Trading app :

1. Project structure:

```
forex_ai_bot/
├── config.py
├── main.py
├── broker_oanda.py
├── strategy.py
├── risk.py
├── news_filter.py
├── ai_explainer.py
├── data_feed.py
├── logger.py
└── backtest.py
```

2. `config.py`

Central place for settings and API keys (you'll fill in secrets yourself).

```
# config.py

OANDA_API_KEY = "YOUR_OANDA_API_KEY_HERE"
OANDA_ACCOUNT_ID = "YOUR_OANDA_ACCOUNT_ID_HERE"
OANDA_API_URL = "https://api-fxpractice.oanda.com"  # practice endpoint

TRADE_INSTRUMENTS = ["EUR_USD", "GBP_USD", "USD_JPY"]

RISK_PER_TRADE = 0.01        # 1% of account
MAX_DAILY_LOSS = 0.05        # 5% of account
MAX_OPEN_TRADES = 3

NEWS_API_URL = "https://example-economic-calendar.com/api"  # placeholder
HIGH_IMPACT_ONLY = True

TIMEFRAME = "M15"            # 15-minute candles
```

3. `broker_oanda.py`

```
# broker_oanda.py

import requests
from config import OANDA_API_KEY, OANDA_ACCOUNT_ID, OANDA_API_URL

HEADERS = {
    "Authorization": f"Bearer {OANDA_API_KEY}",
    "Content-Type": "application/json"
}

def get_account_balance():
    url = f"{OANDA_API_URL}/v3/accounts/{OANDA_ACCOUNT_ID}"
    r = requests.get(url, headers=HEADERS)
    r.raise_for_status()
    data = r.json()
    return float(data["account"]["balance"])

def place_market_order(instrument, units, side):
    url = f"{OANDA_API_URL}/v3/accounts/{OANDA_ACCOUNT_ID}/orders"
    units = units if side == "buy" else -units
    payload = {
        "order": {
            "instrument": instrument,
            "units": str(units),
            "type": "MARKET",
            "timeInForce": "FOK",
            "positionFill": "DEFAULT"
        }
    }
    r = requests.post(url, json=payload, headers=HEADERS)
    r.raise_for_status()
    return r.json()
```

4. News_filter.py

```
# news_filter.py

import datetime as dt

def is_high_impact_news_day(instrument: str) -> bool:
    """
    Placeholder: in production, call a real economic calendar API
    and check for high-impact events affecting the instrument's
currencies.
    """
    today = dt.date.today()
    # TODO: replace with real API call + logic
    # For now, pretend Fridays are high-impact days:
    if today.weekday() == 4:  # 0=Mon ... 4=Fri
        return True
    return False
```

5. risk.py

```
# risk.py

from config import RISK_PER_TRADE, MAX_DAILY_LOSS, MAX_OPEN_TRADES
from broker_oanda import get_account_balance

def calculate_position_size(stop_loss_pips: float, pip_value_per_unit:
float) -> int:
```

```
    balance = get_account_balance()
    risk_amount = balance * RISK_PER_TRADE
    if stop_loss_pips <= 0 or pip_value_per_unit <= 0:
        return 0
    units = risk_amount / (stop_loss_pips * pip_value_per_unit)
    return int(units)

def can_open_new_trade(current_open_trades: int, current_daily_loss:
float) -> bool:
    if current_open_trades >= MAX_OPEN_TRADES:
        return False
    if current_daily_loss <= -MAX_DAILY_LOSS:
        return False
    return True
```

6.

```
# strategy.py

from news_filter import is_high_impact_news_day

def generate_signal(instrument: str, candles) -> str:
    """
    candles: list of OHLC data, newest last.
    Returns: "buy", "sell", or "hold".
    """

    if is_high_impact_news_day(instrument):
        return "hold"

    if len(candles) < 50:
        return "hold"

    closes = [c["close"] for c in candles]
    short_ma = sum(closes[-20:]) / 20
    long_ma = sum(closes[-50:]) / 50
```

```
    if short_ma > long_ma:
        return "buy"
    elif short_ma < long_ma:
        return "sell"
    else:
        return "hold"
```

7. Ai_explainer.py

```
# ai_explainer.py

def explain_decision(instrument: str, signal: str, context: dict) ->
str:
    if signal == "hold":
        if context.get("news_block", False):
            return f"No trade on {instrument}: high-impact news day
filter is active."
        return f"No clear edge on {instrument}: strategy indicates
hold."

    direction = "long" if signal == "buy" else "short"
    return (
        f"Opening a {direction} position on {instrument} based on
moving-average trend signal. "
        f"Short-term trend: {context.get('short_ma')}, long-term
trend: {context.get('long_ma')}."
    )
```

8. main.py

```
# main.py

import time
from config import TRADE_INSTRUMENTS, TIMEFRAME
from data_feed import get_recent_candles
from strategy import generate_signal
from risk import calculate_position_size, can_open_new_trade
from broker_oanda import place_market_order
from ai_explainer import explain_decision
from news_filter import is_high_impact_news_day

def run_bot():
    open_trades = 0
    daily_pnl = 0.0  # placeholder; in real app, compute from closed
trades

    while True:
        for instrument in TRADE_INSTRUMENTS:
            candles = get_recent_candles(instrument, TIMEFRAME)
            signal = generate_signal(instrument, candles)

            context = {
                "news_block": is_high_impact_news_day(instrument),
                # you can add MA values here if you compute them in
strategy
            }

            if signal == "hold":
                print(explain_decision(instrument, signal, context))
                continue

            if not can_open_new_trade(open_trades, daily_pnl):
                print(f"Risk limits reached. No new trades on
{instrument}.")
                continue
```

```
            # Placeholder values for sizing:
            stop_loss_pips = 20
            pip_value_per_unit = 0.0001  # simplified

            units = calculate_position_size(stop_loss_pips,
pip_value_per_unit)
            if units <= 0:
                print(f"Position size too small for {instrument}.
Skipping.")
                continue

            side = "buy" if signal == "buy" else "sell"
            resp = place_market_order(instrument, units, side)
            open_trades += 1

            print(explain_decision(instrument, signal, context))
            print(f"Order response: {resp}")

        time.sleep(60)  # wait before next cycle

if __name__ == "__main__":
    run_bot()
```