In [ ]:

```python
from fastapi import APIRouter
from pydantic import BaseSettings
import aiohttp
import asyncio
from tenacity import retry, stop_after_attempt
from datetime import datetime, timedelta
class CloseConfig(BaseSettings):
    close_api_key: str
    class Config:
        env_file = ".env"

router = APIRouter(prefix="/close")
config = CloseConfig()
# max concurrent requests to the API

sem = asyncio.Semaphore(35)
base_url = "https://api.close.com/api/v1"
days_of_data = 10
starting_date = datetime.now() - timedelta(days=days_of_data)
tomorrow = datetime.now() + timedelta(days=1)
datetime_format = "%Y-%m-%dT%H:%M:%S"
date_format = "%Y-%m-%d"
def format_date(d):
    return datetime.strftime(d, date_format)

def format_datetime(d):
    return datetime.strftime(d, datetime_format)
# This route returns data for today with no input, or takes a date in YYYY-MM-DD for

@router.get("/activities")
async def list_activities():
    day = starting_date
    futures = []
    # Generate a list of days to cycle through in the date range
    while day < tomorrow:
        futures.append(
            get_activities(
                {
                    "day": format_date(day),
                    "start_date": format_datetime(day),
                    "end_date": format_datetime(day + timedelta(days=1)),
                }
            )
        )
        day += timedelta(days=1)
    return await wait_flatten_futures(futures)
# Get all of the specified activities for a specific day.

async def get_activities(day):
    print(f"Getting all data for {day['day']}...")
    activity_fields = "user_id,date_created,_type,direction,duration"
    offset = 0
    activities = []
    while True:
        activities_for_day = await get_basic(
            "/activity",
            params={
                "_skip": offset,
                "date_created__gte": day["start_date"],
                "date_created__lt": day["end_date"],
                "_fields": activity_fields,
            },
        )
        activities += activities_for_day["data"]
        offset += len(activities_for_day["data"])
        if not activities_for_day["has_more"]:
            break
    return activities

@router.get("/opportunities")
async def list_opportunities():
    return await get_pages(
        "/opportunity",
        params={
            "date_updated__gte": format_date(starting_date),
            "_order_by": "-date_updated",
        },
        page_size=250,
    )
# Lists members in groups.

@router.get("/groupMembers")
async def list_group_members():
    fields = {"_fields": "id,name,members"}
    acquisitions = await get_basic(
        "/group/group_5H4vOzrg9gfeFqQFmsksuR/", params=fields
    )
    sales = await get_basic("/group/group_0psprITjQlfVH5qltMJqsj/", params=fields)
    return [acquisitions, sales]
# Lists users

@router.get("/users")
async def list_users():
    return await get_pages("/user")
# get a simple endpoint

@retry(stop=stop_after_attempt(10))
async def get_basic(route, params={}):
    async with sem:
        print("Fetching for", route, params)
        async with aiohttp.ClientSession(
            auth=aiohttp.BasicAuth(login=config.close_api_key, password=""),
            headers={"Accept": "application/json"},
        ) as session:
            query_string = "?" + "&".join(
                f"{key}={value}" for key, value in params.items()
            )
            url = f"{base_url}/{route}{query_string}"
            async with session.get(url) as response:
                return await response.json()
# paginate through data for endpoint

async def get_pages(route, params={}, page_size=100):
    ls = []
    skip = 0
    page = 1
    while True:
        print(f"Fetching page #{page} for {route}")
        results = await get_basic(
            route, params={"_limit": page_size, "_skip": skip, **params}
        )
        ls += results["data"]
        if not results["has_more"]:
            print("No more data to fetch for", route)
            break
        page += 1
        skip += page_size
    return ls

async def wait_flatten_futures(futures):
    return [el for f in await asyncio.gather(*futures) for el in f]
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
c:\Users\Isaia\Projects\Closev2\home-equity-options-python-api-proxy\app\routers\clo
se.py in <module>
----> <a href='file:///c%3A/Users/Isaia/Projects/Closev2/home-equity-options-python-
api-proxy/app/routers/close.py?line=0'>1</a> from fastapi import APIRouter
      <a href='file:///c%3A/Users/Isaia/Projects/Closev2/home-equity-options-python-
api-proxy/app/routers/close.py?line=1'>2</a> from pydantic import BaseSettings
      <a href='file:///c%3A/Users/Isaia/Projects/Closev2/home-equity-options-python-
api-proxy/app/routers/close.py?line=2'>3</a> import aiohttp
      <a href='file:///c%3A/Users/Isaia/Projects/Closev2/home-equity-options-python-
api-proxy/app/routers/close.py?line=3'>4</a> import asyncio
      <a href='file:///c%3A/Users/Isaia/Projects/Closev2/home-equity-options-python-
api-proxy/app/routers/close.py?line=4'>5</a> from tenacity import retry, stop_after_
attempt

ModuleNotFoundError: No module named 'fastapi'
```