Pismo Code Assessment

Welcome candidate!

You are finally at the Code Test stage of Pismo's hiring process for Software Engineering positions.

This code test is divided into two phases. During the first phase, you will work alone to create the proposed project and implement its first requirements.

The second phase will take place on a scheduled day and time, and you will be asked to add more requirements to the project you developed during Phase 1. It is an online code session and should take 1 hour and 30 minutes to complete. You should be ready to share your screen, run your project and answer questions about your work.

The following pages describe the requirements you should complete in Phase 1.

Project Guidelines

Your project must:

- Use **Java** or **Go** programming languages;
- Be published on Github;
- Contain a readme file with run instructions;

Your project should:

- Make use of automated tests;
- Be able to run on Docker;
- Document its API using Swagger / OpenAPI Specification;

Problem: Customer Account & Transactions

A customer has an account. For each operation performed by the customer, a transaction is created and associated with their respective account.

Transactions are specified by an identifier, a type, an amount, and a creation date. The available types are purchase, installment purchase, withdrawal, and payments.

Purchase, installment purchase, and withdrawal transaction types are stored with negative amounts (debt transactions), while payments are stored with positive amounts (credit transactions).

Data Structures

Check below a suggested data structure:

Accounts

Account_ID	Document_Number
1	12345678900

OperationsTypes

OperationType_ID	Description
1	PURCHASE
2	INSTALLMENT PURCHASE
3	WITHDRAWAL
4	PAYMENT

Transactions

Transaction_ID	Account_ID	OperationType_ID	Amount	EventDate
1	1	1	-50.0	2020-01-01T10:32:07.7199222
2	1	2	-23.5	2020-01-01T10:48:12.2135875
3	1	3	-18.7	2020-01-02T19:01:23.1458543
4	1	4	60.0	2020-01-05T09:34:18.5893223

Initial Requirements

1 - Implement a REST endpoint for creating accounts.
	1.1 - The request body should receive a document number that uniquely identifies the account owner;
	1.2 - The response body should return the account object if successful, or an error message otherwise;

Request:

```
POST /accounts

{
      "document_number": "12345678900"
}
```

Response:

```
{
      "account_id": 1,
      "document_number": "12345678900"
}
```

2 - Implement a REST endpoint for retrieving existing accounts.
	2.1 - The endpoint should receive the accountId as a query parameter;
	2.2 - An invalid *accountId* value should return an error message;

Request:

```
GET /accounts/:accountId
```

Response:

```
{
      "account_id": 1,
      "document_number": "12345678900"
}
```

3 - Implement a REST endpoint for creating transactions.

3.1 - The request body should receive the account owner ID, the operation type and the amount for the transaction;

3.2 - The system should validate whether an existing account owner and operation type were informed;

3.3 - The response body should return the transaction object if successful, or an error message otherwise;

Request:

```
POST /transactions

{
      "account_id": 1,
      "operation_type_id": 4,
      "amount": 123.45
}
```

Response:

```
{
      "transaction_id": 1,
      "account_id": 1,
      "operation_type_id": 4,
      "amount": 123.45
}
```