Take-home challenge proposal: Payments Version

By Elizabeth Collins 5 min 7

Introduction

This take-home challenge has two parts, one is implementation of a function that receives data in a **structured text format**. The function should parse the data and return a JSON string, as specified below. The second part is a brief SQL exercise.

Part 1: *processTransaction* Implementation

Technologies

Please use whatever programming language, environment, and IDE you're most comfortable with. Also feel free to use standard and/or third-party libraries in your solution (we don't expect you to create a JSON formatter from scratch).

You are not expected to build a user interface, but you can if that's the easiest solution for you.

Deliverables

You'll deliver the following:

- A short **technical document** describing how you intend to solve the problem.
- A zip archive of **working code**.
- A few **unit tests** and/or **integration tests**.

We don't expect this to be a large time investment, so please don't get stuck on full test coverage, comprehensive documentation, micro-optimizations, or any features not described below.

Grading

We'll assess your code on the following attributes:

- **Readability.** Reviewers should be able to quickly read and understand your code.
- **Code quality.** Your code should be idiomatic and well-formatted. It should follow best practices.
- **Ownership.** We'll ask you to explain and modify your code during a live follow-up call. Please don't submit any code you don't understand.

Specification

Part A: Basic functionality

Implement a function `processTransaction`.

- It should have a single parameter `transaction` of type `string` (if you're using a strongly-typed language).
- It should also return a string.

Input format

The input string is structured as follows:

- The first character is a **tag**. It tells you what value is about to be given.

- The second and third characters are a **length**. They tell you the length (in characters) of the following value.
- The next several characters, up to the given length, are a **value**.
- The pattern repeats until the end of the string.

The **tags** are defined as follows:

- `1` is the **payment network**, e.g. `VISA` or `AMEX`.
- `2` is the **transaction amount**, e.g. `20.00` meaning “20 dollars.” It will always be formatted with two decimal places.
- `3` is the **merchant**, e.g. `WALMART`.
- Any other tags should be ignored.

Tags may be given in any order.

Input examples

`104VISA20522.00310BURGERBARN` or `20522.00104VISA310BURGERBARN`

- `1`: The payment network
  - `04`: is 4 characters long
  - `VISA`: the value is `VISA`
- `2`: The transaction amount
  - `05`: is 5 characters long
  - `22.00`: the value is `22.00`
- `3`: The merchant
  - `10`: is 10 characters long
  - `BURGERBARN`: the value is `BURGERBARN`

`309SMAINFRMR108DISCOVER2070100.95`

- `3`: The merchant
  - `09`: is 9 characters long
  - `SMAINFRMR`: the value is `SMAINFRMR`
- `1`: The payment network
  - `08`: is 8 characters long
  - `DISCOVER`: the value is `DISCOVER`
- `2`: The transaction amount
  - `07`: is 7 characters long
  - `0100.95`: the value is `0100.95`

Output format

The response body should be JSON representing an object with the following properties:

- `version`: (string) The value `"0.1"`. This property never changes.
- `transaction_id`: (string) A randomly-generated GUID.
- `amount`: (string) The transaction amount in cents, without a decimal point. Leading zeros are not allowed.
  - If the input amount is `22.00`, this string should be `2200`.
  - If the input amount is `0100.95`, this string should be `10095`.
- `network`: (string) The payment network, exactly as given in the input string.
- `transaction_descriptor`: (string) This value differs according to the payment network.
  - For `VISA` network transactions, it should be the same as `amount`, but padded with leading zeros as necessary to make it 8 characters long.
    - If the input amount is `22.00`, this string should be `00002200`.

    - If the input amount is `480095.00`, this string should be `48009500`.
  - For all other networks, it should be the first two letters of the payment network followed by the characters `FFFF`.
    - For an `AMEX` transaction, this string should be `AMFFFF`.
    - For a `DISCOVER` transaction, this string should be `DIFFFF`.
- `merchant`: The merchant name. If it's longer than 10 characters, it should be truncated to 10 characters.
  - If the merchant is `SHOES`, this value should be `SHOES`.
  - If the merchant is `SOUTHMAINFARMERSMARKET`, the value should be `SOUTHMAINF`.
- `raw_message`: (string) The input string, exactly as given.

Output examples

Feel free to use the following to define unit tests. JSON properties may be in any order. Since the `transaction_id` is randomly generated, any GUID value is acceptable.

1

Input: `20522.00104VISA310BURGERBARN`

Expected output:

```json
{
  "version": "0.1",
  "transaction_id": "97a85330-ad60-4784-8b0e-30b4485d3885",
  "amount": "2200",
  "network": "VISA",
  "transaction_descriptor": "00002200",
  "merchant": "BURGERBARN",
  "raw_message": "20522.00104VISA310BURGERBARN"
}
```

2

Input: `309SMAINFRMR108DISCOVER2070100.95`

Expected output:

```json
{
  "version": "0.1",
  "transaction_id": "435301e4-515f-4837-83f1-8c0797f99b75",
  "amount": "10095",
  "network": "DISCOVER",
  "transaction_descriptor": "DIFFFF",
  "merchant": "SMAINFRMR",
  "raw_message": "309SMAINFRMR108DISCOVER2070100.95"
}
```

3

Input: `103JCB502QS316COSTSAVERGROCERY20564.80`

Expected output:

```json
{
  "version": "0.1",
  "transaction_id": "9f998f29-7d34-46ca-ac18-8ab16e3d63d5",
  "amount": "6480",
  "network": "JCB",
  "transaction_descriptor": "JCFFFF",
  "merchant": "COSTSAVERG",
  "raw_message": "103JCB502QS316COSTSAVERGROCERY20564.80"
}
```

Part B: Something extra

Finally, take a few minutes to add something that shows your strengths as a developer.

Choose one of the following:

1. Cache or store each transaction as it's processed, and then create a function that returns all previous transactions.
2. Make an architecture diagram for an application like this one where:
   a. A web page provides a front-end UI for the `processTransaction` function, which is exposed through an HTTP endpoint.
   b. All transactions are recorded to a database.
   c. Each decoded transaction is asynchronously submitted to a third-party API.
   d. The application handles thousands of requests per second.
3. Build 2a above to provide a front-end UI for the `processTransaction` function that takes an input string and displays the response in a human-readable way.

Part 2: SQL Exercise

Given the following two tables, write a SQL statement that returns the first and last name of the customer, along with the full street address, city, state and zip code for any customer that meets the following criteria:

- Customer account is active (1)
- The address is their primary address (1)
- The customer account was created on or after 2024-01-01 and before 2024-07-01

`customer` table:

id	first_name	last_name	active	created_at
1	Sam	Smith	1	2024-01-01 23:44:51
2	Jane	Thomas	0	2024-03-10 11:21:43
3	Larry	Jones	1	2024-07-16 14:21:35
4	Laura	Miller	1	2025-01-15 12:34:56

`address` table:

id	customer_id	primary_address	street_address	city	state	zipcode
1	1	0	55 Main St	Lincoln	NE	68501
2	1	1	21 Oak Ln	Westbury	NY	11590
3	3	1	781 Bruce St	Aurora	IN	47001
4	4	0	90 Union Ct	Chandler	AZ	85224
5	3	1	3417 Briar Ave	Kansas	IL	61933