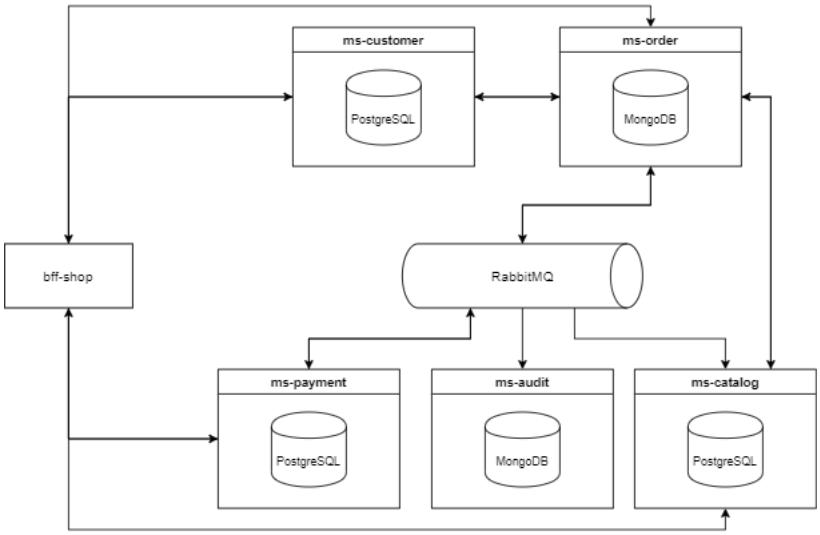
Projeto Shop Style

O Shop Style é uma loja física que vende roupas de todos os tipos e estilos. Os fundadores do Shop Style desejam agora abrir uma loja virtual e contrataram uma equipe para implementar. Os arquitetos já desenharam a solução e agora cabe a você implementar essa solução.

O projeto usará uma arquitetura de micro-serviços. Foi definindo a criação de seis micro-serviços: customer, catalog, payment, order, audit e o bff-shop. Todos os micro-serviços devem ter testes unitários com cobertura de **pelo menos** 80% e também um swagger correspondente.



MS Customer

O MS customer tem a responsabilidade de armazenar e gerenciar os dados de usuário e seus endereços. O MS customer possui os seguintes endpoints:

```
POST - /v1/login

POST - /v1/customers
GET - /v1/customers/:id
PUT - /v1/customers/:id
PUT - /v1/customers/:id/password

POST - /v1/address
PUT - /v1/address/:id
DELETE - /v1/address/:id
```

Campos da tabela customer:

```
ID, CPF, FIRST_NAME, LAST_NAME, SEX, BIRTHDATE, EMAIL, PASSWORD, ACTIVE
```

Campos da tabela address:

```
ID, STATE, CITY, DISTRICT, STREET, NUMBER, CEP, COMPLEMENT, CUSTOMER_ID
```

Observação:

- O campo ID de todas as tabelas deve ser gerado por auto incremento.

Exemplo de um payload para cadastrar cliente:

```
{
  "firstName": "Maria",
  "lastName": "Oliveira",
  "sex": "Feminino",
  "cpf": "000.000.000-00",
  "birthdate": "0000-00-00",
  "email": "maria@email.com",
  "password": "12345678",
  "active": true
}
```

Validações necessárias:

- Os campos `firstName` e `lastName` precisam ter no mínimo 3 caracteres.
- O campo `sex` só pode ter duas opções disponíveis Masculino e Feminino, caso contrário informar um erro ao usuário.
- O campo `email` precisa estar no formato de um email válido e não deve permitir emails duplicados.
- O campo `cpf` precisa seguir o seguinte padrão (xxx-xxx-xxx.xx).
- O campo `password` precisa ter no mínimo 6 caracteres e tem que ser salva criptografada no banco.
- O campo `birthdate` precisa ser salvo no banco como o tipo date e tem que estar no formato ISO-8601(https://pt.wikipedia.org/wiki/ISO_8601), entretanto na hora de serializar o objeto e enviar no payload do response esse campo precisa estar no formato dd/mm/aaaa.
- O campo `active` deve aceitar somente valores booleanos.

Exemplo de um payload para cadastrar endereço:

```
{
  "state": "Ceará",
  "city": "Fortaleza",
  "district": "Conjunto Ceará",
  "street": "Rua 202B",
  "number": "902",
  "cep": "60530-280",
  "complement": "",
  "customerId": 1
}
```

Validações necessárias:

- Todos os campos são obrigatórios exceto o campo `complement`.
- Todos os campos são textos.
- O campo `state` só deve aceitar valores de um dos 27 estados brasileiros, qualquer outro valor deve retornar um erro.

O body do endpoint `POST - /v1/login`:

```
{
  "email": "maria@email.com",
  "password": "12345678"
}
```

Observação:

- No endpoint `/v1/customers/:id` além de retornar os dados do cliente deve trazer todos os seus endereços.
- Usar o PostgreSQL.

MS Catalog

O MS catalog é o responsável por armazenar os produtos, skus e categorias que vão estar disponíveis na aplicação. Um produto tem um ou mais skus e está vinculado a uma categoria e uma categoria pode ter zero ou mais produtos. Uma sku tem uma ou mais medias e uma media é de uma sku. O MS catalog possui os seguintes endpoints:

```
POST - /v1/products
GET - /v1/products
GET - /v1/products/:id
PUT - /v1/products/:id
DELETE - /v1/products/:id

POST - /v1/skus
PUT - /v1/skus/:id
DELETE - /v1/skus/:id

POST - /v1/categories
GET - /v1/categories
GET - /v1/categories/:id/products
PUT - /v1/categories/:id
DELETE - /v1/categories/:id
```

Campos da tabela product:

```
ID, NAME, DESCRIPTION, BRAND, MATERIAL, ACTIVE, CATEGORY_ID
```

Campos da tabela sku:

```
ID, PRICE, QUANTITY, COLOR, SIZE, HEIGHT, WIDTH, PRODUCT_ID
```

Campos da tabela media:

```
ID, IMAGE_URL, SKU_ID
```

Campos da tabela category:

```
ID, NAME, ACTIVE, PARENT_ID
```

Observação:

- O campo ID de todas as tabelas deve ser gerado por auto incremento.

Exemplo de um payload para cadastrar um produto:

```
{
  "name": "Camisa Oficial do Fluminense",
  "description": "A camisa pra você que é tricolor de coração",
  "brand": "Umbro",
  "material": "Algodão"
  "active": true,
  "categoryId": 1
}
```

Exemplo de um payload para cadastrar uma sku:

```
{
  "price": 249.99,
  "quantity": 10,
  "color": "tricolor",
  "size": "M",
  "height": 100
```

```
  "width": 80
  "images": ["http://example.com/image-1.png", "http://example.com/image-2.png", "http://example.com/image-3.png", "http://example.com/imag
  "productId": 1
}
```

Exemplo de um payload para cadastrar uma categoria principal:

```
{
  "name": "Camisas",
  "active": true
}
```

Exemplo de um payload para cadastrar uma categoria filha:

```
{
  "name": "Camisas de Futebol",
  "active": true
  "parentId": 1
}
```

Validações necessárias:

- Os campos `name`, `description`, `brand`, `active` e `categoryId` são obrigatórios para salvar um produto.
- As categorias têm que estar ativa para um produto ser salvo.
- Produtos só podem ser salvos em categorias ativas e que não tem nenhum filho.
- Todos os campos mostrados acima são obrigatórios para cadastrar uma sku.
- Os campos `height` e `width` da sku tem que ser enviado em centímetros.
- Os campos `name` e `active` são obrigatórios para salvar uma categoria.

No endpoint `GET - /v1/categories` o retorno deve ser em formato de árvore, segue um exemplo logo abaixo:

```
[
  {
    "id": 1,
    "name": "Masculino",
    "active": true,
    "children": [
      {
        "id": 2,
        "name": "Roupas",
        "active": true,
        "children": [
          {
            "id": 3,
            "name": "Futebol",
            "active": true
          },
          {
            "id": 4,
            "name": "Elegante",
            "active": true
          }
        ]
      }
    ]
  },
  {
    "id": 5,
    "name": "Feminino",
    "active": true,
    "children": [
      {
        "id": 6,
        "name": "Roupas",
        "active": true,
        "children": [
          {
```

```
            "id": 7,
            "name": "Usual",
            "active": true
          },
          {
            "id": 8,
            "name": "Elegante",
            "active": true
          }
        ]
      }
    ]
  },
]
```

O ms-catalog deve escutar as mensagens enviadas via RabbitMQ pelo ms-order para diminuir o estoque das skus, a mensagem enviada pelo ms-order possui o seguinte formato:

```
{
  "orderId": "6294d4b66f71221237b4d211",
  "skus": [
    {
      "id": 1,
      "quantity": 1
    },
    {
      "id": 2,
      "quantity": 5
    }
  ]
}
```

Observações:

- Usar PostgreSQL e RabbitMQ.
- O endpoint `GET - /v1/products/:id` além de retornar as informações do produto, tem que listar todas as suas skus.
- Ao desativar uma categoria todas as suas categorias filhas serão desativadas.

MS Payment

O MS Payment é o responsável por gerenciar todos os métodos de pagamentos disponíveis. O MS Payment possui os seguintes endpoints:

```
POST - /v1/payments
GET - /v1/payments
PUT - /v1/payments/:id
DELETE - /v1/payments/:id

POST - /v1/installments
PUT - /v1/installments/:id
DELETE - /v1/installments/:id
```

Campos da tabela payments:

```
ID, TYPE, INSTALLMENTS, ACTIVE
```

Campos da tabela installments:

```
ID, AMOUNT, BRAND, PAYMENT_ID
```

Observação:

- O campo ID de todas as tabelas deve ser gerado por auto incremento.

Exemplo de um payload para cadastrar um método de pagamento:

```
{
  "type": "credit card",
  "installments": true,
  "active": true
}
```

Validações necessárias:

- Todos os campos são obrigatórios.

Exemplo de um payload para cadastrar a quantidade de parcelas disponíveis naquele método de pagamento:

```
{
  "amount": 5,
  "brand": "mastercard"
  "paymentId": 1
}
```

Validações necessárias:

- O campo `brand` não é obrigatório.
- Tem que validar se o `installments` do `paymentId` informado é true.

O ms-payment deve escutar as mensagens enviadas via RabbitMQ pelo ms-order com relação ao processamento de pagamento de um pedido. Os ms-payment deve processar essa mensagem que possui o seguinte formato:

```
{
  "orderId": "6294d4b66f71221237b4d211",
  "payment": {
    "id": 1,
    "installments": 0
  }
}
```

Depois de realizar o processamento da mensagem o ms-payment deve retornar o resultado, que possui o seguinte formato:

```
{
  "orderId": "6294d4b66f71221237b4d211",
  "status": "PAYMENT_SUCCESSFUL"
}
```

Os possíveis status que o ms-payment pode enviar para o ms-order são os seguintes:

- Pagamento realizado com sucesso - PAYMENT_SUCCESSFUL
- Pagamento não existe no banco - PAYMENT_NOT_FOUND
- Pagamento está inativado - PAYMENT_INACTIVE
- Pagamento não aceita parcelamento - PAYMENT_NOT_INSTALLMENT
- As parcelas informadas não estão dentro do limite definido - PAYMENT_AMOUNT_NOT_AVAILABLE

Observações:

- Usar PostgreSQL e RabbitMQ.

MS Order

O MS Order é o responsável por gerenciar todos os pedidos de compra realizadas na aplicação. O MS Order possui os seguintes endpoints:

```
POST - /v1/orders
GET - /v1/orders
GET - /v1/orders/customers/:customerId
```

Campos da coleção orders:

```
ID, CUSTOMER, PAYMENT, CART, DATE, STATUS, TOTAL
```

Exemplo de um payload para criar um pedido:

```
{
  "customer": {
    "id": 1,
    "addressId": 1
  },
  "payment": {
    "id": 1,
    "installments": 0
  },
  "cart": [
    {
      "skuId": 1,
      "quantity": 1
    },
    {
      "skuId": 2,
      "quantity": 5
    }
  ]
}
```

Validações necessárias:

- Todos os campos são obrigatórios.
- Dado o valor `id` e `addressId` que está dentro do objeto `customer`, o ms-order deve se comunicar com o ms-customer para saber se esse usuário existe, se está ativo e se o endereço informado realmente existe, caso não deve retornar um erro.
- Dado o valor `skuId` e `quantity` que estão dentro do objeto `cart`, os ms-order deve se comunicar com o ms-catalog para saber se existe essa sku e se tem disponível no estoque a quantidade solicitada, caso não atenda algum dos dois critérios deve retornar um erro.

Após realizar a inserção do documento, deve ser feito uma comunicação com o ms-payment para processar o pagamento desse pedido e o mesmo deve escutar o resultado enviado pelo ms-payment para atualizar o status do pedido no banco. Se o pagamento foi processado com sucesso, o ms-order deve enviar uma mensagem para o ms-catalog diminuir o estoque das skus do pedido.

Observações:

- Usar o MongoDB
- Na hora da inserção do documento na coleção deve ser calculado o total da compra, a partir do objeto `cart` é possível fazer esse calculo, assim como inserir a data e a hora que ocorreu a compra. O campo `status` deve ser salvo com o valor inicial de `PROCESSING_PAYMENT`.
- O endpoint `GET - /v1/orders` necessita de três query param, sendo que um é obrigatório. O query param obrigatório é o `startDate` que informa a partir de qual data que deseja filtrar os pedidos realizados, o segundo query param é o `endDate` que usado em conjunto com o `startDate` define um intervalo de tempo dos pedidos realizados. O ultimo query param é o `status` para filtrar os pedidos a partir do seu status.
- O endpoint `GET - /v1/orders/customers/:customerId` necessita de três query param, mas nenhum é obrigatório. O query param `startDate` informa a partir de qual data que deseja filtrar os pedidos realizados, o segundo query param é o `endDate` que usado em conjunto com o `startDate` define um intervalo de tempo dos pedidos realizados. O ultimo query param é o `status` para filtrar os pedidos a partir do seu status.

MS Audit

O MS audit é o micro-serviço responsável pela auditoria de todos os eventos que ocorreu no processamento de um pedido. O MS audit possui os seguinte endpoint:

```
GET - /v1/audit/orders/:orderId
```

Todos os eventos que transita entre o ms-order, ms-payment e ms-catalog devem ser salvas na base do ms-audit. Com essas informações salvas teremos uma visão ampla de todos os dados que foi transitado entre os micro-serviços que processa um pedido.

Observações:

- Usar o MongoDB.

MS BFF-Shop

Todos os micro-serviços serão de uso interno para os funcionários da empresa. Então precisa ser disponibilizado um ponto de entrada para que os clientes possam se comunicar com as funcionalidades. O MS bff-shop tem os seguintes endpoints:

```
POST - /v1/login

POST - /v1/customers
GET - /v1/customers/:id
PUT - /v1/customers/:id
PUT - /v1/customers/:id/password

POST - /v1/address
PUT - /v1/address/:id
DELETE - /v1/address/:id

GET - /v1/products
GET - /v1/products/:id

GET - /v1/categories
GET - /v1/categories/:id/products

GET - /v1/payments

POST - /v1/orders
GET - /v1/orders/customers/:customerId
```

Observação:

- Todos os endpoints precisam ser autenticados e autorizados via token JWT, exceto os endpoints `POST - /v1/login` e `POST - /v1/customers`.

ATENÇÃO: Esse projeto é para fins didáticos então algumas decisões de arquitetura foi tomada para ajudar no aprendizado de alguns conceitos, então algumas decisões técnicas usadas nesse projeto didático não serão arquitetados igualmente em projetos reais.