Data Flow

Deposit Creation Flow

```
User Form → API Route → Validation → Service → Prisma → Database
                ↓            ↓           ↓         ↓
              Zod        Storage     Business  Transaction
            Schema        Type       Rules     Atomicity
                        Checking
```

Withdrawal Flow (UNALLOCATED - FIFO)

```
User Request
    ↓
Validation
    ↓
Find Account Deposits (oldest first)
    ↓
Check Total Available Quantity
    ↓
Start Transaction
    ↓
Loop Through Deposits (FIFO)
    ├─→ Deduct from Deposit 1
    ├─→ Deduct from Deposit 2
    └─→ Create Withdrawal Records
    ↓
Commit Transaction
    ↓
Return Success
```

Withdrawal Flow (ALLOCATED - Specific Bar)

```
User Request (with optional barSerial)
    ↓
Validation
    ↓
Find Matching Deposits
```

```
    ↓
Select Deposit (by barSerial or oldest)
    ↓
Check Quantity Available
    ↓
Start Transaction
    ├─→ Update Deposit remainingQuantity
    └─→ Create Withdrawal Record
    ↓
Commit Transaction
    ↓
Return Success
```

Storage Model

ALLOCATED Storage (Institutional)

```
Bar-Level Tracking
    ↓
Each deposit = One physical bar
    ↓
Tracked by unique barSerial
    ↓
Partial withdrawals update remainingQuantity
    ↓
Bar stays in system until remainingQuantity = 0
```

UNALLOCATED Storage (Retail)

```
Pooled Storage
    ↓
Multiple deposits in shared pool
    ↓
No bar serial tracking
    ↓
Withdrawals use FIFO (First In, First Out)
    ↓
System automatically selects oldest deposits
```

Key Components

Frontend (Next.js)

- **Server Components**: Data fetching (Dashboard, Lists, Detail pages)
- **Client Components**: Forms with state management
- **Styling**: Tailwind CSS utility classes

Backend (Next.js API Routes)

- **RESTful endpoints**: /api/deposits, /api/withdrawals
- **Validation**: Zod schemas before processing
- **Error handling**: Structured error responses

Services (Business Logic)

- **Deposit Service**: Create, validate storage types, check duplicates
- **Withdrawal Service**: FIFO logic, allocated bar selection, transactions
- **Valuation Service**: Calculate portfolio values with metal prices
- **Inventory Service**: Track holdings, generate reports

Database (PostgreSQL + Prisma)

- **Schema**: Account, Deposits, Withdrawals models
- **Relations**: One-to-Many (Account → Deposits/Withdrawals)
- **Transactions**: Atomic operations for withdrawals
- **Constraints**: Unique deposit/withdrawal numbers, bar serials

Security & Validation

```
Input → Zod Schema → Type Safety → Business Rules → Database Constraints
         ↓              ↓              ↓                ↓
    Format Check   TypeScript    Service Logic    Unique Keys
    Required       Compilation   Validation       Foreign Keys
    Max Length     Type Errors   Edge Cases       Check Constraints
```

Edge Case Handling

1. **Storage Type Mismatch**: Validated at service layer before DB
2. **Duplicate Bar Serial**: Checked via query before insert
3. **Insufficient Balance**: Calculated before transaction starts

4. **Partial Withdrawals**: Handled via FIFO loop with remainingQuantity
5. **Wrong Metal Type**: Filtered at query level
6. **Concurrent Operations**: Database transactions ensure atomicity
7. **Missing Bar Serial**: Validated by Zod schema + service logic
8. **Zero/Negative Quantities**: Rejected by Zod schema
9. **Invalid CUIDs**: Validated by Zod CUID check
10. **Account Deletion**: Prevented if deposits/withdrawals exist