IT643

Software Design and Testing

Faculty Name : Ankush Sir

Project Name: CryptoCortex

Group Name : Pixel Pioneers

Team Members

202412026: Tisha Jain

202412029: Preksha Joshi

202412032: Kaival Shah

202412066: Dhruv Patel

Project Testing Documentation

1. Introduction

This document describes the testing strategy, methodologies, tools, and procedures used to validate the functionality, reliability, and performance of the Crypto Trading Platform. The project includes:

- FastAPI backend
- Celery background worker
- Redis message broker
- MongoDB database (Beanie ODM)
- Frontend (React)
- Binance WebSocket integration
- Portfolio, credits, and trading operations
- Chatbot-based trading

Testing ensures that all components work together as expected and that the system remains stable during real-world usage.

2. Testing Scope

Testing covers the following modules:

2.1 Backend (FastAPI)

- Authentication
- Order placement
- Portfolio operations
- Credit handling
- Transaction history
- Chatbot command parsing

2.2 Celery Worker (Trade Execution)

- Task queuing
- Order execution flow
- Database consistency
- Error handling & retry

2.3 Database Layer (MongoDB + Beanie)

- CRUD operations
- Atomic portfolio updates
- Index enforcement

2.4 Redis Broker

- Task dispatch
- Task completion

2.5 WebSocket Streaming

- Binance price feed stability
- Client broadcasting

2.6 Frontend

- UI rendering
- Data fetching
- Error handling
- User interaction flows

3. Testing Types

3.1 Unit Testing

Unit tests focus on individual, isolated functions.

Modules tested:

- Portfolio math (avg price, quantity updates)
- Credit calculation and fee deduction
- Command parsing (`parse_trade_command`)
- Decimal utilities (`quantize_decimal`, `to_decimal128`)

Tools:

- pytest
- pytest-asyncio

3.2 Integration Testing

Tests how systems interact.

Cases include:

- FastAPI endpoint → Celery task is queued
- Celery worker → MongoDB database update
- Order placed → Transaction + CreditsHistory created
- Portfolio adjustments for BUY/SELL
- Atomic updates prevent duplicate keys

Tools:

- pytest
- httpx (async client)
- MongoDB test database

3.3 End-to-End (E2E) Testing

Simulates the full user workflow.

Scenarios:

1. User places a BUY market order
2. Celery worker executes trade
3. DB updates:
    - Order created
    - Transaction added
    - Portfolio updated
    - Credits updated
4. API returns results to frontend

E2E also includes chatbot trading:

- “Buy 2 XRPUSDT at market price”
- “Sell 1 BTCUSDT at limit price 30000”

Tools:

- Postman

3.4 Security Testing

Ensures the system is safe from misuse.

Scenarios:

- Token authentication validation
- Expired tokens
- Session invalidation
- Unauthorized order attempts
- Rate limiting bypass attempts

4. Test Strategy

4.1 Backend Testing Strategy

- Mock Binance API responses
- Test logic independently from external services
- Validate all API inputs using Pydantic models

- Ensure invalid trade commands are rejected

4.2 Celery Worker Testing Strategy

- Use Celery “always eager” mode for synchronous tests
- Mock Redis and MongoDB during tests
- Validate entire trade execution pipeline
- Test worker retry behavior for failures

4.3 Database Testing Strategy

- Spin up a temporary MongoDB instance
- Use unique collections per test file
- Test index constraints (unique user + symbol)
- Validate Decimal128 precision when saving numeric fields

4.4 WebSocket Testing Strategy

- Mock WebSocket server
- Simulate Binance price stream messages
- Test client subscription & disconnect

5. Test Cases Summary

5.1 API Test Cases

Test Case	Description	Expected Result
Place BUY order	Valid order	Task queued, 200 OK
Place SELL order	Valid sell	Task queued
Invalid symbol	Invalid request	400 Error
Not enough credits	BUY order fails	Error returned
Chatbot trade command	NLP-based command	Creates OrderRequest

5.2 Worker Test Cases

Scenario	Expected
Market BUY execution	Order FILLED, portfolio + credits updated
Limit BUY at in-fillable price	Converted to MARKET
Limit BUY not fillable	Order stays OPEN
SELL with insufficient quantity	Worker rejects
Multiple trades on same symbol	Atomic portfolio update
Redis failure	Worker retries

5.3 Portfolio Test Cases

Operation	Expected
First BUY	Creates portfolio entry
Additional BUY	Updates avg price / quantity
SELL part	Decreases quantity
SELL all	Deletes entry
Concurrent trades	No duplicate key errors

6. Test Environment Setup

6.1 Local Environment

- Python venv
- Redis local instance
- MongoDB local instance
- Celery worker in eager mode
- Mock Binance API

7. Conclusions

Testing validates that the redesigned architecture:

- Handles trades reliably
- Maintains consistent database state
- Runs stably across API + Worker + Redis + MongoDB
- Supports real-world concurrency
- Provides a deployable cloud-ready solution