README.md

fastapi

Implemented backend endpoints using fastapi, pydantic, fastapi-pagination.

*Implemented listing trades with support for **pagination** and **sorting**, **searching** trade, and advanced **filtering***

File Structure

```
.
├── main.py                    # contains endpoints implementation
├── Schema.py                  # contains classes for model as given in assignme
├── Data.py                    # Data as a list of dictionaries
```

Code Logic

- Data is stored in `Data.py` file as list of dictionaries `TRADES`

Understanding main.py

```
DEFUALT_OFFSET = 0
DEFUALT_LIMIT = 10
```

The `DEFUALT_LIMIT` sets the maximum limit to the number of `Trades` that will be listed The `DEFUALT_OFFSET` sets the offset for listing `Trades`

```
@app.get("/", status_code=200)
async def Home():
    return {"msg" : "server running"}
```

Ihis is for testing purpose whether `app` is running or not

Listing Trades and Advanced filtering with sorting and pagination

```
@app.get("/AllTrades/", status_code=200, response_model=Page[Trade])
async def AllTrades(sort: str = Query(None, description="sorting criteria"),
                    sort_order: Optional[bool] = False,
                    assetClass: Optional[str] = None,
                    end: Optional[dt.datetime] = None,
                    maxPrice: Optional[float]= None,
                    minPrice: Optional[float] = None,
                    start: Optional[dt.datetime] = None,
                    tradeType: Optional[str] = None,
                    offset: Optional[int] = DEFUALT_OFFSET,
                    limit: Optional[int] = DEFUALT_LIMIT):
```

The `sort` optional parameter is for field on basis on which sorting will be performed in `sort_order`. `sort_order == False` implies ascending order and negation otherwise. `offest` and `limit` are used for pagination purpose, they will restrict the `Trade` list in `[offset, offset+limit]`

```
    opt_par = not(assetClass or end or maxPrice or minPrice or start or tradeType
    if opt_par:
        if sort=="assetClass" or sort=="counterparty":                    # han
            result = sorted(TRADES, key=lambda x: x[sort] or '', reverse=sort_ord
        elif sort=="price" or sort=="quantity":                           # han
            result = sorted(TRADES, key=lambda x: x["tradeDetails"][sort] or '',
        elif sort:
            result = sorted(TRADES, key=lambda x: x[sort], reverse=sort_order)
        else:
            result = TRADES

        if offset<len(result):
            count_trade = min(limit, len(result)-offset)
            if count_trade > 0:
                return paginate(result[offset : offset+count_trade])
        return paginate(result)
```

This part of code checks whether any filter is applied or not. And if non of the filters are applied `http://127.0.0.1:8000/AllTrades/` will list down all the trades after checking if sorting and limit is applied or not.

```
# filtering
result = []
if assetClass:
    result1 = list(filter(lambda trade: assetClass.lower() in trade["assetClass
    result = result1
if end:
    result1 = [trade for trade in TRADES if trade["tradeDateTime"] <= end]
    result = result + result1
```

```
    if maxPrice:
        result1 = [trade for trade in TRADES if trade["tradeDetails"]["price"] <= r
        result = result + result1
    if minPrice:
        result1 = [trade for trade in TRADES if trade["tradeDetails"]["price"] >= r
        result = result + result1
    if start:
        result1 = [trade for trade in TRADES if trade["tradeDateTime"] >= start]
        result = result + result1
    if tradeType:
        result1 = [trade for trade in TRADES if trade["tradeDetails"]["buySellIndic
        result = result + result1
```

If any of the filters are applied, get `union` of all `Trades` satisfying filtering condition

```
    # sorting
    if sort=="assetClass" or sort=="counterparty":
        result = sorted(result, key=lambda x: x[sort] or '', reverse=sort_order
    elif sort=="price" or sort=="quantity":
        result = sorted(TRADES, key=lambda x: x["tradeDetails"][sort] or '', re
    elif sort:
        result = sorted(result, key=lambda x: x[sort], reverse=sort_order)
```

As `assetClass` and `counterparty` are optional fields and their value can be `string` or `None`. So did sorting by considering `null` as `empty string`.

```
    # pagination
    if offset<len(result):
        count_trade = min(limit, len(result)-offset)
        if count_trade > 0:
            return paginate(result[offset : offset+count_trade])

    return paginate(result)
```

Check `offset` and `limit` bound. Finally returned list of `Trades`

Fetch Trade with trade_id

```
@app.get("/Trade/{trade_id}", status_code=200, response_model=Trade)
async def retrieveTrade(trade_id):
    result = [trade for trade in TRADES if trade["tradeId"] == trade_id]
    if result:
        return result[0]
```

If trade with `trade_id` as its `trade_id` is found in database return the found trade

Searching Trade

```
@app.get("/search/", status_code=200, response_model=Page[Trade])
async def searchTrade(search):
    result = list(filter(lambda trade: search.lower() in trade["instrumentId"].
        + trade["instrumentName"].lower() + trade["trader"].lower(), TRADES))
    temp = [trade for trade in TRADES if isinstance(trade["counterparty"], str)
        search.lower() in trade["counterparty"].lower()]
    result = result + temp
    return paginate(result)
```

The search implemented here is non case-sensitive (converted data to lower-case) and based on substring matching. As `counterparty` can be `None` so handled it separateing using isinstance of string. Finally returned union of all matched `Trade`

Running the code

Used `uvicorn` library. In the same folder as of `main.py` execute

```
uvicorn main:app --reload
```

It will start application at `http://127.0.0.1:8000`

Refrences used for Assignment

- documentation of fastapi with pydantic
- documentation of fastapi-pagination