Documentação

1. Requisitos

- Dois tipos de usuários, os comuns e lojistas, ambos têm carteira com dinheiro e realizam transferências entre eles.
- Usuários podem enviar dinheiro (efetuar transferência) para lojistas e entre usuários.
- Lojistas só recebem transferências, não enviam dinheiro para ninguém.
- Antes de finalizar a transferência, deve-se consultar um serviço autorizador externo, use este mock para simular (https://run.mocky.io/v3/8fafdd68-a090-496f-8c9a-3442cf30dae6).
- A operação de transferência deve ser uma transação (ou seja, revertida em qualquer caso de inconsistência) e o dinheiro deve voltar para a carteira do usuário que envia.
- No recebimento de pagamento, o usuário ou lojista precisa receber notificação enviada por um serviço de terceiro e eventualmente este serviço pode estar indisponível/instável. Use este mock para simular o envio (https://run.mocky.io/v3/b19f7b9f-9cbf-4fc6-ad22-dc30601aec04);
- Este serviço deve ser RESTFul.

2. Tecnologias utilizadas

- Laravel 8 como ferramenta principal de desenvolvimento da API;
- Mysql 8 para banco de dados;
- Redis para cache e queue jobs;
- nginx como web server;
- Docker

3. Padrão de respostas dos endpoints

3.1. Sucesso

- success: status do retorno. “ok” em sucesso.
- message: mensagem de retorno.
- data: Payload de retorno.

3.2. Erro

- success: status do retorno. “error” em caso de erro.
- message: mensagem de retorno.
- data: Payload de retorno.
- reason: Um código único de erro

4. Endpoints

4.1. POST /transaction

Executa a transação entre um pagador e um recebedor.

Como o ID do pagador é enviado por parâmetro, foi entendido que esse módulo possa ser utilizado como microsserviço e logo o usuário pagador não precisaria ser o mesmo do usuário logado. Caso não seja utilizado como microsserviço então a falta dessa validação de mesmo usuário deve ser implementada ou será uma falha grave de segurança.

Como não foi especificado se este módulo será utilizado como microsserviço, então foi implementado um endpoint com essa validação em “/api/v1/transaction”. Logo, em “/transaction” não possui validação de autenticação enquanto em “/api/v1/transaction” possui.

- Payload:
  - value: Valor a ser transferido. Tipo número não negativo. Obrigatório;
  - payer: ID do usuário pagador. Não pode ser um usuário do tipo “loja”. Tipo integer ou string. Obrigatório.
  - payee: ID do usuário recebedor. Tipo integer ou string. Obrigatório.
- Retorno: Sucesso (item 3.1) com a transação do pagador como payload.

4.2. Prefixo /api/v1/

Esse prefixo é protegido por autenticação Bearer Token.

- Headers:
  - Authorization: Bearer XXXXX

4.3. POST /api/v1/transaction

Mesmo corpo e retorno do endpoint “/transaction” (item 4.1) porém com validação de usuário autenticado e validação onde o ID o pagador tem que ser o mesmo do ID do usuário logado.

4.4. GET /api/v1/transaction

Lista de transações do usuário logado como payload da mensagem de sucesso (item 3.1).

4.5. GET /api/v1/me

Dados do usuário logado contendo o saldo atual do mesmo como payload da mensagem de sucesso (item 3.1).

4.6. POST /oapi/v1/login

Login do usuário.

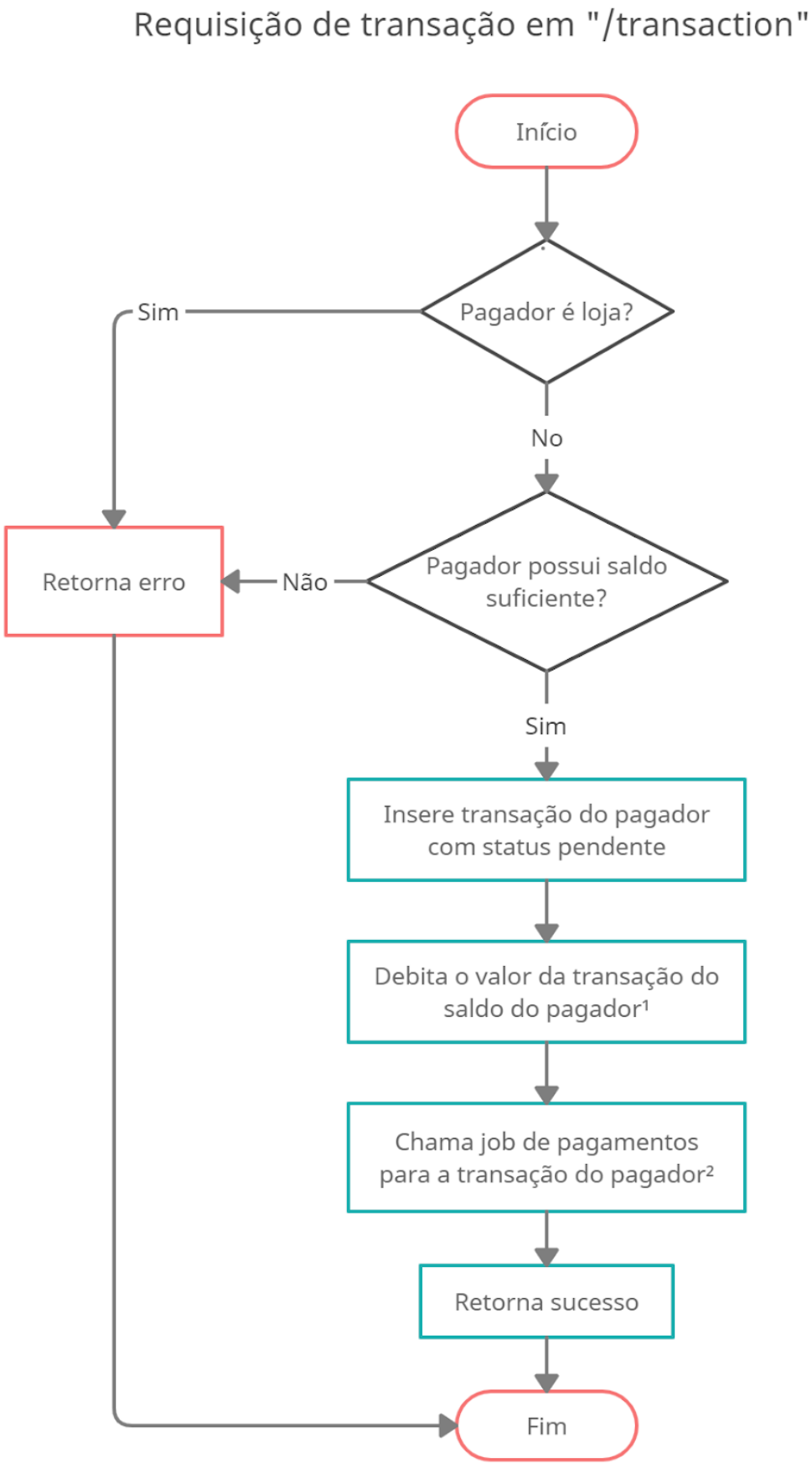
- Payload:
  - email: String. Obrigatório;
  - password: String. Obrigatório;
- Retorno:
  - Mensagem de sucesso (item 3.1) com token e data de expiração no payload.

4.7. POST /oapi/v1/register

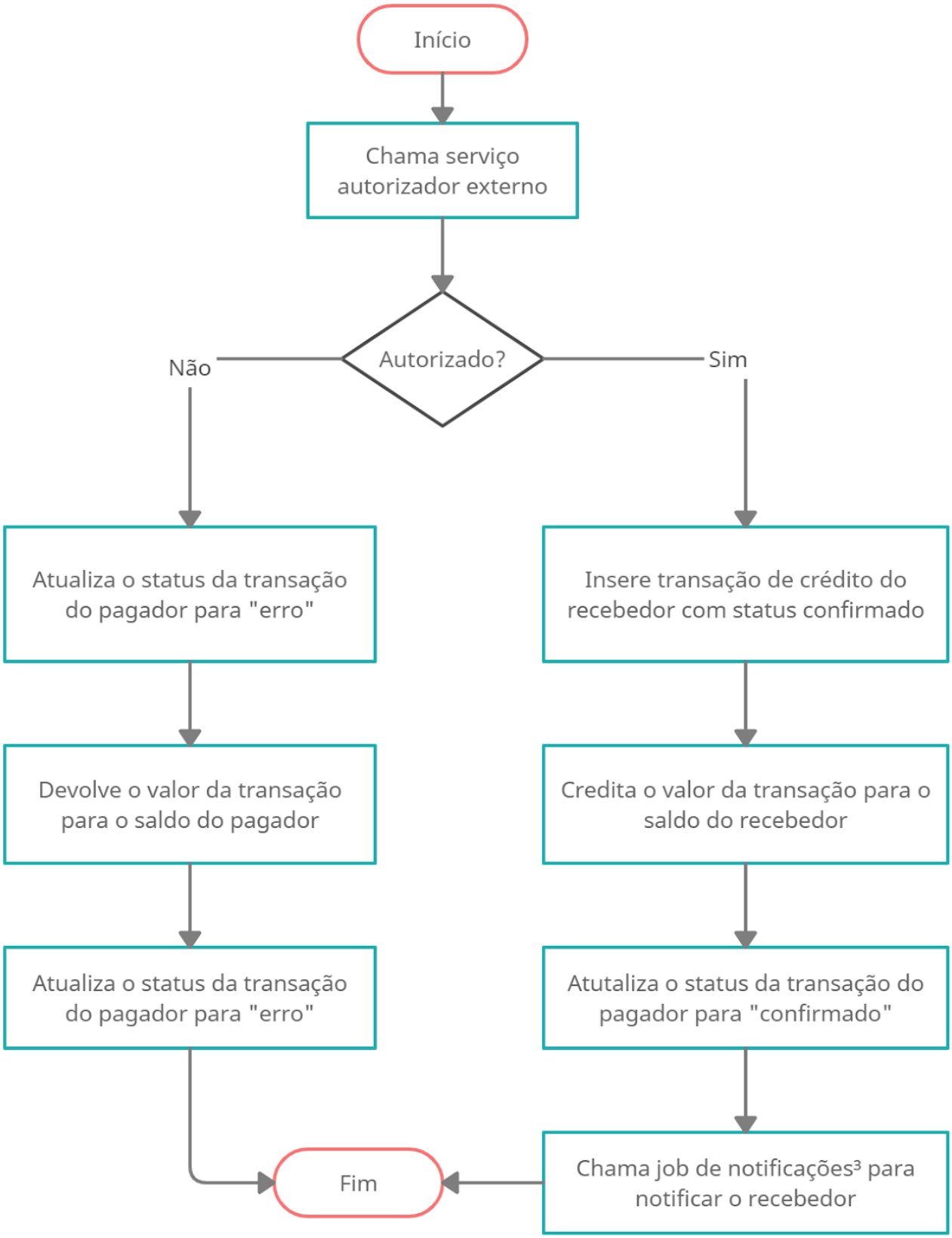
Registro de usuários.

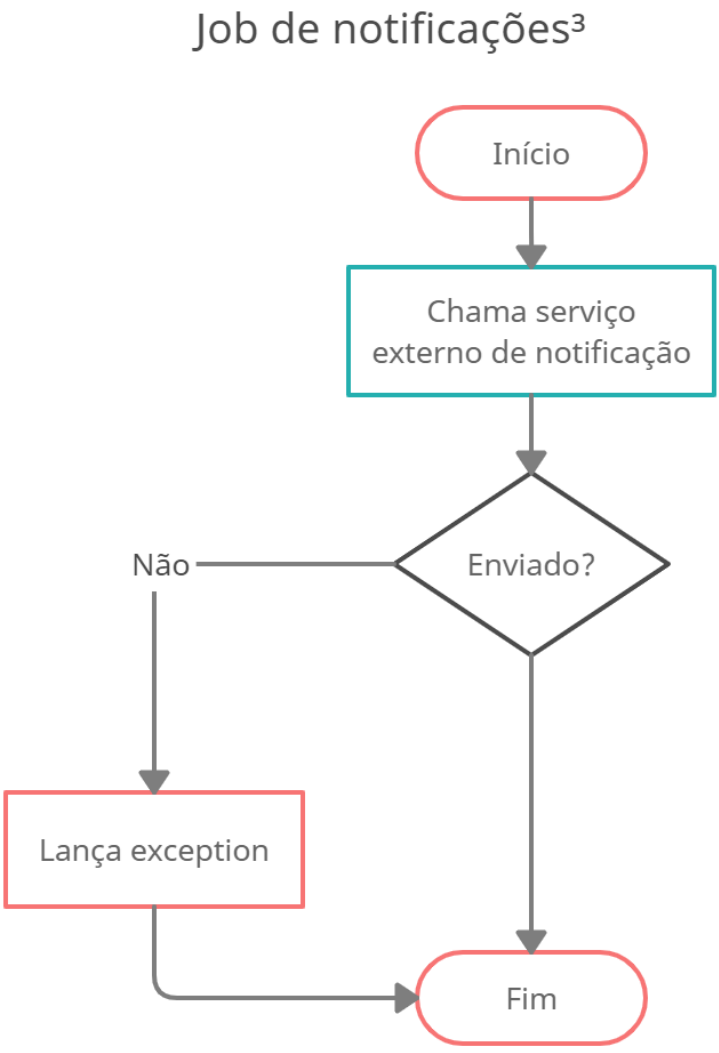
- Payload:
  - email: String. Obrigatório.
  - password: String. Obrigatório.
  - name: String. Obrigatório.
  - person_company_id: String. Obrigatório
- Retorno:
  - Mensagem de sucesso (item 3.1) sem payload.

5. Fluxograma



Job de pagamentos[2]





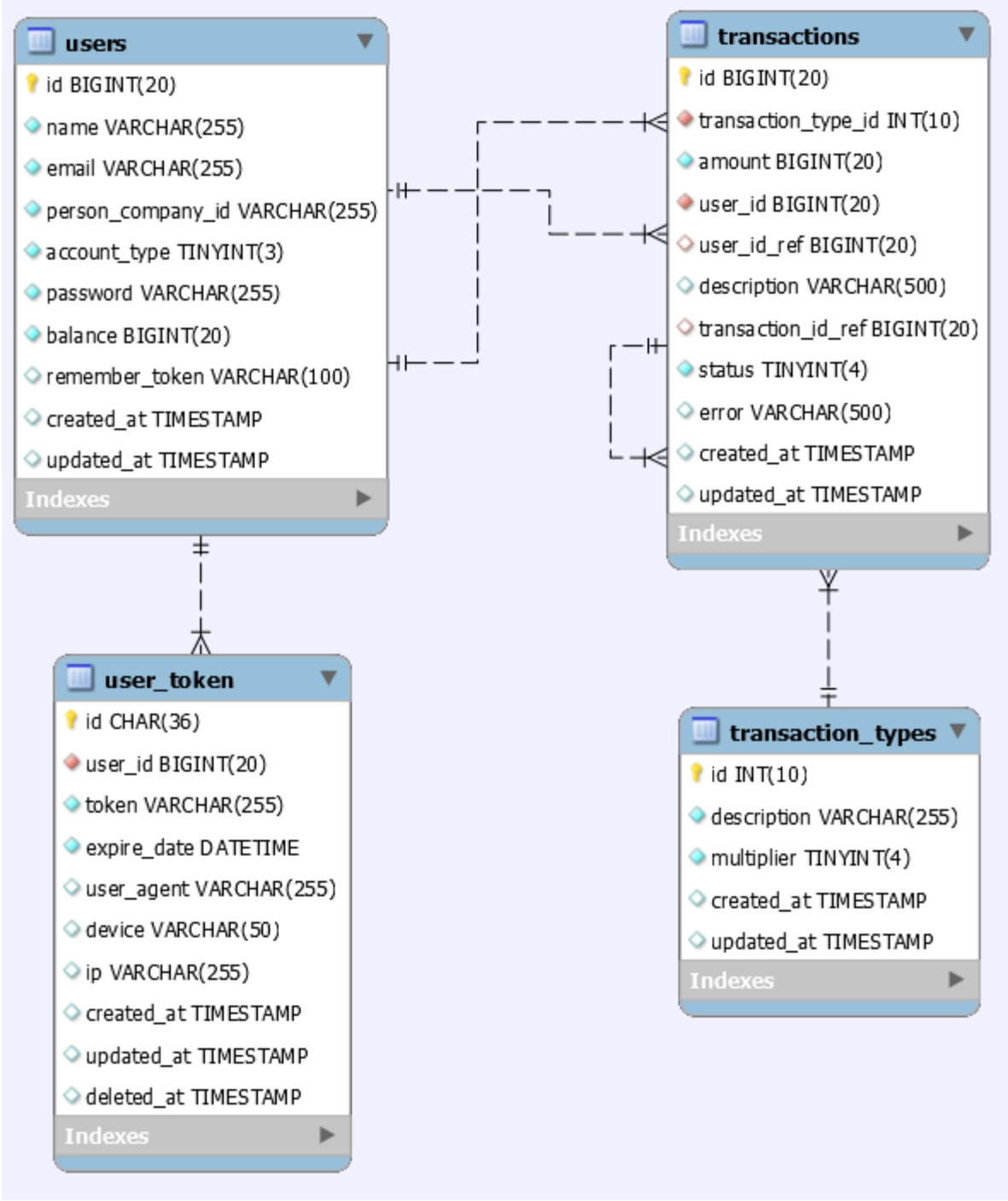
5.1. Notas

[1]Após inserir o registro de transação do pagador, é debitado o saldo da carteira do mesmo como saldo bloqueado para evitar realizar novos pagamentos com saldo inexistente.

[2,3]Jobs de pagamento e de notificação: esses serviços foram implementados em formato assíncrono utilizando os Queues do Laravel pois as chamadas de API externa de autorização de pagamento e serviço de notificações podem estar lentas piorando a experiência do usuário.

Link do fluxograma completo.

6. Diagrama de Entidade e Relacionamento (DER)



- Tabela users:
  - Usuários do sistema;
  - Saldo da carteira do usuário (coluna balance).
  - Tipo do usuário na coluna account_type onde 1 é cliente e 2 é loja.
  - Como fonte inicial de dados será inserido 4 usuários sendo 2 clientes e 2 lojas e todos eles iniciam com R$ 500,00 de saldo para testes da API;

- Tabela user_token:
  - Tokens de login dos usuários.
- Tabela transactions:
  - Transações dos usuários.
  - Coluna user_id: usuário principal do registro da transação;
  - Coluna user_id_ref: usuário secundário da transação;
  - Coluna transaction_id_ref: Transação secundária gerada pela transação principal.

    Exemplo: Usuário A pagou usuário B. Serão dois registros da tabela de transações onde a primeira transação será de débito do usuário A contendo user_id como usuário A e user_id_ref como usuário B. O segundo registro de transação será uma de crédito do usuário B contendo user_id como usuário B, user_id_ref como usuário A e transaction_id_ref como a primeira transação.
- Tabela transaction_types:
  - Tipo da transação;
  - ID 1: crédito;
  - ID 2: débito.

7. Pontos de melhoria

- Validar o saldo e fazer o débito do pagador no job de transações ao invés de debitar logo na primeira inserção da transação;
- Não mudar o status da transação em caso de erro e sim inserir uma nova transação com um novo tipo “Reembolso” para manter melhor o controle.
- Separar a carteira do usuário em uma nova entidade e não na mesma tabela de usuários.