UFPR - TADS
Trabalho de DS152 - DAC
Empresa Aérea

Requisitos Funcionais

O objetivo deste trabalho é o desenvolvimento de um sistema de Gestão de Empresa Aérea usando Angular e Java Spring, baseado na arquitetura de microsserviços.

O sistema possui 2 perfis de acesso:

- **Cliente**: usuários com esse perfil são os clientes da agência;
- **Funcionário**: usuários com esse perfil são funcionários da agência;

Os requisitos funcionais são apresentados a seguir (CRUD significa - Inserir, Remover, Atualizar e Listar todos).

- **R01: Autocadastro** - Um cliente ainda não cadastrado na empresa aérea pode se cadastrar. Ele precisa informar: CPF, Nome, E-mail, Rua/Número, Complemento, CEP, Cidade e Estado. O Cliente inicia com 0 milhas, e essas milhas podem ser usadas para comprar passagens. A senha do usuário é um número de 4 dígitos, aleatório, que deve ser enviado por e-mail;
- **R02: Efetuar Login/Logout** - Login com e-mail/senha, todas as demais funcionalidades não podem ser acessadas sem um login com sucesso;

PERFIL CLIENTE

- **R03: Mostrar Tela Inicial de Cliente** - Deve apresentar um menu, com as operações que podem ser efetuadas pelo cliente, e seu saldo atual em milhas. Além disso, em formato de tabela, deve-se apresentar todas as reservas (que estão somente reservadas), os voos feitos e cancelados. Deve-se mostrar: data/hora, Aeroporto Origem, Aeroporto Destino, ordenado por data/hora. Deve-se ter uma opção para ver a reserva (R04), e uma opção para cancelar reserva (R08);
- **R04: Ver Reserva** - Este requisito vem do R03 que mostra todos os dados da reserva: data/hora, código, origem, destino, valor gasto em reais, milhas gastas, estado da reserva;
- **R05: Comprar de Milhas** - O cliente pode comprar milhas. A proporção é sempre 1 milha a cada R$ 5,00 (essa proporção não muda). Todas as transações de compra de milhas

devem ser registradas: data/hora, valor em reais, quantidade de milhas compradas e uma descrição "COMPRA DE MILHAS";

- **R06: Consultar Extrato de Milhas** - Deve ser mostrado um extrato, em forma tabular, contendo a quantidade atual de milhas do cliente, bem como todas as compras efetuadas: dia/hora, valor em reais, quantidade de milhas compradas, e uma descrição "COMPRA DE MILHAS". Também devem ser apresentadas as milhas usadas nos voos, mostrando: dia/hora do voo, quantidade de milhas usadas e na descrição origem/destino, da seguinte forma "CWB->GRU" ;
- **R07: Efetuar Reserva** - O cliente pode reservar os voos.
    - O sistema deve apresentar uma tela de busca mostrando um campo aeroporto origem e aeroporto destino (ambos podem ser vazios);
    - Ao clicar em "Buscar", devem ser apresentados todos os voos, a partir da data atual, que casam com a busca solicitada, em formato de tabela;
    - O cliente então pode selecionar um desses voos para ir para a próxima tela. Na próxima tela devem aparecer os dados do voo: Origem, Destino, Data/Hora e Preço do Assento; além disso deve apresentar seu saldo de milhas;
    - Nesta tela o cliente escolhe quantas passagens vai comprar e deve ser calculado o valor total, bem como a quantidade de milhas necessárias para comprar as passagens;
    - Então o cliente pode informar a quantidade de milhas que usará do seu saldo, e o valor resultante deverá ser pago em dinheiro;
    - Assim que o cliente confirma o pagamento, seu saldo é atualizado e a compra é efetivada (assumimos que o cliente já fez o pagamento em dinheiro necessário);
    - Como passo final é gerado um código de reserva único, formado por 3 letras maiúsculas e 3 números. Este código será usado para localizar a reserva e confirmar o embarque.
- **R08: Cancelar Reserva** - Este requisito vem do R3, portanto são mostradas todas as informações do voo, valores gastos e milhas. Quando o cliente cancela a reserva, as milhas retornam para seu saldo e um registro indicando que as milhas vieram de cancelamento deve ser registrado. Também deve ser registrada a data/hora de cancelamento e esse voo deve aparecer como CANCELADO na lista;
- **R09: Consultar Reserva** - O sistema deve mostrar uma tela em branco com um campo de entrada de texto, onde o cliente digita o código de reserva. Então, nesta mesma tela devem ser mostrados todos os dados da reserva: data/hora, código da reserva, origem, destino, valor gasto, milhas gastas, e o estado do voo (ex, se já ocorreu, está confirmado, feito check-in, cancelado, etc). Se o voo está para acontecer nas próximas 48h, deve-se apresentar a opção de fazer check-in. Também deve apresentar a opção de cancelar a reserva;
- **R10: Fazer Check-in** - O sistema deve mostrar uma tela com os voos que vão acontecer nas próximas 48h para que o cliente possa fazer o check-in. O Check-in não é uma confirmação de embarque do cliente, mas de que ele tem ciência da data/hora e que tem a intenção de voar;

PERFIL FUNCIONÁRIO

- **R11: Tela Inicial do Funcionário** - Deve apresentar, em formato de tabela, todos os voos que estão para acontecer nas próximas 48h. Deve mostrar data/hora, aeroporto origem e aeroporto destino, ordenados de forma crescente por data/hora. Deve-se ter um botão, em cada voo, que permite a confirmação de embarque do cliente (R12), um que permite o cancelamento do voo (R13), um que permite a realização do voo (R14);
- **R12: Confirmação de Embarque** - A partir do R11, abre-se um tela onde o funcionário digita o código de reserva para confirmar o embarque do cliente. Se esta reserva não for deste voo, deve mostrar uma mensagem de erro. Neste ponto a reserva do usuário passa para o estado EMBARCADO;
- **R13: Cancelamento do Voo** - A partir do R11 o funcionário pode cancelar o voo, desde que ele esteja no estado CONFIRMADO. Assim, haverá o cancelamento de todas as reservas de todos que compraram passagens. O voo passa para o estado CANCELADO e todas as reservas passam para o estado CANCELADO VOO.
- **R14: Realização do Voo** - Este requisito vem do R11, e o funcionário registra que um voo no estado CONFIRMADO realmente ocorreu. O voo passa para o estado REALIZADO e todas as reservas que estão no estado EMBARCADO passam para REALIZADO. As reservas cujo cliente não embarcou passam para o estado NÃO REALIZADO;
- **R15: Cadastro de Voo** - Um funcionário pode cadastrar um voo, que vai conter os seguintes dados: Código do Voo (TADS0000), Data/hora, aeroporto origem, aeroporto destino, valor da passagem em reais (mostrar seu equivalente em milhas) e quantidade de poltronas. Neste ponto o Voo está no estado CONFIRMADO;
- **R16: (CRUD de Funcionário) Listagem de Funcionários** - Apresenta a lista de todos os funcionários ordenados de forma crescente por nome. Deve-se mostrar: Nome, CPF, E-mail e Telefone. O CPF deve ser único e o e-mail será usado como login;
- **R17: (CRUD de Funcionário) Inserção de Funcionário** - Permite a inserção de funcionários. A senha do funcionário deve ser enviada por e-mail e deve ser um número aleatório de 4 dígitos;
- **R18: (CRUD de Funcionário) Alteração de Funcionário** - Deve-se permitir a alteração de dados de funcionário, menos seu CPF.
- **R19: (CRUD de Funcionário) Remoção de Funcionário** - Ao ser removido um funcionário, seus dados não devem ser apagados, somente inativados.

Decomposição Por Subdomínio

Em uma análise preliminar, o sistema foi decomposto em poucos subdomínios e foram determinados os seguintes serviços a serem implementados, contendo seus dados (mínimos, podendo haver mais caso vocês detectem alguma necessidade):

- **Autenticação**: responsável pela autenticação no Sistema.
    - **Tabela para Dados de Usuário**: login, senha, Tipo (cliente/funcionário)
- **Cliente**: responsável pela manutenção de clientes;
    - **Tabela para Dados de Cliente**: CPF, Nome, E-mail, Rua/Número, Complemento, CEP, Cidade, Estado, Milhas
    - **Tabela de Transações de Milhas**: Cliente, data/hora transação, quantidade de milhas, entrada/saída, descrição
- **Voos**: responsável pela manutenção dos voos;
    - **Tabela de Aeroportos**: Código de 3 letras, Nome do aeroporto, Cidade, Estado (deve estar pré-cadastrado)
    - **Tabela de Voos**: Código do voo, Data/hora, aeroporto origem, aeroporto destino, valor da passagem em reais, quantidade de poltronas total, quantidade de poltronas ocupadas
- **Reservas**: responsável pela manutenção das reservas feitas pelos clientes;
    - **Tabela de Reserva**: Código da reserva, código do voo, data/hora da reserva, estado da reserva
    - **Tabela de Histórico de Alteração de Estado de Reserva**: Código da reserva, data/hora da alteração do estado da reserva, estado origem, estado destino
    - **Tabela de Estados de Reserva**: Código do estado, Sigla do estado, Descrição do estado (deve estar pré-cadastrada)
- **Funcionário**: responsável pela manutenção dos dados de funcionários;
    - **Tabela de Funcionário**: Nome, CPF, E-mail e Telefone

Arquitetura

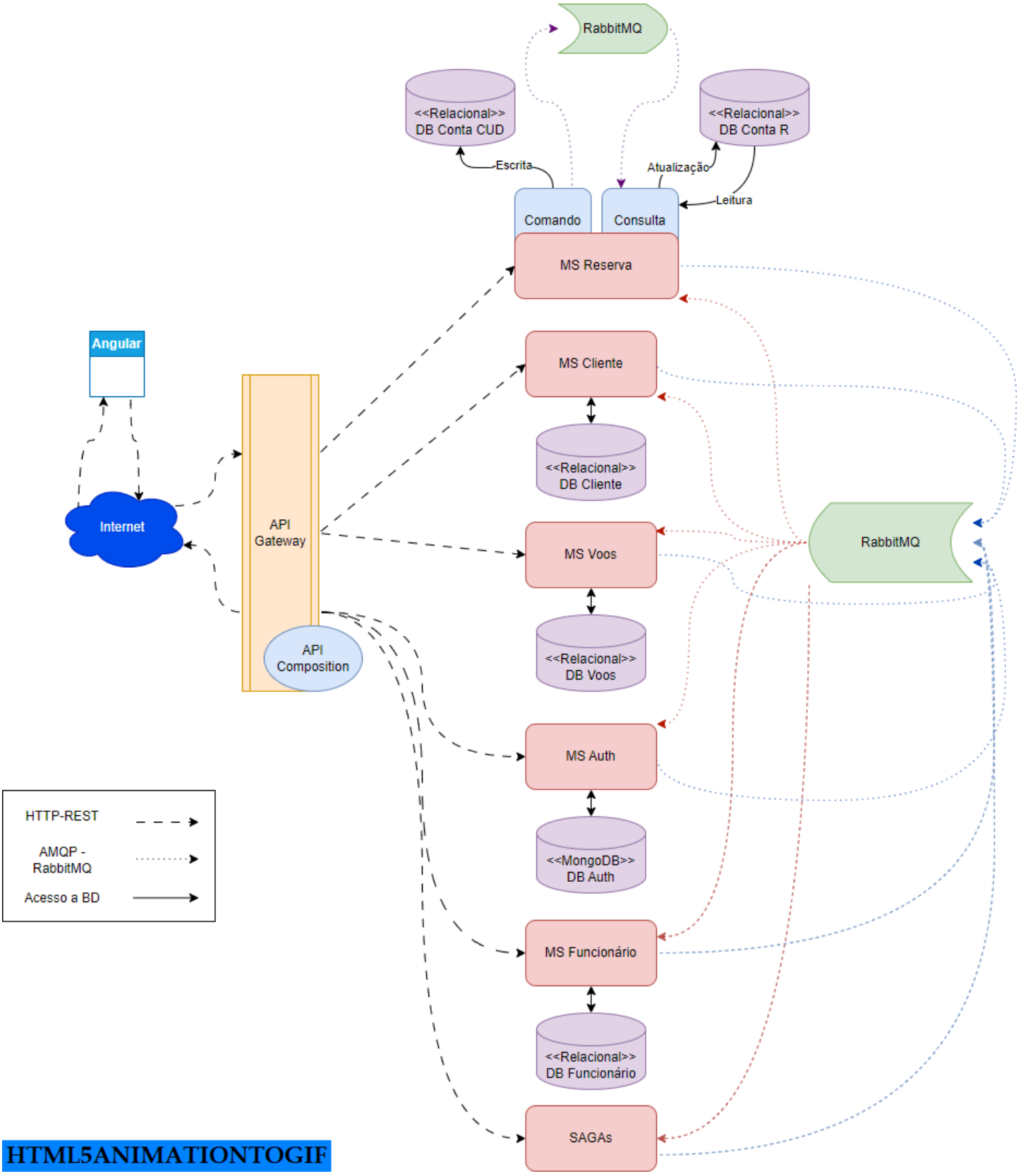
O Sistema deve ser implementado usando-se os seguintes padrões de projeto de microsserviços:

- **Arquitetura de Microsserviços**: para desenvolver o software;
- **Padrão API Gateway**: para expor a API;
- **Padrão Database per Service**: para manter os dados, sendo que cada serviço só tem acesso ao seu SGBD, deve ser implementado o padrão *schema-per-service*;
- **Padrão CQRS**: no microsserviço de Reserva, para todas as consultas. A sincronização dos dois bancos de dados deve ser feita por mensageria;
- **Padrão SAGA Orquestrada**: para transações que abrangem vários serviços;
- **Padrão API Composition**: para agregar resultados de consultas que abrangem vários serviços.

Cada microsserviço, incluindo o API Gateway e cada banco de dados, deve ser executado em uma imagem Docker separada.

A FIGURA 1 ilustra o esboço da arquitetura que deve ser implementada.

FIGURA 1: Arquitetura do Sistema



FONTE: O Autor (2024)

Requisitos Não-Funcionais e Comentários

Toda e qualquer suposição, que não esteja definida aqui e que a equipe faça, deve ser devidamente documentada e entregue em um arquivo **.pdf** que acompanha o trabalho.

- Devem ser usadas as tecnologias vistas na disciplina: Angular 13 (mínimo), Node.js, Spring-boot (Java ou Kotlin), Spring Data JPA, PostgreSQL, Docker e MongoDB, e também as tecnologias apontadas como **Pesquisa**, RabbitMQ para mensageria;
- É extremamente PROIBIDO o uso de geradores de código;
- Os microsserviços são independentes e possuem bancos de dados separados, um microsserviço não pode acessar o BD de outro;
    - Você deve usar *Schema-per-service*, para manter a privacidade dos dados para cada microsserviço;
    - O Banco de dados de autenticação deve ser MongoDB, os demais devem se PostgreSQL;
- Devem ser usados os padrões *Arquitetura de Microsserviços, API Gateway*, *API Composition* (se necessário), *Database Per Service, CQRS (Reserva), SAGA Orquestrada*;
- Cada microsserviço trata de um subdomínio específico. Pode ser criado mais algum microsserviço, desde que validado com o professor;
- Os microsserviços devem estar em conformidade com o **Modelo de Maturidade de Richardson Nível 2**.
- Todos os elementos dos sistemas devem ser conteinerizadas individualmente usando Docker: uma imagem para o API Gateway, uma imagem para cada microsserviço e uma imagem para o banco de dados;
- O Front-end só deve se comunicar com o API Gateway, via API HTTP-REST;
- O API Gateway deve se comunicar com seus microsserviços via API HTTP-REST;
- Os microsserviços, se precisarem, devem se comunicar entre si via mensageria (RabbitMQ);
- Entre o servidor e a aplicação em Angular, somente devem trafegar objetos de classes DTO (nunca objetos persistentes);
- Transações distribuídas devem usar o padrão SAGA
    - A implementação das SAGAs deve ser feita com orquestração usando filas assíncrona com RabbitMQ. **Esse conteúdo não será passado em sala e faz parte do conteúdo de PESQUISA que vocês devem aprender;**
- Vocês deverão identificar e implementar todas as Sagas Orquestradas, aqui algumas de exemplo:
    - Autocadastro:
        - MS Cliente
        - MS Autenticação
    - Inserção/Remoção Funcionário:
        - MS Funcionário
        - MS Autenticação
    - Efetuar Reserva:
        - MS Reserva

        - MS Voo
        - MS Cliente
    - Confirmação de Embarque:
        - MS Reserva
        - MS Voo
    - Etc

- O microsserviço de Reserva deve ser implementado com o padrão CQRS, usando fila assíncrona com RabbitMQ para atualização do banco de dados de consulta. A base de comando deve estar normalizada e a de consulta de estar DESNORMALIZADA;
- O *build*, geração das imagens e execução deve ser feita a partir de um *shell script* automatizado;
- Senhas devem ser criptografadas (SHA256+SALT);
- O leiaute deve ser agradável, usando Bootstrap, Tailwind ou Material no Angular;
- Todos os campos que precisarem devem ter validação;
- Todas as datas e valores monetários devem ser entrados e mostrados no formato brasileiro;
- Todos os campos que tiverem formatação devem possuir máscara;
- Os bancos de dados devem estar normalizados apropriadamente, exceto o banco de leitura do microsserviço Reserva (CQRS) que pode estar desnormalizado;
- O sistema será testado usando o navegador FIREFOX, versão mais recente.