# ДОДАТОК А

Бот для торгівлі криптовалютами

Лістинг А.1 – Програмний код файлу cryptobot.py

```
import os, time, sqlite3, joblib, ccxt, schedule, pandas as
pd, numpy as np
from datetime import datetime, timedelta
from tensorflow import keras
import xgboost as xgb
from hyperliquid.info import Info
from hyperliquid.utils import constants
from eth_account import Account
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.preprocessing.sequence import
TimeseriesGenerator
from sklearn.preprocessing import MinMaxScaler
import necessary_functions as n
import dontshare as d

# ── шляхи
SAVE_DIR = r"C:\Users\zheny\Diploma\pythonProject"
DB_FILE  = os.path.join(SAVE_DIR, "trades.db")
LSTM_F, XGB_F, SCALER_F = 'lstm.h5', 'xgb.pkl',
'scaler.pkl'
os.makedirs(SAVE_DIR, exist_ok=True)

# ── параметри
PAIR='BTC/USDT'; LEV=1; MAX_POS=1; ATR_SL=1.0
SEQ, EPOCHS, MIN_ORDER_USD = 30, 25, 10
TF={'1h':'1h','4h':'4h','1d':'1d'}

binance = ccxt.binance({'enableRateLimit': True})
account = Account.from_key(d.private_key)
```

Продовження лістингу А.1

```
    # ── база
    def ensure_db():
        with sqlite3.connect(DB_FILE) as c:
            c.execute("""CREATE TABLE IF NOT EXISTS trades(
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                timestamp TEXT,symbol TEXT,asset_type TEXT,
                action TEXT,side TEXT,qty REAL,price REAL,
                pnl_pct REAL,reason TEXT)""")
    def log_trade(sym, act, side, qty, price, pnl, reason):
        with sqlite3.connect(DB_FILE) as c:
            c.execute("""INSERT INTO trades

(timestamp,symbol,asset_type,action,side,qty,price,pnl_pct,rea
son)
              VALUES (?,?,?,?,?,?,?,?,?)""",

(datetime.utcnow().isoformat(timespec='seconds'),
               sym,'crypto',act,side,qty,price,pnl,reason))
    ensure_db()

    #  індикатори
    ema=lambda s,sp:s.ewm(span=sp,adjust=False).mean()
    def rsi(c,w=14):
        d=c.diff(); g=d.clip(lower=0); l=-d.clip(upper=0)

rs=g.ewm(alpha=1/w,adjust=False).mean()/l.ewm(alpha=1/w,adjust
=False).mean().replace(0,np.nan)
        return 100-100/(1+rs)
    def  macd(c):  f,s=ema(c,12),ema(c,26);  m=f-s;  return
m,ema(m,9)
    def atr(h,l,c,w=14):
        pc=c.shift();       tr=pd.concat([h-l,(h-pc).abs(),(l-
pc).abs()],axis=1).max(axis=1)
        return tr.ewm(alpha=1/w,adjust=False).mean()
```

Продовження лістингу А.1

```
    def bb(cl,w=20,k=2):
        ma=cl.rolling(w).mean();       sd=cl.rolling(w).std();
return ma+k*sd,ma-k*sd
    mvap=lambda
df,w=20:(df.close*df.volume).rolling(w).sum()/df.volume.rollin
g(w).sum()

    #  дані
    def fetch(tf,days=120):
        since=int((datetime.utcnow()-
timedelta(days=days)).timestamp()*1000)
        rows=[]
        while True:
            batch=binance.fetch_ohlcv(PAIR,TF[tf],since,1000)
            if not batch: break
            since=batch[-1][0]+1; rows+=batch
            if len(batch)<1000: break

df=pd.DataFrame(rows,columns=['ts','open','high','low','close'
,'volume'])
        df.ts=pd.to_datetime(df.ts,unit='ms');          return
df.set_index('ts').astype(float)

    def add_ind(df,tag):
        df[f'rsi_{tag}']=rsi(df.close)
        m,ms=macd(df.close);              df[f'macd_{tag}']=m;
df[f'macd_sig_{tag}']=ms
        df[f'atr_{tag}']=atr(df.high,df.low,df.close)
        up,lo=bb(df.close);               df[f'bbu_{tag}']=up;
df[f'bbl_{tag}']=lo
        df[f'mvap_{tag}']=mvap(df); return df

    def master(days=120):
```

Продовження лістингу А.1

```
    dfs={tf:add_ind(fetch(tf,days),tf)      for      tf      in
('1h','4h','1d')}
        base=dfs['1h']
        for tf in ('4h','1d'):
            cols=[c   for   c   in   dfs[tf]   if   c   not   in
('open','high','low','close','volume')]

base=pd.merge_asof(base,dfs[tf][cols].sort_index(),

left_index=True,right_index=True,direction='backward')
        return base.dropna()

    # ── ML
    def train():
        df=master(120); feats=list(df.columns)
        sc=MinMaxScaler().fit(df[feats]);
X=sc.transform(df[feats])

gen=TimeseriesGenerator(X,X[:,0],length=SEQ,batch_size=32)

lst=Sequential([LSTM(64,return_sequences=True,input_shape=(SEQ
,len(feats))),
                        Dropout(0.3),LSTM(32),Dense(1)])
        lst.compile('adam','mse');
lst.fit(gen,epochs=EPOCHS,verbose=0)

xgbm=xgb.XGBClassifier(n_estimators=300,max_depth=6,learning_r
ate=0.05,

subsample=0.8,colsample_bytree=0.8,eval_metric='logloss').fit(
                df[feats].iloc[:-1],         (df.close.shift(-
1)>df.close).astype(int)[:-1])
        lst.save(os.path.join(SAVE_DIR,LSTM_F))
        joblib.dump(xgbm,os.path.join(SAVE_DIR,XGB_F))
```

Продовження лістингу А.1

```
    joblib.dump(sc ,os.path.join(SAVE_DIR,SCALER_F))
        return lst,xgbm,sc
    def load_or_train():
        try:
            return
(keras.models.load_model(os.path.join(SAVE_DIR,LSTM_F)),
                    joblib.load(os.path.join(SAVE_DIR,XGB_F)),

joblib.load(os.path.join(SAVE_DIR,SCALER_F)))
        except: return train()
    lstm,xgb_clf,scaler = load_or_train()

    # ── сигнал
    last_sig=None
    def gen_sig(df):
        global last_sig,lstm,xgb_clf,scaler
        try: X=scaler.transform(df[scaler.feature_names_in_])
        except:                    lstm,xgb_clf,scaler=train();
X=scaler.transform(df[scaler.feature_names_in_])
        lstm_s=1   if   lstm.predict(X[-SEQ:].reshape(1,SEQ,-
1),verbose=0)[0,0]>X[-1,0] else 0
        xgb_s
=int(xgb_clf.predict(df[scaler.feature_names_in_].iloc[-
1:].values)[0])
        fin   =1 if (lstm_s+xgb_s)>1 else 0
        return None if fin==last_sig else fin

    # ── бот
    cur_sl=None; cur_dir=None
    def bot():
        global last_sig,cur_sl,cur_dir
        print(datetime.utcnow().strftime('%F %T'),"tick")
        df=master(90); atr=df.atr_1h.iloc[-1]; sig=gen_sig(df)
        pos=n.get_position_andmaxpos('BTC',account,MAX_POS)
```

Продовження лістингу А.1

```
        in_pos,long_f,entry=pos[1],pos[6],pos[4]

          # ATR-stop
          if in_pos and cur_sl:
              ask,bid,_=n.ask_bid('BTC'); mkt=bid if long_f else
ask
              if (long_f and mkt<=cur_sl) or (not long_f and
mkt>=cur_sl):
                  log_trade('BTC',"CLOSE","LONG" if long_f else
"SHORT",
                            abs(pos[2]),mkt,pos[5],"ATR-stop")
                  n.kill_switch('BTC',account);
cur_sl=cur_dir=None; in_pos=False

          # немає нового сигналу → нічого не робимо
          if sig is None:
              return

          want_long = (sig == 1)
          last_sig  = sig

          # ── якщо позиція вже є
          if in_pos:
              # протилежний сигнал → закриваємо, БЕЗ відкриття
              if (want_long and not long_f) or (not want_long and
long_f):
                  n.kill_switch('BTC',account)
                  log_trade('BTC',"CLOSE","LONG" if long_f else
"SHORT",
                            abs(pos[2]),entry,None,"opposite-
signal")
                  cur_dir=cur_sl=None
              return  # відкриваємось лише коли будемо flat на
наступному тіку
```

Продовження лістингу А.1

```
        # ── відкриваємо нову позицію (ми flat і є сигнал)
        _,  size  =  n.calculate_position_size('BTC',  LEV,
account)
        ask,bid,_=n.ask_bid('BTC'); px=bid if want_long else
ask
        if size*px < MIN_ORDER_USD:
            log_trade('BTC',"SKIP","LONG" if want_long else
"SHORT",
                      size,px,None,"too-small"); return
        n.limit_order('BTC',want_long,size,px,False,account)
        log_trade('BTC',"OPEN","LONG"  if  want_long  else
"SHORT",size,px,None,"signal")
        cur_dir = 1 if want_long else -1
        cur_sl  = px-ATR_SL*atr if want_long else px+ATR_SL*atr

    # ── scheduler
    bot()
    schedule.every(5).minutes.do(bot)
    while True:
        try:
            schedule.run_pending(); time.sleep(5)
        except Exception as e:
            print('ERR', e); time.sleep(30)
```

# ДОДАТОК Б

Бот для торгівлі акціями

Лістинг Б.1 – stockbot.py

```
"""
alpacabot_v7.py  ·  NVDA live stream
– потік 1-хв барів ➜ агрегуємо 1h / 4h / 1d
– індикатори: RSI-14, MACD, Bollinger, MVAP
– прогнози LSTM + XGBoost  → голосування сигналів
"""

from __future__ import annotations
import numpy as np, pandas as pd
from dataclasses import dataclass
from datetime import datetime, timedelta, timezone, time
from typing import Dict, Any, Optional, List

# ML-моделі
from tensorflow.keras.models import load_model
import xgboost as xgb
import joblib

# Alpaca SDK
from alpaca.data.enums   import DataFeed
from             alpaca.data.historical             import
StockHistoricalDataClient
from alpaca.data.live.stock import StockDataStream, Bar
from alpaca.data.requests   import StockBarsRequest
from alpaca.data.timeframe  import TimeFrame
# імпорт для SQLite-логування
import sqlite3, os
DB_PATH = "trades.db"
```

Продовження лістингу Б.1

```
# ініціалізація БД (створюємо таблиці, якщо їх ще нема)
if not os.path.exists(DB_PATH):
    open(DB_PATH, "w").close()

conn = sqlite3.connect(DB_PATH, check_same_thread=False)
cur  = conn.cursor()



def ensure_db():
    with sqlite3.connect(DB_PATH) as c:
        c.execute("""CREATE TABLE IF NOT EXISTS trades(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp TEXT,symbol TEXT,asset_type TEXT,
            action TEXT,side TEXT,qty REAL,price REAL,
            pnl_pct REAL,reason TEXT)""")
def log_trade(sym, act, side, qty, price, pnl, reason):
    with sqlite3.connect(DB_PATH) as c:
        c.execute("""INSERT INTO trades

(timestamp,symbol,asset_type,action,side,qty,price,pnl_pct,rea
son)
            VALUES (?,?,?,?,?,?,?,?,?)""",

(datetime.utcnow().isoformat(timespec='seconds'),
             sym,'crypto',act,side,qty,price,pnl,reason))
ensure_db()


# ─── Налаштування
API_KEY, API_SECRET = "ВАШ_API_KEY", "ВАШ_SECRET"
SYMBOL              = "NVDA"

feed_min, feed_day  = DataFeed.IEX, DataFeed.SIP
```

Продовження лістингу Б.1

```
DAYS_MIN = 10                          # історія хвилинок
RSI_N    = 14
WIN_MIN, WIN_H, WIN_4H, WIN_D = 9_000, 120, 40, 400

# ML-заготовки

lstm_model = load_model("models/lstm_nvda.h5")
xgb_clf    = joblib.load("models/xgb_nvda.pkl")
scaler     = joblib.load("models/scaler.pkl")
SEQ        = 60
feat_cols  = ['close','volume','rsi','macd','mvap']

# ── Індикатори
def ema(s,n): return s.ewm(span=n, adjust=False).mean()

def rsi(c, n=14):
    d = c.diff()
    gain, loss = d.clip(lower=0), -d.clip(upper=0)
    return 100 - 100 / (1 + gain.rolling(n).mean() /
loss.rolling(n).mean().replace(0, np.nan))

def macd(c,f=12,s=26,sig=9):
    m = ema(c,f) - ema(c,s)
    return m, ema(m,sig)

def bb(c,w=20,k=2):
    ma, sd = c.rolling(w).mean(), c.rolling(w).std()
    return ma+k*sd, ma-k*sd

def mvap(df,w=20):
    return
(df['close']*df['volume']).rolling(w).sum()/df['volume'].rolli
ng(w).sum()
```

Продовження лістингу Б.1

```
    def calc(df):
        out = df.copy()
        if len(out) >= RSI_N:
            out['rsi'] = rsi(out['close'])
            out['macd'],          out['macd_signal']          =
macd(out['close'])
        else:
            out[['rsi','macd','macd_signal']] = np.nan
        out['boll_up'], out['boll_lo'] = bb(out['close'])
        out['mvap'] = mvap(out)
        return out

    #  Агрегатор барів
    @dataclass
    class Agg:
        tf: str; ts: pd.Timestamp
        open: float; high: float; low: float; close: float;
volume: float = 0.
        @classmethod
        def start(cls, bar: Bar, tf: str):
            return                                        cls(tf,
pd.Timestamp(bar.timestamp).floor(tf),
                        bar.open, bar.high, bar.low, bar.close,
bar.volume)
        def upd(self, bar: Bar):
            self.high = max(self.high, bar.high)
            self.low  = min(self.low,  bar.low)
            self.close = bar.close
            self.volume += bar.volume
        def fin(self) -> Dict[str,Any]:
            return
{'open':self.open,'high':self.high,'low':self.low,
                    'close':self.close,'volume':self.volume}
```

Продовження лістингу Б.1

```
    # Логи у консоль
    def plog(lbl,row):
        ts = row.name.tz_convert('UTC') if row.name.tzinfo else
row.name
        r  = f"{row.rsi:.1f}" if not np.isnan(row.rsi) else '—
'
        m   =  f"{row.macd:.3f}/{row.macd_signal:.3f}"  if  not
np.isnan(row.macd) else '—'
        print(f"[{ts:%Y-%m-%d          %H:%M}]          {lbl}
close={row.close:.2f} | rsi={r} | macd={m}")

    #  Історичні дані
    cli = StockHistoricalDataClient(API_KEY, API_SECRET)
    def fetch(sym, tf, start, feed):
        req = StockBarsRequest(sym, tf, start=start, feed=feed,
limit=10_000)
        raw = cli.get_stock_bars(req).df
        return
raw[raw.index.get_level_values(0)==sym].droplevel(0)

    print(" Завантаження історії …")
    utc_now = datetime.now(timezone.utc)
    df_min   =   fetch(SYMBOL,   TimeFrame.Minute,   utc_now-
timedelta(days=DAYS_MIN), feed_min)
    df_min.index = df_min.index.tz_convert("America/New_York")
    df_min                                                    =
df_min.between_time("09:30","16:00").tz_convert("UTC")
    df_min = calc(df_min.tail(WIN_MIN))

    hourly                                                    =
df_min.resample('1h',label='right',closed='right',offset='30mi
n').agg(
```

Продовження лістингу Б.1

```
    {'open':'first','high':'max','low':'min','close':'last','volume':'sum'}).dropna()
    df_h = calc(hourly.tail(WIN_H))

    fourly = hourly.resample('4h',label='right',closed='right',offset='30min').agg(
{'open':'first','high':'max','low':'min','close':'last','volume':'sum'}).dropna()
    df_4h = calc(fourly.tail(WIN_4H))

    df_day = fetch(SYMBOL, TimeFrame.Day, utc_now-timedelta(days=400), feed_day).tail(WIN_D)
    df_day = calc(df_day)

    # Потік та буфери
    stream = StockDataStream(API_KEY, API_SECRET, feed=feed_min)
    buf_h: Optional[Agg] = None
    buf_4h: Optional[Agg] = None
    buf_day: Optional[Agg] = None

    def is_close_ny(ts):                # кінець сесії (16:00 NY)
        return ts.tz_convert("America/New_York").time() == time(16,0)

    # Голосування сигналів
    def generate_signal() -> Optional[int]:
        if len(df_h) < SEQ: return None

        # LSTM
        X = scaler.transform(df_h[feat_cols])
```

Продовження лістингу Б.1

```
    lstm_s = 1 if lstm_model.predict(X[-SEQ:].reshape(1,SEQ,-
1))[0,0] > X[-1,0] else 0
        # XGBoost
        xgb_s    =  int(xgb_clf.predict(df_h[feat_cols].iloc[-
1:].values)[0])

        # RSI
        rsi_last = [df['rsi'].iloc[-1] for df in (df_h, df_4h,
df_day)]
        rsi_s = 1 if all(v < 30 for v in rsi_last) else 0 if
all(v > 70 for v in rsi_last) else None
        # MACD
        macd_b       =       all(df['macd'].iloc[-1]       >
df['macd_signal'].iloc[-1] for df in (df_h, df_4h, df_day))
        macd_sg = 1 if macd_b else 0 if all(df['macd'].iloc[-
1] < df['macd_signal'].iloc[-1] for df in (df_h, df_4h, df_day))
else None
        # Bollinger (1h)
        price = df_h['close'].iloc[-1]
        boll_s = 1 if price < df_h['boll_lo'].iloc[-1] else 0
if price > df_h['boll_up'].iloc[-1] else None
        # MVAP
        mvap_s = 1 if price > df_h['mvap'].iloc[-1] else 0 if
price < df_h['mvap'].iloc[-1] else None

        votes       =       [v       for       v       in
[lstm_s,xgb_s,rsi_s,macd_sg,boll_s,mvap_s] if v is not None]
        if not votes: return None
        return 1 if votes.count(1) > votes.count(0) else 0

    #  Обробка кожного нового 1-хв бару
    async def on_min(bar: Bar):
        global  df_min,  df_h,  df_4h,  df_day,  buf_h,  buf_4h,
buf_day
```

Продовження лістингу Б.1

```
        ts = pd.Timestamp(bar.timestamp)
        # хвилинний DataFrame
        df_min.loc[ts]                                          =
{'open':bar.open,'high':bar.high,'low':bar.low,

'close':bar.close,'volume':bar.volume}
        df_min = calc(df_min.tail(WIN_MIN))

        # годинний буфер
        h_ts = ts.floor('1h')
        if buf_h is None: buf_h = Agg.start(bar,'1h')
        elif h_ts == buf_h.ts: buf_h.upd(bar)
        else:
            df_h.loc[buf_h.ts]   =   buf_h.fin();   df_h   =
calc(df_h.tail(WIN_H))
            buf_h = Agg.start(bar,'1h')

        # 4h буфер
        h4_ts = ts.floor('4h')
        if buf_4h is None: buf_4h = Agg.start(bar,'4h')
        elif h4_ts == buf_4h.ts: buf_4h.upd(bar)
        else:
            df_4h.loc[buf_4h.ts]  =  buf_4h.fin();  df_4h  =
calc(df_4h.tail(WIN_4H))
            buf_4h = Agg.start(bar,'4h')

        # денний буфер
        if buf_day is None: buf_day = Agg.start(bar,'1d')
        else: buf_day.upd(bar)
        if is_close_ny(ts):
            df_day.loc[buf_day.ts] = buf_day.fin(); df_day =
calc(df_day.tail(WIN_D))
            buf_day = None
```

Продовження лістингу Б.1

```
# лог індикаторів раз на хвилину
    plog("1m", df_min.iloc[-1])

    # сформувати торговий сигнал
    sig = generate_signal()
    if sig is not None:
        side = "BUY" if sig else "SELL"
        print(f" {ts:%Y-%m-%d %H:%M} → SIGNAL = {side}")
        # TODO: stream.submit_order(...)

# підписка на хвилинні бари
stream.subscribe_bars(on_min, SYMBOL)

print(" Старт стріму для NVDA (Ctrl-C для зупинки)")
if __name__ == "__main__":
    try:
        stream.run()
    except KeyboardInterrupt:
        print("\n Зупинено")
```

# ДОДАТОК В

Візуальний інтерфейс

Лістинг В.1 – програмний код файлу visual.py

```
import streamlit as st
import pandas as pd
import sqlite3
from streamlit_autorefresh import st_autorefresh

import bot2_with_db

# Автооновлення кожні 15 хвилин
st_autorefresh(interval=15 * 60 * 1000, key="datarefresh")

#  Завантаження з БД
@st.cache_data
@st.cache_data
def load_data():
    conn = sqlite3.connect("trades.db")
    df = pd.read_sql("SELECT * FROM trades", conn)
#  без parse_dates
    conn.close()

    #  уніфікуємо: T → пробіл
    df["timestamp"] = (
        df["timestamp"]
        .astype(str)                 # NaT → 'NaT'
        .str.replace("T", " ", regex=False)
        .mask(lambda s: s == "NaT", None)   # повертаємо
None
    )

    #  остаточне перетворення
```

Продовження лістингу В.1

```
    df["timestamp"] = pd.to_datetime(df["timestamp"],
errors="coerce")

        return df


    df = load_data()
    df_crypto = df[df["asset_type"] == "crypto"]
    df_stocks = df[df["asset_type"] == "stock"]

    st.title("Інтелектуальна торгова система")
    st.caption("Статистика за період 20.05 — 05.06.2025")

    #  Поточний стан
    st.subheader(" Поточний стан ботів")

    def show_latest_position(df_asset, label):
        last_open = df_asset[df_asset["action"] ==
"OPEN"].sort_values("timestamp").iloc[-1]
        last_close = df_asset[df_asset["action"] ==
"CLOSE"].sort_values("timestamp").iloc[-1]
        is_open = last_close["timestamp"] <
last_open["timestamp"]
        status = f"У позиції ({last_open['side']}) з
{last_open['timestamp']}" if is_open else "◯ Позиція закрита"
        st.markdown(f"**{label}:** {status}")

    col1, col2 = st.columns(2)
    with col1:
        show_latest_position(df_crypto, "Бот по крипті")
    with col2:
        show_latest_position(df_stocks, "Бот по акціях")

    #  Графік equity
```

Продовження лістингу В.1

```
    st.subheader(" Крива прибутковості")

    def equity_curve(df_asset, label):
        df_trades = df_asset[df_asset["action"] ==
"CLOSE"].copy()
        df_trades["pnl_pct"] =
pd.to_numeric(df_trades["pnl_pct"], errors='coerce').fillna(0)
        df_trades["equity"] = 10000 * (1 +
df_trades["pnl_pct"] / 100).cumprod()

st.line_chart(df_trades.set_index("timestamp")["equity"],
height=200)

    col3, col4 = st.columns(2)
    with col3:
        st.markdown("**Криптовалюта (BTC)**")
        equity_curve(df_crypto, "BTC")
    with col4:
        st.markdown("**Акції (NVDA)**")
        equity_curve(df_stocks, "NVDA")

    #  Статистика
    st.subheader(" Загальна статистика")

    def full_stats(df_asset, label):
        df_trades = df_asset[df_asset["action"] ==
"CLOSE"].copy()
        df_trades["pnl_pct"] =
pd.to_numeric(df_trades["pnl_pct"], errors='coerce').fillna(0)
        equity_final = 10000 * (1 + df_trades["pnl_pct"] /
100).cumprod().iloc[-1] if not df_trades.empty else 10000
        total = len(df_trades)
        wins = df_trades[df_trades["pnl_pct"] > 0]
        loss = df_trades[df_trades["pnl_pct"] < 0]
```

Продовження лістингу В.1

```
        col1, col2, col3 = st.columns(3)
        col1.metric(" Усього угод", total)
        col2.metric(" Прибуткових", len(wins))
        col3.metric("Збиткових", len(loss))
        st.markdown(f"###  Фінальний equity ({label}):
**${equity_final:,.2f}**")

    st.markdown("###  Крипта")
    full_stats(df_crypto, "BTC")
    st.markdown("###  Акції")
    full_stats(df_stocks, "NVDA")

    #  Кнопка перенавчання
    st.subheader(" Ручне перенавчання моделей")
    if st.button(" Запустити перенавчання"):
        working_alpaca_withdb.update_models()
        bot2_with_db.retrain_models()
    #  Угоди
    st.subheader(" Журнал угод")

    def style_pnl(val):
        if pd.isna(val): return ""
        return f"color: {'green' if val > 0 else 'red'}"

    styled = df.style.applymap(style_pnl, subset=["pnl_pct"])
    st.dataframe(styled, use_container_width=True, height=450)
```

# ДОДАТОК Г

Відомість кваліфікаційної роботи

| Позначення | Найменування | Дод. відомості |
|---|---|---|
| | Текстові документи | |
| **1.** | Пояснювальна записка | 99 с. |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | Інші документи | |
| | | |
| **2.** | Презентаційні матеріали | __ плакатів |
| | | |
| | | |
| | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| . | . | Прізвище та ініціали. | Підп. | Дата | | | |
| Розробив | | Джура Є.С. | | | Інформаційна система для торгівлі акціями та криптовалютами з використанням інтелектуального аналізу ринкових даних | Шифр групи | Код напр./спец. |
| Перевірив | | Кудрявцева М.С. | | | | ІТШІ-21-2 | 122 |
| Н.контр. | | Малєєва І.А. | | | | ХНУРЕ кафедра ШІ | |
| Затв. | | Золотухін О.В. | | | | | |