# **Starlist　アーキテクチャ**

## **1. 技術スタックの概要**

### **フロントエンド技術詳細**

* **最小SDKバージョン**: Android API 21 (Android 5.0)、iOS 12.0
* **Flutter バージョン**: 3.10.0以上
* **Dart バージョン**: 3.0.0以上
* **アーキテクチャパターン**: Clean Architecture + MVVM
* **状態管理**: Riverpod（Provider, StateNotifier, AsyncValue）
* **ルーティング**: go\_router
  + **選定理由**: Flutterチームによる公式サポートがあり、ドキュメントも豊富で学習コストが比較的低いこと、宣言的なAPIによる可読性の向上、Web対応への強み、そしてStarlistの機能要件とシンプルさのバランスを考慮し選定しました。
* **パッケージ依存関係**:
  + **HTTP通信**: dio
    - **選定理由**: 複数の外部API（アフィリエイトASP、オファーウォール等）との連携や、インターセプターによる共通処理（認証、エラーハンドリング、ログ）、タイムアウト設定、ファイルダウンロード等の豊富な機能が、効率的かつ堅牢な実装に寄与するため選定しました。http パッケージは基本的な用途に留めます。
  + **ローカルストレージ**: 用途に応じた使い分け
    - shared\_preferences: 簡単なキーバリューペア（設定値など）の保存に使用します。
    - hive: アプリ内でのデータキャッシュ、頻繁にアクセスするデータの保存、その他オフライン利用を想定する構造化データの保存に使用します。Dartネイティブで高速であり、オブジェクト保存の容易性からFlutterアプリとの親和性が高いため、主要なローカルDBとして選定しました。
    - sqflite: 複雑なリレーショナルデータを扱う強い要件が発生した場合の選択肢として残しますが、基本は hive を優先します。
  + **画像処理**: cached\_network\_image, image\_picker, photo\_view
    - image\_pickerは、ユーザーがレシートや利用履歴のスクリーンショット等をギャラリーから選択し、外部OCRアプリでの利用や、Starlistへの画像添付（必要な場合）に使用することを想定します。
  + **認証**: supabase\_auth
    - **選定理由**: バックエンドがSupabaseで構築されているため、認証機能もSupabaseのエコシステム内で完結させることで、開発効率の向上、ユーザー管理の一元化、Row Level Security (RLS)とのスムーズな連携が期待できるため選定しました。firebase\_auth および flutter\_appauth は採用を見送ります。
  + **UI/UXライブラリ**: flutter\_hooks, intl, shimmer, lottie
  + **解析/追跡**:
    - firebase\_analytics
    - firebase\_crashlytics (アプリケーション監視 - 初期)
      * **選定理由**: Firebaseプロジェクトとの連携が容易で、リアルタイムのクラッシュレポート機能があり、Flutterにも対応しているため、初期のアプリケーション監視ツールとして選定しました。必要に応じて Sentry のような高機能ツールの導入を将来的に検討します。
  + **オファーウォールSDK**: 各事業者指定のもの
    - フェーズ1 (初期): SKYFLAG
    - フェーズ2 (運用安定・収益拡大期): 広告メディエーション: ironSource LevelPlay 等、追加事業者: Tapjoy/ironSource 等

### **バックエンド技術詳細 (Supabase構成)**

* **PostgreSQL**: バージョン14以上
* **ストレージ**: S3互換ストレージ
* **リアルタイム機能**: Postgresの変更通知機能
* **認証**: JWT、ソーシャルログイン連携 (Supabase Authによる)
* **Edge Functions**: Deno ランタイム

### **コード品質管理**

* **静的解析**: dart analyze, flutter\_lints
* **テスト**: flutter\_test, mockito, bloc\_test (またはRiverpod用のテストユーティリティ)
* **テストカバレッジ目標**: 最低70%
* **コーディング規約**: Flutter公式スタイルガイド準拠

## **2. アーキテクチャの層**

* **プレゼンテーション層 (UI)**:
  + 状態管理: Riverpod（Provider, StateNotifier, AsyncValue）
* **ドメイン層 (Usecase/Interactor)**:
  + ビジネスロジックを配置
* **データ層 (Repository)**:
  + ローカルストレージ技術: shared\_preferences, hive
  + HTTPクライアント: dio

## **3. ディレクトリ構造**

lib/  
├── src/  
│ ├── core/ # 共通機能  
│ │ ├── api/ # API通信 (dioクライアント設定等)  
│ │ ├── cache/ # キャッシュ (hive設定等)  
│ │ ├── config/ # 設定  
│ │ ├── error/ # エラー処理  
│ │ ├── navigation/ # ルーティング (go\_router設定)  
│ │ ├── performance/ # パフォーマンス関連  
│ │ └── security/ # セキュリティ関連  
│ ├── features/ # 機能モジュール  
│ │ ├── auth/ # 認証  
│ │ ├── data\_input/ # データ入力処理 (テキスト解析等)  
│ │ ├── payment/ # 支払い  
│ │ ├── privacy/ # プライバシー  
│ │ ├── ranking/ # ランキング  
│ │ ├── subscription/ # サブスクリプション  
│ │ └── youtube/ # YouTube統合  
│ └── shared/ # 共有コンポーネント  
│ └── widgets/ # 共通ウィジェット  
└── main.dart

*注: data\_input 機能モジュールを追記し、テキストデータの解析・処理に関するロジックを配置する想定としました。*

## **4. 主要なコンポーネント**

* **認証機能**: (Supabase Auth を利用)
  + OAuth/OpenID Connect: Google, Apple, Twitter, Instagram などSupabaseがサポートする範囲
  + 多要素認証: SMS, Authenticator App (Supabaseの提供状況と優先度により検討)
  + セッション管理: JWT, リフレッシュトークン (Supabase Authによる)
* **決済処理機能**:
  + 決済ゲートウェイ: Stripe
    - **選定理由**: 開発者フレンドリーなAPI、豊富なドキュメント、サブスクリプション管理機能(Stripe Billing)、スケーラビリティを評価し、Starlistの多様な決済ニーズに対応できるため主軸として選定しました。PayPal は必要に応じて補完的に導入を検討します。
  + サブスクリプション管理: Stripe Billing
  + 課金イベントトラッキング
  + 税務管理: 消費税自動計算、インボイス発行 (Stripeの機能や連携サービスを活用)
* **外部API連携**:
  + アフィリエイトASP API
  + オファーウォール事業者API
  + 将来検討: YouTube Data API, Spotify API
* **コメント管理システム**:
  + フィルタリングエンジン: NLP、機械学習モデル、カスタムルール (Supabase Functionsや外部サービス連携で構築)
  + データベース設計: コメントステータス、階層構造、ユーザー信頼度スコアリング (PostgreSQLで設計)
* **テキストデータ入力・処理機能（ユーザーのコピー＆ペーストを起点とする）**:
  + **入力インターフェース**: スターが外部でOCR処理したテキストデータをStarlistアプリにペーストするためのUIを提供。
  + **入力タイプ選択**: ペースト前に情報の種類（レシート、YouTube履歴等）をスターに選択させる機能。
  + **クライアントサイド解析**: 選択されたタイプに基づき、キーワード検出、正規表現、簡易パターンマッチング等を用いてペーストされたテキストを解析し、構造化データフィールド（日付、品目、金額、タイトル等）に情報を振り分け。
  + **確認・編集UI**: 解析結果を編集可能なフィールドに表示し、元のペーストテキストも参照用に併記。スターが自由に修正・追記可能とする。
  + **カテゴリ分類補助**: 入力タイプに応じた初期カテゴリの割り当てと、スターによるカスタムタグ/カテゴリ設定機能。
* **有料商品提案システム**:
  + 決済処理: マイクロトランザクション、複数料金プラン、返金ポリシー (Stripeで対応)
  + 提案管理: ステータス追跡、カテゴリ分類、画像添付
* **アクセス制御システム**:
  + 権限管理: RBAC (SupabaseのRLSやカスタムロジックで実現)、動的権限割り当て、時間ベース制限
  + コンテンツ保護: E2E暗号化（検討）、DRM（検討）、スクリーンショット検出・防止（検討）
* **コメント削除システム**:
  + 削除処理エンジン: 即時削除、関連データ連動、キャッシュ同期
  + ユーザーインターフェース: 直感的操作、一括削除、削除履歴

## **5. データフロー**

* **テキストデータ入力フロー**: （ユーザーによる対象データ準備と外部OCR処理、Starlistアプリへのコピー＆ペースト、入力タイプ選択、アプリ内でのクライアントサイド解析と構造化、スターによる検証・編集・カテゴリ分類、データのアプリ内統合・整形、ファンへの公開）このフローをサービスに合わせて詳細化します（具体的な図やシーケンスは別途定義）。

## **6. エラーハンドリング**

* APIエラーハンドリング: 標準HTTPステータスコード、詳細エラーメッセージ (dioインターセプターやSupabaseクライアントでの共通処理)

## **7. セキュリティ**

* **認証・認可**:
  + パスワード強度: NIST SP 800-63B準拠 (Supabase Authの設定とクライアントサイドバリデーション)
  + アクセス制御: 最小権限の原則、RBAC (Supabase RLS活用)
* **データ保護**:
  + 暗号化: 保存時AES-256 (Supabase/PostgreSQL標準機能)、転送時TLS 1.3 (Supabase標準)
  + センシティブデータ: PIIの特別保護 (マスキング、アクセス制限等)
  + データ消去: 削除リクエスト後30日以内の完全消去ポリシー
* **脅威対策**:
  + OWASP Top 10対応 (継続的な脆弱性対策)
  + DDoS対策: Supabase標準の対策に加え、必要に応じて Cloudflare 等のCDNサービス利用を検討
  + 脆弱性スキャン: 定期的な自動スキャン

## **8. パフォーマンス最適化**

* **応答時間**:
  + API応答: 95%のリクエストが500ms以内
  + 画面描画: First Contentful Paint 1秒以内
  + インタラクション遅延: 100ms以内
* **スケーラビリティ**:
  + 水平スケーリング: Supabaseのサーバーレスアーキテクチャを活用。Edge Functionsの活用。
  + データベース: 読み取り/書き込み分離（検討）、コネクションプーリング (Supabase管理)
* **キャッシュ戦略**: 多層キャッシング戦略（CDN, アプリケーションレベルでの hive 活用, データキャッシュ）
* **バッチ処理**:
  + 非同期タスク: Supabase Edge Functions やキューベースの処理（外部サービス連携も検討）
  + 定期バッチ: スケジュールされたSupabase Edge Functions
  + 優先度制御: 重要タスクの優先処理（設計考慮）

## **9. テスト戦略**

* テストカバレッジ目標: 最低70%
  + ユニットテスト、ウィジェットテスト、インテグレーションテストをバランス良く実施。

## **10. デプロイメント**

* **CI/CD**:
  + パイプライン: GitHub Actions
    - **選定理由**: GitHubリポジトリとの親和性、豊富なコミュニティアクション、無料枠の活用、Flutterプロジェクトにおける実績を考慮し選定しました。CircleCI は採用を見送ります。
  + デプロイ戦略: Blue-Green、Canary (サービス規模とクリティカリティに応じて検討)

## **11. 監視と分析**

* **アプリケーション監視**: Firebase Crashlytics (初期)
  + 選定理由はフロントエンド技術詳細に記載。
* **パフォーマンス監視**: Firebase Performance (初期)
  + **選定理由**: アプリの起動時間、HTTPリクエスト遅延、カスタムトレースなどを監視でき、Firebaseエコシステムとの連携が容易なため、初期のパフォーマンス監視ツールとして選定しました。必要に応じて New Relic のような高機能ツールの導入を将来的に検討します。
* **ログ管理**: Supabase標準のログ機能 および Firebase Crashlytics 等のアプリケーション監視ツールが提供するログ機能 (初期)
  + **選定理由**: まずはSupabaseとFirebaseの提供機能を活用し、運用負荷を抑えます。より高度なログ集約・分析基盤が必要になった際に Datadog や ELK Stack の導入を検討します。
* **アラート**: 上記監視ツールと連携する形で設定。具体的なアラートツール（例: PagerDuty, OpsGenie）の選定は、監視体制の詳細化と合わせて将来的に決定します。

## **12. インフラストラクチャ要件**

* **クラウドインフラ**:
  + ホスティング: Supabase (主にAWS上でホスト)
  + 補助的なクラウド利用: Google Cloud Platform (GCP)
    - **選定理由**: バックエンド処理は極力Supabaseで完結させますが、Supabaseの機能でカバーできない特定の高度な処理が必要になった場合に、GCPの利用を検討します（例：将来的に高度なAI処理をバックエンドで利用する場合など）。AWS 、Azure は現時点では優先度を下げます。
  + コンテナ化: Docker, Kubernetes（現時点ではSupabaseのアーキテクチャを主とし、必要に応じて検討）
  + CDN: Supabase標準のCDN を主として利用。 必要に応じて、DDoS対策や高度なキャッシュコントロール、WAF機能などを目的に Cloudflare の利用を検討します。CloudFront 、Fastly は現時点では優先度を下げます。
  + 負荷分散: Supabaseのサーバーレスアーキテクチャと標準機能に依存。