AIAlgoTradeHits.com - Step-by-Step Implementation Plan

Project Overview

Platform: Google Cloud Platform (GCP)

Application: Cloud Run (containerized)

Database: BigQuery

Domain: AIAlgoTradeHits.com

Development Environment: VSCode with Claude AI

Data Source: KrakenPro API

Existing Assets:

- ✅ 3 Crypto Scheduled Jobs (daily, hourly, 5-minute)
- ✅ 3 Crypto BigQuery Tables
- ✅ Trading folder structure

To Be Created:

- ❌ 3 Stock Scheduled Jobs (daily, hourly, 5-minute)
- ❌ Stock BigQuery Tables
- ❌ Backend API
- ❌ Frontend Application
- ❌ Trading Engine
- ❌ Sentiment Analysis System

Implementation Timeline Overview

Total Duration: 12-16 weeks

Phase	Duration	Status
Phase 0: Project Setup & Planning	1 week	🟡 Planning
Phase 1: Database Architecture	1 week	🔴 Pending
Phase 2: Data Collection Pipeline	2 weeks	🔴 Pending
Phase 3: Backend API Development	3 weeks	🔴 Pending
Phase 4: Trading Algorithm Core	2 weeks	🔴 Pending
Phase 5: Frontend Development	3 weeks	🔴 Pending
Phase 6: Integration & Testing	2 weeks	🔴 Pending
Phase 7: Paper Trading Phase	2 weeks	🔴 Pending
Phase 8: Small Money Testing	2 weeks	🔴 Pending
Phase 9: Beta User Testing	2 weeks	🔴 Pending
Phase 10: Production Launch	1 week	🔴 Pending

PHASE 0: Project Setup & Planning (Week 1)

Objectives

- Set up development environment
- Configure GCP project
- Establish repository structure
- Define coding standards

Tasks

0.1 GCP Project Configuration

bash

```
# Set up GCP project
gcloud config set project aialgo-tradehits

# Enable required APIs
gcloud services enable \
  run.googleapis.com \
  bigquery.googleapis.com \
  cloudfunctions.googleapis.com \
  cloudscheduler.googleapis.com \
  secretmanager.googleapis.com \
  cloudbuild.googleapis.com \
  artifactregistry.googleapis.com

# Set default region
gcloud config set compute/region us-central1
```

0.2 Repository Structure

```
aialgo-tradehits/
├── backend/
│   ├── api/
│   │   ├── routes/
│   │   ├── controllers/
│   │   ├── middleware/
│   │   └── utils/
│   ├── trading-engine/
│   │   ├── algorithms/
│   │   ├── indicators/
│   │   └── sentiment/
│   ├── schedulers/
│   │   ├── crypto/
│   │   │   ├── daily_crypto_job.py
│   │   │   ├── hourly_crypto_job.py
│   │   │   └── minute5_crypto_job.py
│   │   └── stock/
│   │       ├── daily_stock_job.py
│   │       ├── hourly_stock_job.py
│   │       └── minute5_stock_job.py
│   ├── tests/
│   ├── requirements.txt
│   └── Dockerfile
├── frontend/
```

```
│   ├── src/
│   │   ├── components/
│   │   ├── pages/
│   │   ├── hooks/
│   │   ├── services/
│   │   └── utils/
│   ├── public/
│   ├── package.json
│   └── Dockerfile
├── infrastructure/
│   ├── terraform/
│   ├── bigquery/
│   │   ├── schemas/
│   │   └── migrations/
│   └── cloud-functions/
├── docs/
├── tests/
│   ├── integration/
│   ├── e2e/
│   └── load/
└── scripts/
    ├── deploy/
    └── setup/
```

0.3 VSCode with Claude Setup

1. Install VSCode extensions:
   - Claude for VSCode
   - Python
   - ESLint
   - Prettier
   - Docker
   - Google Cloud Code
2. Configure Claude in VSCode:
   - Set up API key
   - Configure project context
   - Enable code suggestions

0.4 Development Environment

```bash
# Create Python virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install development dependencies
pip install -r requirements-dev.txt

# Install frontend dependencies
cd frontend
npm install
```

0.5 Secret Management

```bash
# Store API keys in Secret Manager
echo -n "YOUR_KRAKEN_API_KEY" | gcloud secrets create kraken-api-key --data-file=-
echo -n "YOUR_KRAKEN_SECRET" | gcloud secrets create kraken-api-secret --data-file=-
echo -n "YOUR_CMC_API_KEY" | gcloud secrets create coinmarketcap-api-key --data-file=-
echo -n "YOUR_TWITTER_KEY" | gcloud secrets create twitter-api-key --data-file=-
```

Deliverables

- ✅ GCP project configured
- ✅ Repository structure created
- ✅ Development environment ready
- ✅ Secrets stored in Secret Manager
- ✅ Documentation started

PHASE 1: Database Architecture (Week 2)

Objectives

- Design BigQuery schema
- Create all required tables

- Set up data partitioning and clustering
- Implement backup strategy

Tasks

1.1 BigQuery Dataset Creation

```bash
# Create datasets
bq mk --dataset \
  --location=US \
  aialgo-tradehits:crypto_data

bq mk --dataset \
  --location=US \
  aialgo-tradehits:stock_data

bq mk --dataset \
  --location=US \
  aialgo-tradehits:user_data

bq mk --dataset \
  --location=US \
  aialgo-tradehits:trading_data

bq mk --dataset \
  --location=US \
  aialgo-tradehits:sentiment_data

bq mk --dataset \
  --location=US \
  aialgo-tradehits:paper_trading_data
```

1.2 Crypto Tables (Already Exist - Verify)

Verify existing tables:

- `crypto_data.daily_crypto_data`
- `crypto_data.hourly_crypto_data`
- `crypto_data.minute5_crypto_data`

1.3 Stock Tables (To Be Created)

sql

```sql
-- File: infrastructure/bigquery/schemas/stock_tables.sql

-- Daily Stock Data
CREATE TABLE IF NOT EXISTS stock_data.daily_stock_data (
  id STRING NOT NULL,
  symbol STRING NOT NULL,
  date DATE NOT NULL,
  open_price FLOAT64,
  high_price FLOAT64,
  low_price FLOAT64,
  close_price FLOAT64,
  volume INT64,
  trade_count INT64,
  vwap FLOAT64,

  -- Technical Indicators
  rsi FLOAT64,
  macd FLOAT64,
  macd_signal FLOAT64,
  macd_histogram FLOAT64,
  bb_upper FLOAT64,
  bb_middle FLOAT64,
  bb_lower FLOAT64,
  sma_9 FLOAT64,
  sma_21 FLOAT64,
  sma_50 FLOAT64,
  sma_200 FLOAT64,
  ema_9 FLOAT64,
  ema_21 FLOAT64,

  -- Stock specific
  market_cap FLOAT64,
  pe_ratio FLOAT64,
  dividend_yield FLOAT64,

  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
)
PARTITION BY date
CLUSTER BY symbol
OPTIONS(
  description="Daily stock OHLCV data with technical indicators"
);
```

```
-- Hourly Stock Data
CREATE TABLE IF NOT EXISTS stock_data.hourly_stock_data (
  id STRING NOT NULL,
  symbol STRING NOT NULL,
  timestamp TIMESTAMP NOT NULL,
  open_price FLOAT64,
  high_price FLOAT64,
  low_price FLOAT64,
  close_price FLOAT64,
  volume INT64,
  trade_count INT64,
  vwap FLOAT64,

  -- Technical Indicators
  rsi FLOAT64,
  macd FLOAT64,
  macd_signal FLOAT64,
  macd_histogram FLOAT64,
  bb_upper FLOAT64,
  bb_middle FLOAT64,
  bb_lower FLOAT64,
  sma_9 FLOAT64,
  sma_21 FLOAT64,
  ema_9 FLOAT64,
  ema_21 FLOAT64,

  is_market_hours BOOL,

  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
)
PARTITION BY DATE(timestamp)
CLUSTER BY symbol
OPTIONS(
  description="Hourly stock OHLCV data with technical indicators"
);

-- 5-Minute Stock Data
CREATE TABLE IF NOT EXISTS stock_data.minute5_stock_data (
  id STRING NOT NULL,
  symbol STRING NOT NULL,
  timestamp TIMESTAMP NOT NULL,
  open_price FLOAT64,
  high_price FLOAT64,
  low_price FLOAT64,
```

```
  close_price FLOAT64,
  volume INT64,
  trade_count INT64,
  vwap FLOAT64,

  -- Technical Indicators
  rsi FLOAT64,
  macd FLOAT64,
  macd_signal FLOAT64,
  macd_histogram FLOAT64,
  sma_9 FLOAT64,
  sma_21 FLOAT64,
  ema_9 FLOAT64,
  ema_21 FLOAT64,

  is_market_hours BOOL,

  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
)
PARTITION BY DATE(timestamp)
CLUSTER BY symbol
OPTIONS(
  description="5-minute stock OHLCV data with technical indicators",
  partition_expiration_days=90
);

-- Stock Market Calendar
CREATE TABLE IF NOT EXISTS stock_data.market_calendar (
  date DATE NOT NULL,
  is_trading_day BOOL DEFAULT TRUE,
  market_open_time TIME,
  market_close_time TIME,
  holiday_name STRING,
  early_close BOOL DEFAULT FALSE,
  notes STRING
)
PARTITION BY date
OPTIONS(
  description="Stock market trading calendar"
);
```

1.4 User & Trading Tables

sql

```sql
-- File: infrastructure/bigquery/schemas/user_trading_tables.sql

-- Users Table
CREATE TABLE IF NOT EXISTS user_data.users (
  user_id STRING NOT NULL,
  username STRING NOT NULL,
  email STRING NOT NULL,
  password_hash STRING NOT NULL,
  full_name STRING,
  role STRING DEFAULT 'user',

  crypto_balance FLOAT64 DEFAULT 0.0,
  stock_balance FLOAT64 DEFAULT 0.0,
  total_balance FLOAT64 DEFAULT 0.0,

  is_active BOOL DEFAULT TRUE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
  last_login TIMESTAMP
)
CLUSTER BY user_id;

-- Trade Parameters
CREATE TABLE IF NOT EXISTS trading_data.trade_parameters (
  param_id STRING NOT NULL,
  user_id STRING NOT NULL,
  asset_type STRING NOT NULL,
  symbol STRING,

  trade_amount FLOAT64,
  take_profit_percentage FLOAT64,
  stop_loss_percentage FLOAT64,
  max_concurrent_trades INT64 DEFAULT 1,

  buy_at_lowest_only BOOL DEFAULT TRUE,
  min_sentiment_score FLOAT64 DEFAULT 0.0,
  require_trump_positive BOOL DEFAULT FALSE,

  auto_trading_enabled BOOL DEFAULT FALSE,
  trading_hours_start TIME,
  trading_hours_end TIME,

  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
```

```
)
PARTITION BY DATE(created_at)
CLUSTER BY user_id, asset_type;

-- Executed Trades
CREATE TABLE IF NOT EXISTS trading_data.executed_trades (
  trade_id STRING NOT NULL,
  user_id STRING NOT NULL,
  asset_type STRING NOT NULL,
  symbol STRING NOT NULL,
  trade_type STRING NOT NULL,

  entry_price FLOAT64,
  exit_price FLOAT64,
  quantity FLOAT64,
  trade_amount FLOAT64,

  profit_loss FLOAT64,
  profit_loss_percentage FLOAT64,

  take_profit_target FLOAT64,
  stop_loss_target FLOAT64,

  execution_reason STRING,
  timeframe_detected STRING,

  sentiment_score_at_buy FLOAT64,
  trump_sentiment_at_buy FLOAT64,
  was_at_lowest_point BOOL,
  cycle_indicators JSON,

  executed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
  closed_at TIMESTAMP,
  status STRING DEFAULT 'open',

  is_paper_trade BOOL DEFAULT FALSE
)
PARTITION BY DATE(executed_at)
CLUSTER BY user_id, asset_type, status;

-- Paper Trading Trades (separate for testing)
CREATE TABLE IF NOT EXISTS paper_trading_data.paper_trades (
  -- Same structure as executed_trades but specifically for virtual testing
  trade_id STRING NOT NULL,
```

```
  user_id STRING NOT NULL,
  asset_type STRING NOT NULL,
  symbol STRING NOT NULL,
  trade_type STRING NOT NULL,

  entry_price FLOAT64,
  exit_price FLOAT64,
  quantity FLOAT64,
  trade_amount FLOAT64,

  profit_loss FLOAT64,
  profit_loss_percentage FLOAT64,

  take_profit_target FLOAT64,
  stop_loss_target FLOAT64,

  execution_reason STRING,
  timeframe_detected STRING,

  sentiment_score_at_buy FLOAT64,
  trump_sentiment_at_buy FLOAT64,
  was_at_lowest_point BOOL,
  cycle_indicators JSON,

  executed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
  closed_at TIMESTAMP,
  status STRING DEFAULT 'open'
)
PARTITION BY DATE(executed_at)
CLUSTER BY user_id, asset_type;

-- Lowest Point Detection
CREATE TABLE IF NOT EXISTS trading_data.lowest_point_detection (
  detection_id STRING NOT NULL,
  asset_type STRING NOT NULL,
  symbol STRING NOT NULL,
  timeframe STRING,

  current_price FLOAT64,
  lowest_price_24h FLOAT64,
  lowest_price_7d FLOAT64,
  is_at_support_level BOOL,

  rsi_value FLOAT64,
```

```
  macd_histogram FLOAT64,
  bb_position FLOAT64,

  sentiment_score FLOAT64,
  trump_sentiment FLOAT64,

  is_lowest_point BOOL,
  rise_probability FLOAT64,
  confidence_score FLOAT64,

  buy_signal_triggered BOOL DEFAULT FALSE,
  trade_executed BOOL DEFAULT FALSE,

  detected_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
)
PARTITION BY DATE(detected_at)
CLUSTER BY symbol, is_lowest_point;
```

1.5 Sentiment Tables

sql

```sql
-- File: infrastructure/bigquery/schemas/sentiment_tables.sql

-- Crypto Sentiment
CREATE TABLE IF NOT EXISTS sentiment_data.crypto_sentiment (
  sentiment_id STRING NOT NULL,
  symbol STRING NOT NULL,
  source STRING NOT NULL,
  sentiment_score FLOAT64,
  sentiment_category STRING,

  article_title STRING,
  article_url STRING,
  tweet_id STRING,
  tweet_text STRING,
  author STRING,

  engagement_score INT64,
  credibility_score FLOAT64,

  timestamp TIMESTAMP NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
)
PARTITION BY DATE(timestamp)
CLUSTER BY symbol, source;

-- Stock Sentiment
CREATE TABLE IF NOT EXISTS sentiment_data.stock_sentiment (
  sentiment_id STRING NOT NULL,
  symbol STRING NOT NULL,
  source STRING NOT NULL,
  sentiment_score FLOAT64,
  sentiment_category STRING,

  article_title STRING,
  article_url STRING,
  tweet_id STRING,
  tweet_text STRING,
  author STRING,

  analyst_rating STRING,
  price_target FLOAT64,

  engagement_score INT64,
```

```
  credibility_score FLOAT64,

  timestamp TIMESTAMP NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
)
PARTITION BY DATE(timestamp)
CLUSTER BY symbol, source;

-- Trump Announcements
CREATE TABLE IF NOT EXISTS sentiment_data.trump_announcements (
  announcement_id STRING NOT NULL,
  tweet_id STRING,
  tweet_text STRING NOT NULL,
  tweet_url STRING,
  posted_at TIMESTAMP NOT NULL,

  mentioned_assets ARRAY<STRING>,
  sentiment_overall FLOAT64,
  impact_prediction STRING,

  triggered_trades BOOL DEFAULT FALSE,
  assets_affected ARRAY<STRING>,

  processed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
)
PARTITION BY DATE(posted_at)
CLUSTER BY DATE(posted_at);

-- Sentiment Aggregation
CREATE TABLE IF NOT EXISTS sentiment_data.sentiment_aggregation (
  agg_id STRING NOT NULL,
  asset_type STRING NOT NULL,
  symbol STRING NOT NULL,
  timeframe STRING,

  average_sentiment FLOAT64,
  sentiment_trend STRING,
  bullish_mentions INT64,
  bearish_mentions INT64,
  total_mentions INT64,

  coinmarketcap_sentiment FLOAT64,
  trump_sentiment FLOAT64,
  news_sentiment FLOAT64,
```

```
  social_sentiment FLOAT64,

  trading_signal STRING,
  confidence_level FLOAT64,

  calculated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
)
PARTITION BY DATE(calculated_at)
CLUSTER BY asset_type, symbol;
```

1.6 Deploy Tables

```
bash

# Deploy all tables
bq query --use_legacy_sql=false < infrastructure/bigquery/schemas/stock_tables.sql
bq query --use_legacy_sql=false < infrastructure/bigquery/schemas/user_trading_tables.sql
bq query --use_legacy_sql=false < infrastructure/bigquery/schemas/sentiment_tables.sql
```

1.7 Initial Data Population

```
bash

# Populate market calendar with 2025-2026 holidays
python scripts/setup/populate_market_calendar.py
```

Testing

- ✅ Verify all tables created
- ✅ Test partitioning and clustering
- ✅ Validate data insertion
- ✅ Check query performance

Deliverables

- ✅ All BigQuery tables created
- ✅ Partitioning and clustering configured
- ✅ Market calendar populated
- ✅ Documentation updated

PHASE 2: Data Collection Pipeline (Weeks 3-4)

Objectives

- Create 3 stock scheduler jobs
- Enhance existing crypto jobs
- Implement error handling and retries
- Set up monitoring

Tasks

2.1 Stock Scheduler Jobs

2.1.1 Daily Stock Job

python

```python
# File: backend/schedulers/stock/daily_stock_job.py

import os
import krakenex
from google.cloud import bigquery
from datetime import datetime, timedelta
import pandas as pd
from technical_indicators import calculate_all_indicators

def fetch_daily_stock_data(symbol, days=90):
    """Fetch daily stock data from Kraken"""
    kraken = krakenex.API()
    kraken.key = os.getenv('KRAKEN_API_KEY')
    kraken.secret = os.getenv('KRAKEN_API_SECRET')

    # Kraken API call for stock data
    response = kraken.query_public('OHLC', {
        'pair': symbol,
        'interval': 1440  # Daily
    })

    return response

def calculate_indicators(df):
    """Calculate technical indicators"""
    indicators = calculate_all_indicators(df)
    return indicators

def store_to_bigquery(data, symbol):
    """Store data in BigQuery"""
    client = bigquery.Client()
    table_id = "aialgo-tradehits.stock_data.daily_stock_data"

    # Transform data
    rows = []
    for row in data:
        rows.append({
            'id': f"{symbol}_{row['date']}",
            'symbol': symbol,
            'date': row['date'],
            'open_price': row['open'],
            'high_price': row['high'],
            'low_price': row['low'],
```

```
            'close_price': row['close'],
            'volume': row['volume'],
            'vwap': row.get('vwap'),
            # ... indicators
            'created_at': datetime.utcnow().isoformat()
        })

    errors = client.insert_rows_json(table_id, rows)
    if errors:
        print(f"Errors: {errors}")
    else:
        print(f"Stored {len(rows)} rows for {symbol}")

def main(request):
    """Cloud Function entry point"""
    stocks = [
        'AAPL', 'GOOGL', 'MSFT', 'AMZN', 'TSLA', 'NVDA', 'AMD',
        'META', 'NFLX', 'DIS', 'SPY', 'QQQ', 'DIA'
    ]

    for symbol in stocks:
        try:
            data = fetch_daily_stock_data(symbol)
            indicators = calculate_indicators(data)
            store_to_bigquery(indicators, symbol)
        except Exception as e:
            print(f"Error processing {symbol}: {e}")
            continue

    return {'status': 'success', 'processed': len(stocks)}

if __name__ == "__main__":
    main(None)
```

2.1.2 Hourly Stock Job

python

```python
# File: backend/schedulers/stock/hourly_stock_job.py

import os
from google.cloud import bigquery
from datetime import datetime
import krakenex

def is_market_open():
    """Check if stock market is open"""
    now = datetime.now()

    # Check if weekend
    if now.weekday() >= 5:
        return False

    # Check if within market hours (9:30 AM - 4:00 PM EST)
    # Convert to EST and check
    # ... implementation

    return True

def fetch_hourly_stock_data(symbol):
    """Fetch hourly stock data"""
    if not is_market_open():
        print("Market closed, skipping")
        return None

    # Fetch from Kraken
    # ... similar to daily job

def main(request):
    """Cloud Function entry point"""
    if not is_market_open():
        return {'status': 'skipped', 'reason': 'market_closed'}

    stocks = ['AAPL', 'GOOGL', 'MSFT', 'TSLA', 'NVDA']

    for symbol in stocks:
        try:
            data = fetch_hourly_stock_data(symbol)
            if data:
                store_to_bigquery(data, symbol, 'hourly_stock_data')
        except Exception as e:
```

```
        print(f"Error: {e}")

    return {'status': 'success'}
```

2.1.3 5-Minute Stock Job

```
python

# File: backend/schedulers/stock/minute5_stock_job.py


# Similar structure to hourly, but runs every 5 minutes during market hours
# Only stores data when market is open
```

2.2 Deploy Cloud Functions

```
bash
```

```
# Deploy daily stock job
gcloud functions deploy daily-stock-job \
  --gen2 \
  --runtime=python311 \
  --region=us-central1 \
  --source=./backend/schedulers/stock \
  --entry-point=main \
  --trigger-http \
  --set-secrets='KRAKEN_API_KEY=kraken-api-key:latest,KRAKEN_API_SECRET=kraken-api-secret:latest'

# Deploy hourly stock job
gcloud functions deploy hourly-stock-job \
  --gen2 \
  --runtime=python311 \
  --region=us-central1 \
  --source=./backend/schedulers/stock \
  --entry-point=main \
  --trigger-http \
  --set-secrets='KRAKEN_API_KEY=kraken-api-key:latest'

# Deploy 5-minute stock job
gcloud functions deploy minute5-stock-job \
  --gen2 \
  --runtime=python311 \
  --region=us-central1 \
  --source=./backend/schedulers/stock \
  --entry-point=main \
  --trigger-http \
  --set-secrets='KRAKEN_API_KEY=kraken-api-key:latest'
```

2.3 Create Cloud Scheduler Jobs

bash

```
# Daily stock job - 6 AM EST every day
gcloud scheduler jobs create http daily-stock-scheduler \
  --location=us-central1 \
  --schedule="0 6 * * *" \
  --time-zone="America/New_York" \
  --uri="https://us-central1-aialgo-tradehits.cloudfunctions.net/daily-stock-job" \
  --http-method=POST

# Hourly stock job - Every hour 9 AM - 5 PM EST on weekdays
gcloud scheduler jobs create http hourly-stock-scheduler \
  --location=us-central1 \
  --schedule="0 9-17 * * 1-5" \
  --time-zone="America/New_York" \
  --uri="https://us-central1-aialgo-tradehits.cloudfunctions.net/hourly-stock-job" \
  --http-method=POST

# 5-minute stock job - Every 5 min 9:30 AM - 4 PM EST on weekdays
gcloud scheduler jobs create http minute5-stock-scheduler \
  --location=us-central1 \
  --schedule="*/5 9-16 * * 1-5" \
  --time-zone="America/New_York" \
  --uri="https://us-central1-aialgo-tradehits.cloudfunctions.net/minute5-stock-job" \
  --http-method=POST
```

2.4 Enhance Crypto Jobs (if needed)

Review and update existing crypto jobs to match the same structure and error handling.

2.5 Monitoring Setup

bash

```
# Create log-based alerts
gcloud logging metrics create scheduler-errors \
  --description="Count of scheduler job errors" \
  --log-filter='resource.type="cloud_function" AND severity>=ERROR'

# Create alert policy
gcloud alpha monitoring policies create \
  --notification-channels=CHANNEL_ID \
  --display-name="Scheduler Job Failures" \
  --condition-display-name="Error count > 5" \
  --condition-threshold-value=5 \
  --condition-threshold-duration=300s
```

Testing

```
bash

# Test individual jobs
curl -X POST https://us-central1-aialgo-tradehits.cloudfunctions.net/daily-stock-job

# Check BigQuery for data
bq query --use_legacy_sql=false \
  'SELECT COUNT(*) FROM `aialgo-tradehits.stock_data.daily_stock_data`'

# Verify scheduler execution
gcloud scheduler jobs describe daily-stock-scheduler --location=us-central1
```

Deliverables

- ✅ 3 stock Cloud Functions deployed
- ✅ 3 Cloud Scheduler jobs configured
- ✅ Monitoring and alerting set up
- ✅ Test data populated in BigQuery
- ✅ Documentation updated

PHASE 3: Backend API Development (Weeks 5-7)

Objectives

- Create RESTful API
- Implement authentication
- Build data access layer
- Deploy to Cloud Run

Tasks

3.1 API Structure

```
backend/api/
├── main.py
├── routes/
│   ├── auth.py
│   ├── users.py
│   ├── markets.py
│   ├── trades.py
│   └── sentiment.py
├── controllers/
│   ├── user_controller.py
│   ├── trade_controller.py
│   └── market_controller.py
├── services/
│   ├── bigquery_service.py
│   ├── auth_service.py
│   └── trading_service.py
├── models/
│   ├── user.py
│   ├── trade.py
│   └── market_data.py
├── middleware/
│   ├── auth_middleware.py
│   └── error_handler.py
└── utils/
    ├── validators.py
    └── helpers.py
```

3.2 Main API Application

python

```python
# File: backend/api/main.py

from fastapi import FastAPI, Depends
from fastapi.middleware.cors import CORSMiddleware
from routes import auth, users, markets, trades, sentiment
import uvicorn

app = FastAPI(
    title="AIAlgoTradeHits API",
    description="AI-Powered Trading Platform API",
    version="1.0.0"
)

# CORS Configuration
app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://aialgotradetits.com", "http://localhost:3000"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include routers
app.include_router(auth.router, prefix="/api/v1/auth", tags=["auth"])
app.include_router(users.router, prefix="/api/v1/users", tags=["users"])
app.include_router(markets.router, prefix="/api/v1/markets", tags=["markets"])
app.include_router(trades.router, prefix="/api/v1/trades", tags=["trades"])
app.include_router(sentiment.router, prefix="/api/v1/sentiment", tags=["sentiment"])

@app.get("/")
def read_root():
    return {"message": "AIAlgoTradeHits API", "status": "running"}

@app.get("/health")
def health_check():
    return {"status": "healthy"}

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8080)
```

3.3 Key API Endpoints

python

```python
# File: backend/api/routes/markets.py

from fastapi import APIRouter, Depends, Query
from typing import List, Optional
from controllers.market_controller import MarketController
from middleware.auth_middleware import get_current_user

router = APIRouter()

@router.get("/crypto/daily")
async def get_crypto_daily_data(
    symbols: Optional[List[str]] = Query(None),
    days: int = Query(30, ge=1, le=365),
    current_user = Depends(get_current_user)
):
    """Get daily crypto data"""
    controller = MarketController()
    return await controller.get_crypto_daily_data(symbols, days)

@router.get("/stocks/daily")
async def get_stock_daily_data(
    symbols: Optional[List[str]] = Query(None),
    days: int = Query(30, ge=1, le=365),
    current_user = Depends(get_current_user)
):
    """Get daily stock data"""
    controller = MarketController()
    return await controller.get_stock_daily_data(symbols, days)

@router.get("/lowest-points")
async def get_assets_at_lowest(
    asset_type: str = Query(..., regex="^(crypto|stock)$"),
    current_user = Depends(get_current_user)
):
    """Get assets currently at lowest points"""
    controller = MarketController()
    return await controller.get_lowest_point_assets(asset_type)
```

3.4 BigQuery Service

python

```python
# File: backend/api/services/bigquery_service.py

from google.cloud import bigquery
from typing import List, Dict, Any
import pandas as pd

class BigQueryService:
    def __init__(self):
        self.client = bigquery.Client()
        self.project_id = "aialgo-tradehits"

    def query(self, sql: str) -> List[Dict[str, Any]]:
        """Execute query and return results"""
        query_job = self.client.query(sql)
        results = query_job.result()

        return [dict(row) for row in results]

    def get_crypto_daily_data(self, symbols: List[str], days: int):
        """Get daily crypto data"""
        symbols_str = "', '".join(symbols)

        query = f"""
        SELECT *
        FROM `{self.project_id}.crypto_data.daily_crypto_data`
        WHERE symbol IN ('{symbols_str}')
          AND date >= DATE_SUB(CURRENT_DATE(), INTERVAL {days} DAY)
        ORDER BY date DESC, symbol
        """

        return self.query(query)

    def get_lowest_point_detections(self, asset_type: str, limit: int = 10):
        """Get recent lowest point detections"""
        query = f"""
        SELECT *
        FROM `{self.project_id}.trading_data.lowest_point_detection`
        WHERE asset_type = '{asset_type}'
          AND is_lowest_point = TRUE
          AND detected_at >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1 HOUR)
        ORDER BY confidence_score DESC
        LIMIT {limit}
        """
```

```
        return self.query(query)

    def insert_trade(self, trade_data: Dict[str, Any]):
        """Insert trade record"""
        table_id = f"{self.project_id}.trading_data.executed_trades"

        errors = self.client.insert_rows_json(table_id, [trade_data])

        if errors:
            raise Exception(f"BigQuery insert error: {errors}")

        return True
```

3.5 Dockerfile for API

dockerfile

```
# File: backend/api/Dockerfile

FROM python:3.11-slim

WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose port
EXPOSE 8080

# Run the application
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

3.6 Deploy to Cloud Run

bash

```
# Build container
gcloud builds submit --tag gcr.io/aialgo-tradehits/api:latest ./backend/api

# Deploy to Cloud Run
gcloud run deploy aialgo-api \
  --image gcr.io/aialgo-tradehits/api:latest \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated \
  --set-env-vars="PROJECT_ID=aialgo-tradehits" \
  --set-secrets="JWT_SECRET=jwt-secret:latest"
```

Testing

```
bash

# Unit tests
cd backend/api
pytest tests/unit/

# Integration tests
pytest tests/integration/

# API endpoint testing
curl https://aialgo-api-xxxxx.run.app/health
curl -X POST https://aialgo-api-xxxxx.run.app/api/v1/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email": "test@example.com", "password": "password123"}'
```

Deliverables

- ✅ FastAPI application developed
- ✅ All endpoints implemented
- ✅ Authentication working
- ✅ Deployed to Cloud Run
- ✅ API documentation generated
- ✅ Tests passing

PHASE 4: Trading Algorithm Core (Weeks 8-9)

Objectives

- Implement lowest point detection algorithm
- Build sentiment analysis system
- Create trading decision engine
- Integrate Trump X monitoring

Tasks

4.1 Lowest Point Detection Algorithm

```python
```

```python
# File: backend/trading-engine/algorithms/lowest_point_detector.py

from typing import Dict, Tuple
from services.bigquery_service import BigQueryService
import pandas as pd

class LowestPointDetector:
    def __init__(self):
        self.bq = BigQueryService()

    def detect_lowest_point(
        self,
        symbol: str,
        asset_type: str
    ) -> Tuple[bool, float]:
        """
        Detect if asset is at lowest point
        Returns: (is_lowest, confidence_score)
        """

        # 1. Get recent price data
        price_data = self._get_price_data(symbol, asset_type)

        # 2. Price position analysis
        near_low = self._check_near_lows(price_data)
        at_support = self._check_support_level(price_data)

        # 3. Technical indicators
        oversold = self._check_oversold(price_data)
        ma_crossing = self._check_ma_crossover(price_data)

        # 4. Volume analysis
        volume_spike = self._check_volume_spike(price_data)

        # 5. Sentiment
        sentiment = self._get_sentiment(symbol, asset_type)

        # 6. Calculate confidence
        confidence = self._calculate_confidence({
            'near_low': near_low,
            'at_support': at_support,
            'oversold': oversold,
            'ma_crossing': ma_crossing,
```

```
        'volume_spike': volume_spike,
        'sentiment': sentiment
    })

    # 7. Decision
    is_lowest = (
        confidence >= 0.70 and
        (near_low['24h'] or near_low['7d']) and
        oversold and
        ma_crossing and
        sentiment > 0
    )

    # 8. Store detection
    self._store_detection(symbol, asset_type, is_lowest, confidence)

    return is_lowest, confidence

def _get_price_data(self, symbol: str, asset_type: str):
    """Get recent price data from BigQuery"""
    # Implementation
    pass

def _calculate_confidence(self, factors: Dict) -> float:
    """Calculate confidence score 0-1"""
    confidence = 0

    if factors['near_low']['24h']: confidence += 0.15
    if factors['near_low']['7d']: confidence += 0.10
    if factors['at_support']: confidence += 0.05

    if factors['oversold']['rsi']: confidence += 0.10
    if factors['oversold']['bb']: confidence += 0.05

    if factors['ma_crossing']['approaching']: confidence += 0.20

    if factors['volume_spike']: confidence += 0.10

    if factors['sentiment'] > 20: confidence += 0.15

    return min(confidence, 1.0)
```

4.2 Sentiment Analysis System

python

```python
# File: backend/trading-engine/sentiment/sentiment_analyzer.py

import requests
from typing import Dict, List
from google.cloud import bigquery

class SentimentAnalyzer:
    def __init__(self):
        self.bq = BigQueryService()

    def analyze_coinmarketcap_sentiment(self, symbol: str) -> float:
        """Fetch and analyze CoinMarketCap sentiment"""
        api_key = os.getenv('CMC_API_KEY')
        url = f"https://pro-api.coinmarketcap.com/v1/cryptocurrency/quotes/latest"

        params = {'symbol': symbol, 'convert': 'USD'}
        headers = {'X-CMC_PRO_API_KEY': api_key}

        response = requests.get(url, params=params, headers=headers)
        data = response.json()

        # Extract sentiment indicators
        sentiment_score = self._calculate_cmc_sentiment(data)

        # Store in BigQuery
        self._store_sentiment(symbol, 'coinmarketcap', sentiment_score)

        return sentiment_score

    def monitor_trump_posts(self):
        """Monitor Trump's X posts"""
        # Use Twitter API v2
        bearer_token = os.getenv('TWITTER_BEARER_TOKEN')

        # Get recent tweets from @realDonaldTrump
        tweets = self._fetch_recent_tweets('realDonaldTrump')

        for tweet in tweets:
            # AI analysis to extract mentioned assets
            mentioned_assets = self._extract_assets(tweet['text'])

            if mentioned_assets:
                sentiment = self._analyze_tweet_sentiment(tweet['text'])
```

```
            impact = self._predict_impact(tweet, mentioned_assets)

            # Store in BigQuery
            self._store_trump_announcement(tweet, sentiment, mentioned_assets)

    def get_aggregated_sentiment(
        self,
        symbol: str,
        asset_type: str
    ) -> Dict:
        """Get aggregated sentiment from all sources"""

        query = f"""
        SELECT
          AVG(sentiment_score) as avg_sentiment,
          COUNT(*) as total_mentions
        FROM `aialgo-tradehits.sentiment_data.{asset_type}_sentiment`
        WHERE symbol = '{symbol}'
          AND timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 24 HOUR)
        """

        result = self.bq.query(query)

        return result[0] if result else {'avg_sentiment': 0, 'total_mentions': 0}
```

4.3 Trading Decision Engine

python

```python
# File: backend/trading-engine/trading_engine.py

from algorithms.lowest_point_detector import LowestPointDetector
from sentiment.sentiment_analyzer import SentimentAnalyzer
from services.bigquery_service import BigQueryService
import time

class TradingEngine:
    def __init__(self):
        self.detector = LowestPointDetector()
        self.sentiment = SentimentAnalyzer()
        self.bq = BigQueryService()

    def run_crypto_engine(self):
        """Main crypto trading loop - runs 24/7"""
        while True:
            try:
                self._process_crypto_trades()
                time.sleep(300)  # 5 minutes
            except Exception as e:
                print(f"Error in crypto engine: {e}")
                time.sleep(60)

    def run_stock_engine(self):
        """Main stock trading loop - market hours only"""
        while True:
            try:
                if self._is_market_open():
                    self._process_stock_trades()
                    time.sleep(300)  # 5 minutes
                else:
                    time.sleep(3600)  # 1 hour
            except Exception as e:
                print(f"Error in stock engine: {e}")
                time.sleep(60)

    def _process_crypto_trades(self):
        """Process crypto trading for all users"""
        active_users = self._get_active_users('crypto')

        for user in active_users:
            params = self._get_user_params(user['user_id'], 'crypto')
            top_cryptos = self._get_top_cryptos_near_lowest()
```

```
        for crypto in top_cryptos:
            # Check if should buy
            should_buy = self._evaluate_buy_signal(
                user_id=user['user_id'],
                symbol=crypto['symbol'],
                asset_type='crypto',
                params=params
            )

            if should_buy:
                self._execute_buy(user['user_id'], crypto['symbol'], 'crypto', params)

            # Check existing positions
            self._check_exit_conditions(user['user_id'], crypto['symbol'], 'crypto')

def _evaluate_buy_signal(
    self,
    user_id: str,
    symbol: str,
    asset_type: str,
    params: Dict
) -> bool:
    """Evaluate if should execute buy"""

    # 1. Check lowest point
    is_lowest, confidence = self.detector.detect_lowest_point(symbol, asset_type)

    if not is_lowest or confidence < 0.70:
        return False

    # 2. Check sentiment
    sentiment_data = self.sentiment.get_aggregated_sentiment(symbol, asset_type)

    if sentiment_data['avg_sentiment'] < params['min_sentiment_score']:
        return False

    # 3. Check Trump sentiment if required
    if params['require_trump_positive']:
        trump_sentiment = self._get_trump_sentiment(symbol)
        if trump_sentiment is not None and trump_sentiment < 0:
            return False

    # 4. Check max concurrent trades
```

```
        open_positions = self._count_open_positions(user_id, symbol)
        if open_positions >= params['max_concurrent_trades']:
            return False

        return True
```

Testing

python

```
# File: tests/test_trading_engine.py

import pytest
from trading_engine import TradingEngine

def test_lowest_point_detection():
    engine = TradingEngine()
    is_lowest, confidence = engine.detector.detect_lowest_point('BTC', 'crypto')

    assert isinstance(is_lowest, bool)
    assert 0 <= confidence <= 1

def test_sentiment_analysis():
    analyzer = SentimentAnalyzer()
    sentiment = analyzer.get_aggregated_sentiment('BTC', 'crypto')

    assert 'avg_sentiment' in sentiment
    assert 'total_mentions' in sentiment

@pytest.mark.integration
def test_buy_signal_evaluation():
    engine = TradingEngine()
    params = {
        'min_sentiment_score': 0,
        'require_trump_positive': False,
        'max_concurrent_trades': 3
    }

    should_buy = engine._evaluate_buy_signal('test_user', 'BTC', 'crypto', params)
    assert isinstance(should_buy, bool)
```

Deliverables

- ✅ Lowest point detector implemented
- ✅ Sentiment analyzer working
- ✅ Trading engine core complete
- ✅ Trump monitoring active
- ✅ Unit tests passing
- ✅ Integration tests passing

PHASE 5: Frontend Development (Weeks 10-12)

Objectives

- Build React frontend
- Implement dual-tab system
- Create real-time updates
- Deploy to Cloud Run

Tasks

5.1 Frontend Structure

```
frontend/
├── src/
│   ├── components/
│   │   ├── layout/
│   │   │   ├── Header.tsx
│   │   │   ├── Sidebar.tsx
│   │   │   └── MainContent.tsx
│   │   ├── charts/
│   │   │   ├── CandlestickChart.tsx
│   │   │   ├── MultiTimeframeChart.tsx
│   │   │   └── SentimentChart.tsx
│   │   ├── crypto/
│   │   │   ├── CryptoTab.tsx
│   │   │   └── CryptoCard.tsx
│   │   ├── stocks/
│   │   │   ├── StockTab.tsx
```

```
│   │   │   └── StockCard.tsx
│   │   ├── trading/
│   │   │   ├── TradeSetup.tsx
│   │   │   ├── ActivePositions.tsx
│   │   │   └── LowestPointsPanel.tsx
│   │   └── sentiment/
│   │       ├── SentimentPanel.tsx
│   │       └── TrumpPostCard.tsx
│   ├── pages/
│   │   ├── Dashboard.tsx
│   │   ├── Login.tsx
│   │   ├── Register.tsx
│   │   └── Settings.tsx
│   ├── services/
│   │   ├── api.ts
│   │   ├── websocket.ts
│   │   └── auth.ts
│   ├── hooks/
│   │   ├── useMarketData.ts
│   │   ├── useTrades.ts
│   │   └── useSentiment.ts
│   ├── store/
│   │   ├── authSlice.ts
│   │   ├── marketSlice.ts
│   │   └── tradeSlice.ts
│   ├── utils/
│   └── App.tsx
├── public/
├── package.json
└── Dockerfile
```

5.2 Key Components (Using Claude for Development)

Instructions for Claude in VSCode:

Claude, please create the following React components based on the mockup:

1. Dashboard.tsx - Main dashboard with dual tabs
2. CryptoTab.tsx - Cryptocurrency tab with all features
3. StockTab.tsx - Stock tab with market hours indicator
4. SentimentPanel.tsx - Sentiment analysis display with Trump posts
5. TradeSetup.tsx - Right sidebar trade configuration
6. MultiTimeframeChart.tsx - Three charts side by side

Use TypeScript, TailwindCSS, and follow these guidelines:
- Real-time data updates via WebSocket
- Responsive design
- Error handling
- Loading states
- TypeScript interfaces for all data

5.3 API Service

typescript

```
// File: frontend/src/services/api.ts

import axios from 'axios';

const API_BASE_URL = process.env.REACT_APP_API_URL || 'https://aialgo-api-xxxxx.run.app';

export const api = axios.create({
  baseURL: `${API_BASE_URL}/api/v1`,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Add auth token to requests
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

export const marketAPI = {
  getCryptoDailyData: (symbols: string[], days: number) =>
    api.get('/markets/crypto/daily', { params: { symbols, days } }),

  getStockDailyData: (symbols: string[], days: number) =>
    api.get('/markets/stock/daily', { params: { symbols, days } }),

  getLowestPoints: (assetType: 'crypto' | 'stock') =>
    api.get('/markets/lowest-points', { params: { asset_type: assetType } }),
};

export const tradeAPI = {
  getActiveTrades: (userId: string) =>
    api.get(`/trades/active/${userId}`),

  executeTrade: (tradeData: any) =>
    api.post('/trades/execute', tradeData),

  getTradeHistory: (userId: string, limit: number) =>
    api.get(`/trades/history/${userId}`, { params: { limit } }),
};
```

```
export const sentimentAPI = {
  getAggregatedSentiment: (symbol: string, assetType: string) =>
    api.get('/sentiment/aggregated', { params: { symbol, asset_type: assetType } }),

  getTrumpPosts: (hours: number) =>
    api.get('/sentiment/trump-posts', { params: { hours } }),
};
```

5.4 Build and Deploy Frontend

dockerfile

```
# File: frontend/Dockerfile

FROM node:18-alpine as build

WORKDIR /app

COPY package*.json ./
RUN npm ci

COPY . .
RUN npm run build

FROM nginx:alpine

COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

bash

```
# Build and deploy
gcloud builds submit --tag gcr.io/aialgo-tradehits/frontend:latest ./frontend

gcloud run deploy aialgo-frontend \
  --image gcr.io/aialgo-tradehits/frontend:latest \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated
```

Testing

```
bash

# Run unit tests
cd frontend
npm test

# Run E2E tests
npm run test:e2e

# Visual regression tests
npm run test:visual
```

Deliverables

- ✅ All React components built
- ✅ Dual-tab system working
- ✅ Real-time updates functional
- ✅ Deployed to Cloud Run
- ✅ Tests passing

PHASE 6: Integration & Automated Testing (Weeks 13-14)

Objectives

- Integrate all components
- Implement comprehensive testing
- Set up CI/CD pipeline

- Performance testing

Tasks

6.1 Integration Testing

```python
```

```python
# File: tests/integration/test_end_to_end.py

import pytest
import requests
from google.cloud import bigquery

class TestEndToEnd:
    def setup_method(self):
        self.api_url = "https://aialgo-api-xxxxx.run.app"
        self.bq = bigquery.Client()

    def test_data_flow_crypto(self):
        """Test complete crypto data flow"""
        # 1. Trigger scheduler job
        response = requests.post(
            "https://us-central1-aialgo-tradehits.cloudfunctions.net/minute5-crypto-job"
        )
        assert response.status_code == 200

        # 2. Wait for data to be stored
        time.sleep(10)

        # 3. Verify data in BigQuery
        query = """
        SELECT COUNT(*) as count
        FROM `aialgo-tradehits.crypto_data.minute5_crypto_data`
        WHERE created_at >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1 HOUR)
        """
        result = self.bq.query(query).result()
        count = list(result)[0]['count']
        assert count > 0

        # 4. Verify API returns data
        response = requests.get(
            f"{self.api_url}/api/v1/markets/crypto/daily",
            params={"symbols": ["BTC"], "days": 1}
        )
        assert response.status_code == 200
        assert len(response.json()) > 0

    def test_trading_engine_flow(self):
        """Test complete trading flow"""
        # 1. Create test user
```

```
    # 2. Set trade parameters
    # 3. Trigger lowest point detection
    # 4. Verify trade execution
    # 5. Check BigQuery records
    pass

def test_sentiment_integration(self):
    """Test sentiment analysis integration"""
    # 1. Fetch CoinMarketCap data
    # 2. Analyze Trump posts
    # 3. Aggregate sentiment
    # 4. Verify API returns sentiment
    pass
```

6.2 CI/CD Pipeline

yaml

```yaml
# File: .github/workflows/deploy.yml

name: Deploy to GCP

on:
  push:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.11'

      - name: Install dependencies
        run: |
          pip install -r backend/requirements.txt
          pip install pytest pytest-cov

      - name: Run tests
        run: |
          pytest backend/tests/ --cov=backend

      - name: Run integration tests
        run: |
          pytest tests/integration/

  deploy-backend:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - uses: google-github-actions/setup-gcloud@v0
        with:
          service_account_key: ${{ secrets.GCP_SA_KEY }}
          project_id: aialgo-tradehits
```

```
    - name: Build and push
      run: |
        gcloud builds submit --tag gcr.io/aialgo-tradehits/api:latest ./backend/api

    - name: Deploy to Cloud Run
      run: |
        gcloud run deploy aialgo-api \
          --image gcr.io/aialgo-tradehits/api:latest \
          --platform managed \
          --region us-central1

  deploy-frontend:
    needs: test
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    - name: Build and deploy frontend
      run: |
        gcloud builds submit --tag gcr.io/aialgo-tradehits/frontend:latest ./frontend
        gcloud run deploy aialgo-frontend \
          --image gcr.io/aialgo-tradehits/frontend:latest \
          --platform managed \
          --region us-central1
```

6.3 Load Testing

python

```python
# File: tests/load/locustfile.py

from locust import HttpUser, task, between

class TradingUser(HttpUser):
    wait_time = between(1, 3)

    def on_start(self):
        """Login"""
        response = self.client.post("/api/v1/auth/login", json={
            "email": "test@example.com",
            "password": "test123"
        })
        self.token = response.json()['token']
        self.client.headers = {"Authorization": f"Bearer {self.token}"}

    @task(3)
    def get_crypto_data(self):
        self.client.get("/api/v1/markets/crypto/daily?symbols=BTC,ETH&days=30")

    @task(2)
    def get_sentiment(self):
        self.client.get("/api/v1/sentiment/aggregated?symbol=BTC&asset_type=crypto")

    @task(1)
    def get_active_trades(self):
        self.client.get("/api/v1/trades/active/test_user")
```

bash

```
# Run load test
locust -f tests/load/locustfile.py --host=https://aialgo-api-xxxxx.run.app
```

Deliverables

- ✅ Integration tests passing
- ✅ CI/CD pipeline configured
- ✅ Load tests completed
- ✅ Performance benchmarks documented

PHASE 7: Paper Trading Phase (Weeks 15-16)

Objectives

- Enable paper trading mode
- Validate trading algorithms
- Collect performance metrics
- Identify and fix issues

Tasks

7.1 Enable Paper Trading Mode

python

```python
# File: backend/trading-engine/paper_trading_engine.py

class PaperTradingEngine(TradingEngine):
    """Paper trading engine - no real money"""

    def __init__(self):
        super().__init__()
        self.is_paper_trading = True
        self.paper_balance = 100000.00  # $100k virtual money

    def _execute_buy(self, user_id, symbol, asset_type, params):
        """Execute virtual buy"""
        # Get current price
        current_price = self._get_current_price(symbol, asset_type)

        # Calculate quantity
        quantity = params['trade_amount'] / current_price

        # Create paper trade record
        trade_data = {
            'trade_id': self._generate_trade_id(),
            'user_id': user_id,
            'asset_type': asset_type,
            'symbol': symbol,
            'trade_type': 'BUY',
            'entry_price': current_price,
            'quantity': quantity,
            'trade_amount': params['trade_amount'],
            'take_profit_target': current_price * (1 + params['take_profit_percentage'] / 100),
            'stop_loss_target': current_price * (1 - params['stop_loss_percentage'] / 100),
            'executed_at': datetime.utcnow().isoformat(),
            'status': 'open',
            'is_paper_trade': True
        }

        # Store in paper trading table
        self.bq.insert_trade(trade_data, table='paper_trading_data.paper_trades')

        print(f"[PAPER] Bought {quantity} {symbol} at ${current_price}")

    def _execute_sell(self, trade, reason):
        """Execute virtual sell"""
        current_price = self._get_current_price(trade['symbol'], trade['asset_type'])
```

```
        # Calculate P&L
        profit_loss = (current_price - trade['entry_price']) * trade['quantity']
        profit_loss_percentage = (profit_loss / trade['trade_amount']) * 100

        # Update trade record
        self.bq.update_trade(trade['trade_id'], {
            'exit_price': current_price,
            'profit_loss': profit_loss,
            'profit_loss_percentage': profit_loss_percentage,
            'execution_reason': reason,
            'closed_at': datetime.utcnow().isoformat(),
            'status': 'closed'
        }, table='paper_trading_data.paper_trades')

        print(f"[PAPER] Sold {trade['quantity']} {trade['symbol']} at ${current_price} - P&L: ${profit_loss:.2f}")
```

7.2 Paper Trading Dashboard

Add to frontend:

typescript

```
// File: frontend/src/components/PaperTradingBanner.tsx

export const PaperTradingBanner = () => {
  return (
    <div className="bg-yellow-900 border border-yellow-700 p-4 mb-4">
      <div className="flex items-center gap-2">
        <AlertCircle className="w-5 h-5 text-yellow-400" />
        <div>
          <h3 className="font-bold text-yellow-400">Paper Trading Mode Active</h3>
          <p className="text-sm text-yellow-200">
            Trading with virtual money - No real funds at risk
          </p>
        </div>
      </div>
    </div>
  );
};
```

7.3 Monitoring and Metrics

bash

```
# Create monitoring dashboard
gcloud monitoring dashboards create --config-from-file=monitoring/paper-trading-dashboard.yaml
```

yaml

```
# File: monitoring/paper-trading-dashboard.yaml

displayName: "Paper Trading Performance"
mosaicLayout:
  columns: 12
  tiles:
    - width: 6
      height: 4
      widget:
        title: "Total Trades Executed"
        scorecard:
          timeSeriesQuery:
            timeSeriesFilter:
              filter: 'resource.type="cloud_run_revision"'
    - width: 6
      height: 4
      widget:
        title: "Win Rate"
        scorecard:
          timeSeriesQuery:
            timeSeriesFilter:
              filter: 'metric.type="custom.googleapis.com/trading/win_rate"'
    - width: 12
      height: 4
      widget:
        title: "Cumulative P&L"
        xyChart:
          dataSets:
            - timeSeriesQuery:
                timeSeriesFilter:
                  filter: 'metric.type="custom.googleapis.com/trading/cumulative_pnl"'
```

7.4 Daily Paper Trading Reports

```python
```

```python
# File: backend/schedulers/generate_paper_trading_report.py

def generate_daily_report():
    """Generate daily paper trading performance report"""

    query = """
    SELECT
      COUNT(*) as total_trades,
      SUM(CASE WHEN profit_loss > 0 THEN 1 ELSE 0 END) as winning_trades,
      SUM(profit_loss) as total_pnl,
      AVG(profit_loss) as avg_pnl,
      MAX(profit_loss) as best_trade,
      MIN(profit_loss) as worst_trade
    FROM `aialgo-tradehits.paper_trading_data.paper_trades`
    WHERE DATE(executed_at) = CURRENT_DATE()
      AND status = 'closed'
    """

    result = bq.query(query)[0]

    win_rate = (result['winning_trades'] / result['total_trades']) * 100 if result['total_trades'] > 0 else 0

    report = f"""
    Paper Trading Daily Report - {datetime.now().date()}
    ================================================

    Total Trades: {result['total_trades']}
    Winning Trades: {result['winning_trades']}
    Win Rate: {win_rate:.2f}%

    Total P&L: ${result['total_pnl']:.2f}
    Average P&L: ${result['avg_pnl']:.2f}
    Best Trade: ${result['best_trade']:.2f}
    Worst Trade: ${result['worst_trade']:.2f}
    """

    # Send via email
    send_email(to="admin@aialgotradetits.com", subject="Daily Paper Trading Report", body=report)

    return report
```

7.5 Paper Trading Validation Checklist

- [ ] Lowest point detection working correctly
- [ ] Sentiment analysis influencing decisions
- [ ] Trump posts triggering appropriate actions
- [ ] TP/SL targets executing properly
- [ ] Market hours respected for stocks
- [ ] No weekend stock trading
- [ ] Data accuracy verified
- [ ] Performance metrics acceptable (>60% win rate)
- [ ] No critical bugs identified

Duration: 2 weeks

Success Criteria:

- 500+ paper trades executed
- Win rate > 60%
- No critical bugs
- Positive overall P&L
- All features functioning

Deliverables

- ✅ Paper trading mode active
- ✅ 2 weeks of trading data collected
- ✅ Performance reports generated
- ✅ Issues identified and fixed
- ✅ Validation checklist complete

PHASE 8: Small Money Testing (Weeks 17-18)

Objectives

- Enable real money trading with small amounts
- Validate with actual market conditions

- Monitor closely for issues
- Build confidence in system

Tasks

8.1 Enable Real Money Mode

```python
# Configuration change
TRADING_MODE = 'real_money'  # Changed from 'paper'
SMALL_MONEY_TESTING = True
MAX_TRADE_AMOUNT = 50.00  # $50 max per trade during testing
DAILY_LOSS_LIMIT = 100.00  # Stop trading if daily loss exceeds $100
```

8.2 Safety Mechanisms

```python
```

```python
# File: backend/trading-engine/safety_mechanisms.py

class SafetyMechanisms:
    def __init__(self):
        self.bq = BigQueryService()

    def check_daily_loss_limit(self, user_id):
        """Check if user exceeded daily loss limit"""
        query = f"""
        SELECT SUM(profit_loss) as daily_pnl
        FROM `aialgo-tradehits.trading_data.executed_trades`
        WHERE user_id = '{user_id}'
          AND DATE(executed_at) = CURRENT_DATE()
          AND is_paper_trade = FALSE
        """

        result = self.bq.query(query)
        daily_pnl = result[0]['daily_pnl'] if result else 0

        if daily_pnl < -100.00:
            self._pause_trading(user_id, reason='daily_loss_limit_exceeded')
            return False

        return True

    def validate_trade_amount(self, amount):
        """Ensure trade amount within limits"""
        if amount > 50.00:
            raise ValueError(f"Trade amount ${amount} exceeds limit of $50 during small money testing")

        return True

    def circuit_breaker(self, symbol, asset_type):
        """Circuit breaker for abnormal market conditions"""
        # Check for flash crashes, extreme volatility
        volatility = self._calculate_recent_volatility(symbol, asset_type)

        if volatility > 20:  # 20% volatility threshold
            print(f"Circuit breaker triggered for {symbol} - volatility: {volatility}%")
            return False

        return True
```

8.3 Real-Time Monitoring

```bash
# Set up real-time alerts
gcloud alpha monitoring policies create \
  --notification-channels=EMAIL_CHANNEL \
  --display-name="Real Money Trade Alert" \
  --condition-display-name="Trade Executed" \
  --condition-threshold-value=1 \
  --condition-threshold-duration=60s

# Alert for losses
gcloud alpha monitoring policies create \
  --notification-channels=EMAIL_CHANNEL,SMS_CHANNEL \
  --display-name="Loss Alert" \
  --condition-display-name="Loss exceeds $50" \
  --condition-threshold-value=-50
```

8.4 Manual Review Process

```python
# All trades require admin review for first week
REQUIRE_ADMIN_APPROVAL = True

def execute_trade_with_approval(trade_data):
    """Execute trade after admin approval"""
    # Store pending trade
    pending_trades.append(trade_data)

    # Send notification to admin
    send_notification(
        to="admin@aialgotradetits.com",
        subject="Trade Approval Required",
        body=f"Trade: {trade_data['symbol']} - ${trade_data['trade_amount']}"
    )

    # Wait for approval
    # Admin approves via dashboard
```

8.5 Testing Checklist

Week 1 (Manual Approval):

- [ ] Execute 5-10 trades per day
- [ ] Review each trade before execution
- [ ] Monitor continuously
- [ ] Document any issues
- [ ] Daily P&L review

Week 2 (Automated):

- [ ] Remove manual approval
- [ ] Monitor automated trades
- [ ] Verify safety mechanisms
- [ ] Track performance
- [ ] Compare with paper trading results

Duration: 2 weeks

Success Criteria:

- 100+ real money trades executed
- Win rate similar to paper trading
- No system errors causing losses
- Safety mechanisms working
- Positive overall P&L
- User experience smooth

Deliverables

- ✅ Real money trading enabled
- ✅ Safety mechanisms tested
- ✅ Performance data collected
- ✅ System stability verified
- ✅ Ready for beta users

PHASE 9: Beta User Testing (Weeks 19-20)

Objectives

- Onboard limited user group (10-20 users)
- Gather user feedback
- Identify UX issues
- Validate at scale

Tasks

9.1 Beta User Selection

Criteria for beta users:

- Trading experience
- Willingness to provide feedback
- Comfortable with new platforms
- Can invest $500-$1000

9.2 Onboarding Process

```python
# Create beta user accounts
beta_users = [
    {"email": "user1@example.com", "name": "John Doe", "initial_balance": 1000.00},
    {"email": "user2@example.com", "name": "Jane Smith", "initial_balance": 500.00},
    # ... 10-20 users
]

for user in beta_users:
    create_user_account(
        email=user['email'],
        name=user['name'],
        role='beta_user',
        crypto_balance=user['initial_balance'] / 2,
        stock_balance=user['initial_balance'] / 2
    )

    send_welcome_email(user['email'])
```

9.3 Beta User Dashboard

Add features:

- Feedback form
- Bug reporting
- Feature requests
- Performance tracking
- Communication channel

9.4 Monitoring Beta Users

python

```python
# Daily beta user report
def generate_beta_report():
    """Generate daily beta user activity report"""

    query = """
    SELECT
      u.user_id,
      u.username,
      COUNT(t.trade_id) as total_trades,
      SUM(t.profit_loss) as total_pnl,
      u.total_balance
    FROM `aialgo-tradehits.user_data.users` u
    LEFT JOIN `aialgo-tradehits.trading_data.executed_trades` t
      ON u.user_id = t.user_id
      AND DATE(t.executed_at) = CURRENT_DATE()
    WHERE u.role = 'beta_user'
    GROUP BY u.user_id, u.username, u.total_balance
    """

    results = bq.query(query)

    # Generate report
    # Send to admin
```

9.5 Feedback Collection

typescript

```tsx
// File: frontend/src/components/FeedbackForm.tsx

export const FeedbackForm = () => {
  const [feedback, setFeedback] = useState('');
  const [rating, setRating] = useState(5);

  const submitFeedback = async () => {
    await api.post('/feedback', {
      user_id: currentUser.id,
      feedback: feedback,
      rating: rating,
      timestamp: new Date().toISOString()
    });

    toast.success('Thank you for your feedback!');
  };

  return (
    <div className="p-4 bg-slate-800 rounded-lg">
      <h3>How's your experience?</h3>
      <textarea
        value={feedback}
        onChange={(e) => setFeedback(e.target.value)}
        placeholder="Tell us what you think..."
        className="w-full p-2 bg-slate-700 rounded"
      />
      <button onClick={submitFeedback} className="mt-2 btn-primary">
        Submit Feedback
      </button>
    </div>
  );
};
```

9.6 Weekly Beta Review Meetings

Schedule:

- Week 1: Initial feedback session
- Week 2: Progress review
- End of week 2: Final assessment

Discussion topics:

- User experience
- Performance results
- Issues encountered
- Feature requests
- Decision to proceed to full launch

Duration: 2 weeks

Success Criteria:

- 10+ beta users active
- Positive feedback (> 4/5 average rating)
- < 5 critical bugs reported
- Win rate maintained
- Users willing to continue

Deliverables

- ✅ Beta users onboarded
- ✅ Feedback collected
- ✅ Issues resolved
- ✅ Platform validated
- ✅ Ready for full launch

PHASE 10: Production Launch (Week 21)

Objectives

- Public launch of AIAlgoTradeHits.com
- Marketing campaign
- Customer support setup
- Scale infrastructure

Tasks

10.1 Pre-Launch Checklist

- [ ] All features tested and working
- [ ] Performance optimized
- [ ] Security audit completed
- [ ] Legal compliance verified
- [ ] Terms of service finalized
- [ ] Privacy policy published
- [ ] Customer support trained
- [ ] Marketing materials ready
- [ ] Domain configured (aialgotradetits.com)
- [ ] SSL certificates active
- [ ] Monitoring dashboards set up
- [ ] Backup systems tested
- [ ] Disaster recovery plan documented

10.2 Launch Day Tasks

```bash
```

```
# 1. Final database backup
bq extract \
  --destination_format=PARQUET \
  'aialgo-tradehits:*' \
  gs://aialgo-tradehits-backups/pre-launch-backup/

# 2. Scale up Cloud Run
gcloud run services update aialgo-api \
  --min-instances=3 \
  --max-instances=100 \
  --cpu=2 \
  --memory=2Gi

gcloud run services update aialgo-frontend \
  --min-instances=2 \
  --max-instances=50

# 3. Enable CDN
gcloud compute backend-services update aialgo-backend \
  --enable-cdn

# 4. Set up auto-scaling
gcloud run services update aialgo-api \
  --concurrency=80 \
  --cpu-throttling

# 5. Remove beta restrictions
# Update database: role='beta_user' -> role='user'
```

10.3 Marketing Launch

- Press release
- Social media campaign
- Email to waitlist
- Influencer partnerships
- Content marketing (blog, videos)
- Paid advertising (Google, Facebook)

10.4 Customer Support

- Live chat integration
- Email support (support@aialgotradetits.com)
- FAQ/Knowledge base
- Video tutorials
- Community forum

10.5 Post-Launch Monitoring

```
bash

# Monitor critical metrics
# - User signups
# - Trading volume
# - System performance
# - Error rates
# - Customer satisfaction

# Daily launch report
python scripts/generate_launch_report.py
```

10.6 Gradual Rollout

Day 1-3: Soft launch (limited advertising)
Day 4-7: Increase marketing
Week 2: Full marketing push
Week 3+: Optimize and scale

Deliverables

- ✅ Platform launched publicly
- ✅ Marketing campaign active
- ✅ Users signing up
- ✅ Trades executing smoothly
- ✅ Support system operational
- ✅ Monitoring active

Post-Launch: Continuous Improvement

Ongoing Tasks

Daily

- Monitor system health
- Review trading performance
- Check error logs
- Respond to support tickets

Weekly

- Generate performance reports
- Review user feedback
- Plan feature updates
- Optimize algorithms

Monthly

- Security audits
- Performance optimization
- Feature releases
- Marketing analysis

Success Metrics

Key Performance Indicators (KPIs)

Technical Metrics

- System uptime: > 99.9%
- API response time: < 200ms (p95)
- Trade execution time: < 5 seconds
- Data freshness: < 5 minutes delay

Trading Metrics

- Win rate: > 60%
- Average P&L per trade: > $5
- Lowest point detection accuracy: > 70%
- Sentiment prediction accuracy: > 65%

Business Metrics

- Active users: 100+ in first month
- Trading volume: $1M+ in first month
- User retention: > 70% after 30 days
- Customer satisfaction: > 4/5 rating

Risk Mitigation

Identified Risks & Mitigation Strategies

Risk	Impact	Probability	Mitigation
API rate limits	High	Medium	Implement caching, backup APIs
Market volatility	High	High	Circuit breakers, loss limits
Data quality issues	Medium	Low	Validation, monitoring
Security breach	Very High	Low	Security audits, encryption
Regulatory issues	High	Medium	Legal review, compliance
System downtime	High	Low	Redundancy, auto-scaling
User fund loss	Very High	Low	Safety mechanisms, insurance

Budget Estimate

Monthly GCP Costs (Estimated)

Service	Monthly Cost
Cloud Run (API)	$200
Cloud Run (Frontend)	$100

Service	Monthly Cost
BigQuery (storage + queries)	$300
Cloud Functions (6 schedulers)	$50
Cloud Scheduler	$10
Cloud Storage (backups)	$20
Cloud Monitoring	$50
Cloud Load Balancing	$30
Total	**~$760/month**

Additional costs:

- Domain: $12/year
- SSL: Free (Google-managed)
- APIs (Kraken, CMC, Twitter): Variable
- Support tools: $50-100/month

Conclusion

This implementation plan provides a structured, phased approach to building and launching AIAlgoTradeHits.com. By following these phases sequentially and ensuring each phase's deliverables are met before proceeding, you'll build a robust, tested, and scalable automated trading platform.

Key Success Factors:

1. Thorough testing at each phase
2. Paper trading validation before real money
3. Small money testing before scaling
4. Beta user feedback integration
5. Continuous monitoring and optimization

Timeline Summary:

- Weeks 1-2: Setup & Database
- Weeks 3-4: Data Collection
- Weeks 5-7: Backend API

- Weeks 8-9: Trading Algorithms
- Weeks 10-12: Frontend
- Weeks 13-14: Integration & Testing
- Weeks 15-16: Paper Trading
- Weeks 17-18: Small Money Testing
- Weeks 19-20: Beta Users
- Week 21: Launch

Total: **21 weeks (5 months)** from start to launch.

Good luck with your implementation! 🚀