アーキテクチャ　スタック

段階的実装ロードマップ

前章で定義した革新的な機能群を効果的に市場投入するためには、戦略的な段階的展開が不可欠である。本章では、価値を漸進的に提供し、リスクを管理しながら、Issuepediaのビジョンを完全に実現するための4段階のロードマップを提案する。このロードマップは、既存の構想 [1] を拡張し、より具体的で野心的な計画へと昇華させるものである。

フェーズ1：基盤構築と中核価値の証明（MVP）（期間：1〜2ヶ月）

目的:
この初期フェーズの目標は、「ミッシングミドル」仮説の核心部分を最小限の労力で証明することにある 1。すなわち、プロンプトの実践（「何を」）とその背後にある理論（「なぜ」）を結びつけることに価値がある、という仮説を市場に対して実証する。

主要機能:

- **ユーザー認証システム**: Replit Authを活用し、セキュアで簡単なユーザー登録・ログイン機能を提供する [1]。
- **構造化された知識ベース**: `Prompts` テーブルと `PromptTechniques` テーブルを中核とする正規化されたデータベーススキーマを実装する。基本的なCRUD（作成、閲覧、更新、削除）機能もこれに含まれる [1]。
- **理論的根拠の表示**: プロンプト詳細ページにおいて、登録されたプロンプトがどの `PromptTechniques` に基づいているか、そしてその `rationale`（理論的根拠）が明確に表示されるUIを構築する。これがMVPにおける最小実行可能価値である [1]。
- **初期コンテンツのキュレーション**: Chain-of-Thought、Few-shot Prompting、Tree-of-Thoughtなど、基礎的かつ重要な技術を網羅する50〜100件の高品質なプロンプトを、開発チームが手動で作成・登録する。これにより、ローンチ直後からユーザーに具体的な価値を提供する。

フェーズ2：コミュニティ・フライホイールの始動（期間：3〜5ヶ月）

目的:

MVPの基盤の上に、ユーザーの貢献を促進し、自律的な品質管理メカニズムを構築するための社会的・インセンティブ的レイヤーを実装する。プラットフォームを静的な情報源から、生きたコミュニティへと変貌させる。

主要機能:

- **ゲーミファイド・レピュテーション＆ピアレビューシステム（アイデア#2）**: `Users`、`Reviews`、`ReputationEvents`、`Badges`といった関連テーブルを実装する。ユーザーがプロンプトを投稿し、それがピアレビューキューに入るフローを構築する。レピュテーションポイントの増減ロジックと、権限アンロックの仕組みを導入する。
- **強化されたユーザープロフィール**: 各ユーザーのプロフィールページで、現在のレピュテーション、獲得したバッジ、自身の投稿履歴、実施したレビュー履歴を一覧表示できるようにする。これにより、貢献が可視化され、ポートフォリオとしての価値が生まれる。
- **コミュニティインタラクション機能**: プロンプトに対する投票（アップ/ダウン）、コメント、そして既存のプロンプトをベースに改良版を作成する「フォーク」機能を実装する。
- **インタラクティブな「技術ツリー」（アイデア#3）**: データベース内の`PromptTechniques`の関係性を読み取り、視覚的なナレッジグラフとして表示する読み取り専用ページをローンチする。
-

フェーズ3：ビジュアル革命とプロフェッショナル・ツーリング（期間：6〜9ヶ月）

目的:
Issuepediaを市場の他のどの製品とも一線を画す、独自の体験を提供するプラットフォームへと飛躍させる。プロフェッショナルな個人が求める高度なツールを投入し、競争上の決定的な堀を築く。

主要機能:

- **ビジュアル・プロンプト・コンポーザー＆デコンストラクタ（アイデア#1）**: プラットフォームの中心的機能となる、ノードベースのプロンプト設計UIをリリースする。初期バージョンでは、主要なプロンプト技術に対応したカスタムノードを提供し、作成したビジュアルグラフをテキストプロンプトに変換する機能を実装する。
- **民主化されたA/Bテスト（アイデア#4）**: プロンプト詳細ページにA/Bテストパネルを統合する。ユーザーは2つ以上のプロンプトバージョンを選択し、対象モデル、評価指標（レイテンシー、コスト等）を設定してテストを実行し、結果をグラフで比較できる。
- **「プロンプト解剖」AIフィードバック（アイデア#8）**: プロンプト投稿フローにAIによる自動レビュー工程を組み込む。ユーザーがプロンプトを投稿すると、AIがその内容を分析し、改善点を提案する。ユーザーはこのフィードバックを参考に、ピアレビューに提出する前にプロンプトを洗練させることができる。

フェーズ4：スケール、拡張、そして収益化（期間：10〜12ヶ月以降）

目的:
プラットフォームのリーチを開発者エコシステム全体に拡大し、将来の技術トレンドに対応するための基盤を固め、持続可能なビジネスモデルを確立する。

主要機能:

- **Issuepedia VS Codeエクステンション（アイデア#5）**: 開発者の生産性を向上させ、Issuepediaを日常的な開発ツールとして定着させるためのVS Code拡張機能をリリースする。
- **エージェント＆ワークフロー・コンポーザー（アイデア#6）**: ビジュアル・コンポーザーを拡張し、複数のLLMコールやツール利用を組み合わせた複雑なエージェントのロジックフローを設計できるようにする。
- **マルチモーダル対応（アイデア#7）**: 画像を入力として受け付けるプロンプトの投稿・表示機能を試験的に導入する。データベースとUIの対応を進める。
- **収益化モデルの導入**:
  - **Issuepedia for Teams:** 企業やチーム向けのプライベートなワークスペースを提供する有料プラン。チーム内でのみ閲覧・編集可能なプロンプト管理、高度な共同編集機能、アクセス権限管理などを提供する。これはPromptHubやLangfuseのビジネスモデルを参考にする [18]。
  - **認定ラーニングパス（アイデア#9）**: 体系的な学習コースと認定証を有料コンテンツとして提供する。
- **コミュニティ主導グロースエンジン（アイデア#10）**: 埋め込み可能なプロンプトウィジェットやリファーラルプログラムを本格的に展開し、プラットフォームのバイラルな成長を加速させる。

推奨技術スタック

ユーザーからの「最小の労力での開発」という要件 [1] と、将来的な拡張性を考慮し、以下のモダンかつ統合された技術スタックを選定する。

- **プラットフォーム**: Replit
  - **選定理由:** 統合された開発環境（IDE）、認証（Auth）、データベース（PostgreSQL）、そしてホスティングまでをシームレスに提供する。特に、AI開発エージェントとの親和性が高く、インフラ構築の複雑さを大幅に削減できる [1]。
- **AI開発エージェント**: Replit Agent
  - **選定理由:** ユーザーの明確な要求に応えるため。自然言語の指示からフルスタックアプリケーションを構築・修正する能力を持ち、開発ライフサイクル全体を支援する「ペアプ

ログラマー」として機能する[1]。

- **フロントエンド**: Next.js (React & TypeScript)
  - **選定理由**: サーバーサイドレンダリング（SSR）による高いパフォーマンスとSEO特性、強力な型付けによるコードの堅牢性、そして広範なエコシステムを持つ、React開発の業界標準であるため[1]。
- **バックエンド**: Node.js (Next.js API Routes経由)
  - **選定理由**: フロントエンドと同一の技術スタック（JavaScript/TypeScript）で記述でき、Next.jsフレームワーク内にバックエンドロジックをシームレスに統合できるため、開発効率が最大化される[1]。
- **データベース**: Replit Managed PostgreSQL
  - **選定理由**: 構造化データと非構造化データ（JSONB）の両方を効率的に扱えるリレーショナルデータベースの堅牢性と、Replitプラットフォームとの簡単な統合を両立しているため[1]。JSONB型は、ビジュアル・コンポーザーのノード構造のような複雑なデータを格納するのに最適である[23]。
- **認証**: Replit Auth
  - **選定理由**: ソーシャルログインを含む複雑な認証機能を、数行のコードや簡単な指示で実装可能。開発初期段階で最も時間を要する部分を自動化し、コア機能の開発にリソースを集中させることができる[1]。
- **主要UIライブラリ**:
  - **スタイリング**: Tailwind CSS: ユーティリティファーストのアプローチにより、迅速なUI開発と一貫したデザインシステムの構築を可能にする。
  - **ビジュアル・コンポーザー**: React Flow: 高度にカスタマイズ可能なノードベースのグラフUIを構築するための、業界で広く採用されているライブラリ。豊富な機能とドキュメントが、複雑なコンポーザー開発のリスクを低減する[2]。

3.2. システムアーキテクチャ図

本システムは、Replitプラットフォーム上で完結する、モダンなモノリシック・フルスタックアーキテクチャを採用する。各コンポーネント間のインタラクションは以下の通りである。

1. **ユーザーブラウザ**: ユーザーはWebブラウザを通じてアプリケーションにアクセスする。
2. **Next.js フロントエンド (Replit上)**: UIのレンダリング、クライアントサイドの状態管理、インタラクションを処理する。React Flowで構築されたビジュアル・コンポーザーもここで動作する。
3. **Next.js API Routes (Replit上)**: バックエンドロジックを担う。データベースへのCRUD操作、外部LLM APIへのリクエスト、ビジネスロジックの実行などを行う。
4. **Replit Auth**: ユーザーの認証・認可を管理する。保護されたAPIルートへのアクセス制御を行うミドルウェアとして機能する。
5. **Replit PostgreSQL DB**: アプリケーションの全データを永続化する。ユーザー情報、プロンプト、技術、レビューなど、正規化されたデータが格納される。

6. **外部LLM API (OpenAI, Google Gemini等)**: A/Bテスト機能や「プロンプト解剖」機能のために、バックエンドから安全に呼び出される。APIキーはReplitのSecrets機能で管理される。

このアーキテクチャは、フロントエンドとバックエンドが密に連携しつつも、関心事が明確に分離されており、保守性と拡張性に優れている。

3.3. 拡張されたコアデータモデル

アプリケーションの独自の価値は、そのデータモデルに凝縮される。以下に、提案された全機能をサポートするための包括的なデータベーススキーマを示す。

包括的データベーススキーマ（データディクショナリ）

テーブル名	カラム名	データ型	説明	制約/インデックス
Users	`id`	`UUID`	ユーザーの主キー	`PRIMARY KEY`
	`username`	`VARCHAR(50)`	ユニークなユーザー名	`UNIQUE, NOT NULL`
	`email`	`VARCHAR(255)`	メールアドレス	`UNIQUE, NOT NULL`
	`reputation`	`INT`	評判ポイント	`DEFAULT 0, NOT NULL`
	`created_at`	`TIMESTAMPTZ`	作成日時	`NOT NULL`
Prompts	`id`	`UUID`	プロンプトの主キー	`PRIMARY KEY`
	`author_id`	`UUID`	作成者のユーザーID	`FK(Users.id), NOT NULL`
	`title`	`VARCHAR(255)`	プロンプトのタイトル	`NOT NULL`
	`prompt_body_text`	`TEXT`	プロンプトのテキスト本文	`NULLABLE`
	`prompt_body_json`	`JSONB`	ビジュアル・コンポーザーのデータ	`NULLABLE`

	rationale	TEXT	なぜこのプロンプトが有効かの説明	NOT NULL
	version	INT	バージョン番号	DEFAULT 1, NOT NULL
	parent_prompt_id	UUID	フォーク元のプロンプトID	FK(Prompts.id), NULLABLE
	status	VARCHAR(20)	状態 (draft, pending_review, approved, rejected)	NOT NULL
	created_at	TIMESTAMPTZ	作成日時	NOT NULL
PromptTechniques	id	SERIAL	技術の主キー	PRIMARY KEY
	name	VARCHAR(100)	技術名 (例: Chain-of-Thought)	UNIQUE, NOT NULL
	description	TEXT	技術の詳細な説明	NOT NULL
	parent_id	INT	親技術ID（技術ツリー用）	FK(PromptTechniques.id)
Prompt_Technique_Links	prompt_id	UUID	プロンプトID	FK(Prompts.id), PK
	technique_id	INT	技術ID	FK(PromptTechniques.id), PK
Reviews	id	UUID	レビューの主キー	PRIMARY KEY
	prompt_id	UUID	レビュー対象のプロンプト	FK(Prompts.id), NOT

			ID	NULL
	reviewer_id	UUID	レビュアーのユーザーID	FK(Users.id), NOT NULL
	vote	VARCHAR(10)	投票 (approve, reject)	NOT NULL
	comment	TEXT	レビューコメント	NULLABLE
	created_at	TIMESTAMPTZ	作成日時	NOT NULL
ReputationEvents	id	SERIAL	イベントの主キー	PRIMARY KEY
	user_id	UUID	対象ユーザーID	FK(Users.id), NOT NULL
	event_type	VARCHAR(50)	イベント種別 (prompt_upvoted, review_approved, etc.)	NOT NULL
	change_amount	INT	評判ポイントの変動量	NOT NULL
	related_prompt_id	UUID	関連プロンプトID	FK(Prompts.id)
	created_at	TIMESTAMPTZ	発生日時	NOT NULL
Badges	id	SERIAL	バッジの主キー	PRIMARY KEY
	name	VARCHAR(100)	バッジ名 (例: First CoT Prompt)	UNIQUE, NOT NULL
	description	TEXT	バッジの説明	NOT NULL

	`icon_url`	`VARCHAR(255)`	アイコン画像のURL	`NULLABLE`
UserBadges	`user_id`	`UUID`	ユーザーID	`FK(Users.id), PK`
	`badge_id`	`INT`	バッジID	`FK(Badges.id), PK`
	`awarded_at`	`TIMESTAMPTZ`	授与日時	`NOT NULL`

3.4. コアAPI仕様（OpenAPI準拠）

バックエンドは、以下のRESTful APIエンドポイントを公開し、フロントエンドとの明確なインターフェースを定義する。これはAPI設計のベストプラクティスに従う[25]。

- **認証** (`/api/auth/`)
  - Replit Authが提供するエンドポイントを利用。
- **プロンプト** (`/api/v1/prompts`)
  - `GET /`: プロンプトのリストを取得（フィルタリング、ソート、ページネーション機能付き）。
  - `POST /`: 新しいプロンプトを`draft`または`pending_review`ステータスで作成。
  - `GET /{promptId}`: 特定のプロンプトの詳細を取得。
  - `PUT /{promptId}`: 特定のプロンプトを更新（作成者のみ）。
  - `POST /{promptId}/fork`: 特定のプロンプトをフォークして新しい`draft`を作成。
- **レビュー** (`/api/v1/reviews`)
  - `GET /queue`: モデレーション権限を持つユーザーがレビュー待ちのプロンプトリストを取得。
  - `POST /`: プロンプトに対するレビュー（承認/拒否）を投稿。
- **ユーザー** (`/api/v1/users`)
  - `GET /{userId}`: 特定のユーザーの公開プロフィール情報を取得。
  - `GET /{userId}/reputation_history`: 特定のユーザーの評判イベント履歴を取得。
- **技術** (`/api/v1/techniques`)
  - `GET /`: 全てのプロンプト技術のリストを取得。
  - `GET /tree`: 「技術ツリー」描画用の階層化されたデータを取得。