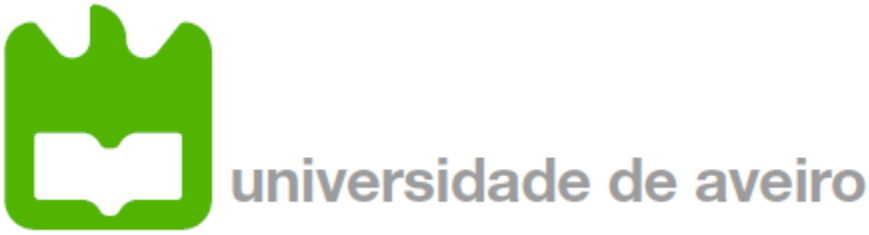



DEPARTAMENTO DE ELECTRÓNICA TELECOMUNICAÇÕES E INFORMÁTICA

MESTRADO EM ENGENHARIA INFORMÁTICA

# Engenharia de Software

## Relatório do Trabalho Prático

André Moreira 62058
João Artur 80390
Diogo Mota 94956
Diogo Alves 96437
Gil Lusquiños 99247

# Índice

# 1. Air Tracking

A plataforma Air Tracking tem como objetivo permitir uma fácil visualização e acesso a várias métricas de voos aéreos.

Com base na identificação única de cada voo pretendemos que o utilizador da plataforma consiga obter informações relevantes ao voo como velocidade, altitude, localização, etc. Também pretendemos que o utilizador consiga localizar o voo na sua posição geográfica para um enquadramento mais eficaz da sua localização territorial.

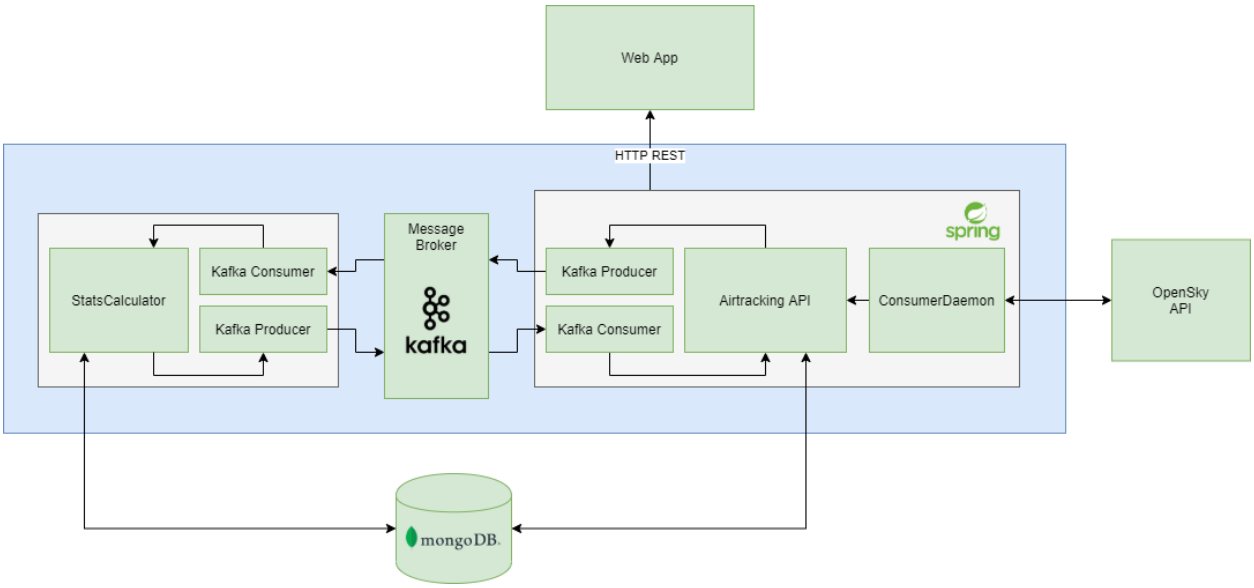
O utilizador da plataforma deve conseguir selecionar um voo que esteja a decorrer neste momento sem que tenha que consultar informações em outros locais. Escolhendo um voo o utilizador deve conseguir aceder às estatísticas do voo e também do avião.

Se um utilizador quiser saber os voos que estão a sobrevoar uma região, o utilizador deve conseguir visualizar num mapa esses voos de forma fácil.

Ao escolher um voo, o utilizador deve conseguir visualizar as informações do respetivo voo.

Ao escolher um avião o utilizador deve conseguir visualizar as informações sobre o avião em questão.

# 2. Arquitetura




A arquitetura do projeto consiste numa rede de microsserviços com diversas funções. A aplicação principal, “Airtracking”, consiste numa API *SpringBoot* que consome dados da OpenSky API referentes aos diferentes voos. Esses dados são colocados numa base de dados *MongoDB*, atualizados de forma a impedir que o repositório fique demasiado populado com informação desatualizada.


Após os dados serem consumidos da API OpenSky, as mensagens relativas aos estados do voo sofrem uma transformação para serem persistidas na nossa base de dados. Determinados campos considerados irrelevantes para o nosso caso de uso são eliminados e a mensagem é reestruturada para o formato chave - valor JSON que posteriormente é inserido na base de dados MongoDB.

À estrutura completa da mensagem, chamamos de **FlightStateMessage** que é o conjunto de todas as informações de todos os voos a acontecer num determinado instante, sendo esse instante (time) a chave primária da sua entrada na BD.

Cada FlightStateMessage tem portanto uma lista de **FlightState**s, que contém as referidas informações de cada voo que, por sua vez, é identificado pelo seu código **icao24**.

Com essa informação, a aplicação “StatsCalculator” é capaz de calcular novos dados, como velocidade média de determinado vôo, informação esta que é passada através de Kafka para a API “Airtracking”, quando requisitada. Este *data flow* segue uma arquitetura de *request/reply*.

Nesta troca de informação dois tópicos de Kafka são responsáveis pelo transporte do pedido e da resposta da informação estatística, o **STATS_REQ** e o **STATS_RESP**.
O primeiro contém as mensagens provenientes do *controller* na API responsável pelo pedido de estatísticas acerca de um certo voo, essas mensagens são apenas o código icao24 do voo que pretendemos obter estatísticas.
O segundo tópico referido, contém as mensagens provenientes do serviço de estatísticas já contendo o corpo JSON da informação requisitada no pedido relativo aquele voo:


```
{
    "icao24": "a09281",
    "max_speed": 60.3,
    "avg_speed": 57.5,
    "avg_vrate": -0.223
}
```


Esta mensagem é depois revolvida como resposta no método /getStats na AirTracking API após ser consumida do broker.
As duas aplicações e a troca de informação entre si, sucintamente apresentadas, constituem o *middleware* da aplicação.

A “Web App” consiste na camada superficial do sistema gerado, contendo o UI a apresentar aos utilizadores. As trocas de informação com o *middleware* ocorrem através de HTTP REST, realizando pedidos às APIs conforme o que for necessário para a página a apresentar.

# 3. Perspetiva dos *Developers*

- Criação da AirTracking API para haver controlo da informação vinda da OpenSky API, ou seja, ter uma API apenas com os dados pretendidos, estes dados são posteriormente guardados na base de dados, MongoDB.

  http://192.168.160.103:9069/flightstates/

- MongoDB - escalabilidade, flexibilidade/modularidade e velocidade de acesso aos dados.

- Kafka para que que não houvesse sobrecarga no envio e receção das estatísticas.

Webapp (http://192.168.160.103:9070/).

Mapa (http://192.168.160.103:9070/map):
A página do mapa faz uma chamada ao endpoint (http://192.168.160.103:9069/flightstates/last, que contém apenas os últimos dados de cada voo) a cada 10 segundos e dá update ao mapa consoante os dados mais recentes do mesmo. Sendo possível ver os aviões a moverem-se.
Ao dar hover no avião é possível obter informação sobre o mesmo também através do mesmo endpoint.
O mapa foi implementado em JavaScript com recurso a D3.js e funciona através de coordenadas reais.

Flight Statistics (http://192.168.160.103:9070/flight):
A página das estatísticas dos voos consegue filtrar a lista dos voos atuais por país, de forma a que seja mais fácil encontrar o voo pretendido.
Após a filtragem e escolha do voo pretendido é feita uma chamada ao endpoint (http://192.168.160.103:9069/getStats/{icao24}, que contém apenas as estatísticas já calculadas do icao24 em questão).
As estatísticas são calculadas através dos 10 últimos flightstates (útlimos 20 minutos real-time).

Plane Data (http://192.168.160.103:9070/plane):
Consegue filtrar da mesma maneira que o Flight Statistics, mas vai buscar os dados do avião ao endpoint (http://192.168.160.103:9069/flightstates/last), podendo mostrar então os dados mais recentes do avião selecionado.

Em implementações futuras, uma feature pertinente para o sistema seria o tracking de um avião específico por parte de um utilizador.

Para tal, seria necessário a implementação de um sistema de sign-in e log-in no qual o utilizador registado poderia escolher um avião no qual quisesse ter informação de tracking em real time ou receber alertas (Email,SMS) que considerasse pertinentes, como por exemplo se

um voo selecionado tivesse levantado voo ou tivesse pousado (campo booleano on_ground), se o avião estivesse em processo a descer de altitude o que indicaria que estivesse em processo de aterragem (valor negativo do vertical_rate e velocidade decrescente), etc.

# 4. Perspetiva das Operações

As operações são geridas através de uma pipeline Jenkins que tem como função fazer build, correr testes, publicar as imagens no repositório e fazer o deploy no servidor de produção.

Estando o projeto separado em três aplicações, para cada uma delas é criada uma Docker Image através da respectiva Dockerfile, sendo posteriormente enviada para o repositório fornecido. O deploy no servidor de produção utiliza estas imagens a partir deste repositório, sendo que em cada deploy apenas são manuseadas estas imagens.

O projeto utiliza o Mongo e Kafka para a transmissão de mensagens e o seu armazenamento, estas dependências correm em Docker Containers mas não são geridas através da pipeline Jenkins, sendo necessário a sua gestão manual por exemplo através de uma ligação remota ao servidor de produção.

A criação de imagens Docker é feita após a compilação de cada uma das aplicações, que é realizada através do plugin Maven do Spring Boot, resultando num ficheiro WAR, este ficheiro é enviado para o repositório Artifactory e é utilizado para criar a imagem Docker.

A pipeline permite escolher qual a operação que queremos realizar, não sendo necessário correr todos os passos, podemos por exemplo executar o deploy, ou fazer build separadamente. O processo de deploy termina e elimina o container e a imagem da aplicação selecionada no parâmetro da pipeline, de seguida uma nova imagem é transferida do repositório Docker e é iniciada, este processo de deploy não verifica se o Kafka ou Mongo estão a executar, tendo este requisito ser assegurado de forma manual, caso seja necessário fazer o deploy numa máquina nova.

# 5. Testes

Os testes de software desenvolvidos pretendem validar os cenários definidos posteriormente ao desenvolvimento do sistema.

Foram então desenvolvidos testes de integração que verificavam que não só as operações CRUD tinham o comportamento esperado, como se algumas outras features funcionavam corretamente, como por exemplo, o cálculo correto de estatísticas de um determinado voo.

Os testes que validam os critérios de aceitação são os seguintes:

*Teste de persistência*

- Dado um FlightStateMessage que ocorra na **timestamp** “123456” com os campos **icao24**, **país de origem**, **posição temporal**, **último contacto**, **longitude**, **latitude**, **velocidade**, **taxa vertical** e o booleano que indica se o avião **está ou não em voo** de respectivos valores: **ab172vs**, **Portugal**, **15273617**, **15273617**, **47.123141**, **-3.127371**, **203.2**, **-0.23** e **False**.
- Sendo o referido FlightStateMessage serializado em JSON e inserido no sistema por um pedido HTTP POST na AirTracking API
- Quando é realizado um pedido HTTP GET especificando o parâmetro temporal “123456”
- Então, é obtido como resposta o FlightStateMessage que foi inserido anteriormente, comprovando assim as operações CRUD da API

*Teste de cálculo correto de estatísticas e message broker*

- Dado 3 FlightStateMessages, todos tendo informações acerca do voo de Icao24 “**123TestMayDay**”, com os respetivos valores de velocidades: **52.7**, **60.2** e **56.9**.
- Sendo as 3 mensagens serializadas em JSON e inseridas no sistema por um pedido HTTP POST à API
- Quando realizado um pedido HTTP GET ao endpoint de estatísticas da AirTracking API com a especificação de identificador de voo “**123TestMayDay**”
- Então, no corpo da resposta relativa às estatísticas desse voo, o valor da velocidade média terá que ser de **56.6**, comprovando assim que o cálculo estatístico da velocidade média de um voo específico está a ser calculado corretamente e as mensagens estão a ser trocadas corretamente entre a API e o serviço de estatísticas por via de Kafka.

Os testes são corridos em embedded runtime, cada vez que o projeto dá build pelo que também podem ser executados manualmente na raíz do projeto através do comando: *./mvnw test.*

# 6. Considerações Finais

Face ao trabalho desenvolvido, é possível verificar que a grande maioria dos objetivos foram atingidos com sucesso, tanto a nível de desenvolvimento como a nível de conhecimento. Foi possível experienciar um ambiente real de trabalho de equipa, e todas as vantagens e adversidades que daí provêm.

As maiores dificuldades foram enfrentadas na implementação do ELK, e faltou automatizar os testes na pipeline. Tudo o resto foi executado de modo a responder aos restantes requisitos propostos para o projeto. Apesar de tudo, existe sempre espaço para melhorar e aprender.

Com este trabalho foi possível desenvolver novas aptidões na área e obter um maior nível de conhecimento nas diferentes tecnologias.