Xero Two-Way Sync - Phase A, B, C Implementation

Status: ✅ Phase A, B, C Complete

This document tracks the implementation of the initial phases of the Xero two-way sync system.

✅ Phase A: Field Mapping Document (COMPLETE)

Deliverable: `docs/xero-field-mapping.md`

What Was Created

- Comprehensive field mapping for Contacts, Invoices, and Payments
- Clear data ownership rules (Xero vs App authority)
- Contact classification logic (Client, Supplier, or Both)
- Item category → Xero account code mapping
- Conflict resolution strategy
- Safety rules and rollback plan
- Testing scenarios checklist

Key Decisions

1. **Contacts: Xero is primary owner** (pull-dominant)
2. **Invoices: App is primary owner** (controlled push)
3. **Payments: Xero is primary owner** (pull-only)
4. **Contacts with IsCustomer + IsSupplier create records in BOTH tables**
5. **Auto-assign client/supplier numbers only when missing**

✅ Phase B: Sync Metadata Schema (COMPLETE)

Deliverable: `prisma/migrations/20251007_xero_sync_mapping/migration.sql`

Database Schema

Created `xero_sync_mappings` table with:
- Entity type (CONTACT, INVOICE, PAYMENT)
- Bidirectional ID mapping (local_id ↔ xero_id)
- Last synced timestamp
- Change hash for idempotent sync
- Status tracking (ACTIVE, ARCHIVED, ERROR, CONFLICT)
- Error logging

Indexes Created

- `idx_xero_sync_entity_local` on (entity_type, local_id)

- `idx_xero_sync_entity_xero` on (entity_type, xero_id)
- `idx_xero_sync_status` on (status)
- `idx_xero_sync_last_synced` on (last_synced_at)
- `idx_xero_sync_unique_mapping` (unique constraint)

Why Raw SQL?

Migration uses raw SQL with explicit enum type casting because Prisma's enum handling in raw queries requires explicit `::` casting to avoid type errors.

✅ Phase C: Contacts Pull (COMPLETE)

Deliverables:
1. `lib/xero-sync-service.ts` - Core sync service
2. `app/api/xero/sync/contacts/route.ts` - API endpoint

Implementation Highlights

XeroSyncService Class

- **Reuses existing OAuth token management** (no duplication)
- **Idempotent operations** (re-running doesn't create duplicates)
- **Change detection** using MD5 hash of relevant fields
- **Transaction safety** with proper error handling

Contact Pull Logic

```
1. Fetch contacts from Xero (with optional modifiedSince filter)
2. For each contact:
   - Calculate change hash
   - Check if already synced and unchanged → skip
   - Extract contact data (name, email, phone, addresses, etc.)
   - Determine classification (Customer, Supplier, Both)
   - Create/update in Client and/or Supplier tables
   - Auto-assign clientNumber/supplierNumber if missing
   - Create/update sync mapping
3. Log results (created, updated, skipped, errors)
```

Classification Logic

```
if (IsCustomer && IsSupplier) → Create in BOTH Client + Supplier
else if (IsCustomer) → Create in Client only
else if (IsSupplier) → Create in Supplier only
else → Skip (log warning)
```

Auto-Numbering

- **Client:** `CL00001`, `CL00002`, etc.
- **Supplier:** `SUP00001`, `SUP00002`, etc.
- Only assigned if `clientNumber` or `supplierNumber` is null
- Preserves existing numbers from manual entries

API Endpoint

POST /api/xero/sync/contacts

Request body (optional):

```json
{
  "modifiedSince": "2025-01-01T00:00:00Z",
  "includeArchived": false,
  "forceRefresh": false
}
```

Response:

```json
{
  "success": true,
  "message": "Successfully synced 42 contacts (15 created, 20 updated, 7 skipped)",
  "created": 15,
  "updated": 20,
  "skipped": 7,
  "errors": 0,
  "errorDetails": [],
  "logId": "clxxx..."
}
```

Permissions

- Allowed roles: `SUPERADMIN`, `FINANCE`, `PROJECT_MANAGER`
- Returns 403 for unauthorized users

Logging

- Uses existing `XeroLogger` service
- Logs to `xero_logs` table
- Tracks: recordsProcessed, recordsSucceeded, recordsFailed, duration
- Includes detailed error messages

Testing Checklist

Before merging, verify:

- [ ] Migration runs successfully: `yarn prisma migrate dev`
- [ ] Prisma client regenerates: `yarn prisma generate`
- [ ] TypeScript compiles: `yarn build`
- [ ] API endpoint accessible: `POST /api/xero/sync/contacts`
- [ ] Pull 10-20 Xero contacts successfully
- [ ] Contacts classified correctly (Client, Supplier, Both)
- [ ] Auto-numbering works (CL00001, SUP00001, etc.)
- [ ] Existing contacts update (do not duplicate)
- [ ] Change hash skips unchanged records
- [ ] Sync logs written to `xero_logs` table

- [ ] Sync mappings created in `xero_sync_mappings` table
- [ ] Re-running sync is idempotent (no duplicates)

Next Steps

Phase D: Invoice Push (App → Xero)

- Implement `POST /api/xero/push/invoices`
- Validate contact mapping exists before push
- Handle Xero validation errors gracefully
- Store returned InvoiceID in sync mapping

Phase E: Payments Pull (Xero → App)

- Implement `POST /api/xero/sync/payments`
- Update invoice status based on payment amounts
- Handle partial payments

Phase F: Webhooks (Optional)

- Register webhook handler at `/api/xero/webhooks`
- Validate Xero signature
- Enqueue lightweight reconciliation jobs

Phase G: Sync Status UI

- Display sync statistics (counts, last sync time)
- Show recent sync logs
- Provide manual conflict resolution

Phase H: Monitoring & Safety

- Implement retry logic with exponential backoff
- Add admin-only sync log viewer
- Create rollback procedures

Files Created/Modified

New Files

1. `docs/xero-field-mapping.md` - Field mapping and ownership rules
2. `prisma/migrations/20251007_xero_sync_mapping/migration.sql` - Sync mappings schema
3. `lib/xero-sync-service.ts` - Core sync service implementation
4. `app/api/xero/sync/contacts/route.ts` - Contacts sync API endpoint
5. `docs/XERO_TWO_WAY_SYNC_PHASE_ABC.md` - This document

Modified Files

- None (isolated implementation, no existing code modified)

Important Notes

What Was NOT Changed

✅ **OAuth token management** - Reuses existing `XeroOAuthService`
✅ **Xero connection flow** - No changes to `/api/xero/connect` or `/api/xero/callback`
✅ **Existing sync endpoints** - All existing endpoints remain functional
✅ **UI components** - No frontend changes yet (Phase G)

Why Raw SQL for Migration?

The migration uses raw SQL instead of Prisma schema changes because:
1. **Explicit control** over enum types and casting
2. **Avoid Prisma schema conflicts** with existing codebase
3. **Easier to review** and rollback if needed
4. **Direct PostgreSQL compatibility** without intermediate steps

To apply this migration:

```
cd /home/ubuntu/ampere_business_management/app
yarn prisma db execute --file prisma/migrations/20251007_xero_sync_mapping/migra-
tion.sql --schema prisma/schema.prisma
yarn prisma generate
```

Rollback Instructions

If you need to rollback this implementation:

1. **Drop sync mappings table:**
   `sql`
   ```
   DROP TABLE IF EXISTS xero_sync_mappings;
   DROP TYPE IF EXISTS "XeroSyncDirection";
   DROP TYPE IF EXISTS "XeroSyncStatus";
   DROP TYPE IF EXISTS "XeroSyncEntityType";
   ```

2. **Remove API endpoint:**
   `bash`
   ```
   rm app/api/xero/sync/contacts/route.ts
   ```

3. **Remove sync service:**
   `bash`
   ```
   rm lib/xero-sync-service.ts
   ```

4. **Regenerate Prisma client:**
   `bash`
   ```
   yarn prisma generate
   ```

Version History

- **v1.0** (2025-10-07): Phase A, B, C implementation complete