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, CEP. O sistema preenche automaticamente Rua/Número, Complemento, Cidade e Estado usando a API do Via CEP. 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: data da transação, código da reserva (ou vazio se não houver), valor em reais, quantidade de milhas, descrição e tipo (SAÍDA/ENTRADA). Se for uma compra de milhas deve-se ter a descrição "COMPRA DE MILHAS". Se for uma reserva de voo, a descrição deve ser da seguinte forma "CWB->GRU" (Código do aeroporto origem e código do aeroporto destino) ;
- **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, cem 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;
    - Neste ponto a reserva está no estado CRIADA.
- **R08: Cancelar Reserva** - Este requisito vem do R3, portanto são mostradas todas as informações do voo, valores gastos e milhas. Uma reserva pode ser cancelada se ela está CRIADA ou em CHECK-IN. 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 essa reserva deve aparecer como CANCELADA 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; Similar ao R04, mas como botões de ação;
- **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; Neste ponto a reserva estará no estado CHECK-IN.

PERFIL FUNCIONÁRIO

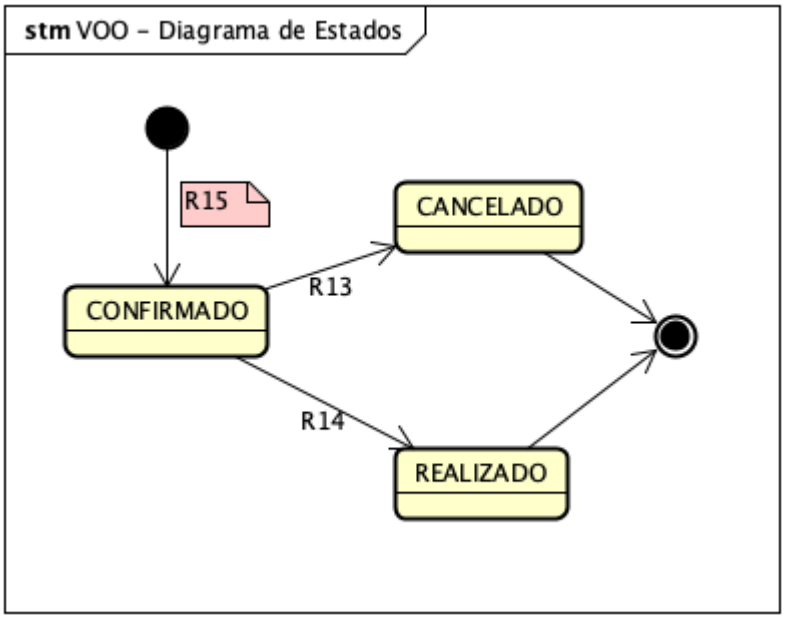
- **R11: Tela Inicial do Funcionário** - Deve apresentar, em formato de tabela, todos os voos que estão para acontecer nas próximas 48h. Deve mostrar data/hora, aeroporto origem e aeroporto destino, ordenados de forma crescente por data/hora. Deve-se ter um botão, em cada voo, que permite a confirmação 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. A reserva deve estar no estado CHECK-IN. 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 EMBARCADA passam para REALIZADA. As reservas cujo cliente não embarcou passam para o estado NÃO REALIZADA;
- **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.

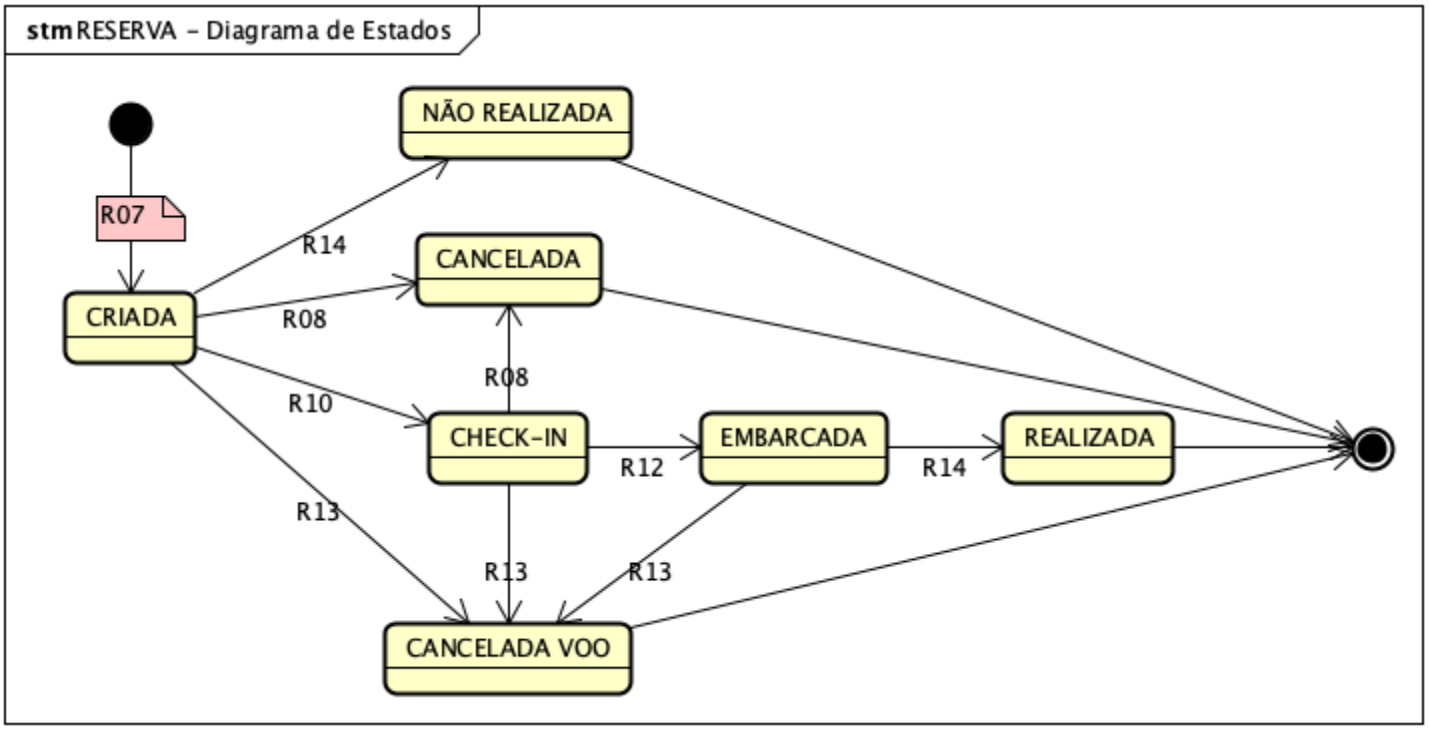
Decomposição Por Subdomínio

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, UF, 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, UF (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, Estado do Voo
    - **Tabela de Estado de Voo**: Código do estado, Sigla do estado, Descrição do estado (deve estar pré-cadastrada: CONFIRMADO, CANCELADO, REALIZADO)
- **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 Estado de Reserva**: Código do estado, Sigla do estado, Descrição do estado (deve estar pré-cadastrada: CRIADA, CHECK-IN, CANCELADA, CANCELADA VOO, EMBARCADA, REALIZADA, NÃO REALIZADA)
- **Funcionário**: responsável pela manutenção dos dados de funcionários;
    - **Tabela de Funcionário**: Nome, CPF, E-mail e Telefone

Diagramas de Estados





Arquitetura

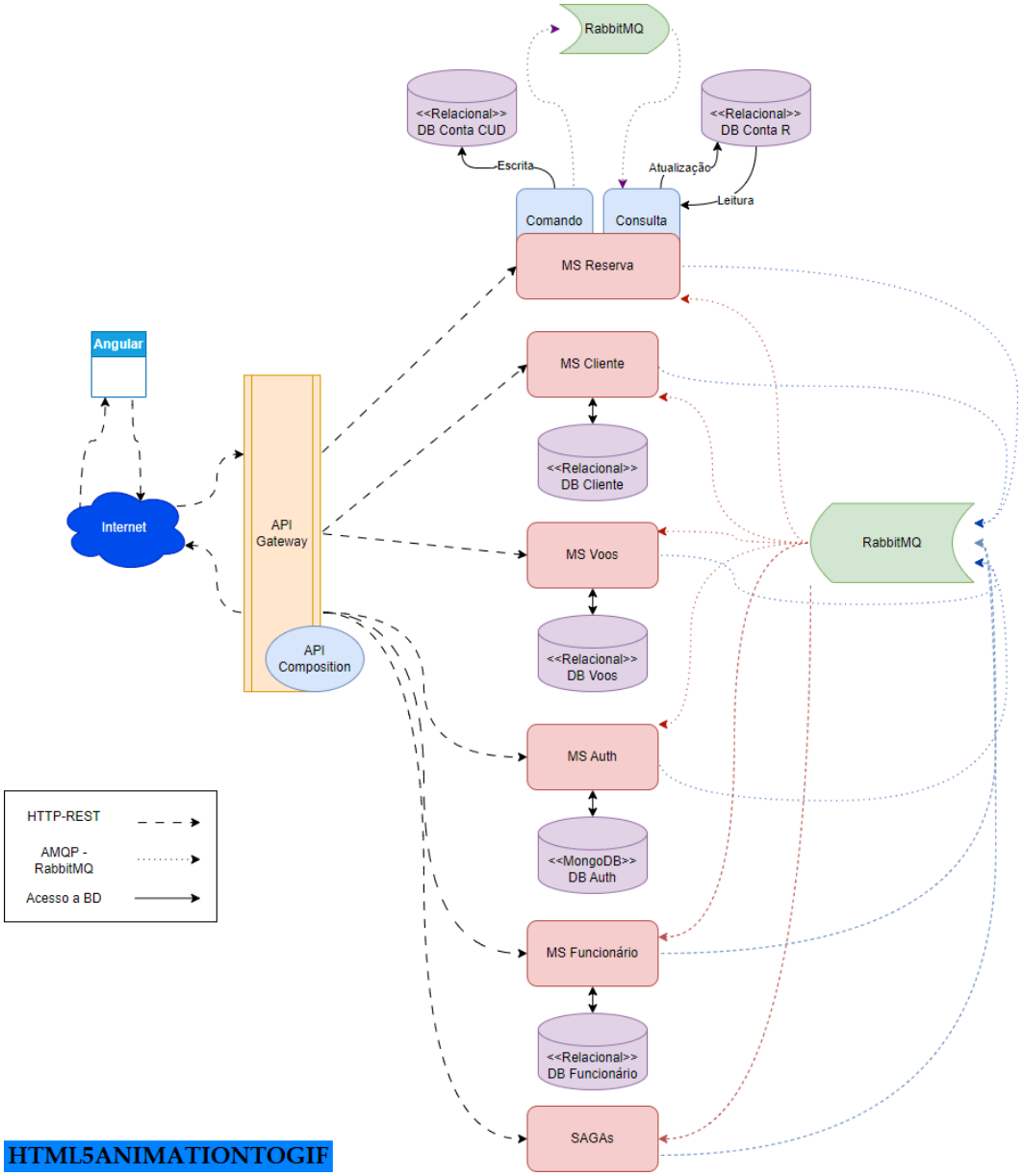
O Sistema deve ser implementado usando-se os seguintes padrões de projeto de microsserviços:

- **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, o API Gateway, cada banco de dados (um para o PostgresSQL e outro para o MongoDB) e o Message Broker (RabbitMQ), 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)

Dados Pré-Cadastrados

Os seguintes dados devem estar pré-cadastrados na sua base de dados, com estes dados em específico (o nome das colunas e localização pode variar conforme sua modelagem de dados):

- **Funcionários**:

cpf	**email**	**senha**
90769281001	func_pre@gmail.com	TADS

- **Aeroportos**:

codigo	**nome**	**cidade**	**uf**
GRU	Aeroporto Internacional de São Paulo/Guarulhos	Guarulhos	SP
GIG	Aeroporto Internacional do Rio de Janeiro/Galeão	Rio de Janeiro	RJ
CWB	Aeroporto Internacional de Curitiba	Curitiba	PR
POA	Aeroporto Internacional Salgado Filho	Porto Alegre	RS

- **Voos**:

data	**codigo_aeroporto_origem**	**codigo_aeroporto_destino**
2025-08-10T10:30Z-03:00	POA	CWB
2025-09-11T09:30Z-03:00	CWB	GIG
2025-10-12T08:30Z-03:00	CWB	POA

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:
  - Front-end: Angular 13 (mínimo) ou React ou Vue.js
  - API Gateway: Node.js
  - Back-end: Spring-boot (Java ou Kotlin)
  - Acesso às bases de dados: Spring Data JPA
  - Banco de dados: PostgreSQL e MongoDB (para usuários)
  - Conteinerização: Docker
  - Mensageria: RabbitMQ.
- É extremamente PROIBIDO o uso de geradores de código;
- Será usada esta aplicação para testar seus end-points:
  https://github.com/razeranthom/test_dac
- *End-points* que precisam de autenticação devem fazê-lo por meio de token JWT:
  - O *end-point* de **login** deve retornar: *token*, tipo do *token*, tipo do usuário e o usuário

```
{
   "access_token": "XXXXX",
   "token_type": "bearer",
   "tipo": "CLIENTE",
   "usuario": {
        ….
   }
}
```

  - Os *end-points* que precisam de um usuário logado devem fazer a verificação;
  - As chamadas aos *end-points* que precisam de um usuário logado devem enviar juntamente no *Header*

```
"Authorization" : "Bearer XXXXX"
```

- 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;
- 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.

NORMAS PARA DEFESA

Diretrizes:

- A defesa deve demorar uns 20 min por equipe.
- Vocês devem trazer suas máquinas para rodar a aplicação.
- No momento da defesa **tudo deve estar no ar:** Front, Back, BD, Containers, etc.
- No momento da defesa o **projeto de teste** deve estar instalado e será executado na hora
- Não serão aceitos projetos sem integração Front x Back, ou rodando com LocalStorage/json-server.
- Não serão aceitos projetos "rodando" somente no Postman.
- Não serão aceitos projetos sem a implementação dos microsserviços solicitados.
- Não serão aceitos projetos sem mensageria (RabbitMQ) implementada.

As tecnologias permitidas são:

- Front-end: Angular, React, Vue.js
- Back-end: API Gateway em Node.js, Microsserviços com Spring Boot em Java ou Kotlin
- Bancos de dados: PostgreSQL e MongoDB para MS de Auth

1) REQUISITOS PARA ENTREGA/DEFESA (Sem isso não há defesa)

- Aplicação de Teste instalada e executando. Ela será executada na hora da defesa;
- Front-end implementado em Angular/React/Vue+Typescript e back-end em Spring Boot (Java ou Kotlin);
- Sistemas usando arquitetura de microsserviços;
- Front-end acessando somente o API Gateway via HTTP-REST;
- Não usar Local Storage nem json-server para armazenar as informações do sistema;
- Usar banco de dados distintos por microsserviço (ou *schema-per-service*);
- Os seguintes requisitos implementados corretamente e de forma completa (arquitetura de MS solicitada, mensageria, SAGA, etc):
    - R01 - Autocadastro
    - R02 - Efetuar Login/Logout
    - R03 - Tela Inicial de Cliente
    - R04 - Ver Reserva
    - R07 - Efetuar Reserva
    - R11 - Tela Inicial de Funcionário
    - R15 - Cadastro de Voo
    - R16 - Listagem de Funcionários
    - R17 - Inserção de Funcionário
    - R18 - Alteração de Funcionário
    - R19 - Remoção de Funcionário

- Uma SAGA completamente implementada;
- Uso de mensageria (RabbitMQ);
- API Gateway básico implementado;
- Sistemas devem possuir interface muito bem elaborada. (Não será permitida a entrega de sistemas em HTML puro ou com interface ruim).

2) O QUE DEVE SER ENTREGUE

Deve ser entregue em arquivo ZIP:

- Todos os fontes do projeto;
- *Scripts* de inicialização do banco de dados (criação e inserções);
- *Scripts* para construção das imagens e execução do projeto;
- Link para UM vídeo no Youtube (ou Google Drive/One Drive se o Youtube não permitir) onde são mostrados os requisitos funcionais.

!!!!! Cuidado para remover arquivos inúteis (executáveis, bibliotecas, diretório node_modules) antes da compactação <= VOU DESCONTAR DE QUEM ENTREGAR NODE_MODULES e ARQUIVOS COMPILADOS)

3) SOBRE O VÍDEO COM OS REQUISITOS FUNCIONAIS

Link para um Vídeo contendo a apresentação de todos os requisitos funcionais implementados.
No vídeo deve aparecer - de forma clara - a identificação do requisito (Número e nome, conforme a especificação do trabalho) que está sendo testado e o teste efetivo de todos os aspectos do requisito.
O vídeo deve ter, no máximo, 20 minutos de duração.
Não há necessidade de todos os integrantes da equipe participarem do vídeo.
Só devem ser mostrados os requisitos funcionais que estão implementados integralmente (front-end e back-end).
Mantenha o banco de dados aberto para mostrar que o requisito funcionou, como uma evidência do teste.

4) SOBRE A DEFESA DOS NÃO-FUNCIONAIS

Defesa dos requisitos não-funcionais. O sistema deve estar funcionando, todos os contêineres carregados.
Todos os fontes devem estar disponibilizados, bem como banco de dados e *scripts*.
O projeto de Testes deve estar instalado e funcionando. Ele será executado na hora da defesa.
A nota será individual, por aluno, que deverá responder aos questionamentos do professor, bem como demonstrar fluência no código para explicá-lo, alterá-lo ou criar funcionalidades novas, no momento da defesa.