Project Files Dump

## CopyFilesContentToWordDoc.py

import os  
from docx import Document  
import logging  
  
# === CONFIGURATION ===  
root\_dir = r"D:\AlphaNivesh\ANQuant"  
allowed\_extensions = {'.py', '.yaml', '.yml', '.rs', '.toml'}  
output\_file = "project\_files\_dump.docx"  
exclude\_dirs = {'.git', '.venv', '\_\_pycache\_\_', 'site-packages', 'build', 'dist', 'node\_modules'}  
  
# === SETUP LOGGING ===  
logging.basicConfig(  
 level=logging.INFO,  
 format="%(asctime)s [%(levelname)s] %(message)s"  
)  
  
logging.info("🚀 Starting file extraction and Word document creation...")  
logging.info(f"📁 Scanning root directory: {root\_dir}")  
  
# === CREATE DOCUMENT ===  
doc = Document()  
doc.add\_heading("Project Files Dump", level=0)  
  
file\_count = 0  
error\_count = 0  
  
# === WALK THROUGH FILES ===  
for dirpath, dirnames, filenames in os.walk(root\_dir):  
 # Modify dirnames in-place to skip excluded folders  
 dirnames[:] = [d for d in dirnames if d not in exclude\_dirs and not d.startswith('.')]  
  
 for file in filenames:  
 \_, ext = os.path.splitext(file)  
 if ext.lower() in allowed\_extensions:  
 full\_path = os.path.join(dirpath, file)  
 rel\_path = os.path.relpath(full\_path, root\_dir)  
  
 logging.info(f"📄 Processing file: {rel\_path}")  
 doc.add\_heading(rel\_path, level=2)  
  
 try:  
 with open(full\_path, 'r', encoding='utf-8') as f:  
 content = f.read()  
 doc.add\_paragraph(content, style='Normal')  
 file\_count += 1  
 except Exception as e:  
 logging.error(f"❌ Failed to read file {rel\_path}: {e}")  
 doc.add\_paragraph(f"[Error reading file: {e}]")  
 error\_count += 1  
  
# === SAVE DOC ===  
try:  
 doc.save(output\_file)  
 logging.info(f"✅ Word document saved successfully as: {output\_file}")  
 logging.info(f"📊 Total files processed: {file\_count}, Errors: {error\_count}")  
except Exception as e:  
 logging.error(f"❌ Failed to save the Word document: {e}")

## test\_multi\_broker.py

from src.py.util.config\_loader import load\_config  
from src.py.core.adapters import get\_adapters  
import asyncio  
import yaml  
  
  
async def main():  
 config = load\_config("config/config.yaml")  
 adapters = get\_adapters(config)  
  
 # Load watchlists  
 with open(config['global']['markets']['india']['watchlists']['meanhunter'], 'r') as f:  
 india\_watchlist = [stock['tradingsymbol'] for stock in yaml.safe\_load(f)['stocks']]  
 with open(config['global']['markets']['usa']['watchlists']['xyzstrategy'], 'r') as f:  
 usa\_watchlist = [stock['tradingsymbol'] for stock in yaml.safe\_load(f)['stocks']]  
  
 await adapters['india']['angelone'].connect()  
 await adapters['usa']['interactive\_brokers'].connect()  
 await adapters['india']['angelone'].subscribe\_to\_ticks(india\_watchlist)  
 await adapters['usa']['interactive\_brokers'].subscribe\_to\_ticks(usa\_watchlist)  
 await asyncio.sleep(60)  
 await adapters['india']['angelone'].disconnect()  
 await adapters['usa']['interactive\_brokers'].disconnect()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 asyncio.run(main())

## config\config.yaml

global:  
###################################################################################################  
# Logging Configurations #  
###################################################################################################  
 logging\_config: config/logging.yaml # Reference to dedicated logging config  
###################################################################################################  
  
###################################################################################################  
# Broker Configurations #  
###################################################################################################  
 brokers:  
 active\_brokers:  
 india:  
 - angelone  
 #usa:  
 # - interactive\_brokers  
 market: india  
###################################################################################################  
  
###################################################################################################  
# Kafka Configurations #  
###################################################################################################  
 kafka:  
 brokers: localhost:9092  
 topics:  
 india: nse\_ticks  
 #nyse: nyse\_ticks  
 ohlcv\_1min: ohlcv\_1min  
 ohlcv\_5min: ohlcv\_5min  
 ohlcv\_15min: ohlcv\_15min # Added for timeframe flexibility  
 ohlcv\_30min: ohlcv\_30min  
 ohlcv\_1hr: ohlcv\_1hr # Added for timeframe flexibility  
 signals: signals  
 trades: trades  
 nse\_ticks\_partitions: 8 # Supports up to 1000 stocks  
 #nyse\_ticks\_partitions: 8  
 ohlcv\_1min\_partitions: 4  
 ohlcv\_5min\_partitions: 4  
 ohlcv\_15min\_partitions: 4  
 ohlcv\_30min\_partitions: 4  
 ohlcv\_1hr\_partitions: 4  
 signals\_partitions: 4  
 trades\_partitions: 4  
###################################################################################################  
  
###################################################################################################  
# Redis Configurations #  
###################################################################################################  
 redis:  
 host: localhost  
 port: 6379  
 # password: secure\_password # Add for production  
###################################################################################################  
  
###################################################################################################  
# Database Configurations #  
###################################################################################################  
 database:  
 host: localhost  
 port: 5432  
 dbname: ANQuantDB  
 user: anquant # Replace  
 password: 078692 # Replace  
###################################################################################################  
  
###################################################################################################  
# Vault Configurations #  
###################################################################################################  
 vault:  
 enabled: false  
 url: http://vault:8200  
 # token: your\_vault\_token # Add for production  
###################################################################################################  
  
###################################################################################################  
# General Configurations #  
###################################################################################################  
 offline\_mode: false  
###################################################################################################  
  
###################################################################################################  
# Historical Data Configurations #  
###################################################################################################  
 historical\_data:  
 timeframes: ["1min", "5min", "15min", "30min", "1hr"] # Added 15min, 1hr for flexibility  
 lookback\_candles: 60 # 60 candles for pre-fetch and startup  
###################################################################################################  
  
###################################################################################################  
# Strategy Configurations #  
###################################################################################################  
 strategies:  
 - name: meanhunter  
 strategy\_file: config/strategies/meanhunter\_strategy.yaml # Points to detailed strategy config  
 watchlist: config/markets/india/watchlists/meanhunter.yaml  
 timeframe: "5min" # Default/validation, overridden by strategy YAML  
 market\_params: # Market-specific conditions for reusability  
 india:  
 volume\_threshold: "avg\_volume\_20 \* 1.5"  
###################################################################################################  
  
###################################################################################################  
# Market Configurations #  
###################################################################################################  
 markets:  
 india:  
 watchlists:  
 master: config/markets/india/watchlists/master.yaml # Complete stock universe (e.g., 500 or 1000 stocks)  
 meanhunter: config/markets/india/watchlists/meanhunter.yaml  
 strategy2: config/markets/india/watchlists/strategy2.yaml # Placeholder for future expansion  
 brokers:  
 angelone:  
 symbols: config/markets/india/brokers/symbol\_mappings/angelone.yaml  
 credentials: config/secrets/india/angelone\_cred.yaml  
 #usa:  
 # watchlists:  
 # master: config/markets/usa/watchlists/master.yaml  
 # xyzstrategy: config/markets/usa/watchlists/xyzstrategy.yaml  
 # brokers:  
 # interactive\_brokers:  
 # symbols: config/markets/usa/brokers/symbol\_mappings/interactive\_brokers.yaml  
 # credentials: config/secrets/usa/ib\_cred.yaml  
###################################################################################################  
  
###################################################################################################  
# Audit Configurations #  
###################################################################################################  
 audit:  
 redis\_channel\_prefix: "signals:audit" # For audit trails (e.g., signals:audit:meanhunter)  
 persist\_to\_db: true # Store audit trails in PostgreSQL  
 db\_table: audit\_trails # Table for audit trail storage  
###################################################################################################

## config\logging.yaml

general:  
 directory: logs  
 filename\_template: "{name}\_{date}.log"  
 level: DEBUG  
 rotation: 10 MB  
 retention: 30 days  
 compression: zip  
 format: "{time:YYYY-MM-DD HH:mm:ss} | {level} | {name} | {message}"  
 stdout\_format: "<green>{time:YYYY-MM-DD HH:mm:ss}</green> | <level>{level}</level> | <cyan>{name}</cyan> | <level>{message}</level>"  
messaging:  
 directory: logs  
 filename\_template: "{name}\_{date}.log"  
 level: TRACE  
 rotation: 10 MB  
 retention: 30 days  
 compression: zip  
 format: "{time:YYYY-MM-DD HH:mm:ss} | {level} | {name} | {message}"  
 stdout\_format: "<green>{time:YYYY-MM-DD HH:mm:ss}</green> | <level>{level}</level> | <cyan>{name}</cyan> | <level>{message}</level>"  
kafka:  
 directory: logs  
 filename\_template: "kafka/{name}\_{date}.log"  
 level: TRACE  
 rotation: 10 MB  
 retention: 30 days  
 compression: zip  
 format: "{time:YYYY-MM-DD HH:mm:ss} | {level} | {name} | {message}"  
 stdout\_format: "<green>{time:YYYY-MM-DD HH:mm:ss}</green> | <level>{level}</level> | <cyan>{name}</cyan> | <level>{message}</level>"

## config\markets\india\brokers\symbol\_mappings\angelone.yaml

JISLJALEQS-EQ: "10397"  
MARKSANS-EQ: "10579"  
PNB-EQ: "10666"  
NUCLEUS-EQ: "10791"  
CANBK-EQ: "10794"  
GODREJIND-EQ: "10925"  
DIVISLAB-EQ: "10940"  
IDFCFIRSTB-EQ: "11184"  
APOLLO-EQ: "1134"  
POONAWALLA-EQ: "11403"  
PILANIINVS-EQ: "11445"  
EMIL-EQ: "11530"  
COFORGE-EQ: "11543"  
JBMA-EQ: "11655"  
JSWSTEEL-EQ: "11723"  
WELCORP-EQ: "11821"  
SSWL-EQ: "11829"  
JSWHL-EQ: "11880"  
YESBANK-EQ: "11915"  
ICIL-EQ: "11987"  
KAYNES-EQ: "12092"  
GSFC-EQ: "1247"  
CASTROLIND-EQ: "1250"  
TEAMLEASE-EQ: "12716"  
KRISHANA-EQ: "12847"  
360ONE-EQ: "13061"  
PVRINOX-EQ: "13147"  
VOLTAMP-EQ: "13577"  
AARTIPHARM-EQ: "13868"  
BANCOINDIA-EQ: "13880"  
TIMKEN-EQ: "14198"  
HUBTOWN-EQ: "14203"  
HINDZINC-EQ: "1424"  
IDEA-EQ: "14366"  
PAGEIND-EQ: "14413"  
ICRA-EQ: "14523"  
KPRMILL-EQ: "14912"  
CIEINDIA-EQ: "14937"  
CENTUM-EQ: "14982"  
TARIL-EQ: "15174"  
MANKIND-EQ: "15380"  
TARC-EQ: "1581"  
INFY-EQ: "1594"  
BECTORFOOD-EQ: "1628"  
ITC-EQ: "1660"  
THYROCARE-EQ: "17032"  
CYIENTDLM-EQ: "17187"  
VINATIORGA-EQ: "17364"  
ADANIPOWER-EQ: "17388"  
NHPC-EQ: "17400"  
RTNPOWER-EQ: "17520"  
QUESS-EQ: "17704"  
YATHARTH-EQ: "17738"  
JSWENERGY-EQ: "17869"  
COHANCE-EQ: "17945"  
PERSISTENT-EQ: "18365"  
POWERINDIA-EQ: "18457"  
NUVAMA-EQ: "18721"  
JSWINFRA-EQ: "19020"  
KIRLOSIND-EQ: "19025"  
KIOCL-EQ: "19126"  
ROSSARI-EQ: "19410"  
DEEPAKNTR-EQ: "19943"  
IREDA-EQ: "20261"  
ASHAPURMIN-EQ: "203"  
RAMCOCEM-EQ: "2043"  
MASTEK-EQ: "2124"  
BDL-EQ: "2144"  
DIXON-EQ: "21690"  
MAXHEALTH-EQ: "22377"  
MANORAMA-EQ: "10227"  
SYNGENE-EQ: "10243"  
FINPIPE-EQ: "1041"  
POWERMECH-EQ: "10473"  
UCOBANK-EQ: "11223"  
IGL-EQ: "11262"  
UPL-EQ: "11287"  
LUXIND-EQ: "11301"  
VAIBHAVGBL-EQ: "11364"  
GOKEX-EQ: "11778"  
AMBER-EQ: "1185"  
NH-EQ: "11840"  
HNDFDS-EQ: "12173"  
TRIVENI-EQ: "13081"  
KEC-EQ: "13260"  
GALLANTT-EQ: "13337"  
SDBL-EQ: "1338"  
RATNAMANI-EQ: "13451"  
NSLNISP-EQ: "14180"  
PGIL-EQ: "14260"  
SEQUENT-EQ: "14296"  
PFC-EQ: "14299"  
ASTRAL-EQ: "14418"  
CSBBANK-EQ: "14966"  
BAJAJELEC-EQ: "15034"  
AVALON-EQ: "15058"  
RELIGARE-EQ: "15068"  
ASTERDM-EQ: "1508"  
DHANI-EQ: "15384"  
TITAGARH-EQ: "15414"  
INGERRAND-EQ: "1597"  
PRINCEPIPE-EQ: "16045"  
IOC-EQ: "1624"  
CROMPTON-EQ: "17094"  
NESTLEIND-EQ: "17963"  
SBICARD-EQ: "17971"  
THANGAMAYL-EQ: "18118"  
JIOFIN-EQ: "18143"  
DIACABS-EQ: "18543"  
RRKABEL-EQ: "18566"  
PNBHOUSING-EQ: "18908"  
SFL-EQ: "19184"  
ARVIND-EQ: "193"  
KSB-EQ: "1949"  
ASHOKA-EQ: "20182"  
CGCL-EQ: "20329"  
INDIASHLTR-EQ: "20556"  
ERIS-EQ: "21154"  
ACC-EQ: "22"  
EPACK-EQ: "22463"  
BANDHANBNK-EQ: "2263"  
FACT-EQ: "1008"  
FEDERALBNK-EQ: "1023"  
UNIONBANK-EQ: "10753"  
SHK-EQ: "11212"  
JSL-EQ: "11236"  
BIOCON-EQ: "11373"  
APARINDS-EQ: "11491"  
WESTLIFE-EQ: "11580"  
LALPATHLAB-EQ: "11654"  
PATELENG-EQ: "11699"  
JPPOWER-EQ: "11763"  
63MOONS-EQ: "11868"  
GRANULES-EQ: "11872"  
GENUSPOWER-EQ: "11905"  
GRASIM-EQ: "1232"  
M&MFIN-EQ: "13285"  
ALLCARGO-EQ: "13501"  
PFOCUS-EQ: "13496"  
EMAMILTD-EQ: "13517"  
GMRAIRPORT-EQ: "13528"  
TECHM-EQ: "13538"  
FIEMIND-EQ: "13710"  
GESHIP-EQ: "13776"  
HINDUNILVR-EQ: "1394"  
NFL-EQ: "13925"  
GODREJAGRO-EQ: "144"  
SANOFI-EQ: "1442"  
PHOENIXLTD-EQ: "14552"  
AKZOINDIA-EQ: "1467"  
PURVA-EQ: "14922"  
KNRCON-EQ: "15283"  
VGUARD-EQ: "15362"  
NESCO-EQ: "15409"  
GILLETTE-EQ: "1576"  
SUNDROP-EQ: "1663"  
BAJAJ-AUTO-EQ: "16669"  
HGINFRA-EQ: "1672"  
MHRIL-EQ: "17333"  
AHLUCONT-EQ: "17833"  
BAYERCROP-EQ: "17927"  
SBFC-EQ: "18026"  
JUBLFOOD-EQ: "18096"  
KARURVYSYA-EQ: "1838"  
ZAGGLE-EQ: "18608"  
MARATHON-EQ: "18659"  
KSL-EQ: "18889"  
M&M-EQ: "2031"  
OBEROIRLTY-EQ: "20242"  
AZAD-EQ: "20905"  
INNOVACAP-EQ: "21062"  
STARCEMENT-EQ: "21091"  
JYOTICNC-EQ: "21334"  
ABCAPITAL-EQ: "21614"  
ICICIGI-EQ: "21770"  
TEXRAIL-EQ: "21828"  
HFCL-EQ: "21951"  
ENTERO-EQ: "22717"  
SHAREINDIA-EQ: "104"  
RADICO-EQ: "10990"  
PPLPHARMA-EQ: "11571"  
GUFICBIO-EQ: "11606"  
ASTRAMICRO-EQ: "11618"  
IIFL-EQ: "11809"  
SHOPERSTOP-EQ: "11813"  
DCXINDIA-EQ: "11895"  
SWSOLAR-EQ: "12489"  
GUJALKALI-EQ: "1267"  
AMBUJACEM-EQ: "1270"  
ROUTE-EQ: "128"  
GALAXYSURF-EQ: "1315"  
GSPL-EQ: "13197"  
SOLARINDS-EQ: "13332"  
ELECON-EQ: "13643"  
DCBBANK-EQ: "13725"  
FLUOROCHEM-EQ: "13750"  
NAUKRI-EQ: "13751"  
BBL-EQ: "13761"  
LTFOODS-EQ: "13816"  
SAGCEM-EQ: "14068"  
NETWORK18-EQ: "14111"  
HSCL-EQ: "14334"  
BALAMINES-EQ: "14501"  
DLF-EQ: "14732"  
IDBI-EQ: "1476"  
KDDL-EQ: "14908"  
EDELWEISS-EQ: "15119"  
COLPAL-EQ: "15141"  
INDIAGLYCO-EQ: "1521"  
ECLERX-EQ: "15179"  
TATAINVEST-EQ: "1621"  
APOLLOTYRE-EQ: "163"  
JBCHEPHARM-EQ: "1726"  
BLS-EQ: "17279"  
LLOYDSENGG-EQ: "17801"  
GODREJPROP-EQ: "17875"  
ABBOTINDIA-EQ: "17903"  
KPIL-EQ: "1814"  
MANINFRA-EQ: "18226"  
LGBBROSLTD-EQ: "18321"  
JLHL-EQ: "18553"  
SAMHI-EQ: "18614"  
STYRENIX-EQ: "19"  
MANAPPURAM-EQ: "19061"  
PRICOLLTD-EQ: "19631"  
LMW-EQ: "1979"  
JWL-EQ: "20224"  
RESPONIND-EQ: "20323"  
COALINDIA-EQ: "20374"  
PSPPROJECT-EQ: "20877"  
PSB-EQ: "21001"  
ASHOKLEY-EQ: "212"  
SIS-EQ: "21501"  
IEX-EQ: "220"  
MRPL-EQ: "2283"  
AIIL-EQ: "23553"  
AWFIS-EQ: "23864"  
CEIGALL-EQ: "24742"  
ONGC-EQ: "2475"  
PREMIERENE-EQ: "25049"  
TDPOWERSYS-EQ: "25178"  
WAAREEENER-EQ: "25907"  
PCBL-EQ: "2649"  
SWIGGY-EQ: "27066"  
BLACKBUCK-EQ: "27144"  
JUBLINGREA-EQ: "2783"  
EASEMYTRIP-EQ: "2792"  
VMM-EQ: "27969"  
KITEX-EQ: "28899"  
TVSHLTD-EQ: "29008"  
CARERATING-EQ: "29113"  
ONESOURCE-EQ: "29224"  
SAIL-EQ: "2963"  
HAL-EQ: "2303"  
HEIDELBERG-EQ: "2316"  
ASIANPAINT-EQ: "236"  
RELAXO-EQ: "24225"  
EMCURE-EQ: "24398"  
NOCIL-EQ: "2442"  
LTF-EQ: "24948"  
EUREKAFORB-EQ: "25162"  
PIDILITIND-EQ: "2664"  
KPEL-EQ: "27079"  
EIEL-EQ: "27213"  
RTNINDIA-EQ: "27297"  
LXCHEM-EQ: "2841"  
RCF-EQ: "2866"  
QPOWER-EQ: "29711"  
ORIENTELEC-EQ: "2972"  
SUDARSCHEM-EQ: "3327"  
TATAELXSI-EQ: "3411"  
HONAUT-EQ: "3417"  
THERMAX-EQ: "3475"  
TITAN-EQ: "3506"  
MARICO-EQ: "4067"  
SHRIRAMFIN-EQ: "4306"  
SBCL-EQ: "4656"  
BLUEDART-EQ: "495"  
SHARDACROP-EQ: "4992"  
GMDCLTD-EQ: "5204"  
ALIVUS-EQ: "5265"  
AAVAS-EQ: "5385"  
CARTRADE-EQ: "5407"  
SERVOTECH-EQ: "5507"  
CANFINHOME-EQ: "583"  
SANSERA-EQ: "5751"  
GRAPHITE-EQ: "592"  
NATIONALUM-EQ: "6364"  
CIPLA-EQ: "694"  
HEMIPROP-EQ: "701"  
GANECOS-EQ: "6944"  
JTLIND-EQ: "7287"  
AVANTIFEED-EQ: "7936"  
DCMSHRIRAM-EQ: "811"  
CHALET-EQ: "8546"  
ANUP-EQ: "9014"  
METROPOLIS-EQ: "9581"  
JAYNECOIND-EQ: "2331"  
NEULANDLAB-EQ: "2406"  
RPEL-EQ: "24735"  
SHAKTIPUMP-EQ: "25574"  
JISLDVREQS-EQ: "25684"  
PFIZER-EQ: "2643"  
SWANENERGY-EQ: "27095"  
NIVABUPA-EQ: "27097"  
AUROPHARMA-EQ: "275"  
GICRE-EQ: "277"  
RALLIS-EQ: "2816"  
IKS-EQ: "28125"  
CRAFTSMAN-EQ: "2854"  
INDUSTOWER-EQ: "29135"  
MOSCHIP-EQ: "29459"  
KALYANKJIL-EQ: "2955"  
NAZARA-EQ: "2987"  
PRABHA-EQ: "30134"  
JINDALSAW-EQ: "3024"  
VEDL-EQ: "3063"  
TIINDIA-EQ: "312"  
HONDAPOWER-EQ: "3138"  
SUBROS-EQ: "3324"  
BALMLAWRIE-EQ: "338"  
SUNFLAG-EQ: "3348"  
SURYAROSNI-EQ: "3375"  
VIPIND-EQ: "3703"  
BATAINDIA-EQ: "371"  
TATACOMM-EQ: "3721"  
ZEEL-EQ: "3812"  
BERGEPAINT-EQ: "404"  
NATCOPHARM-EQ: "3918"  
GULFOILLUB-EQ: "4391"  
MPHASIS-EQ: "4503"  
EPIGRAL-EQ: "5382"  
CESC-EQ: "628"  
LATENTVIEW-EQ: "6818"  
AARTIIND-EQ: "7"  
MEDPLUS-EQ: "7254"  
NPST-EQ: "756324"  
OSWALPUMPS-EQ: "756802"  
NBCC-EQ: "31415"  
LODHA-EQ: "3220"  
ISGEC-EQ: "3329"  
TORNTPHARM-EQ: "3518"  
VOLTAS-EQ: "3718"  
WIPRO-EQ: "3787"  
BEL-EQ: "383"  
CAPLIPOINT-EQ: "3906"  
BANKBARODA-EQ: "4668"  
3MINDIA-EQ: "474"  
SHYAMMETL-EQ: "4693"  
BIRLACORPN-EQ: "480"  
BOMDYEING-EQ: "513"  
APTUS-EQ: "5435"  
ASTRAZEN-EQ: "5610"  
INTELLECT-EQ: "5926"  
SOUTHBANK-EQ: "5948"  
STAR-EQ: "7374"  
RAJESHEXPO-EQ: "7401"  
WOCKPHARMA-EQ: "7506"  
WAAREERTL-EQ: "756038"  
CRISIL-EQ: "757"  
AEGISVOPAK-EQ: "757336"  
DIGITIDE-EQ: "757515"  
KTKBANK-EQ: "8054"  
DALBHARAT-EQ: "8075"  
DEEPAKFERT-EQ: "827"  
EIDPARRY-EQ: "916"  
ARVINDFASN-EQ: "9111"  
ELECTCAST-EQ: "928"  
SKIPPER-EQ: "9428"  
DELHIVERY-EQ: "9599"  
HAVELLS-EQ: "9819"  
NEOGEN-EQ: "9917"  
GOPAL-EQ: "23066"  
SANDHAR-EQ: "2397"  
PIIND-EQ: "24184"  
VADILALIND-EQ: "24196"  
VSTTILLERS-EQ: "24292"  
BANSALWIRE-EQ: "24386"  
OLAELEC-EQ: "24777"  
ADANIENT-EQ: "25"  
WINDMACHIN-EQ: "24969"  
RAYMONDLSL-EQ: "25073"  
PGEL-EQ: "25358"  
POLYMED-EQ: "25718"  
LLOYDSENT-EQ: "25807"  
AVL-EQ: "25984"  
PRECWIRE-EQ: "2717"  
NTPCGREEN-EQ: "27176"  
SANATHAN-EQ: "28805"  
AGARWALEYE-EQ: "29452"  
E2E-EQ: "8937"  
DPABHUSHAN-EQ: "936"  
RAINBOW-EQ: "9408"  
LICI-EQ: "9480"  
WONDERLA-EQ: "3002"  
RHIM-EQ: "31163"  
BALKRISIND-EQ: "335"  
SUNDRMFAST-EQ: "3345"  
BALRAMCHIN-EQ: "341"  
BASF-EQ: "368"  
RITES-EQ: "3761"  
AEGISLOG-EQ: "40"  
VARROC-EQ: "3857"  
HERITGFOOD-EQ: "4598"  
INDRAMEDCO-EQ: "4751"  
ICICIBANK-EQ: "4963"  
CLEAN-EQ: "5049"  
GRINFRA-EQ: "5054"  
KPIGREEN-EQ: "5108"  
UTIAMC-EQ: "527"  
ITDCEM-EQ: "5622"  
CUB-EQ: "5701"  
BHAGCHEM-EQ: "6164"  
CHAMBLFERT-EQ: "637"  
NYKAA-EQ: "6545"  
BSOFT-EQ: "6994"  
ANANDRATHI-EQ: "7145"  
DATAPATTNS-EQ: "7358"  
THELEELA-EQ: "757014"  
DALMIASUG-EQ: "781"  
ALEMBICLTD-EQ: "79"  
AWL-EQ: "8110"  
GMRP&UI-EQ: "8529"  
NLCINDIA-EQ: "8585"  
PRIVISCL-EQ: "8825"  
HIKAL-EQ: "9668"  
EMUDHRA-EQ: "9756"  
ARE&M-EQ: "100"  
SCHAEFFLER-EQ: "1011"  
ZENSARTECH-EQ: "1076"  
SYRMA-EQ: "10793"  
GABRIEL-EQ: "1085"  
WELSPUNLIV-EQ: "11253"  
JAIBALAJI-EQ: "11256"  
LT-EQ: "11483"  
NEWGEN-EQ: "1164"  
GLAND-EQ: "1186"  
JKPAPER-EQ: "11860"  
AURIONPRO-EQ: "12022"  
ACI-EQ: "12024"  
HEROMOTOCO-EQ: "1348"  
JMFINANCIL-EQ: "13637"  
AGI-EQ: "1412"  
TIIL-EQ: "14223"  
V2RETAIL-EQ: "14766"  
CENTRALBK-EQ: "14894"  
RBA-EQ: "1494"  
MOTILALOFS-EQ: "14947"  
POWERGRID-EQ: "14977"  
ADANIPORTS-EQ: "15083"  
KOLTEPATIL-EQ: "15124"  
HCG-EQ: "15555"  
UBL-EQ: "16713"  
PATANJALI-EQ: "17029"  
PREMEXPLN-EQ: "17397"  
POKARNA-EQ: "17651"  
ALOKINDS-EQ: "17675"  
SARDAEN-EQ: "17758"  
DBREALTY-EQ: "18124"  
RBLBANK-EQ: "18391"  
ENDURANCE-EQ: "18822"  
VBL-EQ: "18921"  
ITDC-EQ: "19299"  
BSE-EQ: "19585"  
TRENT-EQ: "1964"  
HONASA-EQ: "19813"  
ASKAUTOLTD-EQ: "20152"  
WABAG-EQ: "20188"  
IRFC-EQ: "2029"  
TATATECH-EQ: "20293"  
INDIGOPNTS-EQ: "2048"  
HOMEFIRST-EQ: "2056"  
DOMS-EQ: "20551"  
JINDWORLD-EQ: "20642"  
VTL-EQ: "2073"  
HUDCO-EQ: "20825"  
MAHSCOOTER-EQ: "2085"  
CDSL-EQ: "21174"  
AUBANK-EQ: "21238"  
MEDIASSIST-EQ: "21705"  
SUNDARMHLD-EQ: "2183"  
SBILIFE-EQ: "21808"  
UNITDSPR-EQ: "10447"  
SHARDAMOTR-EQ: "10530"  
KRBL-EQ: "10577"  
FOSECOIND-EQ: "1073"  
NSIL-EQ: "11239"  
GHCL-EQ: "1127"  
PETRONET-EQ: "11351"  
PTC-EQ: "11355"  
WELENT-EQ: "11626"  
GODFRYPHLP-EQ: "1181"  
GOLDIAM-EQ: "11971"  
RENUKA-EQ: "12026"  
JKCEMENT-EQ: "13270"  
HDFCBANK-EQ: "1333"  
CENTURYPLY-EQ: "13305"  
KKCL-EQ: "13381"  
GPIL-EQ: "13409"  
RSYSTEMS-EQ: "13414"  
JKLAKSHMI-EQ: "13491"  
HINDALCO-EQ: "1363"  
IRCTC-EQ: "13611"  
HBLENGINE-EQ: "13966"  
JASH-EQ: "13982"  
LUMAXTECH-EQ: "14014"  
UNOMINDA-EQ: "14154"  
FINCABLES-EQ: "1038"  
ARVSMART-EQ: "10457"  
UFLEX-EQ: "1053"  
MPSLTD-EQ: "10578"  
BHARTIARTL-EQ: "10604"  
MARUTI-EQ: "10999"  
INDIGO-EQ: "11195"  
AFFLE-EQ: "11343"  
ULTRACEMCO-EQ: "11532"  
FORCEMOT-EQ: "11573"  
NTPC-EQ: "11630"  
ALKEM-EQ: "11703"  
BIKAJI-EQ: "11966"  
RUSTOMJEE-EQ: "12219"  
GREAVESCOT-EQ: "1235"  
ABB-EQ: "13"  
IIFLCAPS-EQ: "13072"  
AIAENG-EQ: "13086"  
GRINDWELL-EQ: "13560"  
ACE-EQ: "13587"  
ANANTRAJ-EQ: "13620"  
SOBHA-EQ: "13826"  
HINDPETRO-EQ: "1406"  
SIYSIL-EQ: "14096"  
FSL-EQ: "14304"  
INDIANB-EQ: "14309"  
FORTIS-EQ: "14592"  
INDIACEM-EQ: "1515"  
KIRLPNU-EQ: "15180"  
RAIN-EQ: "15337"  
LINDEINDIA-EQ: "1627"  
IPCALAB-EQ: "1633"  
RPGLIFE-EQ: "16725"  
ZFCVINDIA-EQ: "16915"  
AJMERA-EQ: "17307"  
NETWEB-EQ: "17433"  
ZYDUSWELL-EQ: "17635"  
SUNTECK-EQ: "17641"  
KAJARIACER-EQ: "1808"  
TVSSCS-EQ: "18151"  
TCPLPACK-EQ: "184"  
HPL-EQ: "18679"  
SIGNATURE-EQ: "18743"  
CUMMINSIND-EQ: "1901"  
TI-EQ: "19196"  
MAXESTATES-EQ: "19646"  
GANESHHOUC-EQ: "14339"  
SPARC-EQ: "14788"  
IFBIND-EQ: "1485"  
CERA-EQ: "15039"  
JYOTHYLAB-EQ: "15146"  
CEATLTD-EQ: "15254"  
JKIL-EQ: "15266"  
IRB-EQ: "15313"  
NMDC-EQ: "15332"  
RECLTD-EQ: "15355"  
INFIBEAM-EQ: "16249"  
PTCIL-EQ: "16682"  
SUMICHEM-EQ: "17105"  
SHRIPISTON-EQ: "17186"  
LLOYDSME-EQ: "17313"  
LTIM-EQ: "17818"  
DBCORP-EQ: "17881"  
ADVENZYMES-EQ: "18039"  
CONCORDBIO-EQ: "18060"  
DBL-EQ: "18086"  
LTTS-EQ: "18564"  
BLUEJET-EQ: "19686"  
CELLO-EQ: "19795"  
MASFIN-EQ: "199"  
DMART-EQ: "19913"  
RAMKY-EQ: "20134"  
PRESTIGE-EQ: "20302"  
FEDFINA-EQ: "20322"  
MAHSEAMLES-EQ: "2088"  
HAPPYFORGE-EQ: "20854"  
SUNCLAY-EQ: "20956"  
GODREJCP-EQ: "10099"  
ADANIENSOL-EQ: "10217"  
LUPIN-EQ: "10440"  
GUJGASLTD-EQ: "10599"  
INDIAMART-EQ: "10726"  
MAHABANK-EQ: "11377"  
GIPCL-EQ: "1145"  
GNFC-EQ: "1174"  
FMGOETZE-EQ: "1190"  
MEDANTA-EQ: "11956"  
NITCO-EQ: "13300"  
KEI-EQ: "13310"  
HEG-EQ: "1336"  
KFINTECH-EQ: "13359"  
GREENPANEL-EQ: "13810"  
TANLA-EQ: "13976"  
PITTIENG-EQ: "14134"  
JKTYRE-EQ: "14435"  
EMBDL-EQ: "14450"  
GOKULAGRO-EQ: "14480"  
WEBELSOLAR-EQ: "14602"  
NAVINFLUOR-EQ: "14672"  
TIMETECHNO-EQ: "14707"  
INDHOTEL-EQ: "1512"  
UJJIVANSFB-EQ: "15228"  
GMMPFAUDLR-EQ: "1570"  
IONEXCHANG-EQ: "1630"  
KIRIINDUS-EQ: "16639"  
BAJAJFINSV-EQ: "16675"  
ITI-EQ: "1675"  
REFEX-EQ: "17764"  
NIITMTS-EQ: "17747"  
WHIRLPOOL-EQ: "18011"  
SANDUMA-EQ: "18359"  
ICICIPRULI-EQ: "18652"  
SJVN-EQ: "18883"  
KINGFA-EQ: "18944"  
KOTAKBANK-EQ: "1922"  
BAJAJCON-EQ: "19531"  
GRAVITA-EQ: "20534"  
INOXINDIA-EQ: "20607"  
PENIND-EQ: "20621"  
TCI-EQ: "10580"  
OLECTRA-EQ: "10637"  
OFSS-EQ: "10738"  
TMB-EQ: "10945"  
GARFIBRES-EQ: "1100"  
HARSHA-EQ: "11162"  
RKFORGE-EQ: "11411"  
DATAMATICS-EQ: "11423"  
CCL-EQ: "11452"  
GLAXO-EQ: "1153"  
TCS-EQ: "11536"  
SUPRAJIT-EQ: "11689"  
KANSAINER-EQ: "1196"  
SUZLON-EQ: "12018"  
FIVESTAR-EQ: "12032"  
MBAPL-EQ: "12686"  
SAFARI-EQ: "13035"  
SUNTV-EQ: "13404"  
SKYGOLD-EQ: "13631"  
TORNTPOWER-EQ: "13786"  
REDINGTON-EQ: "14255"  
IFCI-EQ: "1491"  
KSCL-EQ: "14972"  
MANGCHEFER-EQ: "15007"  
BRIGADE-EQ: "15184"  
APOLLOHOSP-EQ: "157"  
GVT&D-EQ: "16783"  
SINDHUTRAD-EQ: "16859"  
SENCO-EQ: "17271"  
OIL-EQ: "17438"  
MGL-EQ: "17534"  
REDTAPE-EQ: "17859"  
HINDCOPPER-EQ: "17939"  
MMTC-EQ: "17957"  
KIRLOSBROS-EQ: "18581"  
EMSLIMITED-EQ: "18593"  
LAURUSLABS-EQ: "19234"  
IMFA-EQ: "19235"  
BOROLTD-EQ: "19401"  
GPPL-EQ: "19731"  
LICHSGFIN-EQ: "1997"  
LUMAXIND-EQ: "2018"  
CHENNPETRO-EQ: "2049"  
JAMNAAUTO-EQ: "20778"  
MOIL-EQ: "20830"  
KIRLOSENG-EQ: "20936"  
TEJASNET-EQ: "21131"  
OPTIEMUS-EQ: "21469"  
PARKHOTELS-EQ: "22649"  
MFSL-EQ: "2142"  
COCHINSHIP-EQ: "21508"  
CHOLAHLDNG-EQ: "21740"  
BOSCHLTD-EQ: "2181"  
MOREPENLAB-EQ: "2259"  
JSFB-EQ: "22663"  
MRF-EQ: "2277"  
MUTHOOTFIN-EQ: "23650"  
GODIGIT-EQ: "23799"  
ARKADE-EQ: "25398"  
KRN-EQ: "25643"  
HYUNDAI-EQ: "25844"  
LEMONTREE-EQ: "2606"  
ACMESOLAR-EQ: "27061"  
RAYMOND-EQ: "2859"  
PCJEWELLER-EQ: "29124"  
AJAXENGG-EQ: "29624"  
HEXT-EQ: "29666"  
ORIENTCEM-EQ: "30089"  
SHANTIGEAR-EQ: "3078"  
INDOSTAR-EQ: "3088"  
BORORENEW-EQ: "3149"  
JTEKTINDIA-EQ: "3237"  
SRF-EQ: "3273"  
SUNDARMFIN-EQ: "3339"  
SUNPHARMA-EQ: "3351"  
SMLISUZU-EQ: "3387"  
TATAPOWER-EQ: "3426"  
BANARISUG-EQ: "350"  
ADANIGREEN-EQ: "3563"  
UNICHEMLAB-EQ: "3579"  
JUBLPHARMA-EQ: "3637"  
VESUVIUS-EQ: "3676"  
VSTIND-EQ: "3724"  
WSTCSTPAPR-EQ: "3799"  
BHARATRAS-EQ: "3834"  
MOTHERSON-EQ: "4204"  
HDFCAMC-EQ: "4244"  
AARTIDRUGS-EQ: "4481"  
ALKYLAMINE-EQ: "4487"  
SONACOMS-EQ: "4684"  
FDC-EQ: "4898"  
IRCON-EQ: "4986"  
ETERNAL-EQ: "5097"  
CIGNITITEC-EQ: "5142"  
INDUSINDBK-EQ: "5258"  
PDSL-EQ: "5264"  
BRITANNIA-EQ: "547"  
J&KBANK-EQ: "5633"  
TECHNOE-EQ: "6445"  
GOCOLORS-EQ: "6964"  
COROMANDEL-EQ: "739"  
SUPRIYA-EQ: "7390"  
ZENTEC-EQ: "7508"  
ELLEN-EQ: "758563"  
IMAGICAA-EQ: "7672"  
SHAILY-EQ: "8727"  
ELGIEQUIP-EQ: "937"  
CAMPUS-EQ: "9362"  
PGHL-EQ: "940"  
ESCORTS-EQ: "958"  
RVNL-EQ: "9552"  
INDGN-EQ: "23693"  
AADHARHFC-EQ: "23729"  
IXIGO-EQ: "24067"  
DHANUKA-EQ: "24409"  
FIRSTCRY-EQ: "24814"  
BAJAJHFL-EQ: "25270"  
MINDACORP-EQ: "25897"  
AFCONS-EQ: "25977"  
TRANSRAILL-EQ: "28714"  
MTNL-EQ: "2294"  
NCC-EQ: "2319"  
AKUMS-EQ: "24715"  
IGIL-EQ: "28378"  
VMART-EQ: "29284"  
PROTEAN-EQ: "29472"  
SAMMAANCAP-EQ: "30125"  
BAJAJHLDNG-EQ: "305"  
SCI-EQ: "3048"  
SIEMENS-EQ: "3150"  
TATACHEM-EQ: "3405"  
BBTC-EQ: "380"  
BHARATFORG-EQ: "422"  
BHEL-EQ: "438"  
HDFCLIFE-EQ: "467"  
GAIL-EQ: "4717"  
DODLA-EQ: "4822"  
MAZDOCK-EQ: "509"  
BPCL-EQ: "526"  
ROLEXRINGS-EQ: "5279"  
DEVYANI-EQ: "5373"  
VIJAYA-EQ: "5585"  
ATGL-EQ: "6066"  
ABREL-EQ: "625"  
PAYTM-EQ: "6705"  
JINDALSTEL-EQ: "6733"  
STARHEALTH-EQ: "7083"  
MAPMYINDIA-EQ: "7227"  
ATHERENERG-EQ: "757645"  
CMSINFO-EQ: "7603"  
MAHLIFE-EQ: "8050"  
RPSGVENT-EQ: "8119"  
CHOICEIN-EQ: "8866"  
POLYCAB-EQ: "9590"  
SPLPETRO-EQ: "9617"  
KPITTECH-EQ: "9683"  
BAJAJHIND-EQ: "308"  
MCX-EQ: "31181"  
ANGELONE-EQ: "324"  
SKFINDIA-EQ: "3186"  
SUPREMEIND-EQ: "3363"  
SWARAJENG-EQ: "3384"  
TATACONSUM-EQ: "3432"  
THOMASCOOK-EQ: "3481"  
TTKPRESTIG-EQ: "3546"  
BEML-EQ: "395"  
NIACL-EQ: "399"  
GREENPLY-EQ: "3987"  
JCHAC-EQ: "4491"  
CONCOR-EQ: "4749"  
HAPPSTMNDS-EQ: "48"  
SAREGAMA-EQ: "4892"  
CYIENT-EQ: "5748"  
VISHNU-EQ: "6908"  
TEGA-EQ: "7105"  
COSMOFIRST-EQ: "742"  
GLENMARK-EQ: "7406"  
BELRISE-EQ: "757102"  
AJANTPHARM-EQ: "8124"  
MANYAVAR-EQ: "8167"  
BLUESTARCO-EQ: "8311"  
DRREDDY-EQ: "881"  
MSTCLTD-EQ: "9356"  
PNCINFRA-EQ: "9385"  
PRUDENT-EQ: "9553"  
TRIDENT-EQ: "9685"  
PARADEEP-EQ: "9741"  
EPL-EQ: "981"  
ETHOSLTD-EQ: "9750"  
HLEGLAS-EQ: "2289"  
NAVNETEDUL-EQ: "2385"  
RAILTEL-EQ: "2431"  
ABDL-EQ: "24308"  
MAITHANALL-EQ: "24538"  
MIDHANI-EQ: "2463"  
PNGJL-EQ: "25312"  
SANOFICONR-EQ: "25222"  
PGHH-EQ: "2535"  
ATUL-EQ: "263"  
POLYPLEX-EQ: "2687"  
PRAJIND-EQ: "2705"  
SAGILITY-EQ: "27052"  
PRSMJOHNSN-EQ: "2739"  
SAILIFE-EQ: "27839"  
VENTIVE-EQ: "28847"  
ABFRL-EQ: "30108"  
SBIN-EQ: "3045"  
CAMS-EQ: "342"  
NAM-INDIA-EQ: "357"  
TATAMOTORS-EQ: "3456"  
FINEORG-EQ: "3744"  
HATSUN-EQ: "3892"  
NAVA-EQ: "4014"  
DYNAMATECH-EQ: "4525"  
BANKINDIA-EQ: "4745"  
KIMS-EQ: "4847"  
NUVOCO-EQ: "5426"  
CHEMPLASTS-EQ: "5449"  
GRSE-EQ: "5475"  
PARAS-EQ: "5911"  
GOODLUCK-EQ: "6125"  
FCL-EQ: "6198"  
SONATSOFTW-EQ: "6596"  
SJS-EQ: "6643"  
EXIDEIND-EQ: "676"  
SAPPHIRE-EQ: "6718"  
CHOLAFIN-EQ: "685"  
RATEGAIN-EQ: "7177"  
JUNIPER-EQ: "22824"  
BHARTIHEXA-EQ: "23489"  
TBOTEK-EQ: "23740"  
PEL-EQ: "2412"  
SYMPHONY-EQ: "24190"  
ASHIANA-EQ: "24403"  
AVANTEL-EQ: "24610"  
APLLTD-EQ: "25328"  
NORTHARC-EQ: "25426"  
TRITURBINE-EQ: "25584"  
APLAPOLLO-EQ: "25780"  
MTARTECH-EQ: "2709"  
ANURAS-EQ: "2829"  
RELIANCE-EQ: "2885"  
ITCHOTELS-EQ: "29251"  
MCLOUD-EQ: "29482"  
GUJTHEM-EQ: "29731"  
JUSTDIAL-EQ: "29962"  
SHREECEM-EQ: "3103"  
BAJFINANCE-EQ: "317"  
TATASTEEL-EQ: "3499"  
CREDITACC-EQ: "4421"  
ENGINERSIN-EQ: "4907"  
ASAHIINDIA-EQ: "5378"  
ACUTAAS-EQ: "5578"  
CARBORUNIV-EQ: "595"  
AXISBANK-EQ: "5900"  
ABSLAMC-EQ: "6018"  
POLICYBZR-EQ: "6656"  
GREENLAM-EQ: "6848"  
HCLTECH-EQ: "7229"  
METROBRAND-EQ: "7242"  
CGPOWER-EQ: "760"  
KALPATARU-EQ: "758558"  
DABUR-EQ: "772"  
INOXWIND-EQ: "7852"  
ZYDUSLIFE-EQ: "7929"  
GATEWAY-EQ: "8510"  
USHAMART-EQ: "8840"  
EQUITASBNK-EQ: "913"  
ORCHPHARMA-EQ: "926"  
TIPSMUSIC-EQ: "9117"  
STLTECH-EQ: "9309"  
AETHER-EQ: "9810"  
GRWRHITECH-EQ: "7982"  
TVSMOTOR-EQ: "8479"  
MSUMI-EQ: "8596"  
VRLLOG-EQ: "8696"  
GAEL-EQ: "8828"  
EICHERMOT-EQ: "910"  
EIHOTEL-EQ: "919"  
IOB-EQ: "9348"  
ESABINDIA-EQ: "955"  
VENUSPIPES-EQ: "9592"

## config\markets\india\watchlists\master.yaml

stocks:  
- tradingsymbol: JISLJALEQS-EQ  
 exchange: NSE  
- tradingsymbol: MARKSANS-EQ  
 exchange: NSE  
- tradingsymbol: PNB-EQ  
 exchange: NSE  
- tradingsymbol: NUCLEUS-EQ  
 exchange: NSE  
- tradingsymbol: CANBK-EQ  
 exchange: NSE  
- tradingsymbol: GODREJIND-EQ  
 exchange: NSE  
- tradingsymbol: DIVISLAB-EQ  
 exchange: NSE  
- tradingsymbol: IDFCFIRSTB-EQ  
 exchange: NSE  
- tradingsymbol: APOLLO-EQ  
 exchange: NSE  
- tradingsymbol: POONAWALLA-EQ  
 exchange: NSE  
- tradingsymbol: PILANIINVS-EQ  
 exchange: NSE  
- tradingsymbol: EMIL-EQ  
 exchange: NSE  
- tradingsymbol: COFORGE-EQ  
 exchange: NSE  
- tradingsymbol: JBMA-EQ  
 exchange: NSE  
- tradingsymbol: JSWSTEEL-EQ  
 exchange: NSE  
- tradingsymbol: WELCORP-EQ  
 exchange: NSE  
- tradingsymbol: SSWL-EQ  
 exchange: NSE  
- tradingsymbol: JSWHL-EQ  
 exchange: NSE  
- tradingsymbol: YESBANK-EQ  
 exchange: NSE  
- tradingsymbol: ICIL-EQ  
 exchange: NSE  
- tradingsymbol: KAYNES-EQ  
 exchange: NSE  
- tradingsymbol: GSFC-EQ  
 exchange: NSE  
- tradingsymbol: CASTROLIND-EQ  
 exchange: NSE  
- tradingsymbol: TEAMLEASE-EQ  
 exchange: NSE  
- tradingsymbol: KRISHANA-EQ  
 exchange: NSE  
- tradingsymbol: 360ONE-EQ  
 exchange: NSE  
- tradingsymbol: PVRINOX-EQ  
 exchange: NSE  
- tradingsymbol: VOLTAMP-EQ  
 exchange: NSE  
- tradingsymbol: AARTIPHARM-EQ  
 exchange: NSE  
- tradingsymbol: BANCOINDIA-EQ  
 exchange: NSE  
- tradingsymbol: TIMKEN-EQ  
 exchange: NSE  
- tradingsymbol: HUBTOWN-EQ  
 exchange: NSE  
- tradingsymbol: HINDZINC-EQ  
 exchange: NSE  
- tradingsymbol: IDEA-EQ  
 exchange: NSE  
- tradingsymbol: PAGEIND-EQ  
 exchange: NSE  
- tradingsymbol: ICRA-EQ  
 exchange: NSE  
- tradingsymbol: KPRMILL-EQ  
 exchange: NSE  
- tradingsymbol: CIEINDIA-EQ  
 exchange: NSE  
- tradingsymbol: CENTUM-EQ  
 exchange: NSE  
- tradingsymbol: TARIL-EQ  
 exchange: NSE  
- tradingsymbol: MANKIND-EQ  
 exchange: NSE  
- tradingsymbol: TARC-EQ  
 exchange: NSE  
- tradingsymbol: INFY-EQ  
 exchange: NSE  
- tradingsymbol: BECTORFOOD-EQ  
 exchange: NSE  
- tradingsymbol: ITC-EQ  
 exchange: NSE  
- tradingsymbol: THYROCARE-EQ  
 exchange: NSE  
- tradingsymbol: CYIENTDLM-EQ  
 exchange: NSE  
- tradingsymbol: VINATIORGA-EQ  
 exchange: NSE  
- tradingsymbol: ADANIPOWER-EQ  
 exchange: NSE  
- tradingsymbol: NHPC-EQ  
 exchange: NSE  
- tradingsymbol: RTNPOWER-EQ  
 exchange: NSE  
- tradingsymbol: QUESS-EQ  
 exchange: NSE  
- tradingsymbol: YATHARTH-EQ  
 exchange: NSE  
- tradingsymbol: JSWENERGY-EQ  
 exchange: NSE  
- tradingsymbol: COHANCE-EQ  
 exchange: NSE  
- tradingsymbol: PERSISTENT-EQ  
 exchange: NSE  
- tradingsymbol: POWERINDIA-EQ  
 exchange: NSE  
- tradingsymbol: NUVAMA-EQ  
 exchange: NSE  
- tradingsymbol: JSWINFRA-EQ  
 exchange: NSE  
- tradingsymbol: KIRLOSIND-EQ  
 exchange: NSE  
- tradingsymbol: KIOCL-EQ  
 exchange: NSE  
- tradingsymbol: ROSSARI-EQ  
 exchange: NSE  
- tradingsymbol: DEEPAKNTR-EQ  
 exchange: NSE  
- tradingsymbol: IREDA-EQ  
 exchange: NSE  
- tradingsymbol: ASHAPURMIN-EQ  
 exchange: NSE  
- tradingsymbol: RAMCOCEM-EQ  
 exchange: NSE  
- tradingsymbol: MASTEK-EQ  
 exchange: NSE  
- tradingsymbol: BDL-EQ  
 exchange: NSE  
- tradingsymbol: DIXON-EQ  
 exchange: NSE  
- tradingsymbol: MAXHEALTH-EQ  
 exchange: NSE  
- tradingsymbol: MANORAMA-EQ  
 exchange: NSE  
- tradingsymbol: SYNGENE-EQ  
 exchange: NSE  
- tradingsymbol: FINPIPE-EQ  
 exchange: NSE  
- tradingsymbol: POWERMECH-EQ  
 exchange: NSE  
- tradingsymbol: UCOBANK-EQ  
 exchange: NSE  
- tradingsymbol: IGL-EQ  
 exchange: NSE  
- tradingsymbol: UPL-EQ  
 exchange: NSE  
- tradingsymbol: LUXIND-EQ  
 exchange: NSE  
- tradingsymbol: VAIBHAVGBL-EQ  
 exchange: NSE  
- tradingsymbol: GOKEX-EQ  
 exchange: NSE  
- tradingsymbol: AMBER-EQ  
 exchange: NSE  
- tradingsymbol: NH-EQ  
 exchange: NSE  
- tradingsymbol: HNDFDS-EQ  
 exchange: NSE  
- tradingsymbol: TRIVENI-EQ  
 exchange: NSE  
- tradingsymbol: KEC-EQ  
 exchange: NSE  
- tradingsymbol: GALLANTT-EQ  
 exchange: NSE  
- tradingsymbol: SDBL-EQ  
 exchange: NSE  
- tradingsymbol: RATNAMANI-EQ  
 exchange: NSE  
- tradingsymbol: NSLNISP-EQ  
 exchange: NSE  
- tradingsymbol: PGIL-EQ  
 exchange: NSE  
- tradingsymbol: SEQUENT-EQ  
 exchange: NSE  
- tradingsymbol: PFC-EQ  
 exchange: NSE  
- tradingsymbol: ASTRAL-EQ  
 exchange: NSE  
- tradingsymbol: CSBBANK-EQ  
 exchange: NSE  
- tradingsymbol: BAJAJELEC-EQ  
 exchange: NSE  
- tradingsymbol: AVALON-EQ  
 exchange: NSE  
- tradingsymbol: RELIGARE-EQ  
 exchange: NSE  
- tradingsymbol: ASTERDM-EQ  
 exchange: NSE  
- tradingsymbol: DHANI-EQ  
 exchange: NSE  
- tradingsymbol: TITAGARH-EQ  
 exchange: NSE  
- tradingsymbol: INGERRAND-EQ  
 exchange: NSE  
- tradingsymbol: PRINCEPIPE-EQ  
 exchange: NSE  
- tradingsymbol: IOC-EQ  
 exchange: NSE  
- tradingsymbol: CROMPTON-EQ  
 exchange: NSE  
- tradingsymbol: NESTLEIND-EQ  
 exchange: NSE  
- tradingsymbol: SBICARD-EQ  
 exchange: NSE  
- tradingsymbol: THANGAMAYL-EQ  
 exchange: NSE  
- tradingsymbol: JIOFIN-EQ  
 exchange: NSE  
- tradingsymbol: DIACABS-EQ  
 exchange: NSE  
- tradingsymbol: RRKABEL-EQ  
 exchange: NSE  
- tradingsymbol: PNBHOUSING-EQ  
 exchange: NSE  
- tradingsymbol: SFL-EQ  
 exchange: NSE  
- tradingsymbol: ARVIND-EQ  
 exchange: NSE  
- tradingsymbol: KSB-EQ  
 exchange: NSE  
- tradingsymbol: ASHOKA-EQ  
 exchange: NSE  
- tradingsymbol: CGCL-EQ  
 exchange: NSE  
- tradingsymbol: INDIASHLTR-EQ  
 exchange: NSE  
- tradingsymbol: ERIS-EQ  
 exchange: NSE  
- tradingsymbol: ACC-EQ  
 exchange: NSE  
- tradingsymbol: EPACK-EQ  
 exchange: NSE  
- tradingsymbol: BANDHANBNK-EQ  
 exchange: NSE  
- tradingsymbol: FACT-EQ  
 exchange: NSE  
- tradingsymbol: FEDERALBNK-EQ  
 exchange: NSE  
- tradingsymbol: UNIONBANK-EQ  
 exchange: NSE  
- tradingsymbol: SHK-EQ  
 exchange: NSE  
- tradingsymbol: JSL-EQ  
 exchange: NSE  
- tradingsymbol: BIOCON-EQ  
 exchange: NSE  
- tradingsymbol: APARINDS-EQ  
 exchange: NSE  
- tradingsymbol: WESTLIFE-EQ  
 exchange: NSE  
- tradingsymbol: LALPATHLAB-EQ  
 exchange: NSE  
- tradingsymbol: PATELENG-EQ  
 exchange: NSE  
- tradingsymbol: JPPOWER-EQ  
 exchange: NSE  
- tradingsymbol: 63MOONS-EQ  
 exchange: NSE  
- tradingsymbol: GRANULES-EQ  
 exchange: NSE  
- tradingsymbol: GENUSPOWER-EQ  
 exchange: NSE  
- tradingsymbol: GRASIM-EQ  
 exchange: NSE  
- tradingsymbol: M&MFIN-EQ  
 exchange: NSE  
- tradingsymbol: ALLCARGO-EQ  
 exchange: NSE  
- tradingsymbol: PFOCUS-EQ  
 exchange: NSE  
- tradingsymbol: EMAMILTD-EQ  
 exchange: NSE  
- tradingsymbol: GMRAIRPORT-EQ  
 exchange: NSE  
- tradingsymbol: TECHM-EQ  
 exchange: NSE  
- tradingsymbol: FIEMIND-EQ  
 exchange: NSE  
- tradingsymbol: GESHIP-EQ  
 exchange: NSE  
- tradingsymbol: HINDUNILVR-EQ  
 exchange: NSE  
- tradingsymbol: NFL-EQ  
 exchange: NSE  
- tradingsymbol: GODREJAGRO-EQ  
 exchange: NSE  
- tradingsymbol: SANOFI-EQ  
 exchange: NSE  
- tradingsymbol: PHOENIXLTD-EQ  
 exchange: NSE  
- tradingsymbol: AKZOINDIA-EQ  
 exchange: NSE  
- tradingsymbol: PURVA-EQ  
 exchange: NSE  
- tradingsymbol: KNRCON-EQ  
 exchange: NSE  
- tradingsymbol: VGUARD-EQ  
 exchange: NSE  
- tradingsymbol: NESCO-EQ  
 exchange: NSE  
- tradingsymbol: GILLETTE-EQ  
 exchange: NSE  
- tradingsymbol: SUNDROP-EQ  
 exchange: NSE  
- tradingsymbol: BAJAJ-AUTO-EQ  
 exchange: NSE  
- tradingsymbol: HGINFRA-EQ  
 exchange: NSE  
- tradingsymbol: MHRIL-EQ  
 exchange: NSE  
- tradingsymbol: AHLUCONT-EQ  
 exchange: NSE  
- tradingsymbol: BAYERCROP-EQ  
 exchange: NSE  
- tradingsymbol: SBFC-EQ  
 exchange: NSE  
- tradingsymbol: JUBLFOOD-EQ  
 exchange: NSE  
- tradingsymbol: KARURVYSYA-EQ  
 exchange: NSE  
- tradingsymbol: ZAGGLE-EQ  
 exchange: NSE  
- tradingsymbol: MARATHON-EQ  
 exchange: NSE  
- tradingsymbol: KSL-EQ  
 exchange: NSE  
- tradingsymbol: M&M-EQ  
 exchange: NSE  
- tradingsymbol: OBEROIRLTY-EQ  
 exchange: NSE  
- tradingsymbol: AZAD-EQ  
 exchange: NSE  
- tradingsymbol: INNOVACAP-EQ  
 exchange: NSE  
- tradingsymbol: STARCEMENT-EQ  
 exchange: NSE  
- tradingsymbol: JYOTICNC-EQ  
 exchange: NSE  
- tradingsymbol: ABCAPITAL-EQ  
 exchange: NSE  
- tradingsymbol: ICICIGI-EQ  
 exchange: NSE  
- tradingsymbol: TEXRAIL-EQ  
 exchange: NSE  
- tradingsymbol: HFCL-EQ  
 exchange: NSE  
- tradingsymbol: ENTERO-EQ  
 exchange: NSE  
- tradingsymbol: SHAREINDIA-EQ  
 exchange: NSE  
- tradingsymbol: RADICO-EQ  
 exchange: NSE  
- tradingsymbol: PPLPHARMA-EQ  
 exchange: NSE  
- tradingsymbol: GUFICBIO-EQ  
 exchange: NSE  
- tradingsymbol: ASTRAMICRO-EQ  
 exchange: NSE  
- tradingsymbol: IIFL-EQ  
 exchange: NSE  
- tradingsymbol: SHOPERSTOP-EQ  
 exchange: NSE  
- tradingsymbol: DCXINDIA-EQ  
 exchange: NSE  
- tradingsymbol: SWSOLAR-EQ  
 exchange: NSE  
- tradingsymbol: GUJALKALI-EQ  
 exchange: NSE  
- tradingsymbol: AMBUJACEM-EQ  
 exchange: NSE  
- tradingsymbol: ROUTE-EQ  
 exchange: NSE  
- tradingsymbol: GALAXYSURF-EQ  
 exchange: NSE  
- tradingsymbol: GSPL-EQ  
 exchange: NSE  
- tradingsymbol: SOLARINDS-EQ  
 exchange: NSE  
- tradingsymbol: ELECON-EQ  
 exchange: NSE  
- tradingsymbol: DCBBANK-EQ  
 exchange: NSE  
- tradingsymbol: FLUOROCHEM-EQ  
 exchange: NSE  
- tradingsymbol: NAUKRI-EQ  
 exchange: NSE  
- tradingsymbol: BBL-EQ  
 exchange: NSE  
- tradingsymbol: LTFOODS-EQ  
 exchange: NSE  
- tradingsymbol: SAGCEM-EQ  
 exchange: NSE  
- tradingsymbol: NETWORK18-EQ  
 exchange: NSE  
- tradingsymbol: HSCL-EQ  
 exchange: NSE  
- tradingsymbol: BALAMINES-EQ  
 exchange: NSE  
- tradingsymbol: DLF-EQ  
 exchange: NSE  
- tradingsymbol: IDBI-EQ  
 exchange: NSE  
- tradingsymbol: KDDL-EQ  
 exchange: NSE  
- tradingsymbol: EDELWEISS-EQ  
 exchange: NSE  
- tradingsymbol: COLPAL-EQ  
 exchange: NSE  
- tradingsymbol: INDIAGLYCO-EQ  
 exchange: NSE  
- tradingsymbol: ECLERX-EQ  
 exchange: NSE  
- tradingsymbol: TATAINVEST-EQ  
 exchange: NSE  
- tradingsymbol: APOLLOTYRE-EQ  
 exchange: NSE  
- tradingsymbol: JBCHEPHARM-EQ  
 exchange: NSE  
- tradingsymbol: BLS-EQ  
 exchange: NSE  
- tradingsymbol: LLOYDSENGG-EQ  
 exchange: NSE  
- tradingsymbol: GODREJPROP-EQ  
 exchange: NSE  
- tradingsymbol: ABBOTINDIA-EQ  
 exchange: NSE  
- tradingsymbol: KPIL-EQ  
 exchange: NSE  
- tradingsymbol: MANINFRA-EQ  
 exchange: NSE  
- tradingsymbol: LGBBROSLTD-EQ  
 exchange: NSE  
- tradingsymbol: JLHL-EQ  
 exchange: NSE  
- tradingsymbol: SAMHI-EQ  
 exchange: NSE  
- tradingsymbol: STYRENIX-EQ  
 exchange: NSE  
- tradingsymbol: MANAPPURAM-EQ  
 exchange: NSE  
- tradingsymbol: PRICOLLTD-EQ  
 exchange: NSE  
- tradingsymbol: LMW-EQ  
 exchange: NSE  
- tradingsymbol: JWL-EQ  
 exchange: NSE  
- tradingsymbol: RESPONIND-EQ  
 exchange: NSE  
- tradingsymbol: COALINDIA-EQ  
 exchange: NSE  
- tradingsymbol: PSPPROJECT-EQ  
 exchange: NSE  
- tradingsymbol: PSB-EQ  
 exchange: NSE  
- tradingsymbol: ASHOKLEY-EQ  
 exchange: NSE  
- tradingsymbol: SIS-EQ  
 exchange: NSE  
- tradingsymbol: IEX-EQ  
 exchange: NSE  
- tradingsymbol: MRPL-EQ  
 exchange: NSE  
- tradingsymbol: AIIL-EQ  
 exchange: NSE  
- tradingsymbol: AWFIS-EQ  
 exchange: NSE  
- tradingsymbol: CEIGALL-EQ  
 exchange: NSE  
- tradingsymbol: ONGC-EQ  
 exchange: NSE  
- tradingsymbol: PREMIERENE-EQ  
 exchange: NSE  
- tradingsymbol: TDPOWERSYS-EQ  
 exchange: NSE  
- tradingsymbol: WAAREEENER-EQ  
 exchange: NSE  
- tradingsymbol: PCBL-EQ  
 exchange: NSE  
- tradingsymbol: SWIGGY-EQ  
 exchange: NSE  
- tradingsymbol: BLACKBUCK-EQ  
 exchange: NSE  
- tradingsymbol: JUBLINGREA-EQ  
 exchange: NSE  
- tradingsymbol: EASEMYTRIP-EQ  
 exchange: NSE  
- tradingsymbol: VMM-EQ  
 exchange: NSE  
- tradingsymbol: KITEX-EQ  
 exchange: NSE  
- tradingsymbol: TVSHLTD-EQ  
 exchange: NSE  
- tradingsymbol: CARERATING-EQ  
 exchange: NSE  
- tradingsymbol: ONESOURCE-EQ  
 exchange: NSE  
- tradingsymbol: SAIL-EQ  
 exchange: NSE  
- tradingsymbol: HAL-EQ  
 exchange: NSE  
- tradingsymbol: HEIDELBERG-EQ  
 exchange: NSE  
- tradingsymbol: ASIANPAINT-EQ  
 exchange: NSE  
- tradingsymbol: RELAXO-EQ  
 exchange: NSE  
- tradingsymbol: EMCURE-EQ  
 exchange: NSE  
- tradingsymbol: NOCIL-EQ  
 exchange: NSE  
- tradingsymbol: LTF-EQ  
 exchange: NSE  
- tradingsymbol: EUREKAFORB-EQ  
 exchange: NSE  
- tradingsymbol: PIDILITIND-EQ  
 exchange: NSE  
- tradingsymbol: KPEL-EQ  
 exchange: NSE  
- tradingsymbol: EIEL-EQ  
 exchange: NSE  
- tradingsymbol: RTNINDIA-EQ  
 exchange: NSE  
- tradingsymbol: LXCHEM-EQ  
 exchange: NSE  
- tradingsymbol: RCF-EQ  
 exchange: NSE  
- tradingsymbol: QPOWER-EQ  
 exchange: NSE  
- tradingsymbol: ORIENTELEC-EQ  
 exchange: NSE  
- tradingsymbol: SUDARSCHEM-EQ  
 exchange: NSE  
- tradingsymbol: TATAELXSI-EQ  
 exchange: NSE  
- tradingsymbol: HONAUT-EQ  
 exchange: NSE  
- tradingsymbol: THERMAX-EQ  
 exchange: NSE  
- tradingsymbol: TITAN-EQ  
 exchange: NSE  
- tradingsymbol: MARICO-EQ  
 exchange: NSE  
- tradingsymbol: SHRIRAMFIN-EQ  
 exchange: NSE  
- tradingsymbol: SBCL-EQ  
 exchange: NSE  
- tradingsymbol: BLUEDART-EQ  
 exchange: NSE  
- tradingsymbol: SHARDACROP-EQ  
 exchange: NSE  
- tradingsymbol: GMDCLTD-EQ  
 exchange: NSE  
- tradingsymbol: ALIVUS-EQ  
 exchange: NSE  
- tradingsymbol: AAVAS-EQ  
 exchange: NSE  
- tradingsymbol: CARTRADE-EQ  
 exchange: NSE  
- tradingsymbol: SERVOTECH-EQ  
 exchange: NSE  
- tradingsymbol: CANFINHOME-EQ  
 exchange: NSE  
- tradingsymbol: SANSERA-EQ  
 exchange: NSE  
- tradingsymbol: GRAPHITE-EQ  
 exchange: NSE  
- tradingsymbol: NATIONALUM-EQ  
 exchange: NSE  
- tradingsymbol: CIPLA-EQ  
 exchange: NSE  
- tradingsymbol: HEMIPROP-EQ  
 exchange: NSE  
- tradingsymbol: GANECOS-EQ  
 exchange: NSE  
- tradingsymbol: JTLIND-EQ  
 exchange: NSE  
- tradingsymbol: AVANTIFEED-EQ  
 exchange: NSE  
- tradingsymbol: DCMSHRIRAM-EQ  
 exchange: NSE  
- tradingsymbol: CHALET-EQ  
 exchange: NSE  
- tradingsymbol: ANUP-EQ  
 exchange: NSE  
- tradingsymbol: METROPOLIS-EQ  
 exchange: NSE  
- tradingsymbol: JAYNECOIND-EQ  
 exchange: NSE  
- tradingsymbol: NEULANDLAB-EQ  
 exchange: NSE  
- tradingsymbol: RPEL-EQ  
 exchange: NSE  
- tradingsymbol: SHAKTIPUMP-EQ  
 exchange: NSE  
- tradingsymbol: JISLDVREQS-EQ  
 exchange: NSE  
- tradingsymbol: PFIZER-EQ  
 exchange: NSE  
- tradingsymbol: SWANENERGY-EQ  
 exchange: NSE  
- tradingsymbol: NIVABUPA-EQ  
 exchange: NSE  
- tradingsymbol: AUROPHARMA-EQ  
 exchange: NSE  
- tradingsymbol: GICRE-EQ  
 exchange: NSE  
- tradingsymbol: RALLIS-EQ  
 exchange: NSE  
- tradingsymbol: IKS-EQ  
 exchange: NSE  
- tradingsymbol: CRAFTSMAN-EQ  
 exchange: NSE  
- tradingsymbol: INDUSTOWER-EQ  
 exchange: NSE  
- tradingsymbol: MOSCHIP-EQ  
 exchange: NSE  
- tradingsymbol: KALYANKJIL-EQ  
 exchange: NSE  
- tradingsymbol: NAZARA-EQ  
 exchange: NSE  
- tradingsymbol: PRABHA-EQ  
 exchange: NSE  
- tradingsymbol: JINDALSAW-EQ  
 exchange: NSE  
- tradingsymbol: VEDL-EQ  
 exchange: NSE  
- tradingsymbol: TIINDIA-EQ  
 exchange: NSE  
- tradingsymbol: HONDAPOWER-EQ  
 exchange: NSE  
- tradingsymbol: SUBROS-EQ  
 exchange: NSE  
- tradingsymbol: BALMLAWRIE-EQ  
 exchange: NSE  
- tradingsymbol: SUNFLAG-EQ  
 exchange: NSE  
- tradingsymbol: SURYAROSNI-EQ  
 exchange: NSE  
- tradingsymbol: VIPIND-EQ  
 exchange: NSE  
- tradingsymbol: BATAINDIA-EQ  
 exchange: NSE  
- tradingsymbol: TATACOMM-EQ  
 exchange: NSE  
- tradingsymbol: ZEEL-EQ  
 exchange: NSE  
- tradingsymbol: BERGEPAINT-EQ  
 exchange: NSE  
- tradingsymbol: NATCOPHARM-EQ  
 exchange: NSE  
- tradingsymbol: GULFOILLUB-EQ  
 exchange: NSE  
- tradingsymbol: MPHASIS-EQ  
 exchange: NSE  
- tradingsymbol: EPIGRAL-EQ  
 exchange: NSE  
- tradingsymbol: CESC-EQ  
 exchange: NSE  
- tradingsymbol: LATENTVIEW-EQ  
 exchange: NSE  
- tradingsymbol: AARTIIND-EQ  
 exchange: NSE  
- tradingsymbol: MEDPLUS-EQ  
 exchange: NSE  
- tradingsymbol: NPST-EQ  
 exchange: NSE  
- tradingsymbol: OSWALPUMPS-EQ  
 exchange: NSE  
- tradingsymbol: NBCC-EQ  
 exchange: NSE  
- tradingsymbol: LODHA-EQ  
 exchange: NSE  
- tradingsymbol: ISGEC-EQ  
 exchange: NSE  
- tradingsymbol: TORNTPHARM-EQ  
 exchange: NSE  
- tradingsymbol: VOLTAS-EQ  
 exchange: NSE  
- tradingsymbol: WIPRO-EQ  
 exchange: NSE  
- tradingsymbol: BEL-EQ  
 exchange: NSE  
- tradingsymbol: CAPLIPOINT-EQ  
 exchange: NSE  
- tradingsymbol: BANKBARODA-EQ  
 exchange: NSE  
- tradingsymbol: 3MINDIA-EQ  
 exchange: NSE  
- tradingsymbol: SHYAMMETL-EQ  
 exchange: NSE  
- tradingsymbol: BIRLACORPN-EQ  
 exchange: NSE  
- tradingsymbol: BOMDYEING-EQ  
 exchange: NSE  
- tradingsymbol: APTUS-EQ  
 exchange: NSE  
- tradingsymbol: ASTRAZEN-EQ  
 exchange: NSE  
- tradingsymbol: INTELLECT-EQ  
 exchange: NSE  
- tradingsymbol: SOUTHBANK-EQ  
 exchange: NSE  
- tradingsymbol: STAR-EQ  
 exchange: NSE  
- tradingsymbol: RAJESHEXPO-EQ  
 exchange: NSE  
- tradingsymbol: WOCKPHARMA-EQ  
 exchange: NSE  
- tradingsymbol: WAAREERTL-EQ  
 exchange: NSE  
- tradingsymbol: CRISIL-EQ  
 exchange: NSE  
- tradingsymbol: AEGISVOPAK-EQ  
 exchange: NSE  
- tradingsymbol: DIGITIDE-EQ  
 exchange: NSE  
- tradingsymbol: KTKBANK-EQ  
 exchange: NSE  
- tradingsymbol: DALBHARAT-EQ  
 exchange: NSE  
- tradingsymbol: DEEPAKFERT-EQ  
 exchange: NSE  
- tradingsymbol: EIDPARRY-EQ  
 exchange: NSE  
- tradingsymbol: ARVINDFASN-EQ  
 exchange: NSE  
- tradingsymbol: ELECTCAST-EQ  
 exchange: NSE  
- tradingsymbol: SKIPPER-EQ  
 exchange: NSE  
- tradingsymbol: DELHIVERY-EQ  
 exchange: NSE  
- tradingsymbol: HAVELLS-EQ  
 exchange: NSE  
- tradingsymbol: NEOGEN-EQ  
 exchange: NSE  
- tradingsymbol: GOPAL-EQ  
 exchange: NSE  
- tradingsymbol: SANDHAR-EQ  
 exchange: NSE  
- tradingsymbol: PIIND-EQ  
 exchange: NSE  
- tradingsymbol: VADILALIND-EQ  
 exchange: NSE  
- tradingsymbol: VSTTILLERS-EQ  
 exchange: NSE  
- tradingsymbol: BANSALWIRE-EQ  
 exchange: NSE  
- tradingsymbol: OLAELEC-EQ  
 exchange: NSE  
- tradingsymbol: ADANIENT-EQ  
 exchange: NSE  
- tradingsymbol: WINDMACHIN-EQ  
 exchange: NSE  
- tradingsymbol: RAYMONDLSL-EQ  
 exchange: NSE  
- tradingsymbol: PGEL-EQ  
 exchange: NSE  
- tradingsymbol: POLYMED-EQ  
 exchange: NSE  
- tradingsymbol: LLOYDSENT-EQ  
 exchange: NSE  
- tradingsymbol: AVL-EQ  
 exchange: NSE  
- tradingsymbol: PRECWIRE-EQ  
 exchange: NSE  
- tradingsymbol: NTPCGREEN-EQ  
 exchange: NSE  
- tradingsymbol: SANATHAN-EQ  
 exchange: NSE  
- tradingsymbol: AGARWALEYE-EQ  
 exchange: NSE  
- tradingsymbol: E2E-EQ  
 exchange: NSE  
- tradingsymbol: DPABHUSHAN-EQ  
 exchange: NSE  
- tradingsymbol: RAINBOW-EQ  
 exchange: NSE  
- tradingsymbol: LICI-EQ  
 exchange: NSE  
- tradingsymbol: WONDERLA-EQ  
 exchange: NSE  
- tradingsymbol: RHIM-EQ  
 exchange: NSE  
- tradingsymbol: BALKRISIND-EQ  
 exchange: NSE  
- tradingsymbol: SUNDRMFAST-EQ  
 exchange: NSE  
- tradingsymbol: BALRAMCHIN-EQ  
 exchange: NSE  
- tradingsymbol: BASF-EQ  
 exchange: NSE  
- tradingsymbol: RITES-EQ  
 exchange: NSE  
- tradingsymbol: AEGISLOG-EQ  
 exchange: NSE  
- tradingsymbol: VARROC-EQ  
 exchange: NSE  
- tradingsymbol: HERITGFOOD-EQ  
 exchange: NSE  
- tradingsymbol: INDRAMEDCO-EQ  
 exchange: NSE  
- tradingsymbol: ICICIBANK-EQ  
 exchange: NSE  
- tradingsymbol: CLEAN-EQ  
 exchange: NSE  
- tradingsymbol: GRINFRA-EQ  
 exchange: NSE  
- tradingsymbol: KPIGREEN-EQ  
 exchange: NSE  
- tradingsymbol: UTIAMC-EQ  
 exchange: NSE  
- tradingsymbol: ITDCEM-EQ  
 exchange: NSE  
- tradingsymbol: CUB-EQ  
 exchange: NSE  
- tradingsymbol: BHAGCHEM-EQ  
 exchange: NSE  
- tradingsymbol: CHAMBLFERT-EQ  
 exchange: NSE  
- tradingsymbol: NYKAA-EQ  
 exchange: NSE  
- tradingsymbol: BSOFT-EQ  
 exchange: NSE  
- tradingsymbol: ANANDRATHI-EQ  
 exchange: NSE  
- tradingsymbol: DATAPATTNS-EQ  
 exchange: NSE  
- tradingsymbol: THELEELA-EQ  
 exchange: NSE  
- tradingsymbol: DALMIASUG-EQ  
 exchange: NSE  
- tradingsymbol: ALEMBICLTD-EQ  
 exchange: NSE  
- tradingsymbol: AWL-EQ  
 exchange: NSE  
- tradingsymbol: GMRP&UI-EQ  
 exchange: NSE  
- tradingsymbol: NLCINDIA-EQ  
 exchange: NSE  
- tradingsymbol: PRIVISCL-EQ  
 exchange: NSE  
- tradingsymbol: HIKAL-EQ  
 exchange: NSE  
- tradingsymbol: EMUDHRA-EQ  
 exchange: NSE  
- tradingsymbol: ARE&M-EQ  
 exchange: NSE  
- tradingsymbol: SCHAEFFLER-EQ  
 exchange: NSE  
- tradingsymbol: ZENSARTECH-EQ  
 exchange: NSE  
- tradingsymbol: SYRMA-EQ  
 exchange: NSE  
- tradingsymbol: GABRIEL-EQ  
 exchange: NSE  
- tradingsymbol: WELSPUNLIV-EQ  
 exchange: NSE  
- tradingsymbol: JAIBALAJI-EQ  
 exchange: NSE  
- tradingsymbol: LT-EQ  
 exchange: NSE  
- tradingsymbol: NEWGEN-EQ  
 exchange: NSE  
- tradingsymbol: GLAND-EQ  
 exchange: NSE  
- tradingsymbol: JKPAPER-EQ  
 exchange: NSE  
- tradingsymbol: AURIONPRO-EQ  
 exchange: NSE  
- tradingsymbol: ACI-EQ  
 exchange: NSE  
- tradingsymbol: HEROMOTOCO-EQ  
 exchange: NSE  
- tradingsymbol: JMFINANCIL-EQ  
 exchange: NSE  
- tradingsymbol: AGI-EQ  
 exchange: NSE  
- tradingsymbol: TIIL-EQ  
 exchange: NSE  
- tradingsymbol: V2RETAIL-EQ  
 exchange: NSE  
- tradingsymbol: CENTRALBK-EQ  
 exchange: NSE  
- tradingsymbol: RBA-EQ  
 exchange: NSE  
- tradingsymbol: MOTILALOFS-EQ  
 exchange: NSE  
- tradingsymbol: POWERGRID-EQ  
 exchange: NSE  
- tradingsymbol: ADANIPORTS-EQ  
 exchange: NSE  
- tradingsymbol: KOLTEPATIL-EQ  
 exchange: NSE  
- tradingsymbol: HCG-EQ  
 exchange: NSE  
- tradingsymbol: UBL-EQ  
 exchange: NSE  
- tradingsymbol: PATANJALI-EQ  
 exchange: NSE  
- tradingsymbol: PREMEXPLN-EQ  
 exchange: NSE  
- tradingsymbol: POKARNA-EQ  
 exchange: NSE  
- tradingsymbol: ALOKINDS-EQ  
 exchange: NSE  
- tradingsymbol: SARDAEN-EQ  
 exchange: NSE  
- tradingsymbol: DBREALTY-EQ  
 exchange: NSE  
- tradingsymbol: RBLBANK-EQ  
 exchange: NSE  
- tradingsymbol: ENDURANCE-EQ  
 exchange: NSE  
- tradingsymbol: VBL-EQ  
 exchange: NSE  
- tradingsymbol: ITDC-EQ  
 exchange: NSE  
- tradingsymbol: BSE-EQ  
 exchange: NSE  
- tradingsymbol: TRENT-EQ  
 exchange: NSE  
- tradingsymbol: HONASA-EQ  
 exchange: NSE  
- tradingsymbol: ASKAUTOLTD-EQ  
 exchange: NSE  
- tradingsymbol: WABAG-EQ  
 exchange: NSE  
- tradingsymbol: IRFC-EQ  
 exchange: NSE  
- tradingsymbol: TATATECH-EQ  
 exchange: NSE  
- tradingsymbol: INDIGOPNTS-EQ  
 exchange: NSE  
- tradingsymbol: HOMEFIRST-EQ  
 exchange: NSE  
- tradingsymbol: DOMS-EQ  
 exchange: NSE  
- tradingsymbol: JINDWORLD-EQ  
 exchange: NSE  
- tradingsymbol: VTL-EQ  
 exchange: NSE  
- tradingsymbol: HUDCO-EQ  
 exchange: NSE  
- tradingsymbol: MAHSCOOTER-EQ  
 exchange: NSE  
- tradingsymbol: CDSL-EQ  
 exchange: NSE  
- tradingsymbol: AUBANK-EQ  
 exchange: NSE  
- tradingsymbol: MEDIASSIST-EQ  
 exchange: NSE  
- tradingsymbol: SUNDARMHLD-EQ  
 exchange: NSE  
- tradingsymbol: SBILIFE-EQ  
 exchange: NSE  
- tradingsymbol: UNITDSPR-EQ  
 exchange: NSE  
- tradingsymbol: SHARDAMOTR-EQ  
 exchange: NSE  
- tradingsymbol: KRBL-EQ  
 exchange: NSE  
- tradingsymbol: FOSECOIND-EQ  
 exchange: NSE  
- tradingsymbol: NSIL-EQ  
 exchange: NSE  
- tradingsymbol: GHCL-EQ  
 exchange: NSE  
- tradingsymbol: PETRONET-EQ  
 exchange: NSE  
- tradingsymbol: PTC-EQ  
 exchange: NSE  
- tradingsymbol: WELENT-EQ  
 exchange: NSE  
- tradingsymbol: GODFRYPHLP-EQ  
 exchange: NSE  
- tradingsymbol: GOLDIAM-EQ  
 exchange: NSE  
- tradingsymbol: RENUKA-EQ  
 exchange: NSE  
- tradingsymbol: JKCEMENT-EQ  
 exchange: NSE  
- tradingsymbol: HDFCBANK-EQ  
 exchange: NSE  
- tradingsymbol: CENTURYPLY-EQ  
 exchange: NSE  
- tradingsymbol: KKCL-EQ  
 exchange: NSE  
- tradingsymbol: GPIL-EQ  
 exchange: NSE  
- tradingsymbol: RSYSTEMS-EQ  
 exchange: NSE  
- tradingsymbol: JKLAKSHMI-EQ  
 exchange: NSE  
- tradingsymbol: HINDALCO-EQ  
 exchange: NSE  
- tradingsymbol: IRCTC-EQ  
 exchange: NSE  
- tradingsymbol: HBLENGINE-EQ  
 exchange: NSE  
- tradingsymbol: JASH-EQ  
 exchange: NSE  
- tradingsymbol: LUMAXTECH-EQ  
 exchange: NSE  
- tradingsymbol: UNOMINDA-EQ  
 exchange: NSE  
- tradingsymbol: FINCABLES-EQ  
 exchange: NSE  
- tradingsymbol: ARVSMART-EQ  
 exchange: NSE  
- tradingsymbol: UFLEX-EQ  
 exchange: NSE  
- tradingsymbol: MPSLTD-EQ  
 exchange: NSE  
- tradingsymbol: BHARTIARTL-EQ  
 exchange: NSE  
- tradingsymbol: MARUTI-EQ  
 exchange: NSE  
- tradingsymbol: INDIGO-EQ  
 exchange: NSE  
- tradingsymbol: AFFLE-EQ  
 exchange: NSE  
- tradingsymbol: ULTRACEMCO-EQ  
 exchange: NSE  
- tradingsymbol: FORCEMOT-EQ  
 exchange: NSE  
- tradingsymbol: NTPC-EQ  
 exchange: NSE  
- tradingsymbol: ALKEM-EQ  
 exchange: NSE  
- tradingsymbol: BIKAJI-EQ  
 exchange: NSE  
- tradingsymbol: RUSTOMJEE-EQ  
 exchange: NSE  
- tradingsymbol: GREAVESCOT-EQ  
 exchange: NSE  
- tradingsymbol: ABB-EQ  
 exchange: NSE  
- tradingsymbol: IIFLCAPS-EQ  
 exchange: NSE  
- tradingsymbol: AIAENG-EQ  
 exchange: NSE  
- tradingsymbol: GRINDWELL-EQ  
 exchange: NSE  
- tradingsymbol: ACE-EQ  
 exchange: NSE  
- tradingsymbol: ANANTRAJ-EQ  
 exchange: NSE  
- tradingsymbol: SOBHA-EQ  
 exchange: NSE  
- tradingsymbol: HINDPETRO-EQ  
 exchange: NSE  
- tradingsymbol: SIYSIL-EQ  
 exchange: NSE  
- tradingsymbol: FSL-EQ  
 exchange: NSE  
- tradingsymbol: INDIANB-EQ  
 exchange: NSE  
- tradingsymbol: FORTIS-EQ  
 exchange: NSE  
- tradingsymbol: INDIACEM-EQ  
 exchange: NSE  
- tradingsymbol: KIRLPNU-EQ  
 exchange: NSE  
- tradingsymbol: RAIN-EQ  
 exchange: NSE  
- tradingsymbol: LINDEINDIA-EQ  
 exchange: NSE  
- tradingsymbol: IPCALAB-EQ  
 exchange: NSE  
- tradingsymbol: RPGLIFE-EQ  
 exchange: NSE  
- tradingsymbol: ZFCVINDIA-EQ  
 exchange: NSE  
- tradingsymbol: AJMERA-EQ  
 exchange: NSE  
- tradingsymbol: NETWEB-EQ  
 exchange: NSE  
- tradingsymbol: ZYDUSWELL-EQ  
 exchange: NSE  
- tradingsymbol: SUNTECK-EQ  
 exchange: NSE  
- tradingsymbol: KAJARIACER-EQ  
 exchange: NSE  
- tradingsymbol: TVSSCS-EQ  
 exchange: NSE  
- tradingsymbol: TCPLPACK-EQ  
 exchange: NSE  
- tradingsymbol: HPL-EQ  
 exchange: NSE  
- tradingsymbol: SIGNATURE-EQ  
 exchange: NSE  
- tradingsymbol: CUMMINSIND-EQ  
 exchange: NSE  
- tradingsymbol: TI-EQ  
 exchange: NSE  
- tradingsymbol: MAXESTATES-EQ  
 exchange: NSE  
- tradingsymbol: GANESHHOUC-EQ  
 exchange: NSE  
- tradingsymbol: SPARC-EQ  
 exchange: NSE  
- tradingsymbol: IFBIND-EQ  
 exchange: NSE  
- tradingsymbol: CERA-EQ  
 exchange: NSE  
- tradingsymbol: JYOTHYLAB-EQ  
 exchange: NSE  
- tradingsymbol: CEATLTD-EQ  
 exchange: NSE  
- tradingsymbol: JKIL-EQ  
 exchange: NSE  
- tradingsymbol: IRB-EQ  
 exchange: NSE  
- tradingsymbol: NMDC-EQ  
 exchange: NSE  
- tradingsymbol: RECLTD-EQ  
 exchange: NSE  
- tradingsymbol: INFIBEAM-EQ  
 exchange: NSE  
- tradingsymbol: PTCIL-EQ  
 exchange: NSE  
- tradingsymbol: SUMICHEM-EQ  
 exchange: NSE  
- tradingsymbol: SHRIPISTON-EQ  
 exchange: NSE  
- tradingsymbol: LLOYDSME-EQ  
 exchange: NSE  
- tradingsymbol: LTIM-EQ  
 exchange: NSE  
- tradingsymbol: DBCORP-EQ  
 exchange: NSE  
- tradingsymbol: ADVENZYMES-EQ  
 exchange: NSE  
- tradingsymbol: CONCORDBIO-EQ  
 exchange: NSE  
- tradingsymbol: DBL-EQ  
 exchange: NSE  
- tradingsymbol: LTTS-EQ  
 exchange: NSE  
- tradingsymbol: BLUEJET-EQ  
 exchange: NSE  
- tradingsymbol: CELLO-EQ  
 exchange: NSE  
- tradingsymbol: MASFIN-EQ  
 exchange: NSE  
- tradingsymbol: DMART-EQ  
 exchange: NSE  
- tradingsymbol: RAMKY-EQ  
 exchange: NSE  
- tradingsymbol: PRESTIGE-EQ  
 exchange: NSE  
- tradingsymbol: FEDFINA-EQ  
 exchange: NSE  
- tradingsymbol: MAHSEAMLES-EQ  
 exchange: NSE  
- tradingsymbol: HAPPYFORGE-EQ  
 exchange: NSE  
- tradingsymbol: SUNCLAY-EQ  
 exchange: NSE  
- tradingsymbol: GODREJCP-EQ  
 exchange: NSE  
- tradingsymbol: ADANIENSOL-EQ  
 exchange: NSE  
- tradingsymbol: LUPIN-EQ  
 exchange: NSE  
- tradingsymbol: GUJGASLTD-EQ  
 exchange: NSE  
- tradingsymbol: INDIAMART-EQ  
 exchange: NSE  
- tradingsymbol: MAHABANK-EQ  
 exchange: NSE  
- tradingsymbol: GIPCL-EQ  
 exchange: NSE  
- tradingsymbol: GNFC-EQ  
 exchange: NSE  
- tradingsymbol: FMGOETZE-EQ  
 exchange: NSE  
- tradingsymbol: MEDANTA-EQ  
 exchange: NSE  
- tradingsymbol: NITCO-EQ  
 exchange: NSE  
- tradingsymbol: KEI-EQ  
 exchange: NSE  
- tradingsymbol: HEG-EQ  
 exchange: NSE  
- tradingsymbol: KFINTECH-EQ  
 exchange: NSE  
- tradingsymbol: GREENPANEL-EQ  
 exchange: NSE  
- tradingsymbol: TANLA-EQ  
 exchange: NSE  
- tradingsymbol: PITTIENG-EQ  
 exchange: NSE  
- tradingsymbol: JKTYRE-EQ  
 exchange: NSE  
- tradingsymbol: EMBDL-EQ  
 exchange: NSE  
- tradingsymbol: GOKULAGRO-EQ  
 exchange: NSE  
- tradingsymbol: WEBELSOLAR-EQ  
 exchange: NSE  
- tradingsymbol: NAVINFLUOR-EQ  
 exchange: NSE  
- tradingsymbol: TIMETECHNO-EQ  
 exchange: NSE  
- tradingsymbol: INDHOTEL-EQ  
 exchange: NSE  
- tradingsymbol: UJJIVANSFB-EQ  
 exchange: NSE  
- tradingsymbol: GMMPFAUDLR-EQ  
 exchange: NSE  
- tradingsymbol: IONEXCHANG-EQ  
 exchange: NSE  
- tradingsymbol: KIRIINDUS-EQ  
 exchange: NSE  
- tradingsymbol: BAJAJFINSV-EQ  
 exchange: NSE  
- tradingsymbol: ITI-EQ  
 exchange: NSE  
- tradingsymbol: REFEX-EQ  
 exchange: NSE  
- tradingsymbol: NIITMTS-EQ  
 exchange: NSE  
- tradingsymbol: WHIRLPOOL-EQ  
 exchange: NSE  
- tradingsymbol: SANDUMA-EQ  
 exchange: NSE  
- tradingsymbol: ICICIPRULI-EQ  
 exchange: NSE  
- tradingsymbol: SJVN-EQ  
 exchange: NSE  
- tradingsymbol: KINGFA-EQ  
 exchange: NSE  
- tradingsymbol: KOTAKBANK-EQ  
 exchange: NSE  
- tradingsymbol: BAJAJCON-EQ  
 exchange: NSE  
- tradingsymbol: GRAVITA-EQ  
 exchange: NSE  
- tradingsymbol: INOXINDIA-EQ  
 exchange: NSE  
- tradingsymbol: PENIND-EQ  
 exchange: NSE  
- tradingsymbol: TCI-EQ  
 exchange: NSE  
- tradingsymbol: OLECTRA-EQ  
 exchange: NSE  
- tradingsymbol: OFSS-EQ  
 exchange: NSE  
- tradingsymbol: TMB-EQ  
 exchange: NSE  
- tradingsymbol: GARFIBRES-EQ  
 exchange: NSE  
- tradingsymbol: HARSHA-EQ  
 exchange: NSE  
- tradingsymbol: RKFORGE-EQ  
 exchange: NSE  
- tradingsymbol: DATAMATICS-EQ  
 exchange: NSE  
- tradingsymbol: CCL-EQ  
 exchange: NSE  
- tradingsymbol: GLAXO-EQ  
 exchange: NSE  
- tradingsymbol: TCS-EQ  
 exchange: NSE  
- tradingsymbol: SUPRAJIT-EQ  
 exchange: NSE  
- tradingsymbol: KANSAINER-EQ  
 exchange: NSE  
- tradingsymbol: SUZLON-EQ  
 exchange: NSE  
- tradingsymbol: FIVESTAR-EQ  
 exchange: NSE  
- tradingsymbol: MBAPL-EQ  
 exchange: NSE  
- tradingsymbol: SAFARI-EQ  
 exchange: NSE  
- tradingsymbol: SUNTV-EQ  
 exchange: NSE  
- tradingsymbol: SKYGOLD-EQ  
 exchange: NSE  
- tradingsymbol: TORNTPOWER-EQ  
 exchange: NSE  
- tradingsymbol: REDINGTON-EQ  
 exchange: NSE  
- tradingsymbol: IFCI-EQ  
 exchange: NSE  
- tradingsymbol: KSCL-EQ  
 exchange: NSE  
- tradingsymbol: MANGCHEFER-EQ  
 exchange: NSE  
- tradingsymbol: BRIGADE-EQ  
 exchange: NSE  
- tradingsymbol: APOLLOHOSP-EQ  
 exchange: NSE  
- tradingsymbol: GVT&D-EQ  
 exchange: NSE  
- tradingsymbol: SINDHUTRAD-EQ  
 exchange: NSE  
- tradingsymbol: SENCO-EQ  
 exchange: NSE  
- tradingsymbol: OIL-EQ  
 exchange: NSE  
- tradingsymbol: MGL-EQ  
 exchange: NSE  
- tradingsymbol: REDTAPE-EQ  
 exchange: NSE  
- tradingsymbol: HINDCOPPER-EQ  
 exchange: NSE  
- tradingsymbol: MMTC-EQ  
 exchange: NSE  
- tradingsymbol: KIRLOSBROS-EQ  
 exchange: NSE  
- tradingsymbol: EMSLIMITED-EQ  
 exchange: NSE  
- tradingsymbol: LAURUSLABS-EQ  
 exchange: NSE  
- tradingsymbol: IMFA-EQ  
 exchange: NSE  
- tradingsymbol: BOROLTD-EQ  
 exchange: NSE  
- tradingsymbol: GPPL-EQ  
 exchange: NSE  
- tradingsymbol: LICHSGFIN-EQ  
 exchange: NSE  
- tradingsymbol: LUMAXIND-EQ  
 exchange: NSE  
- tradingsymbol: CHENNPETRO-EQ  
 exchange: NSE  
- tradingsymbol: JAMNAAUTO-EQ  
 exchange: NSE  
- tradingsymbol: MOIL-EQ  
 exchange: NSE  
- tradingsymbol: KIRLOSENG-EQ  
 exchange: NSE  
- tradingsymbol: TEJASNET-EQ  
 exchange: NSE  
- tradingsymbol: OPTIEMUS-EQ  
 exchange: NSE  
- tradingsymbol: PARKHOTELS-EQ  
 exchange: NSE  
- tradingsymbol: MFSL-EQ  
 exchange: NSE  
- tradingsymbol: COCHINSHIP-EQ  
 exchange: NSE  
- tradingsymbol: CHOLAHLDNG-EQ  
 exchange: NSE  
- tradingsymbol: BOSCHLTD-EQ  
 exchange: NSE  
- tradingsymbol: MOREPENLAB-EQ  
 exchange: NSE  
- tradingsymbol: JSFB-EQ  
 exchange: NSE  
- tradingsymbol: MRF-EQ  
 exchange: NSE  
- tradingsymbol: MUTHOOTFIN-EQ  
 exchange: NSE  
- tradingsymbol: GODIGIT-EQ  
 exchange: NSE  
- tradingsymbol: ARKADE-EQ  
 exchange: NSE  
- tradingsymbol: KRN-EQ  
 exchange: NSE  
- tradingsymbol: HYUNDAI-EQ  
 exchange: NSE  
- tradingsymbol: LEMONTREE-EQ  
 exchange: NSE  
- tradingsymbol: ACMESOLAR-EQ  
 exchange: NSE  
- tradingsymbol: RAYMOND-EQ  
 exchange: NSE  
- tradingsymbol: PCJEWELLER-EQ  
 exchange: NSE  
- tradingsymbol: AJAXENGG-EQ  
 exchange: NSE  
- tradingsymbol: HEXT-EQ  
 exchange: NSE  
- tradingsymbol: ORIENTCEM-EQ  
 exchange: NSE  
- tradingsymbol: SHANTIGEAR-EQ  
 exchange: NSE  
- tradingsymbol: INDOSTAR-EQ  
 exchange: NSE  
- tradingsymbol: BORORENEW-EQ  
 exchange: NSE  
- tradingsymbol: JTEKTINDIA-EQ  
 exchange: NSE  
- tradingsymbol: SRF-EQ  
 exchange: NSE  
- tradingsymbol: SUNDARMFIN-EQ  
 exchange: NSE  
- tradingsymbol: SUNPHARMA-EQ  
 exchange: NSE  
- tradingsymbol: SMLISUZU-EQ  
 exchange: NSE  
- tradingsymbol: TATAPOWER-EQ  
 exchange: NSE  
- tradingsymbol: BANARISUG-EQ  
 exchange: NSE  
- tradingsymbol: ADANIGREEN-EQ  
 exchange: NSE  
- tradingsymbol: UNICHEMLAB-EQ  
 exchange: NSE  
- tradingsymbol: JUBLPHARMA-EQ  
 exchange: NSE  
- tradingsymbol: VESUVIUS-EQ  
 exchange: NSE  
- tradingsymbol: VSTIND-EQ  
 exchange: NSE  
- tradingsymbol: WSTCSTPAPR-EQ  
 exchange: NSE  
- tradingsymbol: BHARATRAS-EQ  
 exchange: NSE  
- tradingsymbol: MOTHERSON-EQ  
 exchange: NSE  
- tradingsymbol: HDFCAMC-EQ  
 exchange: NSE  
- tradingsymbol: AARTIDRUGS-EQ  
 exchange: NSE  
- tradingsymbol: ALKYLAMINE-EQ  
 exchange: NSE  
- tradingsymbol: SONACOMS-EQ  
 exchange: NSE  
- tradingsymbol: FDC-EQ  
 exchange: NSE  
- tradingsymbol: IRCON-EQ  
 exchange: NSE  
- tradingsymbol: ETERNAL-EQ  
 exchange: NSE  
- tradingsymbol: CIGNITITEC-EQ  
 exchange: NSE  
- tradingsymbol: INDUSINDBK-EQ  
 exchange: NSE  
- tradingsymbol: PDSL-EQ  
 exchange: NSE  
- tradingsymbol: BRITANNIA-EQ  
 exchange: NSE  
- tradingsymbol: J&KBANK-EQ  
 exchange: NSE  
- tradingsymbol: TECHNOE-EQ  
 exchange: NSE  
- tradingsymbol: GOCOLORS-EQ  
 exchange: NSE  
- tradingsymbol: COROMANDEL-EQ  
 exchange: NSE  
- tradingsymbol: SUPRIYA-EQ  
 exchange: NSE  
- tradingsymbol: ZENTEC-EQ  
 exchange: NSE  
- tradingsymbol: ELLEN-EQ  
 exchange: NSE  
- tradingsymbol: IMAGICAA-EQ  
 exchange: NSE  
- tradingsymbol: SHAILY-EQ  
 exchange: NSE  
- tradingsymbol: ELGIEQUIP-EQ  
 exchange: NSE  
- tradingsymbol: CAMPUS-EQ  
 exchange: NSE  
- tradingsymbol: PGHL-EQ  
 exchange: NSE  
- tradingsymbol: ESCORTS-EQ  
 exchange: NSE  
- tradingsymbol: RVNL-EQ  
 exchange: NSE  
- tradingsymbol: INDGN-EQ  
 exchange: NSE  
- tradingsymbol: AADHARHFC-EQ  
 exchange: NSE  
- tradingsymbol: IXIGO-EQ  
 exchange: NSE  
- tradingsymbol: DHANUKA-EQ  
 exchange: NSE  
- tradingsymbol: FIRSTCRY-EQ  
 exchange: NSE  
- tradingsymbol: BAJAJHFL-EQ  
 exchange: NSE  
- tradingsymbol: MINDACORP-EQ  
 exchange: NSE  
- tradingsymbol: AFCONS-EQ  
 exchange: NSE  
- tradingsymbol: TRANSRAILL-EQ  
 exchange: NSE  
- tradingsymbol: MTNL-EQ  
 exchange: NSE  
- tradingsymbol: NCC-EQ  
 exchange: NSE  
- tradingsymbol: AKUMS-EQ  
 exchange: NSE  
- tradingsymbol: IGIL-EQ  
 exchange: NSE  
- tradingsymbol: VMART-EQ  
 exchange: NSE  
- tradingsymbol: PROTEAN-EQ  
 exchange: NSE  
- tradingsymbol: SAMMAANCAP-EQ  
 exchange: NSE  
- tradingsymbol: BAJAJHLDNG-EQ  
 exchange: NSE  
- tradingsymbol: SCI-EQ  
 exchange: NSE  
- tradingsymbol: SIEMENS-EQ  
 exchange: NSE  
- tradingsymbol: TATACHEM-EQ  
 exchange: NSE  
- tradingsymbol: BBTC-EQ  
 exchange: NSE  
- tradingsymbol: BHARATFORG-EQ  
 exchange: NSE  
- tradingsymbol: BHEL-EQ  
 exchange: NSE  
- tradingsymbol: HDFCLIFE-EQ  
 exchange: NSE  
- tradingsymbol: GAIL-EQ  
 exchange: NSE  
- tradingsymbol: DODLA-EQ  
 exchange: NSE  
- tradingsymbol: MAZDOCK-EQ  
 exchange: NSE  
- tradingsymbol: BPCL-EQ  
 exchange: NSE  
- tradingsymbol: ROLEXRINGS-EQ  
 exchange: NSE  
- tradingsymbol: DEVYANI-EQ  
 exchange: NSE  
- tradingsymbol: VIJAYA-EQ  
 exchange: NSE  
- tradingsymbol: ATGL-EQ  
 exchange: NSE  
- tradingsymbol: ABREL-EQ  
 exchange: NSE  
- tradingsymbol: PAYTM-EQ  
 exchange: NSE  
- tradingsymbol: JINDALSTEL-EQ  
 exchange: NSE  
- tradingsymbol: STARHEALTH-EQ  
 exchange: NSE  
- tradingsymbol: MAPMYINDIA-EQ  
 exchange: NSE  
- tradingsymbol: ATHERENERG-EQ  
 exchange: NSE  
- tradingsymbol: CMSINFO-EQ  
 exchange: NSE  
- tradingsymbol: MAHLIFE-EQ  
 exchange: NSE  
- tradingsymbol: RPSGVENT-EQ  
 exchange: NSE  
- tradingsymbol: CHOICEIN-EQ  
 exchange: NSE  
- tradingsymbol: POLYCAB-EQ  
 exchange: NSE  
- tradingsymbol: SPLPETRO-EQ  
 exchange: NSE  
- tradingsymbol: KPITTECH-EQ  
 exchange: NSE  
- tradingsymbol: BAJAJHIND-EQ  
 exchange: NSE  
- tradingsymbol: MCX-EQ  
 exchange: NSE  
- tradingsymbol: ANGELONE-EQ  
 exchange: NSE  
- tradingsymbol: SKFINDIA-EQ  
 exchange: NSE  
- tradingsymbol: SUPREMEIND-EQ  
 exchange: NSE  
- tradingsymbol: SWARAJENG-EQ  
 exchange: NSE  
- tradingsymbol: TATACONSUM-EQ  
 exchange: NSE  
- tradingsymbol: THOMASCOOK-EQ  
 exchange: NSE  
- tradingsymbol: TTKPRESTIG-EQ  
 exchange: NSE  
- tradingsymbol: BEML-EQ  
 exchange: NSE  
- tradingsymbol: NIACL-EQ  
 exchange: NSE  
- tradingsymbol: GREENPLY-EQ  
 exchange: NSE  
- tradingsymbol: JCHAC-EQ  
 exchange: NSE  
- tradingsymbol: CONCOR-EQ  
 exchange: NSE  
- tradingsymbol: HAPPSTMNDS-EQ  
 exchange: NSE  
- tradingsymbol: SAREGAMA-EQ  
 exchange: NSE  
- tradingsymbol: CYIENT-EQ  
 exchange: NSE  
- tradingsymbol: VISHNU-EQ  
 exchange: NSE  
- tradingsymbol: TEGA-EQ  
 exchange: NSE  
- tradingsymbol: COSMOFIRST-EQ  
 exchange: NSE  
- tradingsymbol: GLENMARK-EQ  
 exchange: NSE  
- tradingsymbol: BELRISE-EQ  
 exchange: NSE  
- tradingsymbol: AJANTPHARM-EQ  
 exchange: NSE  
- tradingsymbol: MANYAVAR-EQ  
 exchange: NSE  
- tradingsymbol: BLUESTARCO-EQ  
 exchange: NSE  
- tradingsymbol: DRREDDY-EQ  
 exchange: NSE  
- tradingsymbol: MSTCLTD-EQ  
 exchange: NSE  
- tradingsymbol: PNCINFRA-EQ  
 exchange: NSE  
- tradingsymbol: PRUDENT-EQ  
 exchange: NSE  
- tradingsymbol: TRIDENT-EQ  
 exchange: NSE  
- tradingsymbol: PARADEEP-EQ  
 exchange: NSE  
- tradingsymbol: EPL-EQ  
 exchange: NSE  
- tradingsymbol: ETHOSLTD-EQ  
 exchange: NSE  
- tradingsymbol: HLEGLAS-EQ  
 exchange: NSE  
- tradingsymbol: NAVNETEDUL-EQ  
 exchange: NSE  
- tradingsymbol: RAILTEL-EQ  
 exchange: NSE  
- tradingsymbol: ABDL-EQ  
 exchange: NSE  
- tradingsymbol: MAITHANALL-EQ  
 exchange: NSE  
- tradingsymbol: MIDHANI-EQ  
 exchange: NSE  
- tradingsymbol: PNGJL-EQ  
 exchange: NSE  
- tradingsymbol: SANOFICONR-EQ  
 exchange: NSE  
- tradingsymbol: PGHH-EQ  
 exchange: NSE  
- tradingsymbol: ATUL-EQ  
 exchange: NSE  
- tradingsymbol: POLYPLEX-EQ  
 exchange: NSE  
- tradingsymbol: PRAJIND-EQ  
 exchange: NSE  
- tradingsymbol: SAGILITY-EQ  
 exchange: NSE  
- tradingsymbol: PRSMJOHNSN-EQ  
 exchange: NSE  
- tradingsymbol: SAILIFE-EQ  
 exchange: NSE  
- tradingsymbol: VENTIVE-EQ  
 exchange: NSE  
- tradingsymbol: ABFRL-EQ  
 exchange: NSE  
- tradingsymbol: SBIN-EQ  
 exchange: NSE  
- tradingsymbol: CAMS-EQ  
 exchange: NSE  
- tradingsymbol: NAM-INDIA-EQ  
 exchange: NSE  
- tradingsymbol: TATAMOTORS-EQ  
 exchange: NSE  
- tradingsymbol: FINEORG-EQ  
 exchange: NSE  
- tradingsymbol: HATSUN-EQ  
 exchange: NSE  
- tradingsymbol: NAVA-EQ  
 exchange: NSE  
- tradingsymbol: DYNAMATECH-EQ  
 exchange: NSE  
- tradingsymbol: BANKINDIA-EQ  
 exchange: NSE  
- tradingsymbol: KIMS-EQ  
 exchange: NSE  
- tradingsymbol: NUVOCO-EQ  
 exchange: NSE  
- tradingsymbol: CHEMPLASTS-EQ  
 exchange: NSE  
- tradingsymbol: GRSE-EQ  
 exchange: NSE  
- tradingsymbol: PARAS-EQ  
 exchange: NSE  
- tradingsymbol: GOODLUCK-EQ  
 exchange: NSE  
- tradingsymbol: FCL-EQ  
 exchange: NSE  
- tradingsymbol: SONATSOFTW-EQ  
 exchange: NSE  
- tradingsymbol: SJS-EQ  
 exchange: NSE  
- tradingsymbol: EXIDEIND-EQ  
 exchange: NSE  
- tradingsymbol: SAPPHIRE-EQ  
 exchange: NSE  
- tradingsymbol: CHOLAFIN-EQ  
 exchange: NSE  
- tradingsymbol: RATEGAIN-EQ  
 exchange: NSE  
- tradingsymbol: JUNIPER-EQ  
 exchange: NSE  
- tradingsymbol: BHARTIHEXA-EQ  
 exchange: NSE  
- tradingsymbol: TBOTEK-EQ  
 exchange: NSE  
- tradingsymbol: PEL-EQ  
 exchange: NSE  
- tradingsymbol: SYMPHONY-EQ  
 exchange: NSE  
- tradingsymbol: ASHIANA-EQ  
 exchange: NSE  
- tradingsymbol: AVANTEL-EQ  
 exchange: NSE  
- tradingsymbol: APLLTD-EQ  
 exchange: NSE  
- tradingsymbol: NORTHARC-EQ  
 exchange: NSE  
- tradingsymbol: TRITURBINE-EQ  
 exchange: NSE  
- tradingsymbol: APLAPOLLO-EQ  
 exchange: NSE  
- tradingsymbol: MTARTECH-EQ  
 exchange: NSE  
- tradingsymbol: ANURAS-EQ  
 exchange: NSE  
- tradingsymbol: RELIANCE-EQ  
 exchange: NSE  
- tradingsymbol: ITCHOTELS-EQ  
 exchange: NSE  
- tradingsymbol: MCLOUD-EQ  
 exchange: NSE  
- tradingsymbol: GUJTHEM-EQ  
 exchange: NSE  
- tradingsymbol: JUSTDIAL-EQ  
 exchange: NSE  
- tradingsymbol: SHREECEM-EQ  
 exchange: NSE  
- tradingsymbol: BAJFINANCE-EQ  
 exchange: NSE  
- tradingsymbol: TATASTEEL-EQ  
 exchange: NSE  
- tradingsymbol: CREDITACC-EQ  
 exchange: NSE  
- tradingsymbol: ENGINERSIN-EQ  
 exchange: NSE  
- tradingsymbol: ASAHIINDIA-EQ  
 exchange: NSE  
- tradingsymbol: ACUTAAS-EQ  
 exchange: NSE  
- tradingsymbol: CARBORUNIV-EQ  
 exchange: NSE  
- tradingsymbol: AXISBANK-EQ  
 exchange: NSE  
- tradingsymbol: ABSLAMC-EQ  
 exchange: NSE  
- tradingsymbol: POLICYBZR-EQ  
 exchange: NSE  
- tradingsymbol: GREENLAM-EQ  
 exchange: NSE  
- tradingsymbol: HCLTECH-EQ  
 exchange: NSE  
- tradingsymbol: METROBRAND-EQ  
 exchange: NSE  
- tradingsymbol: CGPOWER-EQ  
 exchange: NSE  
- tradingsymbol: KALPATARU-EQ  
 exchange: NSE  
- tradingsymbol: DABUR-EQ  
 exchange: NSE  
- tradingsymbol: INOXWIND-EQ  
 exchange: NSE  
- tradingsymbol: ZYDUSLIFE-EQ  
 exchange: NSE  
- tradingsymbol: GATEWAY-EQ  
 exchange: NSE  
- tradingsymbol: USHAMART-EQ  
 exchange: NSE  
- tradingsymbol: EQUITASBNK-EQ  
 exchange: NSE  
- tradingsymbol: ORCHPHARMA-EQ  
 exchange: NSE  
- tradingsymbol: TIPSMUSIC-EQ  
 exchange: NSE  
- tradingsymbol: STLTECH-EQ  
 exchange: NSE  
- tradingsymbol: AETHER-EQ  
 exchange: NSE  
- tradingsymbol: GRWRHITECH-EQ  
 exchange: NSE  
- tradingsymbol: TVSMOTOR-EQ  
 exchange: NSE  
- tradingsymbol: MSUMI-EQ  
 exchange: NSE  
- tradingsymbol: VRLLOG-EQ  
 exchange: NSE  
- tradingsymbol: GAEL-EQ  
 exchange: NSE  
- tradingsymbol: EICHERMOT-EQ  
 exchange: NSE  
- tradingsymbol: EIHOTEL-EQ  
 exchange: NSE  
- tradingsymbol: IOB-EQ  
 exchange: NSE  
- tradingsymbol: ESABINDIA-EQ  
 exchange: NSE  
- tradingsymbol: VENUSPIPES-EQ  
 exchange: NSE

## config\markets\india\watchlists\meanhunter.yaml

stocks:  
 - tradingsymbol: RELIANCE-EQ  
 exchange: NSE  
 - tradingsymbol: SBIN-EQ  
 exchange: NSE

## config\markets\usa\brokers\symbol\_mappings\interactive\_brokers.yaml

AAPL: "265598"  
MSFT: "272093"  
TSLA: "76792991"

## config\markets\usa\watchlists\master.yaml

stocks:  
 - tradingsymbol: AAPL  
 exchange: NASDAQ  
 - tradingsymbol: MSFT  
 exchange: NASDAQ  
 - tradingsymbol: TSLA  
 exchange: NASDAQ

## config\secrets\india\angelone\_cred.yaml

api\_key: mCMg8PVs  
client\_code: NEHR4113  
pin: 0786  
totp\_secret: LTOZ3GSPPJ46AT27MEAD2U6PI4

## config\strategies\meanhunter\_strategy.yaml

name: meanhunter  
timeframe: 5min # Switch to 1min, 15min, 30min, 1hr as needed  
watchlist:  
 india: config/markets/india/watchlists/meanhunter.yaml  
threshold: 0.75  
indicators:  
 - type: bollinger\_bands  
 name: bb  
 period: 20  
 std: 2.0  
 - type: rsi  
 name: rsi  
 period: 14  
entry\_rules:  
 - condition: "close < bb\_lower"  
 weight: 0.6  
 - condition: "rsi < 30"  
 weight: 0.2  
 - condition: "volume > volume\_threshold" # References market\_params from config.yaml  
 weight: 0.2  
exit\_rules:  
 - condition: "close > bb\_upper"  
 weight: 0.8  
stop\_loss:  
 type: multi  
 rules:  
 - type: fixed  
 value: "2%"  
target:  
 type: multi  
 rules:  
 - type: fixed  
 value: "5%"  
 partial\_exit: "50%"  
 id: "partial\_1"  
trade\_management:  
 breakeven:  
 trigger: 2.0  
market\_params:  
 india:  
 volume\_threshold: "avg\_volume\_20 \* 1.5" # Fallback if not in config.yaml

## scripts\create\_kafka\_topics.py

import os  
from confluent\_kafka.admin import AdminClient, NewTopic  
from src.py.util.config\_loader import load\_config  
from loguru import logger  
  
  
def create\_kafka\_topics(config):  
 """  
 Create Kafka topics with specified partitions from config.yaml, logging topic names and partitions.  
  
 Args:  
 config (Dict): Configuration dictionary from config.yaml.  
 """  
 # Configure logging  
 log\_dir = os.path.join("logs", "kafka")  
 os.makedirs(log\_dir, exist\_ok=True)  
 logger.add(  
 f"{log\_dir}/create\_topics.log",  
 rotation="10 MB",  
 retention="7 days",  
 level="DEBUG",  
 format="[{time:YYYY-MM-DD HH:mm:ss}] [{level}] topic={extra[topic]} {message}"  
 )  
  
 admin\_client = AdminClient({'bootstrap.servers': config['global']['kafka']['brokers']})  
  
 # Get existing topics  
 try:  
 existing\_topics = admin\_client.list\_topics(timeout=10).topics  
 existing\_topic\_info = {name: len(metadata.partitions) for name, metadata in existing\_topics.items()}  
 logger.debug(  
 f"Retrieved existing topics: {', '.join(f'{name} ({partitions} partitions)' for name, partitions in existing\_topic\_info.items())}",  
 topic="none")  
 except Exception as e:  
 logger.error(f"Failed to list existing topics: {e}", topic="none", exc\_info=True)  
 raise  
  
 # Define topics to create  
 topics = [  
 {  
 'name': config['global']['kafka']['topics']['india'],  
 'partitions': config['global']['kafka']['nse\_ticks\_partitions'],  
 'partition\_key': 'nse\_ticks\_partitions'  
 },  
 {  
 'name': config['global']['kafka']['topics']['ohlcv\_1min'],  
 'partitions': config['global']['kafka']['ohlcv\_1min\_partitions'],  
 'partition\_key': 'ohlcv\_1min\_partitions'  
 },  
 {  
 'name': config['global']['kafka']['topics']['ohlcv\_5min'],  
 'partitions': config['global']['kafka']['ohlcv\_5min\_partitions'],  
 'partition\_key': 'ohlcv\_5min\_partitions'  
 },  
 {  
 'name': config['global']['kafka']['topics']['ohlcv\_30min'],  
 'partitions': config['global']['kafka']['ohlcv\_30min\_partitions'],  
 'partition\_key': 'ohlcv\_30min\_partitions'  
 },  
 {  
 'name': config['global']['kafka']['topics']['signals'],  
 'partitions': config['global']['kafka']['signals\_partitions'],  
 'partition\_key': 'signals\_partitions'  
 },  
 {  
 'name': config['global']['kafka']['topics']['trades'],  
 'partitions': config['global']['kafka']['trades\_partitions'],  
 'partition\_key': 'trades\_partitions'  
 },  
 ]  
  
 # Filter out topics with correct partition count  
 new\_topics = []  
 for topic in topics:  
 topic\_name = topic['name']  
 expected\_partitions = topic['partitions']  
 if topic\_name in existing\_topic\_info:  
 current\_partitions = existing\_topic\_info[topic\_name]  
 if current\_partitions == expected\_partitions:  
 logger.info(f"Topic {topic\_name} already exists with correct {current\_partitions} partitions",  
 topic=topic\_name)  
 continue  
 else:  
 logger.warning(  
 f"Topic {topic\_name} exists with {current\_partitions} partitions, expected {expected\_partitions}. Cannot modify partitions.",  
 topic=topic\_name)  
 continue  
 new\_topics.append(NewTopic(topic\_name, num\_partitions=expected\_partitions, replication\_factor=1))  
  
 if not new\_topics:  
 logger.info(f"No new topics to create; all topics exist: {', '.join(existing\_topic\_info.keys())}", topic="none")  
 return  
  
 try:  
 futures = admin\_client.create\_topics(new\_topics)  
 for topic, future in futures.items():  
 try:  
 future.result()  
 logger.info(  
 f"Created topic {topic} with {config['global']['kafka'].get(f'{topic}\_partitions', 4)} partitions",  
 topic=topic)  
 except Exception as e:  
 if "Topic already exists" in str(e):  
 logger.warning(f"Topic {topic} already exists", topic=topic)  
 else:  
 logger.error(f"Failed to create topic {topic}: {e}", topic=topic, exc\_info=True)  
 # Continue to next topic  
 except Exception as e:  
 logger.error(f"Failed to initiate topic creation: {e}", topic="none", exc\_info=True)  
 # Continue instead of raising  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 config = load\_config("config/config.yaml")  
 create\_kafka\_topics(config)  
 except Exception as e:  
 logger.error(f"Script execution failed: {e}", topic="none", exc\_info=True)  
 raise

## scripts\fetch\_corporate\_actions.py

## scripts\prefetch\_historical\_data.py

from loguru import logger  
from src.py.util.logging\_config import configure\_logging  
import os  
import yaml  
import asyncio  
import pandas as pd  
import datetime  
from src.py.util.config\_loader import load\_config  
from src.py.messaging.redis\_client import RedisClient  
from src.anquant.py.util.database import Database  
from src.py.core.adapters.angelone import AngelOneAdapter  
  
# Configure global logging  
configure\_logging()  
  
  
async def fetch\_ohlcv(symbol, timeframe, lookback, adapter):  
 """Fetch OHLCV data via Angel One API or return mock data in offline mode."""  
 logger.debug(f"Fetching OHLCV for {symbol}:{timeframe}")  
 if adapter.offline\_mode:  
 return pd.DataFrame([  
 {  
 "timestamp": (datetime.datetime.now() - datetime.timedelta(  
 minutes=i \* (1 if timeframe == "1min" else 5 if timeframe == "5min" else 30))).strftime(  
 "%Y-%m-%d %H:%M:%S"),  
 "open": 100.0 - i,  
 "high": 100.0 - i + 5,  
 "low": 100.0 - i - 5,  
 "close": 100.0 - i,  
 "volume": 1000,  
 "tradingsymbol": symbol,  
 "symboltoken": adapter.mappings.get(symbol, "unknown"),  
 "exchange": "NSE"  
 } for i in range(lookback)  
 ])  
 from\_date = datetime.datetime.now() - datetime.timedelta(days=7)  
 to\_date = datetime.datetime.now()  
 return await adapter.fetch\_historical\_data(symbol, timeframe, from\_date, to\_date)  
  
  
async def prefetch\_historical\_data(config):  
 """Prefetch OHLCV data for stocks in master.yaml and store in Redis and PostgreSQL."""  
 redis\_client = RedisClient(config["global"]["redis"])  
 database = Database(config["global"]["database"])  
  
 # Resolve watchlist path  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), ".."))  
 watchlist\_path = os.path.join(project\_root, config["global"]["markets"]["india"]["watchlists"]["master"])  
 logger.debug(f"Attempting to load watchlist from: {watchlist\_path}")  
  
 try:  
 with open(watchlist\_path, 'r') as f:  
 watchlist = [stock['tradingsymbol'] for stock in yaml.safe\_load(f)['stocks']]  
 except FileNotFoundError as e:  
 logger.error(f"Failed to load watchlist from {watchlist\_path}: {e}")  
 raise  
  
 # Initialize Angel One adapter  
 try:  
 credentials\_path = config["global"]["markets"]["india"]["brokers"]["angelone"]["credentials"]  
 mappings\_path = config["global"]["markets"]["india"]["brokers"]["angelone"]["symbols"]  
 credentials = load\_config(os.path.join(project\_root, credentials\_path))  
 mappings = load\_config(os.path.join(project\_root, mappings\_path))  
 adapter = AngelOneAdapter(credentials, mappings, config["global"]["kafka"], config)  
 except Exception as e:  
 logger.error(f"Failed to initialize AngelOneAdapter: {e}")  
 raise  
  
 for symbol in watchlist[:10]: # Limit to 10 for testing  
 for timeframe in config["global"]["historical\_data"]["timeframes"]:  
 try:  
 ohlcv = await fetch\_ohlcv(symbol, timeframe, config["global"]["historical\_data"]["lookback\_candles"],  
 adapter)  
 df = pd.DataFrame(ohlcv)  
 await redis\_client.cache(f"{symbol}:ohlcv:{timeframe}", df.to\_dict('records'), ttl=86400)  
 database.save\_ohlcv(symbol, timeframe, df)  
 logger.info(f"Prefetched OHLCV for {symbol}:{timeframe}")  
 except Exception as e:  
 logger.error(f"Failed to prefetch OHLCV for {symbol}:{timeframe}: {e}")  
 continue # Continue with next symbol/timeframe  
 database.close()  
 await redis\_client.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 logger.debug(f"Loguru handlers before run: {logger.\_core.handlers}")  
 config = load\_config("config/config.yaml")  
 config['global']['offline\_mode'] = True  
 asyncio.run(prefetch\_historical\_data(config))

## src\\_\_init\_\_.py

## src\anquant\\_\_init\_\_.py

## src\anquant\py\main.py

# D:\AlphaNivesh\ANQuant\src\anquant\py\main.py  
import asyncio  
import yaml  
import os  
from typing import List  
from src.anquant.py.util.logging import setup\_logging  
from src.anquant.py.util.config\_loader import load\_config  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.util.database import Database  
from src.anquant.py.core.adapters import get\_adapters  
from anquant.py.data\_management.market\_data.market\_data\_engine import MarketDataEngine  
from anquant.py.indicators.indicator\_engine import IndicatorEngine  
from src.anquant.py.core.strategy.strategy\_engine import StrategyEngine # Updated import  
from src.anquant.py.core.risk\_management.engine import RiskManagementEngine  
from anquant.py.order\_management.order\_execution import OrderExecutionEngine  
from anquant.py.portfolio import PortfolioManager  
from anquant.py.historical\_data import HistoricalDataManager  
from anquant.py.corporate\_actions import CorporateActionManager  
from asyncio import iscoroutine, Future  
  
logger = setup\_logging("main", log\_type="general")  
  
async def load\_watchlist(watchlist\_path: str) -> List[str]:  
 """Load tradingsymbols from a watchlist file."""  
 try:  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "../../../"))  
 absolute\_path = os.path.join(project\_root, watchlist\_path)  
 logger.debug(f"Attempting to load watchlist from: {absolute\_path}")  
 with open(absolute\_path, 'r', encoding='utf-8') as f:  
 watchlist = yaml.safe\_load(f)['stocks']  
 return [stock['tradingsymbol'] for stock in watchlist]  
 except Exception as e:  
 logger.error(f"Failed to load watchlist from {watchlist\_path}: {e}")  
 raise  
  
async def main():  
 logger.debug("Starting ANQuant Trading application")  
 try:  
 config = load\_config("config/config.yaml")  
 redis\_client = RedisClient(config['global']['redis'])  
 database = Database(config["global"]["database"])  
 adapters = get\_adapters(config)  
 watchlists = {}  
 for market in config['global']['brokers']['active\_brokers']:  
 watchlist\_path = config['global']['markets'][market]['watchlists'][  
 'meanhunter' if market == 'india' else 'xyzstrategy']  
 watchlists[market] = await load\_watchlist(watchlist\_path)  
 logger.debug(f"Loaded watchlist for {market}: {watchlists[market]}")  
  
 components = [  
 MarketDataEngine(config, redis\_client, adapters, watchlists),  
 \*[IndicatorEngine(config, redis\_client, adapters, watchlists) for \_ in range(4)],  
 StrategyEngine(config, redis\_client),  
 RiskManagementEngine(config),  
 OrderExecutionEngine(config),  
 PortfolioManager(config, redis\_client, database),  
 HistoricalDataManager(config, redis\_client, database),  
 CorporateActionManager(config, redis\_client)  
 ]  
  
 logger.debug("Initializing components")  
 await asyncio.gather(\*(component.initialize() for component in components))  
  
 logger.debug("Starting components")  
 tasks = []  
 for component in components:  
 start\_result = component.start()  
 if isinstance(start\_result, Future):  
 tasks.append(start\_result)  
 elif iscoroutine(start\_result):  
 tasks.append(asyncio.create\_task(start\_result))  
 else:  
 raise TypeError(f"Unexpected start result type for {component}: {type(start\_result)}")  
  
 try:  
 await asyncio.sleep(5)  
 logger.info("Test run completed successfully")  
 except asyncio.CancelledError:  
 logger.info("Test run cancelled")  
 raise  
 except Exception as e:  
 logger.error(f"Application failed: {str(e)}", exc\_info=True)  
 raise  
 finally:  
 logger.info("Shutting down ANQuant application")  
 for component in components:  
 if hasattr(component, 'stop'):  
 try:  
 await component.stop()  
 except Exception as e:  
 logger.error(f"Failed to stop component {component}: {str(e)}")  
 for market, broker\_adapters in adapters.items():  
 for broker, adapter in broker\_adapters.items():  
 try:  
 await adapter.disconnect()  
 except Exception as e:  
 logger.error(f"Failed to disconnect broker {broker} for market {market}: {str(e)}")  
 await redis\_client.close()  
 database.close()  
 logger.info("Exiting ANQuant application")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 loop = asyncio.new\_event\_loop()  
 asyncio.set\_event\_loop(loop)  
 try:  
 loop.run\_until\_complete(main())  
 except KeyboardInterrupt:  
 logger.info("Received keyboard interrupt; shutting down")  
 tasks = [t for t in asyncio.all\_tasks(loop) if t is not asyncio.current\_task(loop)]  
 for task in tasks:  
 task.cancel()  
 loop.run\_until\_complete(asyncio.gather(\*tasks, return\_exceptions=True))  
 loop.run\_until\_complete(loop.shutdown\_asyncgens())  
 loop.run\_until\_complete(loop.shutdown\_default\_executor())  
 except Exception as e:  
 logger.error(f"Application failed: {str(e)}", exc\_info=True)  
 raise  
 finally:  
 loop.close()  
 logger.info("Event loop closed")

## src\anquant\py\\_\_init\_\_.py

## src\anquant\py\core\\_\_init\_\_.py

## src\anquant\py\core\adapters\angelone.py

# D:\AlphaNivesh\ANQuant\src\anquant\py\core\adapters\angelone.py  
import asyncio  
import json  
import time  
import queue  
from typing import Dict, List, Any, Callable, Optional  
from .base\_adapter import BaseAdapter  
from src.anquant.py.messaging.kafka\_client import KafkaClient  
from src.anquant.py.messaging.redis\_client import RedisClient  
from SmartApi import SmartConnect, SmartWebSocket  
import pyotp  
import pandas as pd  
from datetime import datetime  
from zoneinfo import ZoneInfo  
from functools import wraps  
from src.anquant.py.util.logging import setup\_logging  
import websocket  
  
logger = setup\_logging("angelone", log\_type="general")  
  
  
def rate\_limit(calls\_per\_second: float = 5):  
 """Rate limit decorator to control API calls."""  
  
 def decorator(func):  
 last\_call = 0  
  
 @wraps(func)  
 async def wrapper(\*args, \*\*kwargs):  
 nonlocal last\_call  
 elapsed = time.time() - last\_call  
 wait\_time = 1 / calls\_per\_second - elapsed  
 if wait\_time > 0:  
 await asyncio.sleep(wait\_time)  
 last\_call = time.time()  
 return await func(\*args, \*\*kwargs)  
  
 return wrapper  
  
 return decorator  
  
  
class AngelOneAdapter(BaseAdapter):  
 INTERVAL\_MAPPING = {  
 "1min": "ONE\_MINUTE",  
 "5min": "FIVE\_MINUTE",  
 "30min": "THIRTY\_MINUTE",  
 }  
  
 def \_\_init\_\_(self, credentials: Dict[str, str], mappings: Dict[str, str], kafka\_config: Dict[str, str],  
 config: Dict[str, Any]):  
 self.credentials = credentials  
 self.mappings = mappings  
 self.offline\_mode = config.get('global', {}).get('offline\_mode', False)  
 self.smart\_api = None if self.offline\_mode else SmartConnect(api\_key=credentials.get('api\_key', ''))  
 self.kafka\_client = None if self.offline\_mode else KafkaClient(kafka\_config)  
 self.redis\_client = RedisClient(config.get('global', {}).get('redis', {}))  
 self.watchlist = []  
 self.auth\_token = None  
 self.feed\_token = None  
 self.client\_code = None  
 self.websocket = None  
 self.running = False  
 self.authenticated = False  
 self.last\_auth\_time = 0  
 self.callback\_handler = None  
 self.message\_queue = queue.Queue()  
 self.processing\_task = None  
 logger.info(f"Initialized AngelOneAdapter, offline\_mode={self.offline\_mode}, symbols={len(self.mappings)}")  
  
 async def connect(self) -> None:  
 try:  
 await self.authenticate()  
 logger.info("Connected to Angel One")  
 except Exception as e:  
 logger.error(f"Failed to connect: {e}", exc\_info=True)  
 raise  
  
 async def authenticate(self, retries: int = 3, delay: int = 5) -> Dict[str, Any]:  
 if self.offline\_mode:  
 logger.debug("Skipping authentication in offline mode")  
 self.authenticated = True  
 self.feed\_token = "offline"  
 self.auth\_token = "offline"  
 self.client\_code = self.credentials.get('client\_code', 'mock')  
 return {'clientcode': self.client\_code, 'feedToken': 'offline', 'jwtToken': 'offline'}  
  
 current\_time = time.time()  
 if current\_time - self.last\_auth\_time < 15:  
 await asyncio.sleep(15 - (current\_time - self.last\_auth\_time))  
 logger.debug("Rate-limiting authentication")  
  
 if self.authenticated and self.feed\_token and self.auth\_token:  
 try:  
 if self.smart\_api is not None:  
 self.smart\_api.getProfile(self.auth\_token)  
 logger.debug("Reusing existing session")  
 return {'clientcode': self.client\_code, 'feedToken': self.feed\_token, 'jwtToken': self.auth\_token}  
 else:  
 logger.warning("SmartAPI is None, cannot reuse session")  
 self.authenticated = False  
 except Exception as e:  
 logger.warning(f"Existing session invalid: {e}")  
 self.authenticated = False  
  
 for attempt in range(1, retries + 1):  
 try:  
 start\_time = time.time()  
 if self.smart\_api is None:  
 raise ValueError("SmartAPI is not initialized")  
  
 totp = pyotp.TOTP(self.credentials['totp\_secret']).now()  
 session = self.smart\_api.generateSession(self.credentials['client\_code'], self.credentials['pin'], totp)  
  
 if isinstance(session, dict) and session.get('status', False) and session.get('data'):  
 session\_data = session['data']  
 if isinstance(session\_data, dict):  
 self.feed\_token = session\_data.get('feedToken')  
 self.auth\_token = session\_data.get('jwtToken')  
 self.client\_code = session\_data.get('clientcode')  
 self.authenticated = True  
 self.last\_auth\_time = time.time()  
 logger.info(f"Authentication successful, latency {time.time() - start\_time:.3f}s")  
 return session\_data  
 else:  
 logger.error(f"Invalid session data format: {type(session\_data)}")  
 raise ValueError(f"Invalid session data format: {type(session\_data)}")  
 else:  
 error\_message = session.get('message', 'Unknown error') if isinstance(session,  
 dict) else 'Invalid response format'  
 logger.error(f"Authentication failed: {error\_message}")  
 raise ValueError(f"Authentication failed: {error\_message}")  
 except Exception as e:  
 logger.error(f"Authentication attempt {attempt} failed: {e}", exc\_info=True)  
 if attempt < retries:  
 logger.info(f"Retrying in {delay}s")  
 await asyncio.sleep(delay)  
 else:  
 logger.error("Max authentication retries reached")  
 raise ValueError(f"Max retries reached: {str(e)}")  
  
 @rate\_limit(calls\_per\_second=5)  
 async def fetch\_historical\_data(self, symbol: str, timeframe: str, from\_date: datetime, to\_date: datetime) -> List[  
 Dict[str, Any]]:  
 try:  
 start\_time = time.time()  
 if self.offline\_mode:  
 logger.debug(f"Generating mock historical data for {symbol}")  
 data = [  
 {  
 "timestamp": (datetime.now(ZoneInfo("Asia/Kolkata")) - pd.Timedelta(minutes=i)).strftime(  
 "%Y-%m-%d %H:%M:%S"),  
 "open": 3000.0 - i \* 10,  
 "high": 3000.0 - i \* 10 + 50,  
 "low": 2950.0 - i \* 10,  
 "close": 3000.0 - i \* 10 + (100 if i == 0 else 0),  
 "volume": 1000 + i \* 100,  
 "tradingsymbol": symbol,  
 "symboltoken": self.mappings.get(symbol, "mock"),  
 "exchange": "NSE"  
 } for i in range(20)  
 ]  
 await self.redis\_client.publish(f"historical\_data:NSE:{symbol}", json.dumps(data))  
 logger.debug(  
 f"Published mock historical data, size {len(json.dumps(data))} bytes, latency {time.time() - start\_time:.3f}s")  
 return data  
  
 await self.authenticate()  
 params = {  
 "exchange": "NSE",  
 "symboltoken": self.mappings.get(symbol, ""),  
 "interval": self.INTERVAL\_MAPPING.get(timeframe, "ONE\_MINUTE"),  
 "fromdate": from\_date.strftime("%Y-%m-%d %H:%M"),  
 "todate": to\_date.strftime("%Y-%m-%d %H:%M")  
 }  
 if not params["symboltoken"]:  
 logger.error(f"No symboltoken for {symbol}")  
 raise ValueError(f"No symboltoken for {symbol}")  
  
 if self.smart\_api is None:  
 raise ValueError("SmartAPI is not initialized")  
  
 response = self.smart\_api.getCandleData(params)  
 if isinstance(response, dict) and response.get('status', False):  
 response\_data = response.get('data', [])  
 if isinstance(response\_data, list):  
 data = [  
 {  
 "timestamp": d[0],  
 "open": float(d[1]),  
 "high": float(d[2]),  
 "low": float(d[3]),  
 "close": float(d[4]),  
 "volume": int(d[5]),  
 "tradingsymbol": symbol,  
 "symboltoken": params["symboltoken"],  
 "exchange": "NSE"  
 } for d in response\_data  
 ]  
 logger.info(f"Fetched {len(data)} candles, latency {time.time() - start\_time:.3f}s")  
 return data  
 else:  
 logger.error(f"Invalid response data format: {type(response\_data)}")  
 raise Exception(f"Invalid response data format: {type(response\_data)}")  
 else:  
 error\_message = response.get('message', 'Unknown error') if isinstance(response,  
 dict) else 'Invalid response format'  
 logger.error(f"Historical data fetch failed: {error\_message}")  
 raise Exception(f"Historical data fetch failed: {error\_message}")  
 except Exception as e:  
 logger.error(f"Failed to fetch historical data for {symbol}: {e}", exc\_info=True)  
 raise  
  
 async def subscribe\_to\_ticks(self, watchlist: List[str]) -> None:  
 try:  
 start\_time = time.time()  
 if not self.feed\_token:  
 logger.error("Not connected. Call connect() first.")  
 raise Exception("Not connected. Call connect() first.")  
  
 self.watchlist = [  
 {"tradingsymbol": symbol.upper(), "symboltoken": self.mappings.get(symbol, ""), "exchange": "NSE"}  
 for symbol in watchlist  
 ]  
 invalid\_symbols = [item['tradingsymbol'] for item in self.watchlist if not item['symboltoken']]  
 if invalid\_symbols:  
 logger.error(f"Invalid symbols in watchlist: {invalid\_symbols}")  
 raise ValueError(f"Invalid symbols: {invalid\_symbols}")  
  
 logger.debug(f"Subscribing to {len(self.watchlist)} symbols")  
 await self.start\_websocket\_async(self.\_process\_tick)  
 logger.info(f"Subscribed to ticks, latency {time.time() - start\_time:.3f}s")  
 except Exception as e:  
 logger.error(f"Failed to subscribe to ticks: {e}", exc\_info=True)  
 raise  
  
 async def \_process\_tick(self, tick: Dict[str, Any]) -> None:  
 try:  
 start\_time = time.time()  
 symbol = tick['tradingsymbol']  
 logger.debug(f"[TRACE] Entered \_process\_tick for {symbol}")  
  
 # Publish to Redis  
 try:  
 logger.debug(f"[TRACE] Before await redis\_client.publish for {symbol}")  
 await self.redis\_client.publish(f"ticks:NSE:{symbol}", json.dumps(tick))  
 logger.debug(f"[TRACE] After await redis\_client.publish for {symbol}")  
 logger.debug(f"Redis publish completed for {symbol}")  
 except Exception as redis\_error:  
 logger.error(f"[TRACE] Redis publish failed for {symbol}: {redis\_error}", exc\_info=True)  
 # Continue with Kafka even if Redis fails  
  
 logger.debug(f"[TRACE] After Redis publish, before Kafka produce for {symbol}")  
 logger.debug(  
 f"[TRACE] kafka\_client is {'set' if self.kafka\_client else 'None'} in \_process\_tick for {symbol}")  
  
 if self.kafka\_client:  
 try:  
 logger.debug(f"[TRACE] Before Kafka produce for {symbol}")  
 logger.debug(f"[TRACE] Kafka client type: {type(self.kafka\_client)}")  
 logger.debug(f"[TRACE] About to call kafka\_client.produce with topic=nse\_ticks, key={symbol}")  
 logger.debug(f"[TRACE] Tick type: {type(tick)}, value: {tick}")  
 self.kafka\_client.produce('nse\_ticks', symbol, tick)  
 logger.debug(f"[TRACE] After Kafka produce for {symbol}")  
 logger.debug(f"[TRACE] Kafka produce completed for {symbol}")  
 except Exception as kafka\_error:  
 logger.error(f"[TRACE] Kafka publish failed for {symbol}: {kafka\_error}", exc\_info=True)  
  
 logger.debug(f"[TRACE] Processed tick for {symbol}, latency {time.time() - start\_time:.3f}s")  
 except Exception as e:  
 logger.error(f"[TRACE] Failed to process tick for {symbol}: {e}", exc\_info=True)  
 raise  
  
 async def unsubscribe\_from\_ticks(self, watchlist: List[str]) -> None:  
 try:  
 start\_time = time.time()  
 self.stop\_websocket()  
 logger.info(f"Unsubscribed from {len(watchlist)} symbols, latency {time.time() - start\_time:.3f}s")  
 except Exception as e:  
 logger.error(f"Failed to unsubscribe: {e}", exc\_info=True)  
 raise  
  
 async def place\_order(self, order\_details: Dict[str, Any]) -> str:  
 try:  
 start\_time = time.time()  
 symbol = order\_details.get('tradingsymbol', 'unknown')  
 if self.offline\_mode:  
 logger.debug(f"Simulating order placement for {symbol}")  
 return f"mock\_order\_{symbol}"  
  
 await self.authenticate()  
 order\_details['symboltoken'] = self.mappings.get(symbol, "")  
 order\_details['exchange'] = order\_details.get('exchange', 'NSE')  
 if not order\_details['symboltoken']:  
 logger.error(f"No symboltoken for {symbol}")  
 raise ValueError(f"No symboltoken for {symbol}")  
  
 order\_id = self.smart\_api.placeOrder(order\_details)  
 logger.info(f"Placed order {order\_id} for {symbol}, latency {time.time() - start\_time:.3f}s")  
 return order\_id  
 except Exception as e:  
 logger.error(f"Failed to place order for {symbol}: {e}", exc\_info=True)  
 raise  
  
 async def cancel\_order(self, order\_id: str) -> None:  
 try:  
 start\_time = time.time()  
 self.smart\_api.cancelOrder(order\_id, variety="NORMAL")  
 logger.info(f"Cancelled order {order\_id}, latency {time.time() - start\_time:.3f}s")  
 except Exception as e:  
 logger.error(f"Failed to cancel order {order\_id}: {e}", exc\_info=True)  
 raise  
  
 async def get\_order\_status(self, order\_id: str) -> Dict[str, Any]:  
 try:  
 start\_time = time.time()  
 order\_book = self.smart\_api.orderBook()  
 for order in order\_book.get('data', []):  
 if order['orderid'] == order\_id:  
 logger.debug(f"Retrieved order status for {order\_id}, latency {time.time() - start\_time:.3f}s")  
 return order  
 logger.warning(f"Order {order\_id} not found")  
 return {}  
 except Exception as e:  
 logger.error(f"Failed to get order status for {order\_id}: {e}", exc\_info=True)  
 raise  
  
 async def get\_positions(self) -> List[Dict[str, Any]]:  
 try:  
 start\_time = time.time()  
 response = self.smart\_api.position()  
 positions = response.get('data', []) if response.get('status', False) else []  
 logger.debug(f"Retrieved {len(positions)} positions, latency {time.time() - start\_time:.3f}s")  
 return positions  
 except Exception as e:  
 logger.error(f"Failed to get positions: {e}", exc\_info=True)  
 raise  
  
 async def get\_account\_info(self) -> Dict[str, Any]:  
 try:  
 start\_time = time.time()  
 response = self.smart\_api.rmsLimit()  
 account\_info = response.get('data', {}) if response.get('status', False) else {}  
 logger.debug(f"Retrieved account info, latency {time.time() - start\_time:.3f}s")  
 return account\_info  
 except Exception as e:  
 logger.error(f"Failed to get account info: {e}", exc\_info=True)  
 raise  
  
 async def disconnect(self) -> None:  
 try:  
 start\_time = time.time()  
 self.stop\_websocket()  
 if not self.offline\_mode:  
 self.smart\_api.terminateSession(self.credentials['client\_code'])  
 logger.info(f"Disconnected, latency {time.time() - start\_time:.3f}s")  
 except Exception as e:  
 logger.error(f"Failed to disconnect: {e}", exc\_info=True)  
 raise  
  
 async def get\_quote(self, symbol: str, max\_retries: int = 3, retry\_delay: int = 5) -> Dict[str, Any]:  
 try:  
 start\_time = time.time()  
 if self.offline\_mode:  
 logger.debug(f"Returning mock quote for {symbol}")  
 return {  
 'status': True,  
 'data': {  
 'high': 3050.0,  
 'low': 2950.0,  
 'close': 3000.0  
 }  
 }  
  
 for attempt in range(1, max\_retries + 1):  
 try:  
 await self.authenticate()  
 symboltoken = self.mappings.get(symbol, "")  
 if not symboltoken:  
 logger.error(f"No symboltoken for {symbol}")  
 raise ValueError(f"No symboltoken for {symbol}")  
 ltp\_data = self.smart\_api.getLtpData(exchange="NSE", tradingsymbol=symbol, symboltoken=symboltoken)  
 if ltp\_data.get('status', False) and ltp\_data.get('data'):  
 logger.debug(f"Retrieved quote for {symbol}, latency {time.time() - start\_time:.3f}s")  
 return {  
 'status': True,  
 'data': {  
 'fetched': [{  
 'token': symboltoken,  
 'tradingsymbol': symbol,  
 'ltp': ltp\_data['data'].get('ltp', 0.0),  
 'volume': ltp\_data['data'].get('vtt', 0),  
 'close': ltp\_data['data'].get('close', 0.0)  
 }]  
 }  
 }  
 logger.warning(f"No LTP data for {symbol} on attempt {attempt}")  
 if attempt < max\_retries:  
 await asyncio.sleep(retry\_delay)  
 except Exception as e:  
 logger.error(f"Quote fetch attempt {attempt} failed for {symbol}: {e}", exc\_info=True)  
 if attempt >= max\_retries:  
 raise ValueError(f"Max retries reached: {str(e)}")  
 return {}  
 except Exception as e:  
 logger.error(f"Failed to get quote for {symbol}: {e}", exc\_info=True)  
 raise  
  
 async def start\_websocket\_async(self, callback\_handler: Callable) -> None:  
 if self.offline\_mode:  
 logger.debug("Skipping WebSocket in offline mode")  
 return  
  
 try:  
 start\_time = time.time()  
 self.callback\_handler = callback\_handler  
 ws\_url = (  
 f"wss://smartapisocket.angelone.in/smart-stream?"  
 f"clientCode={self.client\_code}&feedToken={self.feed\_token}&apiKey={self.credentials['api\_key']}"  
 )  
 self.websocket = websocket.WebSocketApp(  
 ws\_url,  
 on\_open=self.\_on\_open,  
 on\_message=self.\_on\_message,  
 on\_error=self.\_on\_error,  
 on\_close=self.\_on\_close  
 )  
 self.running = True  
  
 def run\_websocket():  
 self.websocket.run\_forever(ping\_interval=0, ping\_timeout=None)  
  
 loop = asyncio.get\_event\_loop()  
 loop.run\_in\_executor(None, run\_websocket)  
  
 self.processing\_task = asyncio.create\_task(self.\_process\_message\_queue())  
  
 await asyncio.sleep(1)  
 logger.info(f"Started WebSocket, latency {time.time() - start\_time:.3f}s")  
 except Exception as e:  
 logger.error(f"Failed to start WebSocket: {e}", exc\_info=True)  
 self.running = False  
 raise  
  
 async def \_process\_message\_queue(self):  
 try:  
 while self.running:  
 try:  
 tick = self.message\_queue.get(timeout=1.0)  
 if tick and self.callback\_handler:  
 logger.debug(f"Calling callback handler for tick: {tick['tradingsymbol']}")  
 await self.callback\_handler(tick)  
 logger.debug(f"Callback handler completed for tick: {tick['tradingsymbol']}")  
 else:  
 if not tick:  
 logger.debug("No tick data received from queue")  
 if not self.callback\_handler:  
 logger.warning("No callback handler set")  
 except queue.Empty:  
 continue  
 except Exception as e:  
 logger.error(f"Error processing message from queue: {e}", exc\_info=True)  
 except Exception as e:  
 logger.error(f"Error in message queue processing: {e}", exc\_info=True)  
  
 def \_on\_open(self, ws):  
 try:  
 subscription = {  
 "correlationID": "anquant\_ticks",  
 "action": 1,  
 "params": {  
 "mode": 2,  
 "tokenList": [{"exchangeType": 1, "tokens": [item["symboltoken"] for item in self.watchlist]}]  
 }  
 }  
 ws.send(json.dumps(subscription))  
 logger.info(f"Subscribed to {len(self.watchlist)} tokens")  
 except Exception as e:  
 logger.error(f"WebSocket subscription failed: {e}", exc\_info=True)  
 self.running = False  
  
 def \_on\_message(self, ws, message: Any):  
 try:  
 if message == "pong":  
 logger.debug("Received pong heartbeat")  
 return  
 subscription\_mode = int.from\_bytes(message[0:1], "little")  
 exchange\_type = int.from\_bytes(message[1:2], "little")  
 token = message[2:27].decode("utf-8").rstrip("\x00")  
 tradingsymbol = next((item["tradingsymbol"] for item in self.watchlist if item["symboltoken"] == token),  
 token)  
  
 timestamp\_ms = int.from\_bytes(message[35:43], "little")  
 timestamp = datetime.fromtimestamp(timestamp\_ms / 1000, tz=ZoneInfo("Asia/Kolkata"))  
  
 tick = {  
 "tradingsymbol": tradingsymbol,  
 "symboltoken": token,  
 "ltp": int.from\_bytes(message[43:51], "little") / 100,  
 "volume": int.from\_bytes(message[51:59], "little"),  
 "timestamp": timestamp.isoformat(),  
 "exchange": {1: "NSE", 3: "BSE", 5: "MCX"}.get(exchange\_type, "NSE")  
 }  
 if subscription\_mode == 2:  
 try:  
 tick.update({  
 "open": int.from\_bytes(message[59:67], "little") / 100,  
 "high": int.from\_bytes(message[67:75], "little") / 100,  
 "low": int.from\_bytes(message[75:83], "little") / 100,  
 "close": int.from\_bytes(message[83:91], "little") / 100,  
 })  
 except IndexError:  
 logger.warning(f"Incomplete FULL mode data for {tradingsymbol}")  
 self.message\_queue.put(tick)  
 logger.debug(f"Processed tick for {tradingsymbol}")  
 except Exception as e:  
 logger.error(f"Failed to process WebSocket message for {tradingsymbol}: {e}", exc\_info=True)  
  
 def \_on\_error(self, ws, error: Any):  
 logger.error(f"WebSocket error: {error}", exc\_info=True)  
 self.running = False  
  
 def \_on\_close(self, ws, code=None, reason=None):  
 logger.info(f"WebSocket closed: code={code}, reason={reason}")  
 self.running = False  
 self.websocket = None  
  
 def stop\_websocket(self) -> None:  
 try:  
 if self.offline\_mode:  
 logger.debug("No WebSocket to stop in offline mode")  
 return  
 if self.websocket and self.running:  
 self.running = False  
 self.websocket.close()  
  
 if self.processing\_task and not self.processing\_task.done():  
 self.processing\_task.cancel()  
  
 logger.info("Stopped WebSocket")  
 except Exception as e:  
 logger.error(f"Failed to stop WebSocket: {e}", exc\_info=True)

## src\anquant\py\core\adapters\base\_adapter.py

from abc import ABC, abstractmethod  
from typing import List, Dict, Any  
import datetime  
  
class BaseAdapter(ABC):  
 @abstractmethod  
 def \_\_init\_\_(self, credentials: Dict[str, str], mappings: Dict[str, str], kafka\_config: Dict[str, str], config: Dict[str, Any]):  
 """Initialize the adapter with credentials, symbol mappings, Kafka config, and general config."""  
 pass  
  
 @abstractmethod  
 async def connect(self) -> None:  
 """Establish connection to the broker and authenticate."""  
 pass  
  
 @abstractmethod  
 async def subscribe\_to\_ticks(self, watchlist: List[str]) -> None:  
 """Subscribe to real-time tick data for the given watchlist of symbols."""  
 pass  
  
 @abstractmethod  
 async def unsubscribe\_from\_ticks(self, watchlist: List[str]) -> None:  
 """Unsubscribe from real-time tick data for the given watchlist."""  
 pass  
  
 @abstractmethod  
 async def place\_order(self, order\_details: Dict[str, Any]) -> str:  
 """Place an order with the given details and return the order ID."""  
 pass  
  
 @abstractmethod  
 async def cancel\_order(self, order\_id: str) -> None:  
 """Cancel the order with the given order ID."""  
 pass  
  
 @abstractmethod  
 async def get\_order\_status(self, order\_id: str) -> Dict[str, Any]:  
 """Get the status of the order with the given order ID."""  
 pass  
  
 @abstractmethod  
 async def fetch\_historical\_data(self, symbol: str, timeframe: str, from\_date: datetime.datetime, to\_date: datetime.datetime) -> List[Dict[str, Any]]:  
 """Fetch historical OHLCV data for the symbol in the given timeframe and date range."""  
 pass  
  
 @abstractmethod  
 async def get\_positions(self) -> List[Dict[str, Any]]:  
 """Get current positions and P&L."""  
 pass  
  
 @abstractmethod  
 async def get\_account\_info(self) -> Dict[str, Any]:  
 """Get account balance and margins."""  
 pass  
  
 @abstractmethod  
 async def disconnect(self) -> None:  
 """Cleanly disconnect from the broker."""  
 pass  
  
 @abstractmethod  
 async def get\_quote(self, symbol: str, max\_retries: int = 3, retry\_delay: int = 5) -> Dict[str, Any]:  
 """Get real-time quote for the symbol."""  
 pass

## src\anquant\py\core\adapters\interactive\_brokers.py

# src/py/core/adapters/interactive\_brokers.py  
import json  
from .base\_adapter import BaseAdapter  
from typing import List, Dict, Any  
import datetime  
from loguru import logger  
from confluent\_kafka import Producer  
from src.py.messaging.redis\_client import RedisClient  
  
class InteractiveBrokersAdapter(BaseAdapter):  
 def \_\_init\_\_(self, credentials: Dict[str, str], mappings: Dict[str, str], kafka\_config: Dict[str, str], config: Dict[str, Any]):  
 self.credentials = credentials # {'account\_id': ..., 'api\_key': ..., 'secret\_key': ..., 'host': ..., 'port': ..., 'client\_id': ...}  
 self.mappings = mappings # {'AAPL': '265598', ...}  
 self.offline\_mode = config.get('global', {}).get('offline\_mode', False)  
 if not self.offline\_mode:  
 kafka\_producer\_config = {'bootstrap.servers': kafka\_config.get('brokers', 'localhost:9092')}  
 self.kafka\_producer = Producer(kafka\_producer\_config)  
 else:  
 self.kafka\_producer = None  
 self.redis\_client = RedisClient(config.get('redis', {})) if not self.offline\_mode else None  
 logger.debug(f"Initialized InteractiveBrokersAdapter with mappings for {len(self.mappings)} symbols")  
  
 async def connect(self) -> None:  
 logger.info("Connecting to Interactive Brokers (stub)")  
 if self.offline\_mode:  
 logger.debug("Skipping connection in offline mode")  
  
 async def subscribe\_to\_ticks(self, watchlist: List[str]) -> None:  
 if self.offline\_mode:  
 logger.debug("Skipping tick subscription in offline mode")  
 return  
 async def callback(tick):  
 try:  
 if self.redis\_client:  
 await self.redis\_client.publish(f"ticks:{tick['exchange']}:{tick['tradingsymbol']}", json.dumps(tick))  
 if self.kafka\_producer:  
 self.kafka\_producer.produce('us\_ticks', key=tick['tradingsymbol'], value=json.dumps(tick))  
 self.kafka\_producer.flush()  
 logger.debug(f"Published tick for {tick['tradingsymbol']}")  
 except Exception as e:  
 logger.error(f"Failed to publish tick: {e}")  
 logger.info(f"Subscribing to ticks for {watchlist} (stub)")  
 # Implement IB tick subscription using ib\_insync or IB API  
  
 async def unsubscribe\_from\_ticks(self, watchlist: List[str]) -> None:  
 logger.info(f"Unsubscribing from ticks for {watchlist} (stub)")  
  
 async def place\_order(self, order\_details: Dict[str, Any]) -> str:  
 symbol = order\_details.get('tradingsymbol')  
 symboltoken = self.mappings.get(symbol)  
 if not symboltoken:  
 logger.error(f"No symboltoken for {symbol} in mappings")  
 raise ValueError(f"No symboltoken for {symbol}")  
 logger.info(f"Placing order {order\_details} (stub)")  
 return "mock\_order\_id" if self.offline\_mode else ""  
  
 async def cancel\_order(self, order\_id: str) -> None:  
 logger.info(f"Cancelling order {order\_id} (stub)")  
  
 async def get\_order\_status(self, order\_id: str) -> Dict[str, Any]:  
 logger.info(f"Getting order status for {order\_id} (stub)")  
 return {}  
  
 async def fetch\_historical\_data(self, symbol: str, timeframe: str, from\_date: datetime.datetime, to\_date: datetime.datetime) -> List[Dict[str, Any]]:  
 symboltoken = self.mappings.get(symbol)  
 if not symboltoken:  
 logger.error(f"No symboltoken for {symbol} in mappings")  
 raise ValueError(f"No symboltoken for {symbol}")  
 logger.info(f"Fetching historical data for {symbol} (stub)")  
 return [] if not self.offline\_mode else [  
 {  
 "timestamp": (datetime.datetime.now() - datetime.timedelta(minutes=i)).strftime("%Y-%m-%d %H:%M:%S"),  
 "open": 100.0 - i,  
 "high": 100.0 - i + 5,  
 "low": 100.0 - i - 5,  
 "close": 100.0 - i,  
 "volume": 1000,  
 "tradingsymbol": symbol,  
 "symboltoken": symboltoken,  
 "exchange": "NASDAQ"  
 } for i in range(20)  
 ]  
  
 async def get\_positions(self) -> List[Dict[str, Any]]:  
 logger.info("Getting positions (stub)")  
 return []  
  
 async def get\_account\_info(self) -> Dict[str, Any]:  
 logger.info("Getting account info (stub)")  
 return {}  
  
 async def disconnect(self) -> None:  
 logger.info("Disconnecting from Interactive Brokers (stub)")  
 if self.redis\_client:  
 await self.redis\_client.close()  
  
 async def get\_quote(self, symbol: str, max\_retries: int = 3, retry\_delay: int = 5) -> Dict[str, Any]:  
 symboltoken = self.mappings.get(symbol)  
 if not symboltoken:  
 logger.error(f"No symboltoken for {symbol} in mappings")  
 raise ValueError(f"No symboltoken for {symbol}")  
 logger.info(f"Getting quote for {symbol} (stub)")  
 return {"status": True, "data": {"high": 105.0, "low": 95.0, "close": 100.0}} if self.offline\_mode else {}

## src\anquant\py\core\adapters\\_\_init\_\_.py

from typing import Dict  
from src.anquant.py.util.config\_loader import load\_credentials, load\_symbol\_mappings  
from .base\_adapter import BaseAdapter  
from .angelone import AngelOneAdapter  
try:  
 from .interactive\_brokers import InteractiveBrokersAdapter  
except ImportError:  
 InteractiveBrokersAdapter = None # Fallback if not implemented  
from src.anquant.py.util.config\_loader import load\_credentials, load\_symbol\_mappings  
  
from loguru import logger  
  
def get\_adapters(config: Dict) -> Dict[str, Dict[str, BaseAdapter]]:  
 """  
 Factory function to return a dictionary of broker adapters for each market and active broker.  
 Returns: {market: {broker: adapter\_instance}}  
 """  
 try:  
 adapters = {}  
 active\_brokers = config['global']['brokers']['active\_brokers']  
 for market, brokers in active\_brokers.items():  
 adapters[market] = {}  
 for broker in brokers:  
 credentials = load\_credentials(config, market, broker)  
 mappings = load\_symbol\_mappings(config, market, broker)  
 kafka\_config = config['global']['kafka']  
 if broker == 'angelone':  
 logger.info(f"Loading AngelOneAdapter for market: {market}")  
 adapters[market][broker] = AngelOneAdapter(credentials, mappings, kafka\_config, config)  
 elif broker == 'interactive\_brokers':  
 if InteractiveBrokersAdapter is None:  
 logger.warning(f"InteractiveBrokersAdapter not available for market: {market}")  
 continue  
 logger.info(f"Loading InteractiveBrokersAdapter for market: {market}")  
 adapters[market][broker] = InteractiveBrokersAdapter(credentials, mappings, kafka\_config, config)  
 else:  
 logger.error(f"Unknown broker: {broker} for market: {market}")  
 raise ValueError(f"Unknown broker: {broker}")  
 logger.info(f"Loaded adapters for markets: {list(adapters.keys())}")  
 return adapters  
 except Exception as e:  
 logger.error(f"Failed to load adapters: {e}")  
 raise

## src\anquant\py\core\flexirule\rule\_engine.py

# D:\AlphaNivesh\ANQuant\src\anquant\py\core\flexirule\rule\_engine.py  
import asyncio  
import json  
from typing import Dict, Any, Optional, List  
import operator  
import pandas as pd  
import pandas\_ta as pta  
from src.anquant.py.util.logging import setup\_logging  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.util.database import Database  
import asteval  
  
logger = setup\_logging("rule\_engine", log\_type="strategy")  
  
class RuleEngine:  
 def \_\_init\_\_(self, strategy\_config: Dict[str, Any], redis\_client: RedisClient):  
 self.strategy\_config = strategy\_config  
 self.name = strategy\_config['name']  
 self.timeframe = strategy\_config['timeframe']  
 self.threshold = strategy\_config.get('threshold', 0.75)  
 self.indicators = strategy\_config.get('indicators', [])  
 self.patterns = strategy\_config.get('patterns', [])  
 self.entry\_rules = strategy\_config.get('entry\_rules', [])  
 self.exit\_rules = strategy\_config.get('exit\_rules', [])  
 self.stop\_loss = strategy\_config.get('stop\_loss', {})  
 self.target = strategy\_config.get('target', {})  
 self.trade\_management = strategy\_config.get('trade\_management', {})  
 self.market\_params = strategy\_config.get('market\_params', {}).get('india', {})  
 self.redis\_client = redis\_client  
 self.database = Database(strategy\_config.get('database', {}))  
 self.positions = {} # symbol: {entry\_price, highest\_price, lowest\_price, breakeven\_triggered}  
 self.pattern\_cache = {} # symbol: {pattern\_name: bool}  
 self.interpreter = asteval.Interpreter()  
 logger.debug(f"Initialized RuleEngine for strategy {self.name}")  
  
 async def initialize(self):  
 """Initialize Redis and database connections."""  
 logger.info(f"Initializing RuleEngine for strategy {self.name}")  
 try:  
 await self.redis\_client.connect()  
 await self.redis\_client.redis.ping()  
 logger.debug(f"Verified Redis connectivity for RuleEngine {self.name}")  
 await self.database.connect()  
 for rule in self.entry\_rules + self.exit\_rules:  
 if 'condition' not in rule or 'weight' not in rule:  
 logger.error(f"Invalid rule format in strategy {self.name}: {rule}")  
 raise ValueError(f"Invalid rule format in strategy {self.name}")  
 for indicator in self.indicators:  
 if 'type' not in indicator or 'name' not in indicator:  
 logger.error(f"Invalid indicator format in strategy {self.name}: {indicator}")  
 raise ValueError(f"Invalid indicator format in strategy {self.name}")  
 for pattern in self.patterns:  
 if 'type' not in pattern or 'name' not in pattern:  
 logger.error(f"Invalid pattern format in strategy {self.name}: {pattern}")  
 raise ValueError(f"Invalid pattern format in strategy {self.name}")  
 logger.info(f"RuleEngine for {self.name} initialized successfully")  
 except Exception as e:  
 logger.error(f"Failed to initialize RuleEngine for {self.name}: {e}", exc\_info=True)  
 raise  
  
 async def evaluate(self, symbol: str, ohlcv: Dict, indicators: Dict) -> Optional[str]:  
 """Evaluate rules and generate trading signal."""  
 try:  
 position = self.positions.get(symbol, {  
 'entry\_price': None,  
 'highest\_price': None,  
 'lowest\_price': None,  
 'breakeven\_triggered': False  
 })  
 close = ohlcv['close']  
  
 # Update pattern cache  
 await self.\_update\_patterns(symbol, ohlcv)  
  
 # Check stop-loss and target for open position  
 if position['entry\_price'] is not None:  
 # Check breakeven  
 breakeven\_signal = self.\_evaluate\_breakeven(symbol, close, position)  
 if breakeven\_signal:  
 position['breakeven\_triggered'] = True  
 self.positions[symbol] = position  
  
 sl\_signal = await self.\_evaluate\_stop\_loss(symbol, close, position)  
 if sl\_signal:  
 self.positions.pop(symbol, None)  
 await self.\_log\_audit\_trail(symbol, ohlcv, indicators, sl\_signal, f"Stop-loss triggered ({sl\_signal})")  
 return sl\_signal  
 target\_signal = await self.\_evaluate\_target(symbol, close, position)  
 if target\_signal:  
 if target\_signal.startswith("PARTIAL\_SELL"):  
 position['quantity'] = position.get('quantity', 100) \* (1 - float(target\_signal.split(':')[1]) / 100)  
 self.positions[symbol] = position  
 else:  
 self.positions.pop(symbol, None)  
 await self.\_log\_audit\_trail(symbol, ohlcv, indicators, target\_signal, f"Target triggered ({target\_signal})")  
 return target\_signal  
 exit\_signal = self.\_evaluate\_rules(self.exit\_rules, ohlcv, indicators)  
 if exit\_signal:  
 self.positions.pop(symbol, None)  
 await self.\_log\_audit\_trail(symbol, ohlcv, indicators, "SELL", "Exit rule triggered")  
 return "SELL"  
  
 # Check entry rules  
 entry\_signal = self.\_evaluate\_rules(self.entry\_rules, ohlcv, indicators)  
 if entry\_signal:  
 self.positions[symbol] = {  
 'entry\_price': close,  
 'highest\_price': close,  
 'lowest\_price': close,  
 'breakeven\_triggered': False,  
 'quantity': 100 # Default quantity  
 }  
 await self.\_log\_audit\_trail(symbol, ohlcv, indicators, "BUY", "Entry rule triggered")  
 return "BUY"  
  
 return "HOLD"  
 except Exception as e:  
 logger.error(f"Error evaluating rules for {symbol} in {self.name}: {e}", exc\_info=True)  
 return "HOLD"  
  
 async def \_update\_patterns(self, symbol: str, ohlcv: Dict):  
 """Update pattern cache for the symbol."""  
 try:  
 for pattern in self.patterns:  
 pattern\_name = pattern['name']  
 pattern\_type = pattern['type']  
 lookback = pattern.get('lookback', 20)  
 criteria = pattern.get('criteria', '')  
  
 historical = await self.redis\_client.get(f"{symbol}:ohlcv:{self.timeframe}")  
 if not historical or len(historical) < lookback:  
 logger.debug(f"Insufficient historical data for pattern {pattern\_name} on {symbol}")  
 self.pattern\_cache.setdefault(symbol, {})[pattern\_name] = False  
 continue  
  
 df = pd.DataFrame(historical[-lookback:])  
 if pattern\_type == 'smc':  
 self.pattern\_cache.setdefault(symbol, {})[pattern\_name] = self.\_evaluate\_smc\_pattern(df, criteria)  
 elif pattern\_type == 'price\_action':  
 self.pattern\_cache.setdefault(symbol, {})[pattern\_name] = self.\_evaluate\_price\_action\_pattern(df, criteria)  
 elif pattern\_type == 'harmonic':  
 self.pattern\_cache.setdefault(symbol, {})[pattern\_name] = self.\_evaluate\_harmonic\_pattern(df, criteria)  
 elif pattern\_type == 'wave':  
 self.pattern\_cache.setdefault(symbol, {})[pattern\_name] = self.\_evaluate\_wave\_pattern(df, criteria)  
 else:  
 logger.error(f"Unsupported pattern type {pattern\_type} for {pattern\_name}")  
 self.pattern\_cache.setdefault(symbol, {})[pattern\_name] = False  
 except Exception as e:  
 logger.error(f"Error updating patterns for {symbol}: {e}", exc\_info=True)  
 self.pattern\_cache.setdefault(symbol, {})[pattern\_name] = False  
  
 def \_evaluate\_smc\_pattern(self, df: pd.DataFrame, criteria: str) -> bool:  
 """Evaluate SMC patterns (e.g., order blocks)."""  
 try:  
 if 'order\_block' in criteria:  
 prev\_high = df['high'].shift(1).iloc[-1]  
 prev\_low = df['low'].shift(1).iloc[-1]  
 volume = df['volume'].iloc[-1]  
 avg\_volume = df['volume'].rolling(20).mean().iloc[-1]  
 expr = criteria.replace('prev\_high', str(prev\_high)) \  
 .replace('prev\_low', str(prev\_low)) \  
 .replace('volume', str(volume)) \  
 .replace('avg\_volume\_20', str(avg\_volume))  
 return self.interpreter(expr)  
 return False  
 except Exception as e:  
 logger.error(f"Error evaluating SMC pattern {criteria}: {e}", exc\_info=True)  
 return False  
  
 def \_evaluate\_price\_action\_pattern(self, df: pd.DataFrame, criteria: str) -> bool:  
 """Evaluate price action patterns (e.g., bullish engulfing)."""  
 try:  
 if 'bullish\_engulfing' in criteria:  
 curr = df.iloc[-1]  
 prev = df.iloc[-2]  
 return (curr['close'] > curr['open'] and  
 prev['close'] < prev['open'] and  
 curr['close'] > prev['high'] and  
 curr['open'] < prev['low'])  
 return False  
 except Exception as e:  
 logger.error(f"Error evaluating price action pattern {criteria}: {e}", exc\_info=True)  
 return False  
  
 def \_evaluate\_harmonic\_pattern(self, df: pd.DataFrame, criteria: str) -> bool:  
 """Evaluate harmonic patterns (e.g., Gartley)."""  
 try:  
 # Placeholder: Implement harmonic pattern detection  
 if 'gartley' in criteria:  
 logger.warning("Harmonic pattern detection not fully implemented")  
 return False  
 return False  
 except Exception as e:  
 logger.error(f"Error evaluating harmonic pattern {criteria}: {e}", exc\_info=True)  
 return False  
  
 def \_evaluate\_wave\_pattern(self, df: pd.DataFrame, criteria: str) -> bool:  
 """Evaluate wave patterns (e.g., Elliott Wave)."""  
 try:  
 # Placeholder: Implement wave pattern detection  
 if 'elliott\_wave' in criteria:  
 logger.warning("Wave pattern detection not fully implemented")  
 return False  
 return False  
 except Exception as e:  
 logger.error(f"Error evaluating wave pattern {criteria}: {e}", exc\_info=True)  
 return False  
  
 def \_evaluate\_rules(self, rules: List[Dict], ohlcv: Dict, indicators: Dict) -> bool:  
 """Evaluate a list of rules and return True if conditions are met."""  
 try:  
 total\_weight = 0.0  
 required\_weight = sum(rule['weight'] for rule in rules) \* self.threshold  
 for rule in rules:  
 condition = rule['condition']  
 weight = rule['weight']  
 if self.\_evaluate\_condition(condition, ohlcv, indicators):  
 total\_weight += weight  
 return total\_weight >= required\_weight  
 except Exception as e:  
 logger.error(f"Error evaluating rules for {self.name}: {e}", exc\_info=True)  
 return False  
  
 def \_evaluate\_condition(self, condition: str, ohlcv: Dict, indicators: Dict) -> bool:  
 """Evaluate a single condition (e.g., 'close < bb\_lower and rsi < 30')."""  
 try:  
 # Handle pattern-based conditions  
 for pattern in self.patterns:  
 pattern\_name = pattern['name']  
 if f"{pattern\_name} = true" in condition.lower():  
 return self.pattern\_cache.get(ohlcv['tradingsymbol'], {}).get(pattern\_name, False)  
  
 # Handle market parameters (e.g., volume\_threshold)  
 condition = self.\_replace\_market\_params(condition, ohlcv, indicators)  
  
 # Handle complex conditions with 'and'/'or'  
 if ' and ' in condition.lower():  
 conditions = condition.split(' and ')  
 return all(self.\_evaluate\_single\_condition(c.strip(), ohlcv, indicators) for c in conditions)  
 elif ' or ' in condition.lower():  
 conditions = condition.split(' or ')  
 return any(self.\_evaluate\_single\_condition(c.strip(), ohlcv, indicators) for c in conditions)  
 else:  
 return self.\_evaluate\_single\_condition(condition, ohlcv, indicators)  
 except Exception as e:  
 logger.error(f"Error evaluating condition {condition}: {e}", exc\_info=True)  
 return False  
  
 def \_replace\_market\_params(self, condition: str, ohlcv: Dict, indicators: Dict) -> str:  
 """Replace market parameters (e.g., volume\_threshold) with evaluated values."""  
 try:  
 for param, expr in self.market\_params.items():  
 if param in condition:  
 value = self.interpreter(expr, {'avg\_volume\_20': indicators.get('avg\_volume\_20', 0.0)})  
 condition = condition.replace(param, str(value))  
 return condition  
 except Exception as e:  
 logger.error(f"Error replacing market params in {condition}: {e}", exc\_info=True)  
 return condition  
  
 def \_evaluate\_single\_condition(self, condition: str, ohlcv: Dict, indicators: Dict) -> bool:  
 """Evaluate a single condition (e.g., 'close < bb\_lower')."""  
 try:  
 left, op, right = self.\_parse\_condition(condition)  
 left\_value = ohlcv.get(left, indicators.get(left, 0.0))  
 right\_value = indicators.get(right, ohlcv.get(right, float(right)))  
 operators = {  
 '>': operator.gt,  
 '<': operator.lt,  
 '=': operator.eq,  
 '>=': operator.ge,  
 '<=': operator.le,  
 '!=': operator.ne  
 }  
 if op not in operators:  
 logger.error(f"Unsupported operator in condition: {condition}")  
 return False  
 return operators[op](float(left\_value), float(right\_value))  
 except Exception as e:  
 logger.error(f"Error evaluating single condition {condition}: {e}", exc\_info=True)  
 return False  
  
 def \_parse\_condition(self, condition: str) -> tuple:  
 """Parse a condition string into left, operator, right."""  
 for op in ['>=', '<=', '!=', '>', '<', '=']:  
 if op in condition:  
 left, right = condition.split(op)  
 return left.strip(), op, right.strip()  
 raise ValueError(f"Invalid condition format: {condition}")  
  
 async def \_evaluate\_stop\_loss(self, symbol: str, close: float, position: Dict) -> Optional[str]:  
 """Evaluate stop-loss conditions."""  
 try:  
 if not self.stop\_loss:  
 return None  
 entry\_price = position['entry\_price']  
 if self.stop\_loss['type'] == 'multi':  
 for rule in self.stop\_loss.get('rules', []):  
 if rule['type'] == 'fixed':  
 sl\_value = float(rule['value'].strip('%')) / 100  
 sl\_price = entry\_price \* (1 - sl\_value)  
 if close <= sl\_price:  
 logger.info(f"Stop-loss triggered for {symbol}: close={close}, sl\_price={sl\_price}")  
 return f"SELL:{rule.get('id', 'fixed')}"  
 elif rule['type'] == 'trailing':  
 sl\_value = float(rule['value'].strip('%')) / 100  
 position['highest\_price'] = max(position['highest\_price'], close)  
 sl\_price = position['highest\_price'] \* (1 - sl\_value)  
 if close <= sl\_price:  
 logger.info(f"Trailing stop-loss triggered for {symbol}: close={close}, sl\_price={sl\_price}")  
 return f"SELL:{rule.get('id', 'trailing')}"  
 else:  
 if self.stop\_loss['type'] == 'fixed':  
 sl\_value = float(self.stop\_loss['value'].strip('%')) / 100  
 sl\_price = entry\_price \* (1 - sl\_value)  
 if close <= sl\_price:  
 logger.info(f"Stop-loss triggered for {symbol}: close={close}, sl\_price={sl\_price}")  
 return "SELL"  
 elif self.stop\_loss['type'] == 'trailing':  
 sl\_value = float(self.stop\_loss['value'].strip('%')) / 100  
 position['highest\_price'] = max(position['highest\_price'], close)  
 sl\_price = position['highest\_price'] \* (1 - sl\_value)  
 if close <= sl\_price:  
 logger.info(f"Trailing stop-loss triggered for {symbol}: close={close}, sl\_price={sl\_price}")  
 return "SELL"  
 return None  
 except Exception as e:  
 logger.error(f"Error evaluating stop-loss for {symbol}: {e}", exc\_info=True)  
 return None  
  
 async def \_evaluate\_target(self, symbol: str, close: float, position: Dict) -> Optional[str]:  
 """Evaluate target conditions."""  
 try:  
 if not self.target:  
 return None  
 entry\_price = position['entry\_price']  
 if self.target['type'] == 'multi':  
 for rule in self.target.get('rules', []):  
 if rule['type'] == 'fixed':  
 target\_value = float(rule['value'].strip('%')) / 100  
 target\_price = entry\_price \* (1 + target\_value)  
 if close >= target\_price:  
 logger.info(f"Target triggered for {symbol}: close={close}, target\_price={target\_price}")  
 if rule.get('partial\_exit'):  
 partial\_exit = rule['partial\_exit'].strip('%')  
 return f"PARTIAL\_SELL:{partial\_exit}:{rule.get('id', 'fixed')}"  
 return f"SELL:{rule.get('id', 'fixed')}"  
 elif rule['type'] == 'trailing':  
 target\_value = float(rule['value'].strip('%')) / 100  
 position['lowest\_price'] = min(position['lowest\_price'], close)  
 target\_price = position['lowest\_price'] \* (1 + target\_value)  
 if close >= target\_price:  
 logger.info(f"Trailing target triggered for {symbol}: close={close}, target\_price={target\_price}")  
 if rule.get('partial\_exit'):  
 partial\_exit = rule['partial\_exit'].strip('%')  
 return f"PARTIAL\_SELL:{partial\_exit}:{rule.get('id', 'trailing')}"  
 return f"SELL:{rule.get('id', 'trailing')}"  
 else:  
 if self.target['type'] == 'fixed':  
 target\_value = float(self.target['value'].strip('%')) / 100  
 target\_price = entry\_price \* (1 + target\_value)  
 if close >= target\_price:  
 logger.info(f"Target triggered for {symbol}: close={close}, target\_price={target\_price}")  
 return "SELL"  
 elif self.target['type'] == 'trailing':  
 target\_value = float(self.target['value'].strip('%')) / 100  
 position['lowest\_price'] = min(position['lowest\_price'], close)  
 target\_price = position['lowest\_price'] \* (1 + target\_value)  
 if close >= target\_price:  
 logger.info(f"Trailing target triggered for {symbol}: close={close}, target\_price={target\_price}")  
 return "SELL"  
 return None  
 except Exception as e:  
 logger.error(f"Error evaluating target for {symbol}: {e}", exc\_info=True)  
 return None  
  
 def \_evaluate\_breakeven(self, symbol: str, close: float, position: Dict) -> bool:  
 """Evaluate breakeven trigger."""  
 try:  
 if not self.trade\_management or 'breakeven' not in self.trade\_management:  
 return False  
 if position['breakeven\_triggered']:  
 return False  
 trigger = self.trade\_management['breakeven']['trigger'] / 100  
 breakeven\_price = position['entry\_price'] \* (1 + trigger)  
 if close >= breakeven\_price:  
 logger.info(f"Breakeven triggered for {symbol}: close={close}, breakeven\_price={breakeven\_price}")  
 return True  
 return False  
 except Exception as e:  
 logger.error(f"Error evaluating breakeven for {symbol}: {e}", exc\_info=True)  
 return False  
  
 async def \_log\_audit\_trail(self, symbol: str, ohlcv: Dict, indicators: Dict, signal: str, reason: str):  
 """Log audit trail for signal generation."""  
 try:  
 audit\_data = {  
 'symbol': symbol,  
 'strategy': self.name,  
 'signal': signal,  
 'timestamp': ohlcv['timestamp'],  
 'price': ohlcv['close'],  
 'reason': reason,  
 'ohlcv': ohlcv,  
 'indicators': indicators  
 }  
 await self.redis\_client.publish(f"signals:audit:{self.name}", json.dumps(audit\_data))  
 await self.database.save\_audit\_trail(audit\_data)  
 logger.debug(f"Logged audit trail for {symbol}: {signal} ({reason})")  
 except Exception as e:  
 logger.error(f"Error logging audit trail for {symbol}: {e}", exc\_info=True)  
  
 async def stop(self):  
 """Close RuleEngine connections."""  
 logger.info(f"Closing RuleEngine for {self.name}")  
 try:  
 await self.redis\_client.close()  
 await self.database.close()  
 logger.info(f"RuleEngine for {self.name} closed successfully")  
 except Exception as e:  
 logger.error(f"Error closing RuleEngine for {self.name}: {e}", exc\_info=True)  
 raise

## src\anquant\py\core\flexirule\strategy\_engine.py

# src/anquant/py/core/flexirule/strategy\_engine.py  
import asyncio  
import json  
from typing import Dict, Any, List  
import os  
import yaml  
from src.anquant.py.util.logging import setup\_logging  
from src.anquant.py.messaging.kafka\_client import KafkaClient  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.core.flexirule.validator import StrategyValidator  
from src.anquant.py.core.flexirule.rule\_engine import RuleEngine  
  
logger = setup\_logging("strategy\_manager", log\_type="strategy")  
  
class StrategyManager:  
 def \_\_init\_\_(self, config: Dict[str, Any], redis\_client: RedisClient):  
 self.config = config  
 self.redis\_client = redis\_client  
 self.kafka\_client = KafkaClient(config["global"]["kafka"]) if not config['global'].get('offline\_mode', False) else None  
 self.validator = StrategyValidator(config)  
 self.strategies = self.\_load\_strategy\_configs()  
 self.rule\_engines = {strategy['name']: RuleEngine(strategy, redis\_client) for strategy in self.strategies}  
 logger.debug(f"Initialized StrategyManager with {len(self.strategies)} strategies")  
  
 def \_load\_strategy\_configs(self) -> List[Dict[str, Any]]:  
 """Load all strategy YAML files from config/markets/india/strategies/."""  
 strategies = []  
 strategy\_dir = os.path.join(os.path.dirname(\_\_file\_\_), "../../../../config/markets/india/strategies")  
 try:  
 for strategy\_file in os.listdir(strategy\_dir):  
 if strategy\_file.endswith(('.yaml', '.yml')):  
 strategy\_path = os.path.join(strategy\_dir, strategy\_file)  
 if self.validator.validate\_strategy(strategy\_path):  
 with open(strategy\_path, 'r', encoding='utf-8') as f:  
 strategy\_config = yaml.safe\_load(f)  
 strategies.append(strategy\_config)  
 scheduler.debug(f"Loaded strategy config: {strategy\_file}")  
 else:  
 logger.warning(f"Skipping invalid strategy config: {strategy\_file}")  
 except Exception as e:  
 logger.error(f"Failed to load strategy configs from {strategy\_dir}: {e}", exc\_info=True)  
 raise  
 return strategies  
  
 async def initialize(self):  
 """Initialize Kafka and Redis connections, and RuleEngines."""  
 logger.info("Initializing StrategyManager")  
 try:  
 if self.kafka\_client:  
 self.kafka\_client.connect()  
 topics = list(set(f"ohlcv\_{strategy['timeframe']}" for strategy in self.strategies))  
 self.kafka\_client.subscribe(topics)  
 logger.debug(f"Subscribed to Kafka topics: {topics}")  
 await self.redis\_client.connect()  
 await self.redis\_client.redis.ping()  
 logger.debug("Verified Redis connectivity")  
 for name, rule\_engine in self.rule\_engines.items():  
 await rule\_engine.initialize()  
 logger.debug(f"Initialized RuleEngine for strategy {name}")  
 logger.info("StrategyManager initialized successfully")  
 except Exception as e:  
 logger.error(f"Failed to initialize StrategyManager: {e}", exc\_info=True)  
 raise  
  
 async def start(self):  
 """Start processing OHLCV messages."""  
 logger.info("Starting StrategyManager")  
 try:  
 if not self.config['global'].get('offline\_mode', False):  
 asyncio.create\_task(self.\_process\_messages())  
 logger.info("StrategyManager started successfully")  
 except Exception as e:  
 logger.error(f"Failed to start StrategyManager: {e}", exc\_info=True)  
 raise  
  
 async def \_process\_messages(self):  
 """Process OHLCV messages from Kafka and evaluate strategies."""  
 try:  
 while True:  
 msg = self.kafka\_client.poll(timeout=1.0)  
 if msg:  
 ohlcv = msg['value']  
 symbol = ohlcv['tradingsymbol']  
 topic = msg['topic']  
 timeframe = topic.replace('ohlcv\_', '')  
 logger.debug(f"Processing OHLCV for {symbol} on {timeframe} timeframe")  
 await self.\_process\_strategy\_signals(symbol, timeframe, ohlcv)  
 await asyncio.sleep(0.1)  
 except Exception as e:  
 logger.error(f"Error in strategy message processing loop: {e}", exc\_info=True)  
 raise  
  
 async def \_process\_strategy\_signals(self, symbol: str, timeframe: str, ohlcv: Dict):  
 """Evaluate strategies for a given symbol and timeframe."""  
 try:  
 indicators\_data = await self.redis\_client.get(f"{symbol}:indicators:{timeframe}")  
 indicators = indicators\_data if isinstance(indicators\_data, dict) else indicators\_data[0] if indicators\_data else {}  
 for strategy in self.strategies:  
 if strategy['timeframe'] == timeframe:  
 rule\_engine = self.rule\_engines[strategy['name']]  
 signal = await rule\_engine.evaluate(symbol, ohlcv, indicators)  
 if signal and signal != "HOLD":  
 signal\_data = {  
 'symbol': symbol,  
 'signal': signal,  
 'timestamp': ohlcv['timestamp'],  
 'price': ohlcv['close'],  
 'strategy': strategy['name'],  
 'market': ohlcv.get('market', 'india')  
 }  
 await self.redis\_client.publish(f"signals:{strategy['name']}", json.dumps(signal\_data))  
 if self.kafka\_client:  
 self.kafka\_client.produce("signals", key=symbol, value=signal\_data)  
 logger.info(f"Generated signal for {symbol}: {signal} (strategy: {strategy['name']})")  
 except Exception as e:  
 logger.error(f"Error processing strategy signals for {symbol}:{timeframe}: {e}", exc\_info=True)  
  
 async def close(self):  
 """Close StrategyManager connections."""  
 logger.info("Closing StrategyManager")  
 try:  
 if self.kafka\_client:  
 self.kafka\_client.close()  
 await self.redis\_client.close()  
 for name, rule\_engine in self.rule\_engines.items():  
 await rule\_engine.close()  
 logger.info("StrategyManager closed successfully")  
 except Exception as e:  
 logger.error(f"Error closing StrategyManager: {e}", exc\_info=True)  
 raise

## src\anquant\py\core\flexirule\validator.py

# D:\AlphaNivesh\ANQuant\src\anquant\py\core\flexirule\validator.py  
from pydantic import BaseModel, ValidationError, Field  
from typing import List, Dict, Any, Optional  
from src.anquant.py.util.logging import setup\_logging  
from src.anquant.py.util.config\_loader import load\_config  
import yaml  
import os  
  
logger = setup\_logging("flexirule\_validator", log\_type="strategy")  
  
  
class IndicatorConfig(BaseModel):  
 name: str  
 type: str  
 period: Optional[int] = None  
 std: Optional[float] = None  
 fast: Optional[int] = None  
 slow: Optional[int] = None  
 signal: Optional[int] = None  
  
  
class PatternConfig(BaseModel):  
 name: str  
 type: str  
 lookback: Optional[int] = 20  
 criteria: Optional[str] = ""  
  
  
class RuleConfig(BaseModel):  
 condition: str  
 weight: float = Field(ge=0.0, le=1.0)  
  
  
class StopLossRule(BaseModel):  
 type: str = Field(pattern=r"^(fixed|trailing)$")  
 value: str = Field(pattern=r"^[0-9]+%?$")  
 partial\_exit: Optional[str] = None  
 id: Optional[str] = None  
  
  
class StopLossConfig(BaseModel):  
 type: str = Field(pattern=r"^(fixed|trailing|multi)$")  
 rules: Optional[List[StopLossRule]] = None  
  
  
class TargetRule(BaseModel):  
 type: str = Field(pattern=r"^(fixed|trailing)$")  
 value: str = Field(pattern=r"^[0-9]+%?$")  
 partial\_exit: Optional[str] = None  
 id: Optional[str] = None  
  
  
class TargetConfig(BaseModel):  
 type: str = Field(pattern=r"^(fixed|trailing|multi)$")  
 rules: Optional[List[TargetRule]] = None  
  
  
class TradeManagementConfig(BaseModel):  
 breakeven: Optional[Dict[str, float]] = None  
  
  
class MarketParamsConfig(BaseModel):  
 india: Optional[Dict[str, str]] = None  
  
  
class StrategyConfig(BaseModel):  
 name: str  
 timeframe: str = Field(pattern=r"^(1min|5min|15min|30min|1hr)$")  
 watchlist: Dict[str, str]  
 threshold: Optional[float] = 0.75  
 indicators: Optional[List[IndicatorConfig]] = []  
 patterns: Optional[List[PatternConfig]] = []  
 entry\_rules: List[RuleConfig]  
 exit\_rules: List[RuleConfig]  
 stop\_loss: Optional[StopLossConfig] = None  
 target: Optional[TargetConfig] = None  
 trade\_management: Optional[TradeManagementConfig] = None  
 market\_params: Optional[MarketParamsConfig] = None  
  
  
class StrategyValidator:  
 def \_\_init\_\_(self, config: Dict[str, Any]):  
 self.config = config  
 self.supported\_markets = list(config["global"]["markets"].keys())  
 self.supported\_timeframes = config["global"]["historical\_data"]["timeframes"]  
 logger.info("Initialized StrategyValidator with supported markets: %s, timeframes: %s",  
 self.supported\_markets, self.supported\_timeframes)  
  
 def validate\_strategy(self, strategy\_path: str) -> bool:  
 """Validate a strategy YAML file."""  
 try:  
 with open(strategy\_path, 'r', encoding='utf-8') as f:  
 strategy\_data = yaml.safe\_load(f)  
  
 # Validate using Pydantic  
 strategy\_config = StrategyConfig(\*\*strategy\_data)  
  
 # Check watchlist existence  
 for market, watchlist\_path in strategy\_config.watchlist.items():  
 if market not in self.supported\_markets:  
 logger.error(f"Unsupported market {market} in strategy {strategy\_config.name}")  
 return False  
 if not os.path.exists(watchlist\_path):  
 logger.error(f"Watchlist file not found: {watchlist\_path}")  
 return False  
  
 # Validate timeframe  
 if strategy\_config.timeframe not in self.supported\_timeframes:  
 logger.error(f"Invalid timeframe {strategy\_config.timeframe} in strategy {strategy\_config.name}")  
 return False  
  
 # Validate indicator types  
 supported\_indicators = ["bollinger\_bands", "rsi", "sma", "macd"]  
 for indicator in strategy\_config.indicators:  
 if indicator.type not in supported\_indicators:  
 logger.error(f"Unsupported indicator type {indicator.type} in strategy {strategy\_config.name}")  
 return False  
  
 # Validate pattern types  
 supported\_patterns = ["smc", "price\_action", "harmonic", "wave"]  
 for pattern in strategy\_config.patterns:  
 if pattern.type not in supported\_patterns:  
 logger.error(f"Unsupported pattern type {pattern.type} in strategy {strategy\_config.name}")  
 return False  
  
 # Validate stop-loss and target rules  
 if strategy\_config.stop\_loss and strategy\_config.stop\_loss.type == "multi" and not strategy\_config.stop\_loss.rules:  
 logger.error(f"Multi stop-loss requires rules in strategy {strategy\_config.name}")  
 return False  
 if strategy\_config.target and strategy\_config.target.type == "multi" and not strategy\_config.target.rules:  
 logger.error(f"Multi target requires rules in strategy {strategy\_config.name}")  
 return False  
  
 logger.info(f"Successfully validated strategy: {strategy\_config.name}")  
 return True  
 except ValidationError as e:  
 logger.error(f"Validation error for strategy {strategy\_path}: {e}")  
 return False  
 except Exception as e:  
 logger.error(f"Error validating strategy {strategy\_path}: {e}", exc\_info=True)  
 return False

## src\anquant\py\core\flexirule\\_\_init\_\_.py

## src\anquant\py\core\order\_management\order\_management\_engine.py

## src\anquant\py\core\order\_management\\_\_init\_\_.py

## src\anquant\py\core\risk\_management\risk\_management.py

# src/py/core/risk\_management.py  
from typing import Dict, Any  
from src.anquant.py.util.logging import setup\_logging  
  
logger = setup\_logging("portfolio\_manager", log\_type="general")  
  
class RiskManagementEngine:  
 def \_\_init\_\_(self, config: Dict[str, Any]):  
 self.config = config  
 self.risk\_parameters = config.get("global", {}).get("risk\_parameters", {  
 "max\_position\_size": 1000000,  
 "max\_portfolio\_exposure": 0.5,  
 "max\_daily\_loss": 0.05  
 })  
 logger.debug("Initialized RiskManagementEngine with parameters: {}".format(self.risk\_parameters))  
  
 async def initialize(self):  
 logger.debug("RiskManagementEngine initializing")  
 if not self.risk\_parameters:  
 logger.error("No risk parameters defined in config")  
 raise ValueError("Risk parameters missing")  
  
 async def start(self):  
 logger.debug("RiskManagementEngine starting")  
 # Risk management runs passively, no active tasks needed  
 return None  
  
 async def stop(self):  
 logger.debug("RiskManagementEngine stopping")  
 # No cleanup needed for passive risk management

## src\anquant\py\core\risk\_management\\_\_init\_\_.py

## src\anquant\py\core\strategy\strategy\_engine.py

# src/py/core/strategy/strategy\_engine.py  
import asyncio  
from typing import Dict, Any  
from src.anquant.py.messaging.kafka\_client import KafkaClient  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.util.logging import setup\_logging  
import json  
  
logger = setup\_logging("strategy", log\_type="strategy")  
  
  
class StrategyEngine:  
 def \_\_init\_\_(self, config: Dict[str, Any], redis\_client: RedisClient):  
 self.config = config  
 self.redis\_client = redis\_client  
 self.kafka\_client = KafkaClient(config["global"]["kafka"])  
 self.strategies = config["global"]["strategies"]  
 logger.debug(f"Initialized StrategyEngine with {len(self.strategies)} strategies")  
  
 async def initialize(self):  
 logger.debug("StrategyEngine initializing")  
 try:  
 topics = [f"ohlcv\_{strategy['timeframe']}" for strategy in self.strategies]  
 self.kafka\_client.subscribe(topics)  
 logger.info(f"StrategyEngine subscribed to Kafka topics: {topics}")  
 await self.redis\_client.redis.ping()  
 logger.debug("Verified Redis connectivity for StrategyEngine")  
 except Exception as e:  
 logger.error(f"Failed to initialize StrategyEngine: {e}")  
 raise  
  
 async def start(self):  
 logger.debug("StrategyEngine starting")  
 # Start the message processing loop as a background task  
 asyncio.create\_task(self.\_process\_messages())  
 logger.info("StrategyEngine started")  
  
 async def \_process\_messages(self):  
 """Background task to process OHLCV messages and generate strategy signals."""  
 try:  
 while True:  
 msg = self.kafka\_client.poll(timeout=1.0)  
 if msg:  
 ohlcv = msg['value'] # Already deserialized by KafkaClient  
 symbol = ohlcv['tradingsymbol']  
 topic = msg['topic']  
  
 # Extract timeframe from topic (e.g., "ohlcv\_1min" -> "1min")  
 timeframe = topic.replace('ohlcv\_', '')  
  
 logger.debug(f"Processing OHLCV for {symbol} on {timeframe} timeframe")  
  
 # Process strategy signals  
 await self.\_process\_strategy\_signals(symbol, timeframe, ohlcv)  
 else:  
 # No message received, continue polling  
 await asyncio.sleep(0.1) # Small delay to prevent busy waiting  
 except Exception as e:  
 logger.error(f"Error in strategy message processing loop: {e}", exc\_info=True)  
 raise  
  
 async def \_process\_strategy\_signals(self, symbol: str, timeframe: str, ohlcv: Dict):  
 """Process strategy signals for a given symbol and timeframe."""  
 try:  
 # Get indicators from Redis  
 indicators\_data = await self.redis\_client.get(f"{symbol}:indicators:{timeframe}")  
 if not indicators\_data:  
 logger.debug(f"No indicators available for {symbol}:{timeframe}")  
 return  
  
 # Convert to dict if it's a list  
 indicators = indicators\_data if isinstance(indicators\_data, dict) else indicators\_data[0] if indicators\_data else {}  
  
 # Apply strategy logic  
 for strategy in self.strategies:  
 if strategy['timeframe'] == timeframe:  
 signal = await self.\_apply\_strategy(strategy, symbol, ohlcv, indicators)  
 if signal:  
 logger.info(f"Strategy signal for {symbol}: {signal}")  
 # Store signal in Redis for order execution  
 signal\_data = {  
 'symbol': symbol,  
 'signal': signal,  
 'timestamp': ohlcv['timestamp'],  
 'price': ohlcv['close']  
 }  
 await self.redis\_client.publish(f"signals:{strategy['name']}", json.dumps(signal\_data))  
 except Exception as e:  
 logger.error(f"Error processing strategy signals for {symbol}:{timeframe}: {e}", exc\_info=True)  
  
 async def \_apply\_strategy(self, strategy: Dict, symbol: str, ohlcv: Dict, indicators: Dict) -> str:  
 """Apply a specific strategy and return signal (BUY/SELL/HOLD)."""  
 try:  
 # Simple example strategy using Bollinger Bands  
 if 'bb\_upper' in indicators and 'bb\_lower' in indicators:  
 close = ohlcv['close']  
 bb\_upper = indicators['bb\_upper']  
 bb\_lower = indicators['bb\_lower']  
  
 if close <= bb\_lower:  
 return "BUY"  
 elif close >= bb\_upper:  
 return "SELL"  
 else:  
 return "HOLD"  
  
 return "HOLD" # Default to HOLD if no indicators available  
 except Exception as e:  
 logger.error(f"Error applying strategy {strategy.get('name', 'unknown')}: {e}", exc\_info=True)  
 return "HOLD"  
  
 async def stop(self):  
 logger.debug("StrategyEngine stopping")  
 self.kafka\_client.close()

## src\anquant\py\core\strategy\\_\_init\_\_.py

## src\anquant\py\corporate\_actions\corporate\_actions\_manager.py

# src/anquant/py/core/corporate\_actions\_manager.py  
import asyncio  
import pandas as pd  
from typing import Dict, Any, List  
from src.anquant.py.messaging.redis\_client import RedisClient  
from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from selenium.webdriver.chrome.options import Options  
from selenium.webdriver.common.by import By  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver.support import expected\_conditions as EC  
from webdriver\_manager.chrome import ChromeDriverManager  
from io import StringIO  
import json  
from datetime import datetime  
from src.anquant.py.util.logging import setup\_logging  
  
logger = setup\_logging("corporate\_action\_manager", log\_type="general")  
  
  
class CorporateActionManager:  
 def \_\_init\_\_(self, config: Dict[str, Any], redis\_client: RedisClient):  
 self.config = config  
 self.redis\_client = redis\_client  
 logger.debug("Initialized CorporateActionManager")  
  
 def setup\_driver(self):  
 options = Options()  
 options.add\_argument("--headless")  
 options.add\_argument("--disable-blink-features=AutomationControlled")  
 options.add\_argument("--no-sandbox")  
 options.add\_argument("--disable-dev-shm-usage")  
 options.add\_argument("window-size=1920,1080")  
 options.add\_argument(  
 "user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 Chrome/117.0.0.0 Safari/537.36")  
 options.add\_experimental\_option("excludeSwitches", ["enable-automation"])  
 options.add\_experimental\_option("useAutomationExtension", False)  
 driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)  
 driver.execute\_cdp\_cmd("Page.addScriptToEvaluateOnNewDocument", {  
 "source": """  
 Object.defineProperty(navigator, 'webdriver', {  
 get: () => undefined  
 });  
 """  
 })  
 return driver  
  
 async def initialize(self):  
 logger.debug("CorporateActionManager initializing")  
 try:  
 await self.redis\_client.redis.ping()  
 logger.debug("Verified Redis connectivity for CorporateActionManager")  
 # Pre-fetch corporate actions during initialization (optional)  
 try:  
 await self.fetch\_and\_cache\_all\_actions()  
 except Exception as e:  
 logger.warning(f"Failed to fetch corporate actions during initialization: {e}")  
 logger.info("CorporateActionManager will continue without pre-fetched data")  
 except Exception as e:  
 logger.error(f"Failed to initialize CorporateActionManager: {e}")  
 raise  
  
 async def start(self):  
 logger.debug("CorporateActionManager starting") # Corporate actions management runs passively, no active tasks needed  
 return None  
  
 async def stop(self):  
 logger.debug("CorporateActionManager stopping")  
 # No cleanup needed for passive corporate actions management  
  
 async def fetch\_and\_cache\_all\_actions(self):  
 """Fetch corporate actions from NSE and cache in Redis."""  
 logger.info("Fetching corporate actions from NSE")  
 url = "https://www.nseindia.com/companies-listing/corporate-filings-actions"  
 driver = self.setup\_driver()  
 try:  
 driver.get(url)  
 WebDriverWait(driver, 15).until(EC.presence\_of\_element\_located((By.TAG\_NAME, "body")))  
 driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")  
 await asyncio.sleep(5) # Allow async context for page load  
 html = driver.page\_source  
 raw\_tables = pd.read\_html(StringIO(html), flavor="html5lib")  
  
 target\_df = None  
 for t in raw\_tables:  
 lower\_cols = [str(col).strip().lower() for col in t.columns]  
 if 'symbol' in lower\_cols and 'purpose' in lower\_cols:  
 target\_df = t  
 break  
  
 if target\_df is None:  
 raise Exception("No corporate actions table found")  
  
 df = target\_df.copy()  
 df.columns = [str(col).strip().replace(" ", "\_").lower() for col in df.columns]  
 df = df.rename(columns={  
 'symbol': 'tradingsymbol',  
 'purpose': 'type',  
 'ex-date': 'ex\_date',  
 'record\_date': 'record\_date'  
 })  
  
 # Cache actions per symbol in Redis  
 for \_, row in df.iterrows():  
 symbol = row['tradingsymbol']  
 if not isinstance(symbol, str):  
 continue  
 symbol\_clean = symbol.replace("-EQ", "")  
 action = {  
 "type": row['type'],  
 "ex\_date": row['ex\_date'],  
 "record\_date": row.get('record\_date', ''),  
 "amount": float(row.get('dividend\_amount', 0)) if 'dividend' in row['type'].lower() else 0,  
 "ratio": row.get('ratio', '') if 'bonus' in row['type'].lower() or 'rights' in row[  
 'type'].lower() else ''  
 }  
 cache\_key = f"{symbol\_clean}:corporate\_actions"  
 await self.redis\_client.cache(cache\_key, [action], ttl=86400)  
 logger.debug(f"Cached corporate actions for {symbol\_clean}")  
  
 logger.info("Completed caching corporate actions")  
 except Exception as e:  
 logger.error(f"Error fetching corporate actions: {e}")  
 raise  
 finally:  
 driver.quit()  
  
 async def fetch\_actions(self, symbol: str) -> List[Dict]:  
 """Retrieve cached corporate actions from Redis."""  
 symbol\_clean = symbol.replace("-EQ", "")  
 cache\_key = f"{symbol\_clean}:corporate\_actions"  
 try:  
 actions = await self.redis\_client.get(cache\_key)  
 if actions:  
 logger.debug(f"Retrieved corporate actions for {symbol} from Redis")  
 return actions  
 # Fallback to fetching if not cached  
 await self.fetch\_and\_cache\_all\_actions()  
 actions = await self.redis\_client.get(cache\_key)  
 return actions if actions else []  
 except Exception as e:  
 logger.error(f"Failed to fetch corporate actions for {symbol}: {e}")  
 return []  
  
 async def adjust\_ohlcv(self, symbol: str, timeframe: str, ohlcv\_df: pd.DataFrame,  
 actions: List[Dict]) -> pd.DataFrame:  
 """Adjust OHLCV data for corporate actions."""  
 df = ohlcv\_df.copy()  
 df["timestamp"] = pd.to\_datetime(df["timestamp"])  
 for action in actions:  
 ex\_date = pd.to\_datetime(action["ex\_date"])  
 if action["type"].lower() == "dividend" and ex\_date <= pd.Timestamp("2025-07-15"):  
 df.loc[df["timestamp"] < ex\_date, ["open", "high", "low", "close"]] -= action["amount"]  
 logger.debug(f"Adjusted {symbol} ({timeframe}) for dividend: {action['amount']}")  
 elif action["type"].lower() == "rights":  
 ratio = eval(action["ratio"].replace(":", "/"))  
 df.loc[df["timestamp"] < ex\_date, ["open", "high", "low", "close"]] \*= (1 / (1 + ratio))  
 elif action["type"].lower() == "bonus":  
 ratio = eval(action["ratio"].replace(":", "/"))  
 df.loc[df["timestamp"] < ex\_date, ["open", "high", "low", "close"]] /= (1 + ratio)  
 df.loc[df["timestamp"] < ex\_date, "volume"] \*= (1 + ratio)  
 return df

## src\anquant\py\corporate\_actions\\_\_init\_\_.py

## src\anquant\py\data\_management\\_\_init\_\_.py

## src\anquant\py\data\_management\historical\historical\_data\_manager.py

# src/anquant/py/core/historical\_data\_manager.py  
from typing import Dict, Any  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.util.database import Database  
from anquant.py.corporate\_actions import CorporateActionManager  
from src.anquant.py.util.logging import setup\_logging  
  
logger = setup\_logging("historical\_data\_manager", log\_type="general")  
  
class HistoricalDataManager:  
 def \_\_init\_\_(self, config: Dict[str, Any], redis\_client: RedisClient, database: Database):  
 self.config = config  
 self.redis\_client = redis\_client  
 self.database = database  
 self.corporate\_action\_manager = CorporateActionManager(config, redis\_client)  
 logger.debug("Initialized HistoricalDataManager")  
  
 async def initialize(self):  
 logger.debug("HistoricalDataManager initializing")  
 cursor = self.database.conn.cursor()  
 try:  
 cursor.execute("""  
 CREATE TABLE IF NOT EXISTS anquant.ohlcv (  
 timestamp TIMESTAMP,  
 open FLOAT,  
 high FLOAT,  
 low FLOAT,  
 close FLOAT,  
 volume BIGINT,  
 tradingsymbol VARCHAR(50),  
 exchange VARCHAR(10),  
 timeframe VARCHAR(10),  
 PRIMARY KEY (timestamp, tradingsymbol, timeframe)  
 )  
 """)  
 self.database.conn.commit()  
 logger.debug("Ensured anquant.ohlcv table exists")  
 except Exception as e:  
 self.database.conn.rollback()  
 logger.error(f"Failed to initialize anquant.ohlcv table: {e}")  
 raise  
 finally:  
 cursor.close()  
  
 async def start(self):  
 logger.debug("HistoricalDataManager starting") # Historical data management runs passively, no active tasks needed  
 return None  
  
 async def stop(self):  
 logger.debug("HistoricalDataManager stopping")  
 # No cleanup needed for passive historical data management

## src\anquant\py\data\_management\historical\\_\_init\_\_.py

## src\anquant\py\data\_management\market\_data\market\_data\_engine.py

# src/anquant/py/core/market\_data/market\_data\_engine.py  
import asyncio  
import pandas as pd  
import json  
from typing import Dict, Any, Set  
from datetime import datetime, time  
import pytz  
import asyncpg  
from src.anquant.py.util.logging import setup\_logging  
from src.anquant.py.messaging.kafka\_client import KafkaClient  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.util.config\_loader import load\_config, load\_credentials, load\_symbol\_mappings  
  
logger = setup\_logging("market\_data\_engine", log\_type="market\_data")  
  
class MarketDataEngine:  
 def \_\_init\_\_(self, config: Dict[str, Any], redis\_client: RedisClient, adapters: Dict[str, Dict[str, Any]]):  
 self.config = config  
 self.redis\_client = redis\_client  
 self.adapters = adapters  
 self.kafka\_client = KafkaClient(config["global"]["kafka"]) if not config['global'].get('offline\_mode', False) else None  
 self.timeframes = config["global"]["historical\_data"]["timeframes"]  
 self.markets = config["global"]["markets"]  
 self.watchlists = {market: {name: path for name, path in market\_config["watchlists"].items()}  
 for market, market\_config in self.markets.items()}  
 self.offline\_mode = config['global'].get('offline\_mode', False)  
 self.db\_pool = None  
 logger.info(f"Initialized MarketDataEngine with timeframes: {self.timeframes}, markets: {list(self.markets.keys())}, adapters: {list(self.adapters.keys())}")  
  
 async def initialize(self):  
 """Initialize Kafka, PostgreSQL, Redis, and adapters."""  
 logger.info("Initializing MarketDataEngine")  
 try:  
 if not self.offline\_mode:  
 self.kafka\_client.subscribe(["nse\_ticks"])  
 logger.debug("Kafka client connected and subscribed to topic: nse\_ticks")  
 self.db\_pool = await asyncpg.create\_pool(  
 host=self.config["global"]["database"]["host"],  
 port=self.config["global"]["database"]["port"],  
 database=self.config["global"]["database"]["dbname"],  
 user=self.config["global"]["database"]["user"],  
 password=self.config["global"]["database"]["password"]  
 )  
 await self.redis\_client.connect()  
 await self.redis\_client.redis.ping()  
 logger.debug("Verified Redis and PostgreSQL connectivity")  
 tasks = []  
 for market in self.adapters:  
 for broker in self.adapters[market]:  
 logger.debug(f"Connecting adapter for {market}/{broker}")  
 tasks.append(self.adapters[market][broker].connect())  
 await asyncio.gather(\*tasks)  
 logger.info("All adapters connected successfully")  
 logger.info("MarketDataEngine initialized successfully")  
 except Exception as e:  
 logger.error(f"Failed to initialize MarketDataEngine: {e}", exc\_info=True)  
 raise  
  
 async def start(self):  
 """Start MarketDataEngine, subscribe to ticks, backfill data, and process ticks."""  
 logger.info("Starting MarketDataEngine")  
 try:  
 if self.offline\_mode:  
 logger.info("Offline mode: Skipping subscriptions and tick processing")  
 return  
 tasks = []  
 for market in self.adapters:  
 for watchlist\_name, watchlist\_path in self.watchlists[market].items():  
 symbols = await self.\_load\_watchlist(watchlist\_path)  
 for broker in self.adapters[market]:  
 logger.debug(f"Subscribing to {market}/{broker} with {len(symbols)} symbols")  
 tasks.append(self.adapters[market][broker].subscribe\_to\_ticks(symbols))  
 await asyncio.gather(\*tasks)  
 await self.\_backfill\_missing\_data()  
 asyncio.create\_task(self.\_process\_ticks())  
 logger.info("MarketDataEngine started successfully")  
 except Exception as e:  
 logger.error(f"Failed to start MarketDataEngine: {e}", exc\_info=True)  
 raise  
  
 async def \_backfill\_missing\_data(self):  
 """Fetch missing OHLCV data for the current session."""  
 try:  
 now = datetime.now(pytz.timezone('Asia/Kolkata'))  
 async with self.db\_pool.acquire() as conn:  
 for market, market\_config in self.markets.items():  
 open\_time = datetime.strptime(market\_config['open\_time'], '%H:%M:%S').time()  
 open\_datetime = datetime.combine(now.date(), open\_time, tzinfo=pytz.timezone('Asia/Kolkata'))  
 if now.time() > open\_time and now.date() == open\_datetime.date():  
 logger.info(f"Backfilling missing data for {market} from {open\_datetime} to {now}")  
 for timeframe in self.timeframes:  
 interval\_map = {'1min': '1T', '5min': '5T', '15min': '15T', '30min': '30T', '1hr': '1H'}  
 interval = interval\_map.get(timeframe, '5T')  
 num\_candles = int((now - open\_datetime).total\_seconds() / pd.Timedelta(interval).total\_seconds())  
 for watchlist\_name, watchlist\_path in market\_config['watchlists'].items():  
 symbols = await self.\_load\_watchlist(watchlist\_path)  
 for symbol in symbols:  
 redis\_key = f"{symbol}:ohlcv:{timeframe}"  
 existing\_data = await self.redis\_client.get(redis\_key)  
 if existing\_data and len(existing\_data) >= num\_candles:  
 logger.debug(f"Skipping backfill for {symbol} ({market}, {timeframe}): sufficient data in Redis")  
 continue  
 ohlcv = await self.\_fetch\_historical\_ohlcv(symbol, timeframe, open\_datetime, now, market)  
 if not ohlcv.empty:  
 await self.redis\_client.cache(redis\_key, ohlcv.to\_dict('records'), ttl=86400)  
 async with conn.transaction():  
 for \_, row in ohlcv.iterrows():  
 await conn.execute(  
 """  
 INSERT INTO anquant.ohlcv (timestamp, open, high, low, close, volume, tradingsymbol, exchange, timeframe, market)  
 VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10)  
 ON CONFLICT (timestamp, tradingsymbol, timeframe) DO UPDATE  
 SET open = EXCLUDED.open, high = EXCLUDED.high, low = EXCLUDED.low,  
 close = EXCLUDED.close, volume = EXCLUDED.volume, exchange = EXCLUDED.exchange,  
 market = EXCLUDED.market  
 """,  
 pd.to\_datetime(row['timestamp']), row['open'], row['high'], row['low'],  
 row['close'], row['volume'], symbol, 'NSE', timeframe, market  
 )  
 for \_, row in ohlcv.iterrows():  
 row\_dict = row.to\_dict()  
 row\_dict['market'] = market  
 if self.kafka\_client:  
 self.kafka\_client.produce(f"ohlcv\_{timeframe}", key=symbol, value=row\_dict)  
 logger.debug(f"Backfilled OHLCV for {symbol} ({market}, {timeframe}): {len(ohlcv)} candles")  
 except Exception as e:  
 logger.error(f"Error backfilling missing data: {e}", exc\_info=True)  
 raise  
  
 async def \_fetch\_historical\_ohlcv(self, symbol: str, timeframe: str, start: datetime, end: datetime, market: str) -> pd.DataFrame:  
 """Fetch historical OHLCV from broker adapters."""  
 logger.debug(f"Fetching historical OHLCV for {symbol} ({timeframe}, {market}) from {start} to {end}")  
 try:  
 for market\_name, adapters in self.adapters.items():  
 if market\_name == market:  
 for broker, adapter in adapters.items():  
 ohlcv = await adapter.fetch\_historical\_ohlcv(symbol, timeframe, start, end)  
 logger.debug(f"Fetched {len(ohlcv)} candles for {symbol} ({timeframe}, {market}) from {broker}")  
 return ohlcv  
 logger.warning(f"No adapter found for market {market}")  
 return pd.DataFrame()  
 except Exception as e:  
 logger.error(f"Error fetching historical OHLCV for {symbol} ({timeframe}, {market}): {e}", exc\_info=True)  
 return pd.DataFrame()  
  
 async def \_process\_ticks(self):  
 """Process real-time ticks and aggregate into OHLCV."""  
 logger.info("Starting tick processing loop")  
 try:  
 while True:  
 msg = self.kafka\_client.poll(timeout=1.0)  
 if msg:  
 tick = msg['value']  
 symbol = tick['tradingsymbol']  
 market = tick.get('market', 'india')  
 timestamp = datetime.fromisoformat(tick['timestamp'].replace('Z', '+00:00'))  
 for timeframe in self.timeframes:  
 ohlcv = await self.\_aggregate\_ohlcv(tick, timeframe)  
 if not ohlcv.empty:  
 ohlcv\_dict = ohlcv.iloc[-1].to\_dict()  
 ohlcv\_dict['market'] = market  
 await self.redis\_client.cache(f"{symbol}:ohlcv:{timeframe}", ohlcv.to\_dict('records'), ttl=86400)  
 async with self.db\_pool.acquire() as conn:  
 async with conn.transaction():  
 await conn.execute(  
 """  
 INSERT INTO anquant.ohlcv (timestamp, open, high, low, close, volume, tradingsymbol, exchange, timeframe, market)  
 VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10)  
 ON CONFLICT (timestamp, tradingsymbol, timeframe) DO UPDATE  
 SET open = EXCLUDED.open, high = EXCLUDED.high, low = EXCLUDED.low,  
 close = EXCLUDED.close, volume = EXCLUDED.volume, exchange = EXCLUDED.exchange,  
 market = EXCLUDED.market  
 """,  
 pd.to\_datetime(ohlcv\_dict['timestamp']), ohlcv\_dict['open'], ohlcv\_dict['high'], ohlcv\_dict['low'],  
 ohlcv\_dict['close'], ohlcv\_dict['volume'], symbol, 'NSE', timeframe, market  
 )  
 if self.kafka\_client:  
 self.kafka\_client.produce(f"ohlcv\_{timeframe}", key=symbol, value=ohlcv\_dict)  
 logger.debug(f"Aggregated and cached OHLCV for {symbol} ({market}, {timeframe}): {json.dumps(ohlcv\_dict)}")  
 else:  
 logger.debug("No tick message received, continuing polling")  
 await asyncio.sleep(0.1)  
 except Exception as e:  
 logger.error(f"Error in tick processing loop: {e}", exc\_info=True)  
 raise  
  
 async def \_aggregate\_ohlcv(self, tick: Dict, timeframe: str) -> pd.DataFrame:  
 """Aggregate ticks into OHLCV for the specified timeframe."""  
 logger.debug(f"Aggregating tick for {tick['tradingsymbol']} ({timeframe})")  
 try:  
 interval\_map = {'1min': '1T', '5min': '5T', '15min': '15T', '30min': '30T', '1hr': '1H'}  
 interval = interval\_map.get(timeframe, '5T')  
 df = pd.DataFrame([{  
 'timestamp': pd.to\_datetime(tick['timestamp']),  
 'open': tick.get('price', tick.get('ltp', 0.0)),  
 'high': tick.get('price', tick.get('ltp', 0.0)),  
 'low': tick.get('price', tick.get('ltp', 0.0)),  
 'close': tick.get('price', tick.get('ltp', 0.0)),  
 'volume': tick.get('volume', 0.0),  
 'tradingsymbol': tick['tradingsymbol']  
 }])  
 df = df.set\_index('timestamp').resample(interval).agg({  
 'open': 'first',  
 'high': 'max',  
 'low': 'min',  
 'close': 'last',  
 'volume': 'sum',  
 'tradingsymbol': 'last'  
 }).dropna().reset\_index()  
 logger.debug(f"Aggregated OHLCV for {tick['tradingsymbol']} ({timeframe}): {df.iloc[-1].to\_dict()}")  
 return df  
 except Exception as e:  
 logger.error(f"Error aggregating OHLCV for {tick['tradingsymbol']} ({timeframe}): {e}", exc\_info=True)  
 return pd.DataFrame()  
  
 async def \_load\_watchlist(self, watchlist\_path: str) -> Set[str]:  
 """Load watchlist symbols from YAML file."""  
 logger.debug(f"Loading watchlist: {watchlist\_path}")  
 try:  
 watchlist\_config = load\_config(watchlist\_path)  
 if not watchlist\_config or 'stocks' not in watchlist\_config:  
 logger.warning(f"Empty or invalid watchlist: {watchlist\_path}")  
 return set()  
 symbols = {item['tradingsymbol'] for item in watchlist\_config['stocks']}  
 logger.debug(f"Loaded watchlist with {len(symbols)} symbols")  
 return symbols  
 except Exception as e:  
 logger.error(f"Failed to load watchlist {watchlist\_path}: {e}", exc\_info=True)  
 return set()  
  
 async def stop(self):  
 """Stop MarketDataEngine, disconnect adapters, and close connections."""  
 logger.info("Stopping MarketDataEngine")  
 try:  
 tasks = []  
 for market in self.adapters:  
 for broker in self.adapters[market]:  
 logger.debug(f"Disconnecting adapter for {market}/{broker}")  
 tasks.append(self.adapters[market][broker].disconnect())  
 await asyncio.gather(\*tasks)  
 if self.kafka\_client:  
 self.kafka\_client.close()  
 if self.db\_pool:  
 await self.db\_pool.close()  
 await self.redis\_client.close()  
 logger.info("MarketDataEngine stopped successfully")  
 except Exception as e:  
 logger.error(f"Error stopping MarketDataEngine: {e}", exc\_info=True)  
 raise

## src\anquant\py\data\_management\market\_data\\_\_init\_\_.py

## src\anquant\py\indicators\indicator\_engine.py

# src/anquant/py/core/indicators/indicator\_engine.py  
import asyncio  
import pandas as pd  
import pandas\_ta as ta  
import json  
import numpy as np  
from typing import Dict, Any, List, Set  
from datetime import datetime, timedelta  
import pytz  
from src.anquant.py.util.logging import setup\_logging  
from src.anquant.py.messaging.kafka\_client import KafkaClient  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.util.config\_loader import load\_config  
from src.anquant.py.util.database import Database  
  
logger = setup\_logging("indicator\_engine", log\_type="indicators")  
  
  
class IndicatorEngine:  
 def \_\_init\_\_(self, config: Dict[str, Any], redis\_client: RedisClient, adapters: Dict[str, Dict[str, Any]]):  
 self.config = config  
 self.redis\_client = redis\_client  
 self.adapters = adapters  
 self.kafka\_client = KafkaClient(config["global"]["kafka"]) if not config['global'].get('offline\_mode',  
 False) else None  
 self.database = Database(config["global"]["database"])  
 self.timeframes = config["global"]["historical\_data"]["timeframes"]  
 self.markets = config["global"]["markets"]  
 self.watchlists = {market: {name: path for name, path in market\_config["watchlists"].items()}  
 for market, market\_config in self.markets.items()}  
 self.offline\_mode = config['global'].get('offline\_mode', False)  
 self.indicators\_config = self.\_load\_indicators\_config()  
 self.ohlcv\_state = {} # symbol: {timeframe: list of dicts}  
 self.indicator\_state = {} # symbol: {timeframe: {'closes': np.array}}  
 logger.info(  
 f"Initialized IndicatorEngine with timeframes: {self.timeframes}, markets: {list(self.markets.keys())}")  
  
 def \_load\_indicators\_config(self) -> List[Dict]:  
 """Load indicators from strategy YAML files."""  
 try:  
 indicators = []  
 for strategy in self.config["global"]["strategies"]:  
 strategy\_file = strategy.get("strategy\_file")  
 if not strategy\_file:  
 logger.warning(f"No strategy\_file defined for strategy {strategy['name']}")  
 continue  
 strategy\_config = load\_config(strategy\_file, require\_global=False)  
 # Support both with and without 'global' section  
 if "global" in strategy\_config:  
 strategy\_indicators = strategy\_config["global"].get("indicators", [])  
 else:  
 strategy\_indicators = strategy\_config.get("indicators", [])  
 indicators.extend(strategy\_indicators)  
 logger.debug(f"Loaded indicators for strategy {strategy['name']}: {json.dumps(strategy\_indicators)}")  
 return indicators  
 except Exception as e:  
 logger.error(f"Failed to load indicators config: {e}", exc\_info=True)  
 raise  
  
 async def initialize(self):  
 """Initialize adapters, preload historical OHLCV, and compute indicators."""  
 logger.debug("Initializing IndicatorEngine")  
 try:  
 # Connect adapters  
 tasks = []  
 for market in self.adapters:  
 for broker in self.adapters[market]:  
 logger.debug(f"Connecting adapter for {market}/{broker}")  
 tasks.append(self.adapters[market][broker].connect())  
 await asyncio.gather(\*tasks)  
 logger.info("All adapters connected successfully")  
  
 # Preload historical data (60 candles per timeframe)  
 lookback = self.config['global']['historical\_data']['lookback\_candles']  
 for market in self.watchlists:  
 for watchlist\_name, watchlist\_path in self.watchlists[market].items():  
 symbols = await self.\_load\_watchlist(watchlist\_path)  
 for symbol in symbols:  
 for timeframe in self.timeframes:  
 logger.debug(f"Preloading historical for {symbol}:{timeframe}")  
 historical = await self.load\_historical(symbol, timeframe, lookback)  
 self.ohlcv\_state.setdefault(symbol, {})[timeframe] = historical  
 closes = np.array([c['close'] for c in historical if 'close' in c])  
 self.indicator\_state.setdefault(symbol, {})[timeframe] = {'closes': closes}  
 indicators = self.\_compute\_indicators(pd.DataFrame(historical), self.indicators\_config)  
 await self.redis\_client.cache(f"{symbol}:indicators:{timeframe}", indicators, ttl=86400)  
 logger.debug(  
 f"Preloaded and cached indicators for {symbol} ({market}, {timeframe}): {json.dumps(indicators)}")  
 logger.info("IndicatorEngine initialized successfully")  
 except Exception as e:  
 logger.error(f"Failed to initialize IndicatorEngine: {e}", exc\_info=True)  
 raise  
  
 async def load\_historical(self, symbol: str, timeframe: str, lookback: int) -> List[Dict]:  
 """Load historical OHLCV from Redis or PostgreSQL."""  
 logger.debug(f"Loading historical OHLCV for {symbol}:{timeframe} with lookback {lookback}")  
 try:  
 # Try Redis first  
 cached = await self.redis\_client.get(f"{symbol}:ohlcv:{timeframe}")  
 if cached and len(cached) >= lookback:  
 logger.debug(f"Retrieved {len(cached)} candles from Redis for {symbol}:{timeframe}")  
 return cached  
  
 # Fallback to PostgreSQL  
 async with self.database.pool.acquire() as conn:  
 rows = await conn.fetch("""  
 SELECT \* FROM anquant.ohlcv   
 WHERE tradingsymbol = $1 AND timeframe = $2   
 ORDER BY timestamp DESC LIMIT $3  
 """, symbol, timeframe, lookback)  
 historical = [{  
 'timestamp': r['timestamp'].isoformat(),  
 'open': r['open'],  
 'high': r['high'],  
 'low': r['low'],  
 'close': r['close'],  
 'volume': r['volume'],  
 'tradingsymbol': r['tradingsymbol'],  
 'exchange': r['exchange'],  
 'timeframe': r['timeframe']  
 } for r in rows]  
 await self.redis\_client.cache(f"{symbol}:ohlcv:{timeframe}", historical, ttl=86400)  
 logger.debug(  
 f"Retrieved {len(historical)} candles from PostgreSQL and cached in Redis for {symbol}:{timeframe}")  
 return historical  
 except Exception as e:  
 logger.error(f"Failed to load historical OHLCV for {symbol}:{timeframe}: {e}", exc\_info=True)  
 raise  
  
 async def start(self):  
 """Start IndicatorEngine, backfill indicators, and process ticks."""  
 logger.info("Starting IndicatorEngine")  
 try:  
 if self.offline\_mode:  
 logger.info("Offline mode: Skipping tick processing")  
 return  
  
 if self.kafka\_client:  
 self.kafka\_client.connect()  
 self.kafka\_client.subscribe(["nse\_ticks"])  
 logger.debug("Subscribed to Kafka topic: nse\_ticks")  
 await self.redis\_client.connect()  
 await self.redis\_client.redis.ping()  
 logger.debug("Verified Redis connectivity")  
 await self.\_backfill\_indicators()  
 asyncio.create\_task(self.\_process\_ticks())  
 logger.info("IndicatorEngine started successfully")  
 except Exception as e:  
 logger.error(f"Failed to start IndicatorEngine: {e}", exc\_info=True)  
 raise  
  
 async def \_backfill\_indicators(self):  
 """Compute indicators for backfilled OHLCV data."""  
 try:  
 now = datetime.now(pytz.timezone('Asia/Kolkata'))  
 for market, market\_config in self.markets.items():  
 open\_time = datetime.strptime(market\_config['open\_time'], '%H:%M:%S').time()  
 open\_datetime = datetime.combine(now.date(), open\_time, tzinfo=pytz.timezone('Asia/Kolkata'))  
 if now.time() > open\_time and now.date() == open\_datetime.date():  
 logger.info(f"Backfilling indicators for {market} from {open\_datetime} to {now}")  
 for timeframe in self.timeframes:  
 for watchlist\_name, watchlist\_path in market\_config['watchlists'].items():  
 symbols = await self.\_load\_watchlist(watchlist\_path)  
 for symbol in symbols:  
 ohlcv = await self.redis\_client.get(f"{symbol}:ohlcv:{timeframe}")  
 if ohlcv:  
 df = pd.DataFrame(ohlcv)  
 indicators = self.\_compute\_indicators(df, self.indicators\_config)  
 await self.redis\_client.cache(f"{symbol}:indicators:{timeframe}", indicators,  
 ttl=86400)  
 logger.debug(  
 f"Backfilled indicators for {symbol} ({market}, {timeframe}): {json.dumps(indicators)}")  
 else:  
 logger.warning(  
 f"No OHLCV data found for {symbol} ({market}, {timeframe}) during backfill")  
 except Exception as e:  
 logger.error(f"Error backfilling indicators: {e}", exc\_info=True)  
 raise  
  
 async def \_process\_ticks(self):  
 """Process real-time ticks and compute indicators."""  
 logger.info("Starting tick processing loop")  
 try:  
 while True:  
 msg = self.kafka\_client.poll(timeout=1.0)  
 if msg:  
 tick = msg['value']  
 symbol = tick['tradingsymbol']  
 market = tick.get('market', 'india')  
 timestamp = datetime.fromisoformat(tick['timestamp'].replace('Z', '+00:00'))  
 for timeframe in self.timeframes:  
 ohlcv = self.aggregate\_ohlcv(symbol, timeframe, tick, timestamp)  
 if ohlcv:  
 df = pd.DataFrame([ohlcv])  
 indicators = self.compute\_indicators(symbol, timeframe, ohlcv['close'])  
 await self.redis\_client.cache(f"{symbol}:ohlcv:{timeframe}",  
 self.ohlcv\_state[symbol][timeframe], ttl=86400)  
 await self.redis\_client.cache(f"{symbol}:indicators:{timeframe}", indicators, ttl=86400)  
 await self.database.save\_ohlcv(symbol, timeframe, df)  
 if self.kafka\_client:  
 self.kafka\_client.produce(f"ohlcv\_{timeframe}", key=symbol, value=ohlcv)  
 logger.debug(  
 f"Processed tick for {symbol} ({market}, {timeframe}): OHLCV={json.dumps(ohlcv)}, Indicators={json.dumps(indicators)}")  
 else:  
 logger.debug("No tick message received, continuing polling")  
 await asyncio.sleep(0.1)  
 except Exception as e:  
 logger.error(f"Error in tick processing loop: {e}", exc\_info=True)  
 raise  
  
 def aggregate\_ohlcv(self, symbol: str, timeframe: str, tick: Dict, timestamp: datetime) -> Dict:  
 """Aggregate ticks into OHLCV for the specified timeframe."""  
 logger.debug(f"Aggregating tick for {symbol} ({timeframe})")  
 try:  
 duration\_minutes = {'1min': 1, '5min': 5, '15min': 15, '30min': 30, '1hr': 60}  
 duration = timedelta(minutes=duration\_minutes.get(timeframe, 5))  
 rounded\_timestamp = timestamp - (timestamp - datetime.min) % duration  
  
 self.ohlcv\_state.setdefault(symbol, {}).setdefault(timeframe, [])  
 ltp = tick.get('ltp', tick.get('price', 0.0))  
 volume = tick.get('volume', 1)  
  
 timeframe\_state = self.ohlcv\_state[symbol][timeframe]  
 if timeframe\_state and timeframe\_state[-1]['timestamp'] == rounded\_timestamp.isoformat():  
 last\_ohlcv = timeframe\_state[-1]  
 last\_ohlcv['high'] = max(last\_ohlcv['high'], ltp)  
 last\_ohlcv['low'] = min(last\_ohlcv['low'], ltp)  
 last\_ohlcv['close'] = ltp  
 last\_ohlcv['volume'] += volume  
 logger.debug(f"Updated OHLCV for {symbol} ({timeframe}): {json.dumps(last\_ohlcv)}")  
 return last\_ohlcv  
  
 ohlcv = {  
 'tradingsymbol': symbol,  
 'open': ltp,  
 'high': ltp,  
 'low': ltp,  
 'close': ltp,  
 'volume': volume,  
 'timestamp': rounded\_timestamp.isoformat(),  
 'exchange': 'NSE',  
 'market': 'india'  
 }  
 timeframe\_state.append(ohlcv)  
 logger.debug(f"Created new OHLCV for {symbol} ({timeframe}): {json.dumps(ohlcv)}")  
 return ohlcv  
 except Exception as e:  
 logger.error(f"Error aggregating OHLCV for {symbol} ({timeframe}): {e}", exc\_info=True)  
 return {}  
  
 def compute\_indicators(self, symbol: str, timeframe: str, close: float) -> Dict:  
 """Compute indicators for the given symbol and timeframe."""  
 logger.debug(f"Computing indicators for {symbol} ({timeframe})")  
 try:  
 self.indicator\_state.setdefault(symbol, {}).setdefault(timeframe, {'closes': np.array([])})  
 closes = self.indicator\_state[symbol][timeframe]['closes'].tolist()  
 closes.append(close)  
 self.indicator\_state[symbol][timeframe]['closes'] = np.array(closes[-60:]) # Keep last 60 closes  
 df = pd.DataFrame({'close': closes[-60:]})  
  
 indicators = {}  
 for ind in self.indicators\_config:  
 ind\_type = ind['type']  
 ind\_name = ind['name']  
 if ind\_type == 'rsi':  
 rsi = ta.rsi(df['close'], length=ind.get('period', 14))  
 indicators[ind\_name] = rsi.iloc[-1] if not rsi.empty and not pd.isna(rsi.iloc[-1]) else 0.0  
 logger.debug(  
 f"Computed RSI for {symbol} ({timeframe}, period={ind.get('period', 14)}): {indicators[ind\_name]}")  
 elif ind\_type == 'bollinger\_bands':  
 bb = ta.bbands(df['close'], length=ind.get('period', 20), std=ind.get('std', 2.0))  
 indicators[f"{ind\_name}\_upper"] = bb[f'BBU\_{ind["period"]}\_{ind["std"]}'].iloc[  
 -1] if not bb.empty and not pd.isna(bb.iloc[-1][0]) else 0.0  
 indicators[f"{ind\_name}\_mid"] = bb[f'BBM\_{ind["period"]}\_{ind["std"]}'].iloc[  
 -1] if not bb.empty and not pd.isna(bb.iloc[-1][1]) else 0.0  
 indicators[f"{ind\_name}\_lower"] = bb[f'BBL\_{ind["period"]}\_{ind["std"]}'].iloc[  
 -1] if not bb.empty and not pd.isna(bb.iloc[-1][2]) else 0.0  
 logger.debug(  
 f"Computed Bollinger Bands for {symbol} ({timeframe}, period={ind.get('period', 20)}, std={ind.get('std', 2.0)}): {json.dumps(indicators)}")  
 elif ind\_type == 'atr':  
 atr = ta.atr(df['high'], df['low'], df['close'], length=ind.get('period', 14))  
 indicators[ind\_name] = atr.iloc[-1] if not atr.empty and not pd.isna(atr.iloc[-1]) else 0.0  
 logger.debug(  
 f"Computed ATR for {symbol} ({timeframe}, period={ind.get('period', 14)}): {indicators[ind\_name]}")  
 return indicators  
 except Exception as e:  
 logger.error(f"Error computing indicators for {symbol} ({timeframe}): {e}", exc\_info=True)  
 return {}  
  
 async def \_load\_watchlist(self, watchlist\_path: str) -> Set[str]:  
 """Load watchlist symbols from YAML file."""  
 logger.debug(f"Loading watchlist: {watchlist\_path}")  
 try:  
 watchlist\_config = load\_config(watchlist\_path)  
 if not watchlist\_config or 'stocks' not in watchlist\_config:  
 logger.warning(f"Empty or invalid watchlist: {watchlist\_path}")  
 return set()  
 symbols = {item['tradingsymbol'] for item in watchlist\_config['stocks']}  
 logger.debug(f"Loaded watchlist with {len(symbols)} symbols")  
 return symbols  
 except Exception as e:  
 logger.error(f"Failed to load watchlist {watchlist\_path}: {e}", exc\_info=True)  
 return set()  
  
 async def close(self):  
 """Close IndicatorEngine connections and adapters."""  
 logger.info("Closing IndicatorEngine")  
 try:  
 tasks = []  
 for market in self.adapters:  
 for broker in self.adapters[market]:  
 logger.debug(f"Disconnecting adapter for {market}/{broker}")  
 tasks.append(self.adapters[market][broker].disconnect())  
 await asyncio.gather(\*tasks)  
 if self.kafka\_client:  
 self.kafka\_client.close()  
 await self.redis\_client.close()  
 self.database.close()  
 logger.info("IndicatorEngine closed successfully")  
 except Exception as e:  
 logger.error(f"Error closing IndicatorEngine: {e}", exc\_info=True)  
 raise

## src\anquant\py\indicators\\_\_init\_\_.py

## src\anquant\py\messaging\kafka\_client.py

# D:\AlphaNivesh\ANQuant\src\anquant\py\messaging\kafka\_client.py  
import json  
import time  
from typing import Dict, List, Optional  
from confluent\_kafka import Producer, Consumer, KafkaError, KafkaException  
from src.anquant.py.util.logging import setup\_logging  
  
logger = setup\_logging("kafka\_client", log\_type="messaging")  
  
class KafkaClient:  
 def \_\_init\_\_(self, config: Dict):  
 """  
 Initialize Kafka producer and consumer with market-specific logging.  
  
 Args:  
 config (Dict): Kafka configuration from config.yaml (e.g., brokers, topics).  
 """  
 self.brokers = config.get('brokers', 'localhost:9092')  
 self.topics = config.get('topics', {})  
  
 producer\_config = {  
 'bootstrap.servers': self.brokers,  
 'client.id': 'anquant-producer',  
 'queue.buffering.max.messages': 1000000,  
 'queue.buffering.max.ms': 100,  
 'message.max.bytes': 1000000,  
 'compression.type': 'gzip',  
 'retries': 3,  
 'retry.backoff.ms': 500  
 }  
  
 consumer\_config = {  
 'bootstrap.servers': self.brokers,  
 'group.id': 'anquant',  
 'auto.offset.reset': 'latest',  
 'enable.auto.commit': False, # Changed to manual commits for reliability  
 }  
  
 try:  
 self.producer = Producer(producer\_config)  
 self.consumer = Consumer(consumer\_config)  
 logger.info("Initialized KafkaClient", market="none", topic="none")  
 except KafkaException as e:  
 logger.error(f"Failed to initialize KafkaClient: {str(e)}", market="none", topic="none", exc\_info=True)  
 raise  
  
 def produce(self, topic: str, key: str, value: Dict, partition: Optional[int] = None):  
 """  
 Produce a message to a Kafka topic with market-specific logging.  
  
 Args:  
 topic (str): Kafka topic (e.g., nse\_ticks, ohlcv\_5min).  
 key (str): Message key (e.g., tradingsymbol like RELIANCE-EQ).  
 value (Dict): Message value with market or exchange field.  
 partition (Optional[int]): Specific partition (optional).  
 """  
 try:  
 logger.debug(f"[DEBUG] About to access value.get('market') for topic={topic}, key={key}, value={value}")  
 market = value.get('market', value.get('exchange', 'unknown')).lower()  
 logger.debug(f"[DEBUG] market resolved: {market}")  
 logger.debug(f"[DEBUG] About to access value['tradingsymbol'] for topic={topic}, key={key}, value={value}")  
 \_ = value['tradingsymbol']  
 logger.debug(f"[DEBUG] value['tradingsymbol'] accessed successfully: {value['tradingsymbol']}")  
 logger.debug(f"[DEBUG] Preparing to serialize value for topic={topic}, key={key}, value={value}")  
 value\_str = json.dumps(value)  
 logger.debug(f"[DEBUG] Serialization complete for topic={topic}, key={key}, value\_str={value\_str}")  
 logger.debug(f"[DEBUG] About to call producer.produce for topic={topic}, key={key}")  
 if partition is not None:  
 self.producer.produce(  
 topic=topic,  
 key=key,  
 value=value\_str,  
 partition=partition,  
 on\_delivery=lambda err, msg: self.\_delivery\_callback(err, msg, topic, key, market)  
 )  
 else:  
 self.producer.produce(  
 topic=topic,  
 key=key,  
 value=value\_str,  
 on\_delivery=lambda err, msg: self.\_delivery\_callback(err, msg, topic, key, market)  
 )  
 self.producer.flush(timeout=0.1) # Ensure timely delivery  
 except Exception as e:  
 logger.error(f"[KafkaClient] Exception in produce for topic={topic}, key={key}: {e}", market=market, topic=topic, exc\_info=True)  
 raise  
  
 def \_delivery\_callback(self, err, msg, topic: str, key: str, market: str):  
 """  
 Handle delivery confirmation or errors for produced messages.  
  
 Args:  
 err: Error information (None if successful).  
 msg: Message metadata.  
 topic (str): Kafka topic.  
 key (str): Message key.  
 market (str): Market identifier.  
 """  
 try:  
 if err is not None:  
 logger.error(f"[KafkaClient] Delivery failed for key {key}: {str(err)}", market=market, topic=topic, exc\_info=True)  
 else:  
 logger.info(f"[KafkaClient] Delivered message to partition {msg.partition()} with key {key}", market=market, topic=topic)  
 logger.debug(f"[KafkaClient] Message details - topic: {msg.topic()}, offset: {msg.offset()}, timestamp: {msg.timestamp()}, key: {key}, value: {msg.value()}", market=market, topic=topic)  
 except Exception as e:  
 logger.error(f"[KafkaClient] Exception in delivery callback for topic={topic}, key={key}: {e}", market=market, topic=topic, exc\_info=True)  
  
 def subscribe(self, topics: List[str]):  
 """  
 Subscribe consumer to a list of topics.  
  
 Args:  
 topics (List[str]): List of topics (e.g., ['nse\_ticks', 'ohlcv\_5min']).  
 """  
 try:  
 self.consumer.subscribe(topics)  
 logger.info(f"Subscribed to topics: {topics}", market="none", topic=",".join(topics))  
 except KafkaException as e:  
 logger.error(f"Failed to subscribe to topics {topics}: {str(e)}", market="none", topic="none", exc\_info=True)  
 raise  
  
 def poll(self, timeout: float = 1.0) -> Optional[Dict]:  
 """  
 Poll for new messages from subscribed topics.  
  
 Args:  
 timeout (float): Poll timeout in seconds.  
  
 Returns:  
 Optional[Dict]: Deserialized message or None.  
 """  
 try:  
 start\_time = time.time()  
 msg = self.consumer.poll(timeout=timeout)  
 if msg is None:  
 logger.debug(f"No message received after {timeout}s", market="none", topic="none")  
 return None  
 if msg.error():  
 if msg.error().code() == KafkaError.\_PARTITION\_EOF:  
 logger.debug(f"Reached end of partition for {msg.topic()}", market="none", topic=msg.topic())  
 return None  
 logger.error(f"Kafka error for {msg.topic()}: {msg.error()}", market="none", topic=msg.topic(), exc\_info=True)  
 raise KafkaException(msg.error())  
 value = json.loads(msg.value().decode('utf-8'))  
 market = value.get('market', value.get('exchange', 'unknown')).lower()  
 logger.debug(  
 f"Consumed message from partition {msg.partition()} with key {msg.key().decode('utf-8') if msg.key() else 'None'}, "  
 f"size {len(msg.value())} bytes, latency {time.time() - start\_time:.3f}s",  
 market=market,  
 topic=msg.topic()  
 )  
 self.consumer.commit(msg) # Manual commit for reliability  
 return {  
 'topic': msg.topic(),  
 'partition': msg.partition(),  
 'key': msg.key().decode('utf-8') if msg.key() else None,  
 'value': value  
 }  
 except Exception as e:  
 logger.error(f"Failed to poll messages: {str(e)}", market="none", topic="none", exc\_info=True)  
 raise  
  
 def close(self):  
 """  
 Close producer and consumer connections.  
 """  
 try:  
 self.producer.flush(timeout=5.0)  
 self.consumer.close()  
 logger.info("Closed KafkaClient connections", market="none", topic="none")  
 except Exception as e:  
 logger.error(f"Failed to close KafkaClient: {str(e)}", market="none", topic="none", exc\_info=True)  
 raise

## src\anquant\py\messaging\redis\_client.py

from typing import Dict, Any, List, Optional  
import redis.asyncio as redis  
from src.anquant.py.util.logging import setup\_logging  
import json  
import asyncio  
  
logger = setup\_logging("redis\_client", log\_type="messaging")  
  
class RedisClient:  
 def \_\_init\_\_(self, config: Dict[str, Any]):  
 self.config = config  
 self.redis = redis.Redis(  
 host=config.get('host', 'localhost'),  
 port=config.get('port', 6379),  
 password=config.get('password', None),  
 decode\_responses=True,  
 socket\_connect\_timeout=5,  
 socket\_timeout=5,  
 retry\_on\_timeout=True,  
 health\_check\_interval=30  
 )  
 logger.debug("Initialized RedisClient")  
  
 async def test\_connection(self):  
 """Test Redis connection."""  
 try:  
 logger.debug("[RedisClient] Testing connection...")  
 await asyncio.wait\_for(self.redis.ping(), timeout=5.0)  
 logger.debug("Redis connection test successful")  
 return True  
 except asyncio.TimeoutError:  
 logger.error("Redis connection timeout")  
 return False  
 except Exception as e:  
 logger.error(f"Redis connection test failed: {str(e)}")  
 return False  
  
 async def publish(self, channel: str, message: str):  
 try:  
 logger.debug(f"[RedisClient] [TRACE] Starting publish for channel={channel}")  
 logger.debug(f"[RedisClient] [TRACE] Redis client state: {self.redis}")  
 logger.debug(f"[RedisClient] [TRACE] About to call redis.publish")  
   
 # Direct publish without connection test  
 logger.debug(f"[RedisClient] [TRACE] Calling redis.publish directly")  
 result = await self.redis.publish(channel, message)  
 logger.debug(f"[RedisClient] [TRACE] Redis publish completed, result={result}")  
 logger.debug(f"[RedisClient] Successfully published to channel={channel}")  
 except Exception as e:  
 logger.error(f"[RedisClient] Exception in publish for channel={channel}: {str(e)}", exc\_info=True)  
 raise  
  
 async def cache(self, key: str, value: List[Dict], ttl: int):  
 """Cache data in Redis with a specified TTL (in seconds)."""  
 try:  
 logger.debug(f"[RedisClient] Preparing to serialize value for key={key}, value={value}")  
 value\_str = json.dumps(value)  
 logger.debug(f"[RedisClient] Serialization complete for key={key}, value\_str={value\_str}")  
 await self.redis.setex(key, ttl, value\_str)  
 logger.debug(f"[RedisClient] Successfully cached data for key={key} with TTL={ttl}s")  
 except Exception as e:  
 logger.error(f"[RedisClient] Exception in cache for key={key}: {str(e)}", exc\_info=True)  
 raise  
  
 async def get(self, key: str) -> List[Dict]:  
 """Retrieve cached data from Redis."""  
 try:  
 data = await self.redis.get(key)  
 if data:  
 return json.loads(data)  
 logger.debug(f"No data found for key {key}")  
 return []  
 except Exception as e:  
 logger.error(f"Failed to retrieve data for key {key}: {str(e)}")  
 raise  
  
 async def close(self):  
 await self.redis.close()  
 logger.debug("Closed Redis connection")

## src\anquant\py\messaging\\_\_init\_\_.py

## src\anquant\py\order\_management\order\_execution.py

# src/py/core/order\_execution.py  
from typing import Dict, Any  
from loguru import logger  
from src.anquant.py.core.adapters import get\_adapters  
from src.anquant.py.util.logging import setup\_logging  
  
logger = setup\_logging("order\_execution\_engine", log\_type="general")  
  
class OrderExecutionEngine:  
 def \_\_init\_\_(self, config: Dict[str, Any]):  
 self.config = config  
 self.adapters = get\_adapters(config)  
 logger.debug("Initialized OrderExecutionEngine with adapters for markets: {}".format(list(self.adapters.keys())))  
  
 async def initialize(self):  
 logger.debug("OrderExecutionEngine initializing")  
 try:  
 for market, brokers in self.adapters.items():  
 for broker, adapter in brokers.items():  
 await adapter.connect()  
 logger.info(f"Connected to {broker} adapter for market {market}")  
 except Exception as e:  
 logger.error(f"Failed to initialize OrderExecutionEngine: {e}")  
 raise  
  
 async def start(self):  
 logger.debug("OrderExecutionEngine starting") # Order execution runs passively, no active tasks needed  
 return None  
  
 async def stop(self):  
 logger.debug("OrderExecutionEngine stopping")  
 # No cleanup needed for passive order execution

## src\anquant\py\order\_management\\_\_init\_\_.py

## src\anquant\py\portfolio\portfolio\_manager.py

# src/py/core/portfolio\_manager.py  
from typing import Dict, Any  
from loguru import logger  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.util.database import Database  
from src.anquant.py.util.logging import setup\_logging  
  
logger = setup\_logging("portfolio\_manager", log\_type="general")  
  
class PortfolioManager:  
 def \_\_init\_\_(self, config: Dict[str, Any], redis\_client: RedisClient, database: Database):  
 self.config = config  
 self.redis\_client = redis\_client  
 self.database = database  
 logger.debug("Initialized PortfolioManager")  
  
 async def initialize(self):  
 logger.debug("PortfolioManager initializing")  
 cursor = self.database.conn.cursor()  
 try:  
 cursor.execute("""  
 CREATE TABLE IF NOT EXISTS anquant.trades (  
 trade\_id VARCHAR(50) PRIMARY KEY,  
 timestamp TIMESTAMP,  
 tradingsymbol VARCHAR(50),  
 exchange VARCHAR(10),  
 side VARCHAR(10),  
 quantity INTEGER,  
 price FLOAT,  
 strategy VARCHAR(50)  
 )  
 """)  
 self.database.conn.commit()  
 logger.debug("Ensured anquant.trades table exists")  
 except Exception as e:  
 self.database.conn.rollback()  
 logger.error(f"Failed to initialize anquant.trades table: {e}")  
 raise  
 finally:  
 cursor.close()  
  
 async def start(self):  
 logger.debug("PortfolioManager starting") # Portfolio management runs passively, no active tasks needed  
 return None  
  
 async def stop(self):  
 logger.debug("PortfolioManager stopping")  
 # No cleanup needed for passive portfolio management

## src\anquant\py\portfolio\\_\_init\_\_.py

## src\anquant\py\util\config\_loader.py

# src/py/anquant/util/config\_loader.py  
import os  
import yaml  
import json  
import logging  
from typing import Dict, Any, Optional  
from datetime import datetime  
  
  
class JSONFormatter(logging.Formatter):  
 """Custom JSON formatter for structured logging."""  
  
 def format(self, record):  
 log\_record = {  
 "timestamp": datetime.utcnow().isoformat() + "Z",  
 "level": record.levelname,  
 "message": record.getMessage(),  
 "module": record.module,  
 "funcName": record.funcName,  
 "line": record.lineno,  
 "process": record.process,  
 "thread": record.thread  
 }  
 if record.exc\_info:  
 log\_record["exception"] = self.formatException(record.exc\_info)  
 return json.dumps(log\_record)  
  
  
def setup\_logging(name: str, log\_type: str = "general") -> logging.Logger:  
 """Configure logger with JSON formatting and file output."""  
 logger = logging.getLogger(name)  
 logger.setLevel(logging.DEBUG)  
  
 try:  
 log\_dir = os.path.join("logs", log\_type)  
 os.makedirs(log\_dir, exist\_ok=True)  
  
 log\_file = os.path.join(log\_dir, f"{name}\_{datetime.now().strftime('%Y-%m-%d')}.log")  
 file\_handler = logging.FileHandler(log\_file, encoding='utf-8')  
 file\_handler.setLevel(logging.DEBUG)  
 file\_handler.setFormatter(JSONFormatter())  
  
 console\_handler = logging.StreamHandler()  
 console\_handler.setLevel(logging.INFO)  
 console\_handler.setFormatter(JSONFormatter())  
  
 logger.handlers = [file\_handler, console\_handler]  
 logger.propagate = False  
  
 logger.info(f"Logging initialized for {name} (type: {log\_type}, file: {log\_file})")  
 return logger  
 except Exception as e:  
 fallback\_logger = logging.getLogger(f"{name}\_fallback")  
 fallback\_logger.setLevel(logging.DEBUG)  
 console\_handler = logging.StreamHandler()  
 console\_handler.setLevel(logging.DEBUG)  
 console\_handler.setFormatter(JSONFormatter())  
 fallback\_logger.handlers = [console\_handler]  
 fallback\_logger.error(f"Failed to initialize logging for {name} (type: {log\_type}): {e}", exc\_info=True)  
 return fallback\_logger  
  
  
logger = setup\_logging("config\_loader", log\_type="general")  
  
  
def load\_config(config\_path: str = "config/config.yaml", require\_global: bool = True) -> Dict[str, Any]:  
 """Load and validate configuration from a YAML file."""  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../../../')) # ANQuant/  
 absolute\_path = os.path.join(project\_root, config\_path)  
 logger.info(f"Loading configuration from {absolute\_path}")  
 try:  
 if not os.path.exists(absolute\_path):  
 logger.error(f"Configuration file {absolute\_path} does not exist")  
 raise FileNotFoundError(f"Configuration file {absolute\_path} not found")  
  
 with open(absolute\_path, 'r', encoding='utf-8') as f:  
 config = yaml.safe\_load(f)  
  
 if config is None:  
 logger.error(f"Configuration file {absolute\_path} is empty or invalid")  
 raise ValueError(f"Configuration file {absolute\_path} is empty or invalid")  
  
 # Validate key sections  
 if require\_global:  
 required\_sections = ['global']  
 for section in required\_sections:  
 if section not in config:  
 logger.error(f"Missing required section '{section}' in {absolute\_path}")  
 raise ValueError(f"Missing required section '{section}' in {absolute\_path}")  
  
 # Validate global sub-sections  
 global\_required = ['kafka', 'redis', 'database', 'historical\_data', 'strategies', 'markets', 'audit']  
 for key in global\_required:  
 if key not in config['global']:  
 logger.warning(f"Missing global section '{key}' in {absolute\_path}, using default configuration")  
 config['global'][key] = {}  
  
 # Log loaded markets  
 markets = list(config['global']['markets'].keys()) if 'markets' in config['global'] else []  
 logger.info(f"Loaded markets: {markets}")  
  
 logger.info(f"Successfully loaded and validated configuration from {absolute\_path}")  
 logger.debug(f"Configuration content: {json.dumps(config, default=str)}")  
 return config  
 except yaml.YAMLError as e:  
 logger.error(f"Failed to parse YAML in {absolute\_path}: {e}", exc\_info=True)  
 raise  
 except Exception as e:  
 logger.error(f"Failed to load configuration from {absolute\_path}: {e}", exc\_info=True)  
 raise  
  
  
def load\_symbol\_mappings(config: Dict[str, Any], market: str, broker: str) -> Dict[str, Any]:  
 """Load broker-specific symbol mappings for a given market."""  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../../../'))  
 try:  
 mappings\_path = config["global"]["markets"][market]["brokers"][broker]["symbols"]  
 absolute\_path = os.path.join(project\_root, mappings\_path)  
 logger.debug(f"Attempting to load symbol mappings from: {absolute\_path}")  
  
 if not os.path.exists(absolute\_path):  
 logger.error(f"Symbol mappings file {absolute\_path} not found")  
 raise FileNotFoundError(f"Symbol mappings file {absolute\_path} not found")  
  
 with open(absolute\_path, 'r', encoding='utf-8') as f:  
 mappings = yaml.safe\_load(f)  
  
 if mappings is None:  
 logger.warning(f"Symbol mappings file {absolute\_path} is empty, returning empty dict")  
 mappings = {}  
  
 logger.info(f"Loaded symbol mappings for {market}/{broker} from {absolute\_path}")  
 logger.debug(f"Symbol mappings content: {json.dumps(mappings, default=str)}")  
 return mappings  
 except yaml.YAMLError as e:  
 logger.error(f"Failed to parse YAML in {absolute\_path}: {e}", exc\_info=True)  
 raise  
 except KeyError as e:  
 logger.error(f"Invalid config structure for {market}/{broker} symbol mappings: {e}", exc\_info=True)  
 raise  
 except Exception as e:  
 logger.error(f"Failed to load symbol mappings for {market}/{broker}: {e}", exc\_info=True)  
 raise  
  
  
def load\_credentials(config: Dict[str, Any], market: str, broker: str) -> Dict[str, Any]:  
 """Load credentials for a given market and broker from file or Vault."""  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../../../'))  
 try:  
 offline\_mode = config['global'].get('offline\_mode', False)  
 vault\_enabled = config['global'].get('vault', {}).get('enabled', False)  
  
 if offline\_mode or not vault\_enabled:  
 cred\_path = config['global']['markets'][market]['brokers'][broker]['credentials']  
 absolute\_path = os.path.join(project\_root, cred\_path)  
 logger.debug(f"Attempting to load credentials from: {absolute\_path}")  
  
 if not os.path.exists(absolute\_path):  
 logger.error(f"Credentials file {absolute\_path} not found")  
 raise FileNotFoundError(f"Credentials file {absolute\_path} not found")  
  
 with open(absolute\_path, 'r', encoding='utf-8') as f:  
 credentials = yaml.safe\_load(f)  
  
 if credentials is None:  
 logger.error(f"Credentials file {absolute\_path} is empty")  
 raise ValueError(f"Credentials file {absolute\_path} is empty")  
  
 logger.info(f"Loaded credentials from {absolute\_path}")  
 logger.debug(f"Credentials content (sanitized): {json.dumps({k: '\*\*\*' for k in credentials.keys()})}")  
 return credentials  
 else:  
 from hvac import Client  
 vault\_url = config['global']['vault'].get('url', 'http://vault:8200')  
 logger.debug(f"Attempting to load credentials from Vault at {vault\_url}")  
  
 vault\_client = Client(url=vault\_url)  
 vault\_path = f"anquant/secrets/{market}/{broker}"  
 secret = vault\_client.secrets.kv.v2.read\_secret\_version(path=vault\_path, mount\_point='anquant')  
 credentials = secret['data']['data']  
  
 logger.info(f"Loaded credentials from Vault for {market}/{broker}")  
 logger.debug(f"Credentials content (sanitized): {json.dumps({k: '\*\*\*' for k in credentials.keys()})}")  
 return credentials  
 except yaml.YAMLError as e:  
 logger.error(f"Failed to parse YAML credentials for {market}/{broker}: {e}", exc\_info=True)  
 raise  
 except KeyError as e:  
 logger.error(f"Invalid config structure for {market}/{broker} credentials: {e}", exc\_info=True)  
 raise  
 except Exception as e:  
 logger.error(f"Failed to load credentials for {market}/{broker}: {e}", exc\_info=True)  
 raise

## src\anquant\py\util\database.py

from typing import Dict, Any  
import psycopg2  
from loguru import logger  
from src.anquant.py.util.logging import setup\_logging  
import pandas as pd  
  
logger = setup\_logging("database", log\_type="general")  
  
class Database:  
 def \_\_init\_\_(self, config: Dict[str, Any]):  
 self.conn = psycopg2.connect(  
 dbname=config.get("dbname", "ANQuantDB"),  
 user=config.get("user", "postgres"),  
 password=config.get("password", ""),  
 host=config.get("host", "localhost"),  
 port=config.get("port", 5432)  
 )  
 cursor = self.conn.cursor()  
 try:  
 cursor.execute("CREATE SCHEMA IF NOT EXISTS anquant")  
 cursor.execute("SET search\_path TO anquant")  
 cursor.execute("GRANT CREATE, USAGE ON SCHEMA anquant TO CURRENT\_USER")  
 self.conn.commit()  
 logger.debug("Initialized Database connection with anquant schema")  
 except psycopg2.errors.InsufficientPrivilege as e:  
 self.conn.rollback()  
 logger.error(f"Insufficient privileges for anquant schema in {config.get('dbname', 'ANQuantDB')}: {e}")  
 raise  
 except Exception as e:  
 self.conn.rollback()  
 logger.error(f"Failed to initialize anquant schema: {e}")  
 raise  
 finally:  
 cursor.close()  
  
 def save\_ohlcv(self, symbol: str, timeframe: str, df: pd.DataFrame):  
 cursor = self.conn.cursor()  
 try:  
 cursor.execute("""  
 CREATE TABLE IF NOT EXISTS anquant.ohlcv (  
 timestamp TIMESTAMP,  
 open FLOAT,  
 high FLOAT,  
 low FLOAT,  
 close FLOAT,  
 volume BIGINT,  
 tradingsymbol VARCHAR(50),  
 exchange VARCHAR(10),  
 timeframe VARCHAR(10),  
 PRIMARY KEY (timestamp, tradingsymbol, timeframe)  
 )  
 """)  
 for \_, row in df.iterrows():  
 cursor.execute("""  
 INSERT INTO anquant.ohlcv (timestamp, open, high, low, close, volume, tradingsymbol, exchange, timeframe)  
 VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)  
 ON CONFLICT (timestamp, tradingsymbol, timeframe) DO UPDATE  
 SET open = EXCLUDED.open,  
 high = EXCLUDED.high,  
 low = EXCLUDED.low,  
 close = EXCLUDED.close,  
 volume = EXCLUDED.volume,  
 exchange = EXCLUDED.exchange  
 """, (  
 row['timestamp'], row['open'], row['high'], row['low'],  
 row['close'], row['volume'], symbol, row['exchange'], timeframe  
 ))  
 self.conn.commit()  
 logger.debug(f"Saved OHLCV data for {symbol}:{timeframe} to PostgreSQL")  
 except psycopg2.errors.InsufficientPrivilege as e:  
 self.conn.rollback()  
 logger.error(f"Insufficient privileges to create table for {symbol}:{timeframe} in anquant schema: {e}")  
 raise  
 except Exception as e:  
 self.conn.rollback()  
 logger.error(f"Failed to save OHLCV for {symbol}:{timeframe}: {e}")  
 raise  
 finally:  
 cursor.close()  
  
 def close(self):  
 self.conn.close()  
 logger.debug("Closed Database connection")

## src\anquant\py\util\logging.py

import os  
import sys  
import yaml  
import stat  
from loguru import logger  
from datetime import datetime  
  
def setup\_logging(log\_name: str, log\_type: str = "general", log\_dir: str = "logs", config\_path: str = "config/config.yaml"):  
 """Initialize logging with Loguru for the specified log name and type."""  
 try:  
 # Resolve project root (anquant/) relative to this file  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../../../')) # src/anquant/py/util/ -> ANQuant/  
 config\_path = os.path.join(project\_root, config\_path) # D:\AlphaNivesh\ANQuant\config\config.yaml  
 log\_dir = os.path.join(project\_root, log\_dir) # D:\AlphaNivesh\ANQuant\logs  
 logger.debug(f"Using config path: {config\_path}, log dir: {log\_dir}")  
  
 # Load main config to get logging config path  
 try:  
 with open(config\_path, "r", encoding='utf-8') as f:  
 main\_config = yaml.safe\_load(f) or {}  
 logging\_config\_path = main\_config.get("global", {}).get("logging\_config", "config/logging.yaml")  
 logging\_config\_path = os.path.join(project\_root, logging\_config\_path) # D:\AlphaNivesh\ANQuant\config\logging.yaml  
 except FileNotFoundError:  
 logger.error(f"Main config file {config\_path} not found; using default logging config path")  
 logging\_config\_path = os.path.join(project\_root, "config/logging.yaml")  
 except Exception as e:  
 logger.error(f"Failed to load main config from {config\_path}: {e}; using default logging config path")  
 logging\_config\_path = os.path.join(project\_root, "config/logging.yaml")  
  
 # Load logging configuration  
 try:  
 with open(logging\_config\_path, "r", encoding='utf-8') as f:  
 config = yaml.safe\_load(f) or {}  
 logging\_config = config.get(log\_type, {})  
 directory = os.path.abspath(os.path.join(project\_root, logging\_config.get("directory", "logs"))) # Ensure absolute path  
 date\_str = datetime.now().strftime("%Y-%m-%d")  
 filename\_template = logging\_config.get("filename\_template", "{name}\_{date}.log").replace("{date}", date\_str)  
 if log\_type == "kafka":  
 filename\_template = f"kafka/{log\_name}\_{date\_str}.log"  
 log\_level = logging\_config.get("level", "DEBUG").upper()  
 rotation = logging\_config.get("rotation", "10 MB")  
 retention = logging\_config.get("retention", "30 days")  
 compression = logging\_config.get("compression", "zip")  
 log\_format = logging\_config.get("format", "{time:YYYY-MM-DD HH:mm:ss} | {level} | {name} | {message}")  
 stdout\_format = logging\_config.get("stdout\_format", "<green>{time:YYYY-MM-DD HH:mm:ss}</green> | <level>{level}</level> | <cyan>{name}</cyan> | <level>{message}</level>")  
 enqueue = logging\_config.get("enqueue", True)  
 except FileNotFoundError:  
 logger.error(f"Logging config file {logging\_config\_path} not found; using default settings")  
 directory = os.path.abspath(os.path.join(project\_root, "logs")) # Default to D:\AlphaNivesh\ANQuant\logs  
 date\_str = datetime.now().strftime("%Y-%m-%d")  
 filename\_template = f"{log\_name}\_{date\_str}.log"  
 if log\_type == "kafka":  
 filename\_template = f"kafka/{log\_name}\_{date\_str}.log"  
 log\_level = "DEBUG"  
 rotation = "10 MB"  
 retention = "30 days"  
 compression = "zip"  
 log\_format = "{time:YYYY-MM-DD HH:mm:ss} | {level} | {name} | {message}"  
 stdout\_format = "<green>{time:YYYY-MM-DD HH:mm:ss}</green> | <level>{level}</level> | <cyan>{name}</cyan> | <level>{message}</level>"  
 enqueue = True  
 except Exception as e:  
 logger.error(f"Failed to load logging config from {logging\_config\_path}: {e}; using default settings")  
 directory = os.path.abspath(os.path.join(project\_root, "logs")) # Default to D:\AlphaNivesh\ANQuant\logs  
 date\_str = datetime.now().strftime("%Y-%m-%d")  
 filename\_template = f"{log\_name}\_{date\_str}.log"  
 if log\_type == "kafka":  
 filename\_template = f"kafka/{log\_name}\_{date\_str}.log"  
 log\_level = "DEBUG"  
 rotation = "10 MB"  
 retention = "30 days"  
 compression = "zip"  
 log\_format = "{time:YYYY-MM-DD HH:mm:ss} | {level} | {name} | {message}"  
 stdout\_format = "<green>{time:YYYY-MM-DD HH:mm:ss}</green> | <level>{level}</level> | <cyan>{name}</cyan> | <level>{message}</level>"  
 enqueue = True  
  
 # Resolve absolute directory path  
 directory = os.path.abspath(directory)  
 logger.debug(f"Creating log directory: {directory}")  
  
 # Create logs directory (outside src/) with write permissions  
 try:  
 os.makedirs(directory, exist\_ok=True)  
 os.chmod(directory, stat.S\_IRWXU | stat.S\_IRWXG | stat.S\_IROTH | stat.S\_IXOTH)  
 if log\_type == "kafka":  
 kafka\_dir = os.path.join(directory, "kafka")  
 os.makedirs(kafka\_dir, exist\_ok=True)  
 os.chmod(kafka\_dir, stat.S\_IRWXU | stat.S\_IRWXG | stat.S\_IROTH | stat.S\_IXOTH)  
 except OSError as e:  
 logger.error(f"Failed to create or set permissions for directory {directory}: {e}")  
 raise  
  
 # Verify directory exists and is writable  
 if not os.path.exists(directory):  
 logger.error(f"Directory {directory} does not exist after creation attempt")  
 raise OSError(f"Directory creation failed: {directory}")  
 if not os.access(directory, os.W\_OK):  
 logger.error(f"No write permissions for directory {directory}")  
 raise OSError(f"No write permissions: {directory}")  
  
 # Format filename with log\_name  
 log\_file = os.path.join(directory, filename\_template.format(name=log\_name))  
 logger.debug(f"Log file path: {log\_file}")  
  
 # Verify log file can be created  
 try:  
 with open(log\_file, 'a') as f:  
 pass # Test file creation  
 except Exception as e:  
 logger.error(f"Cannot create or write to log file {log\_file}: {e}")  
 raise  
  
 # Create a new logger instance  
 component\_logger = logger.bind(name=log\_name)  
 component\_logger.remove() # Remove default handlers  
  
 # Add file handler  
 try:  
 handler\_id = component\_logger.add(  
 log\_file,  
 level=log\_level,  
 rotation=rotation,  
 retention=retention,  
 compression=compression,  
 format=log\_format,  
 enqueue=enqueue,  
 backtrace=True,  
 diagnose=True  
 )  
 logger.debug(f"Added file handler for {log\_file}, handler ID: {handler\_id}")  
 except Exception as e:  
 logger.error(f"Failed to add file handler for {log\_file}: {e}")  
 raise  
  
 # Add stdout handler  
 component\_logger.add(  
 sys.stdout,  
 level=log\_level,  
 format=stdout\_format,  
 colorize=True  
 )  
  
 component\_logger.info(f"Logging initialized for {log\_name} (type: {log\_type}, file: {log\_file})")  
 return component\_logger  
 except Exception as e:  
 logger.error(f"Failed to initialize logging for {log\_name}: {e}", exc\_info=True)  
 logger.add(sys.stdout, level="DEBUG")  
 return logger  
  
# Global default logger if needed  
default\_logger = logger

## src\anquant\py\util\\_\_init\_\_.py

## src\anquant\rs\indicator\Cargo.toml

# D:\AlphaNivesh\ANQuant\src\rs\indicator\Cargo.toml  
[package]  
name = "indicator\_engine"  
version = "0.1.0"  
edition = "2021"  
  
[lib]  
name = "indicator\_engine"  
crate-type = ["cdylib"]  
  
[dependencies]  
pyo3 = { version = "0.21.0", features = ["extension-module"] }  
pyo3-asyncio-0-21 = { version = "0.21.0", features = ["tokio-runtime"] }  
tokio = { version = "1.40", features = ["full"] }  
serde = { version = "1.0", features = ["derive"] }  
serde\_json = "1.0"  
rdkafka = { version = "0.36", features = ["cmake-build"] }  
ta = "0.5.0"  
chrono = "0.4"

## src\anquant\rs\indicator\src\lib.rs

use pyo3::prelude::\*;  
use pyo3::types::{PyDict, PyTuple, PyList};  
use rdkafka::consumer::{Consumer, StreamConsumer};  
use rdkafka::producer::{FutureProducer, FutureRecord};  
use rdkafka::config::ClientConfig;  
use rdkafka::message::Message;  
use serde\_json::{json, Value};  
use std::collections::HashMap;  
use std::sync::{Arc, Mutex};  
use ta::indicators::BollingerBands;  
use ta::Next;  
use chrono::{DateTime, Utc, Duration};  
  
#[pyclass]  
struct IndicatorEngine {  
 config: PyObject,  
 adapters: PyObject,  
 watchlists: PyObject,  
 producer: Arc<FutureProducer>,  
 consumer: Arc<StreamConsumer>,  
 offline\_mode: bool,  
 ohlcv\_state: Arc<Mutex<HashMap<String, HashMap<String, Vec<Value>>>>>,  
 bb\_state: Arc<Mutex<HashMap<String, HashMap<String, BollingerBands>>>>,  
}  
  
#[pymethods]  
impl IndicatorEngine {  
 #[new]  
 fn new(config: PyObject, adapters: PyObject, watchlists: PyObject) -> PyResult<Self> {  
 let (offline\_mode, kafka\_brokers) = Python::with\_gil(|py| -> PyResult<(bool, String)> {  
 let config\_dict = config.extract::<&PyDict>(py)?;  
 let global\_dict = config\_dict.get\_item("global")?.ok\_or\_else(|| pyo3::exceptions::PyKeyError::new\_err("Missing 'global' in config"))?.extract::<&PyDict>()?;  
 let offline = global\_dict.get\_item("offline\_mode")?.map\_or(false, |v| v.extract::<bool>().unwrap\_or(false));  
 let kafka = global\_dict.get\_item("kafka")?.extract::<&PyDict>()?.get\_item("brokers")?.extract::<String>()?;  
 Ok((offline, kafka))  
 })?;  
  
 let producer = Arc::new(ClientConfig::new()  
 .set("bootstrap.servers", &kafka\_brokers)  
 .create()  
 .expect("Producer creation failed"));  
  
 let consumer = Arc::new(ClientConfig::new()  
 .set("bootstrap.servers", &kafka\_brokers)  
 .set("group.id", "anquant")  
 .set("auto.offset.reset", "latest")  
 .create()  
 .expect("Consumer creation failed"));  
  
 Ok(IndicatorEngine {  
 config,  
 adapters,  
 watchlists,  
 producer,  
 consumer,  
 offline\_mode,  
 ohlcv\_state: Arc::new(Mutex::new(HashMap::new())),  
 bb\_state: Arc::new(Mutex::new(HashMap::new())),  
 })  
 }  
  
 fn initialize<'py>(&self, py: Python<'py>) -> PyResult<Bound<'py, PyAny>> {  
 let adapters = self.adapters.clone();  
 let watchlists = self.watchlists.clone();  
 let offline\_mode = self.offline\_mode;  
 let bb\_state = self.bb\_state.clone();  
  
 pyo3\_asyncio\_0\_21::tokio::local\_future\_into\_py(py, async move {  
 Python::with\_gil(|py| -> PyResult<()> {  
 let adapters\_dict = adapters.extract::<&PyDict>(py)?;  
 for (\_, brokers) in adapters\_dict.iter() {  
 let brokers\_dict = brokers.extract::<&PyDict>()?;  
 for (\_, adapter) in brokers\_dict.iter() {  
 adapter.getattr("connect")?.call0()?;  
 }  
 }  
 Ok(())  
 })?;  
  
 if !offline\_mode {  
 Python::with\_gil(|py| -> PyResult<()> {  
 let watchlists\_dict = watchlists.extract::<&PyDict>(py)?;  
 for (\_, watchlist) in watchlists\_dict.iter() {  
 let symbols: Vec<String> = watchlist.extract()?;  
 for symbol in symbols {  
 let hist\_module = py.import\_bound("src.py.core.historical\_data\_manager")?;  
 for timeframe in ["1min", "5min", "30min"] {  
 let hist\_list\_obj = hist\_module.getattr("load\_historical\_data")?.call1((symbol.clone(), timeframe.to\_string()))?;  
 let hist\_list = hist\_list\_obj.extract::<&PyList>(py)?;  
 let mut bb = BollingerBands::new(20, 2.0).expect("Failed to initialize Bollinger Bands");  
 for candle in hist\_list.iter() {  
 let close = candle.getattr("close")?.extract::<f64>()?;  
 bb.next(close);  
 }  
 bb\_state.lock().unwrap().entry(symbol.clone()).or\_insert\_with(HashMap::new).insert(timeframe.to\_string(), bb);  
 }  
 }  
 }  
 Ok(())  
 })?;  
 }  
  
 println!("IndicatorEngine initialized");  
 Ok(())  
 })  
 }  
  
 fn start<'py>(&self, py: Python<'py>) -> PyResult<Bound<'py, PyAny>> {  
 let adapters = self.adapters.clone();  
 let watchlists = self.watchlists.clone();  
 let offline\_mode = self.offline\_mode;  
 let consumer = self.consumer.clone();  
 let producer = self.producer.clone();  
 let ohlcv\_state = self.ohlcv\_state.clone();  
 let bb\_state = self.bb\_state.clone();  
  
 pyo3\_asyncio\_0\_21::tokio::local\_future\_into\_py(py, async move {  
 let market\_map = Python::with\_gil(|py| -> PyResult<HashMap<String, (HashMap<String, PyObject>, Vec<String>)>> {  
 let adapters\_dict = adapters.extract::<&PyDict>(py)?;  
 let watchlists\_dict = watchlists.extract::<&PyDict>(py)?;  
 let mut market\_map = HashMap::new();  
  
 for (market, brokers) in adapters\_dict.iter() {  
 let market\_str: String = market.extract()?;  
 let brokers\_dict = brokers.extract::<&PyDict>()?;  
 let symbols: Vec<String> = watchlists\_dict  
 .get\_item(market)?  
 .ok\_or\_else(|| pyo3::exceptions::PyKeyError::new\_err(format!("Missing watchlist for market: {}", market\_str)))?  
 .extract()?;  
  
 let mut broker\_map = HashMap::new();  
 for (broker, adapter) in brokers\_dict.iter() {  
 broker\_map.insert(broker.extract()?, adapter.to\_object(py));  
 }  
  
 market\_map.insert(market\_str, (broker\_map, symbols));  
 }  
  
 Ok(market\_map)  
 })?;  
  
 if !offline\_mode {  
 Python::with\_gil(|py| -> PyResult<()> {  
 for (market, (brokers, symbols)) in market\_map.iter() {  
 for (broker, adapter\_obj) in brokers {  
 let adapter = adapter\_obj.bind(py);  
 let subscribe = adapter.getattr("subscribe\_to\_ticks")?.call1((symbols.clone(),))?;  
 pyo3\_asyncio\_0\_21::tokio::into\_future(subscribe)?.await?;  
 println!("Subscribed to market: {}, broker: {}", market, broker);  
 }  
 }  
 Ok(())  
 })?;  
  
 consumer.subscribe(&["nse\_ticks"]).expect("Failed to subscribe to nse\_ticks");  
 let mut message\_stream = consumer.stream();  
 while let Some(result) = message\_stream.next().await {  
 if let Ok(msg) = result {  
 let tick: Value = serde\_json::from\_slice(msg.payload().unwrap\_or(&[])).unwrap\_or\_default();  
 let symbol = tick["tradingsymbol"].as\_str().unwrap\_or("").to\_string();  
 let timestamp\_str = tick["timestamp"].as\_str().unwrap\_or("");  
 let timestamp = DateTime::parse\_from\_rfc3339(timestamp\_str)  
 .map(|dt| dt.with\_timezone(&Utc))  
 .unwrap\_or(Utc::now());  
  
 for timeframe in ["1min", "5min", "30min"] {  
 let ohlcv = {  
 let mut ohlcv\_guard = ohlcv\_state.lock().unwrap();  
 Self::aggregate\_ohlcv(&mut \*ohlcv\_guard, &symbol, timeframe, &tick, timestamp)  
 };  
  
 let close = ohlcv["close"].as\_f64().unwrap\_or(0.0);  
 let indicators = {  
 let mut bb\_guard = bb\_state.lock().unwrap();  
 let symbol\_bb = bb\_guard.entry(symbol.clone()).or\_insert\_with(HashMap::new);  
 let bb = symbol\_bb.entry(timeframe.to\_string()).or\_insert\_with(|| BollingerBands::new(20, 2.0).expect("Failed to initialize Bollinger Bands"));  
 let result = bb.next(close);  
 json!({  
 "bb\_upper": result.upper,  
 "bb\_average": result.average,  
 "bb\_lower": result.lower  
 })  
 };  
  
 Python::with\_gil(|py| -> PyResult<()> {  
 let redis\_client\_module = py.import\_bound("src.py.messaging.redis\_client")?;  
 let redis\_client = redis\_client\_module.getattr("RedisClient")?.call0()?;  
 let key\_ohlcv = format!("{}:ohlcv:{}", symbol, timeframe);  
 let key\_indicators = format!("{}:indicators:{}", symbol, timeframe);  
 let ohlcv\_dict = Self::value\_to\_pydict(py, &ohlcv)?;  
 let args\_ohlcv = PyTuple::new\_bound(py, &[key\_ohlcv.as\_str(), ohlcv\_dict.as\_any(), 86400]);  
 let indicators\_dict = Self::value\_to\_pydict(py, &indicators)?;  
 let args\_indicators = PyTuple::new\_bound(py, &[key\_indicators.as\_str(), indicators\_dict.as\_any(), 86400]);  
 redis\_client.call\_method\_bound("cache", &args\_ohlcv, None)?;  
 redis\_client.call\_method\_bound("cache", &args\_indicators, None)?;  
  
 let database\_module = py.import\_bound("src.py.util.database")?;  
 let database = database\_module.getattr("Database")?.call0()?;  
 let args\_db = PyTuple::new\_bound(py, &[symbol.as\_str(), timeframe, ohlcv\_dict.as\_any()]);  
 database.call\_method\_bound("save\_ohlcv", &args\_db, None)?;  
  
 Ok(())  
 })?;  
  
 producer  
 .send(  
 FutureRecord::to(&format!("ohlcv\_{}", timeframe))  
 .key(&symbol)  
 .payload(&serde\_json::to\_string(&ohlcv).unwrap()),  
 std::time::Duration::from\_secs(0),  
 )  
 .await  
 .expect("Failed to publish to ohlcv");  
 }  
 }  
 }  
 } else {  
 println!("Offline mode: Skipping subscriptions and tick processing");  
 }  
  
 println!("IndicatorEngine started");  
 Ok(())  
 })  
 }  
  
 fn stop<'py>(&self, py: Python<'py>) -> PyResult<Bound<'py, PyAny>> {  
 let adapters = self.adapters.clone();  
  
 pyo3\_asyncio\_0\_21::tokio::local\_future\_into\_py(py, async move {  
 Python::with\_gil(|py| -> PyResult<()> {  
 let adapters\_dict = adapters.extract::<&PyDict>(py)?;  
 for (\_, brokers) in adapters\_dict.iter() {  
 let brokers\_dict = brokers.extract::<&PyDict>()?;  
 for (\_, adapter) in brokers\_dict.iter() {  
 let disconnect = adapter.getattr("disconnect")?.call0()?;  
 pyo3\_asyncio\_0\_21::tokio::into\_future(disconnect)?.await?;  
 }  
 }  
 Ok(())  
 })?;  
 println!("IndicatorEngine stopped");  
 Ok(())  
 })  
 }  
}  
  
impl IndicatorEngine {  
 fn aggregate\_ohlcv(  
 state: &mut HashMap<String, HashMap<String, Vec<Value>>>,  
 symbol: &str,  
 timeframe: &str,  
 tick: &Value,  
 timestamp: DateTime<Utc>,  
 ) -> Value {  
 let duration = match timeframe {  
 "1min" => Duration::minutes(1),  
 "5min" => Duration::minutes(5),  
 "30min" => Duration::minutes(30),  
 \_ => Duration::minutes(1),  
 };  
 let rounded\_timestamp = timestamp - Duration::milliseconds(timestamp.timestamp\_millis() % duration.num\_milliseconds());  
  
 let symbol\_state = state.entry(symbol.to\_string()).or\_insert\_with(HashMap::new);  
 let timeframe\_state = symbol\_state.entry(timeframe.to\_string()).or\_insert\_with(Vec::new);  
  
 let ltp = tick["ltp"].as\_f64().unwrap\_or(0.0);  
 let volume = tick.get("volume").and\_then(|v| v.as\_i64()).unwrap\_or(1);  
  
 if let Some(last\_ohlcv) = timeframe\_state.last\_mut() {  
 let last\_timestamp: DateTime<Utc> = DateTime::parse\_from\_rfc3339(last\_ohlcv["timestamp"].as\_str().unwrap())  
 .expect("Failed to parse last timestamp")  
 .with\_timezone(&Utc);  
 if last\_timestamp == rounded\_timestamp {  
 last\_ohlcv["high"] = json!(last\_ohlcv["high"].as\_f64().unwrap().max(ltp));  
 last\_ohlcv["low"] = json!(last\_ohlcv["low"].as\_f64().unwrap().min(ltp));  
 last\_ohlcv["close"] = json!(ltp);  
 last\_ohlcv["volume"] = json!(last\_ohlcv["volume"].as\_i64().unwrap() + volume);  
 return last\_ohlcv.clone();  
 }  
 }  
  
 let ohlcv = json!({  
 "tradingsymbol": symbol,  
 "open": ltp,  
 "high": ltp,  
 "low": ltp,  
 "close": ltp,  
 "volume": volume,  
 "timestamp": rounded\_timestamp.to\_rfc3339(),  
 "exchange": "NSE"  
 });  
 timeframe\_state.push(ohlcv.clone());  
 ohlcv  
 }  
  
 fn value\_to\_pydict(py: Python, value: &Value) -> PyResult<PyObject> {  
 let dict = PyDict::new\_bound(py);  
 if let Value::Object(map) = value {  
 for (k, v) in map {  
 match v {  
 Value::Null => dict.set\_item(k, py.None())?,  
 Value::Bool(b) => dict.set\_item(k, \*b)?,  
 Value::Number(n) => {  
 if let Some(i) = n.as\_i64() {  
 dict.set\_item(k, i)?;  
 } else if let Some(f) = n.as\_f64() {  
 dict.set\_item(k, f)?;  
 }  
 }  
 Value::String(s) => dict.set\_item(k, s)?,  
 Value::Array(arr) => {  
 let py\_list = arr.iter().map(|v| Self::value\_to\_pydict(py, v)).collect::<PyResult<Vec<\_>>>()?;  
 dict.set\_item(k, py\_list)?;  
 }  
 Value::Object(\_) => {  
 let nested\_dict = Self::value\_to\_pydict(py, v)?;  
 dict.set\_item(k, nested\_dict)?;  
 }  
 }  
 }  
 }  
 Ok(dict.into\_any().unbind().to\_object(py))  
 }  
}  
  
#[pymodule]  
fn indicator\_engine(py: Python, m: &Bound<'\_, PyModule>) -> PyResult<()> {  
 m.add\_class::<IndicatorEngine>()?;  
 Ok(())  
}

## src\anquant\rs\market\_data\Cargo.toml

# D:\AlphaNivesh\ANQuant\src\rs\market\_data\Cargo.toml  
[package]  
name = "market\_data\_engine"  
version = "0.1.0"  
edition = "2021"  
  
[lib]  
name = "market\_data\_engine"  
crate-type = ["cdylib"]  
  
[dependencies]  
pyo3 = { version = "0.21.0", features = ["extension-module"] }  
pyo3-asyncio-0-21 = { version = "0.21.0", features = ["tokio-runtime"] }  
tokio = { version = "1.40", features = ["full"] }  
serde = { version = "1.0", features = ["derive"] }  
serde\_json = "1.0"  
rdkafka = { version = "0.36", features = ["cmake-build"] }

## src\anquant\rs\market\_data\src\lib.rs

use pyo3::prelude::\*;  
use pyo3::types::PyDict;  
use rdkafka::producer::{FutureProducer, FutureRecord};  
use rdkafka::config::ClientConfig;  
use std::collections::HashMap;  
  
#[pyclass]  
struct MarketDataEngine {  
 config: PyObject,  
 adapters: PyObject,  
 watchlists: PyObject,  
 producer: FutureProducer,  
 offline\_mode: bool,  
}  
  
#[pymethods]  
impl MarketDataEngine {  
 #[new]  
 fn new(config: PyObject, adapters: PyObject, watchlists: PyObject) -> PyResult<Self> {  
 let (offline\_mode, kafka\_brokers) = Python::with\_gil(|py| -> PyResult<(bool, String)> {  
 let config\_dict = config.extract::<&PyDict>(py)?;  
 let global\_dict = config\_dict.get\_item("global")?.ok\_or\_else(|| pyo3::exceptions::PyKeyError::new\_err("Missing 'global' in config"))?.extract::<&PyDict>()?;  
 let offline = global\_dict.get\_item("offline\_mode")?.map\_or(false, |v| v.extract::<bool>().unwrap\_or(false));  
 let kafka = global\_dict.get\_item("kafka")?.extract::<&PyDict>()?.get\_item("brokers")?.extract::<String>()?;  
 Ok((offline, kafka))  
 })?;  
  
 let producer = ClientConfig::new()  
 .set("bootstrap.servers", &kafka\_brokers)  
 .create()  
 .expect("Producer creation failed");  
  
 Ok(MarketDataEngine {  
 config,  
 adapters,  
 watchlists,  
 producer,  
 offline\_mode,  
 })  
 }  
  
 fn initialize<'py>(&self, py: Python<'py>) -> PyResult<Bound<'py, PyAny>> {  
 let adapters = self.adapters.clone();  
  
 pyo3\_asyncio\_0\_21::tokio::local\_future\_into\_py(py, async move {  
 Python::with\_gil(|py| -> PyResult<()> {  
 let adapters\_dict = adapters.extract::<&PyDict>(py)?;  
 for (\_, brokers) in adapters\_dict.iter() {  
 let brokers\_dict = brokers.extract::<&PyDict>()?;  
 for (\_, adapter) in brokers\_dict.iter() {  
 adapter.getattr("connect")?.call0()?;  
 }  
 }  
 Ok(())  
 })?;  
 println!("MarketDataEngine initialized");  
 Ok(())  
 })  
 }  
  
 fn start<'py>(&self, py: Python<'py>) -> PyResult<Bound<'py, PyAny>> {  
 let adapters = self.adapters.clone();  
 let watchlists = self.watchlists.clone();  
 let offline\_mode = self.offline\_mode;  
 let producer = self.producer.clone();  
  
 pyo3\_asyncio\_0\_21::tokio::local\_future\_into\_py(py, async move {  
 let market\_map = Python::with\_gil(|py| -> PyResult<HashMap<String, (HashMap<String, PyObject>, Vec<String>)>> {  
 let adapters\_dict = adapters.extract::<&PyDict>(py)?;  
 let watchlists\_dict = watchlists.extract::<&PyDict>(py)?;  
 let mut market\_map = HashMap::new();  
  
 for (market, brokers) in adapters\_dict.iter() {  
 let market\_str: String = market.extract()?;  
 let brokers\_dict = brokers.extract::<&PyDict>()?;  
 let symbols: Vec<String> = watchlists\_dict  
 .get\_item(market)?  
 .ok\_or\_else(|| pyo3::exceptions::PyKeyError::new\_err(format!("Missing watchlist for market: {}", market\_str)))?  
 .extract()?;  
  
 let mut broker\_map = HashMap::new();  
 for (broker, adapter) in brokers\_dict.iter() {  
 broker\_map.insert(broker.extract()?, adapter.to\_object(py));  
 }  
  
 market\_map.insert(market\_str, (broker\_map, symbols));  
 }  
  
 Ok(market\_map)  
 })?;  
  
 if !offline\_mode {  
 Python::with\_gil(|py| -> PyResult<()> {  
 for (market, (brokers, symbols)) in market\_map {  
 for (broker, adapter\_obj) in brokers {  
 let adapter = adapter\_obj.bind(py);  
 let subscribe = adapter.getattr("subscribe\_to\_ticks")?.call1((symbols.clone(),))?;  
 pyo3\_asyncio\_0\_21::tokio::into\_future(subscribe)?.await?;  
 println!("Subscribed to market: {}, broker: {}", market, broker);  
 }  
 }  
 Ok(())  
 })?;  
 } else {  
 println!("Offline mode: Skipping subscriptions");  
 }  
 println!("MarketDataEngine started");  
 Ok(())  
 })  
 }  
  
 fn stop<'py>(&self, py: Python<'py>) -> PyResult<Bound<'py, PyAny>> {  
 let adapters = self.adapters.clone();  
  
 pyo3\_asyncio\_0\_21::tokio::local\_future\_into\_py(py, async move {  
 Python::with\_gil(|py| -> PyResult<()> {  
 let adapters\_dict = adapters.extract::<&PyDict>(py)?;  
 for (\_, brokers) in adapters\_dict.iter() {  
 let brokers\_dict = brokers.extract::<&PyDict>()?;  
 for (\_, adapter) in brokers\_dict.iter() {  
 let disconnect = adapter.getattr("disconnect")?.call0()?;  
 pyo3\_asyncio\_0\_21::tokio::into\_future(disconnect)?.await?;  
 }  
 }  
 Ok(())  
 })?;  
 println!("MarketDataEngine stopped");  
 Ok(())  
 })  
 }  
}  
  
#[pymodule]  
fn market\_data\_engine(py: Python, m: &Bound<'\_, PyModule>) -> PyResult<()> {  
 m.add\_class::<MarketDataEngine>()?;  
 Ok(())  
}

## tests\test\_adapters.py

from src.py.util.config\_loader import load\_config  
from src.py.core.adapters import get\_adapters  
import pytest  
import asyncio  
import yaml  
  
@pytest.mark.asyncio  
async def test\_multi\_broker\_initialization():  
 config = load\_config("config/config.yaml")  
 adapters = get\_adapters(config)  
 assert 'india' in adapters  
 assert 'usa' in adapters  
 assert 'angelone' in adapters['india']  
 assert 'interactive\_brokers' in adapters['usa']  
 assert len(adapters['india']['angelone'].mappings) > 0  
 assert len(adapters['usa']['interactive\_brokers'].mappings) > 0  
  
 # Load watchlists  
 with open(config['global']['markets']['india']['watchlists']['meanhunter'], 'r') as f:  
 india\_watchlist = [stock['tradingsymbol'] for stock in yaml.safe\_load(f)['stocks']]  
 with open(config['global']['markets']['usa']['watchlists']['xyzstrategy'], 'r') as f:  
 usa\_watchlist = [stock['tradingsymbol'] for stock in yaml.safe\_load(f)['stocks']]  
  
 if config['global']['offline\_mode']:  
 await adapters['india']['angelone'].connect()  
 await adapters['usa']['interactive\_brokers'].connect()  
 await adapters['india']['angelone'].subscribe\_to\_ticks(india\_watchlist)  
 await adapters['usa']['interactive\_brokers'].subscribe\_to\_ticks(usa\_watchlist)  
 assert adapters['india']['angelone'].authenticated

## tests\test\_angelone.py

import pytest  
import asyncio  
from src.py.util.config\_loader import load\_config  
from src.py.core.adapters.angelone import AngelOneAdapter  
from src.py.messaging.kafka\_client import KafkaClient  
import yaml  
from datetime import datetime  
import pandas as pd  
from loguru import logger  
  
  
@pytest.mark.asyncio  
async def test\_angelone\_historical\_data():  
 config = load\_config("config/config.yaml")  
 config['global']['offline\_mode'] = True  
 credentials = load\_config(config["global"]["markets"]["india"]["brokers"]["angelone"]["credentials"])  
 mappings = load\_config(config["global"]["markets"]["india"]["brokers"]["angelone"]["symbols"])  
 adapter = AngelOneAdapter(credentials, mappings, config["global"]["kafka"], config)  
  
 symbol = "RELIANCE-EQ"  
 timeframe = "1min"  
 to\_date = datetime.now()  
 from\_date = to\_date - pd.Timedelta(days=7)  
  
 data = await adapter.fetch\_historical\_data(symbol, timeframe, from\_date, to\_date)  
 assert len(data) >= 20, f"Insufficient historical data for {symbol}:{timeframe}"  
 assert data[0]["exchange"] == "NSE", f"Unexpected exchange for {symbol}"  
 assert data[0]["tradingsymbol"] == symbol, f"Unexpected tradingsymbol for {symbol}"  
 logger.info("Historical data test passed", market="nse", symbol=symbol)  
  
  
@pytest.mark.asyncio  
async def test\_angelone\_ticks():  
 config = load\_config("config/config.yaml")  
 offline\_mode = config['global']['offline\_mode']  
 credentials = load\_config(config["global"]["markets"]["india"]["brokers"]["angelone"]["credentials"])  
 mappings = load\_config(config["global"]["markets"]["india"]["brokers"]["angelone"]["symbols"])  
 adapter = AngelOneAdapter(credentials, mappings, config["global"]["kafka"], config)  
  
 watchlist = [stock['tradingsymbol'] for stock in  
 yaml.safe\_load(open(config['global']['markets']['india']['watchlists']['master']))['stocks']][:10]  
  
 await adapter.connect()  
 await adapter.subscribe\_to\_ticks(watchlist)  
 await asyncio.sleep(10 if not offline\_mode else 2) # Longer wait for online mode  
  
 if not offline\_mode:  
 # Verify Kafka messages  
 kafka\_client = KafkaClient(config["global"]["kafka"])  
 kafka\_client.subscribe(["nse\_ticks"])  
 ticks\_received = 0  
 for \_ in range(10):  
 msg = kafka\_client.poll(timeout=2.0)  
 if msg:  
 ticks\_received += 1  
 assert msg['topic'] == "nse\_ticks", f"Unexpected topic: {msg['topic']}"  
 assert msg['key'] in watchlist, f"Unexpected key: {msg['key']}"  
 assert msg['value']['exchange'] == "NSE", f"Unexpected exchange: {msg['value']['exchange']}"  
 assert 'ltp' in msg['value'], "Missing LTP in tick data"  
 assert 'timestamp' in msg['value'], "Missing timestamp in tick data"  
 assert ticks\_received > 0, "No ticks received in nse\_ticks"  
 kafka\_client.close()  
  
 await adapter.unsubscribe\_from\_ticks(watchlist)  
 await adapter.disconnect()  
 logger.info("Tick subscription test passed", market="nse", symbol="none")

## tests\test\_anquant.py

# D:\AlphaNivesh\ANQuant\tests\test\_anquant.py  
import asyncio  
import pytest  
from src.anquant.py.util.config\_loader import load\_config  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.messaging.kafka\_client import KafkaClient  
from anquant.py.data\_management.market\_data import MarketDataEngine  
from anquant.py.indicators.indicator\_engine import IndicatorEngine  
from src.anquant.py.core.flexirule.strategy\_engine import StrategyEngine  
  
  
@pytest.mark.asyncio  
async def test\_anquant\_initialization():  
 config = load\_config("config/config.yaml")  
 config["global"]["offline\_mode"] = True  
 redis\_client = RedisClient(config["global"]["redis"])  
 await redis\_client.connect()  
 adapters = {'india': {'angelone': {'symbols': ['RELIANCE-EQ', 'SBIN-EQ']}}}  
 watchlists = {'india': {'meanhunter': ['RELIANCE-EQ', 'SBIN-EQ']}}  
  
 components = [  
 MarketDataEngine(config, redis\_client, adapters, watchlists),  
 IndicatorEngine(config, redis\_client, adapters, watchlists),  
 StrategyEngine(config, redis\_client)  
 ]  
  
 await asyncio.gather(\*(component.initialize() for component in components))  
  
 kafka\_client = KafkaClient(config["global"]["kafka"])  
 message = {  
 "tradingsymbol": "RELIANCE-EQ",  
 "timestamp": "2025-07-19T20:10:00+05:30",  
 "open": 100.0,  
 "high": 102.0,  
 "low": 98.0,  
 "close": 100.0,  
 "volume": 3000.0,  
 "market": "india"  
 }  
 kafka\_client.produce("ohlcv\_5min", "RELIANCE-EQ", message)  
  
 log\_files = [  
 "logs/market\_data/market\_data\_engine\_2025-07-19.log",  
 "logs/indicators/indicator\_engine\_2025-07-19.log",  
 "logs/strategy/strategy\_engine\_2025-07-19.log"  
 ]  
 for log\_file in log\_files:  
 assert os.path.exists(log\_file)  
 with open(log\_file, "r") as f:  
 assert "initialized successfully" in f.read().lower()  
  
 await asyncio.gather(\*(component.stop() for component in components))  
 await redis\_client.close()

## tests\test\_config\_loader.py

# tests/test\_config\_loader.py  
from src.py.anquant.util.config\_loader import load\_config, load\_symbol\_mappings, load\_credentials  
  
  
def test\_config\_loader():  
 config = load\_config("config/config.yaml")  
 assert 'global' in config  
 assert 'markets' in config['global']  
 assert 'india' in config['global']['markets']  
  
 mappings = load\_symbol\_mappings(config, 'india', 'angelone')  
 assert isinstance(mappings, dict)  
  
 credentials = load\_credentials(config, 'india', 'angelone')  
 assert isinstance(credentials, dict)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 test\_config\_loader()

## tests\test\_create\_kafka\_topics.py

import pytest  
import os  
from scripts.create\_kafka\_topics import create\_kafka\_topics  
from src.py.util.config\_loader import load\_config  
from confluent\_kafka.admin import AdminClient  
  
  
def test\_create\_kafka\_topics():  
 config = load\_config("config/config.yaml")  
 create\_kafka\_topics(config)  
  
 admin\_client = AdminClient({'bootstrap.servers': config['global']['kafka']['brokers']})  
 topics\_metadata = admin\_client.list\_topics(timeout=10).topics  
  
 expected\_topics = [  
 config['global']['kafka']['topics']['india'],  
 config['global']['kafka']['topics']['ohlcv\_1min'],  
 config['global']['kafka']['topics']['ohlcv\_5min'],  
 config['global']['kafka']['topics']['ohlcv\_30min'],  
 config['global']['kafka']['topics']['signals'],  
 config['global']['kafka']['topics']['trades']  
 ]  
  
 for topic in expected\_topics:  
 assert topic in topics\_metadata, f"Topic {topic} not created"  
  
 # Verify partition counts (tolerate existing topics with different counts)  
 for topic in expected\_topics:  
 partition\_key = f"{topic}\_partitions"  
 expected\_partitions = config['global']['kafka'].get(partition\_key, 4)  
 current\_partitions = len(topics\_metadata[topic].partitions)  
 if current\_partitions != expected\_partitions:  
 print(f"Warning: Topic {topic} has {current\_partitions} partitions, expected {expected\_partitions}")  
  
 # Verify log file contains topic names  
 log\_file = "logs/kafka/create\_topics.log"  
 assert os.path.exists(log\_file), f"Log file {log\_file} not found"  
 with open(log\_file, "r") as f:  
 log\_content = f.read()  
 for topic in expected\_topics:  
 assert topic in log\_content, f"Topic {topic} not found in logs"

## tests\test\_flexirule.py

# tests/test\_flexirule.py  
import asyncio  
import pytest  
import json  
import os  
from src.anquant.py.util.config\_loader import load\_config  
from src.anquant.py.messaging.redis\_client import RedisClient  
from src.anquant.py.core.flexirule.strategy\_engine import StrategyEngine  
from src.anquant.py.core.flexirule.validator import StrategyValidator  
  
@pytest.mark.asyncio  
async def test\_strategy\_validation():  
 config = load\_config("config/config.yaml")  
 validator = StrategyValidator(config)  
 strategy\_path = "config/markets/india/strategies/meanhunter\_strategy.yaml"  
 assert validator.validate\_strategy(strategy\_path), "Strategy validation failed"  
 log\_file = "logs/strategy/flexirule\_validator\_2025-07-19.log"  
 assert os.path.exists(log\_file)  
 with open(log\_file, 'r') as f:  
 assert "Successfully validated strategy: meanhunter" in f.read()  
  
@pytest.mark.asyncio  
async def test\_rule\_evaluation():  
 config = load\_config("config/config.yaml")  
 config["global"]["offline\_mode"] = True  
 redis\_client = RedisClient(config["global"]["redis"])  
 await redis\_client.connect()  
 strategy\_config = {  
 "name": "meanhunter",  
 "timeframe": "5min",  
 "watchlist": {"india": "config/markets/india/watchlists/meanhunter.yaml"},  
 "threshold": 0.75,  
 "indicators": [  
 {"name": "bbands", "type": "bollinger\_bands", "period": 20, "std": 2.0},  
 {"name": "rsi", "type": "rsi", "period": 14}  
 ],  
 "entry\_rules": [  
 {"condition": "close < bb\_lower", "weight": 0.6},  
 {"condition": "rsi < 30", "weight": 0.2},  
 {"condition": "volume > volume\_threshold", "weight": 0.2}  
 ],  
 "exit\_rules": [  
 {"condition": "close > bb\_upper", "weight": 0.8}  
 ],  
 "stop\_loss": {  
 "type": "multi",  
 "rules": [{"type": "fixed", "value": "2%", "id": "fixed"}]  
 },  
 "target": {  
 "type": "multi",  
 "rules": [{"type": "fixed", "value": "5%", "partial\_exit": "50%", "id": "partial\_1"}]  
 },  
 "trade\_management": {  
 "breakeven": {"trigger": 2.0}  
 },  
 "market\_params": {  
 "india": {"volume\_threshold": "avg\_volume\_20 \* 1.5"}  
 }  
 }  
 strategy\_engine = StrategyEngine(config, redis\_client)  
 strategy\_engine.strategies = [strategy\_config]  
 strategy\_engine.rule\_engines = {strategy\_config['name']: RuleEngine(strategy\_config, redis\_client)}  
 await strategy\_engine.initialize()  
  
 ohlcv = {  
 "tradingsymbol": "RELIANCE-EQ",  
 "timestamp": "2025-07-19T18:42:00+05:30",  
 "close": 100.0,  
 "volume": 3000.0,  
 "market": "india"  
 }  
 indicators = {  
 "bb\_lower": 110.0,  
 "bb\_upper": 120.0,  
 "rsi": 25.0,  
 "avg\_volume\_20": 1500.0  
 }  
 signal = await strategy\_engine.rule\_engines["meanhunter"].evaluate("RELIANCE-EQ", ohlcv, indicators)  
 assert signal == "BUY", f"Expected BUY signal, got {signal}"  
  
 # Simulate partial target  
 ohlcv['close'] = 106.0 # Above 5% target (105.0)  
 signal = await strategy\_engine.rule\_engines["meanhunter"].evaluate("RELIANCE-EQ", ohlcv, indicators)  
 assert signal == "PARTIAL\_SELL:50:partial\_1", f"Expected PARTIAL\_SELL signal, got {signal}"  
  
 # Verify audit trail  
 audit\_data = await redis\_client.redis.get(f"signals:audit:meanhunter")  
 assert audit\_data, "Audit trail not found"  
 audit\_data = json.loads(audit\_data)  
 assert audit\_data['symbol'] == "RELIANCE-EQ"  
 assert audit\_data['signal'] in ["BUY", "PARTIAL\_SELL:50:partial\_1"]  
 assert audit\_data['reason'] in ["Entry rule triggered", "Target triggered (PARTIAL\_SELL:50:partial\_1)"]  
  
 await strategy\_engine.stop()  
 log\_file = "logs/strategy/rule\_engine\_2025-07-19.log"  
 assert os.path.exists(log\_file)  
 with open(log\_file, 'r') as f:  
 assert "RuleEngine for meanhunter initialized successfully" in f.read()

## tests\test\_initialization.py

import pytest  
import asyncio  
from src.py.main import main  
from src.py.util.config\_loader import load\_config  
  
@pytest.mark.asyncio  
async def test\_component\_initialization():  
 config = load\_config("config/config.yaml")  
 config["global"]["offline\_mode"] = True  
 task = asyncio.create\_task(main())  
 await asyncio.sleep(1)  
 task.cancel()  
 try:  
 await task  
 except asyncio.CancelledError:  
 pass  
 # Check logs/general/main.log for initialization messages

## tests\test\_kafka\_client.py

# tests/test\_kafka\_client.py  
import pytest  
import asyncio  
import json  
from src.anquant.py.util.config\_loader import load\_config  
from src.anquant.py.messaging.kafka\_client import KafkaClient  
  
  
@pytest.mark.asyncio  
async def test\_kafka\_client():  
 config = load\_config("config/config.yaml")  
 kafka\_client = KafkaClient(config["global"]["kafka"])  
  
 # Subscribe to ohlcv\_5min  
 topics = ["ohlcv\_5min"]  
 kafka\_client.subscribe(topics)  
  
 # Produce a test message  
 message = {  
 "tradingsymbol": "RELIANCE-EQ",  
 "timestamp": "2025-07-19T19:30:00+05:30",  
 "open": 100.0,  
 "high": 102.0,  
 "low": 98.0,  
 "close": 100.0,  
 "volume": 3000.0,  
 "market": "india"  
 }  
 kafka\_client.produce("ohlcv\_5min", "RELIANCE-EQ", message)  
  
 # Poll for the message  
 result = kafka\_client.poll(timeout=2.0)  
 assert result is not None, "Failed to consume message"  
 assert result["topic"] == "ohlcv\_5min"  
 assert result["key"] == "RELIANCE-EQ"  
 assert result["value"]["tradingsymbol"] == "RELIANCE-EQ"  
 assert result["value"]["close"] == 100.0  
  
 # Verify logs  
 log\_file = "logs/messaging/kafka\_client\_2025-07-19.log"  
 assert os.path.exists(log\_file)  
 with open(log\_file, "r") as f:  
 log\_content = f.read()  
 assert "Initialized KafkaClient" in log\_content  
 assert "Subscribed to topics: ['ohlcv\_5min']" in log\_content  
 assert "Delivered message to partition" in log\_content  
  
 kafka\_client.close()

## tests\test\_main.py

import pytest  
import asyncio  
from src.py.main import main, load\_watchlist  
from src.py.util.config\_loader import load\_config  
  
@pytest.mark.asyncio  
async def test\_main\_initialization():  
 config = load\_config("config/config.yaml")  
 watchlists = {}  
 for market in config['global']['brokers']['active\_brokers']:  
 watchlist\_path = config['global']['markets'][market]['watchlists']['meanhunter' if market == 'india' else 'xyzstrategy']  
 watchlists[market] = await load\_watchlist(watchlist\_path)  
 assert 'india' in watchlists  
 assert 'usa' in watchlists  
 assert len(watchlists['india']) > 0  
 assert len(watchlists['usa']) > 0  
 # Run main briefly in offline mode  
 config['global']['offline\_mode'] = True  
 task = asyncio.create\_task(main())  
 await asyncio.sleep(1)  
 task.cancel()  
 try:  
 await task  
 except asyncio.CancelledError:  
 pass

## tests\test\_market\_data\_engine.py

# tests/test\_market\_data\_engine.py  
import asyncio  
  
from anquant.py.core.adapters import AngelOneAdapter  
from src.anquant.py.util.config\_loader import load\_config  
from src.anquant.py.messaging.redis\_client import RedisClient  
from anquant.py.data\_management.market\_data import MarketDataEngine  
  
async def test\_market\_data\_engine():  
 config = load\_config("config/config.yaml")  
 redis\_client = RedisClient(config["global"]["redis"])  
 await redis\_client.connect()  
 adapters = {'india': {'angelone': AngelOneAdapter(config["global"]["markets"]["india"]["brokers"]["angelone"])}}  
 market\_data\_engine = MarketDataEngine(config, redis\_client, adapters)  
 await market\_data\_engine.initialize()  
 await market\_data\_engine.start()  
 await asyncio.sleep(5)  
 await market\_data\_engine.stop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 asyncio.run(test\_market\_data\_engine())

## tests\test\_prefetch\_historical\_data.py

import pytest  
import asyncio  
import os  
import yaml  
from loguru import logger  
from src.py.util.config\_loader import load\_config  
from src.py.messaging.redis\_client import RedisClient  
from src.anquant.py.util.database import Database  
from scripts.prefetch\_historical\_data import prefetch\_historical\_data  
  
# Remove existing handlers and configure logging  
logger.remove()  
logger.add("logs/general/test\_prefetch\_historical\_data.log", rotation="10 MB", retention="7 days", level="DEBUG",  
 format="[{time:YYYY-MM-DD HH:mm:ss}] [{level}] {message}")  
  
  
@pytest.mark.asyncio  
async def test\_prefetch\_historical\_data():  
 config = load\_config("config/config.yaml")  
 config['global']['offline\_mode'] = True  
 await prefetch\_historical\_data(config)  
  
 redis\_client = RedisClient(config["global"]["redis"])  
 database = Database(config["global"]["database"])  
 watchlist\_path = os.path.join(os.path.dirname(\_\_file\_\_), "../config/markets/india/watchlists/master.yaml")  
  
 try:  
 with open(watchlist\_path, 'r') as f:  
 watchlist = [stock['tradingsymbol'] for stock in yaml.safe\_load(f)['stocks']][:10]  
 except FileNotFoundError as e:  
 logger.error(f"Failed to load watchlist: {e}")  
 pytest.fail(f"Failed to load watchlist: {e}")  
  
 for symbol in watchlist:  
 for timeframe in config["global"]["historical\_data"]["timeframes"]:  
 cached = await redis\_client.get(f"{symbol}:ohlcv:{timeframe}")  
 assert cached is not None, f"No Redis cache for {symbol}:{timeframe}"  
 assert len(cached) >= 60, f"Insufficient Redis records for {symbol}:{timeframe}"  
 cursor = database.conn.cursor()  
 cursor.execute("SELECT COUNT(\*) FROM anquant.ohlcv WHERE tradingsymbol = %s AND timeframe = %s",  
 (symbol, timeframe))  
 count = cursor.fetchone()[0]  
 assert count >= 60, f"Insufficient PostgreSQL records for {symbol}:{timeframe} (found {count})"  
 cursor.close()  
 database.close()  
 await redis\_client.close()

## tests\\_\_init\_\_.py