Technical Report - **Product specification**

FlyQuest

Unidade Curricular:	IES - Introdução à Engenharia de Software
Data:	Aveiro, 18/12/2024
Students:	103070: Eduardo Lopes 112665: Tomás Brás 113402: Hugo Ribeiro 113626: Rodrigo Abreu

Resumo: FlyQuest é um software desenvolvido para uma companhias aérea e permite gerenciar os aspetos operacionais relacionados aos voos. Esta app destina-se a administradores e equipas de bordo, facilitando a organização, agendamento e comunicação.

Tabela de conteúdos:

1 Introdução

Este projeto foi desenvolvido no âmbito da Unidade Curricular “Introdução à Engenharia de Software”. Decidimos desenvolver um software para uma companhia aérea para que esta possa acompanhar e gerir os seus voos, aviões, pilotos, etc...

O objetivo da FlyQuest é simplificar e tornar mais eficiente a gestão dos voos, proporcionando um serviço de qualidade aos seus utilizadores.

Este documento serve para descrever os requisitos e a estrutura do projeto, dentro dos objetivos da UC.

2 Conceito de Produto

Declaração de Visão

FlyQuest foi desenvolvido com a intenção de melhorar a gestão interna dos voos de uma companhia aérea.

O sistema foi desenvolvido para ser administrado pela companhia aérea (administrador) e utilizado pelos pilotos e assistentes de bordo (utilizadores regulares).

A administração da companhia terá acesso a todas as informações relacionadas aos voos, podendo verificar lugares livres, ocupados e a lotação total do avião. Além disso, será possível gerenciar aspectos operacionais de cada voo, como a escolha de pilotos e assistentes de bordo.

O software também facilitará a resolução de problemas em relação a condições meteorológicas, permitindo o cancelamento e o reagendamento de voos de maneira fácil e eficiente, em caso de mau tempo. Também é possível substituir membros da equipa de voo em casos de ocorrência de imprevistos.

Com isto, a companhia consegue resolver problemas de uma forma mais rápida, uma vez que consegue comunicar essas alterações diretamente aos seus funcionários.

Personas

1. Rogério Marques

O Rogério tem 55 anos e é responsável pela gestão de uma companhia aérea muito concorrida, devido à prática de preços bastantes baixos.

Como Administrador, desta empresa de sucesso, o seu trabalho é garantir que todos os voos consigam ser efetuados sem atrasos e assegurar que todos os passageiros tenham uma experiência agradável e sem problemas.

Motivação:

Rogério é motivado pela busca de excelência operacional e pela satisfação do cliente. Ele dedica-se a implementar soluções que aumentem a eficiência da empresa, melhorando constantemente os processos e assegurando que a companhia se destaque no mercado.

2. Inês Simões

A Inês tem 29 anos e é assistente de bordo experiente, que trabalha para garantir a segurança e o conforto dos passageiros durante os voos. A Inês gosta de planear bem as suas viagens, porque faz muitos voos por semana. O seu serviço passa por organizar e orientar os passageiros durante o voo.

Presta assistência e resolve qualquer problema que possa surgir e caso seja necessário, comunica o problema ao piloto.

Motivação

Inês é motivada pela paixão por viajar e pela oportunidade de proporcionar uma experiência positiva aos passageiros. Ela esforça-se para fazer com que cada cliente se sinta bem cuidado, contribuindo para uma atmosfera acolhedora a bordo.

3. **Miguel Cunha**

O Miguel tem 35 anos, é um piloto de aviões habilidoso e responsável. Ele tem uma vasta formação na área. Possui um curso de aviação civil e uma licença de piloto comercial com experiência em voos nacionais e internacionais. Conhecido por sua capacidade de manter a calma em situações desafiadoras, ele demonstra um excelente controlo do avião em todas as circunstâncias.

Motivação

Miguel é motivado pelo amor à aviação e pela responsabilidade de transportar pessoas com segurança. Ele tem o compromisso de manter os padrões de segurança e eficiência altos, procurando sempre aperfeiçoar as suas habilidades e conhecimentos técnicos através de cursos de atualização e treinamentos regulares.

Cenários Principais

Rogério cancela um voo. Rogério é avisado pela torre de controlo que o mau tempo não vai passar, dificultando a descolagem do avião. Assim, o Rogério abre a app da *FlyQuest* no seu computador e cancela o voo *F123456*, o que consequentemente alertará os pilotos e os assistentes de bordo que iriam realizar esse voo.

Inês verifica os seus voos. Todos os domingos à noite, Inês abre a app da *FlyQuest* e vê os voos que lhe foram atribuídas esta semana, de modo a conseguir organizar melhor a sua vida atarefada de hospedeira de bordo, como por exemplo: preparar a mala para os dias que vai passar fora de casa.

Miguel é notificado pela app, relativamente ao seu copiloto do voo *F654321*. O copiloto do Miguel no voo *F654321* encontra-se indisposto para exercer a sua função. Assim, este notifica a companhia aérea, que atribui um novo copiloto ao voo. O Miguel recebe uma notificação através da app, informando quem será o seu novo copiloto.

Requisitos do Produto (User stories)

Epic 1 - Gestão e Atualização dos Voos

- Como administrador, quero gerir e ver todos os horários de voos agendados, para que possa garantir uma gestão adequada e minimizar atrasos.
- Como administrador, quero atualizar a informação e o estado dos voos, para que possa garantir que as equipas de cada voo são notificadas das alterações.
- Como administrador, quero ter acesso à informação sobre o número de lugares ocupados no avião para cada voo, para que possa acompanhar a ocupação dos voos.
- Como administrador, quero adicionar/cancelar voos da lista, para que possa garantir que as lista de voos estejam sempre atualizadas e precisas.
- Como administrador, quero conseguir filtrar a lista de voos, para tornar a procura dos voos rápida e eficiente.
- Como administrador, quero estar informado sobre as condições climáticas e receber alertas, para que eu possa tomar decisões sobre cancelamentos ou ajustes de voos.
- Como administrador, quero adicionar um avião à minha frota, para que assim não exista falta de aviões.
- Como administrador, quero ter a informação sobre a localização dos aviões da companhia aérea, para saber os aviões disponíveis no aeroporto de uma certa cidade para poder realizar um certo voo.

Epic 2 - Gestão e Atribuição da Tripulação

- Como administrador, quero gerir as atribuições da tripulação de voo, para que possa alocar pilotos e assistentes de voo específicos a cada voo.
- Como piloto, quero conseguir ver quem vai ser o meu copiloto e os assistentes de bordo, para que possa saber antecipadamente quem é a equipa que vai trabalhar comigo.
- Como piloto, quero poder aceder facilmente ao meu horário de voos, para que possa preparar cada voo adequadamente.
- Como assistente de voo, quero ter acesso a todos os voos no meu horário, para que esteja sempre atualizado sobre os voos em que irei trabalhar.

Epic 3 - Gestão de Contas de Funcionários

- Como administrador, quero adicionar contas de trabalhadores com as devidas permissões, para que todos os trabalhadores novos tenham acesso aos seus voos
- Como administrador, quero desativar contas de trabalhadores, para que só trabalhadores ativos tenham acesso às informações sobre os voos

3 Caderno de Arquitetura

Principais requisitos e restrições

Por ser implementada como Web App, a FlyQuest tem uma série de requisitos e restrições que tem de ser respeitadas como:

- Os dados mostrados pela WebApp devem estar sempre consistentes e atualizados;
- O sistema consome uma API externa, *tomorrow.io*, que permite ter a informação meteorológica mais atualizada para o destino e origem do voo;
- Alguns dados do utilizador devem estar encriptados, visto que são confidenciais;
- O sistema deve ser rápido a emitir notificações ao administrador sobre condições meteorológicas adversas.
- Todos os utilizadores devem ter as permissões previstas, aquando à sua inserção na empresa, para poderem acessar apenas os dados que lhes convém
- O administrador deve ser capaz de acessar todas as operações CRUD, enquanto os outros utilizadores não devem ser capazes de alterar ou inserir dados, para além das suas informações pessoais.
- Todos os utilizadores devem ser capazes de ver/filtrar dados dentro das suas permissões.
- O sistema deve ser capaz de receber um grande fluxo de dados provenientes da API externa (tomorrow.io).
- A Base de Dados deve permanecer acessível no intuito de receber pedidos de dados específicos a qualquer momento.
- Deverá ser possível, e em tempo real, editar a informação do voo (ex. substituir pilotos ou assistentes de bordo) e processar os dados provenientes da API externa.

Vista da Arquitetura

A implementação do sistema vai ser dividida em 6 componentes:

- **External API:** Com o objetivo de fortalecer a qualidade e autenticidade do nosso sistema e usar dados mais precisos e realistas vamos importar dados meteorológicos através da API do *tomorrow.io*.
- **Gerador de Dados:** Para simular os clientes da companhia aérea, implementaremos um script em Python para gerar a compra de bilhetes para os diferentes voos disponíveis de uma forma aleatória. Estes dados serão enviados para o message queue.
- **Message Queue:** O message queue irá receber os dados provenientes do Data Generator e da API. Estas informações serão posteriormente consumidas pelo backend. Para o nosso sistema, escolhemos utilizar o kafka.
- **Backend:** No backend, iremos implementar uma API desenvolvida em Spring Boot, capaz de fornecer dados aos clientes de forma rápida e eficiente. Este módulo tem a responsabilidade de processar os dados armazenados na message queue, de forma a que estes sejam guardados na respetiva base de dados. Esta arquitetura garante a alta coesão e o baixo acoplamento entre os diferentes serviços.
- **Bases de Dados:** Quatro bases de dados MySQL separadas para armazenar dados específicos de diferentes funcionalidades do sistema. Cada uma será dedicada a um conjunto distinto de informações: dados de login e autenticação, todas as informações relacionadas aos voos, detalhes dos aviões, e as condições meteorológicas necessárias para permitir ou cancelar um voo.
- **Serviços:** Quatro micro-serviços independentes, onde cada um acessa exclusivamente sua própria base de dados. Estes serviços comunicam entre si de forma assíncrona, trocando informações necessárias para realizar as operações de maneira colaborativa e distribuída. Esta abordagem garante maior escalabilidade, isolamento de falhas e facilidade de manutenção, já que cada serviço é responsável apenas por um conjunto específico de dados dentro do sistema.
- **Frontend:** Uma Web App desenvolvida em React destinada aos empregados da companhia que permite a visualização/notificação da informação relacionada aos seus voos. O objetivo é proporcionar uma interface amigável e interativa para a apresentação e manipulação de dados.

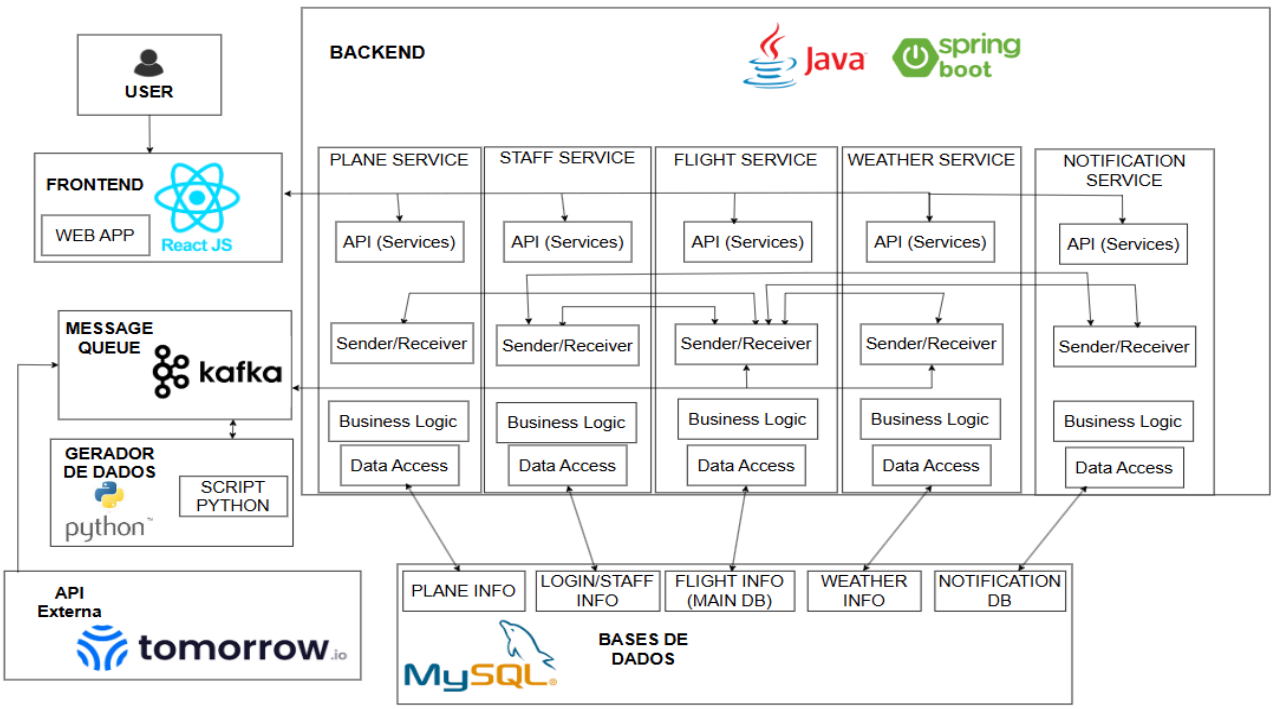

Figura 1 - Diagrama Vista Geral Arquitetura

Interações dos módulos

Ao fazer login, a web app interage com o módulo de Data Access, que por sua vez interage com a Base de Dados com o objetivo de validar os dados de login.

Os dados gerados pelos scripts em python tal como os provenientes da API externa são enviados para o módulo de Data Access, sendo posteriormente armazenados na Base de Dados. Ao mesmo tempo, estes dados também são enviados para a API (Spring Boot), onde são processados. De seguida, são devolvidos à web app de forma a que o utilizador tenha acesso aos mesmos.

Mas o envio de informação não é só feito num sentido, ou seja, quando o administrador cria os dados de login para um novo funcionário, a web app envia estes dados para a API para serem processados. Por fim, estes são enviados para o módulo Data Access com a intenção de serem guardados na Base de Dados.

1. Aceder aos dados guardados na Base de Dados:

Depois dos dados de login serem verificados, o utilizador consegue aceder aos dados guardados na Base de Dados, através de um pedido à API que é reenviado para o módulo de Data Access, que por sua vez irá procurar os dados na BD.

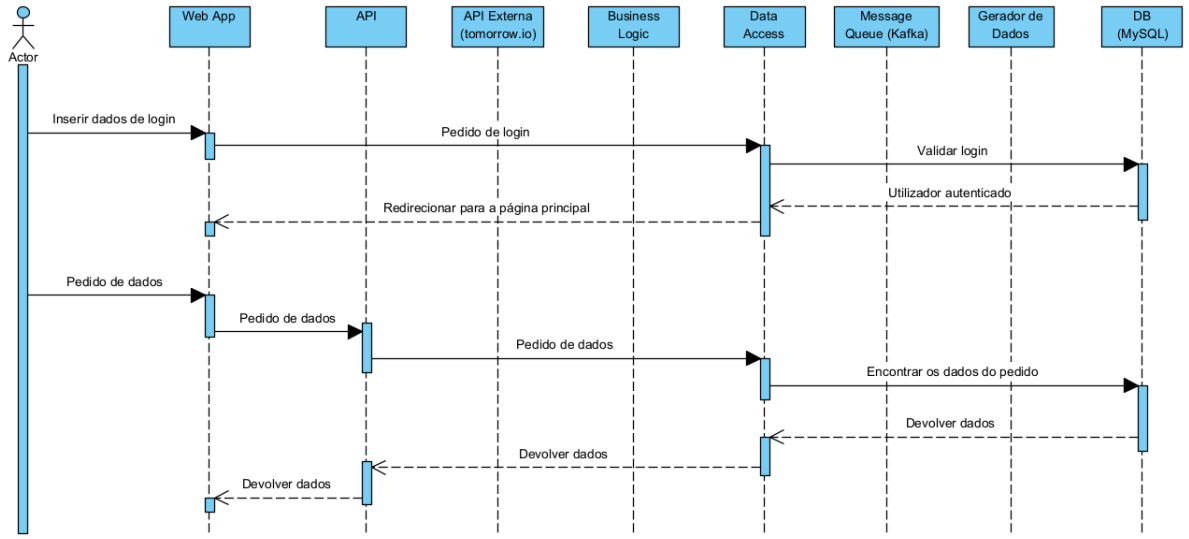

Figura 2 - Diagrama de Sequência 1

2. Aceder aos dados produzidos em tempo real (scripts):

Neste caso, o módulo de Data Access recebe a informação do módulo de geração de dados (scripts), e de seguida envia esses dados para a DB. Ao mesmo tempo, os dados também são enviados para a API para serem processados e por fim, retornados à web app.

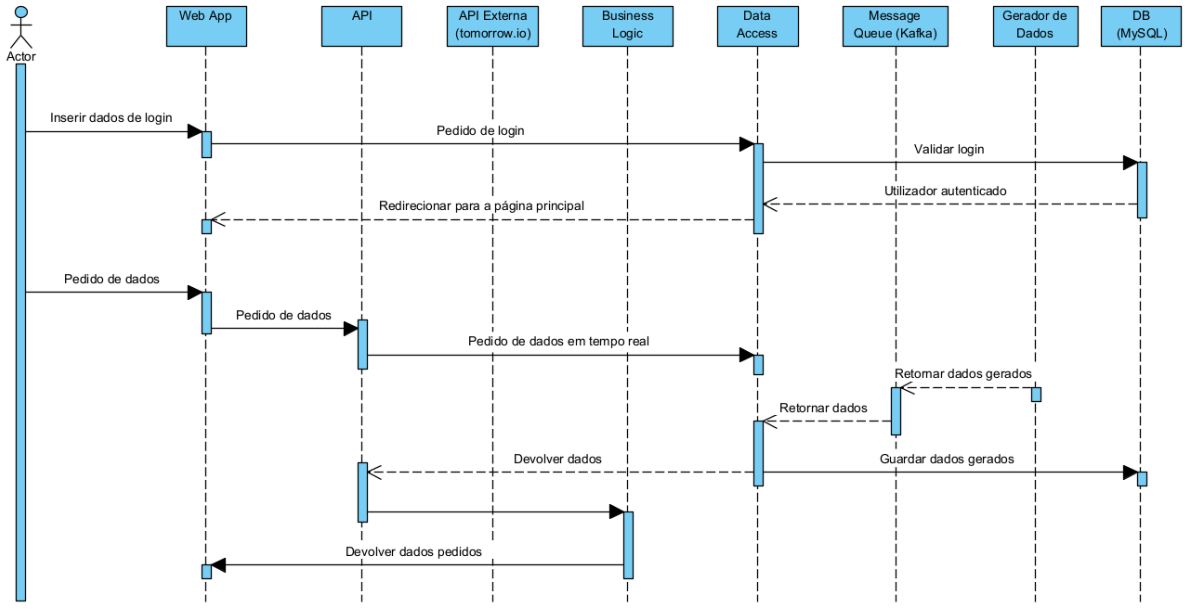

Figura 3 - Diagrama de Sequência 2

3. Acesso aos dados pedidos à API externa:

Ao contrário da situação anterior, a web app vai fazer um pedido de dados à API externa, que envia os dados pedidos para o módulo de Data Access. Os dados vão ser guardados na BD e ao mesmo tempo, vão ser enviados para a API para serem processados. Por fim, estes são devolvidos à web app.

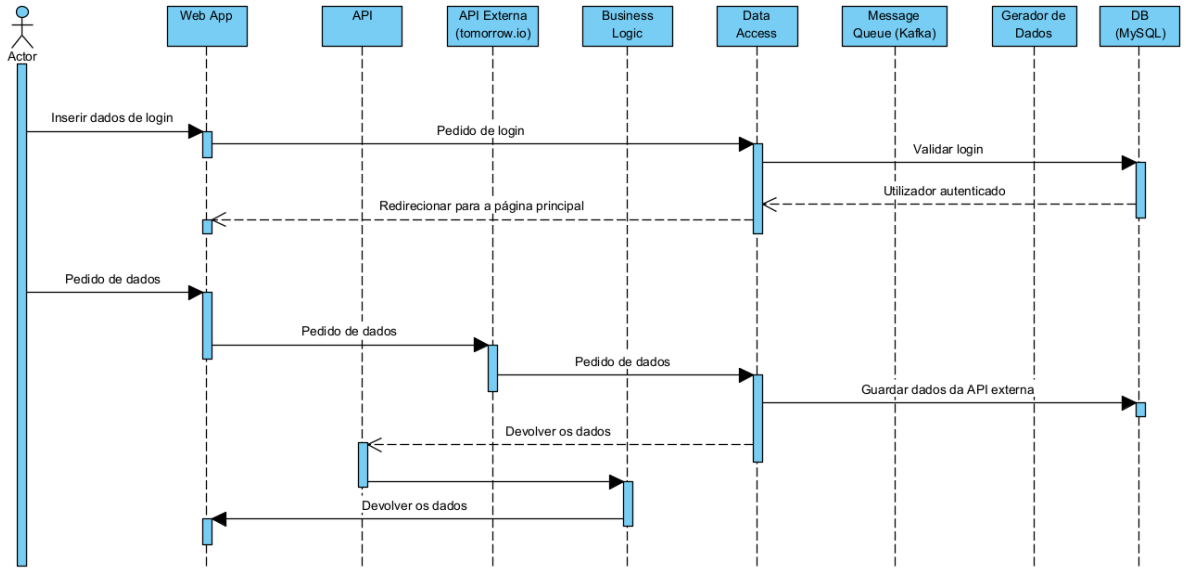

Figura 4 -Diagrama de Sequência 3

4 Perspetiva de Informação

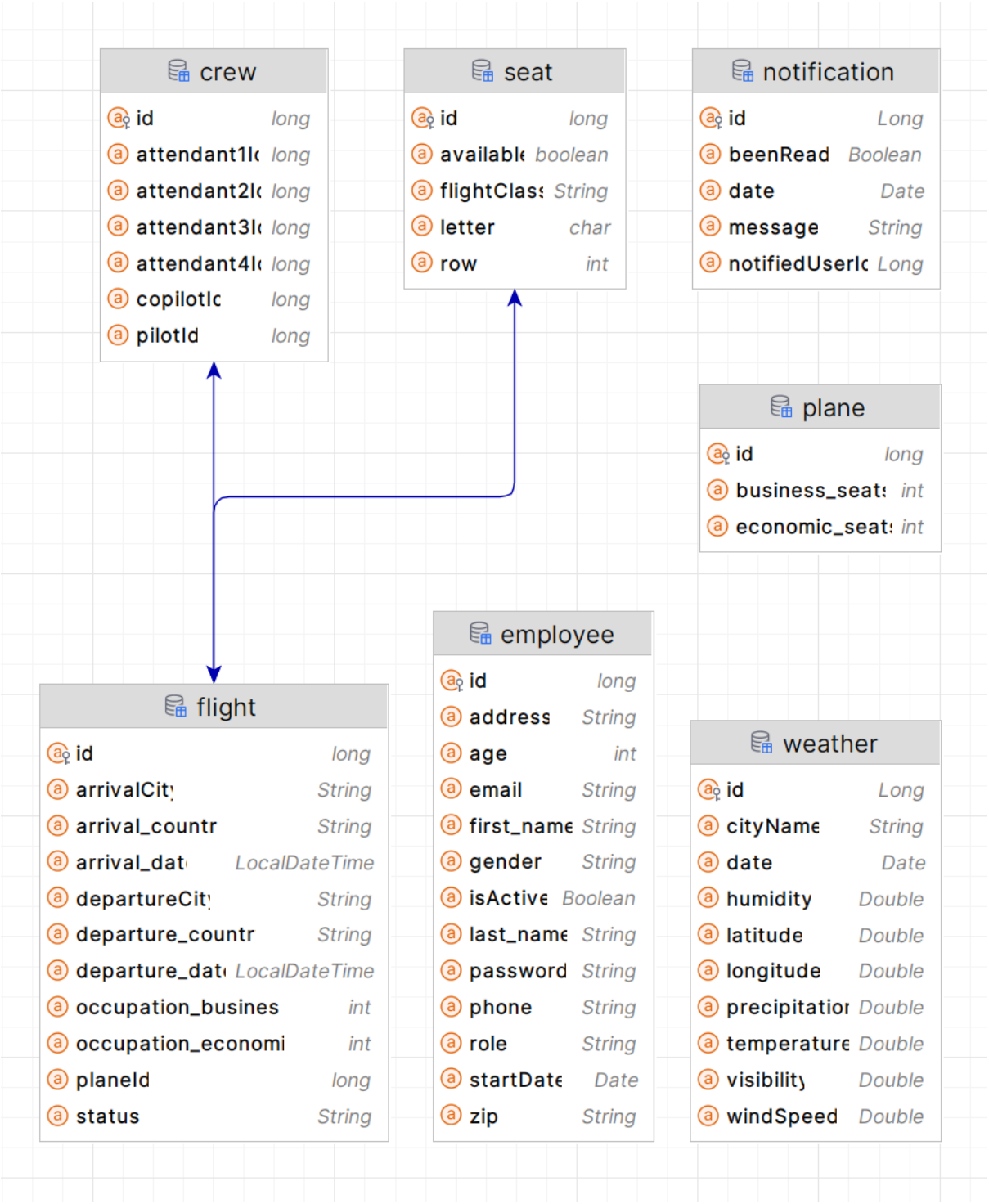

Figura 5 -Diagrama de Entidades

5 Documentação da API

Authorization Controller -> /api/v1/auth

POST	/login	Realizar a autenticação do user com base no email e na password
POST	/createAccount	Criar uma conta

CityWeather Controller -> /api/v1/weather

POST	/add	Adicionar uma cidade(coordenadas)
PUT	/{cityname}	Atualizar informações de cidade
GET	/{cityname}	Obter condições meteorológicas pelo nome da cidade.
GET	/cities	Obter todas as cidades.

Flight Controller -> /api/v1/flight

POST	/add	Adicionar um voo
GET	/all	Obter a lista de voos
GET	/{id}	Obter um voo pelo id.
PUT	/{id}	Atualizar voo pelo id.
PUT	/{id}/cancel	Cancelar um voo pelo id.
GET	/{id}/pilots	Obter pilotos de um voo pelo id do voo.
GET	/{id}/attendants	Obter os hospedeiros de bordo de um voo pelo id do voo.
GET	/{id}/plane	Obter avião de um voo pelo id.
GET	/{id}/departCity	Obter id da cidade de partida do voo.
GET	/{id}/arrivalCity	Obter id da cidade destino do voo.
PUT	{id}/crew/{employee_id}	Trocar um empregado da crew pelo id do

		voo e pelo id do empregado a trocar.
GET	{id}/seats	Obter todos os lugares do avião de um voo pelo id.
GET	/{id}/seats?fligthClass=<flight_class>	Obter todos os lugares de uma classe
GET	/{id}/seats/available	Obter todos os lugares livres do voo
PUT	/{id}/seat	Atualizar o estado de um lugar num voo pelo id do voo para reservado.
GET	/{id}/seats/{flightClass}/occupied	Obter os lugares ocupados de um voo pela classe e pelo id do voo.

Plane Controller -> api/v1/plane

POST	/add	Adicionar um avião.
GET	/{id}/flights	Obter os voos associados ao avião pelo id.
GET	/{id}	Obter um avião pelo id
GET	/all	Obter todos os aviões.

Staff Controller -> api/v1/staff

GET	/all	Obter todos os empregados.
GET	/all?role=<role_name>	Obter todos os empregados de uma certa função.
PUT	/{id}	Atualizar informações de um empregado pelo id.
GET	/{id}/flights	Obter voos de um empregado pelo id.

Notification Controller -> api/v1/notification

PUT	/{notification_id}	Atualizar uma notificação como lida através do id
GET	/all/{user_id}	Obter todas as notificações de um utilizador
DELETE	/{notification_id}	Remover uma notificação pelo id.

6 Conclusão

Em conclusão, o desenvolvimento da aplicação FlyQuest foi uma experiência importante para a compreensão de como desenvolver software de uma forma correta e organizada. Aprendemos a importância de estabelecer os requisitos e uma arquitetura bem definida previamente. A documentação da API revelou ser um aspeto fundamental da implementação. Em geral, estamos satisfeitos com o resultado da nossa implementação e acreditamos que o FlyQuest é um software útil para várias companhias aéreas.

7 Referências e recursos

Na Realização deste projeto utilizamos o material disponível nos guiões práticos e os seguintes recurso:

https://legacy.reactjs.org/docs/getting-started.html

https://react-bootstrap.netlify.app/docs/getting-started/introduction

https://docs.docker.com

https://dev.mysql.com/doc/

https://docs.spring.io/spring-boot/index.html

https://start.spring.io/

https://maven.apache.org/guides/index.html

https://www.freecodecamp.org/news/how-to-consume-rest-apis-in-react/