Infelizmente, não posso criar arquivos ZIP diretamente aqui, mas posso gerar todo o conteúdo do bot modular, pronto para você salvar em arquivos e depois compactar em um ZIP no seu computador.

Segue a estrutura completa para você criar o bot modular, cada seção corresponde a um arquivo:

1️⃣ config.py

```
API_KEY = "SUA_API_KEY_TESTNET"
API_SECRET = "SEU_API_SECRET_TESTNET"

SYMBOLS = ["BTCUSDT", "ETHUSDT", "BNBUSDT", "XRPUSDT", "ADAUSDT"]
INTERVALS = ["1m", "5m"]
LEVERAGE = 5

TRADE_PERCENT = 0.01
STOP_LOSS = 0.01
TAKE_PROFIT = 0.02
MAX_THREADS = 5

LOG_FILE = "bot_logs.txt"

TELEGRAM_TOKEN = ""
TELEGRAM_CHAT_ID = ""
```

2️⃣ utils.py

```
import datetime
import config
import requests

def log(message):
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    msg = f"[{timestamp}] {message}"
    print(msg)
    with open(config.LOG_FILE, "a") as f:
        f.write(msg + "\n")

def send_telegram(message):
    if not config.TELEGRAM_TOKEN or not config.TELEGRAM_CHAT_ID:
        return
    url = f"https://api.telegram.org/bot{config.TELEGRAM_TOKEN}/sendMessage"
    try:
        requests.post(url, data={"chat_id": config.TELEGRAM_CHAT_ID, "text":
message})
    except Exception as e:
        log(f"Erro ao enviar Telegram: {e}")
```

```
def get_balance(client):
    try:
        balance = client.futures_account_balance()
        for b in balance:
            if b['asset'] == 'USDT':
                return float(b['balance'])
    except Exception as e:
        log(f"Erro ao buscar saldo: {e}")
    return 1000
```

3️⃣ **trader.py**

```
import pandas as pd
from binance.client import Client
from binance.enums import *
import config
from utils import log, send_telegram

def connect_client():
    try:
        client = Client(config.API_KEY, config.API_SECRET, testnet=True)
        log("Conectado ao Testnet Binance")
        return client
    except Exception as e:
        log(f"Erro de conexão: {e}")
        return None

client = connect_client()

def get_klines(symbol, interval, limit=100):
    try:
        data = client.futures_klines(symbol=symbol, interval=interval,
limit=limit)
        df = pd.DataFrame(data, columns=[

'timestamp','open','high','low','close','volume','close_time','qav','num_trades','taker_base',
        ])
        for col in ['open','high','low','close','volume']:
            df[col] = df[col].astype(float)
        return df[['open','high','low','close','volume']]
    except Exception as e:
        log(f"Erro ao buscar candles: {e}")
        return None

def place_order(symbol, side, quantity, stop_loss=None, take_profit=None):
    try:
        order = client.futures_create_order(symbol=symbol, side=side,
type=FUTURE_ORDER_TYPE_MARKET, quantity=quantity)
        log(f"Ordem executada: {side} {quantity} {symbol}")
        send_telegram(f"Ordem executada: {side} {quantity} {symbol}")
```

```
        price = float(order['fills'][0]['price'])
        if stop_loss:
            sl_price = price * (1 - stop_loss) if side=='BUY' else
price*(1+stop_loss)
            client.futures_create_order(symbol=symbol, side='SELL' if
side=='BUY' else 'BUY', type=FUTURE_ORDER_TYPE_STOP_MARKET,
stopPrice=round(sl_price,2), closePosition=True)
            log(f"Stop Loss colocado: {round(sl_price,2)}")
        if take_profit:
            tp_price = price*(1+take_profit) if side=='BUY' else price*(1-
take_profit)
            client.futures_create_order(symbol=symbol, side='SELL' if
side=='BUY' else 'BUY', type=FUTURE_ORDER_TYPE_LIMIT, price=round(tp_price,
2), quantity=quantity, reduceOnly=True)
            log(f"Take Profit colocado: {round(tp_price,2)}")
        return order
    except Exception as e:
        log(f"Erro ao executar ordem: {e}")
        return None

def safe_order(symbol, side, quantity, stop_loss=None, take_profit=None):
    max_retries = 3
    for attempt in range(max_retries):
        result = place_order(symbol, side, quantity, stop_loss, take_profit)
        if result:
            return result
        log(f"Tentativa {attempt+1} falhou, retry...")
    log(f"Falha ao executar ordem após {max_retries} tentativas")
    return None
```

4️⃣ indicators.py

```
import pandas as pd
import ta

def calculate_indicators(df):
    df['EMA10'] = df['close'].ewm(span=10, adjust=False).mean()
    df['EMA50'] = df['close'].ewm(span=50, adjust=False).mean()
    df['RSI'] = ta.momentum.RSIIndicator(df['close'], window=14).rsi()
    macd = ta.trend.MACD(df['close'])
    df['MACD'] = macd.macd()
    df['MACD_SIGNAL'] = macd.macd_signal()
    df['ATR'] = ta.volatility.AverageTrueRange(df['high'], df['low'],
df['close'], window=14).average_true_range()
    df['ADX'] = ta.trend.ADXIndicator(df['high'], df['low'], df['close'],
window=14).adx()
    return df
```

5️⃣ strategy.py

```
def signal_generator(df):
    ema_signal = df['EMA10'].iloc[-1] > df['EMA50'].iloc[-1]
    rsi_signal = df['RSI'].iloc[-1] < 70
    macd_signal = df['MACD'].iloc[-1] > df['MACD_SIGNAL'].iloc[-1]
    adx_signal = df['ADX'].iloc[-1] > 20
    signal = 'HOLD'
    if ema_signal and rsi_signal and macd_signal and adx_signal:
        signal = 'BUY'
    elif not ema_signal and df['RSI'].iloc[-1] > 30 and df['MACD'].iloc[-1] <
df['MACD_SIGNAL'].iloc[-1] and adx_signal:
        signal = 'SELL'
    return signal
```

6️⃣risk_manager.py

```
def calculate_position_size(balance, atr, risk_percent=0.01):
    risk_amount = balance * risk_percent
    quantity = risk_amount / atr
    return max(quantity, 0.001)
```

7️⃣trading_pair.py

```
import time
from indicators import calculate_indicators
from strategy import signal_generator
from trader import safe_order, get_klines
from risk_manager import calculate_position_size
from utils import log, get_balance

class TradingPair:
    def __init__(self, symbol, client):
        self.symbol = symbol
        self.client = client
        self.position = None
        self.last_signal = None
        self.atr = None
        self.balance = None

    def update_data(self, interval='1m'):
        df = get_klines(self.symbol, interval)
        if df is None:
            return None
        df = calculate_indicators(df)
        self.atr = df['ATR'].iloc[-1]
        return df

    def execute_trade(self, signal):
```

```
        self.balance = get_balance(self.client)
        quantity = calculate_position_size(self.balance, self.atr)
        if signal in ['BUY','SELL']:
            safe_order(self.symbol, signal, quantity, stop_loss=0.01,
take_profit=0.02)
            self.position = signal
            self.last_signal = signal

    def trade_loop(self):
        while True:
            df = self.update_data()
            if df is None:
                time.sleep(5)
                continue
            signal = signal_generator(df)
            if signal != self.last_signal:
                self.execute_trade(signal)
            time.sleep(60)
```

8️⃣ **bot_core.py**

```
from concurrent.futures import ThreadPoolExecutor
from trading_pair import TradingPair
from trader import client
import config
from utils import log

class BotCore:
    def __init__(self):
        self.pairs = [TradingPair(symbol, client) for symbol in
config.SYMBOLS]

    def start(self):
        log("Iniciando bot modular multi-par")
        with ThreadPoolExecutor(max_workers=config.MAX_THREADS) as executor:
            for pair in self.pairs:
                executor.submit(pair.trade_loop)
```

9️⃣ **main.py**

```
from bot_core import BotCore

if __name__ == '__main__':
    bot = BotCore()
    bot.start()
```

✅Como gerar o ZIP

1. Crie uma pasta, por exemplo `binance_bot_modular`.
2. Salve cada trecho acima como arquivo `.py` com o mesmo nome.
3. No Windows ou Linux, clique com o botão direito na pasta → “Enviar para > Pasta compactada (ZIP)” ou use o comando `zip` no terminal.

Isso criará **um arquivo ZIP completo** pronto para abrir no PyCharm.

Se você quiser, posso gerar **um script Python que cria automaticamente todos os arquivos e compacta em ZIP** para você só executar e já ter o bot pronto.

Quer que eu faça isso?