Backend engineering challenge

Problem

In a wealth management platform, a customer's portfolio is made up of varying amounts of different assets; stocks, bonds and cash. A daily rebalancing process occurs to make sure each customer's portfolio matches:

- The current state of the financial market, e.g. are stocks going up or down in price
- The customer's risk level, e.g. does someone want to risk losing some money in order to potentially make a lot, or take fewer risks in exchange for lower gains
- The customer's financial goals, e.g. they are saving to buy a house in 5 years, or they are saving for retirement in 25 years

Your task is to build a service to automate this daily rebalancing.

Challenge

Your daily rebalancing service is provided 2 CSV files every day to process *customers.csv* and *strategy.csv*.

customers.csv contains customer data in the following format:

```
customerId,email,dateOfBirth,riskLevel,retirementAge
1,bob@bob.com,1961-04-29,3,65
2,sally@gmail.com,1978-05-01,8,67
```

strategy.csv contains a list of investment strategies, each detailing a different percentage of stocks, cash and bonds. Each customer on the platform is assigned at most one investment strategy* based on their risk level and years until retirement. For a customer to match a strategy their risk level and years to retirement must fall within both provided ranges (min to max, inclusive of both). Example data is shown below:

```
strategyId,minRiskLevel,maxRiskLevel,minYearsToRetirement,maxYearsToRetirement,stocksPercentage,cashPercentage,
bondsPercentage
1,0,3,20,30,20,20,60
2,0,3,10,20,10,20,70
3,6,9,20,30,10,0,90
```

* If a customer does not match any defined strategy then they should be assigned 100% cash.

As an example, bob@bob.com (customerId 1) is currently aged 51, and so will retire in 14 years. They therefore match strategyId 2 and should have the assets spread across 10% stocks, 20% cash and 70% bonds.

Sample CSV files are not provided. You may create your own using dummy data if required. The data does not have to make sense or be complete.

Financial Portfolio Service (FPS)

This is an HTTP + JSON external pre-existing system that holds the financial portfolios for each customer (i.e. how much cash, stocks and bonds they currently hold) and allows you to make trades (increase or decrease stocks, bonds and cash) on their behalf. It contains the following endpoints:

```
GET /customer/:customerId

Returns customer details with status code 200

Response Body:

{
        customerId: 1,
        stocks: 6700,
        bonds: 1200,
        cash: 400
}



POST /execute

Execute a batch of trades for a given set of customers. Returns status code 201 if trades succeed.

Request Body:

[{
        customerId: 1,
        stocks: 70,
        bonds: 40,
        cash: -30
},{
        customerId: 2,
        stocks: 170,
        bonds: -30,
        cash: -10
}]
```

Note that the FPS deals in absolute numbers of assets, not percentages. For the purposes of this test the cost of stocks and bonds is ignored. Assume that you can increase and decrease each asset number freely.

An implementation of the FPS is not provided. You are not expected to build a working implementation, but instead fake it in some way to ensure that your service will work when integrated with a functioning FPS.

Your goal is to write a service that does the following (the order may differ slightly in your implementation):

- Consumes **customers** and **strategy** CSV files
- For each customer, find which strategy applies to them using their risk level and years until retirement
- Retrieves the customer portfolios from the Financial Portfolio Service
- Use a customer's selected strategy's asset percentages, calculate the trades that must be made to rebalance their portfolio
- Batch customer trades and send them to the Financial Portfolio Service - the max amount of trades per batch should be configurable

Requirements

- This implementation must be completed in Java or Kotlin with Spring Boot and Maven.
- There should be an explanation of how to start up the application and get it running.
- Spend a maximum of 4 hours on this challenge.