nu-event-processor

nu-event-processor is a tool for processing account events.

Dependencies

nu-event-processor can be run either from a JAR file or from a Docker image. We recommend running it from a JAR file for performance reasons, but feel free to run it from the Docker image as well.

JAR file dependencies

- Java

Docker image dependencies

- Docker

Usage

```
nu-event-processor 0.1.0-SNAPSHOT (75b140e6)

nu-event-processor is a tool for processing account events

Usage:
  java -jar nu-event-processor-0.1.0-SNAPSHOT-standalone.jar < $INPUT_FILE [OPTIONS]

Options:
  -v, --version  Show version
  -h, --help     Show help
```

JAR file

1. Download

   ```
   wget https://s3.amazonaws.com/nu-event-processor/jars/nu-event-processor-0.1.0-SNAPSHOT
   ```

2. Run

   ```
   java -jar nu-event-processor-0.1.0-SNAPSHOT-standalone.jar --help
   ```

Docker image

Based on `openjdk:11.0.4-jre-slim` which is 204MB.

```
docker run mpereira/nu-event-processor:0.1.0 --help
```

Implementation details

The nu-event-processor has one main functionality: reading, parsing and processing JSON-encoded operations that are written to the program via standard input.

The project structure should feel familiar to people with exposure to Clojure.

```
Dockerfile
README.org
project.clj
src/
    nu_event_processor/
        core.clj
        duplicate_preventer.clj
        operation.clj
        processor.clj
        rate_limiter.clj
        rules.clj
        system.clj
        utils.clj
        validations.clj
test/
    nu_event_processor/
        core_test.clj
        duplicate_preventer_test.clj
        rate_limiter_test.clj

4 directories, 15 files
```

There are end-to-end *integration-like* tests in the `nu-event-processor.core-test` namespace. They're *integration* tests in the sense that they call the program's `-main` directly and only deal with the input written to and output written from the program, so they emulate an actual usage of the program from a command line as much as possible other than actually shelling out commands.

The end-to-end make sure that the program follows the provided specification:

1. *Account* operations instantiate the account
2. *Transaction* operations possibly change the account state
3. Every input record has a respective output record

4. Both *account* and *transaction* operations are checked for their respective set of rule violation conditions
5. Input records that violate rules will result in an output account record with a collection of `violations`

One of the specification's rule violation conditions is the rate of events exceeding a specified limit. To be able to identify when that happens, a rate limiter was implemented. There are multiple rate-limiting strategies. The one chosen here was Token Bucket. The reasoning is detailed in the docstring for `nu-event-processor.rate-limiter/rate-limit`, which is encouraged to be read. I'll quote it here for convenience:

`nu-event-processor.rate-limiter/rate-limit:`

> A purely functional implementation of https://en.wikipedia.org/wiki/Token_bucket.
>
> This strategy is arguably more correct than a 'Fixed Window' strategy where event counts are maintained in time interval buckets of arbitrary resolutions, given that that strategy allows up to '2 * max-interval - 1' events in a given interval (where N is the supposed maximum number of events allowed per interval) if they are timed right.
>
> A 'Sliding Window' strategy would be the most correct in terms of making it impossible for more than capacity events in any given time interval of size per-interval-s, at the cost of having both time and space complexity of 'O(capacity)'.
>
> With the 'Token Bucket' strategy it is possible that more than capacity events in a given time interval of size per-interval-s. Events are rate limited based on the actual rate of events. As long as the rate isn't above the desired average (capacity / per-interval-s) for long enough, events won't be rate-limited. This implementation has both time and space complexity of 'O(1)' and is commonly used in high-performance rate-limiters.

Another rule violation described in the specification is transactions with the same amount and merchant happening within a 2 minutes interval. To be able to identify when that happens a simple strategy was implemented:

1. Keep track of the last time a record key was seen
2. When a record with the same key is seen again, compute the time difference
3. If the time difference is smaller than 2 minutes, it's a duplicate

The implementation can be seen in the `nu-event-processor.duplicate-preventer` namespace.

Both the rate limiter and the duplicate preventer were implemented in a way that allows them to be used with record types other than *account* or *transaction* records, thanks to parameterizing the function to extract a record's key (`key-fn`)

and a record's datetime value (`time-fn`). This flexibility is demonstrated in their unit tests, where much simpler record shapes are used for clarity.

In their unit tests it is also possible to see that their implementations are purely functional. This allows for interesting things, like keeping track of all state transitions through time.

The code that checks for rule violation is in the `nu-event-processor.rules` namespace. Functions were implemented as multimethods in a way that adding rules only requires a new `defmethod` associated with the new rule. I also took the liberty to introduce violation error messages in prose instead of type names as shown in the specification. For example, instead of showing `high-frequency-small-interval` in the `violations` field, `There has been more than 3 transactions in the last 2 minutes` is shown instead. Error message handling also takes advantage of the same multimethod structure.

There are no unit tests for a couple of namespaces, but they should be being exercised through the end-to-end tests.

The whole program state is kept in an atom, which is mutated during event processing, which happens in the `nu-event-processor.processor` namespace.

Running a simulation

The `nu-event-processor.core-test` namespace exercises the program through a few scenarios, which are encouraged to be taken a look at.

This simulation will exercise a very simple scenario. Feel free to run the program through your own scenarios as well. Make sure the JAR file shown in the *Dependencies* section is downloaded.

Download operations file

```
wget https://s3.amazonaws.com/nu-event-processor/operations/operations.ndjson
```

Inspect operations file

```
cat operations.ndjson
```

Output:

```
{"account":{"active-card":true,"available-limit":1000}}
{"transaction":{"merchant":"Burger King","amount":30,"time":"2019-02-13T10:00:00.000Z"}}
{"transaction":{"merchant":"McDonald's","amount":20,"time":"2019-02-13T10:00:30.000Z"}}
{"transaction":{"merchant":"Bob's","amount":15,"time":"2019-02-13T10:00:59.000Z"}}
{"transaction":{"merchant":"Cinema","amount":30,"time":"2019-02-13T10:01:00.000Z"}}
{"transaction":{"merchant":"McDonald's","amount":20,"time":"2019-02-13T10:01:30.000Z"}}
```

```
{"transaction":{"merchant":"McDonald's","amount":20,"time":"2019-02-13T10:01:59.000Z"}}
{"transaction":{"merchant":"Bob's","amount":15,"time":"2019-02-13T10:02:00.000Z"}}
{"transaction":{"merchant":"C&A","amount":100,"time":"2019-02-13T10:02:30.000Z"}}
```

Run the program

With Java:

```
java -jar nu-event-processor-0.1.0-SNAPSHOT-standalone.jar < operations.ndjson
```

Or with Docker:

```
docker run -i --rm mpereira/nu-event-processor:0.1.0 < operations.ndjson
```

Output:

```
{"account":{"active-card":true,"available-limit":1000},"violations":[]}
{"account":{"active-card":true,"available-limit":970},"violations":[]}
{"account":{"active-card":true,"available-limit":950},"violations":[]}
{"account":{"active-card":true,"available-limit":935},"violations":[]}
{"account":{"active-card":true,"available-limit":905},"violations":[]}
{"account":{"active-card":true,"available-limit":905},"violations":["There has been a simila
{"account":{"active-card":true,"available-limit":905},"violations":["There has been more tha
{"account":{"active-card":true,"available-limit":905},"violations":["There has been a simila
{"account":{"active-card":true,"available-limit":905},"violations":["There has been more tha
```

Development

Work on nu-event-processor is mostly done on Emacs. The workflow looks like:

1. A CIDER session is started with `M-x cider-jack-in`
2. Code is evaluated with with `cider-eval-sexp-at-point` or `cider-eval-buffer`
3. Tests are run with `cider-test-run-test` or `cider-test-run-ns-tests`

Dependencies

- Java
- Leiningen
- Docker

Check out repository

```
git clone git@github.com:mpereira/nu-event-processor.git
```

cd into repository

```
cd nu-event-processor
```

Running tests

```
lein test
```

Building uberjar

```
lein do clean, uberjar
```

Publishing uberjar

Create AWS bucket if it doesn't exist.

```
aws s3 mb s3://nu-event-processor
```

```
aws s3 cp --acl public-read \
  target/uberjar/nu-event-processor-0.1.0-SNAPSHOT-standalone.jar \
  s3://nu-event-processor/jars/nu-event-processor-0.1.0-SNAPSHOT-standalone.jar
```

Building Docker image

```
docker build -t mpereira/nu-event-processor:0.1.0 .
```

Publishing Docker image

```
docker login
```

```
docker push mpereira/nu-event-processor:0.1.0
```

License

Copyright © 2019 Interviewer

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.