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.

Scope, Assumptions, and Non-Goals

In Scope

- Merchant statement ingestion (upload + manual entry) and normalized storage.
- Secure, role-based access to analyses and artifacts.
- Deterministic pricing/savings calculations (target end-state).
- Proposal artifact generation (PDF) (target end-state).

Assumptions

- Statements can contain sensitive business financial data and must be treated as confidential.
- The backend is the system of record; frontend apps (web/mobile) are thin shells that cache drafts.
- Processor statement formats vary; extraction quality is improved via iteration and per-processor extractors.

Non-Goals (for this architecture document)

- Defining Blockpay's proprietary pricing tables in this document (these belong in admin-configured catalogs).
- Claiming PCI compliance certification; the system is designed to support PCI-aligned security practices, but certification depends on deployment and operational controls.

Key User Flows (End-to-End)

1) Statement → Reviewable normalized data

1. Agent authenticates and receives JWT.
2. Agent uploads a statement (PDF preferred; images supported with OCR).
3. Backend stores the document securely and runs processor detection + extraction.
4. Backend returns normalized data + confidence score; low-confidence fields are flagged for review.

2) Manual entry (no statement available)

1. Agent enters required volumes/fees and statement dates.
2. Backend stores the same normalized schema as the extraction pipeline.

3) Review → Pricing → Proposal PDF

1. Agent reviews extracted fields and corrects flagged values.
2. Agent selects a pricing model and optional devices/SaaS bundles.
3. Backend computes savings and stores a complete, versioned snapshot for auditability.
4. Backend generates a branded proposal PDF and provides a secure, time-limited download link.

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 via API and stored (local media in dev; S3 recommended for production).
- PDF ingestion supports PDF and images (with OCR for images).
- Text extraction uses `pdfplumber` and/or OCR depending on source.

- Processor detection routes to processor-specific extractors (Chase, Clover, Square, Stripe, etc.).
- Hybrid approach: deterministic parsing (regex + table extraction) plus optional LLM-assisted structuring for variable formats.
- Key fields (rates, fees, volumes, transaction counts) normalized into a standard schema.
- Confidence scores assigned to extracted fields; low-confidence values flagged for review and correction.
- Extraction runs asynchronously (job queue) with retries, timeouts, and operator visibility.

Upload & Storage (Recommended)

- **Upload validation**:
  - MIME sniffing + extension allowlist (PDF/JPEG/PNG)
  - max file size policy (configurable)
  - content sanitization and safe PDF parsing
- **Storage**:
  - Store original file in a **private** bucket (S3) with SSE-KMS encryption
  - Store derived artifacts (text, structured JSON, thumbnails) separately with retention controls
  - Use object keys that do not embed PII (UUID-based paths)
- **Access**:
  - Downloads via time-limited signed URLs
  - "Owner or admin" authorization on every file access

Async Extraction Pipeline (Recommended)

- **Job model**: `ExtractionJob` with states `QUEUED → RUNNING → SUCCEEDED|FAILED|NEEDS_REVIEW`
- **Workers**: background workers (Celery) process jobs; API remains responsive
- **Retries**:
  - transient errors: retry with exponential backoff
  - hard failures: mark failed and route to manual entry/review
  - optional DLQ bucket/queue for forensic inspection
- **Timeouts & limits**:
  - per-document page limit (configurable)
  - per-job timeout
  - concurrency limits by tenant/user to control cost

Normalization & Confidence

- Normalize into a stable schema (`StatementData`) regardless of processor.
- Track provenance per value:

    - source (text/table/LLM/manual)
    - confidence score (0–1 or 0–100)
    - raw snippet references (page number / line ranges where feasible)
- Flag review:
    - field-level flags (e.g., missing totals, mismatched volume vs sum of brands)
    - document-level `requires_review` if any critical field is low-confidence

Review & Correction Workflow

- Agents can:
    - view extracted values, confidence, and evidence
    - correct values and provide notes
- System stores:
    - original extracted value(s)
    - corrected value(s)
    - who changed it, when, and why (audit trail)

Processor Support Strategy

- Factory-based detection routes to per-processor extractors.
- New processor onboarding requires:
    - sample statement set (redacted)
    - extractor implementation + unit tests
    - confidence calibration and validation checks
    - rollout behind a feature flag per tenant (optional)

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.

Pricing Model Plug-in Interface

To support multiple pricing models without hard-coding logic throughout the app, the pricing engine treats each model as a plug-in with the same interface:

- **Inputs**:
  - normalized merchant dataset (volumes, counts, fee breakdown, effective rate, optional card mix)
  - selected pricing model + parameters
  - selected device/package bundle (optional)
  - admin-managed catalogs (pricing rules, device costs, fee items, disclaimers)
- **Outputs**:
  - proposed total cost (monthly and annualized views)
  - line-item breakdown (percent fees, per-item fees, monthly fees, one-time fees, device costs)
  - computed rates (effective rate, blended components)
  - warnings/assumptions used (e.g., "card mix unknown; used blended assumptions")

Supported Pricing Models (Structure)

The system supports these models by representing pricing as a combination of reusable fee components:

- **Cost-plus (Interchange+ / pass-through)**:
  - pass-through costs (interchange + assessments/network fees) + markup components (basis points and/or per-item)
- **iPlus (business-defined)**:

    - treated as a configurable "interchange-plus variant" composed of tiered markups/fees; the exact tiers/rules are configured in admin catalogs and documented in the proposal template
- **Discount rate**:
    - percent-of-volume discount rate + per-item fee + monthly fees (as configured)
- **Flat**:
    - flat percent-of-volume and/or flat per-item fee + monthly fees (as configured)

Fee Catalog & Customization (Data Model)

All fees are stored as catalog items and attached to proposals via rules, not hard-coded:

- **FeeItem** (reusable building block):
    - `type` : `PERCENT_VOLUME` | `PER_ITEM` | `MONTHLY` | `ONE_TIME`
    - `amount` : decimal (percent stored as basis points or decimal percent)
    - `applies_to` : filters (card brand, debit/credit, present/not-present, entry mode, etc.)
    - `effective_from` / `effective_to` for versioning
    - `rounding_rule` : banker's rounding vs standard, and decimal places
- **FeeGroup** (a named bundle):
    - "Network fees", "Monthly program fees", "Surcharge program fees", etc.
- **DeviceCatalogItem**:
    - internal device cost, MSRP, financing terms (if applicable), and whether included in ROI calculations

Qualified / Non-Qualified and Interchange Detail

Different processors report fees differently. The engine supports both:

- **Interchange detail available**: store categories and compute pass-through precisely
- **Only qualified/non-qualified or blended data available**: compute using configured assumptions and clearly disclose them in proposal outputs

Surcharge Programs

Surcharge programs are modeled explicitly so their impact is auditable:

- Eligibility rules (MCC/state/brand restrictions as configured)
- Caps and compliance settings (configured)
- Separate line items for surcharge program fees, and clear disclosure language in the proposal PDF
- Savings calculation options:
    - "gross savings" (before surcharge revenue)
    - "net" view including surcharge revenue assumptions (must be clearly labeled)

Savings Math, Timeframes, and Chart Data

All savings are computed from the same normalized monthly baseline:

- **Baseline period**: statement period; normalize to monthly where possible
- **Timeframes**:
  - daily = monthly / 30 (configurable: 30-day month assumption or actual days-in-period)
  - weekly = daily * 7
  - monthly = normalized monthly
  - quarterly = monthly * 3
  - yearly = monthly * 12
- **Chart dataset contract**:
  - backend returns arrays of `{label, current_cost, proposed_cost, savings}` for each timeframe
  - the app renders bar/line charts consistently across iPad and web

Financial Accuracy & Auditability

- All money values are stored as decimals (no floats).
- Consistent rounding rules applied per fee item and at total levels.
- Every proposal stores a **PricingSnapshot**:
  - inputs (statement data version, chosen pricing rules, device items, assumptions)
  - outputs (full line-item breakdown + totals)
  - template version used for PDF
  - immutable once submitted (only superseded by a new version)

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.

Proposal Template & Versioning

- Proposal PDFs are generated from versioned templates:
  - branding assets (logos/colors)
  - disclaimer blocks (jurisdictional variants as needed)
  - configurable sections (devices, SaaS, pricing model explanation)
- Each generated PDF records:
  - template version
  - pricing snapshot version
  - generation timestamp and requesting user

Secure Delivery

- PDFs stored in private object storage (S3) with SSE-KMS.
- Access via time-bound signed URLs (and optional single-use links).
- Optional watermarking ("Confidential", agent name, date) to reduce leakage risk.

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

Admin Configuration Domains

- Pricing rule catalogs (per model; effective dates; per-tenant overrides)
- Fee catalogs (monthly/one-time/per-item/percent) and groupings
- Device and SaaS catalogs (costs, bundles, promo rules)
- Proposal templates, disclaimers, branding assets
- Access control policies (role permissions, MFA enforcement, retention)

Versioning & Change Control

- All catalogs are versioned with effective dates.
- Pricing snapshots always reference a specific catalog version so historical proposals never change.
- Optional approval workflow for pricing changes (draft → approved → effective).

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

Security Notes

- JWT authentication (access + refresh) with rotation and short-lived access tokens.
- Role-based access control (Admin/Agent) and strict per-tenant/user data boundaries.
- MFA capability and policy enforcement for privileged users.
- WAF + rate limits + request size limits at the edge; defense-in-depth validation in-app.
- Immutable audit logging (append-only) with retention policies.
- Secret management (AWS Secrets Manager/SSM) and strict key rotation policies.
- Private file storage with time-bound signed download URLs (S3 presigned).

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

Authentication & Session Policy (Recommended Defaults)

- Access tokens: short-lived (e.g., 15 minutes)
- Refresh tokens: longer-lived with rotation; revoke on logout and suspected compromise
- Device/session tracking: record device identifiers (app-generated) and last activity
- MFA:
  - TOTP for admins by default
  - optional WebAuthn/passkeys for high-assurance deployments

Data Isolation

- Add `tenant_id` (organization) to all business objects and enforce it at the query layer.
- Ensure object-level authorization checks on every read/write, including signed URL issuance.

Secure File Handling

- Validate file type and reject active content.
- Virus/malware scanning for uploads in production (optional but recommended).
- Strip metadata where appropriate (e.g., embedded PDF metadata), and avoid leaking PII in filenames/paths.

6.1) Deployment Topology (Recommended for Production)

Core Components

- **API service**: Django (Gunicorn) behind an HTTPS load balancer.
- **Database**: PostgreSQL (AWS RDS recommended).

- **File storage**: S3 (private bucket) for uploaded statements and generated proposals.
- **Cache/broker**: Redis (for rate limiting, caching, and Celery broker if used).
- **Workers (planned)**: Celery workers for statement extraction and PDF generation.

Network & Edge Security

- TLS termination at the load balancer; enforce HSTS.
- WAF rules for common attacks and request size limits.
- Private subnets for DB/Redis; security groups restrict ingress.

Environments

- **Development**: Local SQLite optional; local media storage; debug tooling.
- **Staging/Production**: PostgreSQL + S3 + Redis; production settings; tighter CORS; secret management.

6.2) Observability & Auditability

Application Logging

- Structured logs with request ID, user ID, statement/proposal IDs.
- Redaction rules to prevent sensitive statement data leaking into logs.

Metrics (recommended)

- API latency (p50/p95/p99), error rate, and throughput.
- Extraction job duration, success/failure rates, and "requires_review" rate by processor.
- Storage usage by tenant/user and document counts.

Tracing (recommended)

- Trace request → extraction → DB writes → artifact generation for debugging and compliance investigations.

Audit Logs (planned)

- Append-only audit events for: login/logout, uploads, edits, proposal generation, downloads, admin changes.
- Retention policy and export tooling for customer/compliance needs.

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

Sync Contract (What Syncs)

- **Draft Analysis** (editable): statement corrections, pricing scenario selections, notes
- **Attachments**: statement PDFs/images (uploaded once, referenced by ID; chunked/resumable upload)
- **Catalog snapshots**: minimal read-only bundles needed to price offline (devices/fees/pricing rules), signed and versioned

Conflict Resolution (Recommended)

- Use optimistic concurrency:
  - each mutable object carries `version` (integer) and `updated_at`
  - the app sends `If-Match` style version; server rejects conflicting updates with a merge-required response
- For drafts, support "last write wins" only when changes are non-overlapping; otherwise prompt the user to reconcile.

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

Background Jobs (Production)

- **Celery 5.3.6** - Async job processing (planned usage)
- **Redis 5.0.1** - Broker/cache (planned usage)

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** - LLM integration scaffold (planned usage)
- **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 app (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)
- **Networking (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

Deployment Runtime (Production)

- **gunicorn 21.2.0** - WSGI server
- **whitenoise 6.6.0** - Static file serving (if not using CDN)
- **django-storages 1.14.2 + boto3 1.34.30** - S3 media/static integration (recommended)
- **sentry-sdk 1.40.0** - Error monitoring (recommended)

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 App Architecture

The Flutter app 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
- **Networking**: 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**: App 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 app. The backend handles extraction pipeline, pricing engine, and administrative controls, while the Flutter app 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.

Appendix A: Proposed API Surface (High-Level)

Authentication (`/api/v1/auth/`)

- Register/login/logout, token refresh
- Profile and password management

- Admin user management (create/disable/revoke)
- MFA enrollment and challenge endpoints (if required)

Statements & Review (`/api/v1/statements/`)

- Upload statement (async processing)
- Manual entry
- Get extraction results and review flags
- Update/correct extracted fields

Jobs (`/api/v1/jobs/`)

- Get job status and progress (extraction/PDF generation)
- Admin visibility into failure reasons and retries

Analyses & Pricing (`/api/v1/analyses/`)

- Create analysis from a statement or manual dataset
- Select pricing model + parameters
- Compute savings and generate a versioned pricing snapshot
- Compare multiple proposal scenarios for the same merchant

Charts (`/api/v1/charts/`)

- Return chart-ready datasets for timeframe comparisons and scenario comparisons

Proposals (`/api/v1/proposals/`)

- Generate proposal PDF from a pricing snapshot
- Download via signed URL (time-limited)
- Track proposal status and audit events

Admin Configuration (`/api/v1/admin/`)

- Manage pricing rule catalogs, device catalogs, fees, and disclaimers/templates
- Reporting (usage, extraction success rates, proposal volume)
- Audit log search/export

Appendix B: Target Data Model (Core Entities)

Identity & Access

- `User` (role, org/tenant, MFA status, auth metadata)
- `AuditEvent` (append-only security/business events)

Documents & Extraction

- `MerchantStatement` (source, file refs, processor, period, status)
- `StatementData` (normalized volumes/counts/fee buckets + raw extraction)
- `ExtractionField` (optional)(per-field value + confidence + review status + provenance)
- `ExtractionJob` (async processing metadata: state, retries, durations, error codes)

Pricing & Proposals

- `Analysis` (ties a merchant dataset to one or more proposal scenarios)
- `PricingModel` / `PricingScenario` (selected model + parameters)
- `PricingSnapshot` (versioned inputs/outputs; immutable after submission)
- `Proposal` (PDF artifact ref + disclaimers + generation metadata)

Admin Catalogs

- `PricingRule` (rate tables/markups by model)
- `DeviceCatalogItem` (devices/accessories + internal costs)
- `FeeCatalogItem` (monthly/one-time/misc fees)
- `Template` (proposal templates, disclaimers, branding)

Appendix C: Pricing Model Parameterization (Inputs & Configuration)

The architecture supports the models listed, but the exact business rules and formulas must be confirmed and documented as structured configuration (admin-managed catalogs + app-supplied inputs):

- For each model, define:
  - required inputs (volume, tx count, card mix, risk flags)
  - rate components (bps markup, per-item, monthly, one-time)
  - tier rules (if any) and effective dates
  - rounding rules and minimums

    - disclosure text to include in the proposal PDF
- The implementation should include a “calculation trace” object per proposal that records how each line item was derived, to support financial accuracy reviews.