Fiat-Crypto On-Ramp/Off-Ramp Home Assignment

Document Information	Details
Version	1.0.0
Date	September 2025
Base URL	`https://api.example.com`
Author	Tony Aizize
Email	aibiertony@gmail.com
Phone	+65 81708060
LinkedIn	linkedin.com/in/tony007

Table of Contents

Core Requirements

Overview

The Fiat-Crypto On-Ramp/Off-Ramp API provides a comprehensive solution for converting between fiat currencies and stablecoins. It supports multiple payment rails, bank integrations, crypto provider integrations, and complete compliance with KYC/AML requirements.

Key Features

- **Unified Ramp Operations**: Single API for both on-ramp (fiat → stablecoin) and off-ramp (stablecoin → fiat)
- **Multiple Payment Rails**: ACH, SEPA, PIX, UPI, M-Pesa, SWIFT, Card processing, DBS (FAST SGD), ICBC CNY
- **Bank Integrations**: Direct integration with major banks (DBS, JPM, HSBC, etc.)
- **Crypto Provider Integrations**: Support for Circle USDC, Tether USDT, Kraken USDK/USDKY, and more
- **Advanced Compliance**: Built-in KYC/AML validation, travel rule compliance, and transaction monitoring
- **Enterprise Security**: MPC/Multisig custody solutions with hardware security modules
- **Treasury Management**: Automated liquidity optimization and yield generation

- **Real-time Status**: Webhook notifications for transaction updates
- **Scalable**: Pagination, rate limiting, and batch operations

Supported Operations

On-Ramp (Fiat → Stablecoin)

1. User requests exchange rate quote for fiat → stablecoin conversion
2. User books the quoted rate (valid for limited time)
3. User submits on-ramp request with booked rate
4. System validates KYC/AML compliance
5. Payment instruction generated for user's bank
6. User sends fiat funds via chosen payment rail
7. System detects incoming payment and mints stablecoins at booked rate
8. Stablecoins credited to user's wallet

Off-Ramp (Stablecoin → Fiat)

1. User requests exchange rate quote for stablecoin → fiat conversion
2. User books the quoted rate (valid for limited time)
3. User submits off-ramp request with booked rate
4. System validates stablecoin balance and compliance
5. Stablecoins burned from user's wallet at booked rate
6. Fiat payout initiated via chosen payment rail
7. Bank processes transfer to user's account

Core Features

On-Ramp Operations (Fiat → Stablecoin)

On-ramp operations allow users to convert fiat currency to stablecoins through various payment methods.

Supported Payment Methods

- **Bank Transfers**: ACH, SEPA, PIX, UPI, M-Pesa, SWIFT, FAST
- **Card Payments**: Visa, Mastercard (funding only)
- **Local Payment Rails**: Region-specific payment systems

On-Ramp Flow

1. **Initiate Request**: User submits on-ramp request with payment details
2. **Compliance Check**: System validates KYC/AML requirements
3. **Payment Instructions**: System generates payment instructions for user
4. **Fiat Collection**: User sends fiat funds via chosen payment rail
5. **Stablecoin Minting**: System mints equivalent stablecoins
6. **Wallet Credit**: Stablecoins are credited to user's wallet

Off-Ramp Operations (Stablecoin → Fiat)

Off-ramp operations allow users to convert stablecoins back to fiat currency.

Supported Payout Methods

- **Bank Transfers**: Direct bank account transfers
- **Card Credits**: Credit to user's debit/credit card
- **Local Payment Rails**: Region-specific payout systems

Off-Ramp Flow

1. **Initiate Request**: User submits off-ramp request with payout details
2. **Balance Verification**: System verifies stablecoin balance
3. **Compliance Check**: System validates AML requirements
4. **Stablecoin Burn**: System burns stablecoins from user's wallet
5. **Fiat Payout**: System initiates fiat payout via chosen method
6. **Fund Transfer**: Fiat funds are transferred to user's account

Payment Rails

The API abstracts local payment systems to provide a unified interface:

Rail	Region	Description
ACH	US	Automated Clearing House for bank transfers
SEPA	EU	Single Euro Payments Area for European transfers
PIX	Brazil	Instant payment system
UPI	India	Unified Payments Interface
M-Pesa	Africa	Mobile money transfer system
SWIFT	Global	International wire transfers

Rail	Region	Description
FAST	Singapore	Fast and Secure Transfers
Card	Global	Credit/debit card processing

Transaction Management

Transaction Types

- **onramp**: Fiat to stablecoin conversion
- **offramp**: Stablecoin to fiat conversion
- **transfer**: Internal account-to-account transfers
- **fx_settlement**: Cross-currency settlement

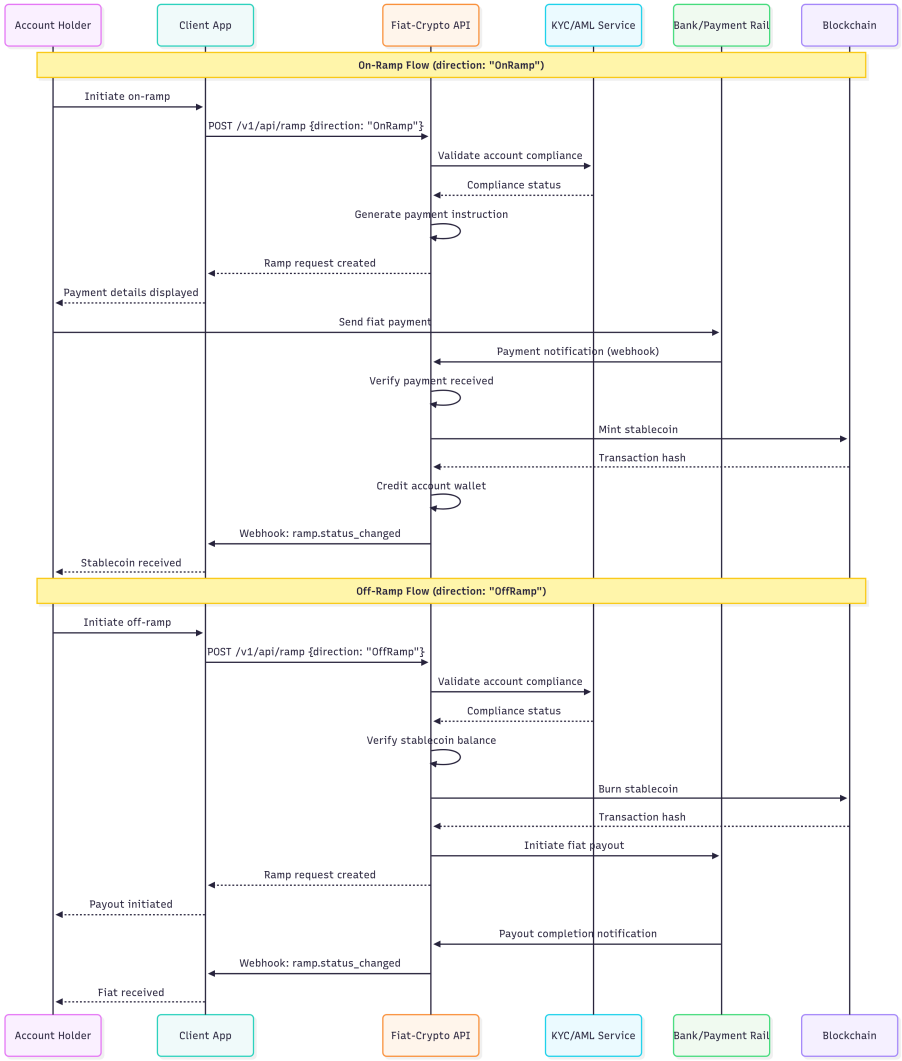
Transaction Statuses

- **initiated**: Acknowledgment received
- **pending**: Processing in progress
- **success**: Completed successfully
- **rejected**: Rejected due to internal validation
- **declined**: Declined by third-party providers
- **in_review**: Under compliance review
- **ambiguous**: Requires human intervention

System Architecture

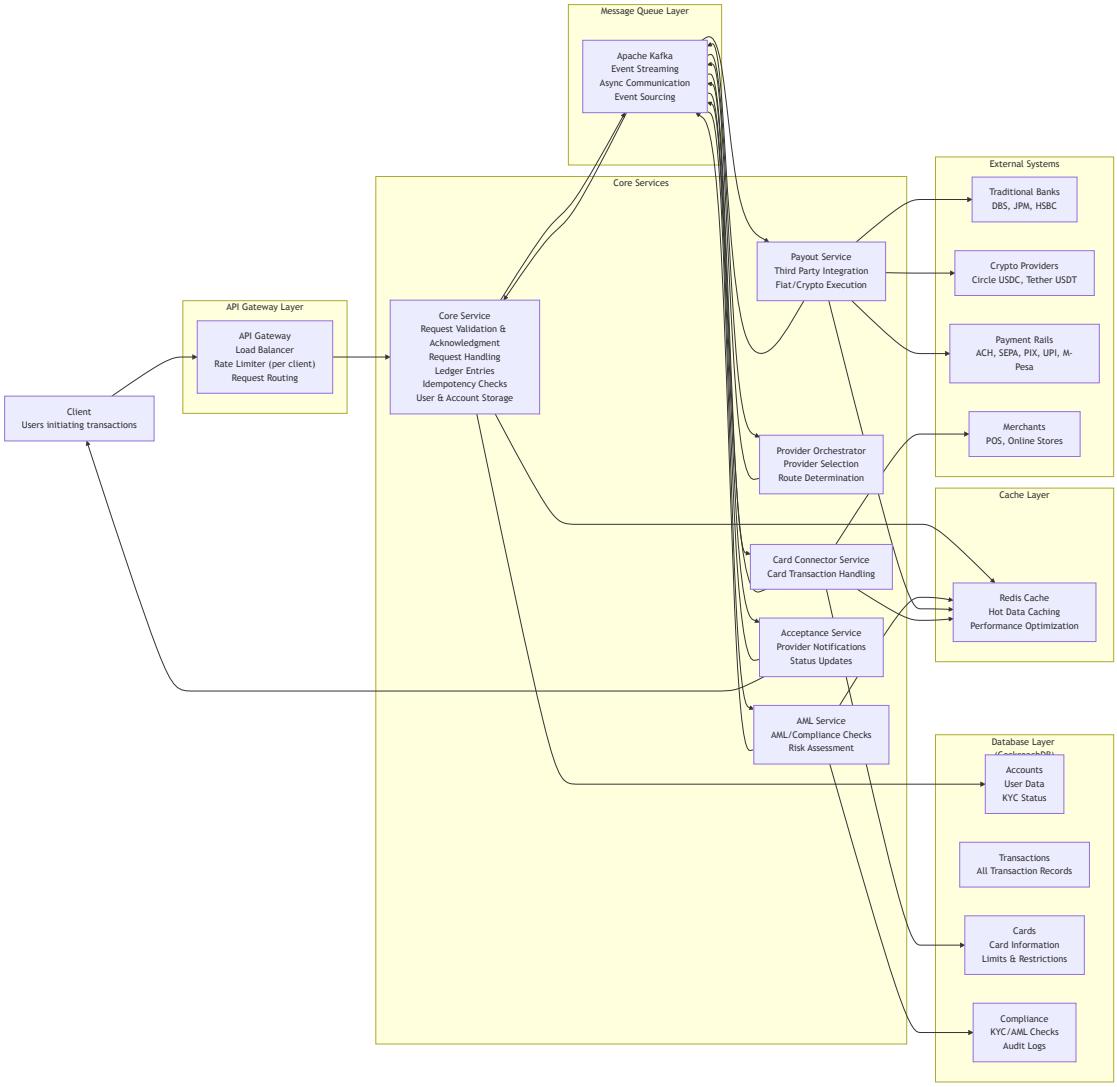
Overall Architecture

The Fiat-Crypto On-Ramp/Off-Ramp API follows a comprehensive architecture that integrates multiple payment rails, crypto providers, and banking systems to provide seamless fiat-to-crypto and crypto-to-fiat operations.



Overall Architecture Diagram

System Architecture Overview 🔗



Core Architecture Flows

The system architecture is designed around several key flows that ensure reliable, compliant, and scalable financial operations:

1. Request Processing Flow

Client → API Gateway → Core Service

- **API Gateway Layer**: Acts as the entry point, providing load balancing, rate limiting per client, and request routing. It ensures that no single client can overwhelm the system and provides a single point of authentication and authorization.
- **Request Validation**: All incoming requests are validated for format, required fields, and business rules before being processed by the Core Service.
- **Idempotency**: The system ensures that duplicate requests with the same idempotency key are handled gracefully, preventing duplicate transactions.

2. Compliance and Risk Assessment Flow

Core Service → AML Service → Kafka → Provider Orchestrator

- **AML Service**: Performs real-time Anti-Money Laundering checks, risk assessment, and compliance validation for every transaction.
- **Risk Scoring**: Each transaction is assigned a risk score based on amount, user history, geographic location, and other factors.
- **Compliance Checks**: Validates KYC status, regulatory requirements, and travel rule compliance before proceeding with transaction execution.
- **Event-Driven Processing**: Compliance results are published to Kafka for asynchronous processing by other services.

3. Provider Selection and Orchestration Flow

Provider Orchestrator → Payout Service → External Providers

- **Provider Orchestrator**: Intelligently selects the optimal provider based on cost, availability, processing time, and success rates.
- **Route Optimization**: Determines the best payment rail and provider combination for each transaction type and geographic region.
- **Load Balancing**: Distributes transactions across multiple providers to ensure high availability and optimal performance.
- **Fallback Mechanisms**: Automatically switches to alternative providers if the primary provider fails or is unavailable.

4. Transaction Execution Flow

Payout Service → External Systems → Acceptance Service → Kafka

- **Fiat Collection**: For on-ramp transactions, initiates payment collection through banks, payment rails, or card networks.
- **Crypto Operations**: For off-ramp transactions, burns stablecoins and initiates fiat payouts through traditional banking systems.
- **Real-time Monitoring**: Continuously monitors transaction status and handles timeouts, failures, and retries.
- **Status Updates**: The Acceptance Service receives notifications from external providers and updates transaction status through Kafka events.

5. Data Persistence and Audit Flow

Core Service → CockroachDB → Audit Logs

- **Immutable Ledger**: All financial transactions are recorded in the TRANSACTIONS table as immutable entries, ensuring complete audit trail and regulatory compliance.

- **Provider Tracking**: External provider interactions are tracked in PROVIDER_TRANSACTIONS table, linked to the main transaction via foreign keys.
- **Session Logging**: Transaction session logs capture all events and state changes for debugging and compliance purposes.
- **Audit Trail**: All system actions are logged in AUDIT_LOGS for regulatory compliance and security monitoring.

6. Caching and Performance Optimization Flow

Core Services → Redis Cache → Hot Data

- **Hot Data Caching**: Frequently accessed data such as user KYC status, account information, and provider configurations are cached in Redis.
- **Performance Optimization**: Reduces database load and improves response times for high-frequency operations.
- **Cache Invalidation**: Ensures data consistency by invalidating cache when underlying data changes.
- **Distributed Caching**: Supports horizontal scaling by allowing multiple service instances to share cached data.

7. Event-Driven Communication Flow

All Services → Kafka → Event Consumers

- **Asynchronous Processing**: All inter-service communication happens through Kafka events, ensuring loose coupling and high availability.
- **Event Sourcing**: Transaction state changes are captured as events, enabling replay and audit capabilities.
- **Exactly Once Semantics**: Event consumers process each event exactly once, preventing duplicate processing and ensuring data consistency in financial transactions.
- **Scalability**: Services can scale independently based on event processing load.
- **Fault Tolerance**: If a service is temporarily unavailable, events are queued and processed when the service recovers.

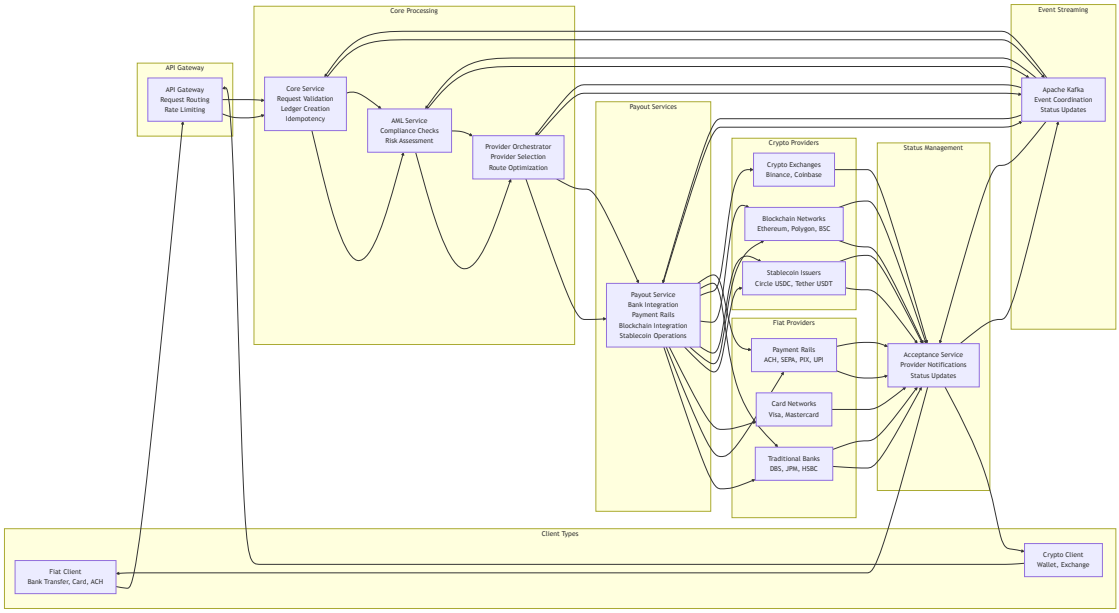
8. Webhook Notification Flow

Acceptance Service → Client Webhooks

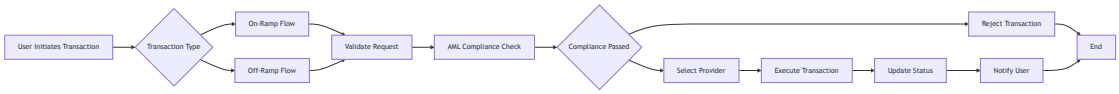
- **Real-time Updates**: Clients receive immediate notifications about transaction status changes, compliance results, and completion events.

- **Reliable Delivery**: Webhook delivery is retried with exponential backoff to ensure clients receive all notifications.
- **Event Filtering**: Clients can subscribe to specific event types based on their integration needs.
- **Security**: Webhook payloads are signed to ensure authenticity and prevent tampering.

Payout Flows



Simple Transaction Flow



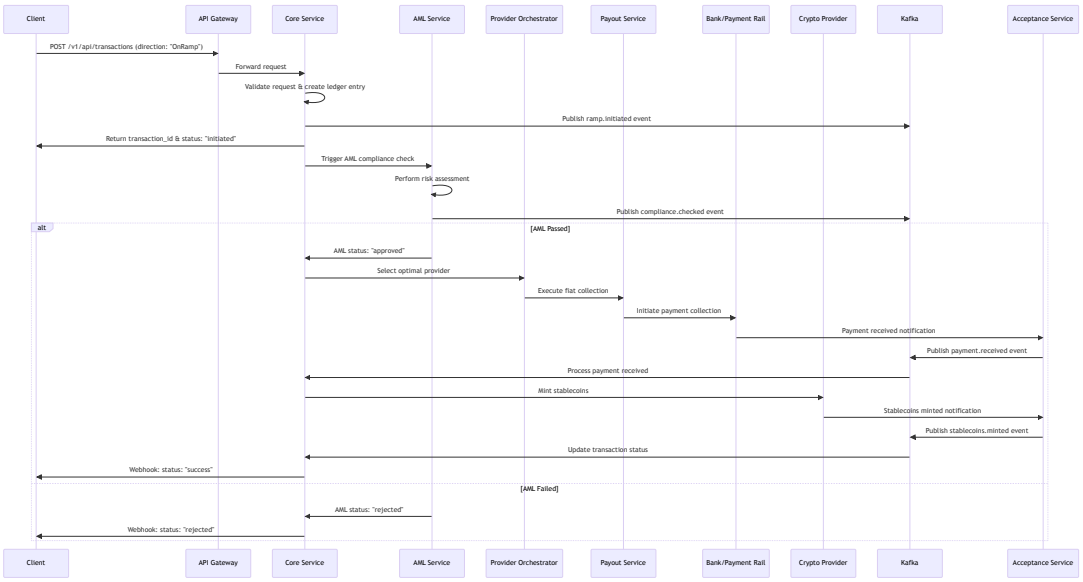
Card Funding/Payout Flow

Key Card Integration Points

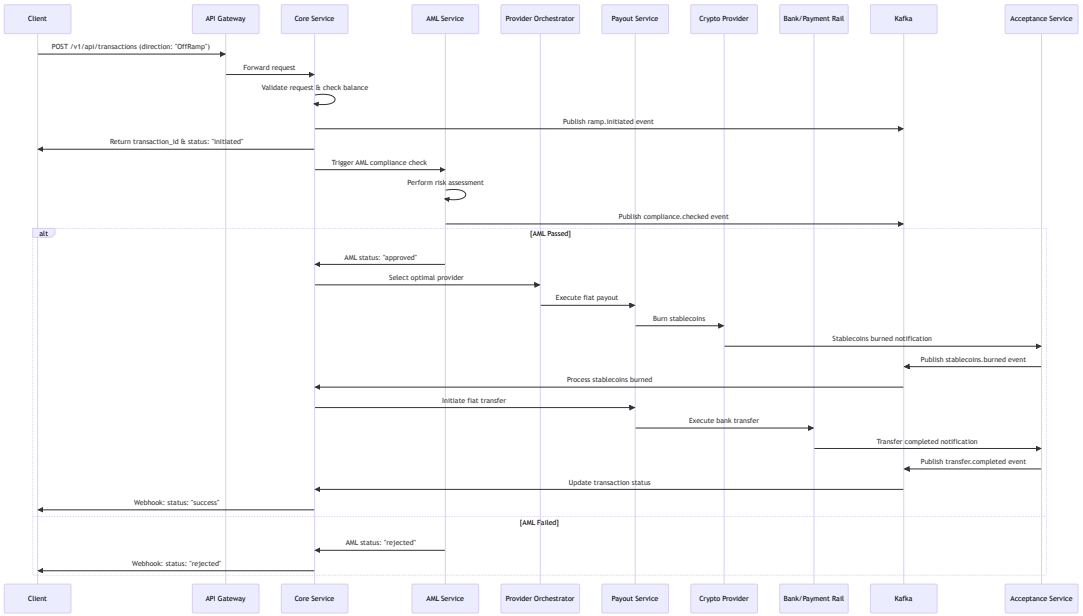
- **Card Funding (On-Ramp)**: User's card is charged, equivalent stablecoins are minted/credited
- **Card Payout (Off-Ramp)**: Stablecoins are burned/debited, card is credited with fiat
- **Card Networks**: Integration with Visa, Mastercard, and other card networks
- **Real-time Processing**: Card transactions require real-time authorization and settlement
- **Fee Structure**: Card transactions typically have higher fees due to network costs

Detailed Sequence Diagrams

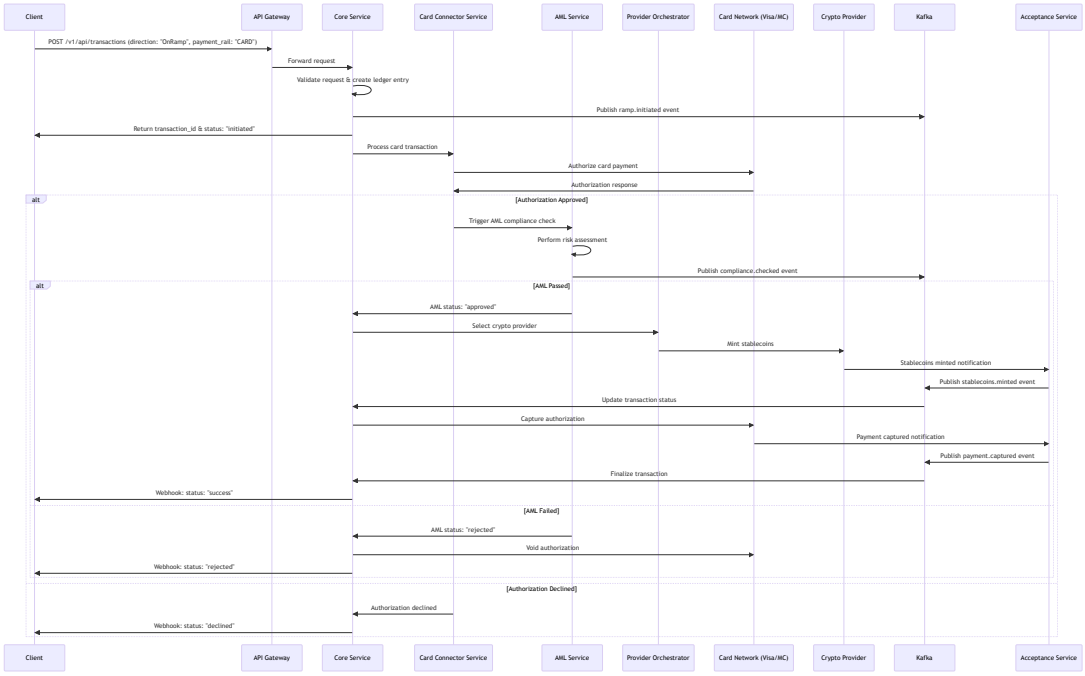
On-Ramp Sequence (Fiat → Stablecoin)



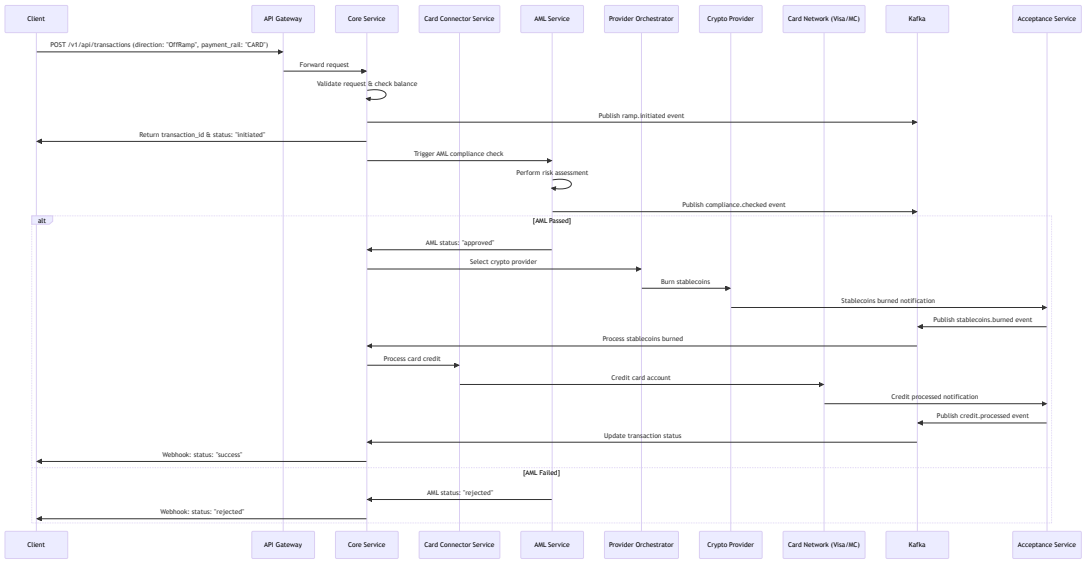
Off-Ramp Sequence (Stablecoin → Fiat)



Card Funding Sequence (On-Ramp)



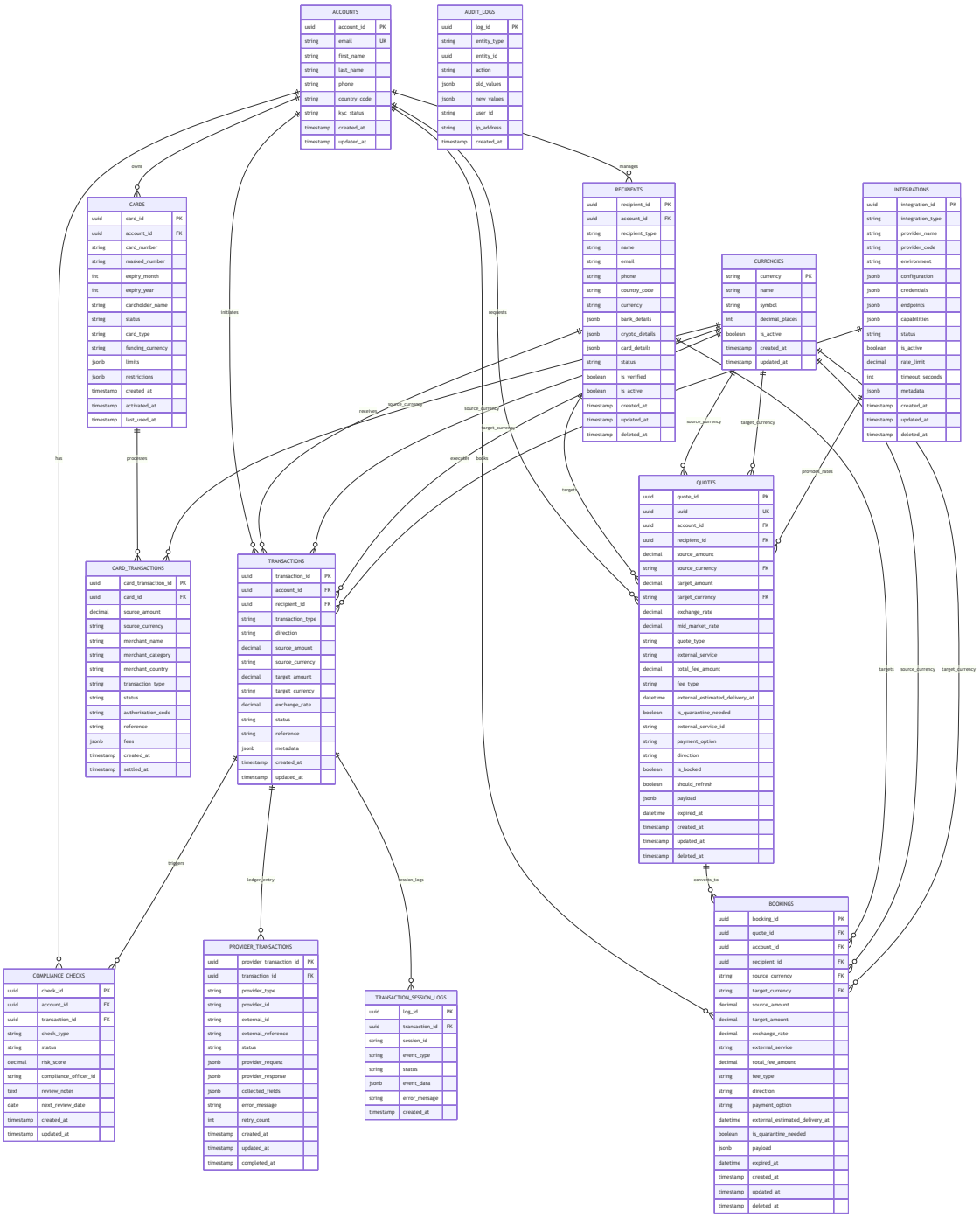
Card Payout Sequence (Off-Ramp)



Database Architecture

The system uses a multi-database architecture optimized for different service requirements:

- **CoreService**: CockroachDB - Primary database providing distributed SQL capabilities with ACID transactions, horizontal scaling, and built-in fault tolerance for all core business operations
- **PayoutService**: DynamoDB - NoSQL database optimized for high-throughput payout operations, storing only essential data (in/out status and idempotency on provider_transaction_id) for maximum performance



Key Database Tables

Multi-Database Architecture:

- **CockroachDB (CoreService)**: Distributed SQL database providing ACID transactions, horizontal scaling, and built-in fault tolerance for all core business operations
- **DynamoDB (PayoutService)**: NoSQL database optimized for high-throughput payout operations with minimal data storage for maximum performance

CockroachDB Tables (CoreService):

- **accounts**: User account information and KYC status

- **recipients**: Recipient management for transactions, quotes, and bookings. Stores bank details, crypto addresses, and verification status for all transaction targets
- **transactions**: Primary immutable ledger of all financial operations (on-ramp, off-ramp, transfers, card transactions). Stores debit/credit entries with complete audit trail
- **quotes**: Comprehensive FX quotes with multi-rate tracking (rate, inversion_rate, mid_market_rate, core_rate), external service integration, detailed fee breakdowns, delivery estimates, and quarantine flags. Supports both inbound and outbound directions with provider-specific configurations
- **bookings**: Rate-locked exchange commitments created from quotes. Provides guaranteed pricing for a specific time period before transaction execution
- **currencies**: Master currency definitions with decimal precision, symbols, and active status
- **provider_transactions**: External provider execution tracking linked to ledger entries via transaction_id FK. Manages third-party integrations and execution details
- **transaction_session_logs**: Session-level event logs created when transactions are initiated
- **payment_rails**: Supported payment rail configurations
- **integrations**: Unified integration settings for both bank APIs and crypto providers. Stores configuration, credentials, endpoints, and operational parameters for all external service integrations
- **compliance_checks**: KYC/AML check results and history
- **webhook_subscriptions**: Webhook endpoint configurations
- **audit_logs**: System audit trail for compliance

DynamoDB Tables (PayoutService):

- **payout_transactions**: High-performance payout tracking with minimal data storage. Stores only essential fields: transaction status (in/out), provider_transaction_id for idempotency, and basic metadata for maximum throughput

Immutable Ledger Design

The system implements an immutable ledger pattern where:

- **TRANSACTIONS table**: Primary immutable ledger - records are never updated or deleted

- **PROVIDER_TRANSACTIONS table**: Tracks external provider executions linked to ledger entries via `transaction_id` FK
- **TRANSACTION_SESSION_LOGS table**: Session-level event logging created when transactions are initiated

Enhanced Table Descriptions

TRANSACTIONS Table (Primary Ledger):

- **Purpose**: Central immutable ledger for all financial operations
- **Transaction Types**: `onramp`, `offramp`, `transfer`, `fx_settlement`, `card_payment`
- **Type Field**: `debit` or `credit` for accounting purposes
- **Key Features**:
  - Immutable records (never updated/deleted)
  - Complete audit trail for all financial activities
  - Supports multi-currency operations (source + target currencies)
  - JSONB metadata for flexible ramp-specific data storage
  - Direct relationship to accounts via sender_id/receiver_id
  - No status field (status tracked in PROVIDER_TRANSACTIONS)

PROVIDER_TRANSACTIONS Table (Execution Tracking):

- **Purpose**: Tracks external provider interactions and execution details
- **Provider Types**: `bank`, `crypto`, `card`, `payment_rail`
- **Key Features**:
  - Links to TRANSACTIONS via `transaction_id` FK
  - **Status tracking**: Execution status for provider operations
  - **External ID tracking**: Third-party transaction ID for reconciliation
  - Stores provider-specific request/response data
  - Tracks retry attempts and error handling
  - Maintains external reference mapping
  - Supports multiple providers per transaction

TRANSACTION_SESSION_LOGS Table (Session Logging):

- **Purpose**: Session-level event logging for transaction lifecycle tracking
- **Creation**: Created when transaction is initiated
- **Key Features**:
  - Links to TRANSACTIONS via `transaction_id` FK
  - **Session tracking**: Groups related events by session_id
  - **Event logging**: Records all transaction lifecycle events

- **Status tracking**: Session-level status changes
- **Error logging**: Captures session-level errors and issues
- **Audit trail**: Complete session history for compliance

Ledger Architecture:

```
TRANSACTIONS (Primary Ledger)
├── transaction_id (PK)
├── sender_id, receiver_id
├── amount, currency
├── transaction_type
└── status, created_at

PROVIDER_TRANSACTIONS (Provider Executions)
├── provider_transaction_id (PK)
├── transaction_id (FK → TRANSACTIONS)
├── provider_type, provider_id
├── external_reference
└── provider_request/response data
```

Benefits:

- **Complete Audit Trail**: Every transaction and provider interaction is permanently recorded
- **Regulatory Compliance**: Meets financial services audit requirements
- **Data Integrity**: No accidental modifications or data loss
- **Reconciliation**: Easy verification against external provider records
- **Debugging**: Full transaction history for troubleshooting
- **Scalability**: Separate tables allow independent scaling and optimization

Transaction Flow:

1. Create immutable transaction record in TRANSACTIONS (primary ledger)
2. Create session log entry in TRANSACTION_SESSION_LOGS with `transaction_id` FK
3. Create provider execution record in PROVIDER_TRANSACTIONS with `transaction_id` FK
4. Log all session events in TRANSACTION_SESSION_LOGS
5. Update status fields in provider tables (never modify core ledger data)

CockroachDB Benefits

- **Distributed Architecture**: Multi-region deployment for global operations
- **ACID Compliance**: Strong consistency for financial transactions

- **Horizontal Scaling**: Automatic sharding and load distribution
- **Built-in Replication**: 3x replication by default for fault tolerance
- **SQL Compatibility**: Standard SQL with PostgreSQL compatibility
- **Time Travel**: Point-in-time recovery capabilities
- **Multi-Active Availability**: No single point of failure

Event-Driven Architecture

The system leverages Apache Kafka as the **central communication hub** for all internal service interactions.

Key Benefits

- **Complete Service Decoupling**: All internal services communicate only through Kafka
- **Async Communication**: Services communicate asynchronously for better resilience
- **Event Sourcing**: All business events are captured for audit trails
- **Scalability**: Kafka's distributed nature supports horizontal scaling
- **Fault Tolerance**: Message persistence ensures no data loss during failures

Kafka Topics & Events

- `transaction.events` : Transaction lifecycle events
- `compliance.events` : AML/KYC check results
- `payout.events` : Payout processing events
- `webhook.events` : Webhook delivery events

API Reference

Core Endpoints

Ramp Transaction Management

- `POST /v1/api/requirements` - Get requirements and available payment rails
- `POST /v1/api/rates` - Get exchange rate quote
- `POST /v1/api/booking` - Book a quoted rate
- `GET /v1/api/booking/{booking_id}` - Get booking status

- `POST /v1/api/transactions` - Initiate transaction (on-ramp/off-ramp with booked rate)
- `GET /v1/api/transactions` - List transactions by account
- `GET /v1/api/transactions/{transaction_id}` - Get transaction status
- `GET /v1/api/transactions/{transaction_id}/status` - Check transaction status
- `POST /v1/api/transactions/optimized` - Optimized transaction request (with booked rate)
- `POST /v1/api/transactions/batch` - Batch transaction operations

Payment Rails

- `GET /v1/api/payment-rails` - Get available payment rails

Integrations

- `GET /v1/api/integrations` - Get all integrations (bank and crypto providers)
- `GET /v1/api/integrations/{integration_id}` - Get specific integration details
- `POST /v1/api/integrations/{integration_id}/test` - Test integration connectivity

Compliance

- `GET /v1/api/compliance/kyc/{account_id}` - Get KYC status
- `GET /v1/api/compliance/aml/{account_id}` - Get AML checks
- `GET /v1/api/compliance/aml/transaction/{transaction_id}` - Get transaction AML checks
- `POST /v1/api/compliance/aml/transaction/{transaction_id}` - Trigger AML check
- `GET /v1/api/compliance/travel-rule/{transaction_id}` - Get travel rule data
- `POST /v1/api/compliance/travel-rule` - Submit travel rule information

Treasury Management

- `GET /v1/api/treasury/liquidity` - Get current liquidity status
- `POST /v1/api/treasury/rebalance` - Trigger manual rebalancing
- `GET /v1/api/treasury/yield` - Get yield optimization opportunities
- `GET /v1/api/treasury/risk` - Get risk assessment metrics
- `POST /v1/api/treasury/emergency` - Emergency liquidity procedures

System

- `GET /v1/api/health` - Health check
- `GET /v1/api/metrics` - System metrics

Complete Ramp Flow with Requirements & Rate Booking

Step 1: Get Rate Quote

```
curl -X POST https://sandbox-api.example.com/v1/api/rates \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "account_id": 12345,
    "source_currency": "USD",
    "source_amount": 1000,
    "target_currency": "USDT",
    "direction": "OnRamp",
    "payment_rail": {
      "type": "ACH",
      "country": "US",
      "currency": "USD"
    },
    "bank_account": {
      "account_number": "1234567890",
      "routing_number": "021000021",
      "account_holder_name": "John Doe",
      "bank_name": "Chase Bank",
      "swift_code": "CHASUS33",
      "is_verified": true
    },
    "compliance_requirements": {
      "kyc_required": true,
      "aml_required": true,
      "travel_rule_required": false,
      "regulatory_jurisdiction": "US"
    },
    "reference": "rate-quote-001"
  }'
```

Response:

```
{
  "id": 987654321,
  "account_id": 12345,
  "source": "USD",
  "source_amount": 1000,
  "target": "USDT",
  "target_amount": 999.50,
```

```
  "rate": 0.9995,
  "direction": "OnRamp",
  "payment_rail": {
    "type": "ACH",
    "country": "US",
    "currency": "USD",
    "estimated_processing_time": "1-3 business days"
  },
  "fees": {
    "total": 0.50,
    "breakdown": {
      "processing": 0.30,
      "network": 0.20
    }
  },
  "compliance_status": {
    "kyc_verified": true,
    "aml_verified": true,
    "travel_rule_verified": true,
    "regulatory_jurisdiction": "US"
  },
  "created_at": "2025-01-15T10:30:00Z",
  "validity": 60,
  "expires_at": "2025-01-15T10:31:00Z"
}
```

Step 2: Book the Rate

```
curl -X POST https://sandbox-api.example.com/v1/api/booking \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "account_id": 12345,
    "source_currency": "USD",
    "source_amount": 1000,
    "target_currency": "USDT",
    "target_amount": 999.50,
    "direction": "OnRamp",
    "payment_rail": {
      "type": "ACH",
      "country": "US",
      "currency": "USD"
    },
    "bank_account": {
      "account_number": "1234567890",
      "routing_number": "021000021",
      "account_holder_name": "John Doe",
      "bank_name": "Chase Bank",
      "swift_code": "CHASUS33",
      "is_verified": true
    },
```

```
    "compliance_requirements": {
      "kyc_required": true,
      "aml_required": true,
      "travel_rule_required": false,
      "regulatory_jurisdiction": "US"
    },
    "reference": "booking-001"
  }'
```

Response:


```
{
  "id": 555001,
  "account_id": 12345,
  "source_currency": "USD",
  "source_amount": 1000,
  "target_currency": "USDT",
  "target_amount": 999.50,
  "direction": "OnRamp",
  "status": "booked",
  "payment_rail": {
    "type": "ACH",
    "country": "US",
    "currency": "USD",
    "estimated_processing_time": "1-3 business days"
  },
  "fees": {
    "total": 0.50,
    "breakdown": {
      "processing": 0.30,
      "network": 0.20
    }
  },
  "compliance_status": {
    "kyc_verified": true,
    "aml_verified": true,
    "travel_rule_verified": true,
    "regulatory_jurisdiction": "US"
  },
  "created_at": "2025-01-15T10:30:00Z",
  "expires_at": "2025-01-15T10:31:00Z",
  "reference": "booking-001"
}
```


Step 3: Complete Ramp with Booked Rate

```
curl -X POST https://sandbox-api.example.com/v1/api/transactions \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "account_id": 12345,
    "direction": "OnRamp",
    "source_currency": "USD",
    "source_amount": 1000.00,
    "target_currency": "USDT",
    "booking_id": 555001,
    "payment_rail": {
      "type": "ACH",
      "country": "US",
      "currency": "USD"
    },
    "bank_account": {
      "account_number": "1234567890",
      "routing_number": "021000021",
      "account_holder_name": "John Doe",
      "bank_name": "Chase Bank",
      "swift_code": "CHASUS33",
      "is_verified": true
    },
    "compliance_requirements": {
      "kyc_required": true,
      "aml_required": true,
      "travel_rule_required": false,
      "regulatory_jurisdiction": "US"
    },
    "transaction_metadata": {
      "customer_reference": "CUST-001",
      "order_id": "ORDER-12345",
      "invoice_number": "INV-67890",
      "description": "On-ramp transaction for stablecoin purchase"
    },
    "reference": "onramp-001"
  }'
```

Data Models

Ramp Request (with Booked Rate)

```
{
  "account_id": 12345,
  "direction": "OnRamp",
  "source_currency": "USD",
  "source_amount": 1000.00,
  "target_currency": "USDT",
  "booking_id": 555001,
  "payment_rail": {
    "type": "ACH",
    "country": "US",
    "currency": "USD"
  },
  "bank_account": {
```

```
    "account_number": "1234567890",
    "routing_number": "021000021"
  },
  "reference": "onramp-001"
}
```

Ramp Response (Detailed Transaction Schema)

```
{
  "id": 1001,
  "account_id": 12345,
  "direction": "OnRamp",
  "status": "pending",
  "source_currency": "USD",
  "source_amount": 1000.00,
  "target_currency": "USDT",
  "target_amount": 999.50,
  "exchange_rate": 0.9995,
  "booking_id": 555001,
  "payment_rail": {
    "type": "ACH",
    "country": "US",
    "currency": "USD",
    "estimated_processing_time": "1-3 business days"
  },
  "bank_account": {
    "account_number": "****7890",
    "routing_number": "021000021",
    "account_holder_name": "John Doe",
    "bank_name": "Chase Bank",
    "swift_code": "CHASUS33",
    "is_verified": true
  },
  "fees": {
    "total": 0.50,
    "breakdown": {
      "processing": 0.30,
      "network": 0.20
    }
  },
  "compliance_status": {
    "kyc_verified": true,
    "aml_verified": true,
    "travel_rule_verified": true,
    "regulatory_jurisdiction": "US"
  },
  "transaction_metadata": {
    "customer_reference": "CUST-001",
    "order_id": "ORDER-12345",
    "invoice_number": "INV-67890",
    "description": "On-ramp transaction for stablecoin purchase"
```

```
  },
  "transaction_hash": null,
  "created_at": "2025-01-15T10:30:00Z",
  "updated_at": "2025-01-15T10:30:00Z",
  "expires_at": "2025-01-15T22:30:00Z",
  "reference": "onramp-001"
}
```

Error Handling

HTTP Status Codes

Code	Description
200	Success
201	Created
400	Bad Request - Invalid parameters
401	Unauthorized - Invalid or missing token
403	Forbidden - Insufficient permissions
404	Not Found - Resource doesn't exist
409	Conflict - Duplicate request (idempotency)
422	Unprocessable Entity - Validation error
429	Too Many Requests - Rate limit exceeded
500	Internal Server Error
503	Service Unavailable

Error Response Format

```
{
  "error": {
    "code": "INVALID_AMOUNT",
    "message": "Amount must be greater than 0.01",
    "details": {
      "field": "source_amount",
      "value": 0.005,
      "constraint": "minimum: 0.01"
    },
    "request_id": "req_123456789",
    "timestamp": "2025-01-15T10:30:00Z"
```

```
  }
}
```

Rate Limiting

Limits

- **Standard**: 1000 requests per hour
- **Burst**: 100 requests per minute
- **Batch Operations**: 10 requests per minute

Headers

```
X-RateLimit-Limit: 1000
X-RateLimit-Remaining: 999
X-RateLimit-Reset: 1642248000
```

Integration & Development

Quick Start Guide

1. Get API Credentials

Contact your account manager to obtain API credentials and configure webhook endpoints.

2. Test with Sandbox

Use the sandbox environment to test your integration: - **Sandbox URL**: `https://sandbox-api.example.com` - **Test API Key**: Provided by your account manager

3. Make Your First API Call

```
curl -X GET https://sandbox-api.example.com/v1/api/health \
  -H "Authorization: Bearer YOUR_SANDBOX_TOKEN"
```

4. Initiate Your First On-Ramp

```
curl -X POST https://sandbox-api.example.com/v1/api/transactions \
  -H "Authorization: Bearer YOUR_SANDBOX_TOKEN" \
  -H "Content-Type: application/json" \
```

```
  -H "Idempotency-Key: unique-request-id" \
  -d '{
    "account_id": 12345,
    "direction": "OnRamp",
    "source_currency": "USD",
    "source_amount": 100.00,
    "target_currency": "USDT",
    "booking_id": 555001,
    "payment_rail": {
      "type": "ACH",
      "country": "US",
      "currency": "USD"
    },
    "bank_account": {
      "account_number": "1234567890",
      "routing_number": "021000021",
      "account_holder_name": "John Doe",
      "bank_name": "Chase Bank",
      "swift_code": "CHASUS33",
      "is_verified": true
    },
    "compliance_requirements": {
      "kyc_required": true,
      "aml_required": true,
      "travel_rule_required": false,
      "regulatory_jurisdiction": "US"
    },
    "transaction_metadata": {
      "customer_reference": "CUST-001",
      "order_id": "ORDER-12345",
      "description": "Test on-ramp transaction for stablecoin purcl
    },
    "reference": "test-onramp-001"
  }'
```

Response Example


```
{
  "id": 1001,
  "account_id": 12345,
  "direction": "OnRamp",
  "status": "pending",
  "source_currency": "USD",
  "source_amount": 100.00,
  "target_currency": "USDT",
  "target_amount": 99.95,
  "exchange_rate": 0.9995,
  "booking_id": 555001,
  "payment_rail": {
    "type": "ACH",
    "country": "US",
```

```
    "currency": "USD",
    "estimated_processing_time": "1-3 business days"
  },
  "fees": {
    "total": 0.05,
    "breakdown": {
      "processing": 0.03,
      "network": 0.02
    }
  },
  "compliance_status": {
    "kyc_verified": true,
    "aml_verified": true,
    "travel_rule_verified": true,
    "regulatory_jurisdiction": "US"
  },
  "transaction_metadata": {
    "customer_reference": "CUST-001",
    "order_id": "ORDER-12345",
    "description": "Test on-ramp transaction for stablecoin purchas
  },
  "transaction_hash": null,
  "created_at": "2025-01-15T10:30:00Z",
  "updated_at": "2025-01-15T10:30:00Z",
  "expires_at": "2025-01-15T22:30:00Z",
  "reference": "test-onramp-001"
}
```


Authentication

All API requests require authentication using Bearer tokens.

Headers

```
Authorization: Bearer <your_jwt_token>
Content-Type: application/json
```

Getting API Keys

Contact your account manager to obtain API credentials and configure webhook endpoints.

Integration Examples

On-Ramp Example (USD → USDT)

```
curl -X POST https://api.example.com/v1/api/transactions \
  -H "Authorization: Bearer YOUR_TOKEN" \
```

```
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: unique-request-id" \
  -d '{
    "account_id": 12345,
    "direction": "OnRamp",
    "source_currency": "USD",
    "source_amount": 1000.00,
    "target_currency": "USDT",
    "booking_id": 555001,
    "payment_rail": {
      "type": "ACH",
      "country": "US",
      "currency": "USD"
    },
    "bank_account": {
      "account_number": "1234567890",
      "routing_number": "021000021",
      "account_holder_name": "John Doe",
      "bank_name": "Chase Bank",
      "swift_code": "CHASUS33",
      "is_verified": true
    },
    "compliance_requirements": {
      "kyc_required": true,
      "aml_required": true,
      "travel_rule_required": false,
      "regulatory_jurisdiction": "US"
    },
    "transaction_metadata": {
      "customer_reference": "CUST-001",
      "order_id": "ORDER-12345",
      "description": "On-ramp transaction for stablecoin purchase"
    },
    "reference": "onramp-001"
  }'
```

Off-Ramp Example (USDT → USD)

```
curl -X POST https://api.example.com/v1/api/transactions \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: unique-request-id" \
  -d '{
    "account_id": 12345,
    "direction": "OffRamp",
    "source_currency": "USDT",
    "source_amount": 500.00,
    "target_currency": "USD",
    "booking_id": 555002,
    "payment_rail": {
      "type": "ACH",
```

```
      "country": "US",
      "currency": "USD"
    },
    "bank_account": {
      "account_number": "1234567890",
      "routing_number": "021000021",
      "account_holder_name": "John Doe",
      "bank_name": "Chase Bank",
      "swift_code": "CHASUS33",
      "is_verified": true
    },
    "compliance_requirements": {
      "kyc_required": true,
      "aml_required": true,
      "travel_rule_required": false,
      "regulatory_jurisdiction": "US"
    },
    "transaction_metadata": {
      "customer_reference": "CUST-001",
      "order_id": "ORDER-12346",
      "description": "Off-ramp transaction for fiat withdrawal"
    },
    "reference": "offramp-001"
  }'
```

Webhooks

Supported Events

- `rate.expired` - Exchange rate quote expired
- `booking.created` - Rate booking created
- `booking.expired` - Rate booking expired
- `transaction.status_changed` - Transaction status update
- `compliance.review_required` - Compliance review needed
- `webhook.test` - Test webhook

Webhook Payload Example

```
{
  "event": "transaction.status_changed",
  "data": {
    "id": 1001,
    "account_id": 12345,
    "direction": "OnRamp",
    "status": "success",
    "previous_status": "pending"
  },
  "timestamp": "2025-01-15T10:30:00Z",
```

```
  "webhook_id": "wh_123456789"
}
```

Additional Features

Advanced Compliance

KYC Levels

- **Basic**: Email verification and basic identity
- **Enhanced**: Government ID verification
- **Institutional**: Business verification and enhanced due diligence

AML Monitoring

- Real-time transaction monitoring
- Risk scoring and assessment
- Compliance officer review workflows
- Audit trail maintenance

Travel Rule Compliance

- **VASP Identification**: Automatic identification of Virtual Asset Service Providers
- **Transaction Reporting**: Mandatory reporting for transactions above regulatory thresholds
- **Originator Information**: Collection and transmission of originator details
- **Beneficiary Information**: Collection and transmission of beneficiary details
- **Regulatory Thresholds**: Configurable limits per jurisdiction (e.g., $1,000 USD equivalent)
- **Cross-Border Monitoring**: Enhanced monitoring for international transfers
- **Data Retention**: Secure storage of travel rule data for regulatory compliance

Travel Rule Data Model

```
{
  "travel_rule_data": {
    "transaction_id": "tx_123456789",
```

```
    "originator": {
      "name": "John Doe",
      "account_number": "1234567890",
      "address": "123 Main St, New York, NY 10001",
      "date_of_birth": "1990-01-01",
      "national_id": "SSN123456789"
    },
    "beneficiary": {
      "name": "Jane Smith",
      "account_number": "0987654321",
      "address": "456 Oak Ave, London, UK",
      "date_of_birth": "1985-05-15",
      "national_id": "UK123456789"
    },
    "transaction_details": {
      "amount": 5000.00,
      "currency": "USDT",
      "purpose": "Business payment",
      "reference": "Invoice #12345"
    },
    "compliance_status": "approved",
    "regulatory_jurisdiction": "US",
    "threshold_exceeded": true,
    "reporting_required": true
  }
}
```


Key Database Tables - Integration Settings

Bank Integration (Unified Schema)


```
{
  "integration_id": "dbs_bank_sg",
  "provider_name": "DBS Bank Singapore",
  "type": "bank",
  "metadata": {
    "country": "SG",
    "bank_code": "7171",
    "supported_rails": ["SEPA", "SWIFT", "FAST"],
    "supported_currencies": ["USD", "SGD", "EUR"],
    "payout_limits": {
      "min_amount": 1.00,
      "max_amount": 1000000.00,
      "daily_limit": 5000000.00,
      "monthly_limit": 50000000.00
    }
  },
  "configurations": {
    "api_endpoint": "https://api.dbs.com",
    "api_version": "v2",
    "authentication_type": "oauth2",
```

```
    "webhook_url": "https://api.example.com/webhooks/dbs",
    "timeout_seconds": 30,
    "retry_attempts": 3
  },
  "capabilities": ["transfer", "balance_check", "transaction_statu
  "is_active": true,
  "rate_limits": {
    "requests_per_minute": 100,
    "requests_per_hour": 1000,
    "requests_per_day": 10000
  },
  "created_at": "2025-01-01T00:00:00Z",
  "updated_at": "2025-01-15T10:00:00Z"
}
```


Crypto Provider Integration (Unified Schema)


```
{
  "integration_id": "circle_usdc",
  "provider_name": "Circle USDC",
  "type": "crypto",
  "metadata": {
    "provider_type": "stablecoin_issuer",
    "supported_networks": [
      {
        "network": "ethereum",
        "chain_id": 1,
        "rpc_url": "https://mainnet.infura.io/v3/...",
        "explorer_url": "https://etherscan.io"
      }
    ],
    "supported_tokens": [
      {
        "symbol": "USDC",
        "contract_address": "0xA0b86a33E6441c8C06DDD4C4C4C4C4C4C4C4
        "decimals": 6,
        "network": "ethereum"
      }
    ],
    "fees": {
      "mint_fee": 0.0,
      "burn_fee": 0.0,
      "transfer_fee": 0.0,
      "gas_fee_multiplier": 1.0
    }
  },
  "configurations": {
    "api_endpoint": "https://api.circle.com",
    "api_version": "v1",
    "authentication_type": "api_key",
    "webhook_url": "https://api.example.com/webhooks/circle",
```

```
    "timeout_seconds": 30,
    "retry_attempts": 3,
    "gas_limit": 21000,
    "gas_price": "20"
  },
  "capabilities": ["mint", "burn", "transfer", "balance_check", "t
  "is_active": true,
  "rate_limits": {
    "requests_per_minute": 100,
    "requests_per_hour": 1000,
    "requests_per_day": 10000
  },
  "created_at": "2025-01-01T00:00:00Z",
  "updated_at": "2025-01-15T10:00:00Z"
}
```


Additional Crypto Providers (Unified Schema)


```
{
  "integration_id": "tether_usdt",
  "provider_name": "Tether USDT",
  "type": "crypto",
  "metadata": {
    "provider_type": "stablecoin_issuer",
    "supported_networks": [
      {
        "network": "ethereum",
        "chain_id": 1,
        "rpc_url": "https://mainnet.infura.io/v3/...",
        "explorer_url": "https://etherscan.io"
      }
    ],
    "supported_tokens": [
      {
        "symbol": "USDT",
        "contract_address": "0xdAC17F958D2ee523a2206206994597C13D8
        "decimals": 6,
        "network": "ethereum"
      }
    ],
    "fees": {
      "mint_fee": 0.0,
      "burn_fee": 0.0,
      "transfer_fee": 0.0,
      "gas_fee_multiplier": 1.0
    }
  },
  "configurations": {
    "api_endpoint": "https://api.tether.to",
    "api_version": "v1",
    "authentication_type": "api_key",
```

```
    "webhook_url": "https://api.example.com/webhooks/tether",
    "timeout_seconds": 30,
    "retry_attempts": 3,
    "gas_limit": 21000,
    "gas_price": "20"
  },
  "capabilities": ["mint", "burn", "transfer", "balance_check", "t
  "is_active": true,
  "rate_limits": {
    "requests_per_minute": 100,
    "requests_per_hour": 1000,
    "requests_per_day": 10000
  },
  "created_at": "2025-01-01T00:00:00Z",
  "updated_at": "2025-01-15T10:00:00Z"
}
```


Security & Custody Architecture

Multi-Party Computation (MPC) Custody

- **Threshold Signatures**: Distributed key generation with configurable thresholds (e.g., 2-of-3, 3-of-5)
- **Hardware Security Modules**: Integration with HSM providers for key protection
- **Geographic Distribution**: Key shares distributed across multiple regions for resilience
- **Zero-Knowledge Proofs**: Privacy-preserving transaction validation
- **Key Rotation**: Automated key rotation schedules for enhanced security

Multisig Wallet Architecture

- **Multi-Signature Wallets**: Configurable signature requirements for different transaction types
- **Time-Locked Transactions**: Delayed execution for high-value transactions
- **Emergency Procedures**: Break-glass procedures for emergency access
- **Audit Trails**: Complete logging of all signature events and key operations

Custody Service Integration


```
{
  "custody_provider": {
    "provider_id": "fireblocks_mpc",
    "name": "Fireblocks MPC",
    "custody_type": "MPC",
```

```
    "api_endpoint": "https://api.fireblocks.io",
    "supported_networks": ["ethereum", "polygon", "binance_smart_c
    "security_features": [
      "threshold_signatures",
      "hardware_security_modules",
      "geographic_distribution",
      "key_rotation"
    ],
    "compliance_features": [
      "travel_rule_support",
      "regulatory_reporting",
      "audit_trails",
      "risk_monitoring"
    ]
  }
}
```

Treasury Management & Liquidity Optimization

Treasury Management Service

- **Liquidity Pool Management**: Automated rebalancing across multiple stablecoin providers
- **Yield Optimization**: Integration with DeFi protocols for yield generation
- **Risk Management**: Real-time monitoring of counterparty and market risks
- **Cash Flow Forecasting**: Predictive analytics for liquidity requirements
- **Multi-Currency Support**: Management across USD, EUR, GBP, and other major currencies

Liquidity Management Features

- **Dynamic Rebalancing**: Automatic adjustment of stablecoin allocations based on demand
- **Provider Diversification**: Risk distribution across multiple stablecoin issuers
- **Emergency Liquidity**: Reserve pools for high-demand scenarios
- **Cross-Chain Liquidity**: Seamless movement of liquidity across blockchain networks
- **Real-Time Monitoring**: Live dashboards for treasury operations

Treasury Management API

```
{
  "treasury_management": {
```

```
    "liquidity_pools": {
      "usdc_pool": {
        "total_balance": 10000000.00,
        "available_balance": 8000000.00,
        "reserved_balance": 2000000.00,
        "yield_rate": 0.045,
        "provider_distribution": {
          "circle": 0.6,
          "coinbase": 0.3,
          "binance": 0.1
        }
      },
      "usdt_pool": {
        "total_balance": 5000000.00,
        "available_balance": 4000000.00,
        "reserved_balance": 1000000.00,
        "yield_rate": 0.038,
        "provider_distribution": {
          "tether": 0.7,
          "binance": 0.3
        }
      }
    },
    "risk_metrics": {
      "total_exposure": 15000000.00,
      "counterparty_risk": 0.15,
      "market_risk": 0.08,
      "liquidity_ratio": 0.85,
      "stress_test_score": 0.92
    },
    "rebalancing_strategy": {
      "auto_rebalance": true,
      "rebalance_threshold": 0.1,
      "max_single_provider": 0.4,
      "emergency_reserve_ratio": 0.2
    }
  }
}
```


Treasury Operations Endpoints

- `GET /v1/api/treasury/liquidity` - Get current liquidity status
- `POST /v1/api/treasury/rebalance` - Trigger manual rebalancing
- `GET /v1/api/treasury/yield` - Get yield optimization opportunities
- `GET /v1/api/treasury/risk` - Get risk assessment metrics
- `POST /v1/api/treasury/emergency` - Emergency liquidity procedures

Best Practices

- **Idempotency**: Always include idempotency keys for POST requests
- **Error Handling**: Implement proper error handling and retry logic
- **Webhooks**: Use webhooks for real-time status updates
- **Rate Limiting**: Respect rate limits and implement backoff strategies
- **Security**: Keep API keys secure and rotate regularly
