Fetch Backend Internship Challenge

Disclaimer

Although this assignment may look long, most of the text below is there to help you understand what is being asked from you for the assignment as well as example requests/responses that will help you verify your code is correct. We want to test your coding skills rather than your ability to interpolate text, so we erred on the side of over-explanation to help you succeed with the assignment.

What do I need to submit?

Please write a program that implements the conditions outlined in the next section. You can use any language. We must be able to compile and run your code. Please provide any documentation necessary to accomplish this as part of the code you submit. Please assume the reviewer has not executed code in your language before when writing your README.

Part 1

Background

Our users have points in their accounts. Users only see a single balance in their account, but for reporting purposes, we track their points per payer. A payer is the producer of the item that points were added through. For example, a person who redeems yogurt may have points added through the payer DANNON. We track the payer from whom points are added, as well as when the points were added, so we know which payer to bill when the user spends their points on a reward.

You are being tasked with building a REST API that will help keep track of points and point transactions. Your API should be served on port 8000, and endpoints should accept and return JSON when required.
The transactions that you will need to implement are:

- Adding points
- Spending points
- Fetching the current point balance

Further details for each endpoint are outlined below.

Although we normally keep track of many users with different IDs/accounts, this functionality is out of the scope of the assignment. Your code can assume it is only working with a single user and their point transactions.

Endpoints

Add Points

Route: /add

Method: POST
Description: When a user has points added, we will use an /add route that accepts a transaction which contains how many points will be added, what payer the points will be added through, and the timestamp for when the transaction takes place. The request body for this endpoint will look like the following:

```JavaScript
{
	"payer" : "DANNON",
	"points" : 5000,
	"timestamp" : "2020-11-02T14:00:00Z"
}
```

Your service should keep track of each transaction when a new one is added. **If the transaction was added successfully, then your endpoint should respond with a status code of 200. You do not need to include a response body.**

Spend Points

Route: /spend
Method: POST
Description: When a user goes to spend their points, they are not aware of what payer their points were added through. Because of this, your request body should look like

```JavaScript
{"points" : 5000}
```

When a spend request comes in, your service should use the following rules to decide which payer to spend points through:

- We want the oldest points to be spent first (oldest based on transaction timestamp, not the order they're received)
- We want no payer's points to go negative

If a request was made to spend more points than what a user has in total, then we should return a status code of 400 and a message saying the user doesn't have enough points. This can be done through a text response rather than a JSON response. If the user does have enough points, then the above rules should be applied to decide which payer the above points should be spent through. **After your service has successfully calculated who to remove points from, the endpoint should respond with a status code of 200 and a list of payer names and the number of points that were subtracted.** An example of a response body looks like the following:

```Unset
[
	{ "payer": "DANNON", "points": -100 },
	{ "payer": "UNILEVER", "points": -200 },
	{ "payer": "MILLER COORS", "points": -4,700 }
]
```

Get Points Balance

Route: /balance
Method: GET
Description: This route should return a map of points the user has in their account based on the payer they were added through. This endpoint can be used to see how many points the user has from each payer at any given time. **Because this is a GET request, there is no need for a request body. This endpoint should always return a 200 and give a response body similar to the following:**

```
Unset

{
  "DANNON": 1000,
  ”UNILEVER” : 0,
  "MILLER COORS": 5300
}
```

Testing Your Solution

Calling the `/add` endpoint with the following transactions (each line being a single call)

- { "payer": "DANNON", "points": 300, "timestamp": "2022-10-31T10:00:00Z" }
- { "payer": "UNILEVER", "points": 200, "timestamp": "2022-10-31T11:00:00Z" }
- { "payer": "DANNON", "points": -200, "timestamp": "2022-10-31T15:00:00Z" }
- { "payer": "MILLER COORS", "points": 10000, "timestamp": "2022-11-01T14:00:00Z" }
- { "payer": "DANNON", "points": 1000, "timestamp": "2022-11-02T14:00:00Z" }

Then calling the `/spend` endpoint with the following call

- { "points": 5000 }

Should have a call to `/balance` endpoint returning the following response

```
Unset

{
  "DANNON": 1000,
  ”UNILEVER” : 0,
  "MILLER COORS": 5300
}
```

Part 2

Please answer the following questions and paste your answers into a file called summary.txt. This file should be located in the same repository as your code.

1. Why did you choose the tools, libraries, and language you used for the coding exercise?
2. What are the advantages and disadvantages of your solution?
3. What has been a favorite school/personal project thus far? What about it that challenged you?

How do I submit

Provide a link to a public repository, such as GitHub or BitBucket, that contains your code via the provided link through Greenhouse. Any submissions in another format (.zip for example) will not be reviewed by our team.

FAQ

How will this exercise be evaluated?

An engineer will review the code you submit. At a minimum, they must be able to run the code and the code must provide the expected results. You should provide any necessary documentation within the repository.

There should not be any commented out debug code (#Print("Inside the loop")) left in the final draft, everything should be formatted nicely and spelled correctly. Humans are grading this, not machines. Impress us!

I have questions about the problem statement.

For any requirements not specified via an example, use your best judgment.

Can I provide a private repository?

If at all possible, we prefer a public repository because we do not know which engineer will be evaluating your submission. Providing a public repository ensures a speedy review of your submission. If you are still uncomfortable providing a public repository, you can work with your recruiter to provide access to the reviewing engineer.