API Developer Assessment

SteelEye

Deployed Link - https://steeleye-backend.onrender.com/docs

Github Link - **https://github.com/Vivek1898/Vivek_Singh_Backend**

Submitted by

Vivek Singh
12007178
viveksingh27795@gmail.com
Lovely professional university

Approach

The solution is a simple RESTful API implemented using FastAPI that provides **CRUD** (Create, Read, Update, Delete) functionality for trades. It uses a **dummy data list trades_db** to store the trades instead of a database for simplicity.

The API has four endpoints:

1. **GET /trades:** Returns a list of trades **filtered by search term, asset class, trade date range, price range, and trade type**.
2. **GET /trades/{trade_id}:** Returns a **single trade** identified by its ID.
3. **POST /trades:** Creates a new trade.
4. **PUT /trades/{trade_id}:** Updates an existing trade identified by its ID.
5. **DELETE /trades/{trade_id}:** Deletes an existing trade identified by its ID.

The API uses **Pydantic** to define the **data model** for the Trade object. Pydantic is a **data validation** library that allows us to specify constraints and data types for our data. The Trade object contains several fields, including assetClass, counterparty, instrumentId, instrumentName, tradeDateTime, tradeDetails, tradeId, and trader.

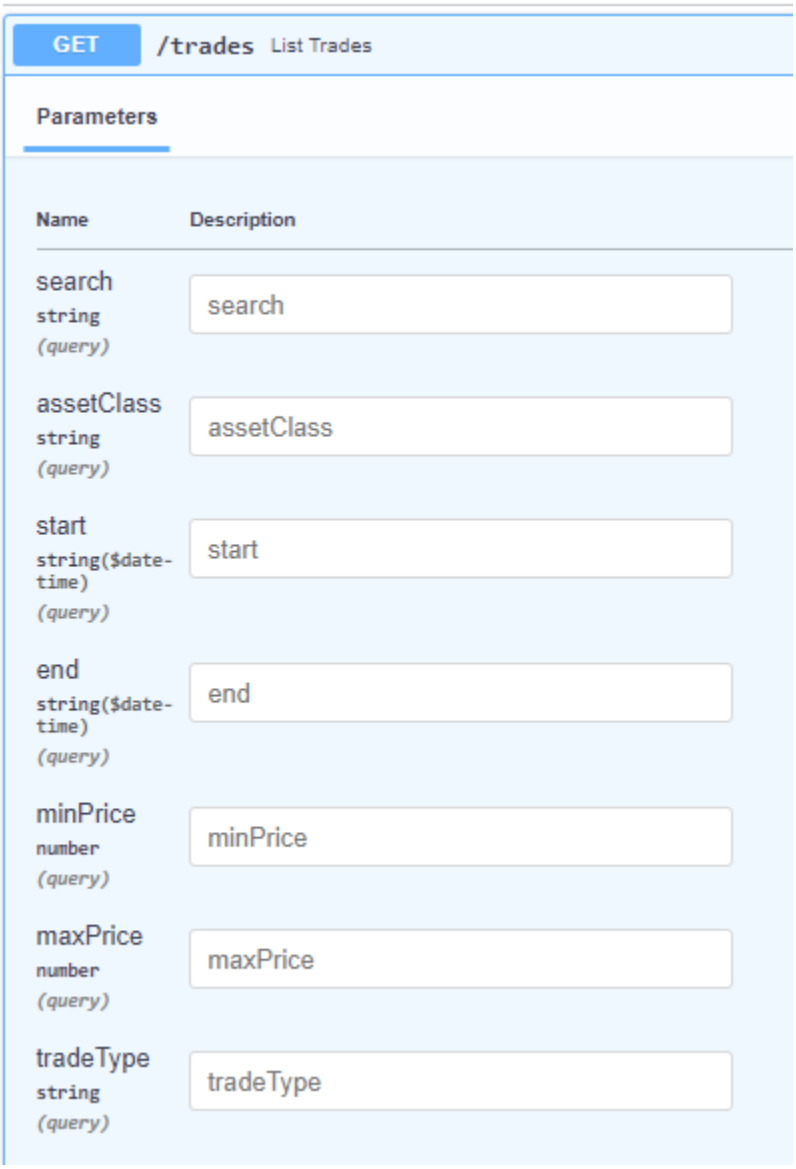
The API **implementation** is straightforward and easy to follow.

1. **list_trades()** function **filters** the list of trades based on the provided parameters, such as **asset class, date range, price range, and trade type.**
2. **get_trade_by_id()** function returns a single trade identified by its ID.
3. **create_trade()** function creates a new trade and appends it to the in-memory list.
4. **update_trade()** function updates an existing trade identified by its ID. Finally,
5. **delete_trade()** function deletes an existing trade identified by its ID.

Overall, the provided code is a simple and effective implementation of a trading data API using **FastAPI and Pydantic**.

Advanced filtering

The users would now like the ability to filter trades. Your endpoint for fetching a list of trades will need to support filtering using the following optional query parameters:



CODE-

```
from fastapi import FastAPI, HTTPException
from typing import List, Optional
import datetime as dt
from pydantic import BaseModel, Field
import uuid

app = FastAPI()

# Dummy data to be used in place of a database
trades_db = [
    {
        "id": 1,
        "assetClass": "Equity",
        "counterparty": "Goldman Sachs",
        "instrumentId": "AAPL",
        "instrumentName": "Apple Inc.",
        "tradeDateTime": "2022-04-14T10:00:00",
        "tradeDetails": {
            "buySellIndicator": "BUY",
            "price": 155.0,
            "quantity": 100
        },
        "trader": "John Doe"
    },
    {
        "id": 2,
        "assetClass": "Bond",
        "counterparty": "JP Morgan",
        "instrumentId": "GOOGL",
        "instrumentName": "Alphabet Inc.",
        "tradeDateTime": "2022-04-14T11:00:00",
        "tradeDetails": {
            "buySellIndicator": "SELL",
            "price": 600.0,
            "quantity": 50
        },
        "trader": "Jane Doe"
    }
```

```
]


# Pydantic model representing a single Trade
class TradeDetails(BaseModel):
    buySellIndicator: str = Field(description="A value of BUY for buys,
SELL for sells.")
    price: float = Field(description="The price of the Trade.")
    quantity: int = Field(description="The amount of units traded.")


class Trade(BaseModel):
    assetClass: Optional[str] = Field(alias="assetClass", default=None,
description="The asset class of the instrument traded. E.g. Bond, Equity,
FX...etc")
    counterparty: Optional[str] = Field(default=None, description="The
counterparty the trade was executed with. May not always be available")
    instrumentId: str = Field(alias="instrumentId", description="The
ISIN/ID of the instrument traded. E.g. TSLA, AAPL, AMZN...etc")
    instrumentName: str = Field(alias="instrumentName", description="The
name of the instrument traded.")
    tradeDateTime: dt.datetime = Field(alias="tradeDateTime",
description="The date-time the Trade was executed")
    tradeDetails: TradeDetails = Field(alias="tradeDetails",
description="The details of the trade, i.e. price, quantity")
    tradeId: Optional[str] = Field(alias="tradeId", default=None,
description="The unique ID of the trade")
    trader: str = Field(description="The name of the Trader")




# Endpoint to fetch a list of trades
@app.get("/trades", response_model=List[Trade])
async def list_trades(
    search: Optional[str] = None,
    assetClass: Optional[str] = None,
    start: Optional[dt.datetime] = None,
```

```
    end: Optional[dt.datetime] = None,
    minPrice: Optional[float] = None,
    maxPrice: Optional[float] = None,
    tradeType: Optional[str] = None
) -> List[Trade]:
    result = trades_db

    # Search for trades by the provided search term
    if search:
        result = [trade for trade in result if search.lower() in
str(trade).lower()]

    # Filter trades by the provided parameters
    if assetClass:
        result = [trade for trade in result if trade["assetClass"] ==
assetClass]

    if start:
        result = [trade for trade in result if
dt.datetime.fromisoformat(trade["tradeDateTime"]) >= start]
    if end:
        result = [trade for trade in result if
dt.datetime.fromisoformat(trade["tradeDateTime"]) <= end]

    if minPrice:
        result = [trade for trade in result if
trade["tradeDetails"]["price"] >= minPrice]
    if maxPrice:
        result = [trade for trade in result if
trade["tradeDetails"]["price"] <= maxPrice]
    if tradeType:
        result = [trade for trade in result if
trade["tradeDetails"]["buySellIndicator"] == tradeType]

    return result

@app.get("/trades/{trade_id}", response_model=Trade)
async def get_trade_by_id(trade_id: str) -> Trade:
    for trade in trades_db:
        if trade["id"] == int(trade_id):
```

```
            return trade
    raise HTTPException(status_code=404, detail="Trade not found")

@app.post("/trades", response_model=Trade)
async def create_trade(trade: Trade) -> Trade:
    trade_dict = trade.dict()
    trade_dict["tradeId"] = str(uuid.uuid4())
    trades_db.append(trade_dict)
    return trade_dict

@app.put("/trades/{trade_id}", response_model=Trade)
async def update_trade(trade_id: str, trade: Trade) -> Trade:
    for t in trades_db:
        if t["id"] == int(trade_id):
            trades_db.remove(t)
            trades_db.append(trade.dict())
            return trade
    raise HTTPException(status_code=404, detail="Trade not found")

@app.delete("/trades/{trade_id}")
async def delete_trade(trade_id: str):
    for t in trades_db:
        if t["id"] == int(trade_id):
            trades_db.remove(t)
            return {"message": "Trade deleted successfully"}
    raise HTTPException(status_code=404, detail="Trade not found")
```

Resources

FastAPI: https://fastapi.tiangolo.com/
Pydantic: https://pydantic-docs.helpmanual.io/