

%% ========================================
%% 5. File System & Internal RAG (OS Sovereignty)
%% ========================================
UPLOADED_FILES ||--o{ FILE_CHUNKS : "vectorized_into"

UPLOADED_FILES {
    uuid id PK
    uuid node_id FK "Context Scope"
    string filename
    string storage_path "OS-managed Path (S3/Local)"
    string mime_type
    int size_bytes
    string vector_status "PENDING, COMPLETED"
    string kc_sync_status "PENDING, SYNCED (Sent to Personalization Engine)"
    datetime uploaded_at
}

FILE_CHUNKS {
    uuid id PK
    uuid file_id FK
    int chunk_index
    text content "Actual text segment"
    vector embedding "pgvector(1536) for Internal RAG"
    jsonb metadata "Page number, etc."
}
```

コンテキストローテーションと要約ロジック (Context Rotation

and Summarization Logic)

- **Trigger (発火):**
  - **Auto:** トークン使用率が規定値（例: **80%**）を超えた場合。
  - **Manual:** ユーザーがコマンド `/archive` を実行した場合（「フェーズ1完了」のタイミング等）。
- **Summarization (要約生成):**
  - システムは現在のチャットセッション全体を読み込み、以下の構造を持つ要約テキストを生成する。
    - **Decisions:** 決定事項リスト。
    - **Pending Issues:** 未解決の課題。
    - **Key Facts:** 重要な事実関係。
- **Rotation (DB更新):**
  - 現在の `chat_sessions` レコードの `is_archived` を `True` に設定し、生成した要約を `summary` カラムに保存する。
  - 新規 `chat_sessions` レコードを作成し、`parent_session_id` に旧セッションIDをリンクさせる。
- **Injection (再注入):**
  - 新たなチャットセッションの開始時、直前のセッション（**Parent**）の `summary` を取得し、システムプロンプトの一部として注入する。これにより、**AI**は詳細なログは読み込まずとも、「これまでの経緯」を保持したまま再スタートできる。

Inbox/Push プロトコル詳細 (Inbox/Push Protocol Details)

Hubのコンテキスト汚染を防ぎ、信頼性の高い「情報の蒸留」と「伝達」を実現する非同期通信プロトコルの詳細を定義する。本システムでは、LLMのネイティブ機能である Function Calling (Tool Use) を採用し、構造化データの確実な生成と処理を実現する。

Push Protocol (Spoke to Inbox)

- **Trigger (発火):**
  - **Explicit (明示的):** ユーザーがSpoke内でコマンド（例: `/share`, `/report`）を実行し、Hubへの報告を指示する場合。
  - **Implicit (暗黙的):** Spokeエージェントが「タスクの完了」「スケジュールの遅延」「リソース不足」などの重要イベントを検知し、Hubへの報告が必要と自律的に判断した場合。
- **Generation (Function Calling):**
  - Spokeエージェントは、チャット画面用の自然言語テキストとは別に、定義されたツール `submit_to_inbox` を呼び出す。これにより、パースエラーのリスクなしに構造化データを送信する。

Tool Schema:

```
{
  "name": "submit_to_inbox",
  "description": "Hub (PM Agent) に報告、提案、またはタスク変更依頼を送信する。",
  "parameters": {
    "type": "object",
    "properties": {
      "action_type": {
        "type": "string",
        "enum": ["TASK_CREATE", "PLAN_CHANGE", "REPORT"],
        "description": "申請の種類"
      },
      "payload": {
        "type": "object",
        "description": "タスク名、期日、報告本文などを含む詳細データ"
      },
      "priority": { "type": "string", "enum": ["low", "medium", "high"] }
    },
    "required": ["action_type", "payload"]
  }
}
```

Inbox Buffer Pattern (Asynchronous Processing)

- **Pooling (蓄積と隔離):**
  - ツール実行の結果として、システムはチャット履歴に以下の戻り値（Tool Message）を記録する。
    - `{"status": "queued", "item_id": "uuid-...", "message": "Inboxへ送信完了。承認待ちです。"}`
  - これにより、Spokeエージェントは「送信が成功したこと」を認識でき、重複送信を防ぐことができる。また、このやり取りはHub側のコンテキストには影響を与えない。
- **Fetching & Review (取得と選別 - Human-in-the-loop):**
  - **UI表示:** ユーザーがInbox画面（またはタブ）を開いた際、`status: PENDING` のアイテムがリスト表示される。
  - **Action:** ユーザーは各アイテムに対し、以下の操作を行うことができる。
    - **Approve (承認):** Hubに取り込み、実行を許可する。
    - **Reject (却下):** 取り込みを拒否する（`status: REJECTED`）。
    - **Edit (修正):** 内容（期日や優先度など）を修正した上で承認する。
- **Processing (統合と実行):**
  - 承認されたアクションは、**Hub (PM Agent)** の権限行使としてシステムにより実行される。
    - **LBS操作:** `TASK_CREATE` 等の場合、Hubが LBS API (`POST /tasks`) を

コールしてスケジュールを更新する。
    - 情報統合: `REPORT` 等の場合、その要約がHubのチャットコンテキストに注入され、全体計画の調整材料として使用される。

Command List (実装コマンド一覧)

以下の **Core Command List** を実装対象として定義する。これらは不必要な複雑化を避け、利用頻度の高い機能に絞り込んでいる。

Command List Implementation

- **Navigation Commands (Client-Side)**

Frontendの状態（State）を操作するコマンド。サーバー通信を行わず、即座に反映される。

Command	Alias	Args	Description	Implementation Details
`/switch`	`/sw`, `/go`	`<node_name>`	指定したNode（Spoke/Hub）へタブを切り替える。	入力された `node_name` に対してFuzzy Searchを行い、最も近い `node_id` をアクティブにする。
`/hub`	-	-	Hub (PM Agent) へ移動する。	`/switch hub` のエイリアス。ホームポジションへの復帰用。
`/inbox`	-	-	Inbox画面（タブ）へ移動する。	View Modeを `Chat` から `Inbox` へ切り替える。
`/clear`	`/cls`	-	画面上のチャットログを一時的に消去する。	React Stateの `messages` 配列を空にする（DB削除は行わない）。リロードで復元可能。

- **Action Commands (Server-Side)**

Backend APIをコールし、データやエージェントの状態を変更するコマンド。

Command	Args	Description	Context / Constraint

`/archive`	-	現在のセッションを強制的に要約・終了し、ログローテーションを実行する。	フェーズの区切り等で、ユーザーが明示的にコンテキストをリフレッシュしたい場合に使用。実行後、UIは自動リロードされる。
`/task`	`<title>`	LBSにタスクをクイック登録する。	**Hub**でのみ有効。会話の流れを止めずにタスクを放り込むための機能。日付等は「今日」または「指定なし」として登録される。
`/report`	`[text]`	現在のSpokeからHubへ、`REPORT` アクションとしてInboxへメッセージを送信する。	**Spoke**でのみ有効。引数が空の場合、直近の会話内容を要約して送信する。

- **System Commands (Debug/Maintenance)**

開発・運用保守用。

Command	Args	Description	Safety
`/reset`	-	現在のノードの記憶（チャットログ）を完全に消去する。	確認ダイアログ必須。コンテキストのリセットではなく、物理削除（または論理削除による不可視化）。
`/help`	-	利用可能なコマンド一覧を表示する。	-

Implementation Schema (Request Payload)

Server-Sideコマンドを実行する際のJSONペイロード定義。

```
// POST /api/commands/execute
{
  "command": "task",
  "args": ["牛乳を買う"],
  "context": {
    "user_id": "uuid-...",
    "node_id": "uuid-...",
```

```
    "session_id": "uuid-..."
  }
}
```

Error Handling

- Unknown Command: 「コマンドが見つかりません: /foo」とシステムメッセージ（赤字）で表示し、チャット履歴には残さない。
- Permission Denied: Spokeで `/task` を実行した場合などは、「このコマンドはHubでのみ実行可能です」と警告し、Inboxへの `/report` を促す。

Core System API Specification

Core System (Hub) が提供するRESTful APIの仕様を定義する。
本APIは主に **Frontend (Next.js)** からの呼び出しと、**LBS Sync Worker** からのWebhook受信に使用される。

Common Specifications (共通仕様)

Base URL: `/api/v1`
Authentication:
    User Client: `Authorization: Bearer <JWT>` (OAuth 2.0)
    Internal Service: `X-Service-Key: <SECRET>` (Service-to-Service)
Response Format: JSON

Endpoints Definition

A. Chat & Agent Interaction

ユーザーとHub/Spokeエージェントとの対話を処理する。

Method	Path	Description	Payload / Params

POST	/chat/messages	ユーザーメッセージを送信し、AIによる応答（ストリームまたは完結文）を取得いたします。	{ "session_id": "uuid", "content": "text", "attachments": [] }
GET	/chat/sessions/{id}/messages	指定されたセッションの会話履歴を取得いたします。無限スクロールに対応したカーソルページネーションを適用しております。	?limit=50&before_id=uuid
POST	/chat/sessions	新しいチャットセッション（またはコンテキスト切り替え）を開始いたします。	{ "node_id": "uuid", "title": "New Topic" }
POST	/chat/sessions/{id}/archive	指定されたセッションを強制的に終了させ、要約を生成した上でアーカイブいたします（ログローテーション）。	-

B. Inbox Management (Triage)

Inbox UIでの表示と、ユーザーによる承認・却下操作を受け付ける。

Method	Path	Description	Payload / Params
GET	/inbox	未処理（Pending）のInboxアイテム一覧を取得いたします。	?status=PENDING

POST	/inbox/external	**[Internal]** LBS Sync Worker等の外部プロセスより、Inboxへの新規アイテム登録を受領いたします（Webhook）。	{ "source": "LBS_WORKER", "action_type": "TASK_CREATE", "payload": {...} }
POST	/inbox/{id}/approve	指定されたアイテムを承認し、LBS/KCへの書き込みを実行いたします。修正承認の場合、payloadに上書き内容を含めます。	{ "override_payload": {...} }
POST	/inbox/{id}/reject	指定されたアイテムを却下し、アーカイブまたは削除いたします。Spokeへのフィードバックコメントを付与することが可能です。	{ "reason": "予算不足のため" }

C. Command Execution (CLI)

チャット欄から入力されたスラッシュコマンドを処理する。

Method	Path	Description	Payload / Params
POST	/commands/execute	サーバーサイドコマンド（/archive、/task など）を解析し、実行します。	{ "command": "task", "args": ["..."], "context": {...} }

D. Master Data & System Status

初期ロード時のデータ取得やヘルスチェック用。

メソッド	パス	説明	ペイロード / パラメータ
GET	/nodes	ユーザーがアクセス可能な全てのプロジェクト（Hub/Spoke）の定義リストを取得します。サイドバー描画に利用します。	-
GET	/system/status	**[FR-ARCH1-3]** 各マイクロサービス（LBS, KC）の接続状況および同期ラグ情報を取得します。ステータスバー表示に利用します。	-

Data Types (主要モデル)

InboxItem (Schema):

```json
{
  "id": "uuid",
  "source_node": { "id": "uuid", "name": "Research", "icon": "💎" },
  "action_type": "TASK_CREATE",
  "payload": {
    "title": "追加実験データの整理",
    "due_date": "2025-10-20",
    "load_score": 3.0
  },
  "created_at": "ISO8601",
  "ai_analysis": "スケジュールに空きがあるため承認を推奨します。"
}
```

E. Task Management (LBS Proxy)

LBS Microserviceが管理するタスクデータへのアクセスを提供します。Core Systemは認証・認可を行った後、リクエストをLBSへプロキシします。

Method	Path	Description	Payload / Params
GET	/tasks	全タスクを一覧形式で取得いたします。フィルタリング、ソート、ページネーションに対応しております。	?limit=50&offset=0&status=TODO,DOING&context_id={uuid}&sort=due_date:asc
POST	/tasks/import	CSVファイルをアップロードし、タスクを一括でご登録いただけます。	Content-Type: multipart/form-data{ "file": <binary>}

Task API Details

1. GET /tasks (Query Parameters)

Frontendの「Tasks」画面での表示・検索要件を満たすため、以下のクエリパラメータをサポートします。

Pagination: `limit` (default: 50), `offset` (default: 0)
Filter:

- `keyword`: タスク名・メモの部分一致検索
- `status`: カンマ区切りで複数指定可 (例: `TODO,DOING`)
- `context_id`: 特定のSpokeに絞り込み
- `due_date_range`: `2025-10-01..2025-10-31` (ISO8601)

Sort: `sort=field:order` (例: `due_date:asc`, `load_score:desc`)

Response Schema:

```json
{
  "data": [
    {
```

```
    "id": "uuid",
    "title": "論文Xの査読",
    "status": "TODO",
    "load_score": 3.5,
    "due_date": "2025-11-15T00:00:00Z",
    "context": { "id": "uuid", "name": "Research", "color": "#4A90E2" }
   }
 ],
 "meta": {
   "total": 120,
   "limit": 50,
   "offset": 0
 }
}
```


2. POST /tasks/import (CSV Specification)

他ツールからの移行や、Excelで洗い出したタスクの一括投入に使用します。

- **Header:** `Content-Type: multipart/form-data`
- **File Format:** CSV (UTF-8, Header Row Required)
    1. **Required Columns:** `title`
    2. **Optional Columns:** `due_date` (YYYY-MM-DD), `load_score` (float), `context` (name or slug), `notes`
- **Processing Logic:**
    1. Core SystemがCSVを受け取り、バリデーションを行う。
    2. 有効な行をJSON配列に変換し、LBSのBulk Insert APIへ送信する。
    3. `context` 列の値から自動的に `node_id` を解決する（"Research" -> `uuid-of-research-spoke`）。

Response Schema (Import Result):


```
JSON
{
 "status": "success",
 "summary": {
   "total_rows": 50,
   "imported": 48,
   "failed": 2
 },
 "errors": [
   { "row": 12, "reason": "Invalid date format: '明日'" },
   { "row": 30, "reason": "Context not found: 'UnknownProject'" }
 ]
```

}

外部連携 (External Integration)

Microsoft ToDo (Graph API) 双方向同期

Microsoft Graph APIを使用し、生活事務や雑務（Life Admin）のシームレスな管理を実現する。

Data Mapping (LBS to MS ToDo)

LBSのタスク (`tasks` Table) とMS ToDoのプロパティの対応関係。

LBS Field	MS ToDo Property	備考
`task_id` (UUID)	`openExtension`	カスタムデータとして拡張プロパティに埋め込む
`external_sync_id`	`id`	ToDo側のIDをローカルに保存
`task_name`	`title`	
`notes`	`body/content`	
`due_date`	`dueDateTime`	
`active` = `FALSE`	`status` = `completed`	
`context`	`listId` (Folder)	プロジェクトごとにリストを分けるか、タグで管理

実装詳細

- **Outbound (App -> ToDo):**
    - **Create:** LBSでタスクが作成されると、対応するToDoリスト（例: "Antigravity"）にタスクを追加し、返却されたIDを保存する。
    - **Update:** タスク名や期限が変更された場合、PATCHリクエストで反映する。負荷スコア（LBS固有値）は本文（Body）の末尾に `[LBS: 3.0]` のように記載するか、無視する。
- **Inbound (ToDo -> App):**
    - **Completion:** スマホでチェック完了すると、次回のSyncでLBS側のタスクも `completed` に更新される。
    - **Inbox Capture:** ToDoの「Tasks（既定のリスト）」に追加されたタスクは、「未分類の新規タスク」としてLBSのInboxに取り込み、Hubに割り振りを依頼する。これにより、移動中に思いついたタスクをスマホから放り込める。
    - **Deletion Handling:**
        - ToDo側でタスクが削除された場合、LBS側では物理削除せず、`active=FALSE` (アーカイブ) にする。
        - LBS側で削除された場合、ToDo側も削除（または完了扱い）にする。

Outlook Calendar 連携 (Time Blocking)

LBSの負荷スコアが高いタスクや、時間が固定されたタスク（講義、MTG）をカレンダー上のブロックとして確保する。これらも Microsoft Graph API (`/me/events`) を使用する。

Logic: Time Blocking & Rules

- **Rule:** `load_score >= 5.0` または `rule_type` が時間固定のタスクのみ同期する。全てのTo-Doでカレンダーを埋め尽くさないため。

Feature: Time Allocation Proposal

Hubエージェントがスケジュールを提案する際、Outlookの空き状況を参照する。

1. **Get Availability:** LBS上の `estimated_hours`（もしあれば）に基づき、カレンダーの空きスロット（Free/Busy情報）を取得。
2. **Propose & Book:** 自動配置案を作成し（「この論文読みは火曜の10:00-12:00でいかがですか？」）、ユーザー承認後にイベントを作成する。

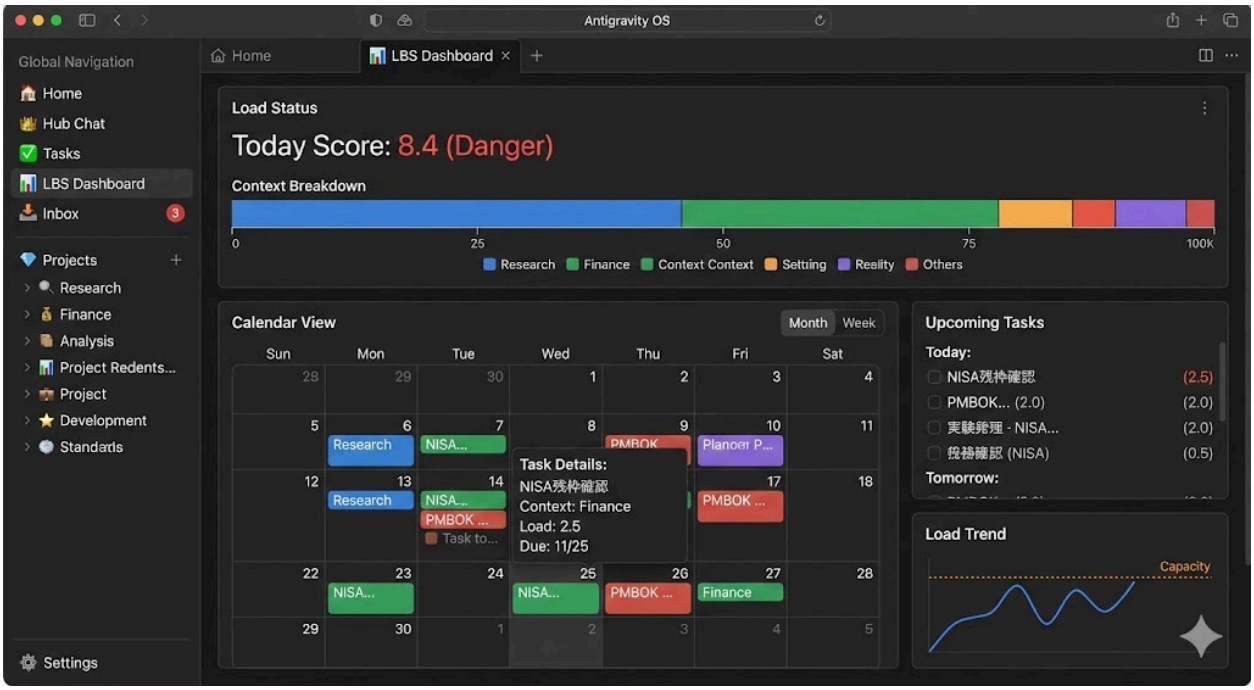
UI/UX設計 (User Interface Design)

本システムは、対話型AIの柔軟性（Chat）と、プロジェクト管理ツールの堅牢性（GUI）を融合させた、エンジニアのための「Chat-Centric IDE (Integrated Development Environment)」としてデザインされる。

画面レイアウト概念 (Layout Concept)

画面全体は「常駐するナビゲーション (Global Navigation)」と「可変のワークスペース (Workspace)」に明確に分離される。これにより、ユーザーは「今どこにいるか」を見失うことなく、複数のタスク（Hubでの戦略策定と、Spokeでの論文執筆など）を並列して進めることが可能となる。

レイアウト構成図:



サイドバー構成 (Navigation Menu)

常時表示される左端のメニューバー（幅50px-200px可変 / Collapsible）。アイコンベースで直感的にモードを切り替える。

- 🏠 **Home:**
  - システムのランディングページ。当日の重要なお知らせや、未処理のInboxアイテムのサマリを表示する。
- 👑 **Hub Chat:**
  - 統合司令塔（Hub Agent）との対話専用画面。
- ✅ **Tasks:**
  - 全タスクをリスト形式で管理し、フィルタリング（Context, Status）、CSVインポート、新規作成を行うマスター管理画面。
  - 各タスク行には `Load Score`, `Task Name`, `Context Tag`, `Due Date` を表示する。
- 📊 **LBS Dashboard:**

    - 負荷状況を可視化する専用アナリティクスビュー（後述）。
- 📥 **Inbox:**
    - Spokeからの報告や外部同期タスクが集まる承認ゲート。
- 💎 **Projects (Spokes Explorer):**
    - `Research`, `Finance` などの各プロジェクト（Spoke）へのアクセス。
- ⚙️ **Settings:**
    - APIキー設定、DB接続設定に加え、System Promptエディタへのショートカット。

ワークスペース (Tabbed Interface)

- **Multi-Tab Interface:**
    - ブラウザやIDEのように、チャット画面やダッシュボードを「タブ」として複数開くことができる。
- **Split View:**
    - タブをドラッグして画面を左右（または上下）に分割可能。「左側で論文を読みながら（Spokeチャット）、右側でスケジュールを確認する（LBS Dashboard）」といった並行作業や、Spoke同士の連携（Research x Finance）を支援する。

LBS Dashboard (Detailed View)

LBS Microserviceが提供する標準的なUIコンポーネントを採用し、認知負荷の「計測」と「予報」に特化したビューを提供する。

System Overview

Real-time cognitive load analysis and predictions.

DAILY CAPACITY
8.0 Units

Recovery Rate	Weekly Avg	Overload Index	Active Tasks
High	**3.5**	**Low**	**2**
42.9% SAFE DAYS	UNITS PER DAY		SCHEDULED FOR TODAY

Load Distribution

Safe · Warn · Danger · Peak

MON	TUE	WED	THU	FRI	SAT	SUN
4.7	4.7	4.7	4.7	2.1	3.1	0.0
Tasks: 2	Tasks: 2	Tasks: 2	Tasks: 2	Tasks: 1	Tasks: 1	Tasks: 0

Today's Context

Remaining capacity:
3.3 Units

4-Week Prediction Trend

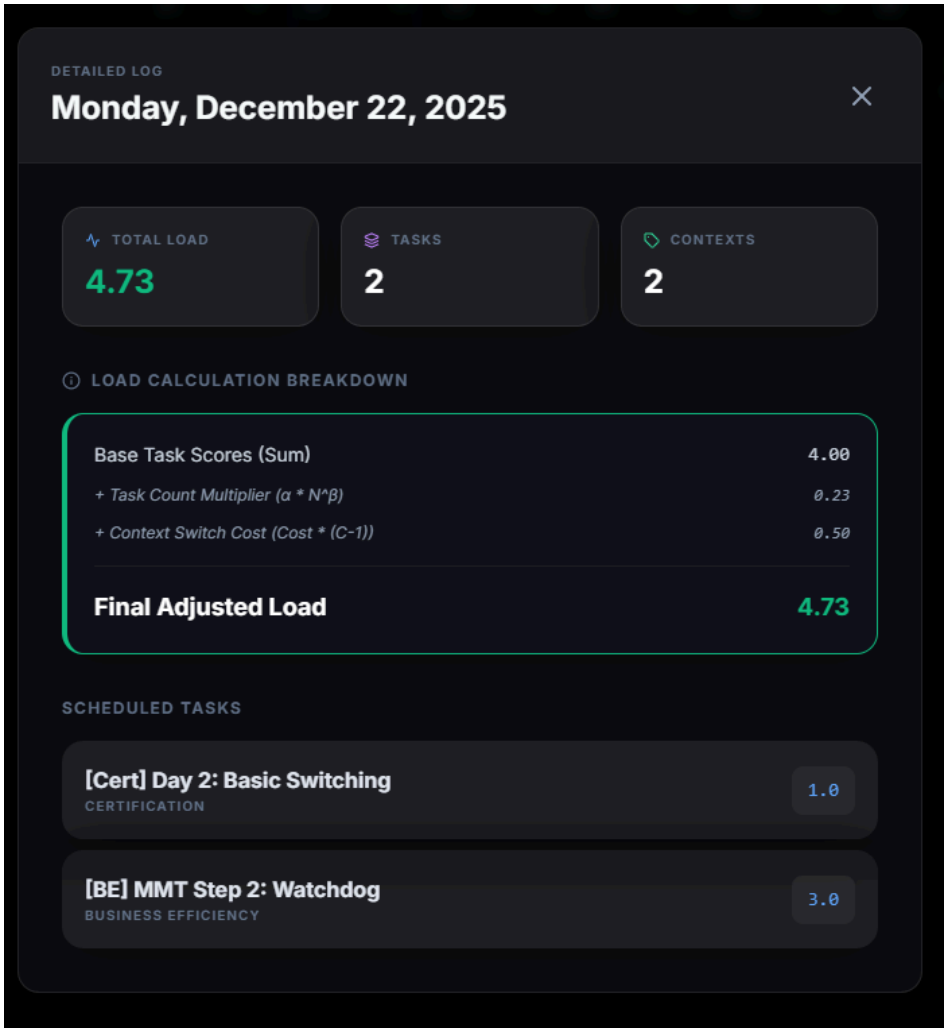
Previous 4W | **Range: 2025/12/22 - 2026/1/19** | Next 4W

表示要素 (Visualization Widgets)

- **System Overview (KPI Cards):**
    - **Recovery Rate:** 休息の質と回復傾向（High/Low）。
    - **Weekly Avg:** 週平均の負荷ユニット数。
    - **Overload Index:** 過負荷のリスクレベル。
    - **Active Tasks:** 当日予定されているタスク数。
- **Load Distribution (Weekly Bar):**
    - 月曜から日曜までの負荷分布をバーチャートで表示。
    - 安全圏（Safe）、警告（Warn）、危険（Danger）を色分けし、週の後半に負荷が偏っていないかを確認する。
- **Today's Context (Ring Chart):**
    - 当日のキャパシティ（例: 8.0 Units）に対する現在の負荷充填率（例: 59%）をリングチャートで表示。
    - "Remaining capacity: 3.3 Units" のように残余力を明示する。
- **4-Week Prediction Trend:**
    - 向こう4週間の負荷推移予測。長期的なプロジェクトによる負荷の山（Peak）を事前に察知する。
- **LBS Calendar:**
    - 月表示のカレンダーで、日ごとの負荷状況（緑のドットや数値）を俯瞰する。日付をクリックすると、その日のタスク詳細と負荷計算の内訳（Detailed Log）を確認できる。

Detailed Log (Calculation Breakdown)

カレンダーやタスクリストから特定の日付を選択した際、モーダルウィンドウで詳細な計算ロジックを表示する。



- **Base Task Scores:** タスク自体の基礎負荷合計。
- **Multipliers:** タスク数による係数（$\alpha * N^\beta$）や、コンテキストスイッチによるペナルティコスト（$Cost * (C-1)$）の加算内訳。
- **Final Adjusted Load:** 補正後の最終負荷スコア。なぜそのスコアになったのかという「根拠」を透明化する。

Inbox UI (Notification Center)

コマンド操作ではなく、LINEやメールクライアントのような「リスト＆詳細」形式のUIを採用する。これは単なる通知確認ではなく、Hubへの情報の流入を制御する「関所（Gatekeeper）」として機能する。

画面構成

- **Message List (Left Column):**
    - Spokeからの報告（Tool Call Result）や、Microsoft ToDoから同期された新規タスク

が時系列で並ぶ。
    - 各アイテムには「送信元アイコン（Spokeの種類）」、「要約テキスト（Subject）」、「受信日時」が表示される。
    - 未読アイテムは太字またはハイライト表示され、処理漏れを防ぐ。
- **Detail View (Right Column):**
    - **Header:** 送信元エージェント名とタイムスタンプ。
    - **Body:** 報告の全文（Markdownレンダリング対応）。「論文Xを読みました。その結果、Yという課題が見つかりました...」といったコンテキストを含む。
    - **Proposed Actions (Diff View):** AIが提案するLBSへの変更内容を差分表示する。
        - `[UPDATE] Task T-101: Status DOING -> DONE`
        - `[CREATE] Task T-New: "追加実験" (Due: 12/05, Load: 3.0)`

アクションフロー (Triage Logic)

ユーザーは詳細ビュー下部のアクションボタン群（Floating Action Bar）で処理を決定する。

- ✅ **Accept (承認・反映):**
    - 提案内容（LBS更新、新規タスク作成）をデータベースにコミットする。
    - 処理後、アイテムは「Archive」タブへ移動し、Hubのチャットログに「〇〇を承認しました」とシステムメッセージが追記される。
- ✏️ **Edit & Accept (修正承認):**
    - 提案内容を編集モードで開く。「タスク追加は認めるが、期日は12/5ではなく余裕を持って12/10にする」「負荷スコアは3.0じゃなくて5.0だ」といった人間の判断を加えてから反映する。
- ❌ **Reject (却下):**
    - LBSへの反映を行わず、アイテムを削除（または却下済みフォルダへ移動）する。
    - オプションで「却下理由」を入力し、送信元のSpokeへフィードバックを返すことができる（例:「今は予算がないため却下」）。
- 💤 **Snooze (保留):**
    - 「後で確認する」としてInboxに残す、または指定時間後に再通知する。