Mestrado em Engenharia Informática

Projeto de Informática

Relatório Técnico

Pack My Bag

PG53601 - Afonso Xavier Cardoso Marques
PG53651 - André Castro Alves
PG53797 - Eduardo Cardoso Pereira
PG53849 - Gonçalo Vilar Vale
PG53923 - João Miguel Faria Leite
PG53929 - João Paulo Peixoto Castro
PG54174 - Renato André Machado Gomes

Braga, 29 de novembro de 2024

Índice

Lista de Figuras

1. Resumo

O objetivo deste relatório técnico é documentar as decisões tomadas e os desenvolvimentos realizados no âmbito da criação da plataforma web da startup Pack My Bag, detalhando os fundamentos técnicos e estratégicos que sustentam a solução proposta. Este documento visa fornecer uma visão clara sobre a abordagem tecnológica adotada, os desafios enfrentados e as soluções implementadas, alinhadas aos objetivos de negócio.

A Pack My Bag posiciona-se como uma startup inovadora, focada em oferecer uma solução prática e sustentável para viajantes que buscam simplicidade e conveniência. A proposta do serviço inclui a possibilidade de os clientes alugarem roupas e acessórios diretamente no destino de viagem, eliminando a necessidade de transportar bagagens volumosas e contribuindo para a redução de custos relacionados ao transporte de bagagens em companhias aéreas.

2. Âmbito do Produto

A aplicação Pack My Bag é uma plataforma web inovadora que visa transformar a experiência de viagem dos utilizadores ao oferecer um serviço de aluguer de peças de roupa. O principal objetivo é eliminar a necessidade de transportar bagagem volumosa, permitindo que os viajantes aluguem roupas diretamente no destino da sua viagem.
Através da aplicação, os clientes podem:

- **Selecionar Peças de Roupa:** Explorar um catálogo diversificado de peças de diferentes marcas e lojas, organizadas por categoria e estilo.
- **Criar Conjuntos Personalizados:** Combinar peças para criar conjuntos que reflitam o seu gosto pessoal, com a possibilidade de incluir itens de várias marcas.
- **Optar por Conjuntos Pré-definidos:** Escolher entre conjuntos já preparados para maior conveniência.
- **Agendar Entregas:** Definir o local e a data de recolha das encomendas, seja num hotel, alojamento local, aeroporto ou outro ponto à sua escolha.
- **Receber Recomendações Personalizadas:** Aceder a sugestões de vestuário fornecidas por estilistas pessoais, com base num questionário de estilo preenchido pelo cliente.
- **Gerir Favoritos:** Guardar peças e conjuntos numa biblioteca pessoal para futuras consultas e alugueres.
- **Avaliar Peças Alugadas:** Deixar reviews sobre as peças alugadas, ajudando outros utilizadores nas suas escolhas.
- **Receber Notificações:** Manter-se informado sobre o estado das encomendas, disponibilidade de peças e lembretes de devolução.

Para as marcas e lojas parceiras, o PackMyBag oferece uma plataforma para expandir o seu alcance e proporcionar uma nova forma de interação com os clientes, através do aluguer das suas peças.

3. Levantamento de Requisitos

Após termos solidificado a ideia principal para a aplicação, procedeu-se ao levantamento de requisitos com o objetivo de compreender as necessidades e preferências dos potenciais utilizadores. Este processo foi fundamental para orientar o desenvolvimento da plataforma, assegurando que ela atenda efetivamente às expectativas do público-alvo.
Para obter uma visão abrangente dos desejos e necessidades dos clientes, foram utilizadas várias técnicas de coleta de informações:

- **Entrevistas com Viajantes Frequentes:** Foram realizadas entrevistas para entender os desafios enfrentados durante as viagens, especialmente relacionados ao transporte de roupas e bagagens volumosas.
- **Questionários Online:** Distribuímos questionários em plataformas digitais para recolher dados quantitativos sobre hábitos de consumo, preferências de estilo e receptividade a um serviço de aluguer de roupas.
- **Análise de Mercado:** Estudamos serviços similares em outros países para identificar funcionalidades bem-sucedidas.

Este levantamento permitiu alinhar as funcionalidades da aplicação com as reais necessidades dos utilizadores, assegurando que a Pack My Bag ofereça uma experiência valiosa e diferenciada no mercado de aluguer de roupas em Portugal.

3.0.1. Modelação de Requisitos

Os requisitos funcionais e não funcionais encontram-se descritos na secção dos anexos na Seção 19.1

Para o levantamento de requisitos foi utilizada a *requirement shell* do modelo *Volere* como forma de representação, para os descrever concisamente.
Como caracterização da tabela de representação de requisitos, é necessário descrever os campos:

- **Requisito**: número de identificação do requisito.
- **Tipo**: tipo de requisito.
- **Use Cases**: número dos Use Cases associados.
- **Descrição**: descrição clara e concisa do requisito.
- **Origem**: quem originou o requisito.
- **Prioridade**:índice de prioridade para a implementação do requisito:
  -Must: requisito obrigatório;
  -Should: requisitos que deve ser implementados;
  -**Could**: requisito que não é necessário, mas é desejado;
  -**Won't**: requisito que pode ser considerado posteriormente.

4. Frontend

O frontend da aplicação foi desenvolvido utilizando o framework *Vue.js*, que permite a criação de interfaces de utilizador interativas e responsivas. A escolha do Vue.js deveu-se à sua facilidade de integração, curva de aprendizado amigável, performance eficiente e permite two-way data bindings. Com ele, construímos componentes reutilizáveis que facilitam a manutenção e evolução da interface. Além disso, a utilização do Vue possibilita a navegação dinâmica entre as diferentes vistas da aplicação. O frontend comunica-se com a API Gateway através de requisições HTTP assíncronas (estas permitem que um cliente faça uma solicitação e continue executando outras tarefas enquanto espera pela respota

do servidor), consumindo assim os serviços disponibilizados pelos microserviços. Esta arquitetura proporciona uma separação clara entre a interface do utilizador e a lógica de negócio, aumentando a modularidade e escalabilidade do sistema.

5. API-Gateway

A API Gateway foi implementada em *Express.js*, servindo como ponto central para a comunicação entre o frontend e os microserviços. A função principal da API Gateway é rotear as requisições para o microserviço adequado, além de agregar, filtrar ou transformar as respostas quando necessário. Nesta haverá uma função de middleware para validar o token e restringir o acesso a rotas protegidas. Esta camada facilita a gestão de autenticação, autorização e outras políticas de segurança. A escolha pelo Express.js se deve-se à sua capacidade de lidar com um grande número de requisições simultâneas de forma não bloqueante, graças ao seu modelo de eventos assíncronos. Concluindo, a API Gateway simplifica a interação do frontend com a arquitetura de microserviços, ocultando a complexidade e proporcionando uma interface unificada. Ela comunica-se com os microserviços, que foram desenvolvidos em Spring Boot, garantindo a integração suave entre as diferentes tecnologias utilizadas no sistema.

6. Estrutura dos Microserviços

O desenvolvimento de cada um dos seguintes microserviços foi feito usando o framework Springboot (Java). Em todos foi usado o pattern Controller-Service-Repository, que é bastante prevalente em aplicações deste framework e permite uma boa separação de responsabilidades. A camada Controller é a responsável por expor a funcionalidade para que ela possa ser consumida por entidades externas. A camada Repository será responsável por armazenamento e obtenção de dados. Por fim, a camada Service é onde toda a lógica de negócios se irá encontrar. Por outro lado, no que toca ao design pattern, foi utilizado o padrão DTO (data transfer object). Estes objetos são mapeados a partir dos modelos de domínio para DTO's através do uso de componentes mapper. Deste modo, este padrão demonstra ser vantajoso, pois podemos reduzir o Round-trip time (RTT) entre o cliente e o servidor, já que podemos restringir quais os dados que queremos serializar para a camada de apresentação e também acabámos por conseguir reduzir o número de chamadas remotas, pelo que tem impacto na performance da aplicação.

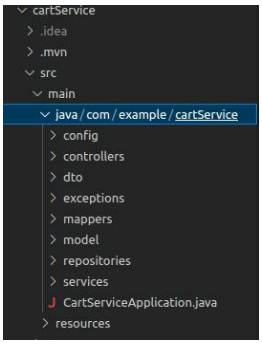

Figura 1: Padrão estrutural utilizado nos microserviços

7. Microserviço do Catálogo

O microserviço de Catálogo, desenvolvido em SpringBoot será responsável pela gestão de items e de toda a informação referente a estes. Neste serviço estão disponíveis as seguintes funcionalidades:

- **Consulta de um item**: Os utilizadores para cada item poderão consultar a designação, preço, disponibilidade, imagem ilustrativa do item, o tamanho, o rating médio, as reviews associadas a ele, e os itens relacionados caso o item se trate de um set.
- **Reservar recursos**: Foi implementado locking otimístico na atualização do número de itens disponíveis quando são realizadas encomendas. Ou seja, duas transações poderão aceder o mesmo registo, mas se fizerem commit ao mesmo tempo uma delas fará rollback e será levantada uma excessão. O que optámos por fazer foi dar catch a essa excessão e invocar novamente o método, de modo a atualizar o número de items disponíveis e a fazer a reserva da encomenda. Deste modo, não causámos transtorno ao utilizador. Caso o número de disponíveis de algum dos items da encomenda esteja a zero, é levantada uma excessão a dizer que esse item já se encontra esgotada. Nesta estratégia de optimistic locking, cada item terá um integer da versão correspondente desse registo.
- **Atualização do item**: O número de exemplares do item é atualizado assim como a sua disponibilidade (POST /api/catalogo/updateItems)
- **Fazer review**: Os utilizadores podem publicar um post com uma descrição e um rating com estrelas de 1 a 5.
- **Remover review**: Os utilizadores podem apagar as suas reviews.
- **Gerir itens**: Os técnicos das lojas podem remover e adicionar peças, sets e calçado.
- **Filtrar itens**: Os utilizadores podem aplicar filtros de procura baseados em parâmetros como preço, género, e designação.

7.1. Diagrama de Classes do Catálogo

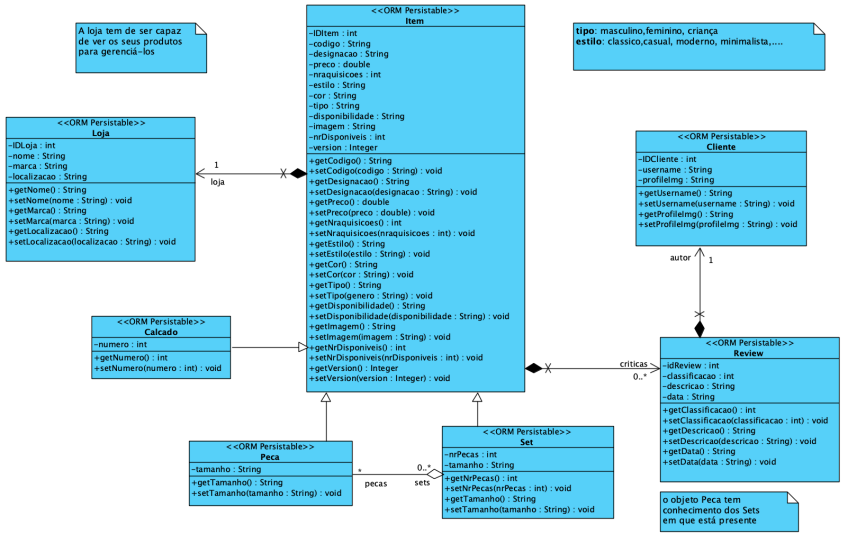

Figura 2: Diagrama de Classes do catálogo

O diagrama de classes mostra as principais entidades e seus relacionamentos:

- Loja: Representa as lojas que disponibilizam itens no catálogo, com informações como nome, marca e localização.
- Item: Representa cada produto do catálogo, contendo detalhes como código, preço, disponibilidade, estilo, cor e imagem.
- Peca: Refere-se às peças de roupa, com informações sobre tamanho, associadas a itens ou conjuntos específicos.
- Set: Representa conjuntos de peças agrupadas, indicando o número e o tamanho das peças.
- Calçado: Especialização de itens voltada para calçados, com o atributo adicional de número.
- Cliente: Representa os utilizadores, armazenando informações como username e imagem de perfil.
- Review: Registra avaliações dos clientes sobre os itens, incluindo classificação, descrição e data.

Este diagrama organiza os objetos que representam o catálogo de produtos disponíveis na plataforma, permitindo a interação entre clientes, lojas e itens.

7.2. Diagrama ER do Catálogo

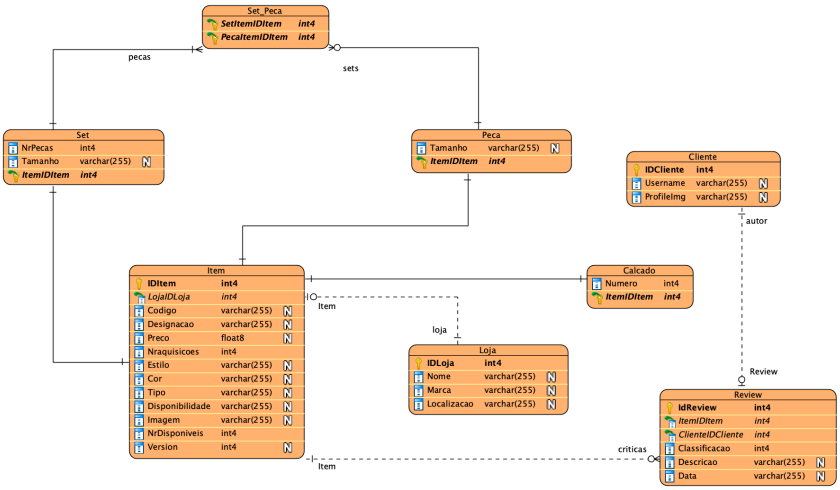

Figura 3: Diagrama ER do catálogo

O diagrama ER mostra a estrutura de dados subjacente ao microserviço de catálogo:

- Loja: Armazena informações sobre as lojas associadas aos itens.
- Item: Representa os produtos no catálogo, com atributos como código, descrição, preço e disponibilidade.
- Peca: Registra peças de roupa individuais, associadas a itens e conjuntos (sets).
- Set: Armazena informações sobre conjuntos de peças, como número de peças e tamanhos.
- Calçado: Armazena dados específicos sobre calçados, como número.
- Cliente: Armazena dados dos utilizadores da plataforma, incluindo username e imagem de perfil.
- Review: Registra as avaliações feitas pelos clientes sobre os itens, associando os comentários aos produtos e usuários.

As relações entre essas entidades suportam o ciclo de vida completo do catálogo, desde o gerenciamento de itens e conjuntos até as avaliações dos clientes.

Este microserviço atende às necessidades dos clientes e lojas ao permitir o gerenciamento eficiente de produtos disponíveis para aluguel e avaliação.

7.3. API do Catalogo

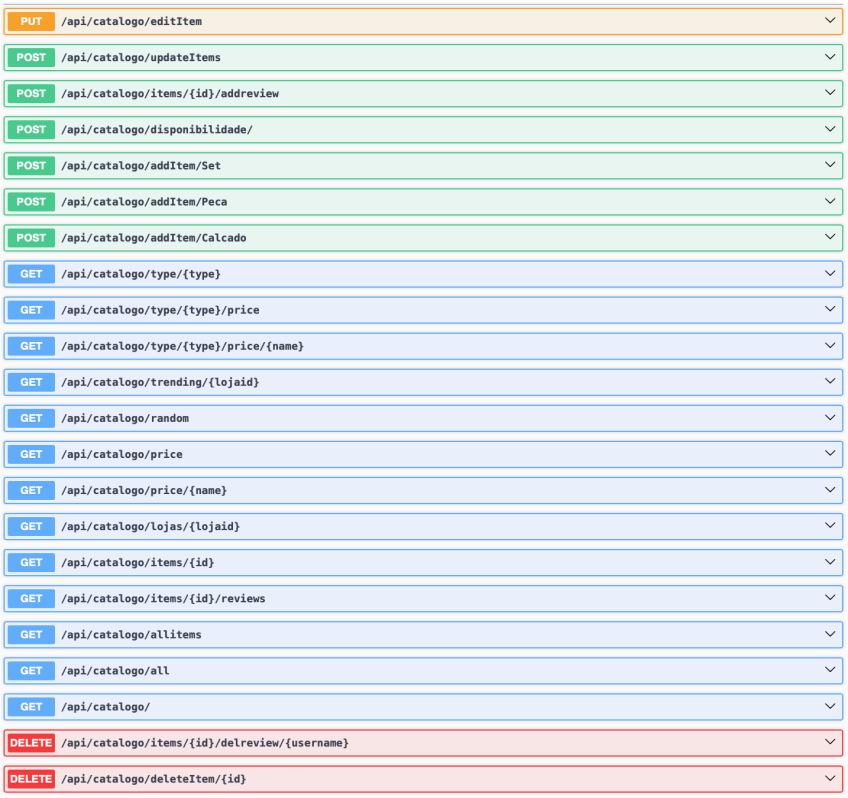

Figura 4: API do microserviço do catálogo

As principais rotas da API do microserviço do Catálogo são:

- **GET /api/catalogo/:** Retorna uma lista paginada de itens do catálogo.
- **GET /api/catalogo/all:** Retorna todos os itens do catálogo.
- **GET /api/catalogo/items/{id}:** Retorna os detalhes de um item específico pelo seu ID.
- **GET /api/catalogo/items/{id}/reviews:** Retorna as reviews de um item específico pelo seu ID.
- **POST /api/catalogo/items/{id}/addreview:** Adiciona uma review a um item específico pelo seu ID.
- **DELETE /api/catalogo/items/{id}/delreview/{username}:** Remove uma review de um item específico pelo seu ID e username do autor da review.
- **GET /api/catalogo/lojas/{lojaid}:** Retorna uma lista paginada de itens de uma loja específica pelo ID da loja.
- **GET /api/catalogo/type/{type}**: Retorna uma lista paginada de itens de um tipo específico.
- **GET /api/catalogo/type/{type}/price:** Retorna uma lista paginada de itens de um tipo específico dentro de um intervalo de preços.
- **GET /api/catalogo/type/{type}/price/{name}:** Retorna uma lista paginada de itens de um tipo específico e nome dentro de um intervalo de preços.
- **GET /api/catalogo/price:** Retorna uma lista paginada de itens dentro de um intervalo de preços.
- **GET /api/catalogo/price/{name}:** Retorna uma lista paginada de itens de um nome específico dentro de um intervalo de preços.
- **GET /api/catalogo/allitems:** Retorna uma lista paginada de itens por designação.
- **POST /api/catalogo/addItem/Peca:** Adiciona uma nova peça ao catálogo.
- **POST /api/catalogo/addItem/Set:** Adiciona um novo conjunto ao catálogo.
- **POST /api/catalogo/addItem/Calcado:** Adiciona um novo calçado ao catálogo.
- **DELETE /api/catalogo/deleteItem/{id}:** Remove um item do catálogo pelo seu ID.
- **PUT /api/catalogo/editItem:** Edita um item existente no catálogo.

- **POST /api/catalogo/disponibilidade/:** Diminui a disponibilidade de itens de uma encomenda.
- **GET /api/catalogo/random:** Retorna uma lista de itens aleatórios para a página inicial.
- **GET /api/catalogo/trending/{lojaid}:** Retorna uma lista de itens em tendência de uma loja específica pelo ID da loja.
- **POST /api/catalogo/updateItems:** Liberta os itens de uma encomenda.

8. Microserviço do Cesto

O microserviço do cesto, construído sobre a tecnologia *Java Spring Boot*, é responsável por gerir o processo de seleção, compra e pagamento de artigos selecionados pelos clientes do nosso sistema. As principais funcionalidades deste microserviço incluem:

- **Exibir o cesto do cliente**: Os clientes podem visualizar o seu cesto, tal que contém todos os artigos que este pretende comprar.
- **Detalhar os produtos do cesto**: Ao visualizar o cesto, os clientes podem ver detalhadamente quais peças de roupa que colocaram no cesto, incluindo informações como o preço, a quantidade de cada item colocado e o total custo do cesto.
- **Fornecer informações sobre a encomenda**: Após a confirmação dos conteúdos do cesto, os clientes fornecem detalhes sobre a entrega, tal como o local desta e a duração da sua viagem.
- **Escolha do método de pagamento**: Após confirmar que todos os detalhes sobre a encomenda e o custo que irá ficar aos clientes pelo aluguer dos artigos do cesto, os clientes podem escolher pagar tanto com cartão multibanco como por paypal.
- **Histórico de pagamentos**: Os clientes podem consultar o histórico completo dos pagamentos passados, permitindo que eles acessem informações sobre pagamentos anteriores e completar pagamentos em curso.
- **Libertar Recursos**: Foi contruído um array de Threads com nomes únicos de códigos de pagamento. Após dois minutos, se o pagamento não foi realizado é feito um POST de uma lista de objetos de items com a correspondente quantidade para a API Gateway para libertar esses recursos (rota que é reencaminhada posteriormente para serviço de Catálogo).

8.1. Diagrama de Classes do Cesto

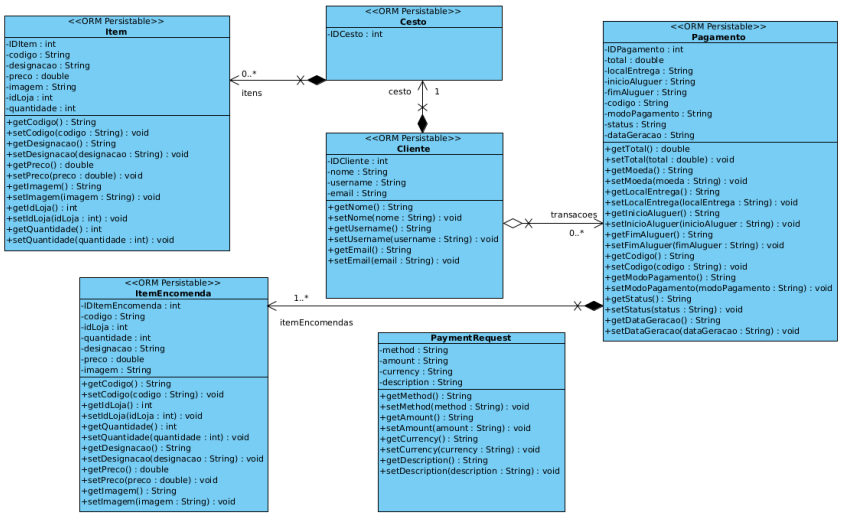

Figura 5: Diagrama de Classes do Microserviço do Cesto

O diagrama de classes mostra as principais entidades e seus relacionamentos:

- **Cliente**: Representa o utilizador que realiza a compra.
- **Cesto**: Contém todos os artigos que o cliente pretende alugar.
- **Item**: Cada produto que compõe um cesto.
- **Pagamento**: Informações sobre o pagamento e a entrega.
- **ItemEncomenda**: Representa os artigos de um cesto para serem usados para criar uma encomenda.
- **PaymentRequest**: Usada para criar os pagamentos com o serviço do *PayPal*.

Estes elementos trabalham em conjunto para permitir que os clientes possam alugar os artigos na nossa aplicação e que o serviço de encomenda possa gerar as encomendas para os clientes.

8.2. Diagrama ER do Cesto

Figura 6: Diagrama ER do Microserviço do Cesto

O diagrama ER acima mostra a estrutura de dados subjacente ao microserviço do cesto:

- A entidade Cliente possui informações como nome, utilizador e e-mail.
- A entidade Pagamento armazena os detalhes do pagamento, como modo de pagamento e status, e detalhes para criação de uma encomenda, como local de entrega, data de geração, etc.
- A entidade Item representa cada produto que compõe um cesto.
- A entidade ItemEncomenda contém informações sobre cada item alugado pelo cliente.

As relações entre estas entidades permitem que o sistema gerencie todo o processo de um cesto, desde a criação pelo cliente até a criação de uma encomenda e confirmação de pedido.
Este microserviço atende tanto às necessidades dos clientes, que podem visualizar o histórico dos seus pagamentos, quanto aos funcionários das lojas, que podem gerir novos pedidos feitos pelos clientes.

8.3. API do Microserviço do Cesto

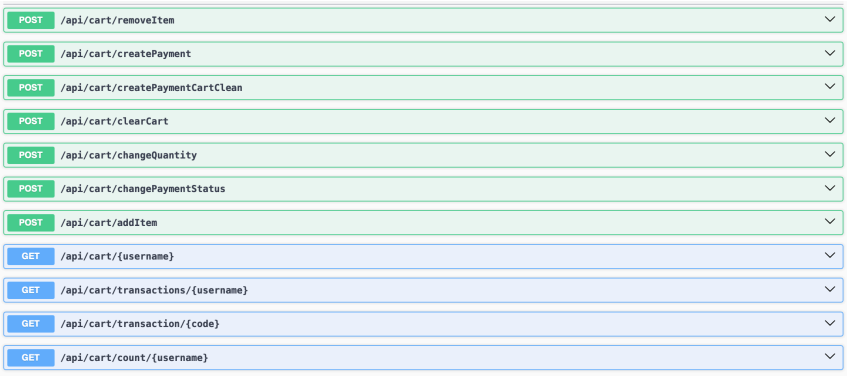

Figura 7: API do Microserviço do Cesto

As principais rotas da API do microserviço do Cesto são:

- **GET /api/cart/{username}**: Retorna todos os itens do cesto de um determinado cliente
- **GET /api/cart/transactions/{username}**: Retorna a lista de pagamentos de um determinado cliente
- **GET /api/cart/transaction/{code}**: Retorna um pagamento pelo id dele
- **GET /api/cart/count/{username}**: Retorna o número de artigos num cesto de um cliente
- **DELETE /api/cart/removeItem**: Remove um artigo de um cesto de um cliente
- **POST /api/cart/createPayment**: Cria um novo pagamento para o formulário dos estilistas
- **POST /api/cart/createPaymentCartClean**: Cria um novo pagamento, onde logo de seguida limpa o cesto do cliente
- **POST /api/cart/clearCart**: Limpa o carrinho de um cliente
- **POST /api/cart/changeQuantity**: Altera a quantidade de um item no cesto
- **POST /api/cart/changePaymentStatus**: Altera o estado de um pagamento
- **POST /api/cart/addItem**: Adiciona um item ao cesto de um cliente

Além disso, o microserviço também fornece endpoints para o pagamento através do serviço do paypal:

- **GET /api/cart/paypal/success**: Executa o pagamento após a aprovação do usuário no PayPal.
- **GET /api/cart/paypal/cancel**: Endpoint chamado quando o pagamento é cancelado pelo usuário.
- **GET /api/cart/paypal/error**: Endpoint chamado quando o pagamento é cancelado pelo usuário.
- **POST /api/cart/paypal/create**: Cria um pagamento no PayPal.

9. Microserviço de Encomendas

O microserviço de Encomendas desempenha um papel fundamental na aplicação *PackMyBag*, servindo tanto os clientes como os técnicos das lojas. Construído sobre a tecnologia *Java Spring Boot*, este microserviço é responsável por gerir de forma abrangente o histórico de encomendas realizadas na plataforma. As principais funcionalidades deste microserviço incluem:

- **Exibir a lista de encomendas do cliente**: Os clientes podem visualizar todas as suas encomendas anteriores, acompanhando o status de cada uma delas.
- **Detalhar os produtos de cada encomenda**: Ao selecionar uma encomenda, os clientes podem ver detalhadamente quais peças de roupa foram alugadas, incluindo informações como o preço.

- **Acompanhar o status da encomenda**: Durante todo o ciclo de vida da encomenda, desde a criação até a devolução, os clientes podem acompanhar o status em tempo real, recebendo notificações sobre eventuais atualizações.
- **Fornecer informações sobre a entrega**: O microserviço armazena e exibe informações relevantes sobre a entrega, como data de entrega e devolução.
- **Histórico de encomendas anteriores**: Os clientes podem consultar o histórico completo das suas encomendas passadas, permitindo que eles acessem informações sobre peças alugadas anteriormente.

Além disso, este microserviço também desempenha um papel fundamental na gestão das encomendas pelas lojas parceiras. Os técnicos e funcionários de cada loja podem:

- **Visualizar a lista de encomendas da sua loja**: Os funcionários têm acesso à lista completa de encomendas realizadas para a sua loja, permitindo que eles acompanhem a demanda e o fluxo de atividades.
- **Alterar o status da encomenda**: Durante todo o processo, desde a separação dos itens até a devolução, os funcionários podem atualizar o status da encomenda, garantindo que os clientes sempre tenham informações precisas sobre o status dos seus pedidos.

Esta abrangência de funcionalidades posiciona o microserviço de Encomendas como um elemento-chave na arquitetura da aplicação *PackMyBag*. Além disso, a sua construção sobre tecnologias modernas, como *Java Spring Boot* e *Vue.js*, asseguram a escalabilidade e a flexibilidade necessárias para suportar eventuais atualizações e novas funcionalidades da plataforma no futuro.

9.1. Diagrama de Classes de Encomendas

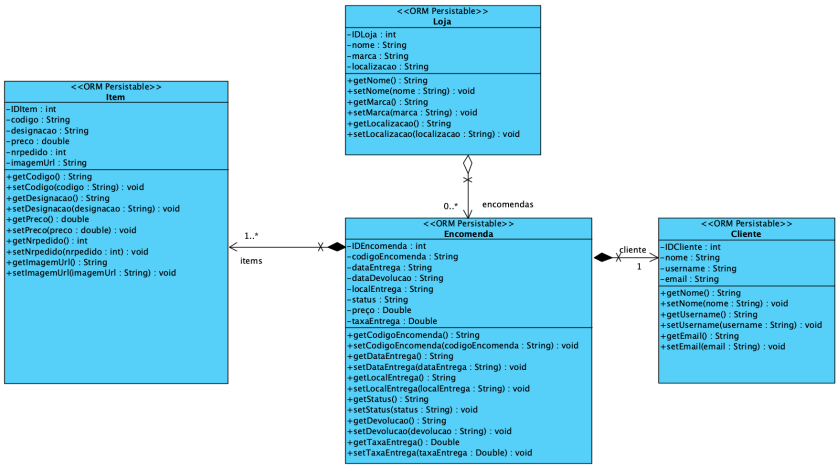

Figura 8: Diagrama de Classes do Microserviço de Encomendas

O diagrama de classes mostra as principais entidades e seus relacionamentos:

- **Cliente**: Representa o utilizador que realiza as encomendas.
- **Encomenda**: Registra os detalhes de cada encomenda realizada, como status, data de entrega, etc.
- **Item**: Cada produto que compõe uma encomenda.
- **Loja**: Informações sobre as lojas parceiras que fornecem as roupas para aluguel.
- **Tecnico/Funcionario**: Representa os funcionários das lojas que podem acessar e atualizar o status das encomendas.

Estes elementos trabalham em conjunto para permitir que os clientes realizem encomendas e os funcionários das lojas gerenciem o processo de entrega e devolução da encomenda.

9.2. Diagrama ER de Encomendas

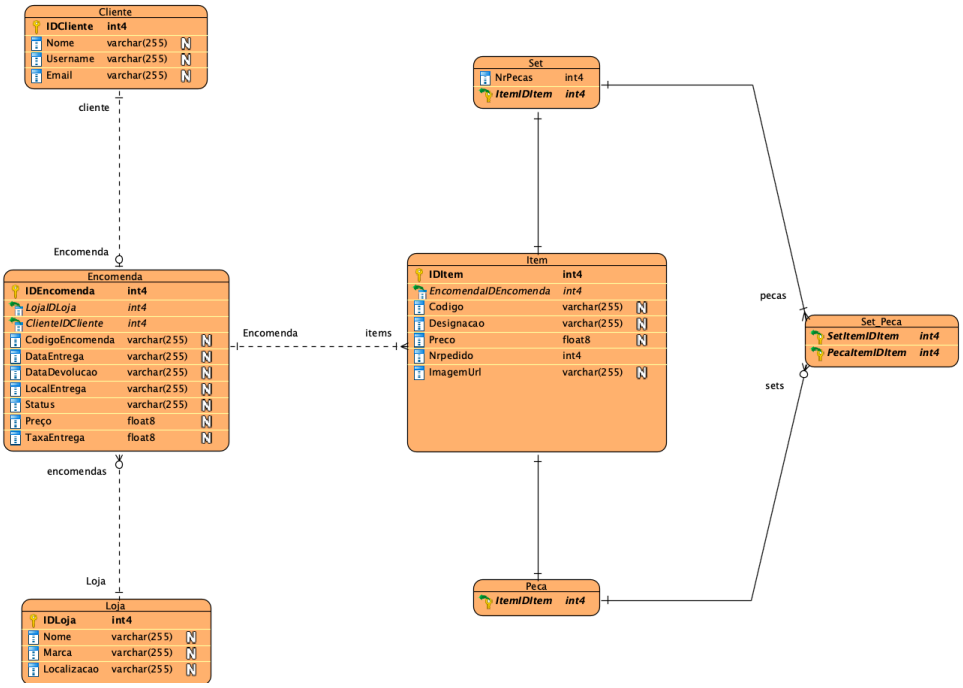

Figura 9: Diagrama ER do Microserviço de Encomendas

O diagrama ER mostra a estrutura de dados subjacente ao microserviço de Encomendas:

- A entidade Cliente possui informações como nome, utilizador e e-mail.
- A entidade Encomenda armazena os detalhes da encomenda, como status, data de entrega, etc.
- A entidade Item representa cada produto que compõe uma encomenda.
- A entidade Loja contém informações sobre as lojas parceiras, como nome, marca e localização.
- A entidade Tecnico/Funcionario regista os funcionários responsáveis por gerir as encomendas.

As relações entre estas entidades permitem que o sistema gerencie todo o ciclo de vida de uma encomenda, desde a criação pelo cliente até a entrega e devolução, com o acompanhamento dos funcionários das lojas.
Este microserviço atende tanto às necessidades dos clientes, que podem visualizar o histórico das suas encomendas, quanto aos funcionários das lojas, que podem gerir o status das encomendas durante todo o processo.

9.3. API do Microserviço de Encomendas

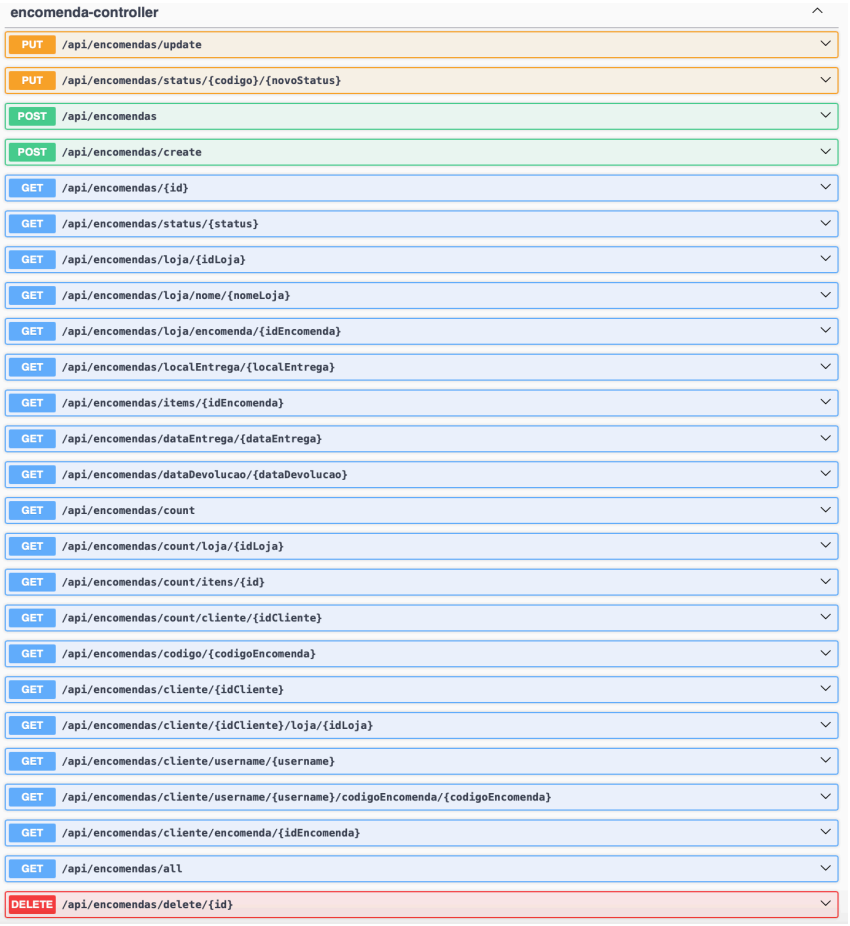

Figura 10: API do Microserviço de Encomendas

As principais rotas da API do microserviço de Encomendas são:

- **GET /api/encomendas/{id}**: Retorna os detalhes de uma encomenda específica
- **GET /api/encomendas/cliente/{idCliente}**: Retorna a lista de encomendas de um determinado cliente
- **GET /api/encomendas/loja/{idLoja}**: Retorna a lista de encomendas de uma determinada loja
- **GET /api/encomendas/dataEntrega/{dataEntrega}**: Retorna as encomendas com uma determinada data de entrega
- **GET /api/encomendas/dataDevoluacao/{dataDevoluacao}**: Retorna as encomendas com uma determinada data de devolução
- **POST /api/encomendas/create**: Cria uma nova encomenda
- **PUT /api/encomendas/update**: Atualiza os detalhes de uma encomenda
- **PUT /api/encomendas/status/{codigo}/{novoStatus}**: Atualiza o status de uma encomenda

Além disso, o microserviço também fornece endpoints para obter informações agregadas, como, por exemplo:

- **GET /api/encomendas/count**: Retorna o total de encomendas
- **GET /api/encomendas/count/loja/{idLoja}**: Retorna o total de encomendas de uma determinada loja
- **GET /api/encomendas/count/cliente/{idCliente}**: Retorna o total de encomendas de um determinado cliente

10. Microserviço dos Favoritos

O microserviço de itens favoritos gere uma das várias componentes de personalização da plataforma, sendo esta a possibilidade de guardar uma lista de artigos de roupa nos favoritos, ficando estes mais facilmente acessíveis para alugueres futuros. Semelhante a outros microserviços já explorados neste documento, o serviço foi implementado em *Java Spring Boot*. As principais funcionalidades deste microserviço incluem:

- **Exibir a lista de artigos favoritos do utilizador**: O utilizador pode aceder a uma página onde todos os itens que colocou como favorito se mantêm até que os elimine ou deixem de estar disponíveis na plataforma.
- **Adicionar um artigo aos favoritos**: O utilizador ao visualizar um artigo no catálogo, pode adicionar o artigo como favorito clicando no botão apropriado.
- **Remover um artigo aos favoritos**: O utilizador, na página dos favoritos, pode remover um artigo clicando no botão apropriado.

10.1. Diagrama de Classes dos Favoritos

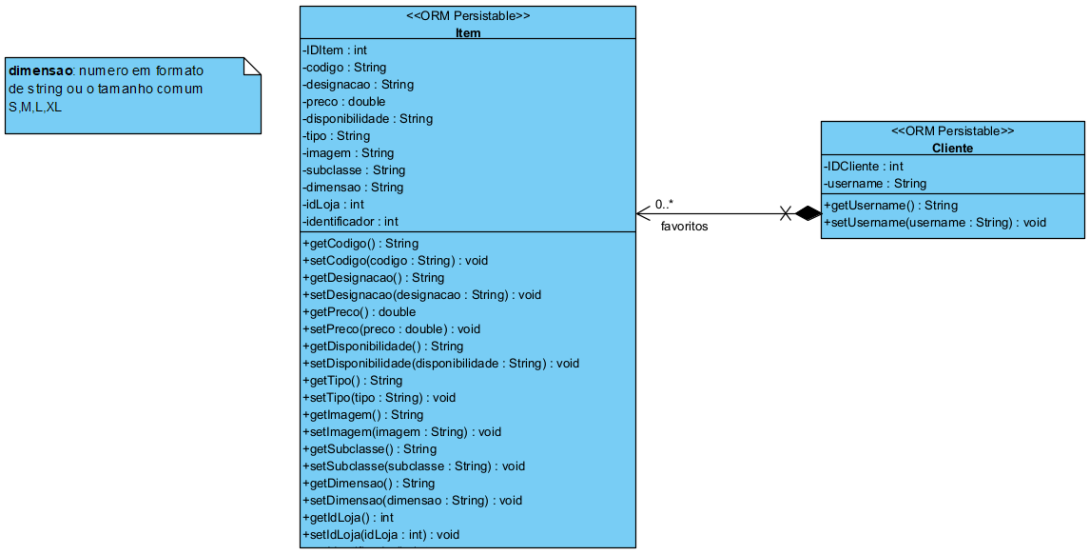

Figura 11: Diagrama de Classes do Microserviço de Favoritos

O diagrama de classes mostra as principais entidades e seus relacionamentos:

- **Cliente**: Representa o utilizador, numa forma mais simplificada, que está associado aos vários itens que considerou como favoritos.
- **Item**: Cada produto que constitui a lista dos favoritos.

Estes elementos trabalham em conjunto para permitir que os clientes possam consultar a lista dos artigos favoritos na nossa aplicação sem a ocorrência de conflitos a nível de dados, garantindo que um cliente consiga estar associado a múltiplos itens.

10.2. Diagrama ER dos Favoritos

O seguinte diagrama ER ilustra a estrutura de dados aplicada ao microserviço de gestão de Favoritos:

- A entidade Cliente possui informações sobre o utilizador.
- A entidade Item representa cada produto que compõe os favoritos.

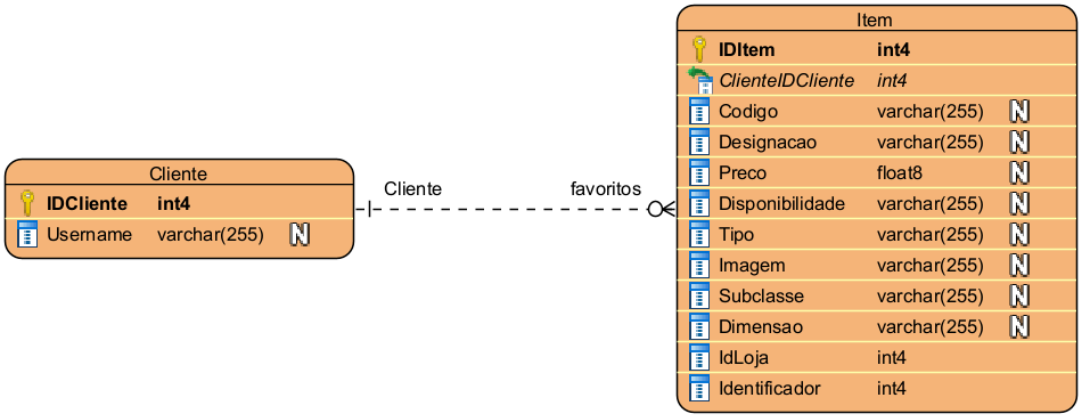

Figura 12: Diagrama ER do Microserviço de Favoritos

Este microserviço atende às necessidades dos clientes, que podem gerir os seus items favoritos presentes na plataforma.

10.3. API dos Favoritos

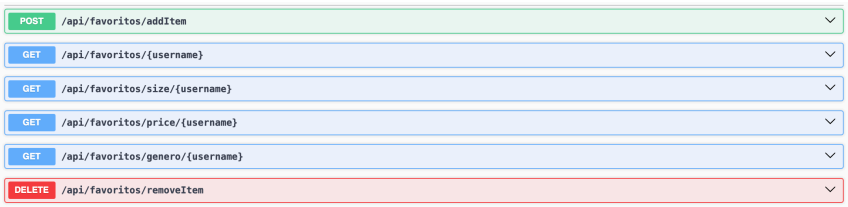

Figura 13: API do Microserviço dos Favoritos

-**POST /api/favoritos/addItem**: Adiciona um novo item à lista de favoritos de um utilizador.

-**GET /api/favoritos/{username}**: Retorna a lista de itens favoritos de um determinado utilizador.

-**GET /api/favoritos/size/{username}**: Retorna os itens da lista de favoritos de um determinado utilizador de acordo com o tamanho selecionado.

-**GET /api/favoritos/price/{username}**: Retorna os itens da lista de favoritos de um determinado utilizador de acordo com o valor selecionado.

-**GET /api/favoritos/genero/{username}**: Retorna os itens da lista de favoritos de um acordo com o gênero selecionado;

-**GET /api/favoritos/removeItem**: Remove um item da lista de favoritos de um utilizador.

11. Microserviço de Notificações

O Microserviço de Notificações é responsável por informar os clientes sobre eventos relevantes relacionados às suas interações com a plataforma, como atualizações de status de encomendas e disponibilidade de itens de interesse. Desenvolvido em Java Spring Boot, este serviço assegura que os clientes estejam sempre atualizados sobre informações importantes. Abrange apenas o cliente e consiste nas seguintes funcionalidades:

- **Exibir notificação quando houver mudança num item interessado**: Sempre que houver uma alteração na disponibilidade de um item, o cliente que estiver interessado neste deve ser notificado.
- **Exibir notificação quando houver mudança do status na encomenda**: O cliente deve ser notificado sempre que houver qualquer alteração no status das suas encomenda.
- **Exibir todas as notificações do cliente**: O cliente deve ter acesso a uma página em que pode visualizar todas as suas notificações (histórico de notificações).
- **Remover notificações da lista de notificações do cliente**: O cliente pode remover qualquer notificação do histórico.

11.1. Design Pattern Observer

Para gerir o fluxo de notificações, o microserviço implementa o Padrão Observer. Este é um padrão comportamental que define uma dependência um-para-muitos entre objetos, de forma que quando um objeto (o Subject) muda de estado, todos os objetos dependentes (os Observers) são notificados e atualizados automaticamente.

No contexto do microserviço:

- **Subjects:**
  - Item: Representa produtos disponíveis na plataforma. Quando a disponibilidade de um item muda, este notifica os clientes interessados.
  - Encomenda: Representa pedidos realizados pelos clientes. Alterações no status da encomenda, como envio ou entrega, geram notificações.
- **Observers:**
  - Cliente: Os clientes atuam como observadores, recebendo notificações sobre mudanças nos itens e encomendas aos quais estão associados.

A implementação deste padrão permite um sistema flexível e escalável, facilitando a manutenção e a extensão das funcionalidades de notificação.

11.2. Apache Kafka

Para viabilizar a comunicação assíncrona e eficiente entre os microserviços, utilizamos o Apache Kafka, uma plataforma de streaming distribuída que segue o padrão publisher-subscriber e que permite o processamento de fluxos de dados em tempo real. Em vez de os microserviços comunicarem diretamente através de chamadas API (comunicação síncrona), o que poderia levar a um forte acoplamento e complexidade na gestão das dependências, o Kafka atua como um intermediário confiável.

No nosso sistema:

- **Produtores:** Os microserviços do Catalogo e da Encomenda publicam eventos no Kafka quando ocorrem alterações relevantes, como a atualização de um item ou mudança de status de uma encomenda.

- **Consumidor:** O microserviço de Notificações subscreve-se a tópicos específicos no Kafka para receber e processar as mensagens enviadas pelos produtores.

Ao utilizar o Kafka, obtemos diversos benefícios:

- **Desacoplamento:** Os microserviços não dependem diretamente uns dos outros, aumentando a modularidade e facilitando a manutenção. Cada serviço pode evoluir e escalar de forma independente.
- **Escalabilidade:** O Kafka é capaz de lidar com grandes volumes de dados e múltiplos eventos simultaneamente, permitindo que o sistema cresça conforme a demanda sem perda de desempenho.
- **Confiabilidade:** Com mecanismos de tolerância a falhas, o Kafka garante a entrega consistente de mensagens, mesmo diante de interrupções ou falhas na rede.

Assim, o uso do Apache Kafka em vez de comunicação direta via APIs entre microserviços aprimora a arquitetura do sistema, tornando-a mais flexível, escalável e resiliente. Isso facilita a integração dos serviços e proporciona uma melhor experiência para os utilizadores finais.

11.3. Diagrama de Classes de Notificações

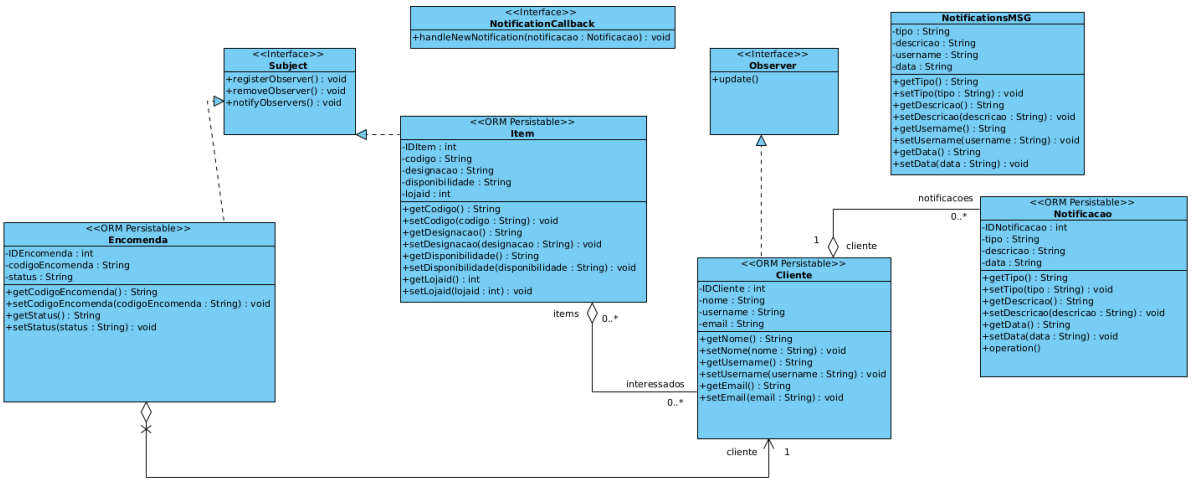

Figura 14: Diagrama de Classes do Microserviço de Notificações

O diagrama de classes mostra as principais entidades e seus relacionamentos:

- Cliente: Representa o usuário que recebe notificações. Contém informações como nome, email e username.
- Encomenda: Representa uma encomenda feita pelo cliente. Contém detalhes como código da encomenda, status, data de entrega e devolução.
- Item: Representa um item. Contém informações como código, designação, disponibilidade e loja associada.
- Notificação: Representa uma notificação enviada ao cliente. Contém detalhes como tipo, descrição, data e referência ao cliente e encomenda ou item associado.

Os objetos que estarão a ser observados pelo cliente são:

- Encomenda: Sempre que acontece uma alteração no status da encomenda, resultará numa notificação ao cliente.
- Item: Sempre que um item observado pelo cliente sofrer alterações em relação à sua disponibilidade, o cliente será notificado.

11.4. Diagrama ER de Notificações

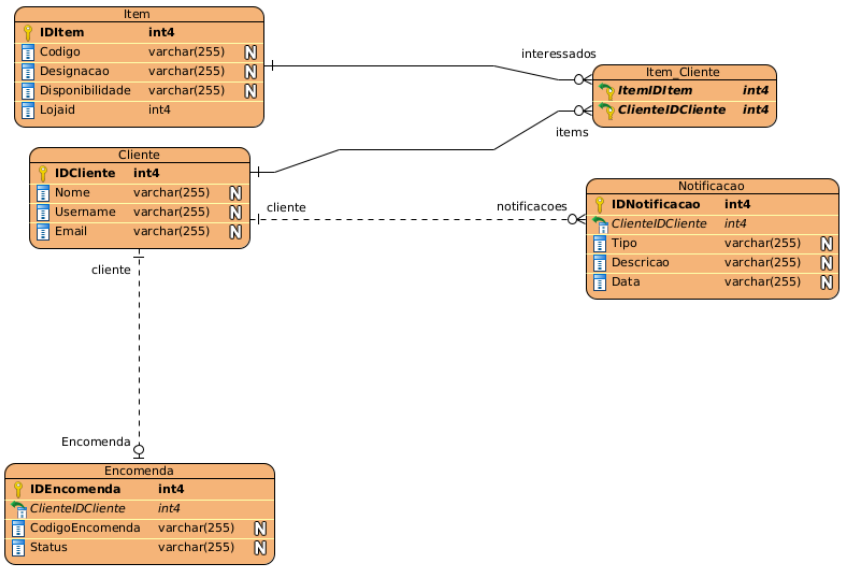

Figura 15: Diagrama ER do Microserviço de Notificações

O diagrama ER mostra a estrutura de dados subjacente ao microserviço de Notificações:

- A entidade Cliente: Armazena informações sobre os clientes que recebem notificações e estão interessados nos itens ou que possuam encomendas.
- A entidade Item: Armazena informações sobre os itens que podem ser alugados e observados pelos clientes.
- A entidade Encomenda: Armazena informações sobre as encomendas feitas pelos clientes.
- A entidade Notificação: Armazena informações sobre as notificações enviadas aos clientes, incluindo o tipo de notificação, descrição, data e referências às entidades Cliente, Encomenda e Item.

As relações entre estas entidades permitem que o sistema gerencie todo o ciclo de vida da notificação, desde a sua criação até à sua eliminação pelo cliente.
Este microsserviço atende ás necessidades dos clientes, notificando-o de toda a informação necessária sobre as suas encomendas, atualizações nos items em que está interessado.

11.5. API do Microserviço de Notificações

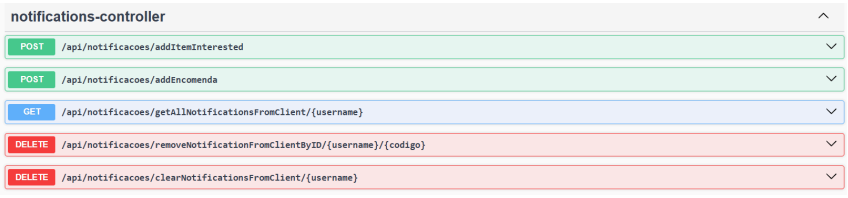

Figura 16: API do Microserviço de Notificações

As principais rotas da API do microserviço de Notificações são:

- **POST /api/notificacoes/addItemInterested**: Adiciona um cliente como interessado em um item específico.
- **POST /api/notificacoes/addEncomenda**: Adiciona uma nova encomenda e notifica o cliente sobre a mesma.

- **GET /api/notificacoes/getAllNotificationsFromClient/{username}**: Retorna todas as notificações de um determinado cliente.
- **DELETE /api/notificacoes/removeNotificationFromClientByID/{username}/{codigo}**: Remove uma notificação específica de um cliente, identificada pelo código.
- **DELETE /api/notificacoes/clearNotificationsFromClient/{username}**: Remove todas as notificações de um determinado cliente.

12. Microserviço dos Utilizadores

O microserviço de utilizadores gere a autenticação e autorização dos utilizadores na plataforma Pack My Bag. Este microserviço é essencial para garantir a segurança e personalização da experiência do utilizador. Implementado em Java Spring Boot, este microserviço oferece funcionalidades como:

- Registo de novos utilizadores: Permite que novos utilizadores, estilistas e técnicos se registem na plataforma.
- Autenticação de utilizadores: Garante que apenas utilizadores autenticados possam aceder a funcionalidades protegidas, através da validação de tokens.
- Gestão de perfis: Permite que os utilizadores atualizem as suas informações pessoais.
- Gestão de Permissões: Controla os níveis de acesso dos utilizadores, diferenciando entre utilizadores comuns e administradores.

12.1. TokenProvider e Autenticação

No microserviço de Utilizadores, o TokenProvider desempenha um papel crucial na segurança e personalização da aplicação. A sua principal função é gerar tokens de autenticação (JWT - JSON Web Tokens) que encapsulam informações essenciais do utilizador, como ID, nome e roles. Estes tokens são utilizados para:

- **Autenticação:** Verificar a identidade do utilizador em cada requisição feita à aplicação.
- **Autorização:** Determinar os níveis de acesso e permissões que cada utilizador possui, permitindo ou restringindo o acesso a determinadas funcionalidades ou áreas da aplicação.

12.1.1. Funcionamento no Frontend

No frontend, implementado em Vue.js, o auth-service é responsável por gerir o fluxo de autenticação utilizando os tokens fornecidos pelo TokenProvider. O processo funciona da seguinte maneira:

- **Login:** Quando um utilizador realiza o login, as suas credenciais são enviadas ao microserviço de Utilizadores, que valida as informações e, em caso de sucesso, gera um token de autenticação.
- **Armazenamento do Token:** O token gerado é armazenado no frontend, em localStorage, permitindo que o utilizador permaneça autenticado durante a sua sessão.
- **Requisições Autenticadas:** Em cada requisição subsequente à API, o frontend introduz o token no header da requisição no campo Authorization. Consequentemente, foi implementada uma função de middleware que faz a validação do token de forma a proteger o acesso a um conjunto de rotas. Esta funcão envia ao serviço de utilizadores o token em questão e faz a validação de acordo com o algoritmo implementado (HMAC256) e o secret.
- **Gestão de Permissões na View:** Com base nas informações presentes no token, o frontend ajusta dinamicamente a interface do utilizador, exibindo ou ocultando elementos e funcionalidades que requerem permissões específicas, e privando o acesso a certas páginas caso não esteja autenticado.

Esta abordagem garante que apenas utilizadores autenticados com as permissões adequadas possam aceder a determinadas funcionalidades, aumentando a segurança e a personalização da experiência do utilizador na plataforma.

12.2. Diagrama de Classes dos Utilizadores

Figura 17: Diagrama de Classes do Microsserviço de Utilizadores

O diagrama de classes mostra as principais entidades e seus relacionamentos:

- **Cliente:** Representa a entidade base para todos os utilizadores da plataforma, incluindo atributos como nome, username, email, e profileImage. Esta classe implementa a interface UserDetails, que facilita a integração com o sistema de autenticação do Spring Security.
- **Estilista:** Subclasse de Cliente, com atributos adicionais como rating, genero, e bio, representando profissionais especializados em estilo.
- **NormalCliente:** Subclasse de Cliente, usada para armazenar informações de clientes regulares, como morada, cartaoCredito, e nrTelemovel.
- **Técnico:** Subclasse de Cliente que é associada a uma Loja. Cada técnico está vinculado a uma loja específica através de uma relação direta.
- **Loja:** Representa as lojas armazenando informações como nome, marca, e localizacao.

Estes elementos trabalham em conjunto para gerenciar diferentes tipos de utilizadores na plataforma, assegurando integração segura com o sistema de autenticação e uma experiência personalizada para cada perfil.

12.3. Diagrama ER dos Utilizadores

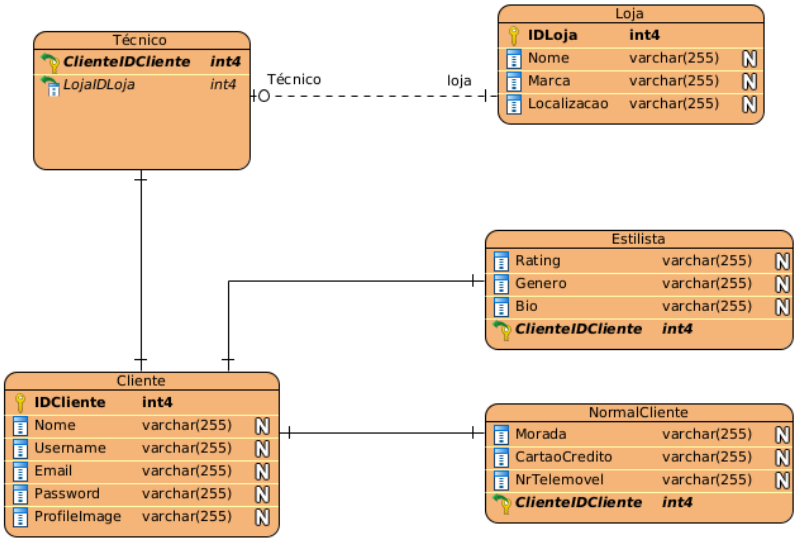

Figura 18: Diagrama de ER do Microsserviço de Utilizadores

O diagrama ER ilustra a estrutura de dados implementada no microserviço:

- A entidade Cliente representa os utilizadores da plataforma e é a tabela principal do modelo.
- Estilista e NormalCliente são entidades relacionadas diretamente com Cliente, permitindo especializações específicas para diferentes tipos de utilizadores.
- Técnico relaciona-se com Cliente e também com Loja, criando um vínculo entre profissionais e lojas específicas.
- A entidade Loja armazena informações sobre lojas, permitindo associá-las a vários técnicos.

Este microserviço garante uma gestão eficiente dos utilizadores da plataforma, oferecendo suporte para autenticação, personalização de perfis e controle de permissões.

12.4. API do Microserviço de Utilizadores

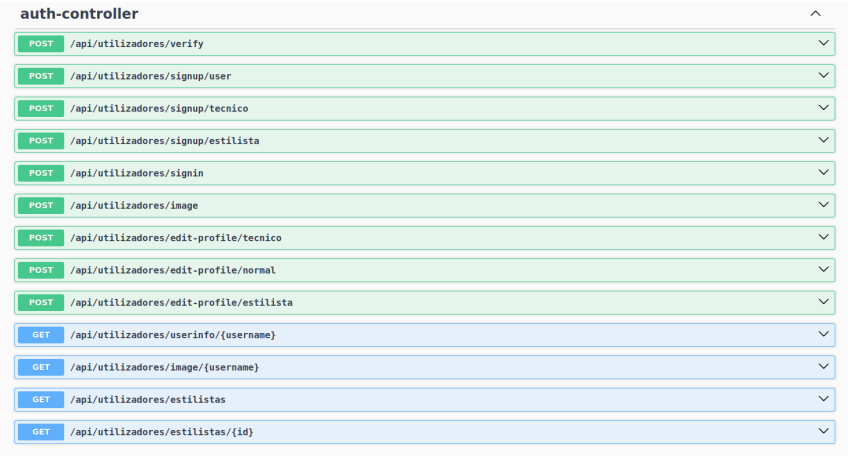

Figura 19: API calls do Microsserviço de Utilizadores

As principais rotas da API do microserviço de Utilizadores são:

- **POST /api/utilizadores/signup/user:** Regista um novo utilizador na plataforma.
- **POST /api/utilizadores/signup/estilista:** Regista um novo estilista na plataforma.
- **POST /api/utilizadores/signup/tecnico:** Regista um novo técnico na plataforma.
- **POST /api/utilizadores/signin:** Autentica um utilizador e emite um token de acesso.

- **POST /api/utilizadores/verify:** Verifica a validade de um token de autenticação.
- **GET /api/utilizadores/estilistas:** Retorna uma lista paginada de todos os estilistas registrados.
- **GET /api/utilizadores/estilistas/{id}:** Retorna as informações de um estilista específico pelo seu ID.
- **POST /api/utilizadores/image:** Faz upload da imagem de perfil de um utilizador.
- **GET /api/utilizadores/image/{username}:** Retorna a imagem de perfil de um utilizador específico.
- **GET /api/utilizadores/userinfo/{username}:** Retorna as informações detalhadas de um utilizador específico.
- **POST /api/utilizadores/edit-profile/normal:** Atualiza o perfil de um utilizador normal.
- **POST /api/utilizadores/edit-profile/estilista:** Atualiza o perfil de um estilista.
- **POST /api/utilizadores/edit-profile/tecnico:** Atualiza o perfil de um técnico.

13. Microserviço de Recomendações

O Microsserviço das Recomendações consiste num serviço de recomendações de outfits, e foi construído sobre a tecnologia *Java Spring Boot*. Este Microsserviço abrange tanto o cliente, como o estilista, e permite as seguintes funcionalidades:

- **Criação de um pedido de recomendação de outfit(s)**: O cliente poderá criar um pedido de recomendação de outfit(s) através do preenchimento e submissão de um *forms*. Apenas se tornará um pedido válido após a confirmação de pagamento;
- **Exibir a lista de pedidos de recomendação**: O estilista tem acesso a uma página com todos os pedidos de recomendação válidos que ainda não foram completados. Este terá acesso, em cada pedido, aos detalhes do *forms* preenchido pelo cliente;
- **Completar as recomendações dos pedidos correspondentes**: O estilista pode adicionar itens a cada recomendação, remover itens e criar uma descrição. A recomendação terá de ter obrigatoriamente itens e descrição para ele ter a possibilidade de completar o pedido;
- **Exibir a lista de recomendações completas**: O cliente tem acesso a uma página onde pode ver todas as recomendações que já recebeu, consoante os pedidos que efetuou;

13.0.1. Fluxo de realização do pedido

O utilizador preenche o formulário com os campos pedidos, sendo estes campos preferências pessoais em certos aspetos, número de *outfits*, *budget*. Quando este termina de preencher e submete, é associado um status de *pending*, e as informações são validadas e armazenadas na base de dados. Este estado tem uma duração máxima de 5 minutos, e após essa duração, se o pagamento não for confirmado, este pedido é invalidado e removido da base de dados. Caso seja pago dentro dos 5 minutos, o estado é alterado para pago, e o pedido é apresentado na página de pedidos e recomendações do estilista.

13.0.2. Fluxo de realização da recomendação

O pedido aparece na página de pedidos do estilista após o seu pagamento. Este irá poder adicionar e remover itens, utilizando para isso uma opção exclusiva para estilistas nas páginas dos items no catálogo, que também permite ver um resumo das várias recomendações por completar através de um pop-up. Este também terá de escrever uma descrição para a recomendação. Depois de ter isto tudo este poderá submeter a recomendação, que será validada e ficará com o estado completo.

13.1. Diagrama de Classes do Microsserviço das Recomendações

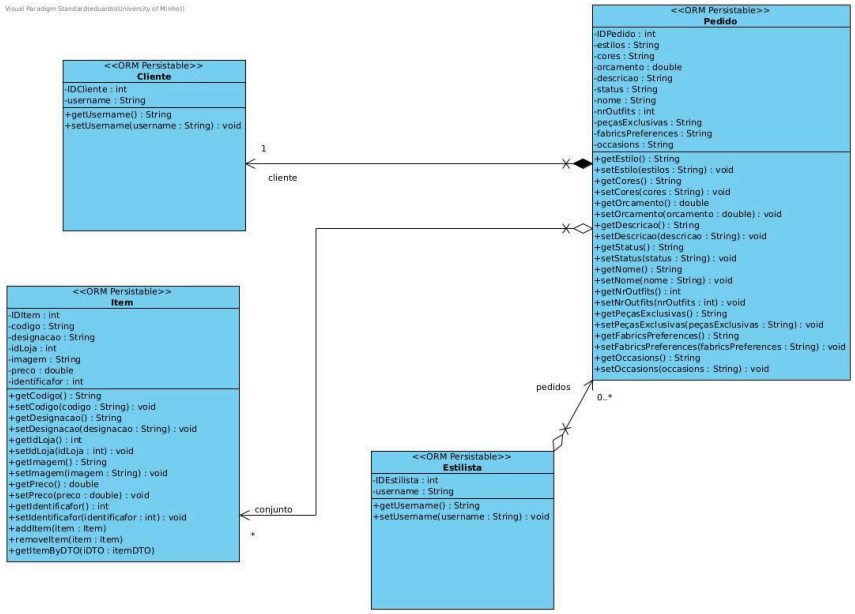

Figura 20: Diagrama de Classes do Microsserviço de Recomendações

O diagrama de classes representa as principais entidades deste Microsserviço, bem como as suas relações.

- **Cliente**: Representa o utilizador que realiza o pedido, e recebe as recomendações;
- **Estilista**: Representa o utilizador que completa as recomendações;
- **Item**: Representa as peças de roupa que são adicionadas as recomendações;
- **Pedido**: Regista os detalhes do pedido efetuado pelo Cliente, bem como o estado do pedido, e as informações da recomendação do estilista.

Estes elementos trabalham em conjunto para permitir que os clientes possam ter um apoio especializado na escolha de peças para aluguer, e os estilistas possam atender a esses pedidos de acordo com as exigências do cliente.

13.2. Diagrama ER do Microsserviço das Recomendações

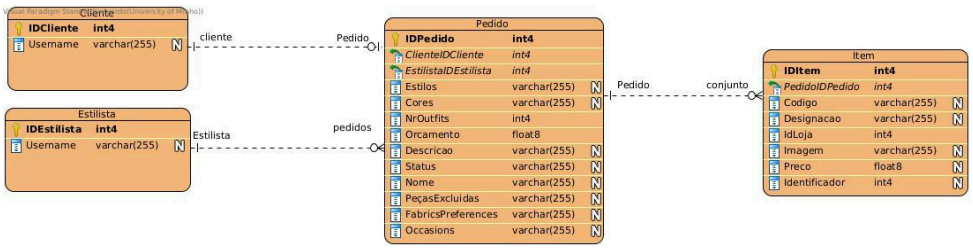

Figura 21: Diagrama ER do Microsserviço de Recomendações

O diagrama ER enfatiza a estrutura de dados subjacente ao microsserviço de Recomendações.

- A entidade *Cliente*, neste microsserviço, apenas é definida pelo *username*;
- A entidade *Estilista*, neste microsserviço, apenas é definida pelo *username*;
- A entidade *Item* é definida pelo seu código, id da loja, identificador no serviço do catálogo, imagem, preço e designação. Estes são necessários para o cliente poder adicionar os itens de uma recomendação completa ao carrinho;

- A entidade *Pedido* é definida pelo nome do pedido, e contém os dados do formulário preenchido pelo cliente, que contém informações como o número de *outfits*, o *budget*, as preferências de materiais, etc. Para além disso, possui os dados da recomendação do estilista, a descrição.

As relações entre estas entidades permitem que o sistema gerencie todo o processo da criação de um pedido, até ao pedido ser concluído e já possuir a recomendação do estilista.
Este microsserviço atende tanto ás necessidades dos clientes, que podem obter recomendações de *outfits* dos estilistas consoante as suas exigências, quanto aos estilistas que podem editar as recomendações enquanto não a sinalizam como completa.

13.3. API do Microsserviço das Recomendações

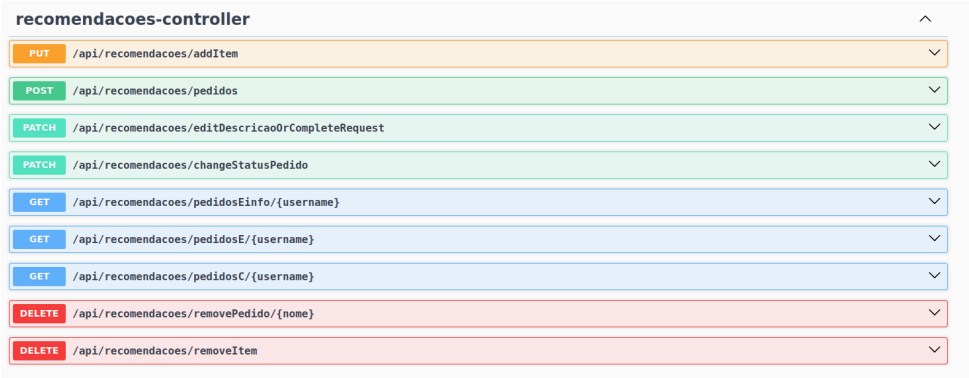

Figura 22: API do Microsserviço das Recomendações

As principais rotas da API deste serviço são:

- **GET /api/recomendacoes/pedidosE/{username}**: Retorna a um estilista a sua lista de pedidos/recomendações pagas e por completar;
- **GET /api/recomendacoes/pedidosEinfo/{username}**: Retorna a um estilista a sua lista de pedidos/recomendações pagas e por completar, com informações resumidas para o menu de adição de um item a uma recomendação nos itens do catálogo;
- **GET /api/recomendacoes/pedidosC/{username}**: Retorna a um cliente a sua lista de recomendações que já foram completadas pelo estilista;
- **POST /api/recomendacoes/pedidos**: Adiciona um pedido à estrutura de dados;
- **DELETE /api/recomendacoes/removePedido/{nome}**: Permite remover um pedido pendente que não foi dado como pago;
- **PUT /api/recomendacoes/addItem**: Adiciona um item ao conjunto de itens da recomendação de um pedido pago por completar;
- **DELETE /api/recomendacoes/removeItem**: Remove um item do conjunto de itens da recomendação de um pedido pago por completar;
- **PATCH /api/recomendacoes/editDescricaoOrCompleteRequest**: Permite ao estilista editar a descrição de uma recomendação e/ou completar um pedido;
- **PATCH /api/recomendacoes/changeStatusPedido**: Permite alterar o estado de um pedido, quando esta alteração é válida.

14. Arquitetura

Com base na ideia principal consolidada e nos requisitos levantados junto dos potenciais utilizadores, procedemos à elaboração do modelo de domínio, do diagrama de casos de uso e do diagrama de componentes. Estas etapas foram fundamentais para estruturar a arquitetura da aplicação PackMyBag, assegurando que todas as funcionalidades essenciais fossem contempladas e que a interação entre os diferentes componentes fosse eficiente e coerente. Esta abordagem permitiu-nos delinear claramente as entidades, os processos e as interações do sistema, servindo como guia para o desenvolvimento e implementação da plataforma.

14.1. Modelo de Domínio

O Modelo de Domínio representa a estrutura conceitual da plataforma web PackMyBag, incluindo as entidades principais e suas relações. Este modelo descreve como as peças de roupa, os clientes, os funcionários técnicos, as encomendas, e outras entidades interagem entre si no contexto do serviço de aluguer. Foca-se na organização lógica dos dados e na modelação das operações básicas que suportam o funcionamento da plataforma, como a gestão de peças, notificações, e acompanhamento do estado das encomendas.

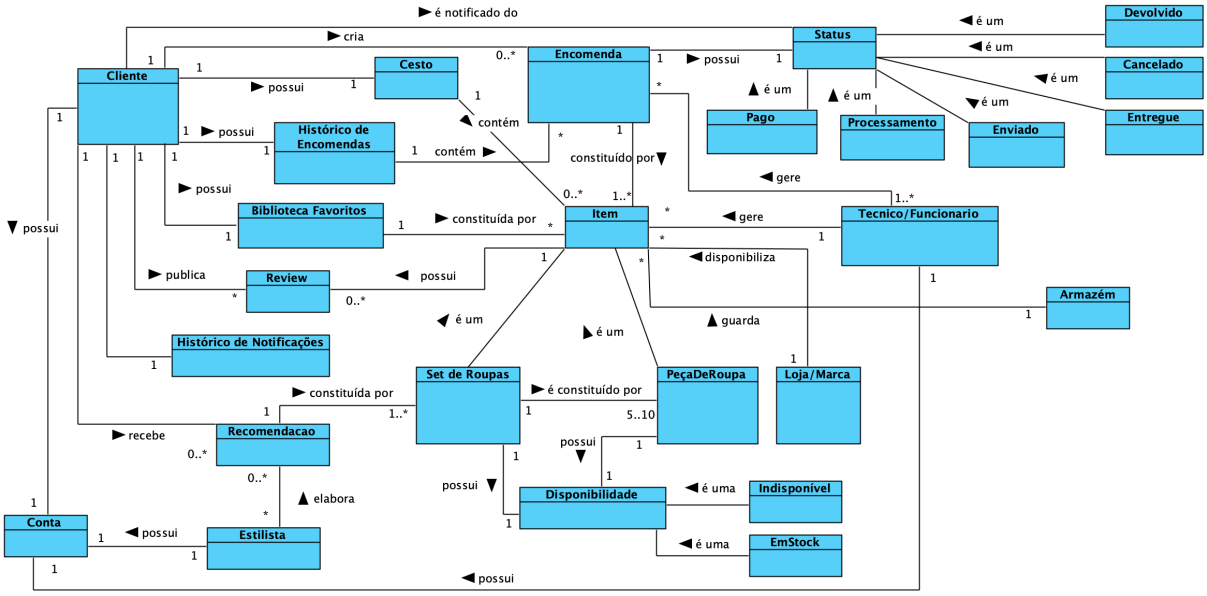

Figura 23: Modelo de Domínio

As principais entidades incluem:

- **Cliente:** Utilizador que aluga as roupas.
- **Cesto:** Entidade onde se encontra um conjunto de roupas selecionadas pelo cliente para alugar e proceder à sua encomenda.
- **Encomenda:** Representa a transação de aluguer, onde estarão presentes informações sobre o estado da encomenda, o lugar de destino, a duração de aluguer e o preço correspondente.
- **Item:** Uma peça de roupa/indumentária.
- **Biblioteca de Favoritos:** Armazena as peças favoritas do cliente.
- **Review:** Avaliação deixada pelo cliente sobre uma determinada peça.
- **Recomendação:** Sugestões de roupas com base no perfil do cliente.
- **SetDeRoupa:** Conjunto de peças de roupa selecionadas.
- **PecaDeRoupa:** Cada peça individual que compõe um conjunto.

- **Disponibilidade:** Condição da peça, se esta se encontra disponível ou não para alugar.
- **Conta:** Informações da conta do cliente.
- **Estilista:** Profissional que faz recomendações personalizadas.
- **Técnico/Funcionário:** Responsável pelos itens e encomendas de uma loja.
- **Loja/Marca:** Responsável por disponibilizar items aos clientes.
- **Histórico de Encomendas:** Contém informações sobre todas as encomendas realizadas pelo cliente.
- **Histórico de Notificações:** Contém informações sobre todas as notificações recebidas pelo cliente.

As interações entre essas entidades permitem que o sistema ofereça os serviços de aluguel de roupas e recomendações personalizadas aos clientes.

14.2. Diagrama de Use Cases

O Diagrama de Use Cases ilustra as interações entre os atores (como clientes, técnicos/funcionários e estilistas) e o PackMyBag. Este diagrama destaca as principais funcionalidades oferecidas pela aplicação, como o aluguer de roupa, gestão de favoritos, sugestões personalizadas, notificações sobre encomendas e estado de disponibilidade, entre outros. Serve como base para identificar e organizar os requisitos funcionais do sistema, fornecendo uma visão clara dos processos suportados.

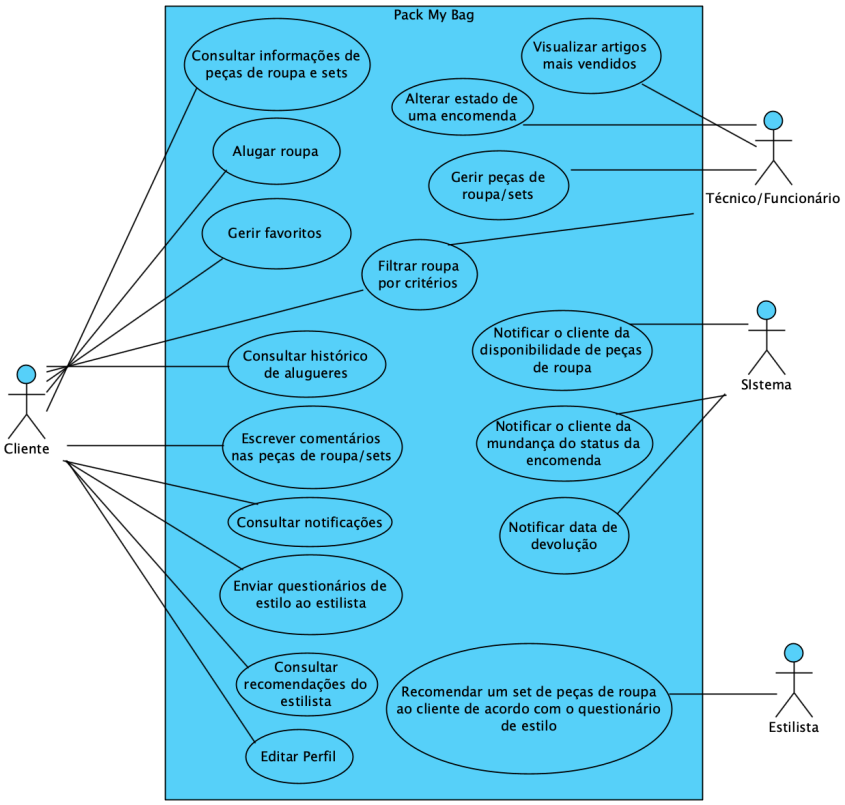

Figura 24: Diagrama de Use Cases

Conforme descrito na Figura 24, as principais funcionalidades do lado do cliente incluem:

- Consultar informações sobre peças de roupa e conjuntos disponíveis
- Alugar roupas
- Gerenciar sua coleção de favoritos
- Consultar o histórico de aluguéis
- Escrever comentários sobre as peças alugadas
- Consultar notificações do sistema
- Enviar questionários de estilo para o Estilista

- Editar seu perfil

Do lado do Técnico/Funcionário, as principais ações incluem:

- Alterar o estado de uma encomenda
- Gerir o stock de roupas de uma loja.
- Visualizar artigos mais vendidos.

Já o Estilista é responsável por:

- Recomendar um conjunto de peças de roupa de acordo com o questionário de estilo do Cliente

Adicionalmente, o sistema PackMyBag desempenha um papel importante, sendo responsável por:

- Notificar o cliente sobre a disponibilidade de peças de roupa
- Notificar o cliente sobre mudanças no status da encomenda
- Notificar a data de devolução

Ou seja, o sistema monitora alterações de estado e proativamente informa o usuário sobre essas atualizações relevantes.

14.3. Diagrama de Componentes

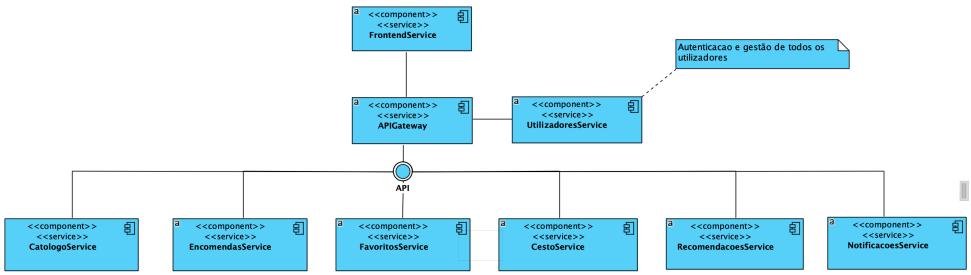

Figura 25: Diagrama de Componentes

O diagrama de componentes da aplicação Pack My Bag ilustra a sua arquitetura baseada em microsserviços. Esta abordagem modular oferece vantagens significativas em termos de escalabilidade, flexibilidade, e manutenção.
Esta arquitetura permitiu que cada componente fosse desenvolvido, implementado e escalado independentemente. Isto resultou numa maior agilidade de desenvolvimento, permitindo futuras atualizações mais rápidas e eficientes sem afetar o funcionamento de outras partes do sistema. Adicionalmente, ao utilizar esta arquitetura o isolamento de falhas é aprimorado, pois uma falha num único microsserviço não compromete a aplicação inteira. Finalmente, a organização em microsserviços permitiu que os membros da equipa de desenvolvimento trabalhassem em paralelo, melhorando a produtividade. Contudo, a adoção de uma arquitetura distribuída como também apresenta as suas desvantagens, como o maior tempo de latência na resposta a pedidos, uma vez que são necessárias consultas a mais do que um serviço.
Os microsserviços representados no diagrama interagem de forma coordenada para proporcionar a experiência completa ao utilizador. Para melhor esclarecer o funcionamento, vamos descrever brevemente a função de cada microsserviço:

- **Frontend:** Responsável pela interface de utilizador (UI) e pela interação com o utilizador. Trata das requisições do lado cliente e renderiza a interface.
- **APIGateway:** Atua como um ponto de entrada único para todas as requisições, roteando-as para o(s) microsserviço(s) apropriado(s). Fornece segurança e gestão de tráfego.

- **UtilizadoresService:** Gere os dados dos utilizadores, incluindo registo, autenticação, perfil e gestão de preferências.
- **CatalogoService:** Responsável pela gestão do catálogo de roupas, incluindo pesquisa, filtragem, e detalhe dos produtos.
- **EncomendasService:** Gere as encomendas dos utilizadores, do processo de criação até à entrega.
- **NotificaçõesService:** Envia notificações aos utilizadores sobre o estado das encomendas, disponibilidade de roupas e outras informações relevantes.
- **RecomendaçõesService:** Fornece recomendações personalizadas de roupas aos utilizadores com base em um inquérito preenchido e enviado ao estilista.
- **FavoritosService:** Gere os itens de favoritos guardados pelo cliente.
- **CartService:** Responsável pela funcionalidade do carrinho de compras do cliente. Isto inclui adicionar itens ao carrinho, atualizar quantidades, remover itens e calcular o custo total dos itens no carrinho (geração de pagamentos e cancelamento dos mesmos após um período de tempo). Prepara os dados do carrinho para a criação da encomenda.

Esta abordagem modular torna a aplicação PackMyBag mais resiliente, escalável e fácil de manter, atendendo melhor às necessidades em constante evolução dos clientes.

15. Docker

A utilização do Docker na aplicação PackMyBag foi essencial para a containerização dos diversos componentes e microserviços, garantindo um ambiente consistente de desenvolvimento, teste e implantação.

15.1. Criação de Imagens Docker

Para cada microserviço da aplicação, criámos um Dockerfile específico que define o processo de construção da imagem Docker correspondente. Essas imagens incluem o código-fonte, bibliotecas e todas as dependências necessárias para a execução do serviço. A criação dessas imagens permite que a aplicação seja implantada de forma consistente em diferentes ambientes, facilitando a escalabilidade e a portabilidade.
Posteriormente, fizemos push dessas imagens para um repositório no Docker Hub. E com recurso à ferramenta Ansible foi possível automatizar e fazer pull dessas imagens para hospedar a aplicação num cluster da Google Cloud. Isso possibilitou que os nossos serviços fossem facilmente geridos e escalados na nuvem. Os detalhes dessa integração são abordados num capítulo dedicado ao Ansible.Seção 16

15.2. Docker Compose

O Docker Compose foi utilizado para orquestrar todos os contêineres dos microserviços da aplicação. Através de um único arquivo de configuração, o *docker-compose.yml*, definimos como os contêineres interagem entre si, quais portas devem ser expostas, as dependências entre serviços e as redes utilizadas.

15.2.1. Configuração do APIGatewayService no Docker Compose

Um aspecto crucial foi a configuração do *apigatewayservice*. No ambiente Docker, os contêineres comunicam-se através de resoluções de nomes de serviço, utilizando o DNS interno do Docker. Isso significa que, dentro da nossa rede Docker, podemos referenciar outros serviços pelos seus nomes definidos no docker-compose.yml.
No caso do *apigatewayservice*, configuramos da seguinte forma:

```
apigatewayservice:
  build:
    context: ./API-GATEWAY
  container_name: apigateway-container
  restart: always
  ports:
    - '8888:8888'
  environment:
    - CATALOGO_SERVICE_URL=http://catalogoservice:8081/api/catalogo
    - ENCOMENDA_SERVICE_URL=http://encomendasservice:8082/api/encomendas
    - FAVORITOS_SERVICE_URL=http://favoritosservice:8083/api/favoritos
    - RECOMENDACOES_SERVICE_URL=http://recomendacoesservice:8084/api/recomendacoes
    - CESTO_SERVICE_URL=http://cartservice:8085/api/cart
    - NOTIFICACOES_SERVICE_URL=http://notificacoesservice:8086/api/notificacoes
    - UTILIZADORES_SERVICE_URL=http://utilizadoresservice:8087/api/utilizadores
  depends_on:
    - catalogoservice
    - encomendasservice
    - favoritosservice
  networks:
    - packmybag
```

Aqui, as variáveis de ambiente apontam para os URLs dos microserviços, utilizando os nomes dos serviços definidos no Docker Compose. Por exemplo, *http://catalogoservice:8081/api/catalogo* onde *catalogoservice* é o nome do serviço que o Docker resolve internamente para o endereço IP correto do contêiner correspondente.
Esta configuração é fundamental porque os endereços IP dos contêineres podem mudar sempre que eles são reiniciados, mas os nomes dos serviços permanecem constantes dentro da rede criada pelo Docker Compose.

15.2.2. Rede PackMyBag

Para permitir que os contêineres comuniquem-se entre si utilizando os nomes dos serviços, criámos uma rede Docker personalizada chamada packmybag. Ao definir uma rede personalizada, garantimos que todos os contêineres conectados a ela possam resolver os nomes dos serviços uns dos outros, facilitando a comunicação interna.
No final do arquivo docker-compose.yml, definimos a rede:

```
networks:
  packmybag:
    driver: bridge
```

E em cada serviço, incluímos:

```
networks:
  - packmybag
```

A rede packmybag utiliza o driver padrão bridge, que é adequado para casos em que os contêineres precisam comunicar num ambiente isolado. Isso assegura que os nossos microserviços possam interagir corretamente sem interferências externas.

16. Ansible

Para hospedar a aplicação PackMyBag num cluster da Google Cloud, utilizamos o Ansible como ferramenta de automação e gestão de configuração. O Ansible permitiu-nos automatizar todo o processo de implantação, desde a criação do cluster até à configuração dos microserviços, garantindo uma implementação consistente e eficiente num ambiente de produção.
Ao utilizar o Ansible, conseguimos orquestrar os diversos componentes da aplicação de forma uniforme. Foi possível provisionar o cluster Kubernetes na Google Cloud Platform e implementar todos os microserviços containerizados de maneira automatizada. Isso assegurou que cada serviço fosse corretamente configurado e integrado no cluster, facilitando a escalabilidade e manutenção da aplicação.
Um ponto crucial foi assegurar que o frontend da aplicação conseguisse comunicar-se com a API Gateway. De modo a automatizar o processo, no ansible desenvolvemos uma tarefa que irá fazer o get do ip externo da API Gateway quando este estiver disponível e será escrito esse valor no inventário. Depois ao criar o contâiner no pod de frontend vamos buscar essa variável e definímos a mesma como variável de ambiente (VUE_API_DOMAIN). No Dockerfile do frontend incluímos a execução de um script *generate_env-config.sh*. Este script é responsável por gerar dinamicamente o arquivo *env--config.js*, que contém a variável VUE_API_DOMAIN com o endereço do API Gateway.
Ao executar o *generate_env-config.sh* durante o processo de build da imagem Docker do frontend, conseguimos inserir dinamicamente o endereço IP do APIGateway na configuração do frontend. Isso é essencial porque, em ambientes de produção ou em clusters onde os endereços podem ser atribuídos dinamicamente, é necessário que o frontend saiba sempre para onde direcionar as requisições à API.Sem o *generate_env-config.sh*, o frontend não teria como saber o endereço correto do API Gateway, o que impediria a comunicação com a API e, consequentemente, o funcionamento adequado da aplicação.
Em resumo, o Ansible foi fundamental para a implantação automatizada da nossa aplicação na Google Cloud, e o uso do *generate_env-config.sh* no Dockerfile do frontend assegurou que o frontend conseguisse comunicar-se corretamente com a API através do API Gateway.

17. Deployment

De seguida encontra-se a arquitetura do deploy da aplicação web 'Pack My Bag' para um cluster da Google Cloud, sendo que para isso foi utilizado o serviço de kubernetes da Google Cloud (GKE). Como podemos observar o serviço de frontend que atua como LoadBalancer irá reencaminhar o tráfego para o pod do frontend. Este pod contém um container da imagem da aplicação que foi contruída com Vue.js e Nginx, servidor HTTP, que serve o conteúdo estático e é amplamente usado em produção devido à sua capacidade de suportar múltiplas conexões em simultâneo. Os pedidos irão ser encaminhados para o serviço da API Gateway e o Pod respetivo irá rotear para um ou mais dos pods dos microserviços. O serviço do API Gateway irá atuar como load balancer, pelo que teremos acesso externo através de um IP externo à semelhança do serviço de frontend, e deste modo, poderá receber tráfego do exterior do cluster. Por sua vez, o tráfego será reencaminhado para os serviços internos do cluster (os microserviços), que serão do tipo default ClusterIP e que desta forma possuirão um endereço IP virtual acessível apenas pelos pods dentro do cluster, permitindo a comunicação entre os componentes internos.

Os microserviços ao receberem os pedidos nas suas respetivas portas irão comunicar-se com as suas respetivas bases de dados que se encontrarão no pod de postgres. O pod de postgres, que terá um volume associado, irá fazer um claim (solicitação) da quantidade de armazenamento desejada e os tipos de acesso. Já o StorageClass que estará associado ao PVC irá definir as características do

armazenamento, neste caso como a classe é standard os dados serão armazenados em discos persistens padrão (HDD). Ao criar o deployment do postgres foram criadas e povoadas as bases de dados em runtime.

Por último, no pod do Kafka foi criado um container com uma imagem pública confluentinc/cp-kafka e tag 7.4.0, que serviu para implementar um broker de comunicação entre os serviços Catalogo--Notificações, e Encomendas-Notificações. Ou seja, sempre que ocorra um pedido de atualização da disponibilidade do item, ou quando um cliente compre um item que consequentemente se esgota, é enviado um objeto do item com a nova disponibilidade. Esse objeto ser enviado para o serviço do Kafka na porta 9092, e, por sua vez, o serviço de Notificações recebe os objetos que vêm do broker e dependendo da tag irá construir um tipo diferente de notificação. Logo, em primeiro, irá verificar quem são os clientes interessados nesse objeto e que se encontram na base de dados, e por fim notifica-os em tempo real (fazendo um post do objeto da mensagem da notificação para a API Gateway) e guarda as notificações novas na base de dados. O serviço de encomendas terá um comportamento semelhante ao de Catálogo no envio de objetos para o broker, mas no que toca ao envio de objetos Encomenda, onde passará informação como o número de dias restantes para o cliente entregar a encomenda, ou quando o status da encomenda se altera.

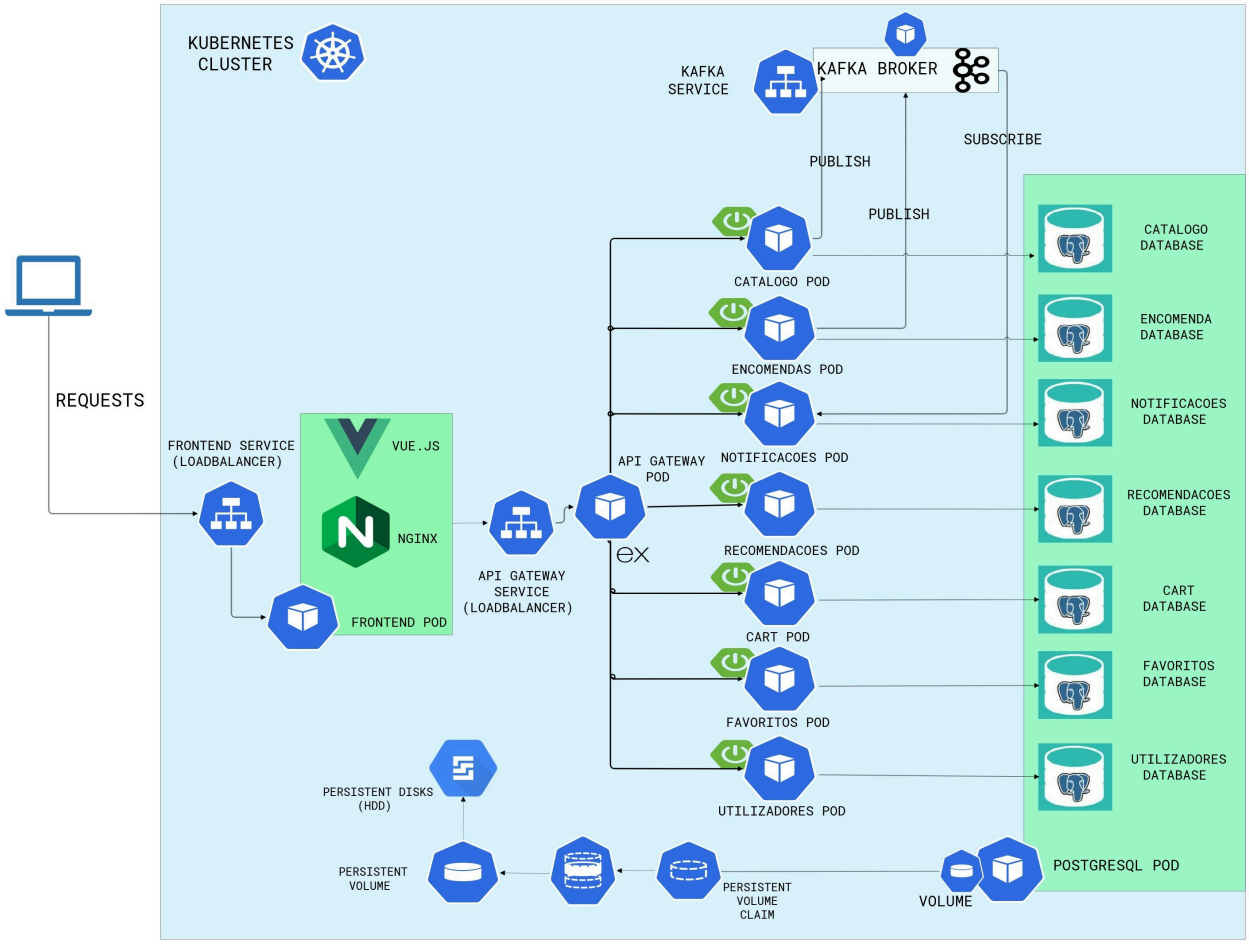

Figura 26: Arquitetura do infraestrutura da aplicação na cloud

18. Conclusão

A realização deste projeto representou um percurso intenso de aprendizagem e superação de desafios. Desde o início, sabíamos que a implementação da aplicação PackMyBag exigiria a incorporação de tecnologias com as quais não estávamos totalmente familiarizados, como Spring Boot, Vue.js e Express.js. Esta necessidade impulsionou-nos a investir tempo e esforço significativos na aquisição de novos conhecimentos e competências técnicas.

Ao longo do processo, enfrentamos diversas dificuldades inerentes à adoção de uma arquitetura baseada em microsserviços. Tivemos de compreender profundamente como esses componentes interagem, garantindo a comunicação eficaz entre si e mantendo a integridade e a performance da aplicação. A integração destes microsserviços através da API Gateway, bem como a implementação de um frontend responsivo e intuitivo com Vue.js, foram etapas desafiadoras que ampliaram consideravelmente a nossa experiência em desenvolvimento web.

Paralelamente, o levantamento de requisitos junto dos potenciais utilizadores forneceu insights valiosos que orientaram o desenvolvimento das funcionalidades da plataforma. Foi essencial assegurar que a nossa solução fosse verdadeiramente alinhada com as necessidades e expectativas do público-alvo, oferecendo uma experiência de utilizador enriquecedora e diferenciada.

Apesar do tempo limitado, dedicámo-nos intensamente para construir uma plataforma web robusta, funcional e escalável. O resultado é uma aplicação que não só atende aos objetivos propostos mas também estabelece uma base sólida para futuras expansões e melhorias.

Em suma, acreditamos que a plataforma tem o potencial de inovar o mercado de aluguer de roupas em Portugal, proporcionando aos utilizadores uma solução conveniente, personalizada e eficiente para as suas necessidades durante viagens.

Bibliografia

[1] [Online]. Disponível em: https://swagger.io/

[2] [Online]. Disponível em: https://vuejs.org/

[3] [Online]. Disponível em: https://docs.spring.io/spring-boot/index.html

[4] [Online]. Disponível em: https://medium.com/@ozziefel/kafka-in-microservices-architecture--enabling-scalable-and-event-driven-systems-7ff474de49f4

[5] [Online]. Disponível em: https://docs.docker.com/compose/gettingstarted/

19. Anexos

19.1. Requisitos Funcionais

Requisito #: 1	Tipo: Funcional Use Cases #: Consultar informações de peças de roupa e sets
Descrição	**Um cliente deverá ser capaz de consultar o inventário onde estarão disponibilizados todos itens de peças de roupa e sets de roupa.**
Origem	Cliente
Prioridade	Must

Requisito #: 2	Tipo: Funcional Use Cases #: Consultar informações de peças de roupa e sets
Descrição	**O cliente deve ser capaz de visualizar o inventário com itens caracterizados por uma imagem representativa do produto, a sua designação e o preço.**
Origem	Cliente
Prioridade	Must

Requisito #: 3	Tipo: Funcional Use Cases #: Consultar informações de peças de roupa e sets
Descrição	**O cliente deve ser capaz de visualizar cada item numa página individual, onde haverá uma imagem representativa do produto, a sua designação, os tamanhos e as cores disponíveis e o preço de aluguer.**
Origem	Cliente
Prioridade	Must

Requisito #: 4	Tipo: Funcional Use Cases #: Alugar roupa
Descrição	**O cliente deverá ser capaz de adicionar uma peça de roupa ou set no seu cesto virtual.**
Origem	Cliente
Prioridade	Must

Requisito #: 5	Tipo: Funcional Use Cases #: Alugar roupa
Descrição	**O cliente deverá ser capaz de remover uma peça de roupa ou set no seu cesto virtual.**
Origem	Cliente

Requisito #: 5	Tipo: Funcional	Use Cases #: Alugar roupa
Prioridade	Must	

Requisito #: 6	Tipo: Funcional	Use Cases #: Alugar roupa
Descrição	**O cliente deverá ser capaz de efetuar o aluguer do conjunto de itens que se encontram no cesto virtual, fornecendo para isso o destino de recolha, o período de aluguer e informações de pagamento.**	
Origem	Cliente	
Prioridade	Must	

Requisito #: 7	Tipo: Funcional	Use Cases #: Comparar preço da viagem vs aluguer
Descrição	**O sistema deve simular o preço da viagem com mala, que varia de acordo com o peso das roupas que se encontram no cesto virtual do cliente, adicionalmente com a distância, a taxa física e a taxa de emissões.**	
Origem	Cliente	
Prioridade	**Could**	

Requisito #: 8	Tipo: Funcional	Use Cases #: Gerir coleção de favoritos
Descrição	**O cliente deverá ser capaz de adicionar um item(peça de roupa ou set) para a sua biblioteca pessoal de produtos favoritos.**	
Origem	Cliente	
Prioridade	Must	

Requisito #: 9	Tipo: Funcional	Use Cases #: Gerir coleção de favoritos
Descrição	**O cliente deverá ser capaz de remover um item(peça de roupa ou set) da sua biblioteca pessoal de produtos favoritos.**	
Origem	Cliente	
Prioridade	Should	

Requisito #: 10	Tipo: Funcional	Use Cases #: Filtrar roupa por critérios
Descrição	**O cliente deverá ser capaz de visualizar itens de uma das marcas disponíveis no sistema, género, o seu preço, e o seu tamanho.**	
Origem	Cliente	
Prioridade	Must	

Requisito #: 11	Tipo: Funcional	Use Cases #: Consultar histórico de alugueres
Descrição	**O cliente deverá ser capaz de visualizar todas as encomendas de aluguer que efetuou até ao momento.**	
Origem	Cliente	
Prioridade	Should	

Requisito #: 12	Tipo: Funcional	Use Cases #: Consultar histórico de alugueres
Descrição	**O histórico deverá mostrar a data da realização da encomenda, o período em que foi alugada, a designação dos produtos e o local em que foi ou irá ser recolhido.**	
Origem	Cliente	
Prioridade	Should	

Requisito #: 13	Tipo: Funcional	Use Cases #: Publicar review de roupas
Descrição	**O cliente tem a opção de fazer uma review em peças de roupa que alugou.**	
Origem	Cliente	
Prioridade	Must	

Requisito #: 14	Tipo: Funcional	Use Cases #: Publicar review de roupas
Descrição	**O cliente pode editar as reviews que fez.**	
Origem	Cliente	
Prioridade	Must	

Requisito #: 15	Tipo: Funcional	Use Cases #: Publicar review de roupas
Descrição	**O cliente pode eliminar as reviews que fez.**	
Origem	Cliente	
Prioridade	Must	

Requisito #: 16	Tipo: Funcional	Use Cases #: Consultar notificações
Descrição	**O cliente deve ser capaz de aceder a uma aba/secção na plataforma onde pode visualizar as notificações recebidas.**	
Origem	Cliente	
Prioridade	Must	

Requisito #: 17	Tipo: Funcional	Use Cases #: Consultar recomendações do estilista
Descrição	**O cliente pode pagar por um serviço extra, onde é capaz de aceder a uma aba/secção que lhe permite receber recomendações de um estilista.**	
Origem	Cliente	
Prioridade	Must	

Requisito #: 18	Tipo: Funcional	Use Cases #: Consultar recomendações do estilista
Descrição	**O cliente deve ser capaz de enviar um pedido de recomendação a um estilista, respondendo a um formulário.**	
Origem	Cliente	
Prioridade	Should	

Requisito #: 19	Tipo: Funcional	Use Cases #: Recomendar outfits
Descrição	**O estilista deve ser capaz de visualizar os formulários dos pedidos dos clientes.**	

Requisito #: 19	Tipo: Funcional	Use Cases #: Recomendar outfits
Origem	Estilista	
Prioridade	Should	

Requisito #: 20	Tipo: Funcional	Use Cases #: Recomendar outfits
Descrição	**O estilista deve ser capaz de enviar a clientes combinações de peças feitas por ele.**	
Origem	Estilista	
Prioridade	Should	

Requisito #: 21	Tipo: Funcional	Use Cases #: Alterar estado de uma encomenda
Descrição	**O funcionário deve ser capaz de alterar o estado de uma encomenda.**	
Origem	Funcionário	
Prioridade	Must	

Requisito #: 22	Tipo: Funcional	Use Cases #: Gerir peças de roupas
Descrição	**O funcionário tem de ter permissão para adicionar e retirar peças do inventário.**	
Origem	Funcionário	
Prioridade	Must	

Requisito #: 23	Tipo: Funcional	Use Cases #: Filtrar roupa por critérios
Descrição	**O funcionário deve ser capaz de procurar por peças por determinados critérios como o "número de encomendas" para ser mais fácil de resolver problemas relacionados com a gestão de itens.**	
Origem	Funcionário	
Prioridade	**Could**	

Requisito #: 24	Tipo: Funcional	Use Cases #: Notificar o cliente da disponibilidade de peças de roupa
Descrição	**O sistema deve notificar o cliente, que pediu para ser avisado, quando determinada peça se encontrar novamente disponível para aluguer, disponibilizando a designação do item e a nova disponibilidade.**	
Origem	Sistema	
Prioridade	Should	

Requisito #: 25	Tipo: Funcional	Use Cases #: Notificar estado de trânsito das peças de roupa
Descrição	**O sistema deve notificar o cliente, com informações sobre a sua encomenda sempre que existe uma mudança de estado da encomenda.**	
Origem	Sistema	
Prioridade	Must	

Requisito #: 26	Tipo: Funcional Use Cases #: Notificar sobre a data de devolução
Descrição	**O sistema deve notificar o cliente que a data de devolução da roupa se está a aproximar, 48h e 24h antes).**
Origem	Sistema
Prioridade	Must

Requisito #: 27	Tipo: Funcional Use Cases #: Aplicação de coima
Descrição	**Caso a roupa não seja devolvida, o sistema aplica uma coima diária, removendo dinheiro da caução.**
Origem	Sistema
Prioridade	Must

Requisito #: 28	Tipo: Funcional Use Cases #: Recomendar outfits conforme o conjunto
Descrição	**O sistema deve recomendar roupas ao cliente tendo em conta o seu estilo, alugueres prévios e o destino para que este vai viajar.**
Origem	Estilista
Prioridade	Must

Requisito #: 29	Tipo: Funcional Use Cases #: Autenticar na plataforma
Descrição	**O cliente, o estilista e o técnico devem ser capazes de fazerem login no sistema, acedendo à sua vista da aplicação e funcionalidades correspondentes.**
Origem	Cliente, Estilista, Técnico
Prioridade	Must

Requisito #: 30	Tipo: Funcional Use Cases #: Autenticar na plataforma
Descrição	**O cliente deve ser capaz de criar uma conta na plataforma.**
Origem	Cliente
Prioridade	Must

19.2. Requisitos Não Funcionais

Requisito #: 31	Tipo: Performance Use Cases #
Descrição	**A aplicação deve ser capaz de escalar de acordo com o número de utilizadores.**
Origem	Cliente
Prioridade	Should

Requisito #: 32	Tipo: Disponibilidade Use Cases #
Descrição	**A plataforma deverá ter uma alta disponibilidade, pelo que deve funcionar corretamente todo o tempo.**
Origem	Cliente

Requisito #: 32	Tipo: Disponibilidade Use Cases #
Prioridade	Must

Requisito #: 33	Tipo: Performance Use Cases #
Descrição	**As reviews dadas aos itens podem ser classificadas de uma a cinco estrelas (obrigatoriamente) e com uma avaliação textual opcional.**
Origem	Cliente
Prioridade	Must

Requisito #: 34	Tipo: Usabilidade Use Cases #
Descrição	**O sistema deve ter uma interface atrativa e de fácil usabilidade de modo a que os clientes consigam efetuar os seus alugueres sem constrangimentos.**
Origem	Cliente
Prioridade	Must

Requisito #: 35	Tipo: Manutenção e Suporte Use Cases #
Descrição	**O sistema deve permitir a adição de novas funcionalidades sem interromper o funcionamento atual.**
Origem	Cliente
Prioridade	Must

Requisito #: 36	Tipo: Segurança Use Cases #
Descrição	**O sistema terá uma parte com acesso autorizado e outra com acesso não autorizado sendo que para realizar compras terá de efetuar um processo de autenticação.**
Origem	Cliente
Prioridade	Must

Requisito #: 37	Tipo: Cultural e Politico Use Cases #
Descrição	**O sistema deverá ter suporte ao idioma Inglês.**
Origem	Cliente
Prioridade	Must

Requisito #: 38	Tipo: Functional Use Cases #
Descrição	**Os tipos de notificação do estado de encomenda devem conter Enviado, Não Entregue, Devolvido, Entregue.**
Origem	Cliente
Prioridade	Must

Requisito #: 39	Tipo: Functional Use Cases # Alugar roupa
Descrição	**O destino de recolha da encomenda deverá ser verificada pelo sistema.**
Origem	Equipa desenvolvimento
Prioridade	Must