Instructions for Running the Fetch Rewards Web Servlet

This is a Java 11 Spring Boot servlet. The requirements for running this is a Java 11 runtime environment installation. The web server is a tomcat web server embedded in the Spring Boot application, so no other web server container is needed. Please make sure any other web servers on this machine are down, so that this application can bind to port 8080.

Once you have the runtime installed, go inside the folder containing the application (cd FetchRewardsServlet/Fetch_Rewards_Spring_Demo_Server) and change to the target directory. To bring the application up, run java -jar fetchrewards-0.0.1-SNAPSHOT.jar. The window you are running this application in will show java console output. The servlet will continue to run until you interrupt this command (on linux/Mac, ctrl C) This server is a rest server, and has as input json, and it also outputs json, as needed per the requirements in the Points document. I used postman to test this servlet.

The application persists its data in memory (as required by the Points document), so once you bring down the application, all data disappears and you start over again. The three routes are as follows (localhost assumes you are launching the request from the same machine, but fill in with the appropriate host or network address).

Return All Payer Point Balances

A **GET** request with the url http://localhost:8080/api/points, no request body will return all payer point balances as specified in the points document (json output in the response body).

Add Transactions for a Specific Payer and Date

A **POST** request with the url http://localhost:8080/api/points, and a request body like this one: { "payer": "DANNON", "points": 1000, "timestamp": "2020-11-02T14:00:00Z" } will add the specified points for the specified payer and specified timezone to the in-memory persistence layer. If all goes well, there is a response with no body, and a 200 status. If the request has negative points, you'll have 3 possible results: If there are not enough points to compensate for the negative points for this payer, an error will occur, and a BAD REQUEST status will be returned with a message to this effect in the body. If there is another record with a timestamp older than this one, the negative points will be subtracted from the balance of the oldest timestamp record. Any leftover negative points, or any negative points with no older timestamp record will be held separately. The payer's balance will reflect the negative points.

Spend points using the rules above and return a list of { "payer": <string>, "points": <integer> } for each call.

A **DELETE** request with the http://localhost:8080/api/points, and a request body like this one: { "points": 5000 } will spend the points indicated. The successful response will have a body in json like the following: { "payer": "DANNON", "points": -100 },{ "payer": "UNILEVER", "points": -200 }, { "payer": "MILLER COORS", "points": -4,700 }]. If there are not enough points, a BAD_REQUEST response will be sent, with an error message to the effect that there are not enough points.