学習計画ツール (PlanMaker) 開発プロジェクトにおける学習・知見サマリー

1. はじめに

本ドキュメントは、学習計画ツール「PlanMaker」のAI主導開発プロジェクトを通じて、私が習得した技術的知見、開発プロセス、および品質管理のノウハウをまとめたものです。基礎資料として作成しました。

2. 外部リソースからの学習: AI開発・セキュリティの最新知見

プロジェクトの品質と効率を高めるため、まず以下の最新記事やベストプラクティスを学習し、開発の指針としました。

2.1 高度なAI開発手法

参照:

- 技術負債も理解負債も生まないAIコーディング手法
- ゼロから始めるContext Engineering
- AIが一斉に走り出して設計・実装・テスト・レビューを駆け抜けるClaude Codeプラグイン
- **理解負債 (Understanding Debt)**: AIが生成したコード量に対して、人間の理解が追いつかなくなる現象。「動けばよい」で済ませず、仕様や構造を深く理解する必要があります。
- **多重レビュー体制**: AI (Claude) に実装計画を作成させ、別のAI (Codex等) にレビューさせ、最後に人間が承認するフローにより手戻りを防ぎます。
- **Context Engineering**: LLMの記憶容量（コンテキスト）を管理するため、System Instructionを維持しつつ、古い実行結果（Observation）を要約・削除する戦略が有効です。
- **DevFlow**: 設計・実装・テスト・レビューを専門AIエージェントが分業して行う未来像を意識し、人間はPM（要件定義）に徹するスタイルを目指します。

2.2 セキュリティ・Git運用のベストプラクティス

参照:

- GitHub Skills
- APIキー漏洩インシデント対応記録
- 個人開発で「やりすぎない」APIセキュリティ設計
- 個人開発でFirestoreを選ぶべき？
- Next.js Security Headers
- **GitHub Flow**: 直コミット禁止、ブランチによる作業分離、PRレビューの徹底。
- **機密情報管理**: `google-services.json` などの自動生成ファイルも必ず `.gitignore` し、漏洩時は `git-filter-repo` で履歴から抹消する。
- **多層防御 (Defense in Depth)**: Edge (Cloudflare) -> App (Auth/JWT) -> DB (RLS) の3層防御を意識する。

- **Firestoreセキュリティ**: 「Firestoreを使う = セキュリティルールを書く」こと。最低限のルール（ `request.auth.uid == userId` ）は必須。
- **Next.jsヘッダー**: HSTS（HTTPS強制）、X-Frame-Options（クリックジャッキング防止）、CSP（XSS対策）を適切に設定する。

3.【実践編】PlanMakerプロジェクトへの適用

上記の学習を踏まえ、本プロジェクトでは以下の技術スタックとプロセスを採用・確立しました。

3.1 プロジェクト概要と技術スタック

オフラインでも動作する「ローカルファースト」な学習計画管理アプリケーションです。

- **Frontend**: React, Vite, TypeScript (型安全性と高速ビルド)
- **UI Framework**: Tailwind CSS, shadcn/ui (再利用可能なコンポーネント設計)
- **State Management**: React Hook Form, Zod (堅牢なバリデーション)
- **Database**: Dexie.js (IndexedDB wrapper)
    - **Local-First Architecture**: ユーザーのブラウザ内にデータを保存し、プライバシー保護と高速動作を実現。

3.2 開発プロセス (AI-Driven & Git Flow)

外部リソースから学んだ手法を実践しています。

- **AIパートナーシップ**: 私はPMとして「要件」「成果物」「制約」を定義することに集中し、コーディングはAIに委ねつつも「理解負債」を溜めないよう仕様を把握しています。
- **Git運用**: `feat/` や `fix/` ブランチで作業し、必ずPRを作成・確認してからマージする GitHub Flow を遵守しています。

3.3 品質管理 (Quality & Design)

- **コード品質**: `// TODO` や `any` 型を避け、一貫性のある設計（コンポーネント駆動）を維持しています。
- **検証**: 実装ごとのビルドチェック（ `npm run build` ）とブラウザ確認による短いフィードバックループを回しています。

3.4 セキュリティ実装 (Local-First Security)

学んだセキュリティ知見を以下のように適用しています。

- **データ保護**: 個人情報（生徒データ）は外部サーバーに送らず、ローカル (IndexedDB) で完結させることで漏洩リスクを最小化。
- **安全な実装**: ログへのPII混入防止、 `npm audit` による脆弱性チェック、コードレビュー時のセキュリティ確認を徹底しています。

4. 今後の展望

現在は「カリキュラム作成（Phase 3）」まで完了しており、今後は以下のフェーズに進みます。

- **Phase 4**: 授業記録機能（日々の進捗データの蓄積）
- **Phase 5**: レポート・可視化（蓄積データに基づく分析）

これまでのフェーズで確立した「ローカルファースト」「コンポーネント駆動」「厳格な型チェック」の基盤、そして新たに学んだ「多層防御」「AI協調フロー」の知見を活かし、データの連携が複雑になる今後のフェーズでも品質を維持しながら開発を進めていきます。