

# Zus Trading Agent API Specification

**Version:** 1.0.0

**Last Updated:** November 2024

**Document Type:** Enterprise API Reference

---

## Table of Contents

---

1. [Executive Summary](#)
  2. [Architecture Overview](#)
  3. [Bot Tier System](#)
  4. [API Reference](#)
  5. [Data Models](#)
  6. [Integration Guides](#)
  7. [Security & Best Practices](#)
  8. [Appendices](#)
- 

## Executive Summary

---

### What is the Zus Trading Agent Engine

The Zus Trading Agent Engine is a production-ready trading simulation platform that provides:

- **AI-Powered Trading Logs:** Real-time trading activity generation using Workers AI (Llama-3)
- **Multi-Tier Bot System:** Four distinct bot tiers (Probot, Chainpulse, Titan, Omega) with varying ROI rates

- **User Balance Management:** REST API for managing user balances and tracking projected profits
- **Enterprise Integration:** Designed for seamless integration with existing financial platforms

Core Capabilities

Capability	Description
Trading Simulation	Autonomous trading activity with realistic P&L generation
Multi-User Support	Individual balance tracking per user via KV storage
Tier-Based ROI	Configurable profit projections based on investment tier
Real-Time Data	WebSocket integration with Binance for live market data
AI Narratives	Automated trading log generation using LLM technology

Use Cases

1. **Fintech Applications:** Add trading simulation features to portfolio management tools
2. **Trading Education:** Build educational platforms demonstrating market dynamics
3. **Investment Simulators:** Create paper trading environments with realistic outcomes
4. **White-Label Solutions:** Deploy branded trading simulation platforms

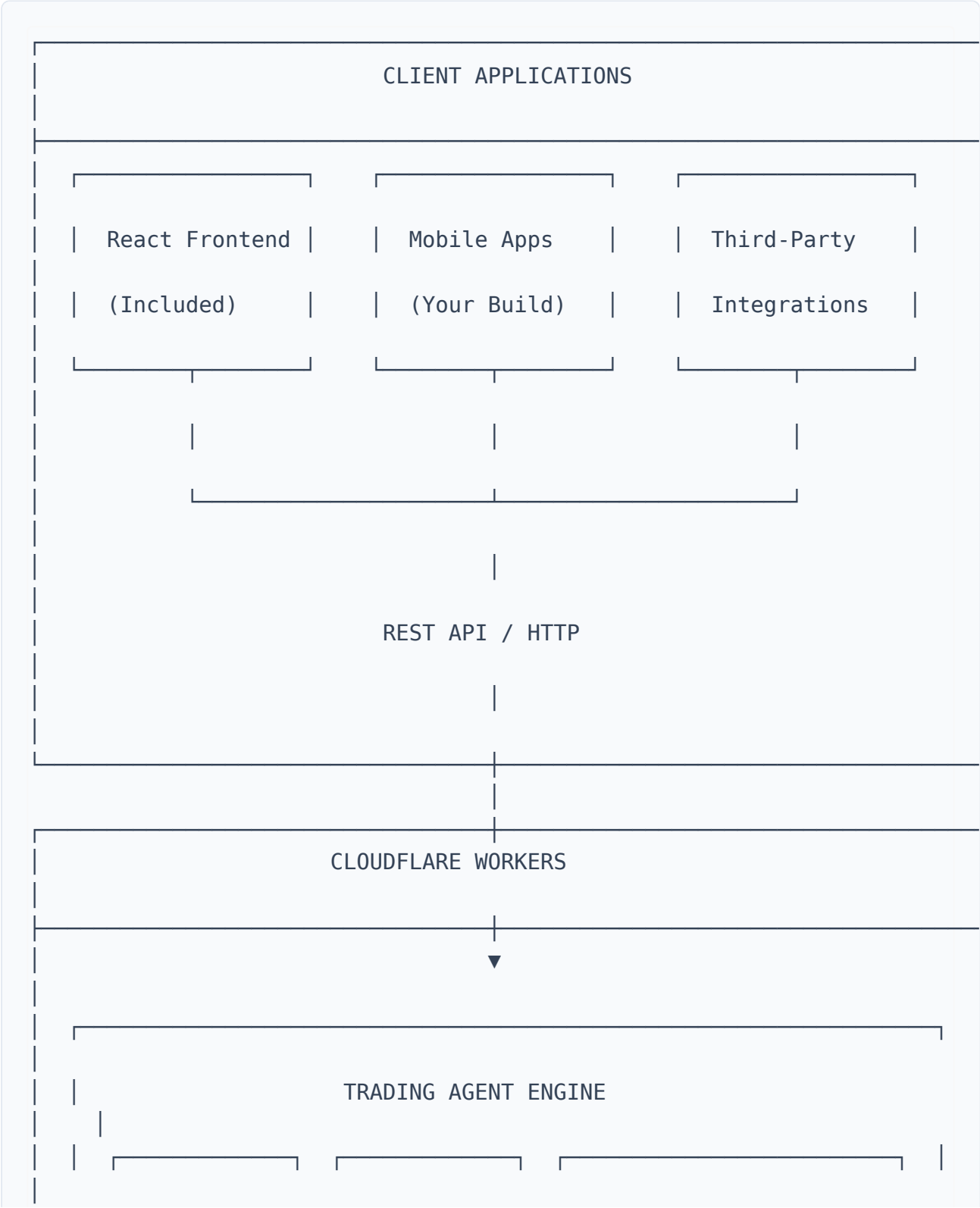
Integration Options

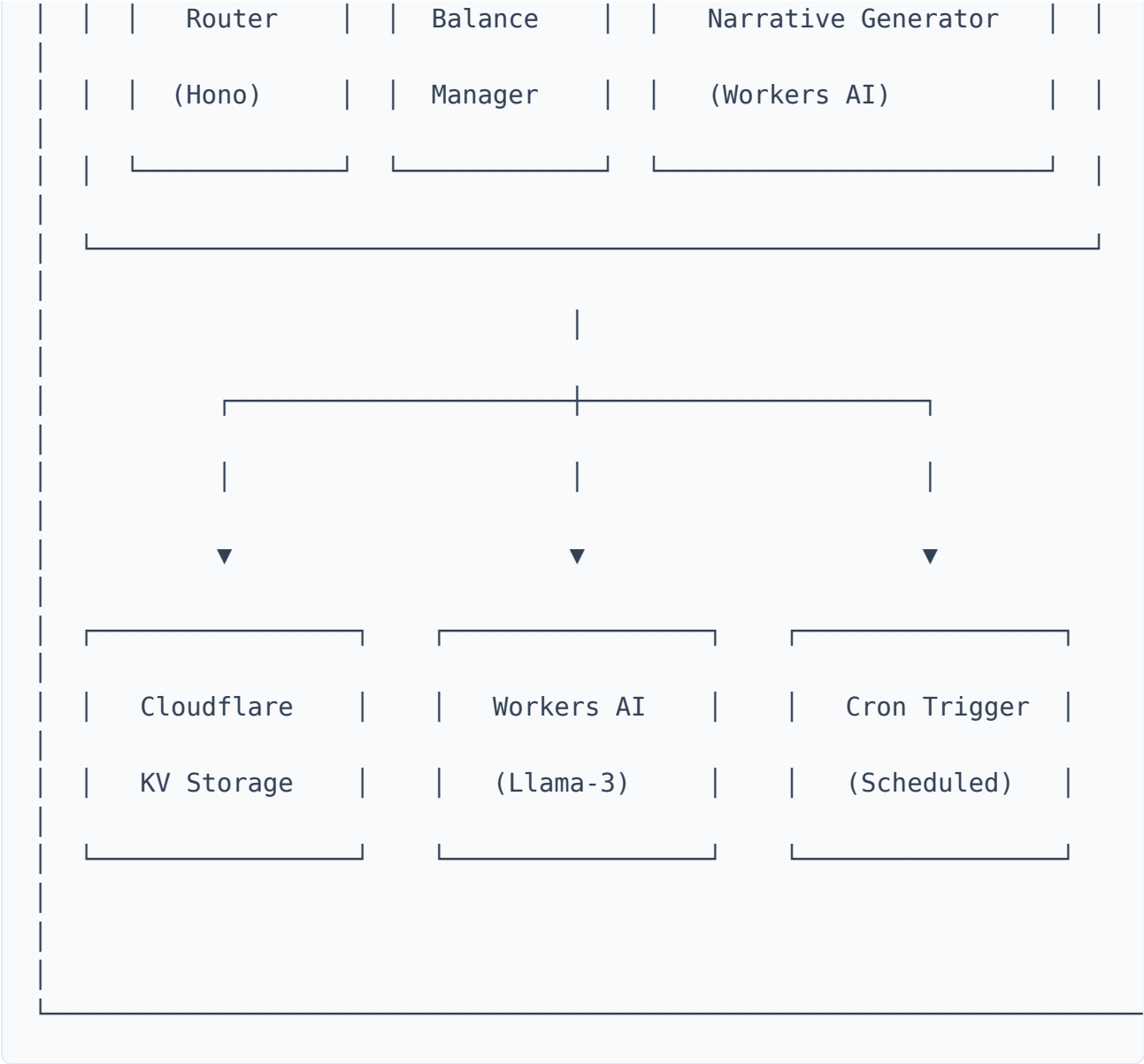
The Zus Trading Agent offers two integration paths:

Option	Description	Best For
API-Only Integration	Use REST endpoints to integrate trading features into your existing platform	Teams with existing frontends who need backend trading logic
Full-Stack Fork	Clone the entire repository including React frontend and Cloudflare Workers backend	Teams building new trading platforms from scratch

# Architecture Overview

## System Architecture





## Technology Stack

Layer	Technology	Purpose
Frontend	React 19, TypeScript, Vite	User interface
Styling	TailwindCSS 4, Framer Motion	UI components and animations
State Management	Zustand	Client-side state
Charts	Lightweight Charts	Trading visualizations
Backend	Cloudflare Workers	Serverless API
Storage	Cloudflare KV	User balance persistence
AI	Workers AI (Llama-3-8b-instruct)	Trading log generation
Market Data	Binance WebSocket API	Real-time price feeds

## Infrastructure Requirements

### Cloudflare Services Required

Service	Purpose	Configuration
Workers	API hosting	<code>compatibility_date: 2024-01-01</code>
KV Namespace	Data storage	Binding: <code>TRADING_CACHE</code>
Workers AI	Log generation	Binding: <code>AI</code>
Pages	Frontend hosting	Project: <code>orion-dashboard</code>

## Environment Configuration

```
# wrangler.toml
name = "trading-agent-engine"
main = "src/index.ts"
compatibility_date = "2024-01-01"
compatibility_flags = ["nodejs_compat"]

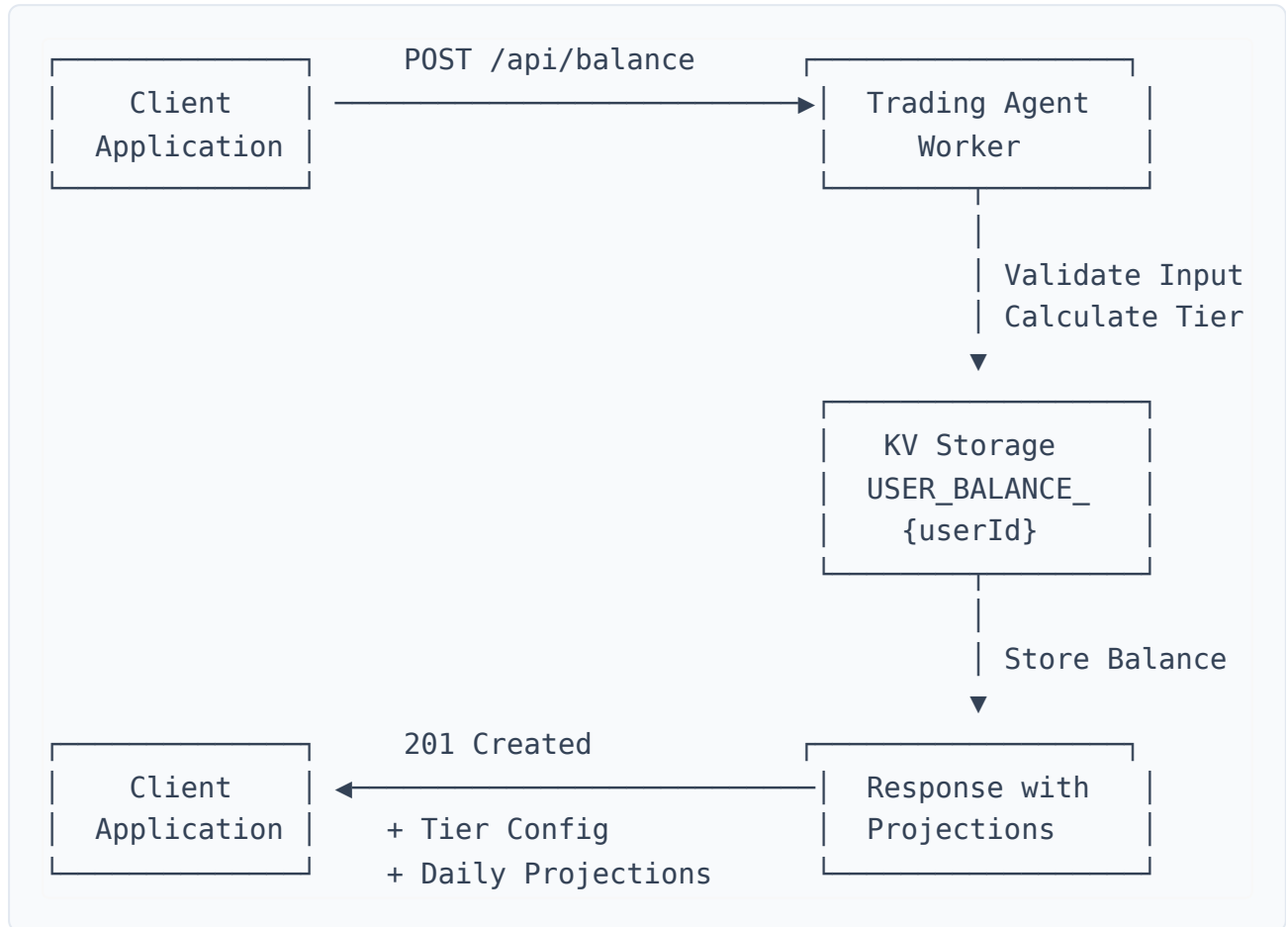
[[kv_namespaces]]
binding = "TRADING_CACHE"
id = "your-kv-namespace-id"
preview_id = "your-preview-kv-namespace-id"

[ai]
binding = "AI"

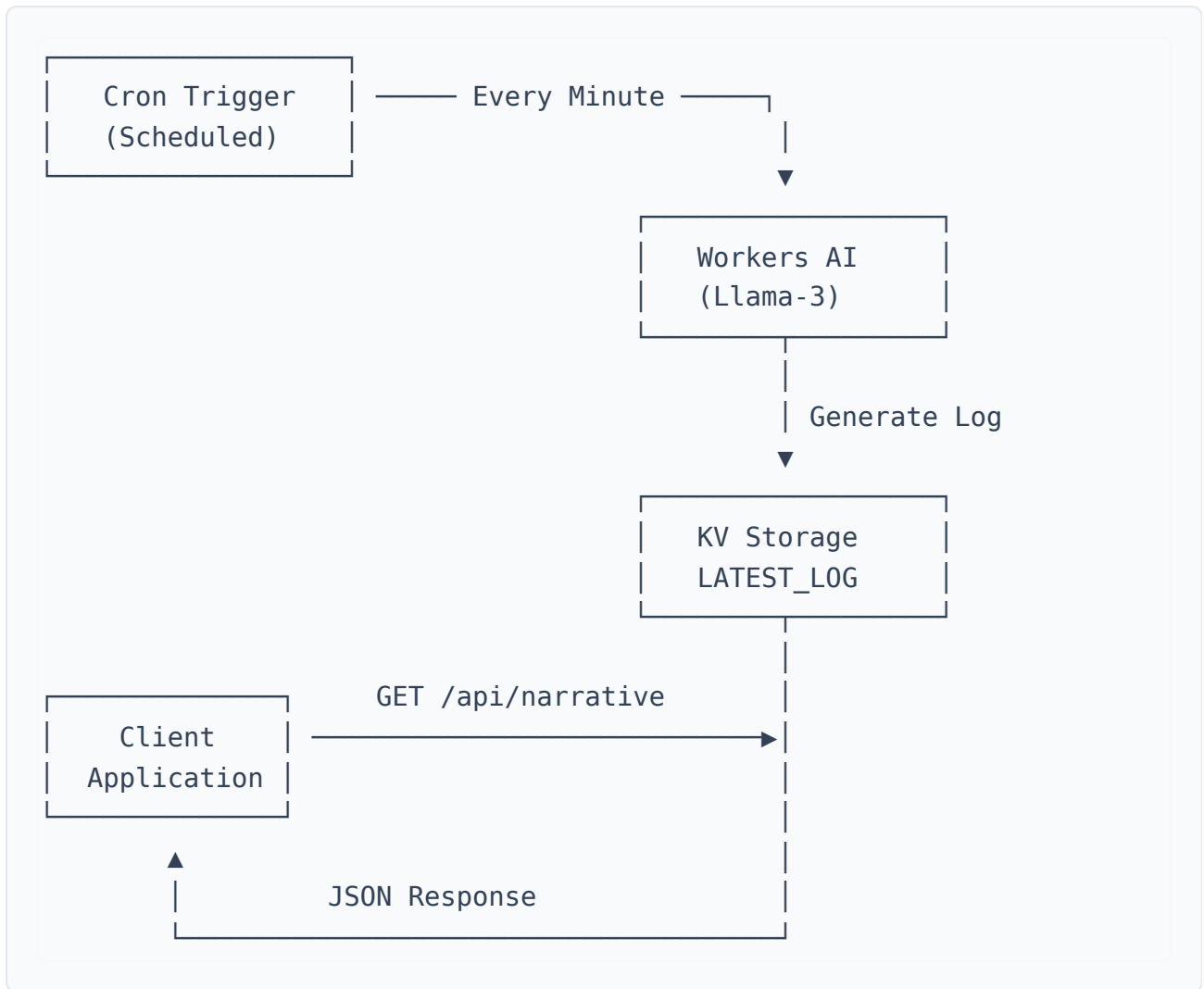
[triggers]
crons = ["* * * * *"]
```

## Data Flow

### Balance Update Flow



## Trading Narrative Flow



## Bot Tier System

### Overview

The Zus Trading Agent implements a four-tier bot system, each with distinct ROI characteristics, minimum stake requirements, and operational parameters.



## Tier Specifications

### Protonbot (Entry Tier)

Parameter	Value
Name	Protonbot
Minimum Stake	\$100
Hourly ROI Range	0.10% - 0.12%
Daily ROI Range	0.80% - 0.96%
Trading Hours/Day	8
Trading Days/Week	6
ROI Withdrawal	After 24 hours
Capital Withdrawal	After 40 days
Investment Duration	365 days

**Best For:** New users testing the platform with minimal capital.

## Chainpulse Bot (Standard Tier)

Parameter	Value
Name	Chainpulse Bot
Minimum Stake	\$4,000
Hourly ROI Range	0.12% - 0.14%
Daily ROI Range	0.96% - 1.12%
Trading Hours/Day	8
Trading Days/Week	6
ROI Withdrawal	After 24 hours
Capital Withdrawal	After 45 days
Investment Duration	365 days

**Best For:** Serious investors seeking balanced risk/reward.

## Titan Bot (Premium Tier)

Parameter	Value
Name	Titan Bot
Minimum Stake	\$25,000
Hourly ROI Range	0.14% - 0.16%
Daily ROI Range	1.12% - 1.28%
Trading Hours/Day	8
Trading Days/Week	6
ROI Withdrawal	After 24 hours
Capital Withdrawal	After 65 days
Investment Duration	365 days

**Best For:** High-net-worth investors seeking enhanced returns.

## Omega Bot (Elite Tier)

Parameter	Value
Name	Omega Bot
Minimum Stake	\$50,000
Hourly ROI	0.225% (fixed)
Daily ROI	1.80% (fixed)
Trading Hours/Day	8
Trading Days/Week	6
ROI Withdrawal	After 24 hours
Capital Withdrawal	After 85 days
Investment Duration	365 days

**Best For:** Institutional investors requiring maximum returns with fixed rates.

## Tier Comparison Table

Feature	Probot	Chainpulse	Titan	Omega
Min. Stake	\$100	\$4,000	\$25,000	\$50,000
Daily ROI (Min)	0.80%	0.96%	1.12%	1.80%
Daily ROI (Max)	0.96%	1.12%	1.28%	1.80%
Capital Lock (Days)	40	45	65	85
Rate Type	Variable	Variable	Variable	Fixed

## ROI Calculation Methodology

### Daily Profit Calculation

$$\text{Daily Profit (Min)} = \text{Balance} \times \text{Daily ROI Min}$$
$$\text{Daily Profit (Max)} = \text{Balance} \times \text{Daily ROI Max}$$

#### Example for Chainpulse Bot with \$10,000 balance:

$$\text{Daily Profit (Min)} = \$10,000 \times 0.0096 = \$96.00$$
$$\text{Daily Profit (Max)} = \$10,000 \times 0.0112 = \$112.00$$

### Annual Projection

$$\text{Annual Profit (Min)} = \text{Balance} \times \text{Daily ROI Min} \times \text{Trading Days Per Year}$$
$$\text{Annual Profit (Max)} = \text{Balance} \times \text{Daily ROI Max} \times \text{Trading Days Per Year}$$
$$\text{Trading Days Per Year} = 6 \text{ days/week} \times 52 \text{ weeks} = 312 \text{ days}$$

#### Example for Titan Bot with \$50,000 balance:

$$\text{Annual Profit (Min)} = \$50,000 \times 0.0112 \times 312 = \$174,720$$
$$\text{Annual Profit (Max)} = \$50,000 \times 0.0128 \times 312 = \$199,680$$

## Tier Selection Logic

The system automatically selects the highest tier for which the user's balance qualifies:

```
function getBotTierForStake(stakeAmount: number): BotTier {  
  if (stakeAmount >= 50000) return 'omega';  
  if (stakeAmount >= 25000) return 'titan';  
  if (stakeAmount >= 4000) return 'chainpulse';  
  return 'protobot';  
}
```

## Manual Tier Selection

Users may explicitly select a lower tier than their balance qualifies for. However, selecting a higher tier than the balance supports will result in an error.

### Validation Rule:

User Balance  $\geq$  Selected Tier's Minimum Stake

---

## API Reference

### Base URL

Production: <https://your-worker.workers.dev>  
Development: <http://localhost:8787>

### Authentication

Currently, the API does not require authentication. For production deployments, implement authentication at the Cloudflare Workers level using:

- API Keys
- JWT Tokens

- Cloudflare Access

## Endpoints Overview

Method	Endpoint	Description
GET	/api/narrative	Get latest trading log
GET	/api/balance/:userId	Retrieve user balance
POST	/api/balance	Create user balance
PUT	/api/balance/:userId	Update user balance
GET	/health	Health check

### GET /api/narrative

Retrieves the latest AI-generated trading log.

#### Request

```
GET /api/narrative HTTP/1.1
Host: your-worker.workers.dev
Accept: application/json
```

#### Response

Success (200 OK):

```
{
  "log": "Long BTC @ 97,450 - RSI divergence on 15m, volume spike
confirming breakout",
  "timestamp": 1701234567890
}
```

## Response Schema

```
interface NarrativeResponse {
  log: string; // Trading log entry (max 100 characters)
  timestamp: number; // Unix timestamp in milliseconds
}
```

## Code Examples

### cURL:

```
curl -X GET "https://your-worker.workers.dev/api/narrative" \
-H "Accept: application/json"
```

### JavaScript (Fetch):

```
const response = await fetch('https://your-
worker.workers.dev/api/narrative');
const data = await response.json();
console.log(data.log);
```

### Python (requests):

```
import requests

response = requests.get('https://your-
worker.workers.dev/api/narrative')
data = response.json()
print(data['log'])
```

## Notes

- Logs are generated every minute via cron trigger
- If no cached log exists, one is generated on-demand
- Response is cached for 5 seconds (CDN edge cache)
- KV storage TTL is 60 seconds

## GET /api/balance/:userId

Retrieves the balance and tier configuration for a specific user.

## Request

```
GET /api/balance/user123 HTTP/1.1
Host: your-worker.workers.dev
Accept: application/json
```

## Path Parameters

Parameter	Type	Required	Description
<code>userId</code>	string	Yes	Unique user identifier (1-128 chars, alphanumeric with hyphens/underscores)



## Response

### Success (200 OK):

```
{
  "status": "success",
  "data": {
    "userId": "user123",
    "balance": 10000,
    "currency": "USD",
    "botTier": "chainpulse",
    "botTierConfig": {
      "name": "Chainpulse Bot",
      "hourlyRoiMin": 0.0012,
      "hourlyRoiMax": 0.0014,
      "dailyRoiMin": 0.0096,
      "dailyRoiMax": 0.0112,
      "minimumStake": 4000,
      "tradingHoursPerDay": 8,
      "tradingDaysPerWeek": 6,
      "roiWithdrawalHours": 24,
      "capitalWithdrawalDays": 45,
      "investmentDurationDays": 365
    },
    "dailyTargetPct": {
      "min": 0.0096,
      "max": 0.0112
    },
    "projectedDailyProfit": {
      "min": 96,
      "max": 112
    },
    "lastUpdated": 1701234567890
  }
}
```

### Error - User Not Found (404):

```
{
  "status": "error",
  "error": "User balance not found. Use POST /api/balance to set initial balance."
}
```

**Error - Invalid userId (400):**

```
{
  "status": "error",
  "error": "Invalid userId format. Must be alphanumeric with optional hyphens/underscores, 1-128 characters."
}
```

**Code Examples****cURL:**

```
curl -X GET "https://your-worker.workers.dev/api/balance/user123" \
-H "Accept: application/json"
```

**JavaScript (Fetch):**

```
async function getUserBalance(userId) {
  const response = await fetch(
    `https://your-worker.workers.dev/api/balance/${userId}`
  );

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.error);
  }

  const data = await response.json();
  return data.data;
}

// Usage
const balance = await getUserBalance('user123');
console.log(`Balance: ${balance.balance}`);
console.log(`Tier: ${balance.botTier}`);
console.log(`Daily Profit Range:
${balance.projectedDailyProfit.min} -
${balance.projectedDailyProfit.max}`);
```

**Python (requests):**

```
import requests

def get_user_balance(user_id: str) -> dict:
    response = requests.get(
        f'https://your-worker.workers.dev/api/balance/{user_id}'
    )

    if response.status_code != 200:
        error = response.json()
        raise Exception(error.get('error', 'Unknown error'))

    return response.json()['data']

# Usage
balance = get_user_balance('user123')
print(f"Balance: ${balance['balance']}")
print(f"Tier: {balance['botTier']}")
```

---

## POST /api/balance

Creates a new user balance record.

## Request

```
POST /api/balance HTTP/1.1
Host: your-worker.workers.dev
Content-Type: application/json
Accept: application/json

{
  "userId": "user123",
  "balance": 10000,
  "currency": "USD",
  "botTier": "chainpulse"
}
```

## Request Body

Field	Type	Required	Description
userId	string	Yes	Unique user identifier
balance	number	Yes	Initial balance ( $\geq 0$ )
currency	string	No	Currency code (default: "USD")
botTier	string	No	Bot tier (auto-selected if omitted)

## Supported Currencies

USD, EUR, GBP, JPY, AUD, CAD, CHF, CNY, HKD, SGD

## Valid Bot Tiers

protobot, chainpulse, titan, omega

## Response

### Success (201 Created):

```
{
  "status": "success",
  "data": {
    "userId": "user123",
    "balance": 10000,
    "currency": "USD",
    "botTier": "chainpulse",
    "botTierConfig": {
      "name": "Chainpulse Bot",
      "hourlyRoiMin": 0.0012,
      "hourlyRoiMax": 0.0014,
      "dailyRoiMin": 0.0096,
      "dailyRoiMax": 0.0112,
      "minimumStake": 4000,
      "tradingHoursPerDay": 8,
      "tradingDaysPerWeek": 6,
      "roiWithdrawalHours": 24,
      "capitalWithdrawalDays": 45,
      "investmentDurationDays": 365
    },
    "dailyTargetPct": {
      "min": 0.0096,
      "max": 0.0112
    },
    "projectedDailyProfit": {
      "min": 96,
      "max": 112
    },
    "lastUpdated": 1701234567890
  }
}
```

### Error - Missing Required Fields (400):

```
{
  "status": "error",
  "error": "userId (string) and balance (number) are required"
}
```

**Error - Insufficient Stake (400):**

```
{
  "status": "error",
  "error": "Insufficient stake for Titan Bot. Minimum stake: $25,000"
}
```

**Error - Invalid Currency (400):**

```
{
  "status": "error",
  "error": "Invalid currency. Supported currencies: USD, EUR, GBP, JPY, AUD, CAD, CHF, CNY, HKD, SGD"
}
```

**Code Examples****cURL:**

```
curl -X POST "https://your-worker.workers.dev/api/balance" \
-H "Content-Type: application/json" \
-d '{
  "userId": "user123",
  "balance": 10000,
  "currency": "USD",
  "botTier": "chainpulse"
}'
```

**JavaScript (Fetch):**

```

async function createUserBalance(userId, balance, currency = 'USD',
botTier) {
  const body = { userId, balance, currency };
  if (botTier) body.botTier = botTier;

  const response = await fetch('https://your-
worker.workers.dev/api/balance', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(body),
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.error);
  }

  return (await response.json()).data;
}

// Usage - Auto tier selection
const user1 = await createUserBalance('user123', 10000);
console.log(`Auto-selected tier: ${user1.botTier}`); // chainpulse

// Usage - Manual tier selection
const user2 = await createUserBalance('user456', 50000, 'USD',
'titan');
console.log(`Selected tier: ${user2.botTier}`); // titan

```

**Python (requests):**



```

import requests

def create_user_balance(
    user_id: str,
    balance: float,
    currency: str = 'USD',
    bot_tier: str = None
) -> dict:
    payload = {
        'userId': user_id,
        'balance': balance,
        'currency': currency,
    }

    if bot_tier:
        payload['botTier'] = bot_tier

    response = requests.post(
        'https://your-worker.workers.dev/api/balance',
        json=payload
    )

    if response.status_code != 201:
        error = response.json()
        raise Exception(error.get('error', 'Unknown error'))

    return response.json()['data']

# Usage
user = create_user_balance('user123', 10000)
print(f"Created user with tier: {user['botTier']}")

```

## PUT /api/balance/:userId

Updates an existing user's balance.

## Request

```
PUT /api/balance/user123 HTTP/1.1
Host: your-worker.workers.dev
Content-Type: application/json
Accept: application/json

{
  "balance": 15000,
  "currency": "USD",
  "botTier": "titan"
}
```

## Path Parameters

Parameter	Type	Required	Description
userId	string	Yes	Unique user identifier

## Request Body

Field	Type	Required	Description
balance	number	Yes	New balance ( $\geq 0$ )
currency	string	No	Currency code (default: "USD")
botTier	string	No	Bot tier (auto-selected if omitted)

## Response

### Success (200 OK):

```
{
  "status": "success",
  "data": {
    "userId": "user123",
    "balance": 15000,
    "currency": "USD",
    "botTier": "titan",
    "botTierConfig": {
      "name": "Titan Bot",
      "hourlyRoiMin": 0.0014,
      "hourlyRoiMax": 0.0016,
      "dailyRoiMin": 0.0112,
      "dailyRoiMax": 0.0128,
      "minimumStake": 25000,
      "tradingHoursPerDay": 8,
      "tradingDaysPerWeek": 6,
      "roiWithdrawalHours": 24,
      "capitalWithdrawalDays": 65,
      "investmentDurationDays": 365
    },
    "dailyTargetPct": {
      "min": 0.0112,
      "max": 0.0128
    },
    "projectedDailyProfit": {
      "min": 168,
      "max": 192
    },
    "lastUpdated": 1701234567890
  }
}
```

## Code Examples

cURL:

```
curl -X PUT "https://your-worker.workers.dev/api/balance/user123" \  
-H "Content-Type: application/json" \  
-d '{  
  "balance": 15000,  
  "botTier": "titan"  
'
```

**JavaScript (Fetch):**

```

async function updateUserBalance(userId, balance, currency,
botTier) {
  const body = { balance };
  if (currency) body.currency = currency;
  if (botTier) body.botTier = botTier;

  const response = await fetch(
    `https://your-worker.workers.dev/api/balance/${userId}`,
    {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(body),
    }
  );

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.error);
  }

  return (await response.json()).data;
}

// Upgrade user balance and tier
const updated = await updateUserBalance('user123', 50000, 'USD',
'omega');
console.log(`New tier: ${updated.botTier}`);
console.log(`New daily profit: ${updated.projectedDailyProfit.min}
- ${updated.projectedDailyProfit.max}`);

```

### Python (requests):

```

import requests

def update_user_balance(
    user_id: str,
    balance: float,
    currency: str = None,
    bot_tier: str = None
) -> dict:
    payload = {'balance': balance}

    if currency:
        payload['currency'] = currency
    if bot_tier:
        payload['botTier'] = bot_tier

    response = requests.put(
        f'https://your-worker.workers.dev/api/balance/{user_id}',
        json=payload
    )

    if response.status_code != 200:
        error = response.json()
        raise Exception(error.get('error', 'Unknown error'))

    return response.json()['data']

# Usage
updated = update_user_balance('user123', 50000, bot_tier='omega')
print(f"Updated tier: {updated['botTier']}")

```

## GET /health

Health check endpoint for monitoring and load balancer configuration.

## Request

```
GET /health HTTP/1.1
Host: your-worker.workers.dev
Accept: application/json
```

## Response

### Success (200 OK):

```
{
  "status": "ok",
  "service": "trading-agent-engine"
}
```

## Code Examples

### cURL:

```
curl -X GET "https://your-worker.workers.dev/health"
```

### JavaScript:

```
async function checkHealth() {
  const response = await fetch('https://your-
worker.workers.dev/health');
  const data = await response.json();
  return data.status === 'ok';
}
```

# Data Models

---

## TypeScript Interface Definitions

### UserBalance

```
interface UserBalance {  
  userId: string;      // Unique identifier (1-128 chars)  
  balance: number;     // Current balance ( $\geq 0$ )  
  currency: string;    // ISO currency code  
  botTier: BotTier;    // Selected bot tier  
  lastUpdated: number; // Unix timestamp (milliseconds)  
}
```

### BotTier

```
type BotTier = 'protobot' | 'chainpulse' | 'titan' | 'omega';
```



## BotTierConfig

```
interface BotTierConfig {  
    name: string; // Display name  
    hourlyRoiMin: number; // Minimum hourly ROI (decimal)  
    hourlyRoiMax: number; // Maximum hourly ROI (decimal)  
    dailyRoiMin: number; // Minimum daily ROI (8 × hourly)  
    dailyRoiMax: number; // Maximum daily ROI (8 × hourly)  
    minimumStake: number; // Minimum balance required  
    tradingHoursPerDay: number; // Active trading hours  
    tradingDaysPerWeek: number; // Active trading days  
    roiWithdrawalHours: number; // Hours until ROI withdrawal  
    capitalWithdrawalDays: number; // Days until capital withdrawal  
    investmentDurationDays: number; // Total investment period  
}
```

## BalanceResponse

```
interface BalanceResponse {
  status: 'success' | 'error';
  data?: {
    userId: string;
    balance: number;
    currency: string;
    botTier: BotTier;
    botTierConfig: BotTierConfig;
    dailyTargetPct: {
      min: number;
      max: number;
    };
    projectedDailyProfit: {
      min: number;
      max: number;
    };
    lastUpdated: number;
  };
  error?: string;
}
```

## NarrativeResponse

```
interface NarrativeResponse {
  log: string; // AI-generated trading log
  timestamp: number; // Generation timestamp
}
```

## Integration Guides

---

### Multi-User Platform Integration (API-Only)

This guide covers integrating the Trading Agent API into your existing platform without forking the repository.

#### Step 1: Register New Users

When a user signs up for your platform and deposits funds:

```
async function onUserDeposit(userId, amount) {
  try {
    const response = await fetch('https://your-
worker.workers.dev/api/balance', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        userId: userId,
        balance: amount,
        currency: 'USD'
      })
    });

    const result = await response.json();

    if (result.status === 'success') {
      // Store tier info in your database
      await saveUserTierInfo(userId, result.data.botTier,
result.data.botTierConfig);

      // Notify user of their tier assignment
      showNotification(`Welcome! You've been assigned to
${result.data.botTierConfig.name}`);
    }
  } catch (error) {
    console.error('Failed to register user balance:', error);
  }
}
```

## Step 2: Display Profit Projections

Fetch and display projected profits to users:

```

async function displayProfitProjection(userId) {
  const response = await fetch(`https://your-
worker.workers.dev/api/balance/${userId}`);
  const result = await response.json();

  if (result.status === 'success') {
    const { balance, botTierConfig, projectedDailyProfit } =
result.data;

    // Calculate weekly and monthly projections
    const weeklyProfit = {
      min: projectedDailyProfit.min *
botTierConfig.tradingDaysPerWeek,
      max: projectedDailyProfit.max *
botTierConfig.tradingDaysPerWeek
    };

    const monthlyProfit = {
      min: projectedDailyProfit.min *
botTierConfig.tradingDaysPerWeek * 4,
      max: projectedDailyProfit.max *
botTierConfig.tradingDaysPerWeek * 4
    };

    // Update UI
    document.getElementById('daily-profit').textContent =
      `${projectedDailyProfit.min.toFixed(2)} -
${projectedDailyProfit.max.toFixed(2)}`;
    document.getElementById('weekly-profit').textContent =
      `${weeklyProfit.min.toFixed(2)} -
${weeklyProfit.max.toFixed(2)}`;
    document.getElementById('monthly-profit').textContent =
      `${monthlyProfit.min.toFixed(2)} -
${monthlyProfit.max.toFixed(2)}`;
  }
}

```

### Step 3: Handle Balance Updates

When users deposit or withdraw:

```
async function handleBalanceChange(userId, newBalance) {
  const response = await fetch(`https://your-
worker.workers.dev/api/balance/${userId}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ balance: newBalance })
  });

  const result = await response.json();

  if (result.status === 'success') {
    // Check if tier changed
    const previousTier = await getUserStoredTier(userId);
    if (previousTier !== result.data.botTier) {
      showNotification(`Congratulations! You've been upgraded to
${result.data.botTierConfig.name}`);
    }

    // Update stored tier info
    await saveUserTierInfo(userId, result.data.botTier,
result.data.botTierConfig);
  }
}
```

#### Step 4: Polling for Live Data

For real-time trading activity display:

```

class TradingActivityPoller {
  constructor(updateInterval = 5000) {
    this.interval = updateInterval;
    this.isRunning = false;
  }

  start() {
    this.isRunning = true;
    this.poll();
  }

  stop() {
    this.isRunning = false;
  }

  async poll() {
    if (!this.isRunning) return;

    try {
      const response = await fetch('https://your-
worker.workers.dev/api/narrative');
      const data = await response.json();

      // Update UI with new trading log
      this.updateActivityFeed(data.log, data.timestamp);
    } catch (error) {
      console.error('Polling failed:', error);
    }

    // Schedule next poll
    setTimeout(() => this.poll(), this.interval);
  }

  updateActivityFeed(log, timestamp) {
    const feed = document.getElementById('activity-feed');
    const entry = document.createElement('div');
    entry.className = 'activity-entry';
    entry.innerHTML = `
      <span class="timestamp">${new
Date(timestamp).toLocaleTimeString()}</span>
      <span class="log">${log}</span>
    `;
    feed.appendChild(entry);
  }
}

```

```

    `;
    feed.prepend(entry);

    // Keep only last 50 entries
    while (feed.children.length > 50) {
        feed.removeChild(feed.lastChild);
    }
}

// Usage
const poller = new TradingActivityPoller(5000);
poller.start();

```

## Frontend JavaScript API

If embedding the Trading Agent frontend, use the global window functions:

```

// Get current trading status
const status = window.getTradingAgentStatus();
console.log(status.data.wallet_balance);
console.log(status.data.bot_tier);
console.log(status.data.projected_daily_profit);

// Set user balance (triggers tier recalculation)
const result = window.setTradingAgentBalance(25000);
console.log(result.bot_tier); // 'titan'

// Explicitly set bot tier
const tierResult = window.setBotTier('chainpulse');
console.log(tierResult.bot_tier_config);

// Get all tier configurations
const tiers = window.getBotTiers();
console.log(tiers.omega.minimumStake); // 50000

```



## Full-Stack Fork Integration

This guide covers forking the entire repository to build your platform on top of it.

### Step 1: Fork and Clone

```
# Fork via GitHub UI or CLI
gh repo fork sherrymcadams001-afk/Zus

# Clone your fork
git clone https://github.com/YOUR_USERNAME/Zus.git
cd Zus
```

### Step 2: Install Dependencies

```
# Install frontend dependencies
npm install

# Install backend dependencies
cd backend
npm install
cd ..
```

### Step 3: Configure Cloudflare

Create KV namespace:

```
cd backend
npx wrangler kv:namespace create "TRADING_CACHE"
```

Update `backend/wrangler.toml` :

```
name = "your-trading-platform"
main = "src/index.ts"
compatibility_date = "2024-01-01"
compatibility_flags = ["nodejs_compat"]

[[kv_namespaces]]
binding = "TRADING_CACHE"
id = "YOUR_KV_NAMESPACE_ID"
preview_id = "YOUR_PREVIEW_KV_NAMESPACE_ID"

[ai]
binding = "AI"

[triggers]
crons = ["* * * * *"]
```

## Step 4: Local Development

```
# Terminal 1: Run frontend
npm run dev

# Terminal 2: Run backend
cd backend
npm run dev
```

## Step 5: Customize the Platform

### Modify Bot Tiers:

Edit `backend/src/index.ts` :

```
const BOT_TIERS: Record<BotTier, BotTierConfig> = {
  protobot: {
    name: 'Your Custom Tier Name',
    hourlyRoiMin: 0.002, // Customize ROI
    hourlyRoiMax: 0.003,
    // ... other config
  },
  // ... other tiers
};
```

### Customize UI Components:

Modify React components in `src/components/` :

- `MainChart.tsx` - Trading chart
- `Watchlist.tsx` - Asset list
- `LiveTrades.tsx` - Trade history
- `BotActivityLog.tsx` - Activity feed

### Add Custom Styling:

Edit `src/index.css` and `tailwind.config.ts` :

```
@layer base {
  :root {
    --orion-neon-cyan: your-brand-color;
    --orion-neon-green: your-success-color;
    --orion-neon-red: your-error-color;
  }
}
```

## Step 6: Deploy

### Deploy Backend:

```
cd backend  
npx wrangler deploy
```

**Deploy Frontend:**

```
npm run build  
npx wrangler pages deploy dist --project-name=your-project-name
```

**Step 7: Set Up CI/CD**

The repository includes `.github/workflows/deploy.yml` :

Configure GitHub Secrets:

- `CLOUDFLARE_API_TOKEN`
- `CLOUDFLARE_ACCOUNT_ID`

Push to `main` branch to trigger automatic deployment.

---

# Security & Best Practices

---

## Input Validation Rules

### userId Validation

```
function isValidUserId(userId: string): boolean {  
  if (!userId || typeof userId !== 'string') return false;  
  if (userId.length < 1 || userId.length > 128) return false;  
  return /^[a-zA-Z0-9_-]+$/.test(userId);  
}
```

#### Rules:

- Required field
- 1-128 characters
- Alphanumeric only
- Hyphens ( - ) and underscores ( \_ ) allowed
- Case-sensitive

### Balance Validation

- Must be a number
- Must be  $\geq 0$
- If botTier specified, must meet minimum stake

## Currency Validation

```
const SUPPORTED_CURRENCIES = [  
  'USD', 'EUR', 'GBP', 'JPY', 'AUD',  
  'CAD', 'CHF', 'CNY', 'HKD', 'SGD'  
];
```

## Bot Tier Validation

```
const VALID_BOT_TIERS = ['protobot', 'chainpulse', 'titan',  
  'omega'];
```

## Rate Limiting Recommendations

For production deployments, implement rate limiting:

```
// Example: Using Cloudflare Rate Limiting
export default {
  async fetch(request: Request, env: Env, ctx: ExecutionContext) {
    // Check rate limit (example: 100 requests per minute per IP)
    const ip = request.headers.get('CF-Connecting-IP');
    const rateLimitKey = `rate_limit:${ip}`;

    const count = await env.TRADING_CACHE.get(rateLimitKey);
    if (count && parseInt(count) > 100) {
      return new Response('Rate limit exceeded', { status: 429 });
    }

    // Increment counter
    await env.TRADING_CACHE.put(
      rateLimitKey,
      String((parseInt(count) || '0') + 1),
      { expirationTtl: 60 }
    );

    // Process request...
  }
};
```

## Error Handling Patterns

### Consistent Error Response Format

```
interface ErrorResponse {
  status: 'error';
  error: string;
  code?: string; // Optional error code
}
```

## Client-Side Error Handling

```
async function apiCall(endpoint, options = {}) {
  try {
    const response = await fetch(endpoint, options);
    const data = await response.json();

    if (data.status === 'error') {
      throw new ApiError(data.error, response.status);
    }

    return data;
  } catch (error) {
    if (error instanceof ApiError) {
      // Handle API errors
      if (error.statusCode === 400) {
        showValidationError(error.message);
      } else if (error.statusCode === 404) {
        showNotFoundError(error.message);
      } else {
        showGenericError(error.message);
      }
    } else {
      // Handle network errors
      showNetworkError('Unable to connect to server');
    }
    throw error;
  }
}

class ApiError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
    this.name = 'ApiError';
  }
}
```



## CORS Configuration

The API includes permissive CORS headers for development:

```
const corsHeaders = {  
  'Access-Control-Allow-Origin': '*',  
  'Access-Control-Allow-Methods': 'GET, POST, PUT, OPTIONS',  
  'Access-Control-Allow-Headers': 'Content-Type',  
};
```

**For production, restrict to your domains:**

```
const corsHeaders = {  
  'Access-Control-Allow-Origin': 'https://your-domain.com',  
  'Access-Control-Allow-Methods': 'GET, POST, PUT, OPTIONS',  
  'Access-Control-Allow-Headers': 'Content-Type, Authorization',  
};
```

---

# Appendices

---

## Appendix A: Glossary

Term	Definition
Bot Tier	Investment level determining ROI rates and features
KV Storage	Cloudflare's key-value storage for persistent data
ROI	Return on Investment, expressed as decimal (0.01 = 1%)
Stake	User's invested balance
Workers AI	Cloudflare's AI inference platform
Trading Narrative	AI-generated log describing simulated trading activity

## Appendix B: ROI Calculation Formulas

### Daily ROI Calculation

```
Hourly ROI = Base Rate × Market Modifier  
Daily ROI = Hourly ROI × Trading Hours Per Day  
Daily Profit = Balance × Daily ROI
```

### Annual Projection

```
Trading Days Per Year = Trading Days Per Week × 52  
Annual Profit (Min) = Balance × Daily ROI Min × Trading Days Per Year  
Annual Profit (Max) = Balance × Daily ROI Max × Trading Days Per Year
```

### Example Calculation

For a \$10,000 balance in Chainpulse tier:

Daily ROI Min = 0.0096 (0.96%)

Daily ROI Max = 0.0112 (1.12%)

Daily Profit Min = \$10,000 × 0.0096 = \$96

Daily Profit Max = \$10,000 × 0.0112 = \$112

Weekly Profit Min = \$96 × 6 = \$576

Weekly Profit Max = \$112 × 6 = \$672

Annual Profit Min = \$96 × 312 = \$29,952

Annual Profit Max = \$112 × 312 = \$34,944

## Appendix C: Complete Code Examples

### Full Integration Example (JavaScript)

```
class TradingAgentClient {
  constructor(baseUrl) {
    this.baseUrl = baseUrl;
  }

  async createUser(userId, balance, currency = 'USD', botTier =
null) {
    const body = { userId, balance, currency };
    if (botTier) body.botTier = botTier;

    const response = await fetch(`${this.baseUrl}/api/balance`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(body)
    });

    return this._handleResponse(response);
  }

  async getUser(userId) {
    const response = await
fetch(`${this.baseUrl}/api/balance/${userId}`);
    return this._handleResponse(response);
  }

  async updateUser(userId, balance, currency = null, botTier =
null) {
    const body = { balance };
    if (currency) body.currency = currency;
    if (botTier) body.botTier = botTier;

    const response = await
fetch(`${this.baseUrl}/api/balance/${userId}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(body)
    });
  }
}
```

```

    });

    return this._handleResponse(response);
  }

  async getNarrative() {
    const response = await fetch(`${this.baseUrl}/api/narrative`);
    return response.json();
  }

  async healthCheck() {
    const response = await fetch(`${this.baseUrl}/health`);
    const data = await response.json();
    return data.status === 'ok';
  }

  async _handleResponse(response) {
    const data = await response.json();

    if (data.status === 'error') {
      throw new Error(data.error);
    }

    return data.data;
  }
}

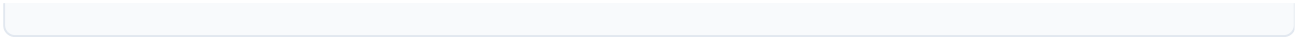
// Usage
const client = new TradingAgentClient('https://your-worker.workers.dev');

// Create user
const user = await client.createUser('user123', 10000);
console.log(`User tier: ${user.botTier}`);

// Get profit projections
const daily = user.projectedDailyProfit;
console.log(`Daily profit: ${daily.min} - ${daily.max}`);

// Update balance
const updated = await client.updateUser('user123', 50000);
console.log(`New tier: ${updated.botTier}`);

```



## Full Integration Example (Python)

```
import requests
from typing import Optional, Dict, Any

class TradingAgentClient:
    def __init__(self, base_url: str):
        self.base_url = base_url

    def create_user(
        self,
        user_id: str,
        balance: float,
        currency: str = 'USD',
        bot_tier: Optional[str] = None
    ) -> Dict[str, Any]:
        payload = {
            'userId': user_id,
            'balance': balance,
            'currency': currency
        }

        if bot_tier:
            payload['botTier'] = bot_tier

        response = requests.post(
            f'{self.base_url}/api/balance',
            json=payload
        )

        return self._handle_response(response)

    def get_user(self, user_id: str) -> Dict[str, Any]:
        response = requests.get(
            f'{self.base_url}/api/balance/{user_id}'
        )
        return self._handle_response(response)

    def update_user(
        self,
        user_id: str,
```

```

        balance: float,
        currency: Optional[str] = None,
        bot_tier: Optional[str] = None
    ) -> Dict[str, Any]:
        payload = {'balance': balance}

        if currency:
            payload['currency'] = currency
        if bot_tier:
            payload['botTier'] = bot_tier

        response = requests.put(
            f'{self.base_url}/api/balance/{user_id}',
            json=payload
        )

        return self._handle_response(response)

    def get_narrative(self) -> Dict[str, Any]:
        response = requests.get(f'{self.base_url}/api/narrative')
        return response.json()

    def health_check(self) -> bool:
        response = requests.get(f'{self.base_url}/health')
        data = response.json()
        return data.get('status') == 'ok'

    def _handle_response(self, response: requests.Response) ->
Dict[str, Any]:
        data = response.json()

        if data.get('status') == 'error':
            raise Exception(data.get('error', 'Unknown error'))

        return data.get('data')

# Usage
if __name__ == '__main__':
    client = TradingAgentClient('https://your-worker.workers.dev')

    # Create user
    user = client.create_user('user123', 10000)

```



```
print(f"User tier: {user['botTier']}")

# Get profit projections
daily = user['projectedDailyProfit']
print(f"Daily profit: ${daily['min']} - ${daily['max']}")

# Update balance
updated = client.update_user('user123', 50000)
print(f"New tier: {updated['botTier']}")
```

## Appendix D: Troubleshooting Guide

### Common Issues

#### Issue: "User balance not found" (404)

*Cause:* The userId has not been registered yet.

*Solution:* Use `POST /api/balance` to create the user first.

```
curl -X POST "https://your-worker.workers.dev/api/balance" \
-H "Content-Type: application/json" \
-d '{"userId": "user123", "balance": 1000}'
```

---

#### Issue: "Invalid userId format" (400)

*Cause:* The userId contains invalid characters or is too long.

*Solution:* Ensure userId:

- Is 1-128 characters
- Contains only alphanumeric characters, hyphens, or underscores
- Does not contain spaces or special characters

**Issue: "Insufficient stake for [Tier Name]" (400)**

*Cause:* The balance is below the minimum stake for the selected tier.

*Solution:* Either:

1. Increase the balance to meet the minimum
  2. Select a lower tier
  3. Omit `botTier` to use auto-selection
- 

**Issue: CORS Errors**

*Cause:* Browser blocking cross-origin requests.

*Solution:* The API includes CORS headers. If issues persist:

1. Ensure you're making requests from an allowed origin
  2. Check that preflight (OPTIONS) requests are succeeding
  3. For development, use a local proxy or disable CORS in browser
- 

**Issue: Empty Narrative Response**

*Cause:* AI generation failed or no cached log exists.

*Solution:* The system auto-generates on demand. If persistent:

1. Check Workers AI binding is configured
2. Verify cron trigger is running
3. Check Cloudflare dashboard for errors

## Appendix E: FAQ

### Q: How often are trading narratives generated?

A: Every minute via cron trigger. On-demand generation occurs if no cached log exists.

---

### Q: Can I use multiple currencies for different users?

A: Yes, each user can have a different currency. Supported: USD, EUR, GBP, JPY, AUD, CAD, CHF, CNY, HKD, SGD.

---

### Q: What happens if a user's balance drops below their tier minimum?

A: The tier is not automatically downgraded. Use `PUT /api/balance` with auto-selection (omit `botTier`) to recalculate.

---

### Q: Is user data encrypted?

A: Data is stored in Cloudflare KV with encryption at rest. For additional security, implement your own encryption layer.

---

### Q: What's the difference between hourly and daily ROI?

A:  $\text{Daily ROI} = \text{Hourly ROI} \times \text{Trading Hours (8)}$ . The system uses daily ROI for projections.

---

### Q: Can I customize the bot tier configurations?

A: Yes, modify the `BOT_TIERS` constant in `backend/src/index.ts` and redeploy.

---

**Q: How do I add authentication?**

A: Implement authentication at the Workers level using:

- Cloudflare Access
  - Custom JWT validation
  - API key verification
- 

**Q: Is there a webhook for balance changes?**

A: Not currently. Implement polling or add webhook functionality in your fork.

---

**Q: What's the data retention policy?**

A: KV data persists indefinitely unless explicitly deleted or TTL-expired.

---

## Document Information

---

Property	Value
Document Version	1.0.0
API Version	1.0.0
Last Updated	November 2024
Author	Zus Trading Agent Team
License	Proprietary

---

*This document is confidential and intended for authorized developers only.*