Bank Statement Parser App - Complete Development Plan

Project Goal: Build a monetizable desktop application that allows users to upload bank statement PDFs and automatically extract transaction data using Claude AI, exportable to Excel.

Target: Enterprise-grade security, zero hardcoded values, full database-driven architecture.

---

1. SECURITY ARCHITECTURE

1.1 API Key Management

- [ ] Implement secure API key vault (encrypted storage in database)
  - Never store API keys in plaintext
  - Use AES-256 encryption for stored keys
  - Keys should be per-user (each user provides their own Claude API key)
  - Add option for admin-provided backend key (for SaaS model later)
- [ ] Create key rotation mechanism
- [ ] Implement API key validation before allowing usage
- [ ] Add audit logging for all API key access attempts
- [ ] Display API key only once on creation (users can't retrieve it later)

1.2 Authentication & Authorization

- [ ] User registration/login system
  - Email + strong password requirements
  - Optional two-factor authentication (2FA)
  - Bcrypt for password hashing (min 12 rounds)
- [ ] User role management (Admin, User)
- [ ] Session management with secure tokens (JWT with expiration)
- [ ] Password reset flow with email verification
- [ ] Account lockout after failed login attempts (5 attempts, 30min lockout)

1.3 File Security

- [ ] Validate uploaded file types (only PDFs)
- [ ] Implement file size limits (configurable, e.g., 50MB max)
- [ ] Scan for malicious content using antivirus library (e.g., ClamAV integration)
- [ ] Quarantine suspicious files
- [ ] Implement secure file storage (separate directory, restricted access)
- [ ] Delete original PDF files after processing (user choice: delete immediately or keep for 30 days)

- [ ] Add file integrity checks (hash verification)

1.4 Data Transmission Security

- [ ] Use HTTPS/TLS for all communications
- [ ] Implement certificate pinning for Claude API calls
- [ ] Sign sensitive data exchanges with HMAC
- [ ] Rate limiting on API endpoints (prevent brute force)
- [ ] Input sanitization for all user-provided data
- [ ] CSRF protection for web endpoints

1.5 Code Security

- [ ] Use environment variables for all sensitive config (never in code)
- [ ] Implement dependency scanning (outdated/vulnerable packages)
- [ ] Code signing for distributed executables
- [ ] Obfuscation for sensitive logic
- [ ] Remove debug logs in production builds

---

2. DATA PROTECTION & PRIVACY

2.1 Regulatory Compliance

- [ ] **GDPR Compliance** (EU users)
  - Privacy policy clearly stating data handling
  - User consent before processing bank data
  - Right to be forgotten implementation
  - Data processing agreements
  - Data breach notification plan
- [ ] **CCPA Compliance** (California users)
  - Right to access, delete, opt-out
- [ ] **Banking Regulations**
  - Check if handling financial data requires special licensing/registration
  - Comply with PSD2 (Payment Services Directive) if applicable

2.2 Data Encryption

- [ ] Encrypt sensitive data at rest (database)
  - Transaction data: encrypted with user-specific key
  - Bank account numbers: encrypted

- Personal identifiers: encrypted

- [ ] Encrypt data in transit (all network calls over TLS)
- [ ] Separate encryption keys per user (derive from master key)
- [ ] Key escrow/recovery mechanism (users can never access data if key is lost)

2.3 Data Retention

- [ ] Implement configurable data retention policies
  - Default: keep for 7 years (standard accounting requirement)
  - User can delete specific statements
  - Automatic deletion based on age
- [ ] Create audit trail for all deletions
- [ ] Implement soft deletes initially (mark as deleted, hard delete after 90 days)

2.4 User Consent & Transparency

- [ ] Terms of Service document (cover liability, data handling)
- [ ] Privacy Policy (explain Claude API usage, data storage, retention)
- [ ] Show user explicitly what data will be sent to Claude API
- [ ] Allow users to review and approve extracted data before storage
- [ ] Implement consent logging (when user agreed to what)

2.5 Audit Logging

- [ ] Log all data access (who, when, what was accessed)
- [ ] Log all processing events (file upload, API calls, exports)
- [ ] Log all authentication events (login, logout, failed attempts)
- [ ] Log all data modifications/deletions
- [ ] Immutable audit logs (cannot be modified after creation)
- [ ] Retention: keep audit logs for minimum 2 years
- [ ] Option to export audit logs for compliance

---

3. DATABASE ARCHITECTURE

3.1 Database Schema (Core Tables)

users

- user_id (UUID, primary key)
- email (unique, indexed)
- password_hash (bcrypt)
- first_name, last_name

- role (enum: admin, user)
- api_key_id (FK to api_keys)
- created_at, updated_at
- last_login
- is_active (soft delete support)

api_keys

- api_key_id (UUID, primary key)
- user_id (FK, indexed)
- encrypted_key (AES-256 encrypted)
- key_alias (user-friendly name, e.g., "My Claude Key")
- is_active
- last_used
- created_at, expires_at (optional)
- ip_whitelist (JSON, optional)

statements

- statement_id (UUID, primary key)
- user_id (FK, indexed)
- original_filename
- file_hash (SHA-256)
- file_size_bytes
- upload_date
- processing_status (enum: pending, processing, completed, failed, quarantined)
- processing_started_at, processing_completed_at
- error_message (if failed)
- bank_name (extracted by AI, nullable initially)
- account_number_masked (last 4 digits only)
- statement_period_start, statement_period_end
- currency (extracted)

- retention_delete_date (when to auto-delete)
- is_deleted (soft delete)
- created_at

transactions

- transaction_id (UUID, primary key)
- statement_id (FK, indexed)
- user_id (FK, indexed)
- transaction_date (indexed)
- amount (decimal, encrypted at rest)
- currency
- description (encrypted)
- transaction_type (enum: debit, credit, transfer, etc.)
- merchant_name (encrypted)
- category (nullable, for user tagging)
- duplicate_group_id (UUID, for duplicate detection)
- confidence_score (0-100, AI confidence in extraction)
- ai_response_metadata (JSON, raw Claude response for debugging)
- is_deleted (soft delete)
- created_at, updated_at

processing_logs

- log_id (UUID, primary key)
- statement_id (FK)
- user_id (FK)
- processing_step (enum: pdf_extraction, ai_analysis, validation, storage)
- status (enum: success, warning, error)
- details (JSON)
- timestamp (indexed)

- tokens_used (for API cost tracking)
- duration_ms

audit_logs (immutable)

- log_id (UUID, primary key, immutable)
- user_id (FK, indexed)
- action_type (enum: login, data_access, data_modification, data_deletion, export, settings_change)
- resource_type (enum: statement, transaction, user, api_key)
- resource_id (FK, nullable)
- old_value (JSON, for modifications)
- new_value (JSON, for modifications)
- ip_address
- user_agent
- timestamp (indexed)
- data_classification (enum: public, internal, confidential, restricted)

settings

- setting_id (UUID, primary key)
- user_id (FK, indexed, nullable for global settings)
- setting_key (indexed, e.g., "retention_days", "auto_delete_enabled")
- setting_value (JSON, polymorphic)
- is_encrypted (boolean, flag to encrypt sensitive settings)
- updated_at

exports

- export_id (UUID, primary key)
- user_id (FK, indexed)
- export_type (enum: excel, csv, json)
- statement_ids (JSON array of UUIDs)
- filter_criteria (JSON, what filters were applied)
- file_hash (SHA-256)

- size_bytes
- created_at
- is_deleted (soft delete for cleanup)

api_call_logs (for cost tracking & rate limiting)

- call_id (UUID, primary key)
- user_id (FK, indexed)
- statement_id (FK)
- api_provider (enum: claude)
- model_name (e.g., "claude-3-opus")
- tokens_input
- tokens_output
- cost (calculated)
- status (success/error)
- timestamp (indexed)

duplicate_groups

- duplicate_group_id (UUID, primary key)
- statement_id (FK)
- transaction_ids (JSON array)
- confidence_score (how confident this is a duplicate)
- user_reviewed (boolean)
- user_decision (enum: keep_all, keep_first, keep_most_recent, custom)
- created_at

user_preferences

- preference_id (UUID, primary key)
- user_id (FK, unique)
- language (enum: en, de, fr)
- date_format

- number_format (decimal separator, thousands separator)
- theme (enum: light, dark)
- notifications_enabled
- auto_categorize (boolean)
- show_confidence_scores (boolean)
- updated_at

3.2 Database Indexing Strategy

- [ ] Index on frequently queried columns (user_id, created_at, processing_status)
- [ ] Composite indexes for common queries (user_id + created_at)
- [ ] Index on soft-delete columns to exclude deleted records
- [ ] Consider partitioning large tables (transactions) by user_id or date
- [ ] Regular index maintenance/rebuilding

3.3 Database Security

- [ ] Use parameterized queries (prevent SQL injection)
- [ ] Principle of least privilege (DB user has only needed permissions)
- [ ] Database-level encryption (if supported)
- [ ] Regular backups (encrypted, tested recovery)
- [ ] Connection pooling with timeouts
- [ ] No default credentials

---

4. EFFICIENCY & PERFORMANCE

4.1 PDF Processing

- [ ] Implement efficient PDF text extraction (use `pdfjs-dist` with streaming)
- [ ] Handle large PDFs without blocking UI (worker threads in Node.js)
- [ ] Implement progressive PDF parsing (extract in chunks)
- [ ] Cache extracted text for re-processing requests
- [ ] Add timeout on API calls (max 60 seconds per request)

4.2 AI Integration Optimization

- [ ] Batch process statements (send multiple statements in single API call if possible, respecting token limits)
- [ ] Implement prompt caching (reuse same system prompt)
- [ ] Smart token usage (limit response format, be specific in prompts)
- [ ] Fall back to simpler models for partial extraction (cost optimization)
- [ ] Queue system for processing (use Bull/RabbitMQ for scalability)

4.3 Database Performance

- [ ] Connection pooling (prevent connection exhaustion)
- [ ] Query optimization (EXPLAIN plans)
- [ ] Pagination for list views (never fetch all records at once)
- [ ] Lazy loading for related data
- [ ] Caching layer (Redis for frequently accessed data)
  - Cache user settings
  - Cache API key status
  - Cache summary statistics

4.4 UI/Rendering Optimization

- [ ] Virtualized lists (only render visible rows in transaction tables)
- [ ] Debounce search/filter inputs (avoid excessive queries)
- [ ] Lazy load table columns (don't load all data upfront)
- [ ] Progressive table rendering (header first, then data)

4.5 API Rate Limiting

- [ ] Implement rate limiting per user (e.g., 100 statements/day for free tier)
- [ ] Implement Claude API rate limiting (respect their limits)
- [ ] Queue excess requests instead of rejecting
- [ ] Provide clear feedback on rate limit status

---

5. USER EXPERIENCE

5.1 Core Workflows

- [ ] **Upload & Processing**
  - Drag-and-drop interface (or file picker)
  - Show file preview (first page of PDF)
  - Display upload progress
  - Real-time processing status updates (via WebSocket or polling)
  - Show when statement is being sent to Claude API
  - Show extraction confidence scores
  - Estimated time remaining
- [ ] **Data Review & Confirmation**
  - Show extracted data in editable table before saving

- Highlight low-confidence entries
- Allow inline editing of transactions
- Show side-by-side comparison: original PDF vs extracted data
- One-click approve/reject entire statement
- Add transactions not found by AI

☐ **Export**

- Select statements to export
- Choose export format (Excel with formatting, CSV, JSON)
- Apply filters (date range, transaction type, amount range)
- Custom column selection
- Batch export multiple statements
- Download automatically or email to user

5.2 Dashboard

☐ **Statistics**

- Total statements processed
- Total transactions extracted
- Date range of statements
- Accounts/banks represented
- Total money flowing in/out

☐ **Quick Actions**

- Recent statements
- Pending uploads
- Upcoming auto-deletions

☐ **Activity**

- Timeline of recent uploads
- Processing queue status

5.3 Statement Management

☐ View all uploaded statements in list/grid view

☐ Sort by upload date, bank, status

☐ Filter by status (pending, completed, failed), date range

☐ Bulk operations (delete multiple, re-process, export together)

- [ ] Statement details view (metadata, all transactions, original PDF preview)
- [ ] Duplicate detection (show when same statement uploaded twice)

5.4 Transaction Management

- [ ] View all transactions across statements
- [ ] Filter by date, amount, type, merchant
- [ ] Search transactions (free text)
- [ ] Categorize transactions (optional tagging)
- [ ] Flag suspicious transactions (unusual amounts, etc.)
- [ ] Manual correction interface (fix AI errors)
- [ ] Transaction merge (if duplicates are found)

5.5 Error Handling & User Feedback

- [ ] Clear, non-technical error messages
- [ ] Explain what went wrong and suggested fixes
- [ ] "Contact support" button with pre-filled context
- [ ] Error logging to help you debug
- [ ] Warnings for common issues (PDFs from unsupported banks, corrupted files)
- [ ] Success notifications with next steps

5.6 Settings & Preferences

- [ ] API key management (add/edit/remove keys, test connection)
- [ ] Data retention policy (how long to keep statements)
- [ ] Auto-delete settings (delete after 90 days, etc.)
- [ ] Export preferences (default format, location)
- [ ] Language/localization (English, German, French)
- [ ] Date/number format preferences
- [ ] Notification preferences (email on completion, errors)
- [ ] Theme (dark/light mode)

5.7 Help & Support

- [ ] In-app help tooltips
- [ ] FAQ section
- [ ] Video tutorials for key workflows
- [ ] Contact support form (pre-fill with system info)
- [ ] Knowledge base/docs

---

6. SCALABILITY

6.1 Architecture for Growth

- [ ] **Microservices-ready** (don't build monolith)
  - PDF processing service (separate from main app)
  - AI extraction service
  - Export service
  - Notification service
- [ ] **API-first design** (Electron app calls APIs, not direct DB)
  - Easier to scale backend later
  - Easier to build web/mobile versions
- [ ] **Asynchronous processing**
  - Use job queues (Bull with Redis) for long-running tasks
  - PDF processing doesn't block user
  - Batch AI requests
  - Scheduled tasks (auto-delete old statements)

6.2 Database Scalability

- [ ] Plan for database replication (master-slave for read scaling)
- [ ] Plan table partitioning (statements/transactions by date or user_id)
- [ ] Archive old data (move statements >7 years to cold storage)
- [ ] Consider database migration path (PostgreSQL → cloud DB)

6.3 Server Infrastructure

- [ ] Load balancing (multiple API instances)
- [ ] Horizontal scaling (add more workers for PDF processing)
- [ ] Caching layer (Redis)
- [ ] CDN for static assets
- [ ] Database clustering

6.4 Monitoring & Observability

- [ ] Application performance monitoring (APM)
  - Track slow queries
  - Track slow API endpoints
  - Track Claude API response times
- [ ] Error tracking (Sentry, LogRocket)

- [ ] Logging strategy (structured logs to centralized system)
- [ ] Health checks for all services
- [ ] Alerts for critical issues

---

7. TECHNOLOGY STACK (Detailed)

7.1 Desktop App (Electron)

- Framework: Electron 28+
- UI: React 18 + TypeScript
- State: Zustand (lightweight, simple)
- HTTP: axios with interceptors (auth, error handling)
- PDF: pdfjs-dist (extraction)
- OCR: tesseract.js (for scanned statements)
- Excel Export: exceljs (formatting support)
- Security: electron-secure-preferences (encrypted storage)
- Auto-update: electron-updater
- Database (local): better-sqlite3 or electron-sqlite3
- Styling: Tailwind CSS
- Icons: Lucide React

7.2 Backend (if needed for SaaS model)

- Runtime: Node.js 20+
- Framework: Express or Fastify
- Database: PostgreSQL 15+ (primary)
- Caching: Redis
- Job Queue: Bull (Redis-backed)
- Authentication: jsonwebtoken (JWT)
- Validation: Zod or Joi
- ORM: Prisma (type-safe, migrations)
- API Documentation: Swagger/OpenAPI
- Testing: Jest + Supertest
- Monitoring: Prometheus + Grafana

7.3 Supporting Services

- File Storage: Cloud (AWS S3, GCP, Azure Blob) OR local encrypted storage
- Email: SendGrid or similar (for notifications)
- Antivirus: ClamAV (open source)
- Secrets Management: HashiCorp Vault (production)

- CI/CD: GitHub Actions
- Code Quality: SonarQube

---

8. MONETIZATION ARCHITECTURE

8.1 Licensing & Trial System

☐ License key generation (unique per machine)

- Hardware ID based (prevent sharing)
- Time-limited (30-day trial default)
- Feature-limited tiers (free/pro/enterprise)

☐ License validation

- Offline validation (cached license check)
- Online validation (periodically verify with server)
- Graceful degradation (work offline, validate when online)

☐ Trial tracking

- Track trial start date
- Track feature usage (statements processed, transactions exported)
- Show countdown (X days left in trial)

☐ Upgrade flow

- Show "upgrade to pro" prompts at appropriate times
- Easy payment integration (Stripe, PayPal)
- Automatic license upgrade after payment

8.2 Usage Tracking (for billing/analytics)

☐ Track API call costs (tokens sent to Claude)

- Store in api_call_logs
- Calculate cost per statement
- Show cost breakdown to user

☐ Track feature usage

- Statements processed
- Transactions exported
- Storage used

☐ Analytics (for business intelligence)

- Popular export formats
- Most problematic banks
- Error rates per bank

8.3 Pricing Tiers (example structure)

```
- Free: 5 statements/month, up to 100 transactions, basic support
- Pro: Unlimited statements, advanced analytics, priority support
- Enterprise: Custom pricing, custom integrations, SLA
```

---

9. IMPORTANT CONSIDERATIONS & IMPROVEMENTS

9.1 Multi-User Support

☐ Consider building multi-user account support early

- Invite family members/accounting team
- Role-based permissions (view, edit, export)
- Shared account linking
- Different audit trails per user

9.2 Bank Integration (Future)

☐ Research if you should integrate with bank APIs (Plaid, etc.)

- Pros: Real-time data, more accurate
- Cons: Requires bank partnership, more complex, regulatory burden
- Decision: Start with PDF parsing, add API integration as premium feature later

9.3 Duplicate & Reconciliation

☐ Implement duplicate transaction detection

- Hash similar transactions (date, amount, merchant)
- Show confidence score
- Allow user to merge/keep duplicate

☐ Cross-statement reconciliation

- Show which transactions appear in multiple statements
- Flag missing transactions

9.4 Data Validation & Anomaly Detection

☐ Validate extracted amounts (reasonable currency ranges)

- [ ] Validate dates (within statement period)
- [ ] Flag unusual patterns (sudden large transactions, many small ones)
- [ ] Suggest potential data entry errors

9.5 Mobile Companion (Future)

- [ ] Plan for mobile app (React Native)
  - View statements on phone
  - Capture receipts with camera
  - Approve extracted data on-the-go
  - Still encrypted end-to-end

9.6 AI Prompt Optimization

- [ ] Build a good system prompt that:
  - Specifies exact JSON output format
  - Handles multiple statement formats
  - Requests confidence scores
  - Asks for flagged fields where AI is unsure
  - Handles edge cases (multi-currency, converted amounts)
- [ ] Test with real bank statements and iterate

9.7 Backup & Disaster Recovery

- [ ] Automatic backups (encrypted)
  - Database backups daily
  - File backups to cloud
- [ ] User-initiated backups
  - "Backup my data" button → encrypted zip file
- [ ] Recovery plan
  - Users can restore from backup
  - Plan for account recovery if local data is lost

9.8 Internationalization

- [ ] Support multiple languages (English, German, French initially)
  - All UI text in database/config
  - Date/number/currency formatting per locale
  - Separate bank formats per country

- [ ] Support multiple currencies
  - Show currency with amounts
  - Support multi-currency statements
  - Optional conversion display

9.9 Compliance & Legal

- [ ] **Terms of Service**
  - Limitation of liability
  - Prohibition of reverse engineering
  - Acceptable use policy
  - Dispute resolution
- [ ] **Privacy Policy**
  - Explain Claude API data usage
  - Explain local vs cloud storage
  - Data processing agreements for EU users
- [ ] **EULA for Software**
  - License grant (personal/commercial use)
  - Restrictions
  - Warranty disclaimer
  - Support/updates terms
- [ ] **Financial Services Compliance**
  - Check if you need disclaimers (not financial advice)
  - Check liability insurance needs
  - Check regulatory registration requirements (varies by jurisdiction)

9.10 Support Infrastructure

- [ ] Build support system
  - Help documentation (searchable)
  - FAQ with search
  - Email support with templated responses
  - Bug report form (pre-fill system info)
  - Feature request tracker
- [ ] Create knowledge base

- How to export data
- Troubleshooting common issues
- Video tutorials
- Supported banks list

9.11 Version Management & Updates

- [ ] Auto-update mechanism
  - Check for updates daily
  - Notify user of available updates
  - Background download
  - Restart required (minimal disruption)
- [ ] Update strategy
  - Semantic versioning
  - Changelog visible to users
  - Critical security patches forced
  - Optional feature updates

9.12 Testing Strategy

- [ ] Unit tests (business logic, data extraction)
- [ ] Integration tests (AI API, export functions)
- [ ] E2E tests (full workflows with real PDFs)
- [ ] Security tests (penetration testing, vulnerability scanning)
- [ ] Performance tests (large statement processing, export speed)
- [ ] Compatibility tests (different Windows/Mac/Linux versions)

9.13 Launch & Distribution

- [ ] Code signing (Windows, Mac)
  - Prevent security warnings on install
  - Build trust with users
- [ ] Distribution channels
  - Direct download from website
  - Windows Store (optional)
  - Mac App Store (requires changes)
  - Installer customization (company name, icons)

- [ ] Installation experience
  - One-click installer
  - Minimal dependencies
  - Auto-run on first start

---

10. DEVELOPMENT PHASES (REVISED)

Phase 1: MVP - Local PDF Analysis (Weeks 1-4)

Goal: Prove AI extraction works

Tasks:

- [ ] Electron + React setup
- [ ] Drag-and-drop PDF upload
- [ ] PDF text extraction (pdfjs-dist)
- [ ] Claude API integration (basic prompt)
- [ ] Display extracted transactions in table
- [ ] Basic Excel export
- [ ] SQLite local database
- [ ] User authentication (local)
- [ ] Settings management

Deliverable: Working desktop app that processes 1-2 bank statements

---

Phase 2: Production Hardening (Weeks 5-8)

Goal: Enterprise-grade security & UX

Tasks:

- [ ] Database security implementation
  - Encryption at rest
  - Audit logging
  - User/permission model
- [ ] Enhanced UI
  - Data review before saving
  - Confidence scores display
  - Duplicate detection
  - Better error messages

- [ ] PDF improvements
  - OCR for scanned statements
  - Multi-page handling
  - Error recovery
- [ ] Testing
  - Unit tests for extraction logic
  - E2E tests with real statements
  - Security review

Deliverable: Hardened, tested version ready for beta

Phase 3: Scaling & Features (Weeks 9-12)

Goal: Multi-user, advanced features, monetization

Tasks:

- [ ] Backend API setup (if planning SaaS)
- [ ] Multi-user support
- [ ] License/trial system
- [ ] Batch processing queue
- [ ] Advanced export options
- [ ] Transaction categorization
- [ ] Backup/restore
- [ ] Localization (German, French)
- [ ] Performance optimization
- [ ] Monitoring setup

Deliverable: Feature-complete product

Phase 4: Polish & Launch (Weeks 13-16)

Goal: Ready for paying customers

Tasks:

- [ ] Installer creation & code signing
- [ ] Documentation & help system
- [ ] Legal docs (TOS, Privacy Policy)
- [ ] Support system setup
- [ ] Marketing site

- [ ] Payment integration (if SaaS)
- [ ] Final security audit
- [ ] Performance optimization
- [ ] Beta testing with real users

Deliverable: Launchable product

---

11. CONFIGURATION MANAGEMENT

All hardcoded values should be moved to configuration:

11.1 Configuration Database Table

```
config_key (string)       | config_value (JSON)
-----------               | -------------------
max_file_size             | 52428800 (50MB)
max_statements_per_day    | 100
retention_days            | 2555 (7 years)
auto_delete_enabled       | true
ocr_enabled               | true
api_timeout_seconds       | 60
batch_size_api_calls      | 3
max_confidence_threshold  | 0.85
supported_languages       | ["en", "de", "fr"]
```

11.2 Environment Variables

```
NODE_ENV=production
DB_HOST=localhost
DB_PORT=5432
DB_NAME=bank_statements
DB_USER=app_user
DB_PASSWORD=(encrypted or vault)
CLAUDE_API_KEY=(user-provided or backend)
REDIS_URL=redis://localhost:6379
JWT_SECRET=(vault)
ENCRYPTION_MASTER_KEY=(vault)
LOG_LEVEL=info
```

---

12. SUCCESS METRICS

Functional Metrics

- [ ] 95%+ transaction extraction accuracy (validated against user reviews)
- [ ] Support for 20+ banks by month 6
- [ ] <5% error rate on PDF processing
- [ ] <2 seconds to extract 1-page statement
- [ ] 99% API uptime (if SaaS)

User Experience Metrics

- [ ] Average statement processing time <30 seconds
- [ ] <3 manual corrections per 10 transactions
- [ ] User satisfaction >4.5/5 stars
- [ ] Churn rate <5% (monthly)

Business Metrics

- [ ] Acquire 100 users by month 6
- [ ] 50% convert to paid by month 12
- [ ] Customer acquisition cost < $20
- [ ] 30-day retention > 80%

---

13. RISK MITIGATION

Risk	Impact	Mitigation
AI extraction fails on new bank format	High	Build manual review UI, maintain feedback loop, test regularly with new PDFs
Data breach (user bank data)	Critical	Encryption at rest/transit, audit logs, insurance, incident response plan
Regulatory issues (licensing)	High	Consult lawyer early, review compliance requirements per jurisdiction
High API costs	Medium	Batch requests, optimize prompts, implement usage limits, pass costs to users
Poor user adoption	Medium	Focus on UX, gather user feedback early, beta testing with target audience
Duplicate detection failures	Medium	Allow user override, show confidence scores, manual merge feature
Scalability bottleneck	Medium	Design for microservices from day 1, use job queues, test under load

---

14. NEXT STEPS

1. **Gather sample PDFs** (5-10 statements from different banks and countries)

2. **Create detailed AI prompt** (test with Claude manually first)

3. **Design database schema** (refine the schema above)

4. **Set up development environment** (Electron + Node.js + PostgreSQL locally)

5. **Begin Phase 1 development** (MVP)

Appendix A: AI Extraction Prompt Template

```
You are an expert financial data analyst. You will be given bank statement text and must extract all transaction information.

Rules:
1. Return ONLY valid JSON, no other text
2. All amounts must be numbers with decimal points
3. All dates in YYYY-MM-DD format
4. transaction_type must be one of: debit, credit, transfer, fee, interest
5. confidence_score: 0-100, indicating how confident you are in this extraction
6. If a field is unclear or missing, set confidence_score < 50 and include "uncertain_fields" array

Return this JSON structure:

{
 "bank_name": "string (detected or null)",
 "account_number_last4": "string",
 "statement_period": {
  "start_date": "YYYY-MM-DD",
  "end_date": "YYYY-MM-DD"
 },
 "currency": "string (ISO 4217 code)",
 "transactions": [
  {
   "date": "YYYY-MM-DD",
   "amount": number,
   "currency": "string",
   "description": "string (merchant or description)",
   "transaction_type": "string",
   "confidence_score": number (0-100),
   "uncertain_fields": ["field_name"]
  }
 ],
 "extraction_confidence": number (0-100 for overall quality),
```

```
  "warnings": ["string"]
}

[Attach actual PDF text here]
```

END OF DOCUMENT