Clover Pricing Calculator – Architecture

Objective

Build a secure backend system that analyzes merchant card processing costs and generates Blockpay pricing proposals. The solution automates data extraction from statements, supports manual entry when needed, and maintains full security and auditability.

Backend Requirements

Core Features

- PDF upload with auto-extraction of merchant statement data (multiple processors)
- Support complex payment pricing models:
  - Cost-plus
  - iPlus
  - Discount rate
  - Flat
- Surcharge programs
- Dynamic cost comparison and savings calculations
- Admin-controlled configuration for pricing rules, devices, fees, and user permissions
- Role-based access control (admin vs agents)
- PDF proposal generation with branded summaries and disclaimers

Fee Structure Support

- Qualified / non-qualified interchange
- Card brand and network fees
- Per-item transaction fees
- Monthly, one-time, and miscellaneous fees
- Internal device costs (Clover POS and Payment)

Security Requirements

- Strong authentication with JWT tokens
- Encrypted data at rest and in transit
- Strict access boundaries between users
- Secure handling of uploaded PDFs and generated reports

1) Backend API Layer

The backend is the central system of record and control.

Key Responsibilities

- Authentication and authorization (agent vs admin)
- Creation and lifecycle management of analyses
- Secure storage of uploaded statements and generated documents
- Pricing and savings calculations
- Submission workflows and notifications
- Audit logging and access control
- Admin-controlled configuration for pricing rules, devices, fees, and user permissions

Infrastructure

- Secure cloud infrastructure (AWS)
- Encrypted data storage
- Strict access policies
- PostgreSQL database

2) Document Ingestion and Extraction

Automates the analysis of merchant statements.

Flow

- Statements uploaded securely to cloud storage
- PDF text extraction using pdfplumber combined with LLaMA 4 Maverick via Groq API for intelligent data structuring
- Support multiple processors and statement layouts (Chase, Clover, Square, etc.)
- Key fields (rates, fees, volumes, transaction counts) normalized into standard schema
- Confidence scores assigned to extracted fields
- Low-confidence values flagged for review and correction
- Hybrid approach: text parsing with regex patterns, fallback to table extraction

Fallback Support

If no statement is provided, system switches to manual-entry flow.

3) Pricing and Savings Engine

Deterministic calculation engine for consistent results.

Inputs

- Merchant's current processing costs (extracted or manual)
- Blockpay proposed pricing

- Selected devices, accessories, and SaaS plans
- Pricing model and fee structure configuration

Outputs

- Current processor total cost
- Blockpay proposed total cost
- Net and percentage savings
- Savings across multiple timeframes
- Structured data for charts and reporting

Data Integrity

Each submission stores a complete pricing snapshot so historical results remain unchanged even if pricing tables are updated later.

4) PDF Generation

Backend generates branded proposal PDFs after submission.

PDF Contents

- Competitor comparison
- Pricing breakdown
- Devices and SaaS details
- Savings charts and summaries
- Required disclaimers and compliance copy

Generated PDFs are stored securely and linked to the corresponding analysis.

5) Admin and Oversight

Admin layer provides visibility and control.

Admin Capabilities

- View all analyses and submissions
- Track agent activity and submission history
- Revoke agent access when needed
- Review audit logs and basic reporting
- Configure pricing rules and fee structures
- Manage user permissions

6) Security and Compliance

Security is a core design principle for handling sensitive financial statements.

Security Controls

- Encrypted data at rest and in transit
- Secure file uploads and downloads
- Role-based access control
- MFA support
- Immutable audit logs
- Configurable data retention policies
- JWT token authentication
- PBKDF2 password hashing

Key Management

- **AWS KMS (Key Management Service)** - Centralized key management for encryption
- **Automatic key rotation** - Regular rotation of encryption keys
- **Separate keys per environment** - Development, staging, and production isolation
- **Audit trail for key access** - Complete logging of key usage
- **Envelope encryption pattern** - Application encrypts sensitive fields with data keys protected by KMS
- **Storage integration** - S3 object encryption with KMS keys; database/storage encryption uses KMS-backed keys where applicable

Rate Limiting & Abuse Protection

- **API rate limiting** - Prevent excessive requests per user/IP
- **Request throttling** - Control concurrent operations
- **Upload size limits** - Restrict PDF file sizes
- **Request validation** - Input sanitization and validation
- **Monitoring and alerting** - Track suspicious patterns
- **IP-based restrictions** - Block malicious actors
- **Implementation note** - Enforce limits at the edge (e.g., WAF / gateway) and in-app (e.g., DRF throttles) for defense in depth

7) Offline-First Sync Strategy

Enables agents to work without constant connectivity, particularly on iPad devices.

Sync Architecture

- **Local-first data storage** - All data stored on device first
- **Background synchronization** - Automatic sync when connection available
- **Queue-based uploads** - Reliable upload retry mechanism

- **Optimistic updates** - Immediate UI feedback with eventual consistency

Conflict Resolution

- **Last-write-wins strategy** - Timestamp-based conflict resolution
- **Server authority** - Server state takes precedence in conflicts
- **Conflict detection** - Version tracking on all mutable entities
- **User notification** - Alert users when conflicts are detected and resolved

Multiple Draft Handling

- **Draft versioning** - Track multiple versions of incomplete analyses
- **Auto-save functionality** - Periodic local saves to prevent data loss
- **Draft synchronization** - Sync drafts across devices for same user
- **Abandoned draft cleanup** - Automatic cleanup of old, incomplete drafts

Device-Level Encryption (iPad Storage)

- **iOS Keychain integration** - Secure credential storage
- **File-level encryption** - Encrypt all stored PDFs and sensitive data
- **Secure enclave usage** - Hardware-backed encryption for keys
- **Data wiping on logout** - Clear local data on user logout or device change
- **Web offline storage note** - For web/PWA, store offline drafts in IndexedDB with best-effort at-rest encryption via WebCrypto; treat browser storage as lower assurance than iOS Keychain + File Protection

Flutter-Specific Offline Implementation

- **drift (formerly moor)** - Type-safe, reactive SQLite wrapper for local database
- **Sync Queue Manager** - Custom Flutter service for reliable upload retry with exponential backoff
- **Connectivity monitoring** - connectivity_plus package to detect online/offline state
- **Background sync** - workmanager for iOS background task execution
- **State synchronization** - Riverpod with StateNotifier for managing offline/online state transitions
- **Data flow**: User Action → Local DB (immediate) → Sync Queue → API (when online) → UI update
- **Isolates for heavy tasks** - PDF compression and encryption in background isolates

8) Cost Controls & Optimization

Managing operational costs, particularly for AI-powered extraction.

AI Extraction Cost Management

- **Cost visibility dashboard** - Real-time tracking of AI API costs
- **Per-user cost tracking** - Monitor costs by agent and admin
- **Budget alerts** - Notifications when approaching cost thresholds
- **Usage analytics** - Detailed reports on extraction patterns

Throttling & Batching

- **Request batching** - Group multiple extractions when possible
- **Rate limiting per user** - Prevent excessive AI API usage
- **Queue management** - Batch processing during off-peak hours
- **Cache extraction results** - Avoid re-processing identical documents
- **Fallback to manual entry** - Option to skip AI extraction for cost savings
- **Budget guardrails** - Per-tenant daily/monthly spend caps, per-document page limits, and concurrency limits for extraction jobs

Resource Optimization

- **PDF preprocessing** - Optimize file sizes before extraction
- **Selective AI usage** - Use AI only for complex documents
- **Response caching** - Cache common extraction patterns
- **Cost allocation tracking** - Attribute costs to specific business units

9) Technology Stack

Backend Framework & Core

- **Python 3.10+** - Primary programming language
- **Django 5.0.1** - Web framework
- **Django REST Framework 3.14.0** - RESTful API development
- **PostgreSQL** (psycopg2-binary 2.9.9) - Production database
- **SQLite** - Development database

Authentication & Security

- **djangorestframework-simplejwt 5.3.1** - JWT token authentication
- **PBKDF2** - Password hashing algorithm
- **django-password-validators 1.7.1** - Password strength validation
- **django-cors-headers 4.3.1** - CORS handling
- **MFA Support** - Multi-factor authentication

PDF Processing & Extraction

- **pdfplumber 0.11.0** - Fast PDF text extraction
- **PyMuPDF 1.23.26** - PDF rendering and manipulation
- **Pillow 10.2.0** - Image processing
- **groq 0.11.0** - LLaMA 4 Maverick integration via Groq API

- **Hybrid extraction approach** - Text parsing with regex patterns, fallback to table extraction

Data Validation & Utilities

- **django-filter 23.5** - Advanced filtering for querysets
- **django-phonenumber-field 7.3.0** - Phone number validation
- **phonenumbers 8.13.27** - Phone number parsing library
- **pytz 2024.1** - Timezone support
- **python-decouple 3.8** - Environment variable management

Frontend (Flutter + React)

- **Flutter 3.x+** - Primary iPad-first client (and can target web if using a single-codebase approach)
- **Dart** - Primary programming language for Flutter
- **State Management (Flutter)** - Riverpod or Bloc (recommended for offline-first patterns)
- **Local Storage (Flutter)** - drift/sqflite for offline relational database
- **Secure Storage (Flutter)** - flutter_secure_storage (iOS Keychain integration)
- **HTTP Client (Flutter)** - dio with interceptors for JWT authentication and caching
- **PDF Handling (Flutter)** - file_picker (upload), pdf/printing (generation), syncfusion_flutter_pdfviewer (display)
- **Charts & Visualization (Flutter)** - fl_chart or syncfusion_flutter_charts for savings graphs
- **React (Web frontend)** - Web UI for admin portal and/or agent web access
- **React stack (suggested)** - TypeScript, Next.js (or Vite), React Query, component library (MUI/Ant), charting (Recharts/ECharts), PDF viewing (react-pdf)
- **PWA/offline (React, optional)** - Service worker + IndexedDB (for drafts/caching), with JWT session handling and secure storage considerations

Infrastructure & Cloud

- **AWS (Amazon Web Services)** - Cloud hosting platform
- **Encrypted Storage** - Data at rest encryption
- **Secure File Uploads** - S3 or equivalent for PDF storage

Development Tools

- **Git** - Version control
- **pytest** - Testing framework (implied from Django best practices)
- **black** - Code formatting

- **isort** - Import sorting
- **flake8** - Code quality checking

API & Integration

- **RESTful API** - JSON-based communication
- **Groq API** - LLM inference for intelligent data extraction
- **Factory Pattern** - Processor detection and routing

Security Features

- JWT token-based authentication
- Role-based access control (RBAC)
- Encrypted data at rest and in transit
- File upload validation and sanitization
- Immutable audit logs
- Configurable data retention policies

10) Flutter Client Architecture

The Flutter client provides a unified iPad-first and web-compatible interface for agents and admins.

Application Structure

- **Clean architecture with 3 layers**: Presentation (UI widgets) → Domain (business logic) → Data (repositories, API, local DB)
- **Feature-based organization**: Authentication, analysis, pricing, admin modules
- **State management**: Riverpod (recommended for offline-first) or Bloc (for complex workflows)

Core Technologies

- **Local database**: drift (type-safe SQLite) for offline storage
- **API client**: dio with JWT interceptors, retry logic, certificate pinning
- **Data models**: freezed for immutable models, json_serializable for API contracts
- **Secure storage**: flutter_secure_storage (iOS Keychain) for tokens and sensitive data
- **Biometrics**: local_auth for Face ID/Touch ID authentication

Offline-First Implementation

Sync flow: User action → Local DB (immediate) → Sync queue → API (when online) → Conflict resolution - **Background sync**: Runs every 30 seconds when connectivity available - **Retry strategy**: Exponential backoff (5s, 15s,

45s, 2m, 5m), flag after 5 failures - **Conflict resolution**: Server timestamp wins, local backup preserved, user notified - **Database tables**: analyses, pricing_snapshots, sync_queue, user_credentials, extraction_cache

PDF & Document Handling

Upload: file_picker → compression in isolate → chunked upload (>10MB) → encrypted local cache **Display**: syncfusion_flutter_pdfviewer with zoom, annotations, offline viewing **Features**: Drag-and-drop (iPad), Files app integration, resume interrupted uploads

Charts & Visualization

Library: fl_chart for interactive savings charts (bar, line, pie, combined) **Features**: Touch interactions, tooltips, swipe navigation, responsive sizing for iPad (10.2", 11", 12.9") **Time periods**: Daily, weekly, monthly, quarterly, yearly views **Export**: Capture as image, include in PDF proposals

Platform-Specific Features

iPad Optimizations

- **Adaptive UI**: Split view (master-detail), slide over, picture-in-picture, multitasking
- **Apple Pencil**: Signature capture, PDF annotations, handwriting recognition
- **Files integration**: Import/export via Files app, iCloud sync (optional), AirDrop sharing
- **Performance**: 120Hz ProMotion support, Metal rendering for charts

Web Compatibility

- **Responsive breakpoints**: Desktop (>1200px), tablet (768-1200px), mobile (<768px)
- **PWA support**: Service worker for offline web, deep linking, shareable URLs
- **Limitations**: Performance not as smooth as iOS, limited file system access, some plugins unsupported

Security Implementation

- **Token storage**: JWT tokens in iOS Keychain via flutter_secure_storage
- **Encryption**: AES-256 for local draft data, keys stored separately in Keychain
- **Session management**: 15-minute idle timeout, auto-refresh JWT, multi-device sync support
- **Certificate pinning**: Pin AWS backend SSL certificates to prevent MITM attacks

- **Input validation**: Client and server-side validation, prevent XSS/SQL injection

Key Dependencies

State: riverpod ^2.4.0 | **Networking**: dio ^5.4.0, retrofit ^4.0.0 | **Database**: drift ^2.14.0 | **Security**: flutter_secure_storage ^9.0.0, local_auth ^2.1.7, encrypt ^5.0.3 | **PDF**: file_picker ^6.1.1, syncfusion_flutter_pdfviewer ^24.1.41, pdf ^3.10.7 | **Charts**: fl_chart ^0.66.0 | **Utilities**: connectivity_plus ^5.0.2, workmanager ^0.5.1

Development Workflow

- **Code generation**: build_runner for drift, json_serializable, freezed models
- **Testing**: Widget tests, integration tests, golden tests, mockito for API mocking
- **CI/CD**: Codemagic/GitHub Actions for iOS (TestFlight) and Web (Firebase Hosting)
- **Flavors**: Development, staging, production environments

Critical Implementation Notes

- **Offline priority**: All UI operations work without network, sync happens in background
- **Pricing calculations**: Backend is source of truth, local cache for offline, re-validate on sync
- **Data wiping**: Complete local data wipe on logout or user revocation
- **Development strategy**: Build iPad version first (primary), then optimize for web (secondary)

11) Summary

This architecture provides a comprehensive full-stack solution combining a secure Django backend with a Flutter cross-platform client. The backend handles extraction pipeline, pricing engine, and administrative controls, while the Flutter client delivers an iPad-first, web-compatible experience with offline-first capabilities. The system ensures scalability, security, and clarity while supporting automation where possible, manual input where required, and provides a consistent, auditable process for generating merchant pricing proposals and savings analyses.