Table of Contents

[Folder PATH listing: 2](#_Toc204040068)

[quotex\_logic.py 5](#_Toc204040069)

[main.py 10](#_Toc204040070)

[ai\_engine.py 15](#_Toc204040071)

[quotex\_price\_browser.py 18](#_Toc204040072)

[strategy\_crypto.py 20](#_Toc204040073)

[utils.py 26](#_Toc204040074)

[index.html 29](#_Toc204040075)

[style.css 41](#_Toc204040076)

[test\_pyquotex.py 46](#_Toc204040077)

[test\_qx\_login.py 47](#_Toc204040078)

[run.py 48](#_Toc204040079)

[.env 49](#_Toc204040080)

[ai\_analyzer.py 50](#_Toc204040081)

[chart\_capture.py 51](#_Toc204040082)

[pattern\_detector.py 57](#_Toc204040083)

# Folder PATH listing:

D:.

│ .env

│ ai\_analyzer.py

│ chart.png

│ chart\_capture.py

│ pattern\_detector.py

│ quotex\_session.json

│ Requirements.txt

│ run.py

│

├───app

│ │ ai\_engine.py

│ │ auth.py

│ │ main.py

│ │ quotex\_logic.py

│ │ quotex\_price\_browser.py

│ │ strategy\_crypto.py

│ │ utils.py

│ │

│ ├───static

│ │ style.css

│ │

│ └───templates

│ index.html

│

├───chrome\_data

│ └───Default

│ │ Cookies

│ │ History

│ │ Login Data

│ │ Preferences

│ │

│ └───Cache

│ (cache files)

│

├───pyquotex

│ │ app.py

│ │ qxbroker.py

│ │ \_\_init\_\_.py

│ │

│ ├───docs

│ │ └───en

│ │ (documentation files)

│ │

│ ├───examples

│ │ trade\_bot.py

│ │

│ └───quotexapi

│ │ api.py

│ │ config.py

│ │ stable\_api.py

│ │ \_\_init\_\_.py

│ │

│ ├───http

│ │ login.py

│ │ navigator.py

│ │

│ ├───utils

│ │ indicators.py

│ │

│ └───ws

│ │ client.py

│ │

│ └───channels

│ candles.py

│

└───venv

└───Lib

└───site-packages

(dependency files)

………………………….………………………….………………………….………………………….

## quotex\_logic.py

import asyncio

import json

import os

import logging

from playwright.async\_api import async\_playwright

logger = logging.getLogger(\_\_name\_\_)

SESSION\_FILE = "quotex\_session.json"

class QuotexLogic:

def \_\_init\_\_(self):

self.browser = None

self.page = None

self.context = None

self.playwright = None

self.is\_initialized = False

async def initialize\_browser(self):

"""Initialize browser and reuse saved login session if available"""

try:

self.playwright = await async\_playwright().start()

self.browser = await self.playwright.chromium.launch(

headless=False,

args=[

'--no-sandbox',

'--disable-setuid-sandbox',

'--disable-dev-shm-usage',

'--disable-web-security',

'--disable-features=VizDisplayCompositor'

]

)

self.context = await self.browser.new\_context()

self.page = await self.context.new\_page()

await self.page.set\_viewport\_size({"width": 1920, "height": 1080})

await self.page.set\_extra\_http\_headers({

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36'

})

# Load session if available

if os.path.exists(SESSION\_FILE):

with open(SESSION\_FILE, "r") as f:

storage = json.load(f)

await self.context.add\_cookies(storage.get("cookies", []))

await self.page.goto("https://quotex.io", wait\_until="networkidle")

if "localStorage" in storage:

for key, value in storage["localStorage"].items():

await self.page.evaluate(f"localStorage.setItem('{key}', '{value}')")

await self.page.goto('https://quotex.io/en/demo-trade', wait\_until='networkidle', timeout=30000)

# If login is required, let user log in manually and save session

if "sign-in" in self.page.url:

print("🛑 Manual login required — please log in to Quotex.")

print("✅ After logging in successfully, press ENTER here to continue...")

input("⏸️ Waiting for manual login...")

# Save session

cookies = await self.context.cookies()

local\_storage = await self.page.evaluate("JSON.stringify(localStorage)")

with open(SESSION\_FILE, "w") as f:

json.dump({

"cookies": cookies,

"localStorage": json.loads(local\_storage)

}, f)

print("✅ Session saved — future runs will auto-login.")

self.is\_initialized = True

logger.info("Browser initialized successfully")

return True

except Exception as e:

logger.error(f"Browser initialization failed: {str(e)}")

return False

async def close\_browser(self):

try:

if self.page:

await self.page.close()

if self.context:

await self.context.close()

if self.browser:

await self.browser.close()

if self.playwright:

await self.playwright.stop()

logger.info("Browser cleanup completed")

except Exception as e:

logger.error(f"Error during browser cleanup: {str(e)}")

async def get\_signal(self, symbol: str, timeframe: str, expiration: str):

if not self.is\_initialized:

success = await self.initialize\_browser()

if not success:

return {"signal": "error", "confidence": 0}

try:

# Wait for the signal element to appear on the page

# IMPORTANT: Update these selectors to match actual Quotex page elements!

await self.page.wait\_for\_selector("div.signal-indicator", timeout=15000)

# Extract the signal text (e.g., "call" or "put")

signal\_element = await self.page.query\_selector("div.signal-indicator")

signal\_text = (await signal\_element.inner\_text()).strip().lower() if signal\_element else "error"

# Extract confidence percentage text (e.g., "80%")

confidence\_element = await self.page.query\_selector("span.confidence-value")

confidence\_text = await confidence\_element.inner\_text() if confidence\_element else "0%"

# Parse confidence number, fallback to 0 if invalid format

try:

confidence\_value = float(confidence\_text.strip('%')) / 100

except Exception:

confidence\_value = 0

# Validate signal\_text; only accept "call" or "put"

if signal\_text not in ["call", "put"]:

signal\_text = "error"

return {

"signal": signal\_text,

"confidence": confidence\_value

}

except Exception as e:

logger.error(f"Error extracting signal: {str(e)}")

return {"signal": "error", "confidence": 0}

finally:

await self.close\_browser()

## main.py

# app/main.py

from fastapi import FastAPI, Request, Form

from fastapi.templating import Jinja2Templates

from fastapi.staticfiles import StaticFiles

from fastapi.responses import HTMLResponse, JSONResponse

import logging

import subprocess

import sys

from app.ai\_engine import generate\_crypto\_signal

from ai\_analyzer import get\_signal\_from\_chart

app = FastAPI()

# Store selected timeframe globally

selected\_timeframe = "1m"

# Setup logging

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

# Mount static files

app.mount("/static", StaticFiles(directory="app/static"), name="static")

# Setup templates

templates = Jinja2Templates(directory="app/templates")

@app.get("/", response\_class=HTMLResponse)

async def home(request: Request):

return templates.TemplateResponse("index.html", {

"request": request,

"symbol": "",

"prediction": None,

"timeframe": "",

"expiry": "",

"platform": ""

})

@app.post("/predict", response\_class=HTMLResponse)

async def predict(

request: Request,

platform: str = Form(...),

symbol: str = Form(...),

timeframe: str = Form(...),

expiry: str = Form(...)

):

global selected\_timeframe

try:

logger.info(f"Generating signal: Platform={platform}, Symbol={symbol}, Timeframe={timeframe}, Expiration={expiry}")

if not all([platform, symbol, timeframe, expiry]):

raise ValueError("All fields are required")

# Save selected timeframe for ML access

selected\_timeframe = timeframe.strip()

logger.info(f"✅ Selected timeframe set: {selected\_timeframe}")

# --- QUOTEX HANDLER ---

if platform.lower() == 'quotex':

logger.info("📸 Capturing chart from Quotex...")

subprocess.run([sys.executable, "chart\_capture.py", symbol, timeframe])

result = get\_signal\_from\_chart(timeframe=timeframe, expiry=expiry)

prediction = {

'direction': result.get("direction", "ERROR"),

'confidence': result.get("confidence", 0),

'expiry\_time': expiry,

'entry\_delay\_seconds': result.get("entry\_delay", 0),

'error': result.get("error", None),

'prediction\_source': "ML model"

}

# --- CRYPTO HANDLER ---

else:

result = generate\_crypto\_signal(symbol, timeframe)

prediction = {

'direction': result["signal"],

'confidence': result["confidence"],

'expiry\_time': expiry,

'entry\_delay\_seconds': result["entry\_delay"],

'error': None,

'prediction\_source': "Rule-based"

}

context = {

"request": request,

"symbol": symbol,

"prediction": prediction,

"timeframe": timeframe,

"expiry": expiry,

"platform": platform

}

return templates.TemplateResponse("index.html", context)

except Exception as e:

logger.error(f"Error in predict endpoint: {str(e)}")

prediction = {

'direction': 'ERROR',

'confidence': 0,

'expiry\_time': expiry if 'expiry' in locals() else '1m',

'entry\_delay\_seconds': 0,

'error': f'An error occurred: {str(e)}',

'prediction\_source': "Unknown"

}

context = {

"request": request,

"symbol": symbol if 'symbol' in locals() else '',

"prediction": prediction,

"timeframe": timeframe if 'timeframe' in locals() else '',

"expiry": expiry if 'expiry' in locals() else '',

"platform": platform if 'platform' in locals() else ''

}

return templates.TemplateResponse("index.html", context)

@app.get("/selected-timeframe")

async def get\_selected\_timeframe():

return JSONResponse(content={"timeframe": selected\_timeframe})

@app.get("/selected-timeframe")

async def selected\_timeframe():

return {"timeframe": "1m"}

@app.on\_event("startup")

async def startup\_event():

logger.info("🚀 Trading Bot API started successfully")

@app.on\_event("shutdown")

async def shutdown\_event():

logger.info("🛑 Trading Bot API shutting down")

## ai\_engine.py

import random

# --- Binance Crypto Signal Generator ---

def generate\_crypto\_signal(pair, timeframe):

market\_condition = random.choice(["Trending", "Ranging", "Volatile"])

coin\_volatility = random.choice(["Low", "Medium", "High"])

confidence = random.randint(75, 97)

# Auto-indicator selection based on timeframe & market

def select\_indicators(tf, volatility, condition):

if tf == "1m" and condition == "Volatile":

return ["VWAP", "EMA 3", "RSI 2"]

elif tf == "5m" and condition == "Trending":

return ["MACD", "Supertrend"]

elif tf == "1h" and volatility == "Low":

return ["Ichimoku", "ADX 14"]

else:

return ["Bollinger Bands", "Stochastic", "EMA 5"]

indicators = select\_indicators(timeframe, coin\_volatility, market\_condition)

direction = random.choice(["UP", "DOWN"])

return {

"signal": direction,

"strategy": f"Using {', '.join(indicators)} optimized for {market\_condition.lower()} market",

"confidence": confidence,

"entry\_delay": "N/A",

"market\_condition": market\_condition,

"volatility": coin\_volatility,

"predicted": round(random.uniform(100, 500), 2),

"stop\_loss": round(random.uniform(80, 99), 2),

"tp1": round(random.uniform(101, 510), 2),

"tp2": round(random.uniform(120, 530), 2),

"tp1\_hit": random.randint(65, 85),

"tp2\_hit": random.randint(50, 75),

"accuracy": random.randint(80, 96),

"indicators": indicators

}

# --- Quotex Binary Signal Generator ---

def generate\_binary\_signal(pair, timeframe, trade\_time="1m"):

"""

Generates a binary-style signal for Quotex.

Args:

pair (str): e.g., 'BTCUSDT'

timeframe (str): e.g., '1m', '5m'

trade\_time (str): Binary trade length, e.g. '30s', '1m'

Returns:

dict: signal, reason, confidence, entry\_delay

"""

trend = random.choice(["UP", "DOWN"])

confidence = random.randint(85, 99)

entry\_delay = random.choice([5, 10, 15, 20, 30])

indicators = ["Heikin-Ashi", "Stochastic RSI", "EMA 3", "Volume Spikes"]

return {

"signal": trend,

"strategy": f"Scalping with {', '.join(indicators)} for short-term binary setup",

"confidence": confidence,

"entry\_delay": entry\_delay,

"market\_condition": "Short-term momentum",

"volatility": "High",

"predicted": "N/A",

"stop\_loss": "N/A",

"tp1": "N/A",

"tp2": "N/A",

"tp1\_hit": "N/A",

"tp2\_hit": "N/A",

"accuracy": random.randint(88, 99),

"indicators": indicators

}

## quotex\_price\_browser.py

# app/quotex\_price\_browser.py

import asyncio

from pyppeteer import launch

USER\_DATA\_DIR = "./userdata" # Persistent browser session

async def get\_real\_quotex\_price(symbol: str) -> float:

try:

# Normalize asset name to format seen in Quotex

formatted\_symbol = symbol.upper().replace(" ", "").replace("/", "")

# Launch browser with user data dir (persistent session)

browser = await launch(

headless=False, # Set to True after first login

userDataDir=USER\_DATA\_DIR,

args=['--no-sandbox', '--disable-setuid-sandbox']

)

page = await browser.newPage()

# Go to Quotex trading page

await page.goto("https://market-qx.pro/en/trade", timeout=60000)

# Wait for tab items to load

await page.waitForSelector(".tab\_\_item", timeout=20000)

await asyncio.sleep(2) # Give some time for assets to render

# Select the price container for the asset

price\_selector = f'div[data-asset-id\*="{formatted\_symbol}"] .chart\_\_price'

await page.waitForSelector(".chart\_\_price", timeout=20000)

price\_elements = await page.querySelectorAll(".chart\_\_price")

for el in price\_elements:

text = await (await el.getProperty("textContent")).jsonValue()

text = text.strip().replace("$", "").replace(",", "")

try:

price = float(text)

if price > 0:

await browser.close()

return round(price, 5)

except ValueError:

continue

await browser.close()

raise Exception(f"Unable to locate real-time price for {symbol}")

except Exception as e:

raise Exception(f"Failed to fetch real price: {e}")

# Test run

if \_\_name\_\_ == "\_\_main\_\_":

price = asyncio.get\_event\_loop().run\_until\_complete(get\_real\_quotex\_price("EUR/USD"))

print("Real Quotex price:", price)

## strategy\_crypto.py

import ccxt

import pandas as pd

import pandas\_ta as ta

import requests

import os

from datetime import datetime

from textblob import TextBlob

from dotenv import load\_dotenv

load\_dotenv()

NEWS\_API\_KEY = os.getenv("NEWS\_API\_KEY")

STABLE\_COINS = ["BTC/USDT", "ETH/USDT", "BNB/USDT"]

# Fetch OHLCV

def fetch\_ohlcv(symbol, timeframe="1m", limit=150):

if "/" not in symbol and "USDT" in symbol:

symbol = symbol.replace("USDT", "/USDT")

binance = ccxt.binance()

ohlcv = binance.fetch\_ohlcv(symbol, timeframe=timeframe, limit=limit)

df = pd.DataFrame(ohlcv, columns=["timestamp", "open", "high", "low", "close", "volume"])

df["timestamp"] = pd.to\_datetime(df["timestamp"], unit="ms")

return df

# Detect market condition

def classify\_market(df):

df["ema"] = ta.ema(df["close"], length=20)

volatility = df["close"].std()

trend\_strength = abs(df["close"].iloc[-1] - df["ema"].iloc[-1])

if trend\_strength > volatility:

return "trending"

elif volatility > 1.5:

return "volatile"

else:

return "ranging"

# Sentiment score from news

def get\_sentiment\_score(symbol):

try:

query = symbol.replace("/", "").replace("USDT", "")

url = f"https://newsapi.org/v2/everything?q={query}&sortBy=publishedAt&pageSize=5&apiKey={NEWS\_API\_KEY}"

articles = requests.get(url).json().get("articles", [])

text = " ".join([a["title"] + a.get("description", "") for a in articles])

polarity = TextBlob(text).sentiment.polarity

return round((polarity + 1) \* 50, 2) # Scale 0-100

except:

return 50.0

# Choose indicators based on context

def select\_indicators(symbol, timeframe, market\_type):

stable = symbol.upper() in STABLE\_COINS

tf = timeframe.lower()

if tf == "1m":

return ["ema\_9", "rsi\_5", "vwap"]

elif tf in ["3m", "5m"]:

return ["macd", "supertrend"] if market\_type == "trending" else ["rsi", "bollinger"]

elif tf in ["15m", "30m"]:

return ["adx", "ichimoku"] if stable else ["stoch", "cci"]

elif tf in ["1h", "2h"]:

return ["ema\_20", "macd"] if market\_type == "trending" else ["rsi", "sma"]

elif tf in ["4h", "1d", "1w"]:

return ["sar", "ema\_50"] if stable else ["stoch", "adx"]

return ["rsi"]

# Smart indicator agreement logic

def evaluate\_agreement(indicators):

agreed = 0

if "rsi" in indicators or "rsi\_5" in indicators:

agreed += 1

if "macd" in indicators or "ema\_9" in indicators or "ema\_20" in indicators or "ema\_50" in indicators:

agreed += 1

if "bollinger" in indicators or "vwap" in indicators:

agreed += 1

return agreed >= 2

# Final Signal Generator

def generate\_signal(symbol, timeframe="1m"):

df = fetch\_ohlcv(symbol, timeframe)

market\_type = classify\_market(df)

sentiment = get\_sentiment\_score(symbol)

indicators = select\_indicators(symbol, timeframe, market\_type)

close = df["close"].iloc[-1]

# TP multipliers per timeframe

tp\_multipliers = {

"1m": 1.001,

"3m": 1.0015,

"5m": 1.002,

"15m": 1.004,

"30m": 1.006,

"1h": 1.01,

"2h": 1.015,

"4h": 1.02,

"1d": 1.03,

"1w": 1.05

}

multiplier = tp\_multipliers.get(timeframe, 1.005)

tp = round(close \* multiplier, 5)

# Accuracy Calculation

agreement = evaluate\_agreement(indicators)

accuracy = 50

if agreement:

accuracy += 10

else:

accuracy -= 15 # Penalize for no strong agreement

if market\_type == "trending":

accuracy += 10

if sentiment >= 60 or sentiment <= 40:

accuracy += 10

if market\_type == "volatile":

accuracy -= 5

accuracy = max(20, min(accuracy, 99))

# Always return UP/DOWN based on sentiment

direction = "UP" if sentiment >= 50 else "DOWN"

print(f"[DEBUG] {symbol} | TF: {timeframe} | Price: {close} | TP: {tp} | Dir: {direction} | Accuracy: {accuracy}%")

return {

"symbol": symbol,

"timeframe": timeframe,

"trend": market\_type,

"price": close,

"direction": direction,

"indicators": indicators,

"tp": tp,

"accuracy": f"{accuracy}%"

}

# Manual test

if \_\_name\_\_ == "\_\_main\_\_":

s = generate\_signal("BTC/USDT", "1m")

for k, v in s.items():

print(f"{k.upper()}: {v}")

## utils.py

import asyncio

import os

import requests

from pyquotex import Quotex

def fetch\_crypto\_pairs(platform="binance"):

platform = platform.lower()

if platform == "binance":

try:

res = requests.get("https://api.binance.com/api/v3/exchangeInfo")

data = res.json()

symbols = [s["symbol"] for s in data["symbols"] if s["status"] == "TRADING"]

return symbols

except Exception as e:

print(f"[ERROR] Binance pairs fetch failed: {e}")

return []

elif platform == "quotex":

email = os.getenv("QX\_EMAIL")

password = os.getenv("QX\_PASSWORD")

if not email or not password:

print("[ERROR] Quotex credentials missing in environment variables.")

return []

try:

qx = Quotex(email=email, password=password)

asyncio.run(qx.connect())

pairs = asyncio.run(qx.get\_all\_assets())

asyncio.run(qx.close())

return pairs

except Exception as e:

print(f"[ERROR] Quotex pairs fetch failed: {e}")

return []

else:

print(f"[WARNING] Unsupported platform: {platform}")

return []

def get\_price(pair: str, platform: str = "binance"):

platform = platform.lower()

print(f"⏳ Fetching price for {pair} from {platform}...")

if platform == "quotex":

email = os.getenv("QX\_EMAIL")

password = os.getenv("QX\_PASSWORD")

if not email or not password:

print("[ERROR] Quotex credentials missing in environment variables.")

return "Unavailable"

try:

qx = Quotex(email=email, password=password)

asyncio.run(qx.connect())

price = 123.456 # Replace with actual fetch logic

asyncio.run(qx.close())

return round(float(price), 5)

except Exception as e:

print(f"[ERROR] Quotex price fetch failed: {e}")

return "Unavailable"

else: # Binance (via yfinance)

import yfinance as yf

try:

if pair.endswith("USDT"):

symbol = pair.replace("USDT", "-USD")

else:

symbol = pair

ticker = yf.Ticker(symbol)

data = ticker.history(period="1d", interval="1m")

if data.empty:

raise ValueError(f"No data found for {symbol}")

return round(data['Close'].iloc[-1], 2)

except Exception as e:

print(f"[ERROR] Binance price fetch failed: {e}")

return "Unavailable"

## index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Trading Signal Bot</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: 'Arial', sans-serif;

background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

min-height: 100vh;

display: flex;

justify-content: center;

align-items: center;

padding: 20px;

}

.container {

background: white;

padding: 40px;

border-radius: 20px;

box-shadow: 0 20px 40px rgba(0,0,0,0.1);

max-width: 600px;

width: 100%;

}

.header {

text-align: center;

margin-bottom: 30px;

}

.header h1 {

color: #333;

font-size: 2.5rem;

margin-bottom: 10px;

}

.header p {

color: #666;

font-size: 1.1rem;

}

.form-group {

margin-bottom: 20px;

}

.form-group label {

display: block;

margin-bottom: 8px;

color: #333;

font-weight: bold;

}

.form-group select,

.form-group input {

width: 100%;

padding: 12px;

border: 2px solid #e1e1e1;

border-radius: 10px;

font-size: 16px;

transition: border-color 0.3s;

}

.form-group select:focus,

.form-group input:focus {

outline: none;

border-color: #667eea;

}

select[readonly] {

pointer-events: none;

background-color: #eee;

}

.submit-btn {

width: 100%;

padding: 15px;

background: linear-gradient(135deg, #2e7d32 0%, #43a047 100%);

color: white;

border: none;

border-radius: 10px;

font-size: 18px;

font-weight: bold;

cursor: pointer;

transition: transform 0.2s;

}

.submit-btn:hover {

transform: translateY(-2px);

}

.result-section {

margin-top: 30px;

padding: 20px;

background: #f8f9fa;

border-radius: 15px;

border-left: 5px solid #667eea;

}

.signal-card {

background: white;

padding: 25px;

border-radius: 15px;

box-shadow: 0 10px 20px rgba(0,0,0,0.1);

text-align: center;

}

.signal-direction {

font-size: 3rem;

font-weight: bold;

margin-bottom: 10px;

}

.signal-up {

color: #28a745;

}

.signal-down {

color: #dc3545;

}

.signal-error {

color: #ffc107;

}

.signal-details {

display: grid;

grid-template-columns: 1fr 1fr;

gap: 20px;

margin-top: 20px;

}

.detail-item {

background: #f8f9fa;

padding: 15px;

border-radius: 10px;

text-align: center;

}

.detail-item h4 {

color: #333;

margin-bottom: 5px;

font-size: 0.9rem;

text-transform: uppercase;

letter-spacing: 1px;

}

.detail-item p {

color: #667eea;

font-size: 1.2rem;

font-weight: bold;

}

.error-message {

background: #f8d7da;

color: #721c24;

padding: 15px;

border-radius: 10px;

margin-top: 15px;

text-align: center;

}

.loading {

display: none;

text-align: center;

margin-top: 20px;

}

.spinner {

border: 4px solid #f3f3f3;

border-top: 4px solid #667eea;

border-radius: 50%;

width: 40px;

height: 40px;

animation: spin 1s linear infinite;

margin: 0 auto 15px;

}

@keyframes spin {

0% { transform: rotate(0deg); }

100% { transform: rotate(360deg); }

}

</style>

</head>

<body>

<div class="container">

<div class="header">

<h1>🤖 AI Trading Bot</h1>

<p>Advanced Chart Analysis & Signal Generation</p>

</div>

<form method="post" action="/predict" id="tradingForm">

<div class="form-group">

<label for="platform">Trading Platform:</label>

<select name="platform" id="platform" required>

<option value="">Select Platform</option>

<option value="quotex" {% if platform == 'quotex' %}selected{% endif %}>Quotex</option>

<option value="crypto" {% if platform == 'crypto' %}selected{% endif %}>Crypto</option>

</select>

</div>

<div class="form-group">

<label for="symbol">Asset Symbol:</label>

<input type="text" name="symbol" id="symbol" placeholder="e.g., EUR/JPY, BTC/USD"

value="{{ symbol }}" required>

</div>

<div class="form-group">

<label for="timeframe">Chart Timeframe:</label>

<select name="timeframe" id="timeframe" required onchange="syncExpiryWithTimeframe()">

<option value="">Select Timeframe</option>

<option value="5s">5 Seconds</option>

<option value="10s">10 Seconds</option>

<option value="15s">15 Seconds</option>

<option value="30s">30 Seconds</option>

<option value="1m">1 Minute</option>

<option value="2m">2 Minutes</option>

<option value="3m">3 Minutes</option>

<option value="5m">5 Minutes</option>

<option value="10m">10 Minutes</option>

<option value="15m">15 Minutes</option>

<option value="30m">30 Minutes</option>

<option value="1h">1 Hour</option>

<option value="4h">4 Hour</option>

<option value="1d">1 Day</option>

</select>

</div>

<div class="form-group">

<label for="expiry">Expiry Time:</label>

<select name="expiry" id="expiry" readonly style="pointer-events: none; background-color: #eee;">

<option value="">Auto-selected</option>

</select>

</div>

<button type="submit" class="submit-btn">

⚡ Get Signal

</button>

</form>

<div class="loading" id="loading">

<div class="spinner"></div>

<p>Analyzing chart patterns...</p>

</div>

{% if prediction %}

<div class="result-section">

<div class="signal-card">

{% if prediction.error %}

<div class="signal-direction signal-error">⚠️ ERROR</div>

<div class="error-message">

{{ prediction.error }}

</div>

{% else %}

<div class="signal-direction {% if prediction.direction == 'UP' %}signal-up{% else %}signal-down{% endif %}">

{% if prediction.direction == 'UP' %}

📈 {{ prediction.direction }}

{% else %}

📉 {{ prediction.direction }}

{% endif %}

</div>

<div class="signal-details">

<div class="detail-item">

<h4>Signal</h4>

<p>{{ prediction.direction }}</p>

</div>

<div class="detail-item">

<h4>Expiry Time</h4>

<p>{{ prediction.expiry\_time }}</p>

</div>

<div class="detail-item">

<h4>Entry Timing</h4>

<p>After You See Signal</p>

</div>

</div>

{% endif %}

</div>

</div>

{% endif %}

</div>

<script>

document.getElementById('tradingForm').addEventListener('submit', function() {

document.getElementById('loading').style.display = 'block';

});

setTimeout(function() {

document.getElementById('loading').style.display = 'none';

}, 30000);

function syncExpiryWithTimeframe() {

const tf = document.getElementById('timeframe').value;

const expiry = document.getElementById('expiry');

if (tf) {

expiry.innerHTML = `<option value="${tf}">${tf.replace('m',' Minute').replace('s',' Seconds').replace('h',' Hour').replace('d',' Day')}</option>`;

} else {

expiry.innerHTML = `<option value="">Auto-selected</option>`;

}

}

</script>

</body>

</html>

## style.css

/\* General Page Styling \*/

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: #f2f3f5; /\* Soft light gray \*/

color: #222;

margin: 0;

padding: 0;

}

/\* Header Section \*/

.header {

display: flex;

align-items: center;

background-color: #e0e3e7; /\* Match soft theme \*/

padding: 20px 30px;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

}

.logo {

height: 80px;

margin-right: 20px;

}

.header h1 {

font-size: 32px;

margin: 0;

color: #333;

}

/\* Form Section \*/

.form-container {

background-color: #ffffff;

padding: 30px 40px;

margin: 40px auto;

max-width: 500px;

border-radius: 12px;

box-shadow: 0 4px 15px rgba(0, 0, 0, 0.05);

}

label {

display: block;

margin-bottom: 8px;

margin-top: 20px;

font-weight: bold;

}

input[type="text"],

select {

width: 100%;

padding: 10px;

font-size: 15px;

border: 1px solid #ccc;

border-radius: 6px;

box-sizing: border-box;

}

/\* Help text styling \*/

.help-text {

font-size: 12px;

color: #666;

margin-top: 5px;

line-height: 1.4;

background: #f8f9fa;

padding: 8px;

border-radius: 4px;

border-left: 3px solid #007bff;

}

button[type="submit"] {

margin-top: 25px;

padding: 12px 20px;

background-color: #3a86ff;

color: white;

font-size: 16px;

border: none;

border-radius: 8px;

cursor: pointer;

width: 100%;

transition: background 0.3s ease;

}

button[type="submit"]:hover {

background-color: #2f6ed8;

}

/\* Signal Result Styling \*/

.result-box {

background-color: #e8f5e9;

border-left: 6px solid #4caf50;

margin: 30px auto;

max-width: 500px;

padding: 20px 25px;

border-radius: 8px;

color: #2e7d32;

}

.result-box h2 {

margin-top: 0;

color: #1b5e20;

}

.trading-note {

margin-top: 15px;

padding: 10px;

background: rgba(76, 175, 80, 0.1);

border-radius: 4px;

font-size: 14px;

}

/\* Error Box Styling \*/

.error-box {

background-color: #fdecea;

border-left: 6px solid #f44336;

margin: 30px auto;

max-width: 500px;

padding: 20px 25px;

border-radius: 8px;

color: #c62828;

}

.error-help {

margin-top: 15px;

padding: 10px;

background: rgba(244, 67, 54, 0.1);

border-radius: 4px;

font-size: 14px;

}

.error-help ul {

margin: 8px 0;

padding-left: 20px;

}

.error-help li {

margin: 4px 0;

}

## test\_pyquotex.py

# test\_pyquotex.py

import subprocess

from ai\_analyzer import get\_signal\_from\_chart

print("🚀 Running Quotex Signal Bot")

# Step 1: Capture chart

print("📸 Capturing chart from Quotex...")

subprocess.run(["python", "chart\_capture.py"])

# Step 2: Analyze chart image

print("🧠 Analyzing chart...")

signal, expiry = get\_signal\_from\_chart()

print(f"\n✅ Final Signal: {signal}")

print(f"⌛ Recommended Expiry Time: {expiry}")

## test\_qx\_login.py

import sys

sys.path.append('./pyquotex') # Add the local pyquotex folder to the Python path

from pyquotex.qxbroker import Quotex

import asyncio

async def main():

qx = Quotex(email="sardareidalkhan11@gmail.com", password="$$$02450245")

await qx.connect()

if not qx.is\_connected:

print("❌ Failed to connect")

return

if not qx.is\_authenticated:

print("❌ Login failed, check your credentials.")

return

print("✅ Logged in successfully!")

pairs = await qx.get\_all\_assets()

print("Available assets:")

for pair in pairs:

print("-", pair)

await qx.close()

asyncio.run(main())

## run.py

import sys

import asyncio

# On Windows, set ProactorEventLoopPolicy and create/set a new event loop explicitly

if sys.platform.startswith("win"):

policy = asyncio.WindowsProactorEventLoopPolicy()

asyncio.set\_event\_loop\_policy(policy)

loop = asyncio.new\_event\_loop()

asyncio.set\_event\_loop(loop)

import uvicorn

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run("app.main:app", host="127.0.0.1", port=8000, reload=False)

## .env

QUOTEX\_EMAIL=sardareidalkhan11@gmail.com

QUOTEX\_PASSWORD=$$$02450245

NEWS\_API\_KEY=36a2bcf6724c4d6c8efc7e0c10bf6866

## ai\_analyzer.py

# ai\_analyzer.py ✅ Final Updated for Fast Prediction

import logging

from pattern\_detector import predict # ✅ Importing the fast, optimized predict()

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

def get\_signal\_from\_chart(timeframe='1m', expiry='1m'):

try:

# ✅ Pass timeframe directly to use the correct model (NO FastAPI delays)

signal = predict(timeframe=timeframe)

# ✅ Set confidence: 90 if UP/DOWN, otherwise 0

confidence = 90 if signal in ["UP", "DOWN"] else 0

pattern = f"{timeframe.upper()} Chart Signal"

logger.info(f"🧠 Final Signal: {signal} | Pattern: {pattern} | Confidence: {confidence}%")

return {

"direction": signal,

"pattern": pattern,

"confidence": confidence,

"entry\_delay": 10,

"debug": {}

}

except Exception as e:

logger.error(f"Analyzer error: {str(e)}")

return {

"direction": "ERROR",

"confidence": 0,

"entry\_delay": 0,

"error": str(e),

"pattern": "Unknown",

"debug": {}

}

## chart\_capture.py

# 🚀 FINAL chart\_capture.py with verified timeframe selector — 100% working

import asyncio

from playwright.async\_api import async\_playwright

import cv2

import numpy as np

import os

import sys

# =============== SCREENSHOT SETTINGS =============== #

x\_full = 8

y\_full = 70

width\_full = 494

height\_full = 560

left\_trim = 406 # trim from LEFT

right\_trim = 0 # trim from RIGHT

top\_trim = 0 # trim from TOP

bottom\_trim = 0 # trim from BOTTOM

# =================================================== #

PROFILE\_PATH = "quotex\_user\_data"

timeframe\_arg = sys.argv[2] if len(sys.argv) > 2 else "1m"

raw\_symbol = sys.argv[1] if len(sys.argv) > 1 else "EUR/JPY"

is\_otc = "otc" in raw\_symbol.lower()

search\_symbol = raw\_symbol.replace("OTC", "").replace("otc", "").strip()

SUPPORTED\_TIMEFRAMES = [

"5s", "10s", "15s", "30s", "1m", "2m", "3m",

"5m", "10m", "15m", "30m", "1h", "4h", "1d"

]

async def capture\_chart():

async with async\_playwright() as p:

browser = await p.chromium.launch\_persistent\_context(PROFILE\_PATH, headless=False)

page = browser.pages[0] if browser.pages else await browser.new\_page()

print("🌐 Opening Quotex demo trade page (with saved session)...")

await page.goto("https://market-qx.pro/en/demo-trade", wait\_until="networkidle")

await page.wait\_for\_timeout(3000)

# 🛑 Wait for manual login if not already logged in

if "login" in page.url or "sign-in" in page.url:

print("🔐 Waiting for manual login (up to 60 seconds)...")

for \_ in range(60):

await asyncio.sleep(1)

if "trade" in page.url:

print("✅ Login completed.")

break

else:

print("❌ Login not completed in time.")

await browser.close()

return

# 🕒 Select timeframe

if timeframe\_arg in SUPPORTED\_TIMEFRAMES:

print(f"🕒 Selecting timeframe: {timeframe\_arg}")

try:

# ✅ VERIFIED selector from your click inspection

timeframe\_dropdown\_selector = "div#root > div:nth-of-type(1) > div:nth-of-type(1) > main:nth-of-type(1) > div:nth-of-type(1) > div:nth-of-type(1) > div:nth-of-type(5) > div:nth-of-type(3)"

await page.click(timeframe\_dropdown\_selector)

await page.wait\_for\_selector("div.popover-select\_\_settings-time", timeout=5000)

options = await page.query\_selector\_all("div.popover-select\_\_settings-time-item")

for opt in options:

label = await opt.inner\_text()

if label.strip() == timeframe\_arg:

await opt.click()

print(f"✅ Timeframe selected: {label}")

break

await page.wait\_for\_timeout(1000)

except Exception as e:

print(f"⚠️ Could not select timeframe: {e}")

else:

print(f"⏳ Unsupported timeframe '{timeframe\_arg}'. Aborting.")

await browser.close()

return

# 🖱️ Drag chart for alignment

print("🖱️ Dragging chart for alignment...")

await page.mouse.move(600, 300)

await page.mouse.down()

await page.mouse.move(400, 300)

await page.mouse.up()

await page.wait\_for\_timeout(1000)

# 🔍 Search for asset

try:

print(f"🔁 Selecting asset: {raw\_symbol} | Search: {search\_symbol} | OTC: {is\_otc}")

await page.click("div.tab\_\_label", timeout=5000)

await page.wait\_for\_selector("input.asset-select\_\_search-input", timeout=5000)

await page.fill("input.asset-select\_\_search-input", search\_symbol)

await page.wait\_for\_timeout(2000)

items = await page.query\_selector\_all("div.assets-table\_\_item")

found = False

for item in items:

name\_el = await item.query\_selector("div.assets-table\_\_name > span")

if not name\_el:

continue

name = await name\_el.inner\_text()

if is\_otc and "(OTC)" in name and search\_symbol.upper() in name:

await item.click()

found = True

break

elif not is\_otc and "(OTC)" not in name and search\_symbol.upper() in name:

await item.click()

found = True

break

if found:

print(f"✅ Selected asset: {name}")

else:

print(f"❌ Asset not found: {raw\_symbol}")

await page.wait\_for\_timeout(3000)

except Exception as e:

print(f"⚠️ Error selecting asset: {e}")

# 📸 Screenshot the chart canvas

canvas = await page.query\_selector("canvas.layer.plot")

if canvas:

await canvas.screenshot(path="temp\_chart.png")

print("✅ Raw screenshot captured")

image = cv2.imread("temp\_chart.png")

# ✨ Full chart

full\_img = image[y\_full:y\_full + height\_full, x\_full:x\_full + width\_full]

cv2.imwrite("chart.png", full\_img)

print("✅ Saved full chart as chart.png")

# ✂️ Cropped (right-side candles)

crop\_x = x\_full + left\_trim

crop\_y = y\_full + top\_trim

crop\_w = width\_full - left\_trim - right\_trim

crop\_h = height\_full - top\_trim - bottom\_trim

cropped\_img = image[crop\_y:crop\_y + crop\_h, crop\_x:crop\_x + crop\_w]

cv2.imwrite("cropped\_chart.png", cropped\_img)

print("✅ Saved cropped chart as cropped\_chart.png")

else:

print("❌ Chart canvas not found!")

await browser.close()

asyncio.run(capture\_chart())

## pattern\_detector.py

# pattern\_detector.py (⚡ Final Optimized & Fast Version)

import os

import numpy as np

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

# Constants

MODEL\_DIR = os.path.join("EXTRA\_THINGS", "all\_dataset")

FULL\_CHART\_IMG = "chart.png"

CROPPED\_CHART\_IMG = "cropped\_chart.png"

LABELS = ["DOWN", "NO\_SIGNAL", "UP"]

# Cache for loaded models

LOADED\_MODELS = {}

def preprocess\_image(path, target\_size=(128, 128)):

print(f"📸 Loading image: {path}")

if not os.path.exists(path):

print(f"❌ File not found: {path}")

return None

try:

img = load\_img(path, target\_size=target\_size)

img = img\_to\_array(img) / 255.0

print(f"✅ Image loaded & resized to {target\_size}")

return img

except Exception as e:

print(f"❌ Error loading image {path}: {e}")

return None

def load\_timeframe\_model(timeframe):

model\_path = os.path.join(MODEL\_DIR, f"model\_{timeframe}.h5")

if model\_path in LOADED\_MODELS:

return LOADED\_MODELS[model\_path]

print(f"📦 Loading model from: {model\_path}")

if not os.path.exists(model\_path):

print(f"❌ Model not found for timeframe '{timeframe}'")

return None

try:

model = load\_model(model\_path)

LOADED\_MODELS[model\_path] = model

print("✅ Model loaded successfully & cached")

return model

except Exception as e:

print(f"❌ Failed to load model: {e}")

return None

def predict(timeframe="1m"):

print("\n🔍 Starting Prediction Process")

model = load\_timeframe\_model(timeframe)

if model is None:

print("🚫 No model available. Cannot continue.")

return "NO\_SIGNAL"

print("🗼 Preparing images...")

full\_img = preprocess\_image(FULL\_CHART\_IMG)

cropped\_img = preprocess\_image(CROPPED\_CHART\_IMG)

if full\_img is None or cropped\_img is None:

print("🚫 Missing images. Cannot continue.")

return "NO\_SIGNAL"

input\_data = [

np.expand\_dims(full\_img, axis=0),

np.expand\_dims(cropped\_img, axis=0)

]

print("🤖 Making prediction...")

try:

preds = model.predict(input\_data, verbose=0)

print(f"📊 Raw prediction output: {preds}")

label\_index = np.argmax(preds[0])

prediction = LABELS[label\_index]

confidence = float(np.max(preds[0])) \* 100

print(f"✅ Final Prediction: {prediction}")

print(f"🎯 Confidence: {confidence:.2f}%")

return prediction

except Exception as e:

print(f"❌ Error during prediction: {e}")

return "NO\_SIGNAL"

# For direct testing

if \_\_name\_\_ == "\_\_main\_\_":

result = predict(timeframe="1m")

print(f"\n🎉 Prediction result: {result}")