binance_interface.py

```
# ====================== Imports ======================
# Library imports
import requests
import pydoc
import pandas as pd
import asyncio

# Local imports
from classes.interface_classes import Interface, DataStore, SymbolsManagerBase

# Binance imports
from binance.spot import Spot as Client
from binance.websocket.spot.websocket_api import SpotWebsocketAPIClient


# ====================== Classes ======================

class BinanceAPI(Interface):
    def __init__(self, access_key, secret_key, host="api.binance.com"):
        super().__init__(access_key, secret_key, host)
        self.__access_key = access_key
        self.__secret_key = secret_key
        self.__host = host

    def __get(self, path, params=None):
        return requests.get("https://{host}{path}".format(host=self.__host, path=path),
params=params).json()

    def get_symbols(self):
        return self.__get("/api/v3/exchangeInfo")["symbols"]

    def get_kline_history(self, symbol, interval, limit):
        return self.__get("/api/v3/klines", {"symbol": symbol, "interval": interval,
"limit": limit})

    def subscribe_to_candlestick(self, symbol="btcusdt", interval="1m",
callback_func=None):
        def callback(_, kline_data):
            print(kline_data)
            print("\n")
        if (callback_func == None):
            callback_func = callback

        async def subscribe():
            ws_client = SpotWebsocketAPIClient(on_message=callback_func)
            ws_client.klines(symbol=symbol, interval=interval, limit=1) # only get the last
candlestick

        async_loop = asyncio.new_event_loop()
        asyncio.set_event_loop(async_loop)
        async_loop.run_until_complete(subscribe())

    def request_trades(self, symbol="btcusdt", callback_func=None):
        pass
```

```python
class BinanceSymbolsManager(SymbolsManagerBase):
    def __init__(self, interface: Interface):
        self.interface = interface

    def convert_to_dataframe(self, symbols: list):
        df = pd.DataFrame().from_dict(symbols)
        return df

    def filter_excluded(self, symbols: pd.DataFrame, excluded_coins: list = ["BTCUSDT",
"ETHUSDT"]):
        # Filter out Ethereum and Bitcoin
        symbols = symbols[~symbols['baseAsset'].isin(excluded_coins)]
        symbols = symbols[~symbols['quoteAsset'].isin(excluded_coins)]
        return symbols
        #return super().filter_excluded(symbols, excluded_coins)

    def filter_offline(self, symbols: pd.DataFrame):
        # Only include symbols that are 'TRADING'
        symbols = symbols[symbols['status'] == 'TRADING']
        return symbols
        #return super().filter_offline(symbols)

    def convert_to_list(self, symbols: pd.DataFrame):
        return symbols['symbol'].tolist()
```