**🏦 Bank Reconciliation Agent – Project Roadmap & Executive Summary**

*Last Updated: 2025-07-27*

**✨ Executive Summary**

The **Bank Reconciliation Agent** automates the preparation and transformation of bank statements and remittance files into Xero-compatible formats. It eliminates manual formatting and enables seamless data ingestion across different banks and payment formats. The long-term vision is to build a full AI-powered reconciliation engine capable of mapping, classifying, and posting transactions directly into Xero.

**📊 Key Features**

* Upload interface via FastAPI Swagger UI
* Auto-detection of bank type from filename
* Custom parsers for:
  + NBB (CSV)
  + KFH Account (Excel)
  + KFH Card (Excel & PDF-ready)
* Smart date normalization (all exports: yyyy/mm/dd)
* Currency sanitization and sign detection (debit/credit based)
* Remittance Excel support:
  + Sheet 1: Invoice-based payments from Xero
  + Sheet 2: Manual payments (e.g., salaries, transfers)
* Multi-file upload support
* Dynamic CSV export with timestamped filenames (e.g., KFH20250725.csv)
* Export matches custom Xero-compatible template
* Download endpoint for exporting formatted CSVs

**🚀 Architecture & Tech Stack**

* **Backend**: FastAPI
* **Parsing/Processing**: Pandas
* **OCR (planned)**: Tesseract or Google Vision
* **Storage**: Local file system (CSV), Google Sheets (future)
* **Integration**: Xero API (future), ApprovalMax (future)
* **Authentication**: None (local testing phase)
* **Interaction Layer (planned)**:
  + Google Chat Bot (inline approvals, prompts)
  + Streamlit UI (recon dashboard)
  + Gmail API (automated alerts and PO flows)

**📂 Project Structure (Simplified)**

bank\_reco\_agent/

├── app/

│ ├── main.py # App entry point

│ ├── routes/

│ │ ├── upload.py # Upload handler (bank files)

│ │ ├── remittance.py # Upload route for single + multi remittance

│ │ └── download.py # CSV download endpoint

│ ├── services/

│ │ ├── parser.py # Bank statement parsing logic

│ │ ├── remittance\_parser.py # Dual-sheet invoice + manual payment logic

│ │ └── xero\_format.py # Statement formatter

│ ├── templates/ # Xero import template

│ └── exports/ # Auto-generated Xero CSVs

**🧠 Core Logic**

* File uploaded via /upload/...
* Bank files: detect bank, parse statement, map to export format
* Remittance files:
  + Sheet 1: extract supplier invoices from Aged Payables Detail
  + Sheet 2: extract manual payments from Manual Payments
* Dates and currency normalized
* CSVs saved with structured filenames (e.g., NBB20250727.csv)
* Files returned via download endpoint
* Multiple file uploads supported via /remittance-multi

**📄 Recommended Architecture**

| **Component** | **Status** |
| --- | --- |
| File Upload | ✅ FastAPI Swagger UI |
| Bank Parsing | ✅ Done for KFH, NBB |
| Remittance Parsing | ✅ Dual-sheet supported |
| CSV Formatting | ✅ Template-driven |
| Export Storage | ✅ Local /exports/ |
| Download Endpoint | ✅ Implemented |
| OCR Parsing | ⏳ Next phase |
| Multi-file Upload | ✅ Added for remittance |
| Reconciliation Engine | ⏳ Upcoming |
| User Interaction Layer | ⏳ Planned (Chat, Web UI) |

**🔄 Visual Flowchart**

User Uploads File(s)

|

v

Identify File Type (bank vs remittance vs receipt)

|

+----------------------------+

| |

Bank File Remittance

| |

Parse by Bank Sheet 1: Invoices

Normalize + Clean Sheet 2: Manual Payments

| |

Map to Xero Format Combine, Validate

| v

Save to /exports/ Return parsed JSON

|

+----------------------------+

|

Future: Receipt OCR + Matching

**📁 Folder/Code Structure**

app/

├── routes/

│ ├── upload.py # Handles bank statements

│ ├── remittance.py # Single + multi-file Excel parsing

│ └── download.py # Exports

├── services/

│ ├── parser.py # KFH, NBB logic

│ ├── remittance\_parser.py # Invoices + manual payments

│ └── xero\_format.py # Standard output mapping

├── templates/ # Import template

├── exports/ # Final output files

**✅ Completed Steps / Phase 1–2**

* Bank file parsing for NBB, KFH (Account & Card)
* Date format normalization (yyyy/mm/dd)
* Xero-compatible CSV export structure
* Multi-bank support with auto-routing
* Export file naming logic (BANKyyyymmdd)
* Remittance file parser:
  + Aged Payables Detail → supplier invoices
  + Manual Payments → salary/credit card/etc.
* Manual payment template finalized (5 columns)
* Multi-file upload support for remittance

**⏳ Next Steps / Phase 3–7**

**Phase 3: OCR + Image Receipt Parsing (3–5 days)**

* Add /upload/payment-receipt endpoint
* Integrate OCR (Tesseract or Google Vision)
* Extract info from BenefitPay & Talabat screenshots
* Map payment receipts to bank/remittance entries

**Phase 4: Reconciliation Logic (3–7 days)**

* Match bank transactions with invoices + receipts
* Highlight unmatched or duplicate payments
* Generate recon summary (JSON/CSV)

**Phase 5: Xero API Integration (3–5 days)**

* Post matched payments to Xero
* Fetch unpaid invoices / suppliers
* Validate data before posting

**Phase 6: Interaction Layer (2–4 days)**

* Google Chat Bot for recon review + approvals
* Streamlit dashboard to view unmatched items
* Email alerts (Gmail API) for exceptions

**Phase 7: Machine Learning (5–10 days)**

* Auto-categorize payment descriptions (NLP)
* Suggest likely suppliers or accounts
* Learn from corrections (feedback loop)
* Fuzzy match invoice numbers or names

**⏹ Timeline Estimate (Realistic Forecast)**

Assuming momentum continues:

| **Phase** | **Feature** | **Timeline** |
| --- | --- | --- |
| ✅ 1–2 | Parser, format, export | ✅ Complete |
| 3 | OCR receipts | ~3–5 days |
| 4 | Matching & Reconciliation | ~3–7 days |
| 5 | Xero API integration | ~3–5 days |
| 6 | Interaction Layer | ~2–4 days |
| 7 | ML Suggestions & Matching | ~5–10 days |

**End-to-End MVP: ~3–4 weeks** for full interactive, semi-autonomous recon pipeline

This document is the single source of truth for the Bank Reconciliation Agent’s architecture, implementation, and roadmap. It will continue evolving as each phase is completed.

# ✅ Phase 3 – Reconciliation Matching Engine (Updated 2025-07-27)

The reconciliation engine now supports full rule-based matching between bank statement entries and remittance data.  
  
Key Enhancements:  
- Matching logic compares:  
 • Absolute amount (±0.05 BHD)  
 • Date range (±7 days, where available)  
- Manual Payments sheet: Matched successfully (7/7) using amount and date.  
- Aged Payables Detail sheet:  
 • Supplier payments now sourced from 'Total [Supplier]' rows only  
 • Extracted from column 0, amount from column 13  
 • Used to match bank payments (no exact date in remittance rows)  
  
Duplicate Handling:  
- Remittance rows are matched only once  
- If a remittance row is matched by multiple bank rows, subsequent matches are flagged as:  
 ➤ Review Status = 'Duplicate Match – Review'  
  
Duplicate Amount Alert:  
- New logic flags all matched rows with the same amount as:  
 ➤ Review Status = 'Duplicate Amount – Review'  
- Ensures reviewer can inspect and confirm if repetition is valid (e.g. bulk payments)  
  
Review Status Values:  
✓ → Confident match  
Needs Review → No match found  
Duplicate Match – Review → Same remittance row used more than once  
Duplicate Amount – Review → Same amount matched more than once across different rows  
  
These flags will help guide future ML training and downstream approval flows.  
  
Export: Final reviewed output saved as 'Bank\_Reco\_Final\_With\_Review\_And\_DuplicatesFlagged.csv'

# 📝 Phase Adjustment Note (2025-07-27)

The original Phase 3 (OCR + Image Receipt Parsing) has been temporarily deferred.  
  
Rationale:  
- Reconciliation logic was core to validating the agent's ability to perform accurate matching  
- OCR receipt parsing (for Talabat, BenefitPay, etc.) can be added as a modular Phase 5 feature  
- Phase 3 and 4 have been reclassified to reflect the actual build sequence  
  
Updated Phase Mapping:  
✅ Phase 3 → Reconciliation Logic (formerly Phase 4)  
⏳ Phase 4 → OCR Receipt Parsing (deferred)  
🔜 Phase 5 → Xero API Integration  
🔜 Phase 6 → Interaction Layer (Chatbot, Streamlit UI)  
🔜 Phase 7 → Machine Learning Matching & Suggestions