Blockchain Developer Guide: Web3 Bill Splitting App on Aptos

Project Overview

This document outlines the complete smart contract implementation for a Web3 bill-splitting application on Aptos that leverages existing protocols, particularly **Aptos native multisig accounts**, to avoid reinventing the wheel.

Architecture & Integration with Existing Protocols

1. Aptos Native Multisig Integration

Instead of building custom multisig logic, we leverage **Aptos native multisig accounts (MultiEd25519)** which provide:

- Protocol-level signature verification
- Built-in threshold signature support
- Secure key management without custom smart contracts
- Event emission for off-chain monitoring

2. Smart Contract Modules Created

Module 1: `bill_splitter.move` (Core Logic)

Purpose: Main bill session management and payment processing

Key Features:

- Bill session creation with native multisig account generation
- Participant management with customizable amounts
- Multi-signature collection using Aptos native multisig
- USDC stablecoin payment processing
- Event emission for real-time sync with backend/frontend

Integration Points:

- Uses `multisig_account` module for signature collection
- Emits events consumed by backend API
- Handles USDC coin transfers for settlements

Module 2: `usdc_utils.move` (Stablecoin Support)

Purpose: USDC integration and currency conversion utilities

Key Features:

- USDC coin initialization for testnet
- Exchange rate management for fiat conversion
- Coin manipulation utilities (split, merge, extract)
- Testing functions for hackathon demo

Module 3: `test_helpers.move` (Hackathon Support)

Purpose: Demo setup and testing utilities

Key Features:

- Complete hackathon demo initialization
- Participant funding and registration
- Pre-configured test scenarios
- Stress testing capabilities

Detailed Work Breakdown (18 Hours)

Phase 1: Smart Contract Design (1.5 hours)

Tasks:

1. Define data structures for bill sessions and participants
2. Design multisig integration using Aptos native accounts
3. Plan event schema for backend/frontend synchronization
4. Design USDC payment flow and settlement logic

Deliverables:

- Contract architecture document
- Data structure definitions
- Event emission schema

Phase 2: Core Contract Implementation (3 hours)

Tasks:

1. Implement `BillSession` resource with participant management
2. Integrate Aptos native multisig account creation
3. Implement signature collection and validation

4. Build USDC payment processing functions
5. Add event emission for all state changes

Key Functions Implemented:

```
public entry fun create_bill_session(...)
public entry fun sign_bill_agreement(...)
public entry fun submit_payment(...)
public entry fun update_participant_amount(...)
```

Integration with Existing Protocols:

- Uses `multisig_account::create_with_owners()` for multisig setup
- Leverages `coin` module for USDC transfers
- Utilizes `event` framework for real-time updates

Phase 3: USDC Integration (1 hour)

Tasks:

1. Implement USDC coin registration and utilities
2. Create exchange rate management for fiat conversion
3. Build coin manipulation helpers for bill splitting
4. Add testing functions for demo setup

Phase 4: Testing & Validation (1.5 hours)

Tasks:

1. Write comprehensive unit tests for all functions
2. Test multisig signature collection scenarios
3. Validate USDC payment flows
4. Test edge cases (partial payments, signature failures)

Test Scenarios:

- Multiple participants signing agreements
- Payment submission with exact and excess amounts
- Bill settlement with all participants paid
- Error handling for invalid states

Phase 5: Deployment & Demo Setup (1 hour)

Tasks:

1. Deploy contracts to Aptos testnet
2. Initialize USDC coin for testing
3. Create demo bill sessions
4. Fund test participants with USDC
5. Provide contract addresses and ABIs to backend team

Deployment Script: Complete bash script for one-command setup

Backend/Frontend Integration Points

Event Schema for Real-Time Updates

```
struct SessionCreatedEvent {
    session_id: String,
    merchant_address: address,
    multisig_address: address,
    total_amount: u64,
    required_signatures: u64,
}

struct ParticipantAddedEvent {
    session_id: String,
    participant_address: address,
    amount_owed: u64,
}

struct BillApprovedEvent {
    session_id: String,
    signatures_collected: u64,
}

struct PaymentReceivedEvent {
    session_id: String,
    participant_address: address,
    amount_paid: u64,
    remaining_amount: u64,
}
```

View Functions for Frontend Queries

```
#[view]
public fun get_bill_session(session_id: String): (...)

#[view]
public fun get_participants(session_id: String): vector&lt;Participant&gt;
```

```
#[view]
public fun has_participant_signed(session_id: String, participant_address: address): bool

#[view]
public fun has_participant_paid(session_id: String, participant_address: address): bool
```

Backend API Integration Requirements

Required API Endpoints:

1. `POST /api/bills` - Create new bill session
2. `GET /api/bills/{id}` - Get bill status
3. `POST /api/bills/{id}/sign` - Submit signature
4. `POST /api/bills/{id}/pay` - Submit payment
5. `GET /api/bills/{id}/participants` - Get participant status

Blockchain Event Monitoring:

- Backend should listen to contract events using Aptos SDK
- Real-time updates to frontend via WebSocket or polling
- State synchronization between blockchain and database

Contract Addresses & Configuration

Testnet Deployment:

```
Contract Address: 0x{DEPLOYED_ADDRESS}
Network: Testnet
Node URL: https://fullnode.testnet.aptoslabs.com/v1
Faucet URL: https://faucet.testnet.aptoslabs.com
```

Demo Configuration:

```
Demo Bill ID: "HACKATHON_DEMO_BILL"
Test USDC Amount: 200 USDC per participant
Required Signatures: All participants
```

Security Considerations

1. **Multisig Security**: Uses Aptos native multisig for proven security
2. **Payment Validation**: Validates participant eligibility and payment amounts
3. **State Management**: Prevents double payments and invalid state transitions
4. **Access Controls**: Merchant-only functions for bill management
5. **Event Integrity**: All state changes emit events for auditability

Testing & Demo Instructions

1. **Deploy contracts** using provided script: `./deploy.sh`
2. **Initialize system** with test data and USDC
3. **Create demo bill** with pre-configured participants
4. **Test signature flow** with all participants signing
5. **Submit payments** and verify settlement
6. **Monitor events** for real-time updates

Next Steps & Enhancements

Post-Hackathon Improvements:

1. Gas optimization for batch operations
2. Advanced dispute resolution mechanisms
3. Integration with more stablecoins (USDT, DAI)
4. Cross-chain payment support
5. Advanced fee structures and revenue sharing

Production Considerations:

1. Mainnet deployment with proper USDC contract
2. Oracle integration for real-time exchange rates
3. KYC/AML compliance for large transactions
4. Advanced multisig policies (time locks, recovery)
5. Integration with existing payment processors

This implementation leverages proven Aptos protocols while providing a complete, hackathon-ready solution for Web3 bill splitting with multisig agreements and stablecoin settlements.