Payment Arch

- □□ NextGate Payment & Financial System Architecture

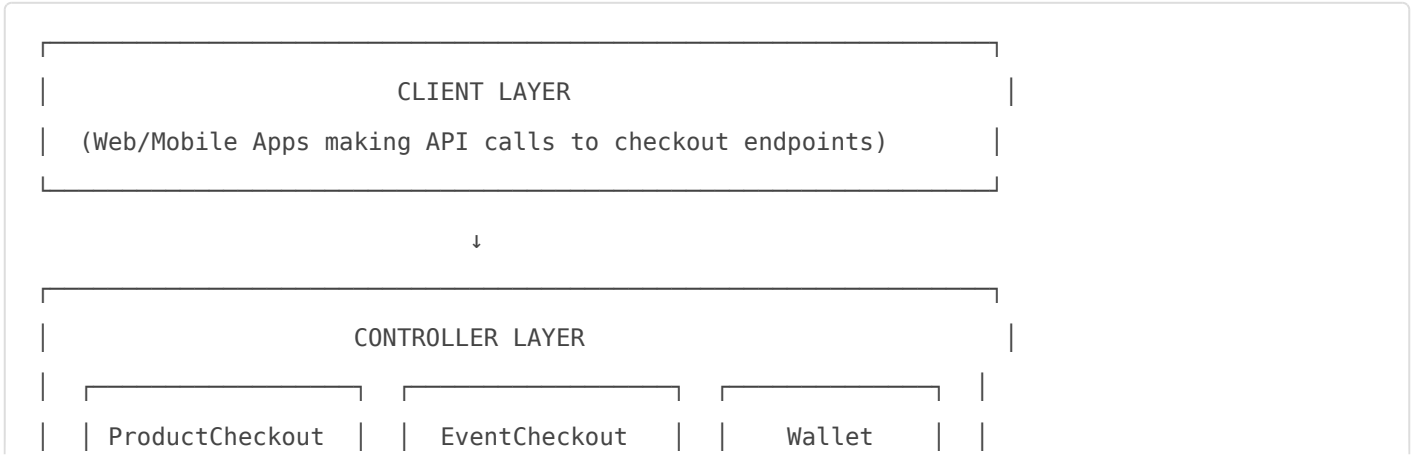
? NextGate Payment & Financial System Architecture

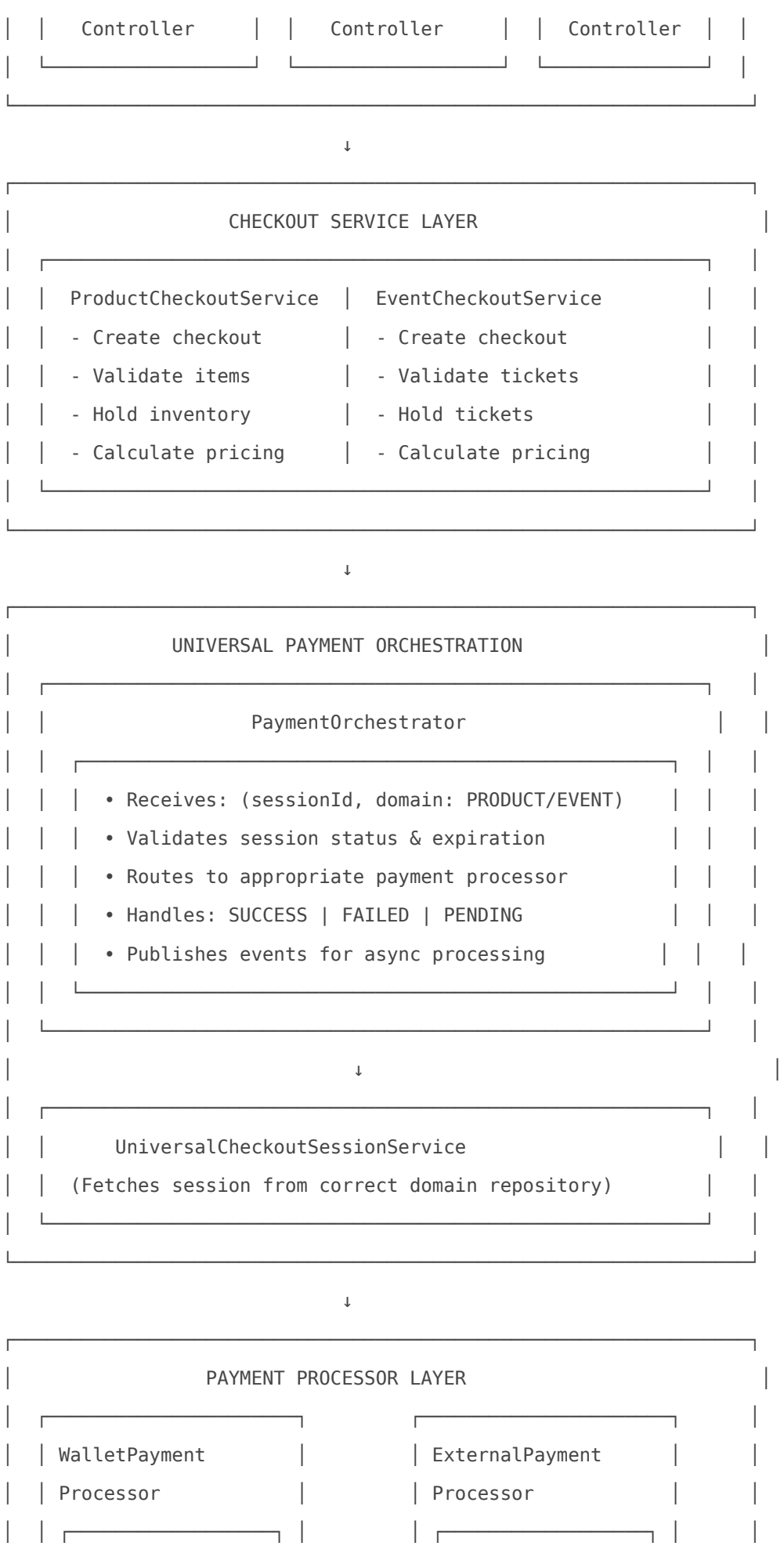
? Table of Contents

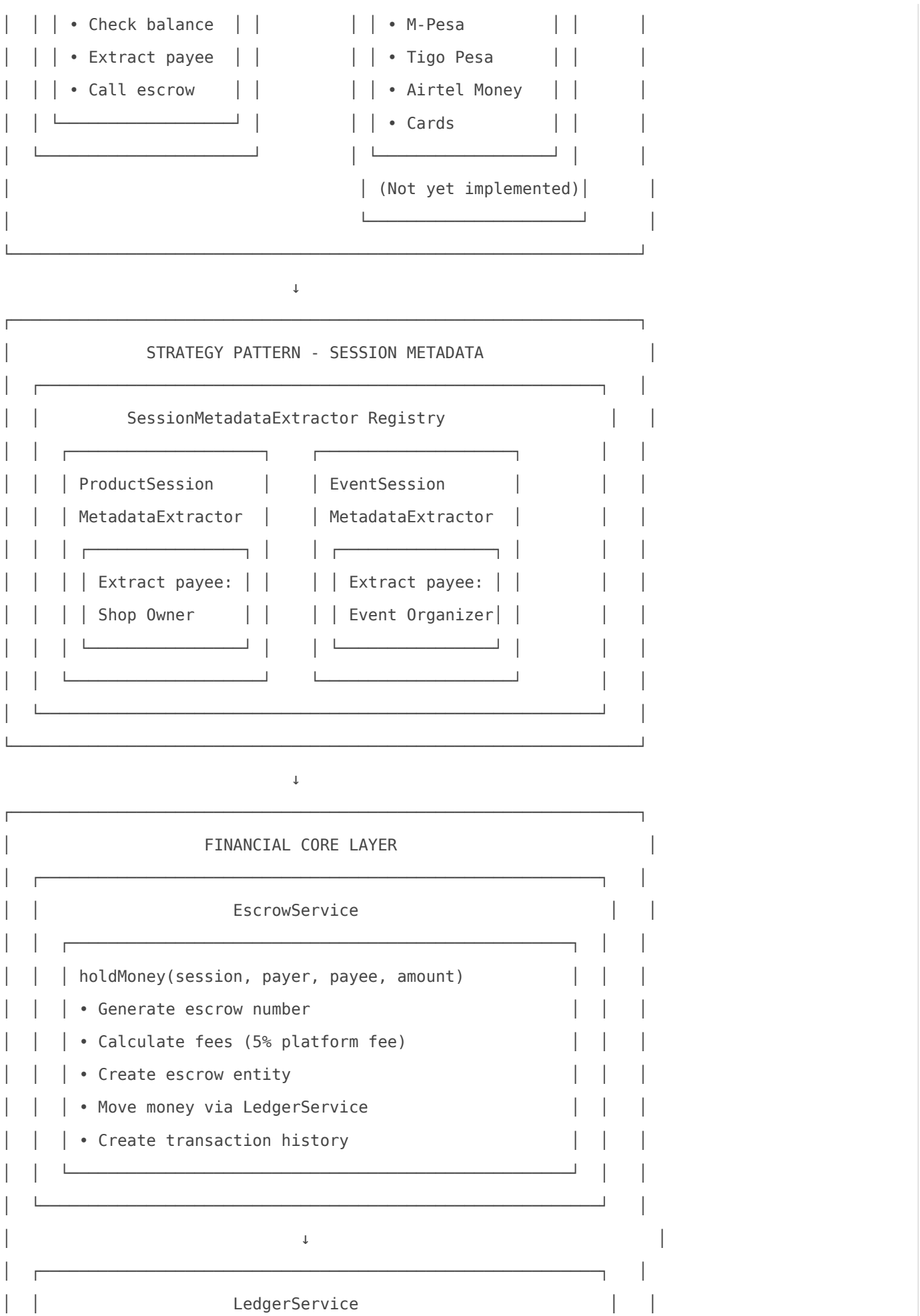
? System Overview

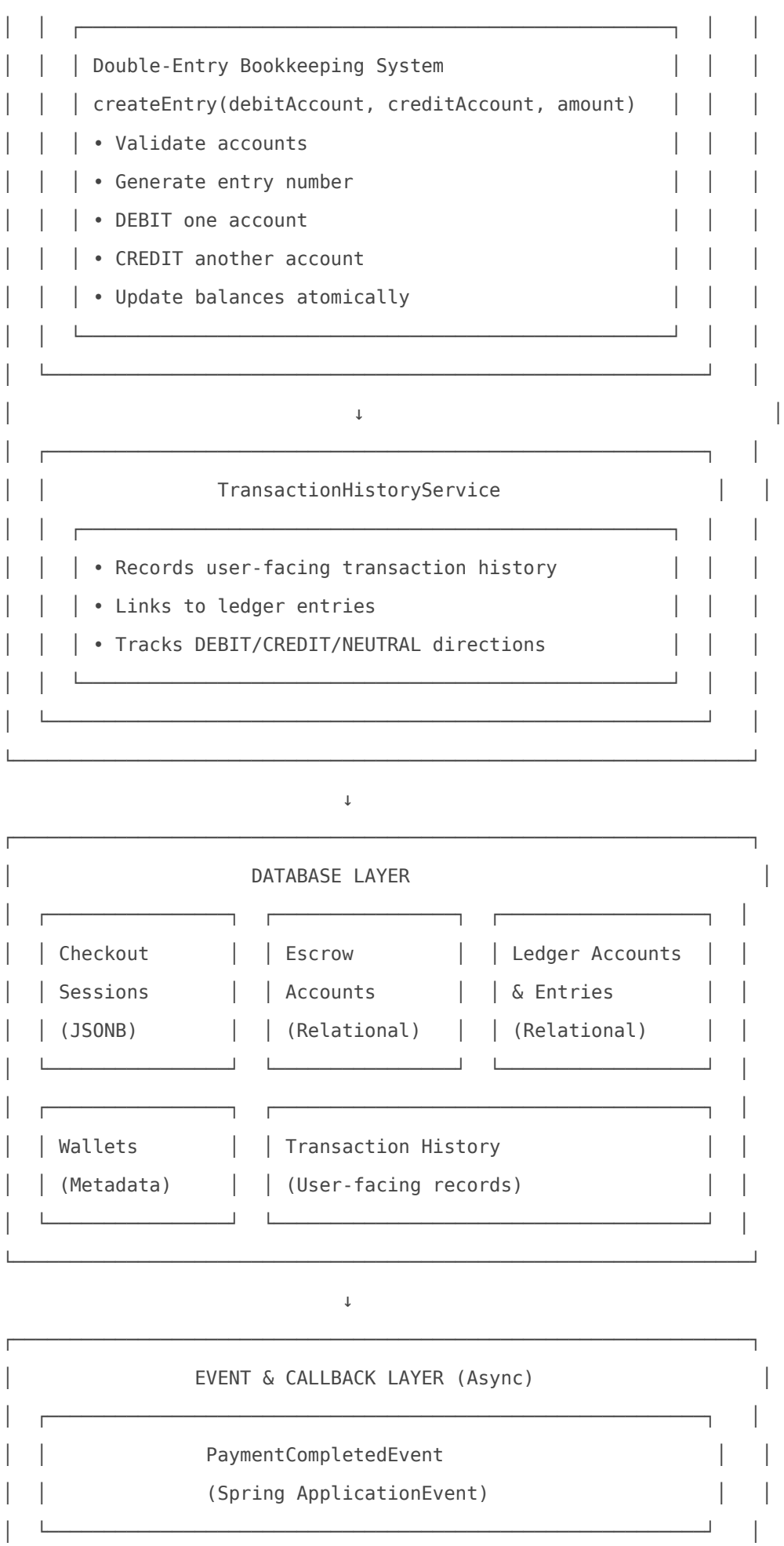
NextGate uses a **universal payment orchestration system** that handles payments across multiple domains (Products, Events, etc.) through a unified interface while maintaining domain-specific business logic.

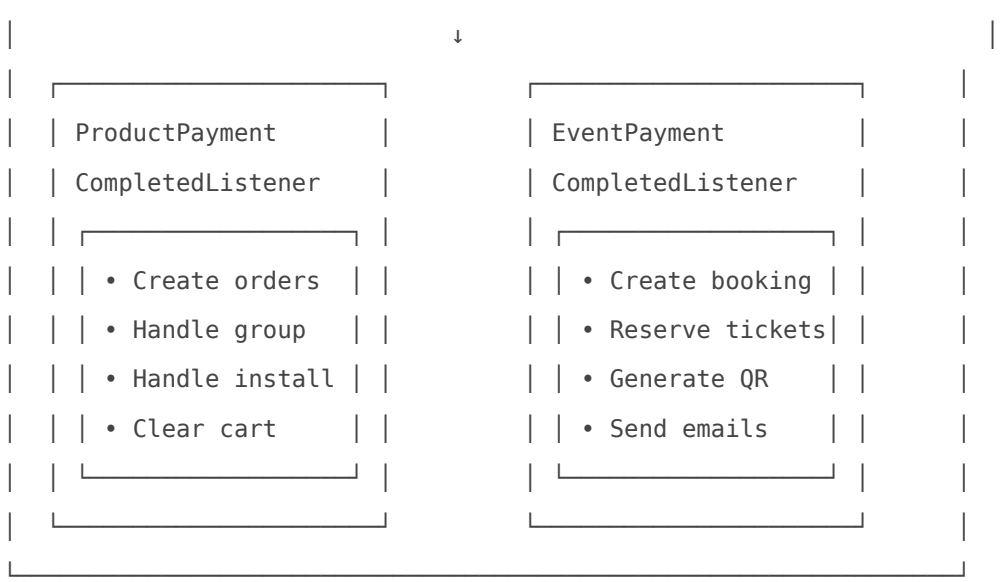
High-Level Architecture




Controller
Controller
Controller
CHECKOUT SERVICE LAYER
ProductCheckoutService
- Create checkout
- Validate items
- Hold inventory
- Calculate pricing
EventCheckoutService
- Create checkout
- Validate tickets
- Hold tickets
- Calculate pricing
UNIVERSAL PAYMENT ORCHESTRATION
PaymentOrchestrator
• Receives: (sessionId, domain: PRODUCT/EVENT)
• Validates session status & expiration
• Routes to appropriate payment processor
• Handles: SUCCESS | FAILED | PENDING
• Publishes events for async processing
UniversalCheckoutSessionService
(Fetches session from correct domain repository)
PAYMENT PROCESSOR LAYER
WalletPayment
Processor
ExternalPayment
Processor

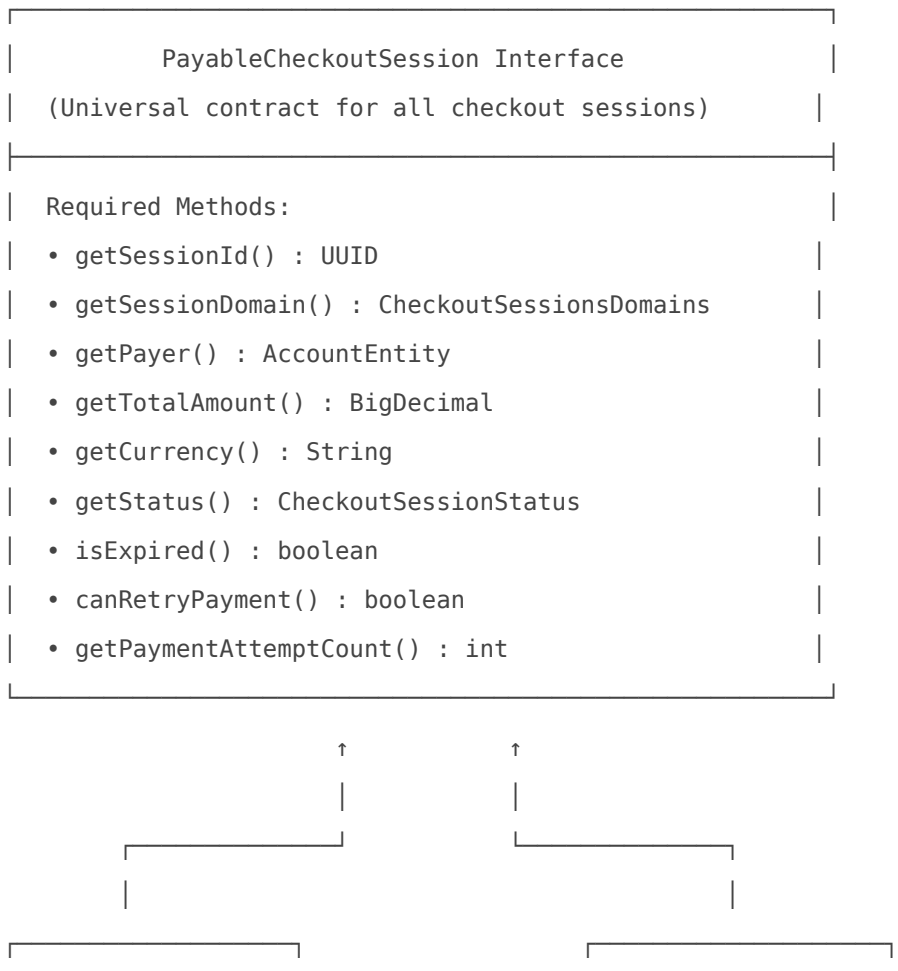

• Check balance
• Extract payee
• Call escrow
• M-Pesa
• Tigo Pesa
• Airtel Money
• Cards
(Not yet implemented)
STRATEGY PATTERN - SESSION METADATA
SessionMetadataExtractor Registry
ProductSession MetadataExtractor
Extract payee: Shop Owner
EventSession MetadataExtractor
Extract payee: Event Organizer
FINANCIAL CORE LAYER
EscrowService
holdMoney(session, payer, payee, amount)
• Generate escrow number
• Calculate fees (5% platform fee)
• Create escrow entity
• Move money via LedgerService
• Create transaction history
LedgerService

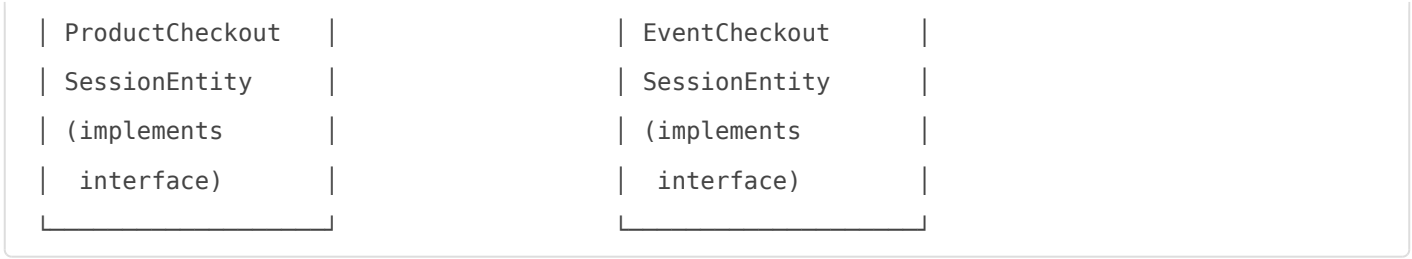

Double-Entry Bookkeeping System
createEntry(debitAccount, creditAccount, amount)
• Validate accounts
• Generate entry number
• DEBIT one account
• CREDIT another account
• Update balances atomically
TransactionHistoryService
• Records user-facing transaction history
• Links to ledger entries
• Tracks DEBIT/CREDIT/NEUTRAL directions
DATABASE LAYER
Checkout Sessions (JSONB)
Escrow Accounts (Relational)
Ledger Accounts & Entries (Relational)
Wallets (Metadata)
Transaction History (User-facing records)
EVENT & CALLBACK LAYER (Async)
PaymentCompletedEvent
(Spring ApplicationEvent)



? Core Components

1. PayableCheckoutSession (Contract Interface)





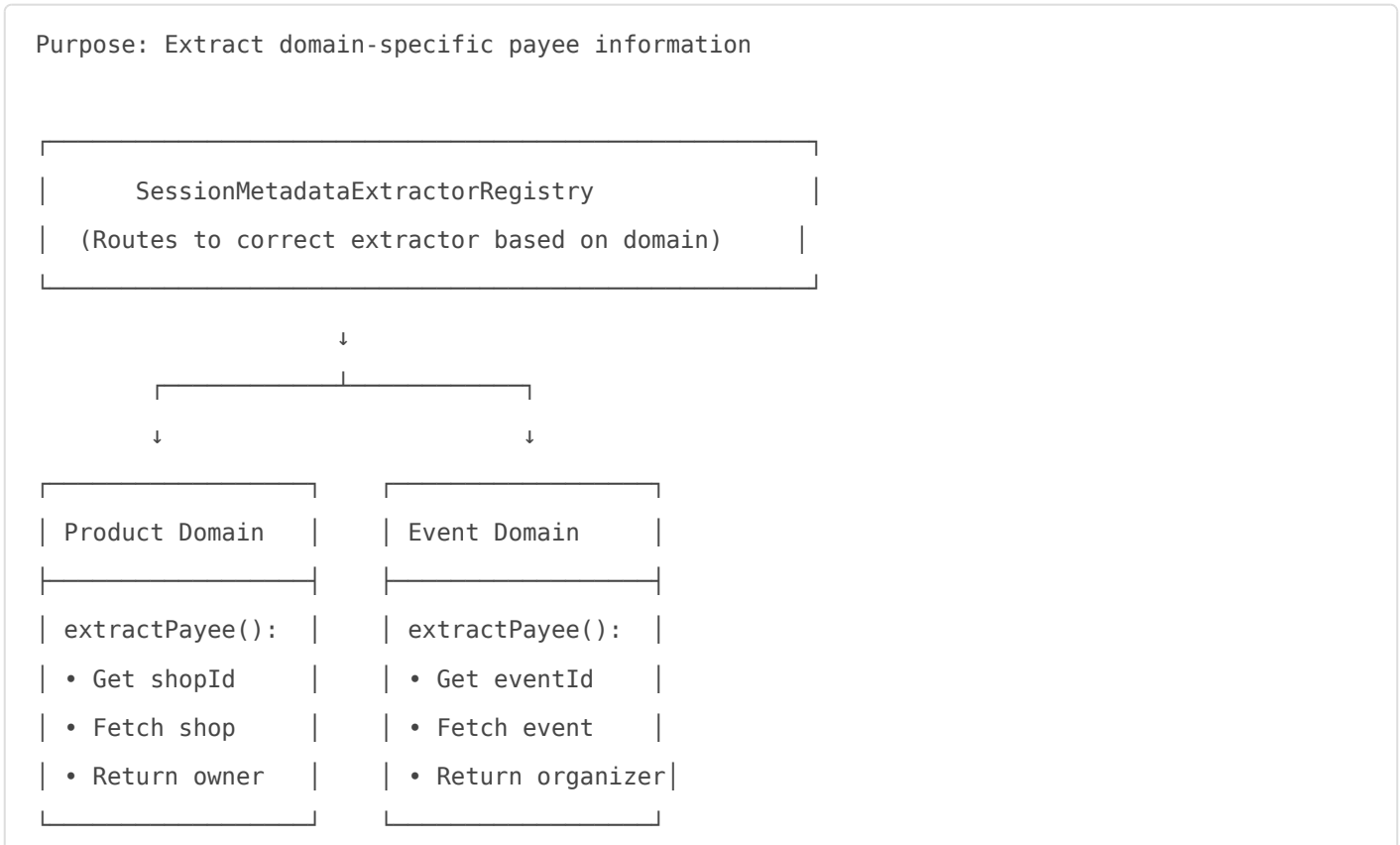
2. Payment Orchestrator

The central coordinator that:

- Validates checkout session
- Routes to correct payment processor
- Handles callbacks
- Publishes events
- Returns unified response

3. Strategy Pattern Components

SessionMetadataExtractor Strategy



PostPaymentHandler Strategy

```
Purpose: Handle domain-specific post-payment logic


┌─────────────────────────────────────────────────────────┐
│         PostPaymentHandlerRegistry                      │
│  (Routes to correct handler based on domain)            │
└─────────────────────────────────────────────────────────┘

                      ↓
              ┌───────┴────────┐
              ↓                ↓

┌──────────────────┐    ┌──────────────────┐
│ Product Domain   │    │ Event Domain     │
├──────────────────┤    ├──────────────────┤
│ • Handle group   │    │ • Update session │
│ • Handle install │    │ • Async booking  │
│ • Clear cart     │    │   via listener   │
│ • Update session │    └──────────────────┘
└──────────────────┘
```

? Payment Flow - Step by Step

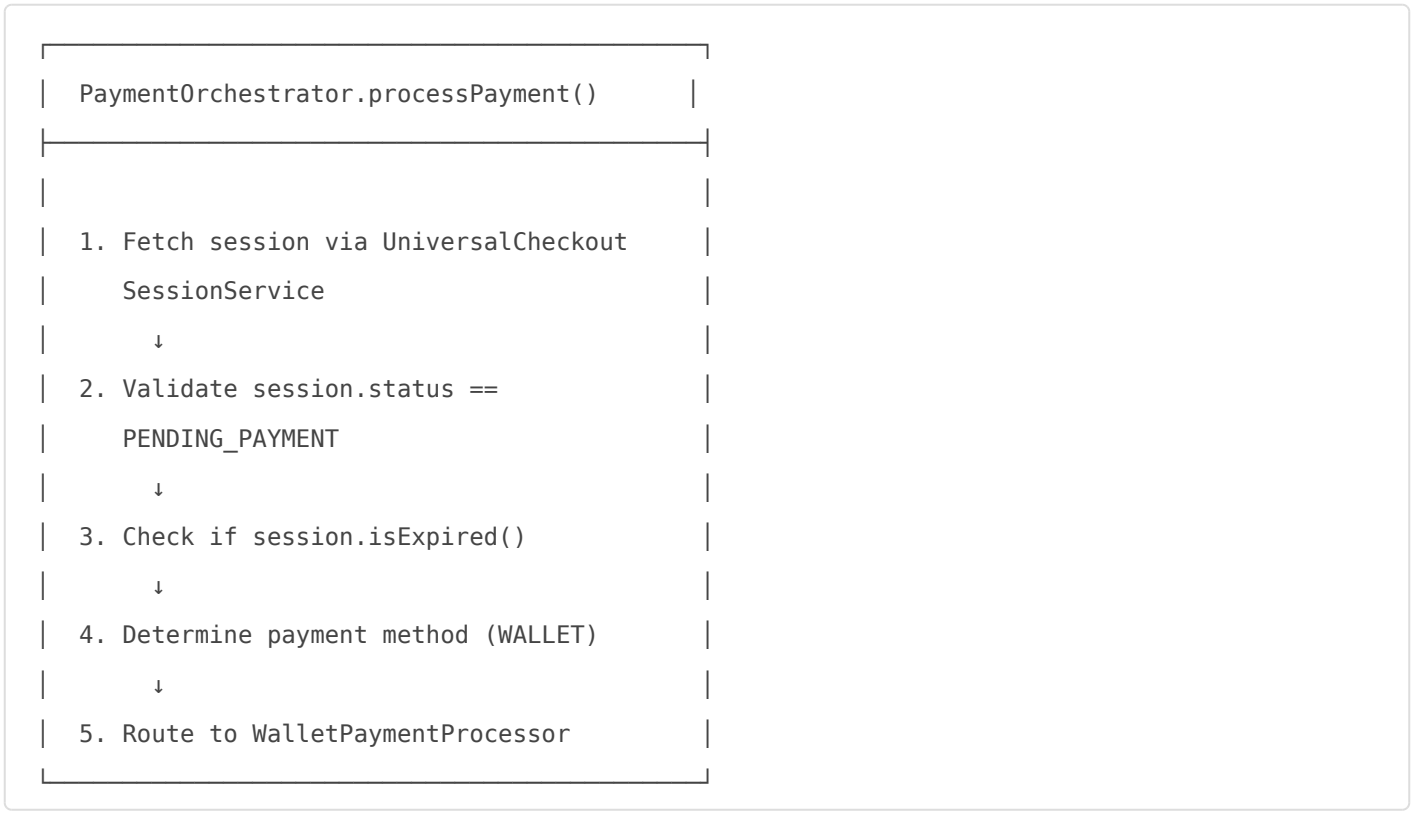
Step 1: User Initiates Payment

```
User clicks "Pay Now"
     ↓
POST /api/v1/e-commerce/checkout-sessions/{sessionId}/process-payment
     OR
POST /api/v1/e-events/checkout/{sessionId}/payment
```

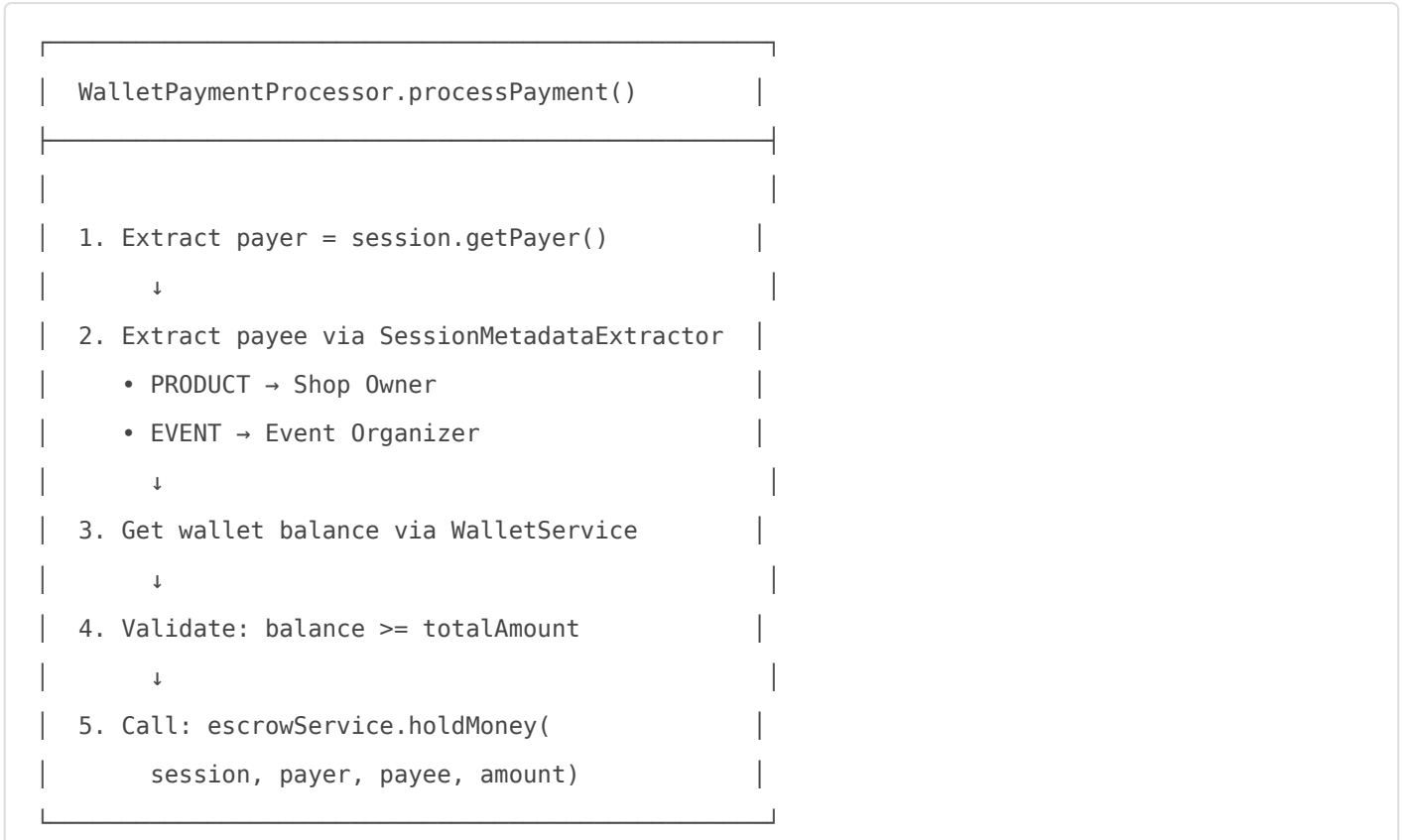
Step 2: Controller ? Service

```
Controller receives request
     ↓
Calls: checkoutService.processPayment(sessionId)
     ↓
Service delegates to: paymentOrchestrator.processPayment(sessionId, domain)
```

Step 3: Payment Orchestrator Validates



Step 4: Wallet Payment Processing



Step 5: Escrow Creation

```
┌──────────────────────────────────────────────────────┐
│  EscrowService.holdMoney()                           │
├──────────────────────────────────────────────────────┤
│                                                      │
│  1. Generate escrow number: "ESC-2025-000123"        │
│       ↓                                              │
│  2. Calculate fees:                                  │
│     platformFee = amount × 5%                        │
│     sellerAmount = amount - platformFee              │
│       ↓                                              │
│  3. Create EscrowAccountEntity                       │
│     - buyer: payer                                   │
│     - seller: payee                                  │
│     - totalAmount: amount                            │
│     - status: HELD                                   │
│       ↓                                              │
│  4. Create ledger account for escrow                 │
│       ↓                                              │
│  5. Move money via LedgerService:                    │
│     createEntry(                                     │
│       debit: payerWalletAccount,                     │
│       credit: escrowAccount,                         │
│       amount: totalAmount                            │
│     )                                                │
│       ↓                                              │
│  6. Create transaction history records               │
│     - PRODUCT_PURCHASE (DEBIT) for buyer             │
│     - ESCROW_HOLD (DEBIT) for admin tracking         │
└──────────────────────────────────────────────────────┘
```

Step 6: Ledger Entry (Double-Entry Bookkeeping)

```
┌──────────────────────────────────────────────────────┐
│  LedgerService.createEntry()                         │
├──────────────────────────────────────────────────────┤
```

```
│                                                    │
│  Input:                                            │
│  • debitAccount: Payer's Wallet Ledger Account     │
│  • creditAccount: Escrow Ledger Account            │
│  • amount: 50,000 TZS                              │
│  • type: PURCHASE                                  │
│                                                    │
│  Process:                                          │
│  1. Validate accounts are different                │
│  2. Validate amount > 0                            │
│  3. Check: debitAccount.balance >= amount          │
│  4. Generate entry number: "LE-2025-000456"        │
│       ↓                                            │
│  5. Create LedgerEntryEntity                       │
│       ↓                                            │
│  6. Update balances ATOMICALLY:                    │
│     debitAccount.balance  -= 50,000                │
│     creditAccount.balance += 50,000                │
│       ↓                                            │
│  7. Save entry and accounts in transaction         │
│                                                    │
│  Result: Money moved from payer to escrow          │
└────────────────────────────────────────────────────┘
```

Step 7: Success Handling

```
┌────────────────────────────────────────────────────┐
│  PaymentOrchestrator.handleSuccessfulPayment()     │
├────────────────────────────────────────────────────┤
│                                                    │
│  1. Update session status:                         │
│     PAYMENT_COMPLETED                              │
│       ↓                                            │
│  2. Set: session.escrowId = escrow.getId()         │
│       ↓                                            │
│  3. Publish: PaymentCompletedEvent                 │
│     (Async - handled by listeners)                 │
│       ↓                                            │
│  4. Call: paymentCallback.onPaymentSuccess()       │
```

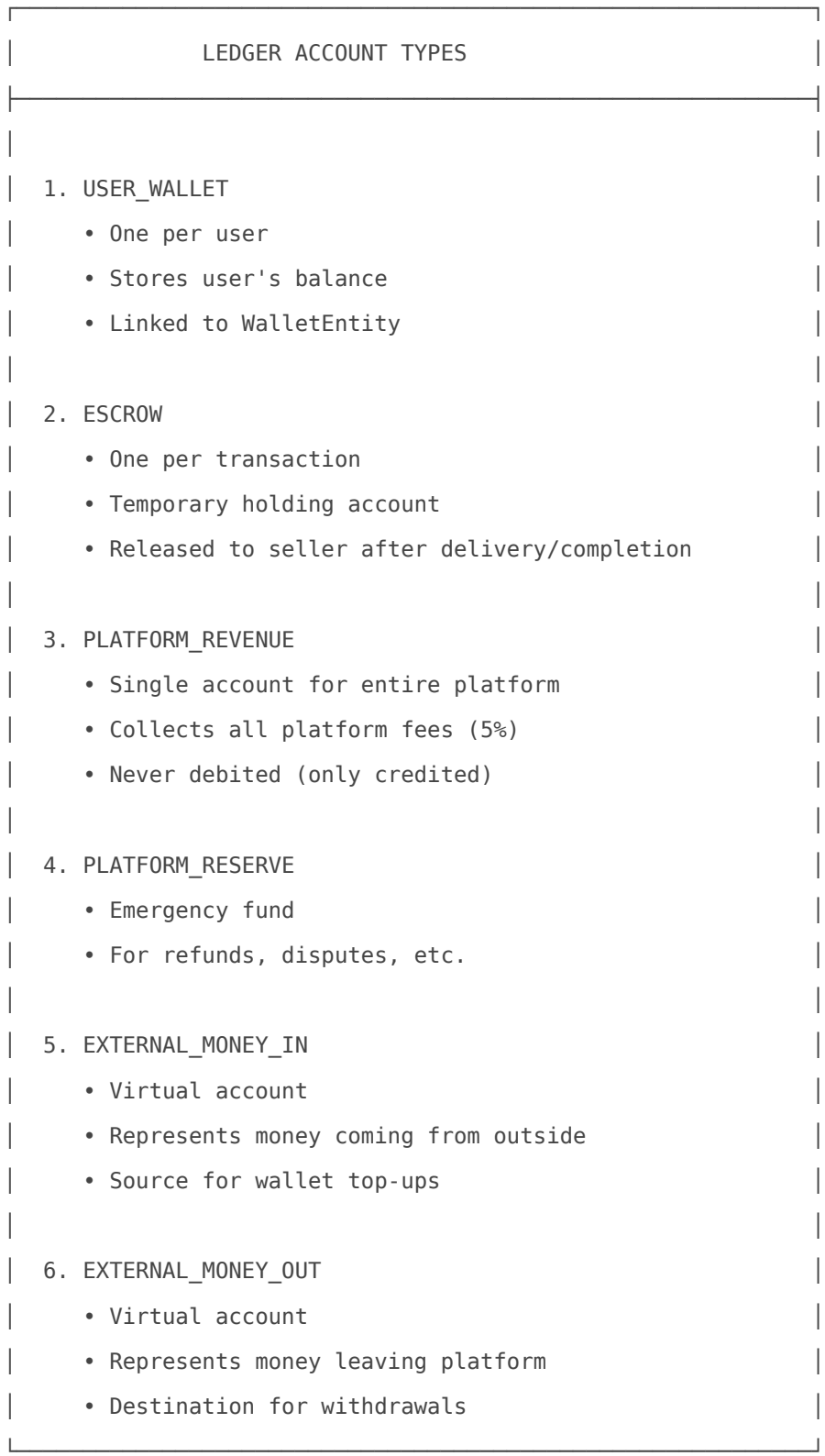
```
│     (Routes to PostPaymentHandler)                  │
│       ↓                                             │
│  5. Return PaymentResponse to user                  │
└─────────────────────────────────────────────────────┘
```

Step 8: Async Processing (Event Listeners)

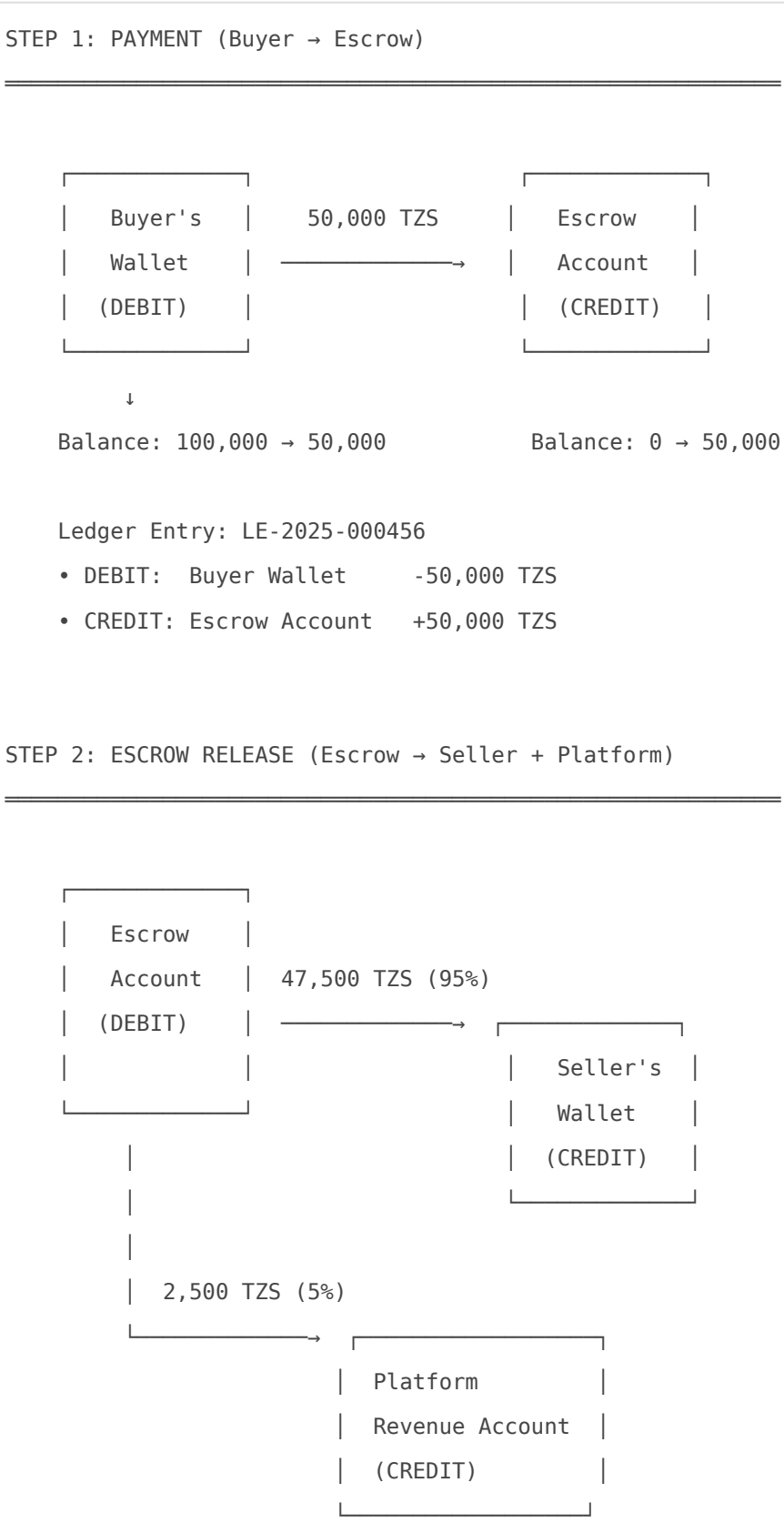
```
┌─────────────────────────────────────────────────────┐
│  PaymentCompletedEvent Published                    │
└─────────────────────────────────────────────────────┘

                    ↓
            ┌───────┴────────┐
            ↓                ↓

┌──────────────────────┐  ┌──────────────────────┐
│ ProductPayment       │  │ EventPayment         │
│ CompletedListener    │  │ CompletedListener    │
├──────────────────────┤  ├──────────────────────┤
│ @Async               │  │ @Async               │
│ @Transactional       │  │ @Transactional       │
│                      │  │                      │
│ Switch by type:      │  │ Actions:             │
│                      │  │                      │
│ REGULAR_DIRECTLY:    │  │ 1. Create booking    │
│ • Create order       │  │ 2. Reserve tickets   │
│                      │  │ 3. Generate QR       │
│ REGULAR_CART:        │  │ 4. Send emails       │
│ • Create order       │  │ 5. Update status:    │
│ • Clear cart         │  │    COMPLETED         │
│                      │  └──────────────────────┘
│ GROUP_PURCHASE:      │
│ • Join/create group  │
│                      │
│ INSTALLMENT:         │
│ • Create agreement   │
│ • Schedule payments  │
└──────────────────────┘
```

? Money Movement Architecture

Account Types in the System



Money Flow: Purchase Transaction



```
Entry 1: LE-2025-000457
• DEBIT:  Escrow Account      -47,500 TZS
• CREDIT: Seller Wallet       +47,500 TZS

Entry 2: LE-2025-000458
• DEBIT:  Escrow Account       -2,500 TZS
• CREDIT: Platform Revenue     +2,500 TZS

Final Escrow Balance: 50,000 → 0 TZS
```

Money Flow: Wallet Top-Up

```
WALLET TOP-UP (External → User Wallet)
═══════════════════════════════════════════════════════════════


    ┌──────────────────┐                  ┌──────────────┐
    │  External        │  100,000 TZS     │   User's     │
    │  Money In        │  ──────────────→ │   Wallet     │
    │  (Virtual)       │                  │  (CREDIT)    │
    │  (DEBIT)         │                  └──────────────┘
    └──────────────────┘


    Ledger Entry: LE-2025-000459
    • DEBIT:  External Money In    -100,000 TZS (virtual)
    • CREDIT: User Wallet          +100,000 TZS

    Transaction History:
    • Type: WALLET_TOPUP
    • Direction: CREDIT
    • Amount: 100,000 TZS
```

Money Flow: Wallet Withdrawal

```
WALLET WITHDRAWAL (User Wallet → External)
═══════════════════════════════════════════════════════════════


    ┌──────────────┐                  ┌──────────────────┐
    │   User's     │  50,000 TZS      │  External        │
    │   Wallet     │  ──────────────→ │  Money Out       │
```

```
    |  (DEBIT)    |                |  (Virtual)      |
    └─────────────┘                |  (CREDIT)       |
                                   └─────────────────┘


    Ledger Entry: LE-2025-000460
    • DEBIT:  User Wallet           -50,000 TZS
    • CREDIT: External Money Out    +50,000 TZS (virtual)

    Transaction History:
    • Type: WALLET_WITHDRAWAL
    • Direction: DEBIT
    • Amount: 50,000 TZS
```

? Ledger System (Double-Entry Bookkeeping)

Core Principle

```
┌──────────────────────────────────────────────────────┐
│     FUNDAMENTAL ACCOUNTING RULE                      │
│                                                      │
│     For every transaction:                           │
│     TOTAL DEBITS = TOTAL CREDITS                     │
│                                                      │
│     Money never appears or disappears.               │
│     It only moves between accounts.                  │
└──────────────────────────────────────────────────────┘
```

Ledger Accounts

```
┌──────────────────────────────────────────────────────┐
│  LedgerAccountEntity                                 │
├──────────────────────────────────────────────────────┤
│  • id: UUID                                          │
│  • accountNumber: "WALLET-johndoe"                   │
```

```
│  • accountType: USER_WALLET | ESCROW | PLATFORM_...  │
│  • owner: AccountEntity (nullable for platform)      │
│  • currentBalance: BigDecimal (cached)               │
│  • currency: "TZS"                                   │
│  • isActive: Boolean                                 │
└──────────────────────────────────────────────────────┘
```

Ledger Entries

```
┌──────────────────────────────────────────────────────┐
│  LedgerEntryEntity                                   │
├──────────────────────────────────────────────────────┤
│  • id: UUID                                          │
│  • entryNumber: "LE-2025-000123"                     │
│  • debitAccount: LedgerAccountEntity                 │
│  • creditAccount: LedgerAccountEntity                │
│  • amount: BigDecimal                                │
│  • entryType: PURCHASE | ESCROW_RELEASE | REFUND     │
│  • referenceType: "PRODUCT_CHECKOUT"                 │
│  • referenceId: UUID (checkout session ID)           │
│  • description: "Product purchase payment"           │
│  • createdAt: LocalDateTime                          │
└──────────────────────────────────────────────────────┘
```

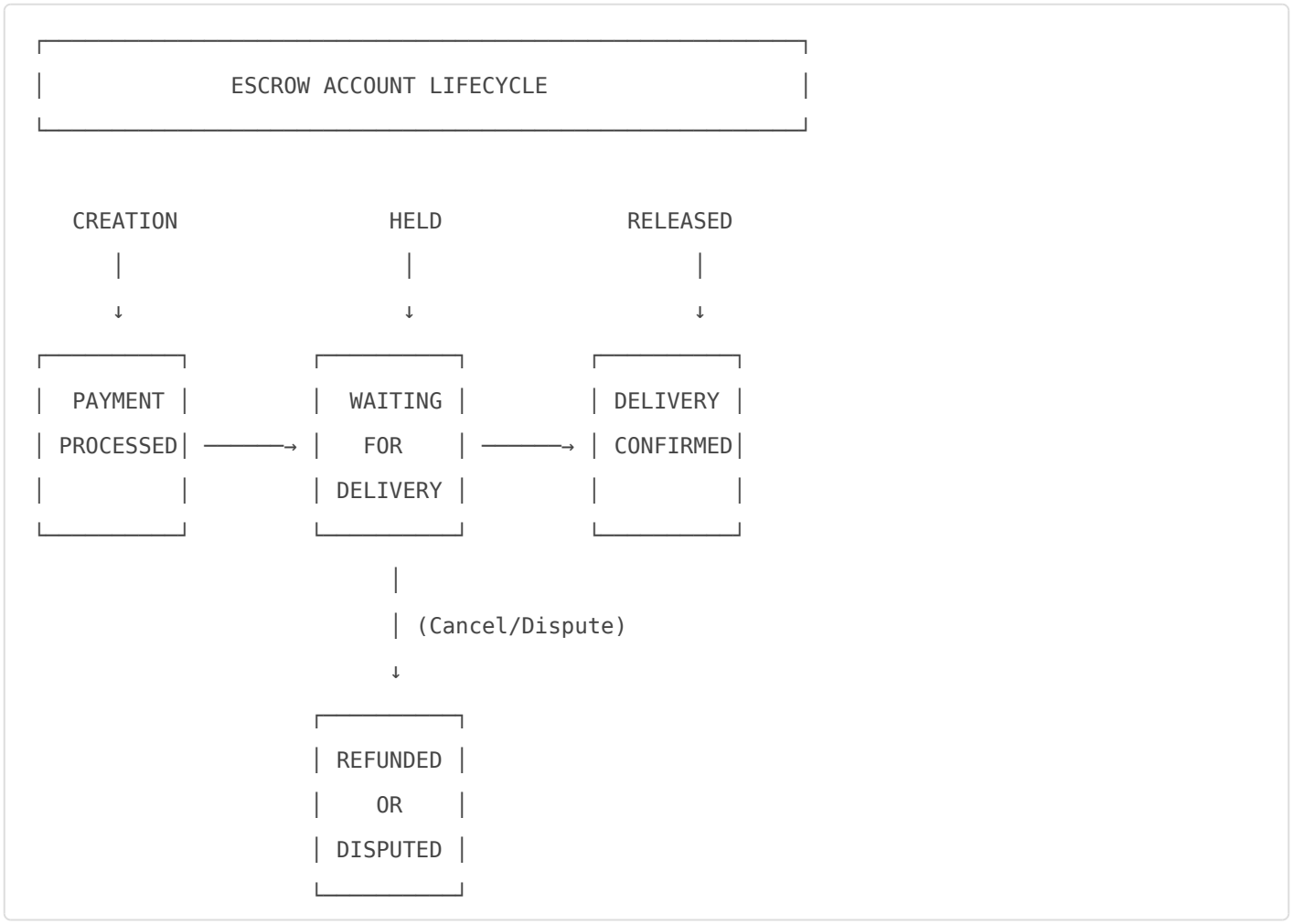
Example Ledger Flow

```
SCENARIO: User buys product for 10,000 TZS

BEFORE TRANSACTION:
┌──────────────────────────────────────────┐
│  Buyer Wallet:    100,000 TZS            │
│  Escrow Account:        0 TZS            │
│  Seller Wallet:    50,000 TZS            │
│  Platform Revenue:  5,000 TZS            │
└──────────────────────────────────────────┘

STEP 1: Create Ledger Entry (Payment)
┌──────────────────────────────────────────┐
│  Entry: LE-2025-000123                   │
```

```
│  • DEBIT:  Buyer Wallet     -10,000 TZS     │
│  • CREDIT: Escrow           +10,000 TZS     │
└─────────────────────────────────────────────┘

AFTER PAYMENT:

┌─────────────────────────────────────────────┐
│  Buyer Wallet:     90,000 TZS  (-10,000)   │
│  Escrow Account:   10,000 TZS  (+10,000)   │
│  Seller Wallet:    50,000 TZS  (no change) │
│  Platform Revenue:  5,000 TZS  (no change) │
└─────────────────────────────────────────────┘

STEP 2: Create Ledger Entries (Escrow Release)

┌─────────────────────────────────────────────┐
│  Entry: LE-2025-000124                      │
│  • DEBIT:  Escrow           -9,500 TZS      │
│  • CREDIT: Seller Wallet    +9,500 TZS      │
│                                             │
│  Entry: LE-2025-000125                      │
│  • DEBIT:  Escrow             -500 TZS      │
│  • CREDIT: Platform Revenue   +500 TZS      │
└─────────────────────────────────────────────┘

AFTER ESCROW RELEASE:

┌─────────────────────────────────────────────┐
│  Buyer Wallet:     90,000 TZS  (no change) │
│  Escrow Account:        0 TZS  (-10,000)   │
│  Seller Wallet:    59,500 TZS  (+9,500)    │
│  Platform Revenue:  5,500 TZS  (+500)      │
└─────────────────────────────────────────────┘

VERIFICATION (Double-Entry Rule):
Total Debits  = 10,000 + 9,500 + 500 = 20,000 TZS ✓
Total Credits = 10,000 + 9,500 + 500 = 20,000 TZS ✓
```

? Escrow Mechanism

Escrow Lifecycle



Escrow Account Structure

```
┌─────────────────────────────────────────────────────┐
│  EscrowAccountEntity                                │
├─────────────────────────────────────────────────────┤
│  • escrowNumber: "ESC-2025-000123"                  │
│  • checkoutSessionId: UUID                          │
│  • sessionDomain: PRODUCT | EVENT                   │
│  • buyer: AccountEntity                             │
│  • seller: AccountEntity                            │
│  • totalAmount: 50,000 TZS                          │
│  • platformFeePercentage: 5.00%                     │
│  • platformFeeAmount: 2,500 TZS                     │
│  • sellerAmount: 47,500 TZS                         │
│  • status: HELD | RELEASED | REFUNDED | DISPUTED    │
```

```
│  • ledgerAccountId: UUID                                  │
│  • createdAt: 2025-02-16 10:30:00                        │
│  • releasedAt: null                                       │
└──────────────────────────────────────────────────────────┘
```

Fee Calculation

```
┌──────────────────────────────────────────┐
│  Escrow Fee Calculation                  │
├──────────────────────────────────────────┤
│                                          │
│  Total Amount:        50,000 TZS         │
│  Platform Fee (5%):    2,500 TZS         │
│  ──────────────────────────              │
│  Seller Receives:     47,500 TZS         │
│                                          │
│  Formula:                                │
│  platformFee = total × (5 / 100)         │
│  sellerAmount = total - platformFee      │
└──────────────────────────────────────────┘
```

Escrow Operations

1. Hold Money (Payment)

```
escrowService.holdMoney(session, payer, payee, amount)
     ↓
1. Create escrow entity
2. Calculate fees
3. Create escrow ledger account
4. Move money: payer wallet → escrow
5. Create transaction history
     ↓
Result: Money held in escrow
```

2. Release Money (Delivery Confirmed)

```
escrowService.releaseMoney(escrowId)
     ↓
```

```
1. Validate escrow.status = HELD
2. Get seller wallet ledger account
3. Get platform revenue account
4. Split money:
   • 95% → seller wallet
   • 5%  → platform revenue
5. Update escrow.status = RELEASED
6. Create transaction history
     ↓
Result: Money distributed
```

3. Refund Money (Order Cancelled)

```
escrowService.refundMoney(escrowId)
     ↓
1. Validate escrow.status = HELD or DISPUTED
2. Get buyer wallet ledger account
3. Move money: escrow → buyer wallet
4. Update escrow.status = REFUNDED
5. Create transaction history
     ↓
Result: Money returned to buyer
```

? Domain-Specific Handling

Product Domain

```
┌──────────────────────────────────────────────────────┐
│  PRODUCT CHECKOUT SESSION TYPES                      │
└──────────────────────────────────────────────────────┘


1. REGULAR_DIRECTLY
   ═══════════════
   • Single product, direct purchase
   • "Buy Now" button
   • No cart involved
```

Flow:

User → Product Page → Buy Now → Checkout → Payment

↓

Create Order Immediately

2. REGULAR_CART

═══════════════

• Multiple products from cart

• Cart-based purchase

Flow:

User → Add to Cart → View Cart → Checkout → Payment

↓

Create Order + Clear Cart

3. GROUP_PURCHASE

═══════════════

• Group buying deal

• Wait for minimum participants

• Uses groupPrice (discounted)

Flow:

User → Join/Create Group → Payment (wallet only)

↓

Group Instance Created/Joined

↓

Wait for group completion → Create Orders

4. INSTALLMENT

═══════════════

• Buy now, pay in installments

• Down payment required

• Monthly payments scheduled

Flow:

User → Select Plan → Pay Down Payment (wallet only)

↓

Installment Agreement Created

↓

```
                Schedule Future Payments → Create Order (maybe)
```

Event Domain

```
┌─────────────────────────────────────────────────────────┐
│  EVENT CHECKOUT SESSION                                 │
└─────────────────────────────────────────────────────────┘


TICKET TYPES:
═══════════

1. PAID - Regular paid ticket
2. FREE - Free entry ticket
3. DONATION - Pay-what-you-want ticket

Flow:
User → Select Tickets → Add Attendee Info → Payment
                                                ↓
                                  ┌─────────────┴─────────────┐
                                  ↓                           ↓
                             PAID TICKET                 FREE TICKET
                                  ↓                           ↓
                          Create Escrow              Skip Escrow (amount = 0)
                          Move Money                          ↓
                                  ↓                           ↓
                          ┌───────┴───────────────────────────┘
                          ↓
                Publish PaymentCompletedEvent
                          ↓
                EventPaymentCompletedListener
                          ↓
                ┌─────────────────────────┐
                │ 1. Create Booking       │
                │ 2. Reserve Tickets      │
                │ 3. Generate QR Codes    │
                │ 4. Send to Attendees    │
                └─────────────────────────┘
```

? Transaction History

Transaction History provides a user-friendly view of all financial activities.

Structure

```
┌──────────────────────────────────────────────────────┐
│  TransactionHistory Entity                           │
├──────────────────────────────────────────────────────┤
│  • transactionRef: "#2025T000123"                    │
│  • account: AccountEntity (user)                     │
│  • type: PRODUCT_PURCHASE | WALLET_TOPUP | ...       │
│  • direction: DEBIT | CREDIT | NEUTRAL               │
│  • amount: 50,000 TZS                                │
│  • title: "Product Purchase"                         │
│  • description: "Payment for order #..."             │
│  • ledgerEntryId: UUID (link to ledger)              │
│  • referenceType: "PRODUCT_CHECKOUT"                 │
│  • referenceId: UUID                                 │
│  • status: COMPLETED | PENDING | FAILED              │
│  • createdAt: 2025-02-16 10:30:00                   |
└──────────────────────────────────────────────────────┘
```

Transaction Types

```
┌──────────────────────────────────────────────────┐
│  TRANSACTION TYPES                               │
├──────────────────────────────────────────────────┤
│                                                  │
│  WALLET OPERATIONS:                              │
│  • WALLET_TOPUP          (CREDIT)                │
│  • WALLET_WITHDRAWAL     (DEBIT)                 │
│                                                  │
│  PRODUCT PURCHASES:                              │
│  • PRODUCT_PURCHASE      (DEBIT)   - paid        │
│  • FREE_PRODUCT          (NEUTRAL) - free        │
│  • CASH_PRODUCT_PAYMENT  (NEUTRAL) - cash        │
│  • PRODUCT_REFUND        (CREDIT)  - refunded    │
```

```
│                                                  │
│  EVENT PURCHASES:                                │
│  • EVENT_TICKET_PURCHASE (DEBIT)   - paid ticket  │
│  • FREE_EVENT_TICKET     (NEUTRAL) - free ticket  │
│  • CASH_EVENT_TICKET     (NEUTRAL) - cash ticket  │
│  • EVENT_TICKET_REFUND   (CREDIT)  - refunded     │
│                                                  │
│  SELLER OPERATIONS:                              │
│  • SALE                  (CREDIT)  - money earned  │
│  • SALE_REFUND           (DEBIT)   - refund issued│
│                                                  │
│  PLATFORM OPERATIONS:                            │
│  • PLATFORM_FEE_COLLECTED (CREDIT) - platform fee │
│                                                  │
│  ESCROW OPERATIONS:                              │
│  • ESCROW_HOLD           (DEBIT)   - for tracking │
│  • ESCROW_RELEASE        (CREDIT)  - for tracking │
│  • ESCROW_REFUND         (CREDIT)  - for tracking │
└──────────────────────────────────────────────────┘
```

Direction Types

```
┌──────────────────────────────────────────┐
│  TRANSACTION DIRECTIONS                  │
├──────────────────────────────────────────┤
│                                          │
│  DEBIT (−)                               │
│  • Money leaving your wallet             │
│  • Displayed as negative amount          │
│  • Examples: Purchases, Withdrawals      │
│                                          │
│  CREDIT (+)                              │
│  • Money entering your wallet            │
│  • Displayed as positive amount          │
│  • Examples: Top-ups, Refunds, Sales     │
│                                          │
│  NEUTRAL (◆)                             │
│  • No wallet impact                      │
│  • Used for: Free items, Cash payments   │
```

```
│  • Amount shown but doesn't affect balance │
└────────────────────────────────────────────┘
```

? Key Design Patterns

1. Strategy Pattern

```
Purpose: Handle domain-specific logic

Components:
• SessionMetadataExtractor - Extract payee
• PostPaymentHandler - Handle post-payment

Benefit: Easy to add new domains (Subscriptions, etc.)
```

2. Contract Interface (PayableCheckoutSession)

```
Purpose: Universal payment interface

Benefit:
• Payment system domain-agnostic
• Add new checkout types easily
• Type-safe polymorphism
```

3. Observer Pattern (Events)

```
Purpose: Async processing after payment

Components:
• PaymentCompletedEvent
• ProductPaymentCompletedListener
• EventPaymentCompletedListener

Benefit:
• Non-blocking payment response
```

```
• Decoupled order/booking creation
• Easy to add new listeners
```

4. Service Layer Pattern

```
Purpose: Separation of concerns

Layers:
Controller → Service → Orchestrator → Processor → Core

Benefit:
• Clear responsibilities
• Easy testing
• Maintainable code
```

5. Repository Pattern

```
Purpose: Data access abstraction

Components:
• ProductCheckoutSessionRepo
• EventCheckoutSessionRepo
• EscrowAccountRepo
• LedgerAccountRepo

Benefit:
• Database-agnostic
• Easy to mock for testing
```

? Security & Validation

Payment Validation Checklist

```
✓ Session status = PENDING_PAYMENT
✓ Session not expired
```

```
✓ Session belongs to authenticated user
✓ Wallet is active
✓ Sufficient balance
✓ Amount > 0
✓ No duplicate payment (no existing escrow)
✓ Payment method allowed for session type
✓ Inventory/tickets available
```

Transaction Safety

```
✓ All money movements in transactions
✓ Double-entry bookkeeping verified
✓ Atomic balance updates
✓ Idempotent operations
✓ Retry mechanism for failed operations
✓ Audit trail via transaction history
```

? Payment Scenarios & Special Cases

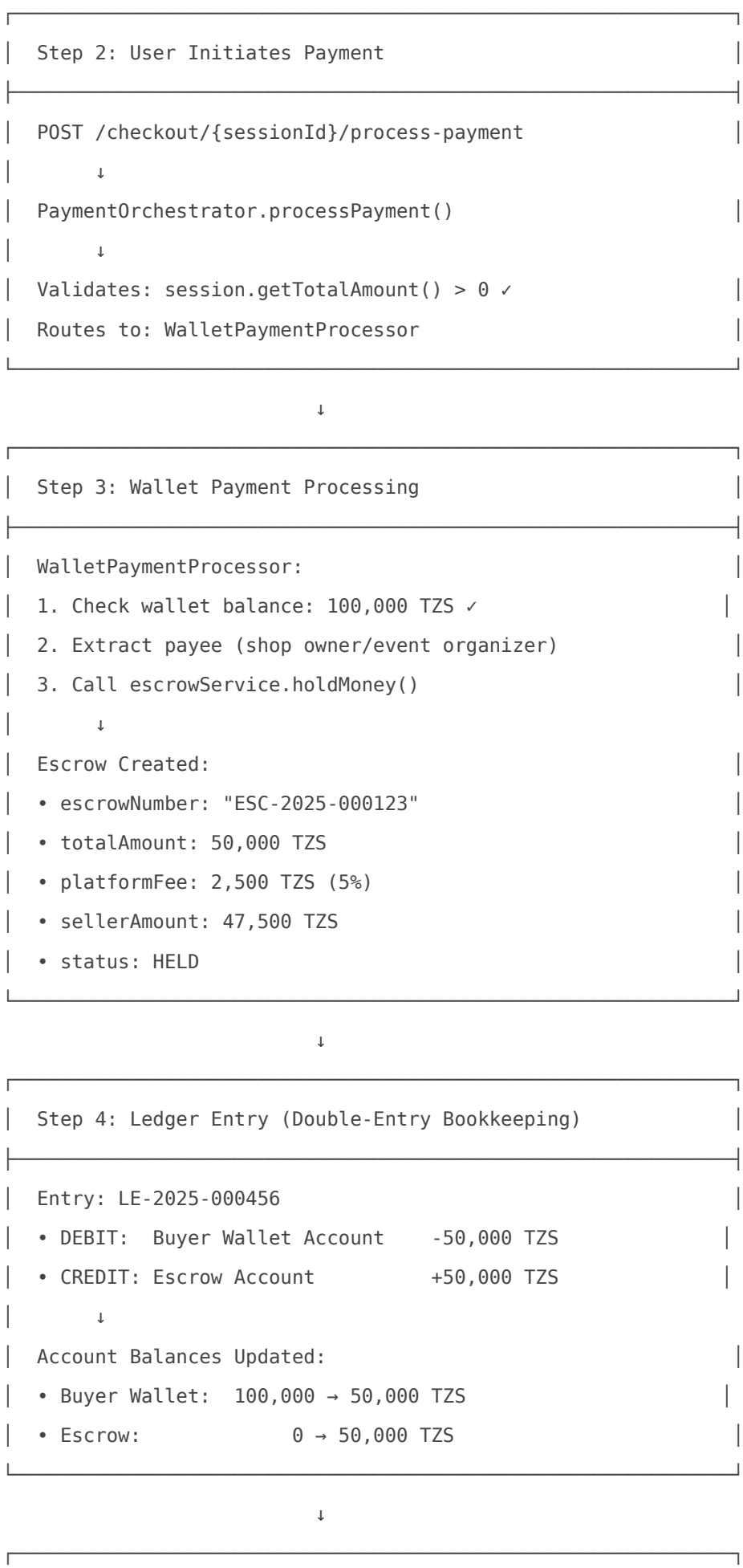
Overview of Payment Types

```
┌──────────────────────────────────────────────────────────────┐
│              PAYMENT SCENARIO MATRIX                         │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  PAID (Regular Payment)                                      │
│  • Amount > 0                                                │
│  • Wallet payment (current)                                  │
│  • External payment (future: M-Pesa, cards, etc.)            │
│  • Creates escrow                                            │
│  • Platform fee collected (5%)                               │
│                                                              │
│  FREE (No Payment)                                           │
│  • Amount = 0                                                │
│  • No wallet deduction                                       │
│  • No escrow created                                         │
```

```
│  • No platform fee                                                    │
│  • Direct order/booking creation                                      │
│                                                                       │
│  CASH (Physical Payment)                                              │
│  • Amount > 0                                                         │
│  • Payment collected physically                                       │
│  • No wallet involvement                                              │
│  • No escrow created                                                  │
│  • No ledger entry                                                    │
│  • Tracked in transaction history only                                │
│                                                                       │
│  DONATION (Pay-What-You-Want)                                         │
│  • Amount >= 0 (user decides)                                         │
│  • Uses wallet payment                                                │
│  • Creates escrow if amount > 0                                       │
│  • Platform fee collected if amount > 0                               │
│  • Limited to 1 per user                                              │
└───────────────────────────────────────────────────────────────────────┘
```

Scenario 1: PAID TICKET/PRODUCT (Standard Flow)

```
USER JOURNEY:
═══════════════

User selects paid ticket/product → Checkout → Pay with Wallet

PAYMENT FLOW:
═══════════════


┌─────────────────────────────────────────────────────────────┐
│  Step 1: Checkout Session Created                           │
├─────────────────────────────────────────────────────────────┤
│  • sessionType: REGULAR_DIRECTLY / EVENT_TICKET_PURCHASE    │
│  • pricing.total: 50,000 TZS                                │
│  • status: PENDING_PAYMENT                                  │
│  • paymentIntent.provider: "WALLET"                         │
└─────────────────────────────────────────────────────────────┘

                            ↓
```


Step 2: User Initiates Payment
POST /checkout/{sessionId}/process-payment
↓
PaymentOrchestrator.processPayment()
↓
Validates: session.getTotalAmount() > 0 ✓
Routes to: WalletPaymentProcessor
↓
Step 3: Wallet Payment Processing
WalletPaymentProcessor:
1. Check wallet balance: 100,000 TZS ✓
2. Extract payee (shop owner/event organizer)
3. Call escrowService.holdMoney()
↓
Escrow Created:
• escrowNumber: "ESC-2025-000123"
• totalAmount: 50,000 TZS
• platformFee: 2,500 TZS (5%)
• sellerAmount: 47,500 TZS
• status: HELD
↓
Step 4: Ledger Entry (Double-Entry Bookkeeping)
Entry: LE-2025-000456
• DEBIT: Buyer Wallet Account -50,000 TZS
• CREDIT: Escrow Account +50,000 TZS
↓
Account Balances Updated:
• Buyer Wallet: 100,000 → 50,000 TZS
• Escrow: 0 → 50,000 TZS
↓

```
│  Step 5: Transaction History Created                          │
├──────────────────────────────────────────────────────────────┤
│  Transaction 1 (Buyer's View):                                │
│  • ref: "#2025T000789"                                        │
│  • type: PRODUCT_PURCHASE / EVENT_TICKET_PURCHASE             │
│  • direction: DEBIT                                           │
│  • amount: 50,000 TZS                                         │
│  • title: "Product Purchase" / "Ticket Purchase"              │
│  • status: COMPLETED                                          │
│       ↓                                                       │
│  Transaction 2 (Admin Tracking):                              │
│  • type: ESCROW_HOLD                                          │
│  • direction: DEBIT                                           │
│  • amount: 50,000 TZS                                         │
└──────────────────────────────────────────────────────────────┘

                            ↓

┌──────────────────────────────────────────────────────────────┐
│  Step 6: Session Updated & Event Published                    │
├──────────────────────────────────────────────────────────────┤
│  • session.status → PAYMENT_COMPLETED                         │
│  • session.escrowId → escrow.id                               │
│  • session.completedAt → now()                                │
│       ↓                                                       │
│  PaymentCompletedEvent published                              │
│       ↓                                                       │
│  Async Listeners:                                             │
│  • Create order/booking                                       │
│  • Send notifications                                         │
│  • Clear cart (if applicable)                                 │
└──────────────────────────────────────────────────────────────┘

                            ↓

┌──────────────────────────────────────────────────────────────┐
│  Step 7: Response to User                                     │
├──────────────────────────────────────────────────────────────┤
│  PaymentResponse:                                             │
│  • success: true                                              │
│  • status: SUCCESS                                            │
│  • message: "Payment processed successfully"                  │
│  • escrowNumber: "ESC-2025-000123"                            │
│  • amountPaid: 50,000 TZS                                     │
```

```
│  • platformFee: 2,500 TZS                                    │
│  • sellerAmount: 47,500 TZS                                  │
└──────────────────────────────────────────────────────────────┘


RESULT:
═══════
✓ Money moved from buyer to escrow
✓ Ledger balanced
✓ Transaction history recorded
✓ Order/booking created (async)
✓ User receives confirmation
```

Scenario 2: FREE TICKET/PRODUCT (Zero Payment)

```
USER JOURNEY:
═════════════
User selects free ticket/product → Checkout → Submit (no payment)


SPECIAL CHARACTERISTICS:
════════════════════════
• pricing.total = 0 TZS
• No wallet deduction
• No escrow created
• No platform fee
• Direct completion


PAYMENT FLOW:
═════════════


┌──────────────────────────────────────────────────────────────┐
│  Step 1: Checkout Session Created                            │
├──────────────────────────────────────────────────────────────┤
│  • ticketPricingType: FREE                                   │
│  • pricing.total: 0 TZS                                      │
│  • status: PENDING_PAYMENT (initial)                         │
│  • paymentIntent: null (no payment needed)                   │
└──────────────────────────────────────────────────────────────┘
```

```
                              ↓
┌──────────────────────────────────────────────────────────────┐
│  Step 2: Automatic Processing (No User Action)               │
├──────────────────────────────────────────────────────────────┤
│  EventCheckoutServiceImpl.createCheckoutSession():           │
│       ↓                                                      │
│  Detects: ticket.ticketPricingType == FREE                   │
│       ↓                                                      │
│  Sets: session.status → PAYMENT_COMPLETED (immediately)      │
│       ↓                                                      │
│  Calls: processFreeTicketCheckout(session)                   │
└──────────────────────────────────────────────────────────────┘
                              ↓
┌──────────────────────────────────────────────────────────────┐
│  Step 3: Free Checkout Processing                            │
├──────────────────────────────────────────────────────────────┤
│  processFreeTicketCheckout():                                │
│  1. session.markAsCompleted()                                │
│  2. session.completedAt → now()                              │
│  3. Save session                                             │
│       ↓                                                      │
│  NO Escrow Creation ✗                                        │
│  NO Wallet Deduction ✗                                       │
│  NO Ledger Entry ✗                                           │
└──────────────────────────────────────────────────────────────┘
                              ↓
┌──────────────────────────────────────────────────────────────┐
│  Step 4: Transaction History (Optional Tracking)             │
├──────────────────────────────────────────────────────────────┤
│  trackFreeTransaction():                                     │
│  • ref: "#2025T000790"                                       │
│  • type: FREE_PRODUCT / FREE_EVENT_TICKET                    │
│  • direction: NEUTRAL (no balance impact)                    │
│  • amount: 0 TZS                                             │
│  • title: "Free Ticket Booking"                              │
│  • ledgerEntryId: null (no ledger)                           │
│  • status: COMPLETED                                         │
└──────────────────────────────────────────────────────────────┘
                              ↓
┌──────────────────────────────────────────────────────────────┐
```

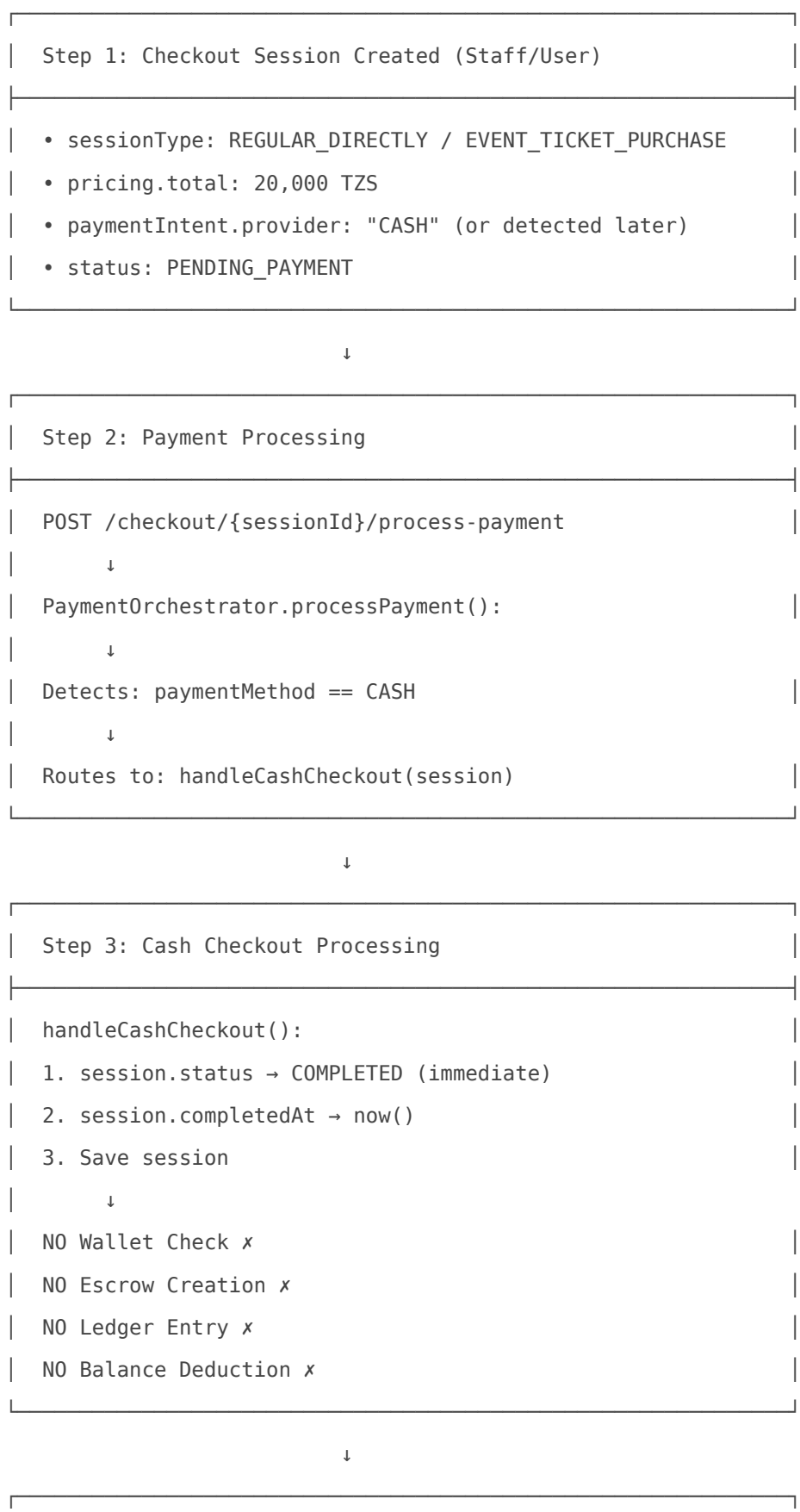
```
│  Step 5: Event Published                                          │
├──────────────────────────────────────────────────────────────────┤
│  PaymentCompletedEvent(                                           │
│    sessionId,                                                     │
│    domain: EVENT,                                                 │
│    session,                                                       │
│    escrow: null  ← NO ESCROW FOR FREE                             │
│  )                                                                │
│       ↓                                                           │
│  EventPaymentCompletedListener:                                   │
│  • Detects: escrow == null (free ticket)                          │
│  • Creates booking                                                │
│  • Reserves tickets                                               │
│  • Generates QR codes                                             │
│  • Sends confirmation emails                                      │
└──────────────────────────────────────────────────────────────────┘

                          ↓

┌──────────────────────────────────────────────────────────────────┐
│  Step 6: Response to User                                         │
├──────────────────────────────────────────────────────────────────┤
│  PaymentResponse:                                                 │
│  • success: true                                                  │
│  • status: SUCCESS                                                │
│  • message: "Free ticket booking confirmed!"                      │
│  • amountPaid: 0 TZS                                              │
│  • escrowId: null                                                 │
│  • escrowNumber: null                                             │
└──────────────────────────────────────────────────────────────────┘


COMPARISON: FREE vs PAID
═══════════════════════════


┌──────────────────────┬──────────────┬──────────────┐
│   Operation          │    PAID      │    FREE      │
├──────────────────────┼──────────────┼──────────────┤
│ Wallet Check         │     YES      │     NO       │
│ Escrow Created       │     YES      │     NO       │
│ Ledger Entry         │     YES      │     NO       │
│ Platform Fee         │     YES      │     NO       │
│ Transaction History  │     YES      │  OPTIONAL    │
```

```
| Order/Booking       |  ASYNC       |   ASYNC      |
| User Balance Impact|     YES      |     NO       |
└─────────────────────┴──────────────┴──────────────┘


RESULT:
═══════
✓ No money moved
✓ No escrow created
✓ Session marked completed immediately
✓ Booking created (async)
✓ User receives free ticket
```

Scenario 3: CASH PAYMENT (Physical Money)

```
USER JOURNEY:
═══════════════
User pays cash at door/venue → Staff creates checkout → Marks as cash

SPECIAL CHARACTERISTICS:
═══════════════════════
• Amount > 0 (but not from wallet)
• Payment collected physically (offline)
• No wallet deduction
• No escrow (money never in system)
• No ledger entry (money outside system)
• Tracked in transaction history for records
• Used for: Door sales, in-person events

IMPLEMENTATION:
═══════════════


Option 1: Payment Method Detection (Current)
──────────────────────────────────────────────
PaymentOrchestrator detects paymentMethod == CASH


Option 2: Pricing Detection (Alternative)
──────────────────────────────────────────
Session has special cashPaymentIndicator flag
```

PAYMENT FLOW:



```
│  Step 4: Transaction History (Record Keeping)                │
├──────────────────────────────────────────────────────────────┤
│  trackCashTransaction():                                     │
│  • ref: "#2025T000791"                                       │
│  • type: CASH_PRODUCT_PAYMENT / CASH_EVENT_TICKET            │
│  • direction: NEUTRAL (no wallet impact)                     │
│  • amount: 20,000 TZS                                        │
│  • title: "Cash Product Payment"                             │
│  • description: "Cash payment for product (Session: ...)"    │
│  • ledgerEntryId: null (no ledger entry)                     │
│  • status: COMPLETED                                         │
└──────────────────────────────────────────────────────────────┘

                            ↓

┌──────────────────────────────────────────────────────────────┐
│  Step 5: Event Published                                     │
├──────────────────────────────────────────────────────────────┤
│  PaymentCompletedEvent(                                      │
│    sessionId,                                                │
│    domain: PRODUCT/EVENT,                                    │
│    session,                                                  │
│    escrow: null  ← NO ESCROW FOR CASH                        │
│  )                                                           │
│       ↓                                                      │
│  Listener processes:                                         │
│  • Creates order/booking                                     │
│  • NO escrow release needed                                  │
│  • NO platform fee collected                                 │
└──────────────────────────────────────────────────────────────┘

                            ↓

┌──────────────────────────────────────────────────────────────┐
│  Step 6: Response to User                                    │
├──────────────────────────────────────────────────────────────┤
│  PaymentResponse:                                            │
│  • success: true                                             │
│  • status: SUCCESS                                           │
│  • message: "Cash payment confirmed!"                        │
│  • paymentMethod: CASH                                       │
│  • amountPaid: 20,000 TZS                                    │
│  • escrowId: null                                            │
│  • escrowNumber: null                                        │
```

└──┘

USE CASES:

1. Event Door Sales
   - User arrives at venue
   - Pays cash at entrance
   - Staff creates checkout & marks as cash
   - User receives ticket immediately

2. In-Store Purchase
   - Customer buys product in physical store
   - Pays with physical cash
   - Staff records sale in system
   - Order created for tracking

3. COD (Cash on Delivery)
   - User orders online
   - Chooses cash payment
   - Courier collects cash on delivery
   - System updated after confirmation

COMPARISON: CASH vs WALLET

Operation	WALLET	CASH
Money in System	YES	NO
Wallet Deduction	YES	NO
Escrow Created	YES	NO
Ledger Entry	YES	NO
Platform Fee	YES	NO*
Transaction History	YES	YES
Balance Impact	YES	NO
Use Case	Online	Physical

* Platform fee for cash could be handled separately
  in business reconciliation, outside the system

RESULT:
═══════

✓ No money in digital system
✓ No wallet/ledger impact
✓ Transaction recorded for tracking
✓ Order/booking created
✓ Physical cash collected separately

Scenario 4: DONATION TICKET (Pay-What-You-Want)

USER JOURNEY:
═══════════

User sees donation event → Decides amount → Pays via wallet

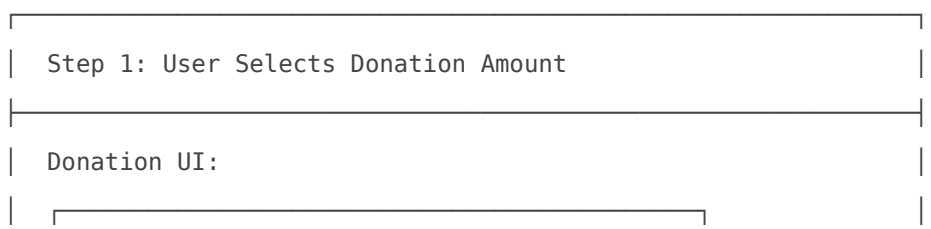
SPECIAL CHARACTERISTICS:
═══════════════════════

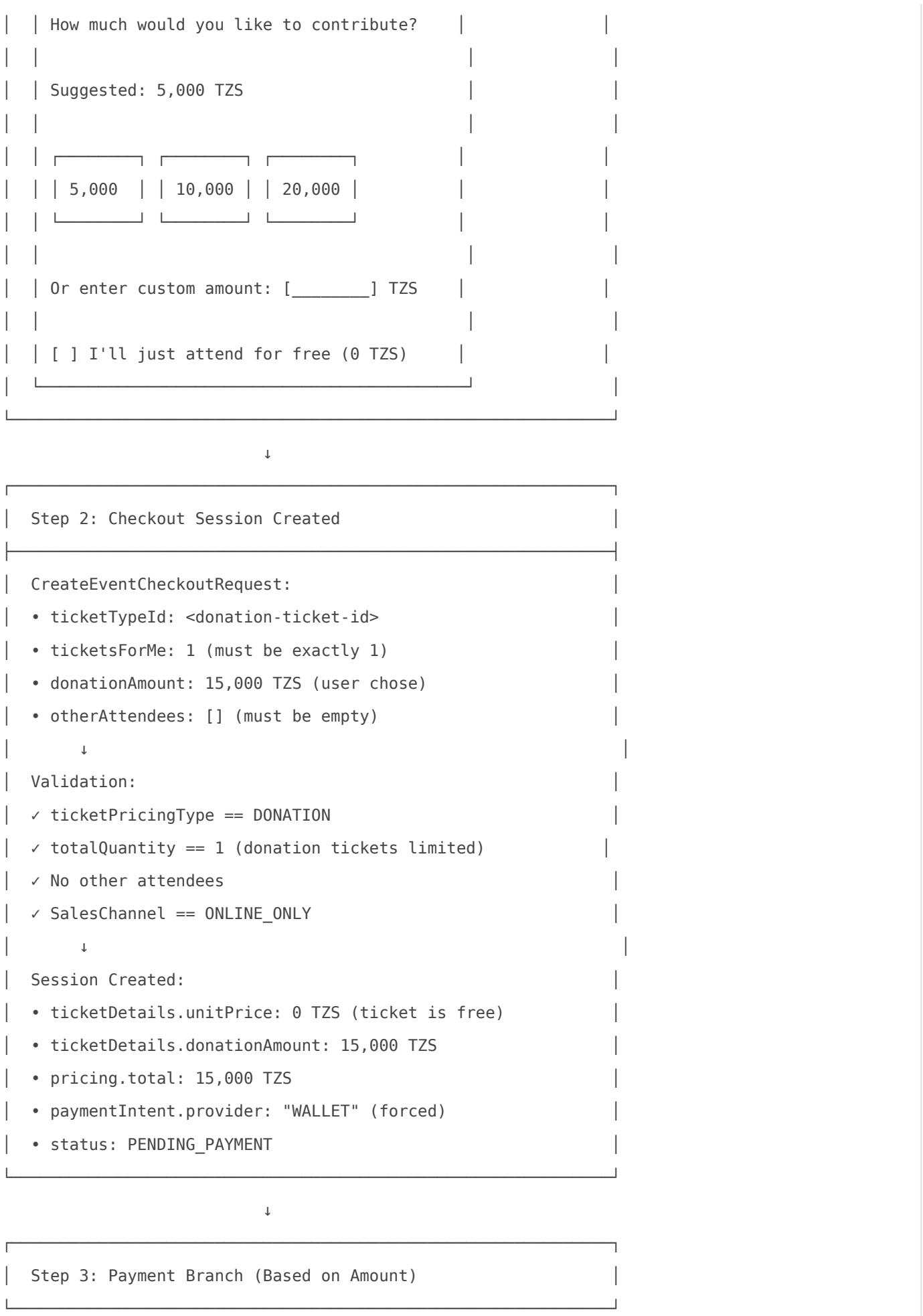
• Amount >= 0 (user decides)
• Can be 0 (free) or any amount
• Only available for EVENT domain (not products)
• Wallet payment required (no external payments)
• Limited to 1 ticket per person
• No tickets for other attendees
• Online purchase only (no door sales)

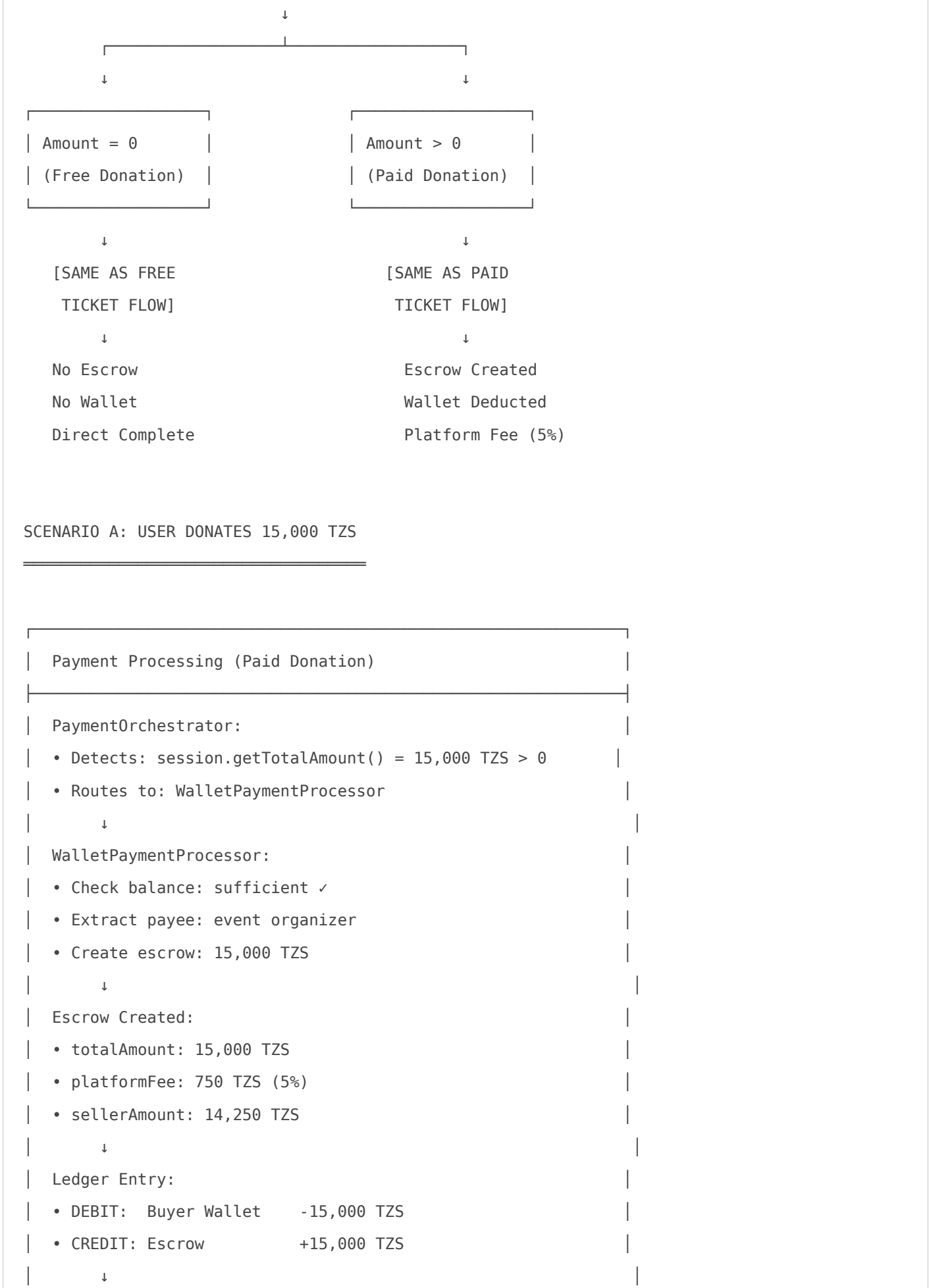
TICKET TYPE:
═══════════

ticketPricingType: DONATION

PAYMENT FLOW:
═══════════




How much would you like to contribute?
Suggested: 5,000 TZS
5,000
10,000
20,000
Or enter custom amount: [_____] TZS
[ ] I'll just attend for free (0 TZS)
Step 2: Checkout Session Created
CreateEventCheckoutRequest:
• ticketTypeId: <donation-ticket-id>
• ticketsForMe: 1 (must be exactly 1)
• donationAmount: 15,000 TZS (user chose)
• otherAttendees: [] (must be empty)
Validation:
✓ ticketPricingType == DONATION
✓ totalQuantity == 1 (donation tickets limited)
✓ No other attendees
✓ SalesChannel == ONLINE_ONLY
Session Created:
• ticketDetails.unitPrice: 0 TZS (ticket is free)
• ticketDetails.donationAmount: 15,000 TZS
• pricing.total: 15,000 TZS
• paymentIntent.provider: "WALLET" (forced)
• status: PENDING_PAYMENT
Step 3: Payment Branch (Based on Amount)



SCENARIO A: USER DONATES 15,000 TZS

```
│  Transaction History:                                          │
│  • type: DONATION_EVENT_TICKET                                 │
│  • direction: DEBIT                                            │
│  • amount: 15,000 TZS                                          │
│  • title: "Donation Ticket"                                    │
│  • metadata: { donationAmount: 15000, suggested: 5000 }        │
└────────────────────────────────────────────────────────────────┘


SCENARIO B: USER CHOOSES FREE (0 TZS)
═══════════════════════════════════════


┌────────────────────────────────────────────────────────────────┐
│  Payment Processing (Free Donation)                            │
├────────────────────────────────────────────────────────────────┤
│  PaymentOrchestrator:                                          │
│  • Detects: session.getTotalAmount() = 0 TZS                   │
│  • Routes to: handleFreeCheckout()                             │
│       ↓                                                         │
│  Free Processing:                                              │
│  • NO wallet check                                             │
│  • NO escrow                                                   │
│  • NO ledger entry                                             │
│  • Direct session completion                                   │
│       ↓                                                         │
│  Transaction History (Optional):                               │
│  • type: FREE_EVENT_TICKET (or DONATION_EVENT_TICKET)          │
│  • direction: NEUTRAL                                          │
│  • amount: 0 TZS                                               │
│  • title: "Free Donation Ticket"                               │
│  • metadata: { donationAmount: 0, suggested: 5000 }            │
└────────────────────────────────────────────────────────────────┘


VALIDATION RULES FOR DONATION:
═══════════════════════════════


✓ Ticket type must be DONATION
✓ Quantity must be exactly 1
✓ No other attendees allowed
✓ Sales channel must be ONLINE_ONLY
```

✓ Payment method forced to WALLET (no cash, no external)

✓ maxQuantityPerUser enforced (1 per person)

Checked in: EventCheckoutValidations.validateDonationTicket()

COMPARISON: DONATION vs REGULAR PAID

Feature	REGULAR	DONATION
Fixed Price	YES	NO
User Sets Amount	NO	YES
Can be Free (0)	NO	YES
Quantity Limit	Variable	1 Only
Other Attendees	Allowed	Blocked
Sales Channel	Any	Online Only
Payment Method	Any	Wallet Only
Escrow (if > 0)	YES	YES
Platform Fee (if >0)	YES	YES

USE CASES:

1. Charity Event
   - Event organizer hosts fundraiser
   - Tickets free, but donations welcomed
   - Users choose contribution amount
   - Platform collects fee only on donations > 0

2. Community Event
   - Local meetup or workshop
   - Entry is free
   - Optional donation to cover costs
   - Organizer receives donations minus platform fee

3. Awareness Campaign
   - NGO hosts event
   - Free entry for all

```
   • Donations support the cause
   • Transparent tracking via transaction history

RESULT:
═══════
✓ Flexible payment (free or any amount)
✓ User decides contribution
✓ Platform fee only on donations > 0
✓ Limited to 1 per person
✓ Wallet payment only
✓ Same escrow/ledger logic if amount > 0
```

? Payment Scenario Decision Tree

```
User Initiates Payment
         ↓
    ┌────┴────┐
    ↓         ↓
Amount > 0?  Amount = 0?
    ↓         ↓
   YES       [FREE FLOW]
    ↓         • No escrow
    ↓         • No wallet
    ↓         • Complete immediately
    ↓         • Transaction: NEUTRAL
Payment
Method?
    ↓
    ├─ WALLET ──→ [PAID FLOW]
    │              • Check balance
    │              • Create escrow
    │              • Ledger entry
    │              • Platform fee (5%)
    │              • Transaction: DEBIT
    │
    ├─ CASH ────→ [CASH FLOW]
    │              • No wallet
    │              • No escrow
```

```
    │               • No ledger
    │               • Complete immediately
    │               • Transaction: NEUTRAL (tracking)
    │
    └─ EXTERNAL ──→ [EXTERNAL FLOW]
                    • (Not yet implemented)
                    • M-Pesa / Cards / etc.
                    • Would follow PAID flow
```

? Money Movement Summary by Scenario

```
┌──────────────────────────────────────────────────────────────┐
│              MONEY FLOW BY PAYMENT TYPE                      │
└──────────────────────────────────────────────────────────────┘


PAID (Wallet):
═══════════════

    Buyer Wallet  ──[amount]──→  Escrow Account
                                        ↓
                                        │
                        ┌───────────────┴──────────────┐
                        ↓                              ↓
                Seller Wallet                  Platform Revenue
                [95% of amount]                 [5% of amount]


FREE:
═════

    (No money movement)
    User → Direct Order/Booking


CASH:
═════

    (Physical money, outside system)
    User → [Physical Cash] → Seller/Organizer
    System: Records transaction for tracking only


DONATION (Amount > 0):
```

```
═══════════════════
    [Same as PAID flow]
    Buyer Wallet  ──[user-chosen amount]──→  Escrow Account
                                                    ↓
                                                    |
                                    ┌───────────────┴──────────────┐
                                    ↓                              ↓
                            Organizer Wallet                Platform Revenue
                            [95% of donation]                [5% of donation]

DONATION (Amount = 0):
═══════════════════
    [Same as FREE flow]
    (No money movement)
    User → Direct Booking
```

? Transaction History by Scenario

```
┌──────────────────────────────────────────────────────────────┐
│         TRANSACTION HISTORY ENTRIES BY SCENARIO              │
└──────────────────────────────────────────────────────────────┘

PAID Purchase (Product):
═══════════════════════
```

Buyer sees:

- Type: PRODUCT_PURCHASE
- Direction: DEBIT (-)
- Amount: 50,000 TZS
- Balance impact: YES

Seller sees (after escrow release):

- Type: SALE
- Direction: CREDIT (+)
- Amount: 47,500 TZS
- Balance impact: YES

Platform sees:

- Type: PLATFORM_FEE_COLLECTED
- Direction: CREDIT (+)

- Amount: 2,500 TZS

PAID Ticket (Event):

═══════════════════

Buyer sees:

- Type: EVENT_TICKET_PURCHASE
- Direction: DEBIT (-)
- Amount: 30,000 TZS
- Balance impact: YES

Organizer sees (after escrow release):

- Type: SALE
- Direction: CREDIT (+)
- Amount: 28,500 TZS
- Balance impact: YES

Platform sees:

- Type: PLATFORM_FEE_COLLECTED
- Direction: CREDIT (+)
- Amount: 1,500 TZS

FREE Product:

═══════════

Buyer sees:

- Type: FREE_PRODUCT
- Direction: NEUTRAL (◆)
- Amount: 0 TZS
- Balance impact: NO

FREE Ticket:

═══════════

Buyer sees:

- Type: FREE_EVENT_TICKET
- Direction: NEUTRAL (◆)
- Amount: 0 TZS
- Balance impact: NO

CASH Product:

═══════════

Buyer sees:

- Type: CASH_PRODUCT_PAYMENT
- Direction: NEUTRAL (◆)
- Amount: 20,000 TZS
- Balance impact: NO
- Note: "Physical cash payment"

CASH Ticket:

═══════════

Buyer sees:

- Type: CASH_EVENT_TICKET
- Direction: NEUTRAL (◆)
- Amount: 15,000 TZS
- Balance impact: NO
- Note: "Cash payment at door"

DONATION (Paid):

═══════════════

Buyer sees:

- Type: DONATION_EVENT_TICKET
- Direction: DEBIT (-)
- Amount: 10,000 TZS (user chose)
- Balance impact: YES
- Metadata: { suggested: 5000, donated: 10000 }

Organizer sees (after escrow release):

- Type: SALE
- Direction: CREDIT (+)
- Amount: 9,500 TZS
- Balance impact: YES

DONATION (Free):

═══════════════

Buyer sees:

- Type: FREE_EVENT_TICKET (or DONATION_EVENT_TICKET)
- Direction: NEUTRAL (◆)
- Amount: 0 TZS
- Balance impact: NO
- Metadata: { suggested: 5000, donated: 0 }

? Code Implementation Points

PaymentOrchestrator Detection Logic

```
@Override
@Transactional
public PaymentResponse processPayment(PaymentRequest request)
        throws ItemNotFoundException, RandomExceptions, BadRequestException {

    PayableCheckoutSession session = checkoutSessionService.findCheckoutSession(
        request.getCheckoutSessionId(),
        request.getSessionDomain()
    );

    // ========================================
    // SCENARIO DETECTION
    // ========================================

    // FREE Detection
    if (session.getTotalAmount().compareTo(BigDecimal.ZERO) == 0) {
        return handleFreeCheckout(session);
    }

    // Payment Method Detection
    PaymentMethod paymentMethod = determinePaymentMethod(session, request);

    // CASH Detection
    if (paymentMethod == PaymentMethod.CASH) {
        return handleCashCheckout(session);
    }
```

```
    // PAID (Wallet/External) Detection
    return routeToProcessor(session, paymentMethod);
}
```

Event Ticket Type Validation

```
// EventCheckoutValidations.validateTicketTypeAndPrice()

switch (ticketPricingType) {
    case FREE -> validateFreeTicket(ticket, request);
    case PAID -> validatePaidTicket(ticket, request);
    case DONATION -> validateDonationTicket(ticket, request);
}

private void validateDonationTicket(...) {
    // Total quantity must be exactly 1
    if (totalQuantity > 1) {
        throw new BadRequestException(
            "Donation tickets are limited to 1 per person");
    }

    // No other attendees
    if (request.getOtherAttendees() != null
        && !request.getOtherAttendees().isEmpty()) {
        throw new BadRequestException(
            "Donation tickets cannot be purchased for other attendees");
    }

    // Online only
    if (ticket.getSalesChannel() != SalesChannel.ONLINE_ONLY) {
        throw new BadRequestException(
            "Donation tickets can only be purchased online");
    }
}
```

? Data Consistency

ACID Properties Maintained

```
Atomicity:
• Ledger entries in transactions
• Balance updates atomic
• No partial money movements

Consistency:
• Double-entry rule enforced
• Total debits = Total credits
• Balance integrity checks

Isolation:
• Transaction-level isolation
• No concurrent balance corruption

Durability:
• All operations persisted
• Audit trail maintained
```

? Scalability Considerations

Async Processing

```
Payment → Return immediately → Process in background

Benefits:
• Fast API response
• Better user experience
• Handles spike loads
```

Event-Driven Architecture

```
Payment success → Publish event → Multiple listeners

Benefits:
```

```
• Loosely coupled
• Easy to add features
• Horizontal scaling
```

Separate Read/Write Models

```
Write: Ledger entries (normalized)
Read: Transaction history (denormalized)

Benefit:
• Fast queries
• Optimized for each use case
```

? Summary

Key Strengths of the Architecture

1. **Universal Payment System**
   - Handles multiple domains (Products, Events)
   - Easy to extend to new domains
2. **Financial Integrity**
   - Double-entry bookkeeping
   - Escrow protection
   - Audit trail
3. **Scalability**
   - Async processing
   - Event-driven
   - Loosely coupled
4. **Flexibility**
   - Strategy pattern for domain logic
   - Multiple payment methods support
   - Multiple checkout types
5. **Security**
   - Transaction-level safety
   - Validation at every step
   - Balance integrity

? Key Classes Reference

```
CHECKOUT:
• ProductCheckoutSessionEntity
• EventCheckoutSessionEntity
• PayableCheckoutSession (interface)

PAYMENT:
• PaymentOrchestrator
• WalletPaymentProcessor
• ExternalPaymentProcessor

STRATEGY:
• SessionMetadataExtractor
• PostPaymentHandler

FINANCIAL:
• EscrowService
• LedgerService
• WalletService
• TransactionHistoryService

ENTITIES:
• EscrowAccountEntity
• LedgerAccountEntity
• LedgerEntryEntity
• WalletEntity
• TransactionHistory

EVENTS:
• PaymentCompletedEvent
• ProductPaymentCompletedListener
• EventPaymentCompletedListener
```

End of Documentation

This architecture provides a solid foundation for a multi-domain payment system with financial integrity, scalability, and extensibility built in from the ground up.