```
ftmo_bot/
├── data/                    # Ham veri (CSV) dizini
├── models/                  # Eğitilmiş modellerin saklanacağı dizin
├── logs/                    # Log dosyaları
├── reports/                 # Backtest raporları
├── src/
│   ├── alternative_data.py      # Sosyal medya, haber, COT, order-book
│   ├── feature_engineering.py   # M1/H1/H4 indikatörleri
│   ├── execution.py             # Emir açma/kapama fonksiyonları
│   ├── live_pipeline.py         # Canlı işlem akışı
│   ├── risk_manager.py          # FTMO risk kontrolleri
│   ├── backtester.py            # Offline backtest altyapısı
│   ├── telegram_bot.py          # Telegram komutları
│   └── dashboard/
│       └── dashboard_app.py     # Flask + Dash gösterge paneli
├── Dockerfile                   # Docker imajı tanımı
├── docker-compose.yml           # Servis orkestra dosyası
├── requirements.txt             # Python bağımlılıkları
├── .github/workflows/ci.yml     # GitHub Actions CI
└── README.md                    # Proje açıklaması
```

```
import os
import datetime
import requests
import pandas as pd
import MetaTrader5 as mt5
import tweepy
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from newsapi import NewsApiClient
from dotenv import load_dotenv

load_dotenv()

class MultiLingualTwitterSentiment:
    def __init__(self, bearer_token=None):
        token = bearer_token or os.getenv("TWITTER_BEARER_TOKEN")
        if not token:
            raise ValueError("TWITTER_BEARER_TOKEN ayarlanmalı.")
        self.client = tweepy.Client(bearer_token=token)
        self.analyzer = SentimentIntensityAnalyzer()
        self.hashtags = ["#forex", "#forextrading", "#EURUSD", "#GBPUSD",
"#USDJPY"]
        self.langs = ["en", "tr", "es", "ar"]

    def fetch_and_analyze(self):
```

```
        scores=[]
        for tag in self.hashtags:
            for lang in self.langs:
                q=f"{tag} lang:{lang} -is:retweet"
                try:
                    res=self.client.search_recent_tweets(query=q, max_results=50)
                except:
                    continue
                if not res or not res.data: continue
                for t in res.data:
                    vs=self.analyzer.polarity_scores(t.text)
                    scores.append(vs["compound"])
        return sum(scores)/len(scores) if scores else 0.0

class RedditSentiment:
    def __init__(self):
        self.subreddits=["forex","algotrading"]
        self.base_url="https://api.pushshift.io/reddit/search/submission/"
        self.analyzer=SentimentIntensityAnalyzer()

    def fetch_recent_posts(self,sub,hours=2):
        now=int(datetime.datetime.utcnow().timestamp())
        after=now-3600×hours
        p={"subreddit":sub,"size":100,"after":after,"sort":"desc"}
        try:
            r=requests.get(self.base_url,params=p,timeout=10)
            r.raise_for_status()
            data=r.json().get("data",[])
            return [d["title"]+" "+d.get("selftext",") for d in data]
        except:
            return []

    def analyze_sentiment(self):
        sc=[]
        for s in self.subreddits:
            for txt in self.fetch_recent_posts(s):
                vs=self.analyzer.polarity_scores(txt)
                sc.append(vs["compound"] )
        return sum(sc)/len(sc) if sc else 0.0

class NewsSentiment:
    def __init__(self,api_key=None):
        key=api_key or os.getenv("NEWSAPI_KEY")
        if not key: raise ValueError
        self.client=NewsApiClient(api_key=key)
        self.analyzer=SentimentIntensityAnalyzer()
        self.keywords=["forex","USD","EUR","FED rate","interest rate"]
```

```
    def fetch_headlines(self):
        today=datetime.datetime.utcnow().date().isoformat()
        try:
            arts=self.client.get_everything(q=" OR ".join(self.keywords),
from_param=today,to=today,language="en",sort_by="relevancy",page_size=100)
            return [a["title"]+" "+a.get("description","") for a in arts["articles"] if a.get("description")]
        except:
            return []

    def analyze_sentiment(self):
        sc=[]
        for txt in self.fetch_headlines():
            vs=self.analyzer.polarity_scores(txt)
            sc.append(vs["compound"])
        return sum(sc)/len(sc) if sc else 0.0

class EconomicCalendar:
    def __init__(self): self.url="https://cdn-nfs.forexfactory.net/ff_calendar_thisweek.json"
    def fetch_high_impact(self):
        try:
            r=requests.get(self.url,timeout=10); r.raise_for_status()
            ev=r.json().get("events",[])
            return [e for e in ev if e.get("impact")=="High"]
        except: return []

class COTLoader:
    def __init__(self,url): self.url=url
    def fetch(self):
        try: return pd.read_csv(self.url)
        except: return None

class OrderBookFeatures:
    def __init__(self,symbol):
        self.symbol=symbol
        if not mt5.initialize(): raise RuntimeError
    def fetch_order_book(self):
        book=mt5.market_book_get(self.symbol)
        if not book: return {"bid_volume":0,"ask_volume":0,"bid_ask_ratio":0}
        bids=sum(e.volume for e in book if e.type==mt5.BOOK_TYPE_SELL)
        asks=sum(e.volume for e in book if e.type==mt5.BOOK_TYPE_BUY)
        return {"bid_volume":bids,"ask_volume":asks,"bid_ask_ratio":bids/asks if asks>0 else 0}
```

```
import pandas as pd
import numpy as np
import MetaTrader5 as mt5
import talib
import datetime

class FeatureEngineer:
    def __init__(self, symbol):
        self.symbol = symbol
        if not mt5.initialize():
            raise RuntimeError(f"MT5 initialize hatası: {mt5.last_error()}")

    def get_bars(self, timeframe, n):
        rates = mt5.copy_rates_from_pos(self.symbol, timeframe, 0, n)
        df = pd.DataFrame(rates)
        df['time'] = pd.to_datetime(df['time'], unit='s')
        return df

    def normalize(self, series):
        min_v, max_v = series.min(), series.max()
        denom = max_v - min_v
        return (series - min_v) / denom if denom else series.apply(lambda x: 0.5)

    def compute_multitimeframe_features(self):
        df_m1 = self.get_bars(mt5.TIMEFRAME_M1, 500)
        df_h1 = self.get_bars(mt5.TIMEFRAME_H1, 500)
        df_h4 = self.get_bars(mt5.TIMEFRAME_H4, 500)

        # M1 indicators
        df_m1['RSI7'] = talib.RSI(df_m1['close'], timeperiod=7)
        df_m1['CCI14'] = talib.CCI(df_m1['high'], df_m1['low'], df_m1['close'],
timeperiod=14)
        df_m1['MOM5'] = talib.MOM(df_m1['close'], timeperiod=5)
        m1_rsi = self.normalize(df_m1['RSI7']).iloc[-1]
        m1_cci = self.normalize(df_m1['CCI14']).iloc[-1]
        m1_mom = self.normalize(df_m1['MOM5']).iloc[-1]

        # H1 indicators
        df_h1['RSI14'] = talib.RSI(df_h1['close'], timeperiod=14)
        macd, signal, hist = talib.MACD(df_h1['close'], fastperiod=12,
slowperiod=26, signalperiod=9)
        df_h1['MACD_hist'] = hist
        h1_rsi = self.normalize(df_h1['RSI14']).iloc[-1]
        h1_hist = self.normalize(df_h1['MACD_hist']).iloc[-1]
        df_h1['ATR14'] = talib.ATR(df_h1['high'], df_h1['low'], df_h1['close'],
```

```
timeperiod=14)
        h1_atr = self.normalize(df_h1['ATR14']).iloc[-1]

        # H4 indicators
        df_h4['EMA50'] = talib.EMA(df_h4['close'], timeperiod=50)
        df_h4['EMA200'] = talib.EMA(df_h4['close'], period=200)
        df_h4['EMA_cross'] = (df_h4['EMA50'] > df_h4['EMA200']).astype(int)
        h4_cross = df_h4['EMA_cross'].iloc[-1]

        return {
            'M1_RSI7': float(m1_rsi),
            'M1_CCI14': float(m1_cci),
            'M1_MOM5': float(m1_mom),
            'H1_RSI14': float(h1_rsi),
            'H1_MACD_HIST': float(h1_hist),
            'H1_ATR14': float(h1_atr),
            'H4_EMA_CROSS': int(h4_cross)
        }

(import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import random
from collections import namedtuple

Experience = namedtuple("Experience",
["state","action","reward","next_state","done","n_step_return","next_state_n"])

class PrioritizedReplayBuffer:
    def __init__(self, capacity, alpha=0.6, n_step=3, gamma=0.99):
        self.capacity = capacity
        self.buffer = []
        self.pos = 0
        self.priorities = np.zeros((capacity,), dtype=np.float32)
        self.alpha = alpha
        self.n_step = n_step
        self.gamma = gamma
        self.n_step_buffer = []

    def _get_n_step_info(self):
        reward, next_state, done = self.n_step_buffer[-1][2:]
        for transition in reversed(self.n_step_buffer[:-1]):
            r, n_s, d = transition[2], transition[3], transition[4]
            reward = r + self.gamma * reward * (1 - d)
            next_state = n_s
            done = d or done
```

```
        state, action = self.n_step_buffer[0][:2]
        return state, action, reward, next_state, done

    def add(self, state, action, reward, next_state, done):
        transition = (state, action, reward, next_state, done)
        self.n_step_buffer.append(transition)
        if len(self.n_step_buffer) < self.n_step:
            return
        s, a, r, ns, d = self._get_n_step_info()
        exp = Experience(s,a,r,ns,d,r,ns)
        max_prio = self.priorities.max() if self.buffer else 1.0
        if len(self.buffer) < self.capacity:
            self.buffer.append(exp)
        else:
            self.buffer[self.pos] = exp
        self.priorities[self.pos] = max_prio
        self.pos = (self.pos + 1) % self.capacity
        self.n_step_buffer.pop(0)

    def sample(self, batch_size, beta=0.4):
        prios = self.priorities if len(self.buffer)==self.capacity else
self.priorities[:self.pos]
        probs = prios ** self.alpha
        probs /= probs.sum()
        indices = np.random.choice(len(self.buffer), batch_size, p=probs)
        experiences = [self.buffer[idx] for idx in indices]
        total = len(self.buffer)
        weights = (total × probs[indices]) ** (-beta)
        weights /= weights.max()
        batch = Experience(×zip(×experiences))
        return batch, indices, weights

    def update_priorities(self, idxs, prios):
        for i,p in zip(idxs, prios): self.priorities[i]=p
    def __len__(self): return len(self.buffer)

class NoisyDense(layers.Layer):
    def __init__(self, units):
        super(NoisyDense,self).__init__()
        self.units=units
    def build(self,input_shape):
        input_dim=input_shape[-1]
        self.mu_w=self.add_weight('mu_w',
(input_dim,self.units),initializer=tf.random_uniform_initializer(-1/
np.sqrt(input_dim),1/np.sqrt(input_dim)))
        self.sigma_w=self.add_weight('sigma_w',
(input_dim,self.units),initializer=tf.constant_initializer(0.017))
```

```
        self.mu_b=self.add_weight('mu_b',
(self.units,),initializer=tf.random_uniform_initializer(-1/np.sqrt(input_dim),1/
np.sqrt(input_dim)))
        self.sigma_b=self.add_weight('sigma_b',
(self.units,),initializer=tf.constant_initializer(0.017))
    def call(self,inputs):
        i_dim=self.mu_w.shape[0]
        eps_i=tf.random.normal((i_dim,)); eps_j=tf.random.normal((self.units,))
        f_i=tf.sign(eps_i)*tf.sqrt(tf.abs(eps_i))
        f_j=tf.sign(eps_j)*tf.sqrt(tf.abs(eps_j))
        w_noise=tf.einsum('i,j->ij',f_i,f_j); b_noise=f_j
        w=self.mu_w+self.sigma_w*w_noise
        b=self.mu_b+self.sigma_b*b_noise
        return tf.matmul(inputs,w)+b

class DuelingRainbowDQN(tf.keras.Model):
    def __init__(self, state_size, action_size, atom_size=51, v_min=-10,
v_max=10):
        super().__init__()
        self.atom_size, self.v_min, self.v_max = atom_size, v_min, v_max
        self.support=tf.linspace(v_min,v_max,atom_size)
        self.fc1=NoisyDense(128); self.fc2=NoisyDense(128)
        self.val_fc=NoisyDense(128); self.val=NoisyDense(atom_size)
        self.adv_fc=NoisyDense(128);
self.adv=NoisyDense(action_size*atom_size)
    def call(self,x):
        x=tf.nn.relu(self.fc1(x)); x=tf.nn.relu(self.fc2(x))
        v=tf.nn.relu(self.val_fc(x)); v=self.val(v)
        adv=tf.nn.relu(self.adv_fc(x)); adv=self.adv(adv)
        adv=tf.reshape(adv,(-1,adv.shape[-1]//self.atom_size,self.atom_size))
        v=tf.reshape(v,(-1,1,self.atom_size))
        q_dist=v+(adv-tf.reduce_mean(adv,axis=1,keepdims=True))
        return tf.nn.softmax(q_dist,axis=2)

class RainbowAgent:
    def
__init__(self,state_size,action_size,buffer_size=50000,batch_size=32,gamma=
0.99,lr=1e-4,atom_size=51,v_min=-10,v_max=10,update_target_every=1000):

import MetaTrader5 as mt5
import datetime

class RiskManager:
    def __init__(self, starting_balance, daily_limit_pct, total_limit_pct,
max_equity_dd_pct):
        self.starting_balance = starting_balance
        self.daily_limit = starting_balance * (daily_limit_pct / 100)
```

```
        self.total_limit = starting_balance * (total_limit_pct / 100)
        self.max_equity_dd_pct = max_equity_dd_pct
        self.max_equity = starting_balance
        self.daily_start_balance = starting_balance
        self.current_day = None
        if not mt5.initialize():
            raise RuntimeError(f"MT5 initialize hatası: {mt5.last_error()}")

    def update_daily_balance(self):
        info = mt5.account_info()
        if info is None:
            return
        today = datetime.datetime.utcfromtimestamp(info.time).date()
        if self.current_day != today:
            self.daily_start_balance = info.balance
            self.current_day = today

    def check_daily_loss(self):
        info = mt5.account_info()
        if info is None:
            return False
        loss = self.daily_start_balance - info.balance
        return loss >= self.daily_limit

    def check_total_loss(self):
        info = mt5.account_info()
        if info is None:
            return False
        loss = self.starting_balance - info.equity
        return loss >= self.total_limit

    def update_max_equity(self):
        info = mt5.account_info()
        if info is None:
            return
        self.max_equity = max(self.max_equity, info.equity)

    def check_equity_drawdown(self):
        info = mt5.account_info()
        if info is None:
            return False
        dd_pct = (self.max_equity - info.equity) / self.max_equity * 100
        return dd_pct >= self.max_equity_dd_pct

    def enforce_limits(self):
        self.update_daily_balance()
        if self.check_daily_loss():
```

```
            positions = mt5.positions_get()
            if positions:
                for pos in positions:
                    if pos.profit < 0:
                        req = {
                            "action": mt5.TRADE_ACTION_DEAL,
                            "symbol": pos.symbol,
                            "volume": pos.volume,
                            "type": mt5.ORDER_TYPE_BUY if pos.type ==
mt5.ORDER_TYPE_SELL else mt5.ORDER_TYPE_SELL,
                            "position": pos.ticket,
                            "price": mt5.symbol_info_tick(pos.symbol).ask if pos.type ==
mt5.ORDER_TYPE_SELL else mt5.symbol_info_tick(pos.symbol).bid,
                            "deviation": 10,
                            "magic": pos.magic,
                            "comment": "Daily limit enforcement"
                        }
                        mt5.order_send(req)
            return False
        if self.check_total_loss() or self.check_equity_drawdown():
            positions = mt5.positions_get()
            if positions:
                for pos in positions:
                    req = {
                        "action": mt5.TRADE_ACTION_DEAL,
                        "symbol": pos.symbol,
                        "volume": pos.volume,
                        "type": mt5.ORDER_TYPE_BUY if pos.type ==
mt5.ORDER_TYPE_SELL else mt5.ORDER_TYPE_SELL,
                        "position": pos.ticket,
                        "price": mt5.symbol_info_tick(pos.symbol).ask if pos.type ==
mt5.ORDER_TYPE_SELL else mt5.symbol_info_tick(pos.symbol).bid,
                        "deviation": 10,
                        "magic": pos.magic,
                        "comment": "Total/equity drawdown enforcement"
                    }
                    mt5.order_send(req)
            return False

import MetaTrader5 as mt5

class OrderExecutor:
    @staticmethod
    def open_long(symbol, lot, deviation=20, magic=0, comment=""):
        tick = mt5.symbol_info_tick(symbol)
        if tick is None:
            print(f"Symbol {symbol} bilgisi alınamadı.")
```

```
            return None
        price = tick.ask
        request = {
            "action": mt5.TRADE_ACTION_DEAL,
            "symbol": symbol,
            "volume": lot,
            "type": mt5.ORDER_TYPE_BUY,
            "price": price,
            "deviation": deviation,
            "magic": magic,
            "comment": comment,
            "type_time": mt5.ORDER_TIME_GTC,
            "type_filling": mt5.ORDER_FILLING_IOC,
        }
        result = mt5.order_send(request)
        if result.retcode != mt5.TRADE_RETCODE_DONE:
            print(f"Long order hatası: {result.retcode}")
        return result

    @staticmethod
    def open_short(symbol, lot, deviation=20, magic=0, comment=""):
        tick = mt5.symbol_info_tick(symbol)
        if tick is None:
            print(f"Symbol {symbol} bilgisi alınamadı.")
            return None
        price = tick.bid
        request = {
            "action": mt5.TRADE_ACTION_DEAL,
            "symbol": symbol,
            "volume": lot,
            "type": mt5.ORDER_TYPE_SELL,
            "price": price,
            "deviation": deviation,
            "magic": magic,
            "comment": comment,
            "type_time": mt5.ORDER_TIME_GTC,
            "type_filling": mt5.ORDER_FILLING_IOC,
        }
        result = mt5.order_send(request)
        if result.retcode != mt5.TRADE_RETCODE_DONE:
            print(f"Short order hatası: {result.retcode}")
        return result

    @staticmethod
    def close_position(position):
        symbol = position.symbol
        lot = position.volume
```

```
        if position.type == mt5.ORDER_TYPE_BUY:
            price = mt5.symbol_info_tick(symbol).bid
            tp_type = mt5.ORDER_TYPE_SELL
        else:
            price = mt5.symbol_info_tick(symbol).ask
            tp_type = mt5.ORDER_TYPE_BUY
        request = {
            "action": mt5.TRADE_ACTION_DEAL,
            "symbol": symbol,
            "volume": lot,
            "type": tp_type,
            "position": position.ticket,
            "price": price,
            "deviation": 20,
            "magic": position.magic,
            "comment": "Position closed by bot",
            "type_time": mt5.ORDER_TIME_GTC,
            "type_filling": mt5.ORDER_FILLING_IOC,
        }
        result = mt5.order_send(request)
        if result.retcode != mt5.TRADE_RETCODE_DONE:
            print(f"Position close hatası: {result.retcode}")
        return result

    @staticmethod
    def modify_order(ticket, price, sl=None, tp=None):
        request = {
            "action": mt5.TRADE_ACTION_SLTP,
            "position": ticket,
            "sl": sl,
            "tp": tp,
            "comment": "Modify SL/TP"
        }
        result = mt5.order_send(request)
        if result.retcode != mt5.TRADE_RETCODE_DONE:
            print(f"Modify order hatası: {result.retcode}")
        return result

import asyncio
import numpy as np
import MetaTrader5 as mt5
import datetime
import requests

from feature_engineering import FeatureEngineer
from alternative_data import MultiLingualTwitterSentiment, RedditSentiment,
NewsSentiment, OrderBookFeatures
```

```
from risk_manager import RiskManager
from models.dqn_agent import RainbowAgent
from execution import OrderExecutor

# Ayarlar
MT5_LOGIN = None  # config üzerinden ayarlanmalı
MT5_PASSWORD = None
MT5_SERVER = None
MT5_PATH = None
SYMBOLS = ["EURUSD", "GBPUSD", "USDJPY"]
STARTING_BALANCE = 25000
DAILY_LIMIT_PCT = 5
TOTAL_LIMIT_PCT = 10
MAX_EQUITY_DD_PCT = 4
DASHBOARD_URL = "http://dashboard:8050"

async def on_tick(symbol, agent, risk_manager):
    # 1) Pause bayrağı kontrolü
    try:
        resp = requests.get(f"{DASHBOARD_URL}/api/pause_status")
        paused = resp.json().get('pause_flags', {}).get(symbol, False)
    except:
        paused = False
    if paused:
        return

    # 2) Feature hesaplama
    features = FeatureEngineer(symbol).compute_multitimeframe_features()
    tw = MultiLingualTwitterSentiment().fetch_and_analyze()
    rd = RedditSentiment().analyze_sentiment()
    nw = NewsSentiment().analyze_sentiment()
    ob = OrderBookFeatures(symbol).fetch_order_book()['bid_ask_ratio']

    state = np.array(list(features.values()) + [tw, rd, nw, ob], dtype=np.float32)

    # 3) Risk kontrolleri
    risk_manager.update_daily_balance()
    if not risk_manager.enforce_limits():
        return

    # 4) Override bayrağı
    try:
        resp = requests.get(f"{DASHBOARD_URL}/api/override_status")
        override = resp.json().get('override_flags', {}).get(symbol)
    except:
        override = None
```

```
    # 5) Ajan karar
    action = agent.get_action(state)
    if override == 'buy':
        OrderExecutor.open_long(symbol, lot=0.01)
    elif override == 'sell':
        OrderExecutor.open_short(symbol, lot=0.01)
    else:
        if action == 1:
            OrderExecutor.open_long(symbol, lot=0.01)
        elif action == 2:
            OrderExecutor.open_short(symbol, lot=0.01)

async def main_loop():
    if not mt5.initialize(login=MT5_LOGIN, password=MT5_PASSWORD,
server=MT5_SERVER, path=MT5_PATH):
        print("MT5 initialize hatası:", mt5.last_error())
        return

    risk_manager = RiskManager(STARTING_BALANCE, DAILY_LIMIT_PCT,
TOTAL_LIMIT_PCT, MAX_EQUITY_DD_PCT)
    state_size = 8 + 4  # feature + alternatifs
    agent = RainbowAgent(state_size, action_size=3)

    while True:
        await asyncio.gather(*(on_tick(sym, agent, risk_manager) for sym in
SYMBOLS))
        await asyncio.sleep(1)

if __name__ == '__main__':
    asyncio.run(main_loop())

import pandas as pd
import numpy as np
import os

from models.dqn_agent import RainbowAgent

class Position:
    def __init__(self, entry_price, direction, lot):
        self.entry_price = entry_price
        self.direction = direction
        self.lot = lot
        self.open = True

    def close_reward(self, exit_price, pip_value):
        pl = (exit_price - self.entry_price) * self.direction * self.lot * pip_value
```

```
        self.open = False
        return pl

class Backtester:
    def __init__(self, symbol, timeframe, start_date, end_date, data_dir,
starting_balance=25000):
        self.symbol = symbol
        self.timeframe = timeframe
        self.start_date = pd.to_datetime(start_date)
        self.end_date = pd.to_datetime(end_date)
        self.data_dir = data_dir
        self.starting_balance = starting_balance
        self.balance = starting_balance
        self.equity = starting_balance
        self.position = None
        self.history = []

        state_size = 8 + 4
        self.agent = RainbowAgent(state_size=state_size, action_size=3)

    def load_historical(self):
        fname = f"{self.symbol}_{self.timeframe}.csv"
        path = os.path.join(self.data_dir, fname)
        df = pd.read_csv(path)
        df['time'] = pd.to_datetime(df['time'])
        return df[(df['time'] >= self.start_date) & (df['time'] <=
self.end_date)].reset_index(drop=True)

    def run_backtest(self):
        df = self.load_historical()
        pip_value = 10
        for idx in range(500, len(df)-1):
            ts = df.loc[idx, 'time']
            state = np.zeros((8+4,))
            action = self.agent.get_action(state)

            if self.position is None or not self.position.open:
                if action == 1:
                    self.position = Position(df.loc[idx,'close'], 1, 0.01)
                elif action == 2:
                    self.position = Position(df.loc[idx,'close'], -1, 0.01)

            if self.position and self.position.open:
                exit_price = df.loc[idx+1, 'close']
                reward = self.position.close_reward(exit_price, pip_value)
                self.balance += reward
                self.equity = self.balance
```

```
                next_state = np.zeros_like(state)
                self.agent.store_transition(state, action, reward, next_state, True)
                self.agent.learn(beta=0.4)

            self.history.append((ts, self.equity, self.balance))

        total_return = self.balance - self.starting_balance
        print(f"Backtest tamamlandı: Toplam kar: {total_return:.2f} USD")
        return self.history

if __name__ == '__main__':
    bt=Backtester('EURUSD','H1','2020-01-01','2020-12-31','data',25000)
    hist=bt.run_backtest()
    pd.DataFrame(hist,columns=['time','equity','balance']).to_csv('reports/
backtest.csv',index=False)```

---

### src/dashboard/dashboard_app.py
```python
...dashboard code above...

# Temel imaj
FROM python:3.9-slim

RUN apt-get update && \
    apt-get install -y --no-install-recommends build-essential wget libatlas-
base-dev libsndfile1-dev \
    libta-lib0 libta-lib0-dev && rm -rf /var/lib/apt/lists/*

WORKDIR /app
COPY . /app
ENV VIRTUAL_ENV=/app/venv
RUN python3 -m venv $VIRTUAL_ENV && \
    $VIRTUAL_ENV/bin/pip install --upgrade pip && \
    $VIRTUAL_ENV/bin/pip install -r requirements.txt

ENV MT5_LOGIN="${MT5_LOGIN}"
ENV MT5_PASSWORD="${MT5_PASSWORD}"
ENV MT5_SERVER="${MT5_SERVER}"
ENV MT5_PATH="${MT5_PATH}"

EXPOSE 8050 8051
CMD ["python", "src/live_pipeline.py"]

version: '3.8'
services:
```
```

```
  dashboard:
    build: .
    container_name: ftmo_dashboard
    command: python src/dashboard/dashboard_app.py
    ports:
      - "8050:8050"
    environment:
      DASHBOARD_URL: http://localhost:8050

  telegram_bot:
    build: .
    container_name: ftmo_telegram_bot
    command: python src/telegram_bot.py
    environment:
      TELEGRAM_TOKEN: ${TELEGRAM_TOKEN}
      DASHBOARD_URL: http://dashboard:8050
    depends_on:
      - dashboard

  live_pipeline:
    build: .
    container_name: ftmo_live_pipeline
    command: python src/live_pipeline.py
    environment:
      MT5_LOGIN: ${MT5_LOGIN}
      MT5_PASSWORD: ${MT5_PASSWORD}
      MT5_SERVER: ${MT5_SERVER}
      MT5_PATH: ${MT5_PATH}
      DASHBOARD_URL: http://dashboard:8050
    depends_on:
      - dashboard
      - telegram_bot

name: CI
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
jobs:
  lint-and-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v4
        with: python-version: '3.9'
      - run: |
```

```
          python -m venv venv
          source venv/bin/activate
          pip install --upgrade pip
          pip install -r requirements.txt
      - name: Lint
        run: |
          source venv/bin/activate
          pip install flake8
          flake8 src
      - name: Smoke test backtester
        run: |
          source venv/bin/activate
          python - <<'PYCODE'
import pandas as pd
from backtester import Backtester
try:
    bt = Backtester('EURUSD','H1','2020-01-01','2020-01-07','data',1000)
    bt.run_backtest()
    print('Smoke test OK')
except FileNotFoundError:
    echo 'No data, skipping smoke test'
PYCODE

  docker_build:
    needs: lint-and-test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Build Docker image
        run: docker build -t ftmo_bot:latest .
```