Antenor Moreira de Barros Leal

Aplicativo web de auxílio a navegação aérea

PROJETO FINAL

DEPARTAMENTO DE INFORMÁTICA
Programa de Graduação em Engenharia da Computação

Rio de Janeiro
Abril de 2024

Pontifícia Universidade Católica
do Rio de Janeiro

Antenor Moreira de Barros Leal

Aplicativo web de auxílio a navegação aérea

Relatório de Projeto Final II

Relatório de Projeto Final, apresentado ao Programa de Engenharia da Computação, do Departamento de Informática da PUC-Rio como requisito parcial para a obtenção do titulo de Bacharel em Engenharia da Computação.

Orientador: Prof. Adriano Francisco Branco

Rio de Janeiro
Abril de 2024

Antenor Moreira de Barros Leal

Graduando em Engenharia da Computação na PUC - Rio

Ficha Catalográfica

Leal, Antenor Moreira de Barros

Aplicativo web de auxílio a navegação aérea / Antenor Moreira de Barros Leal; orientador: Adriano Francisco Branco. – 2024.

32 f: il. color. ; 30 cm

Projeto Final - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2024.

Inclui bibliografia

1. Informática – Trabalho de Conclusão de Curso (Graduação). 2. Aviação. 3. Navegação. 4. Aplicativo. 5. Algoritmo. 6. Web. 7. Internet. I. Branco, Adriano Francisco. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

xxxxxxxxxxxxxxxxxxxxxxxxx Dedicatória aqui xxxxxxxxxxxxxxxxxxxxxxxxx

Resumo

Leal, Antenor Moreira de Barros; Branco, Adriano Francisco. **Aplicativo web de auxílio a navegação aérea**. Rio de Janeiro, 2024. 32p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

É um aplicativo web de código aberto como objetivo de auxiliar usuários de simuladores de voo que não possuem acesso à ferramenta (Electronic Flight Bag) que um piloto de linha aérea teria. Ao acessar o aplicativo, o usuário se depara com a lista de aeroportos cadastrados e, após escolher um, são exibidas informações da pista, frequências do aeroporto (torre, solo, ATIS, etc.), e frequências de navegação (ILS, VOR, etc.). Também são apresentadas as informações das condições meteorológicas atuais do aeródromo (vento, visibilidade, temperatura, etc.), tanto no formato oficial (METAR), obtidas a cada hora de uma API externa, como em um texto em linguagem natural para melhor entendimento do jogador iniciante. Um usuário com permissão de administrador pode adicionar e editar aeroportos. A partir de informação atual vento, a pista em uso é calculada.

Palavras-chave

Aviação; Navegação; Aplicativo; Algoritmo; Web; Internet.

Abstract

Leal, Antenor Moreira de Barros; Branco, Adriano Francisco (Advisor). **Aerial navigation aid web application**. Rio de Janeiro, 2024. 32p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

It is an open source web application to auxialiate the flight simulator's users that doesn't have access to the tool (Electronic Flight Bag) that an airline pilot would have. When accessing the app, the users encounters the list of registered airports and, after choosing one, runway information, airport frequencies (tower, ground, ATIS etc) and navigation frequencies (ILS, VOR etc) are showed. Also provided are the current meteorological conditions of the aerodrome (wind, visibility, temperature, etc.), both in the official format (METAR), obtained hourly from an external API, and in natural language text for better understanding by novice players. An administrator user can add and edit airports. Based on the current wind information, the active runway is calculated.

Keywords

Aviation; Navigation; Application; Algoritm; Web; Internet.

Sumário

Lista de figuras

Lista de tabelas

Lista de Códigos

xxxxxxx epígrafe

epígrafe xxxxxxxxxxxxxxxx, *epígrafe xxxxxxxxx.*

1
Introdução

Com o aumento da capacidade de passageiros e carga e a necessidade de uma maior segurança, começou a se fazer necessário trazer ao cockpit vários documentos como checklist de procedimentos; log book; cartas de navegação, de saída, de aproximação, do aeródromo; tabelas de performance da aeronave etc.

Para levar tudo isto costumava-se usar uma maleta (a Flight Bag), obviamente esta ficava muito pesada.

Com a miniaturização dos computadores e surgimentos dos tablets, começaram a ser desenvolvidos programas que substituíam partes ou todos estes documentos, é a chamada maleta de voo eletrônica, mais conhecida pela sigla em Inglês EFB (*Electronic Flight Bag*).

Atualmente existem hardware dedicados para esta função, mas é mais comum se usar um tablet com um aplicativo disponibilizado pela companhia aérea.

2
Sistemas Similares

Os EFBs possuem funções variadas como cálculo de combustível, de performance. Para aeronaves mais novas como o Airbus A320 é difícil realizar cálculos de performance, porque não é disponibilizado ao público como este cálculo é feito. Ferramentas encontradas na Internet [1] normalmente fazem engenharia reversa, e portanto, podem apresentar resultados diferentes de um cálculo oficial.

Nos simuladores de voo para computador pessoal, algumas aeronaves simulam este equipamento como o Airbus A320neo desenvolvido pela *FlyByWire Simulations*. Apesar de ser uma aeronave *freeware*, ela é bem sofisticada chegando ao nível de realismo da *Fenix Simulations* ou da *ToLiss Simulations*, duas produtoras com modelos pagos do A320.

Figura 2.1: Exemplo de um EFB no Flight Simulator 2020 na aeronave A320neo

Contudo, o METAR do aeródromo não se encontra disponível no EFB. É possível usar o computador de bordo da aeronave (FMC) e conseguir esta informação. Também é possível sintonizar na frequência do ATIS, mas isto só funcionará se o avião já estiver perto do aeródromo.

O que muitos jogadores fazem é acessar o AISWEB (`https://aisweb.decea.mil.br/`) , sistema oficial brasileiro de informações aeronáuticas.

É um site extremamente completo, podendo ser usado em operações reais, mas para o jogador iniciante seria de valia uma interface mais simples.

Figura 2.2: AISWEB com informações de pista, frequências de comunicação e navegação para o Santos Dumont

O AISWEB exibe o METAR no aeroporto, mas não explica para o que cada campo serve.

O site METAR-TAF (`https://metar-taf.com/`) é o decoder mais conhecido, possui uma interface gráfica bem construída e muito fácil de entender, mas não possui a lista de frequência dos aeroportos e de radionavegação.

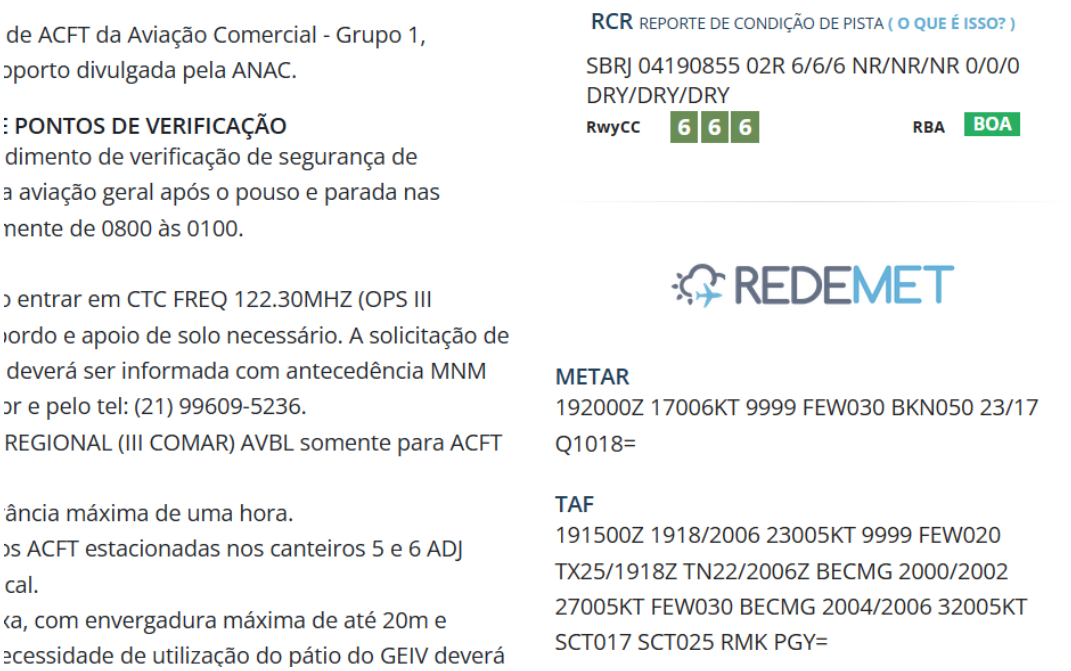

Figura 2.3: METAR do Santos Dumont no AISWEB

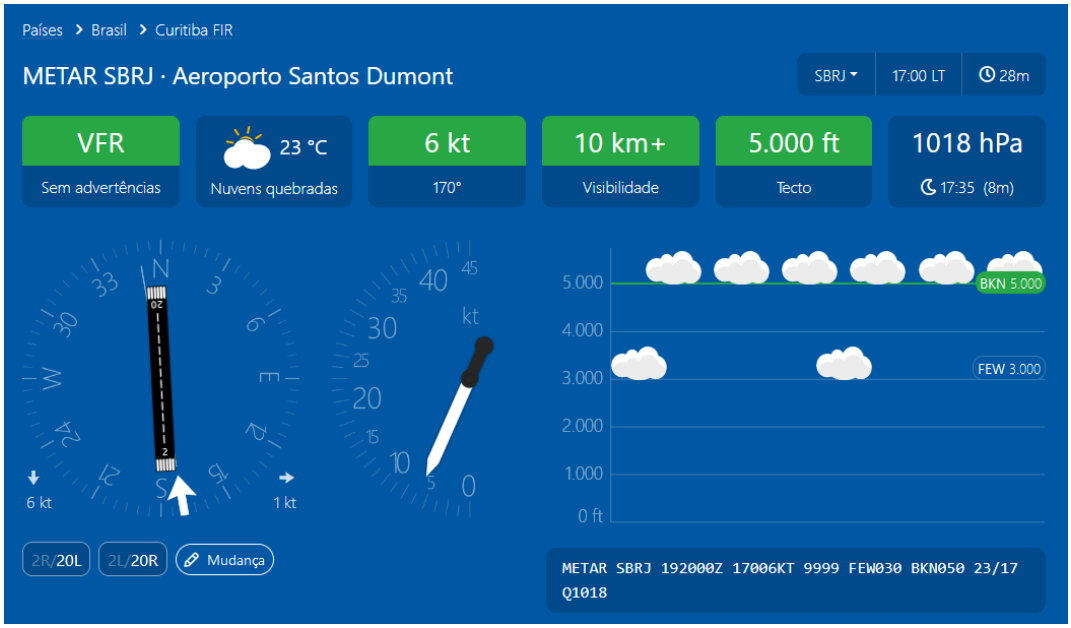

Figura 2.4: Interface gráfica do METAR-TAF

3
A Proposta

A ideia do trabalho seria unir as funcionalidades do METAR-TAF com o AISWEB de forma que o usuário iniciante consiga usar sem dificuldades.

Pelo fato de aviação necessitar ter um ambiente seguro e bastante regulado, considerando que meu projeto é apenas um protótipo, prefiro restringir o caso de uso apenas para jogadores de simuladores de voo que desejam que a simulação seja parecida com o real. Nas páginas do sistema conterá um aviso de que o sistema **não deve ser usado para um voo real**.

Dito isto, o sistema possui backend escrito na linguagem Python fazendo uso da biblioteca Flask. A renderização de página é server-side, usando a funcionalidades de templates do Flask junto com a biblioteca Jinja2.

No segundo semestre de 2023 comecei a fazer um projeto para uso próprio. O código está disponível em `https://github.com/antenor-z/aero`. Atualmente o projeto funciona, mas a arquitetura foi feita sem muito planejamento, as informações do aeroporto são hardcoded.

O usuário tem acesso a informações de frequência da torre, solo, tráfego, rampa e operações, bem como das frequências e dados para VOR (um sistema de radionavegação por antenas no solo), ILS (sistema de pouso por instrumentos) e informações de pista. Neste trabalho quero, armazenar estas em um banco de dados relacional com uma arquitetura bem planejada. Farei testes de desempenho simulando uma alta taxa de acesso e, dependendo dos resultados, fazer uso de um banco em memória como intermediário.

Os aerodromos podem ao longo do tempo mudarem alguma frequência e outras informações como o número da pista mudam a depender da variação do norte magnético, uma pista pode ser ampliada etc. Atualmente o código precisa ser alterado para atualizar as informações. Desejo implementar um sistema diretamente no site, com uma autenticação por senha e TOTP, para que seja possível mudar qualquer informação no banco.

Através de uma API do serviço americano National Weather Service, são coletadas as informações atuais de meteorologia. Estas informações (que vêm em um formato chamado METAR) são processadas pelo backend e mostradas ao usuário de uma forma fácil de entender. Esta parte em específico possui um código de difícil manutenção, desejo refatorar esta parte e adicionar suporte para a maior parte de códigos da especificação do METAR.

4
Cronograma

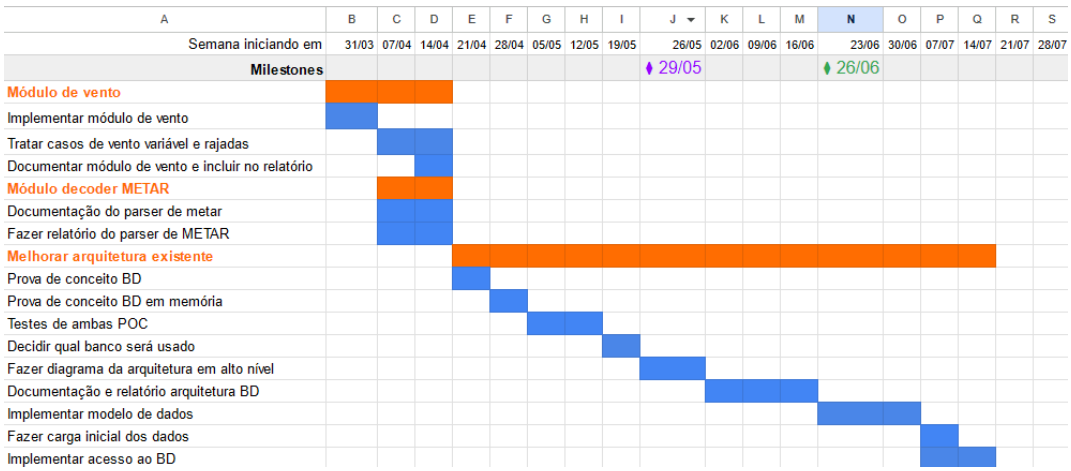

Figura 4.1: Cronograma planejado

Tabela 4.1: Milestones

Data	**Milestone**
29/05	Entrega da proposta de Projeto Final I
26/06	Entrega do relatório de Projeto Final I

5
Modelo de Dados

5.1
Modelo Lógico

O modelo de dados foi feito para ser o mais simples possível para que futuras alterações possam ser feitas facilmente. Abaixo está o diagrama entidade-relacionamento no qual o nome no topo do retângulo identifica o nome da tabela. A lista abaixo mostra as colunas dessas tabelas, com um ícone de uma chave preta para a *chave primária* e uma chave verde para a *chave estrangeira*.

Uma relação é denotada pelas setas ligando duas tabelas. A cardinalidade da relação é indicada pelos números entre parênteses.

Para a relação `Aerodrome-ILS`, temos (1, 1) para (0, n), significando que um aeródromo pode ter zero ou mais frequências de ILS e esta só pode ser de um único aeródromo.

Já na relação `Aerodrome` com `Runway`, um aeródromo deve ter `uma ou mais` pistas.

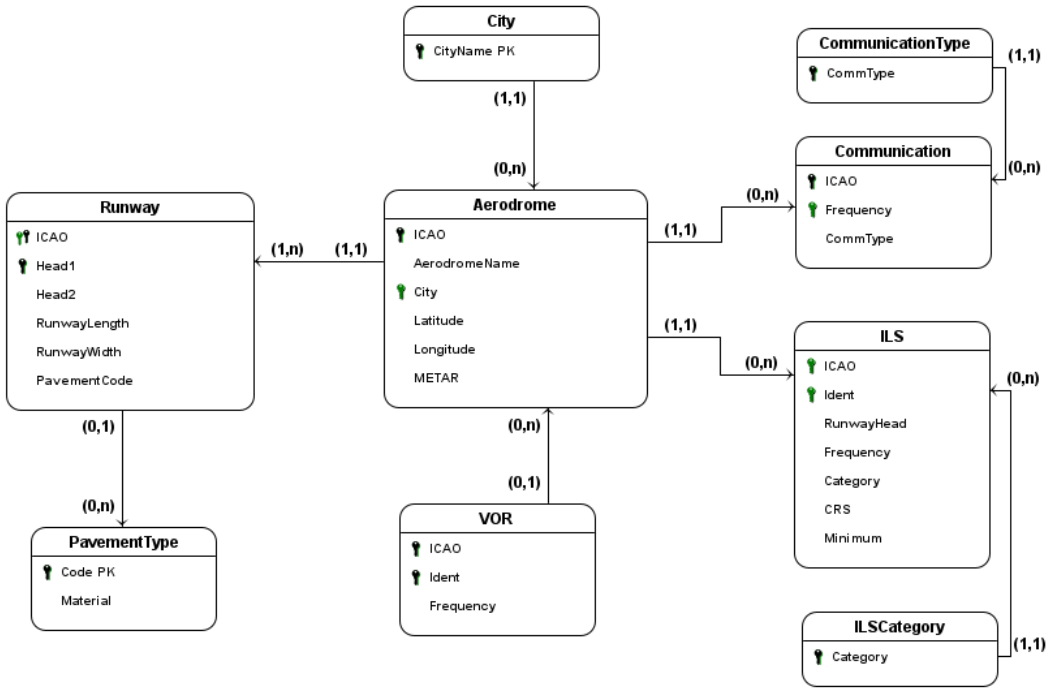

Figura 5.1: Diagrama E/R

Note que as tabelas "CommunicationType", "ILSCategory"e "PavementType"poderiam ser substituídas por colunas enum nas tabelas "Commu-

nication", "ILS"e "Runway", porém a manutenção seria difícil [2], pois teríamos que alterar a estrutura das tabelas (possívelmente tirando o sistema do ar) caso fosse necessário adicionar um tipo novo de comunicação, por exemplo. Fazendo com uma tabela externa é necessário apenas adicionar uma nova linha.

5.2 Tabelas

Tabela 5.1: City

Nome	**Descrição**	**Tipo**
CityName (PK)	Nome da cidade escrito em Português com a primeira letra maiúscula	VARCHAR (50)

Tabela 5.2: Aerodrome

Nome	**Descrição**	**Tipo**
ICAO (PK)	O código ICAO do aeródromo emitido pela Organização Internacional de Aviação Civil (ICAO).	VARCHAR(4)
AerodromeName	O nome do aeródromo conforme definido pelo AISWEB, sistema nacional de informações aeronáuticas.	VARCHAR(50)
City (FK)	Nome da cidade escrito em Português. A primeira letra é maiúscula. Chave estrangeira para a tabela `City`.	VARCHAR(30)
Latitude	A latitude do aeroporto em graus no formato de graus decimais (DD, Decimal Degrees). Três dígitos para representar a parte inteira e seis dígitos para a fracionária.	DECIMAL(9, 6)

Continua na próxima página

Tabela 5.2 – Continuação da página anterior

Nome	**Descrição**	**Tipo**
Longitude	A longitude do aeroporto, seguindo o mesmo formato da latitude.	DECIMAL(9, 6)
METAR	O METAR válido atualmente para este aeródromo.	VARCHAR(100)

Considerei colocar o METAR como tabela separada com chave estrangeira para o aerodrómo para, deste modo, ter o METAR histórico. Mas, no contexto de planejamento de voo, os METARs anteriores não possuem muita serventia.

Tabela 5.3: PavementType

Nome	**Descrição**	**Tipo**
Code (PK)	O código (em Inglês) do tipo de pavimento usado. É formado por três letras maiúsculas.	VARCHAR(3)
Material	O nome do pavimento em Português, com a primeira letra maiúscula.	VARCHAR(20)

```
Exemplo de siglas:

Code    Material
ASP     Asfalto
CON     Concreto
GVL     Brita
```

É importante conhecer as características dos tipos de pistas de pouso e decolagem, pois seu comprimento determinará a quantidade de freio necessária para parar uma determinada aeronave.

Se a pista for muito curta, determinados modelos de avião não poderão pousar. A largura da pista determina a envergadura máxima que uma aeronave pode ter para operar nessa pista. Se uma pista for muito estreita, uma aeronave quadrimotora como o Boeing 747 pode sofrer ingestão de materiais, já que os dois motores mais externos ficarão para fora da área da pista, sobre o gramado.

Tabela 5.4: Runway

Nome	**Descrição**	**Tipo**
ICAO (FK e PK)	O código ICAO do aeródromo ao qual a pista está associada, utilizado como chave estrangeira fazendo a ligação com a tabela 'Aerodrome'.	VARCHAR(4)
Head1 (PK)	Número e possível letra que identifica uma das cabeceiras da pista. Um aeroporto nunca terá cabeceiras repetidas, então ICAO e Head1 formam uma chave primária mínima.	VARCHAR(3)
Head2 (PK)	O mesmo, mas para o outra cabeceira.	VARCHAR(3)
RunwayLength	Comprimento da pista em metros.	INTEGER
RunwayWidth	Largura da pista em metros.	INTEGER
PavementCode (FK)	O tipo de pavimento da pista, referenciando a tabela 'PavementType'.	VARCHAR(3)

Para cabeceiras paralelas, ou seja, que apontam para a mesma direção, temos:

5.2.1 Pista única

A proa em que a pista aponta, com divisão por 10 arredondada. Por exemplo, em Fortaleza, temos uma cabeceira com curso de 126 graus, dividindo por 10 temos 12,6, arredondando temos o número 13 da cabeceira.

5.2.2 Pista dupla

Para duas pistas paralelas, usamos L para a cabeceira da esquerda e R para a da direita. Por exemplo, no Santos Dumont, de costas para o Pão de Açúcar, temos as cabeceiras 02L (na esquerda) e 02R (na direita).

5.2.3 Pista tripla

Não temos aeroportos com três pistas paralelas no Brasil, mas são usadas as letras L, C e R. C para a pista central.

Tabela 5.5: CommunicationType

Nome	**Descrição**	**Tipo**
CommType (PK)	O tipo de comunicação, podendo ser "Torre", "Solo", "ATIS", "Tráfego"ou "Operação". Mais adiante outros tipos podem ser adicionados.	VARCHAR(10)

Tabela 5.6: Communication

Nome	**Descrição**	**Tipo**
ICAO (PK e FK)	O código ICAO do aeródromo ao qual a frequência de comunicação está associada, utilizado como chave estrangeira referenciando a tabela 'Aerodrome'.	VARCHAR(4)
Frequency (PK)	A frequência em MHz multiplicada por 1000. Já que as frequências de comunicação possui três digitos decimais, multiplicamos por mil para armazenar em inteiro de ponto fixo. ICAO e frequency formam chave primária e usar um DECIMAL para uma PK, não é muito eficiente. Note que uma frequência, não necessariamente é única em todo o país, para distâncias longas, onde não há risco de interferência, é possível haverem frequências repetidas.	INTEGER

Continua na próxima página

Tabela 5.6 – Continuação da página anterior

Nome	**Descrição**	**Tipo**
CommType (FK)	O tipo de comunicação, chave estrangeira para 'CommunicationType'.	VARCHAR(20)

Esta tabela lista as diferentes categorias de Sistema de Pouso por Instrumentos (Instrument Landing System).

Tabela 5.7: ILSCategory

Nome	**Descrição**	**Tipo**
Category (PK)	A categoria de ILS, sendo "CAT I", "CAT II", "CAT IIIA", "CAT IIIB"ou "CAT IIIC". Será explicado melhor em "Minimus"na tabela "ILS".	VARCHAR(10)

Tabela 5.8: ILS

Nome	**Descrição**	**Tipo**
ICAO (PK e FK)	O código ICAO do aeródromo ao qual o sistema de pouso está associado, utilizado como chave estrangeira referenciando a tabela 'Aerodrome'.	VARCHAR(4)
Frequency (PK)	A frequência de operação do ILS em MHz, multiplicado por 10. Fazemos isso para poder usar o tipo INTEGER, já que um DECIMAL como chave primária não seria eficiente, como já explicado na tabela de comunicação.	INTEGER

Continua na próxima página

Tabela 5.8 – Continuação da página anterior

Nome	**Descrição**	**Tipo**
Ident	Identificação de três letras maisculas única do ILS para aquele aeródromo. Aparece na carta aérea do procedimento ILS.	VARCHAR(3)
RunwayHead	Para qual pista este ILS se refere.	VARCHAR(3)
Category (FK)	A categoria do ILS, referenciando a tabela 'ILSCategory'.	VARCHAR(10)
CRS	A referência do curso de aproximação do ILS. É a proa final que a aeronave deve manter para o correto alinhamento nesta cabeceira.	INTEGER
Minimum	A altura mínima de decisão em pés para operação do ILS. A partir desta altura, é desligado o piloto automático e o resto da aproximação é feita manualmente. Se a altitude da aeronave ficar abaixo deste valor e ainda não for possível ter visual da pista é obrigatória a arremetida. Quando maior a categoria do ILS, maior a precisão do sistema, portanto a Minimus será mais baixa. Uma "CAT IIIC"(pronuncia-se cat três charlie), possui Minimus zero, portanto a aeronave pode pousar de forma totalmente automática.	INTEGER

Esta tabela registra os sistemas de navegação VOR/DME disponíveis em um aeródromo. Não foi incluída uma tabela para as frequências de NDB porque este sistema está caíndo em desuso.

Tabela 5.9: VOR

Nome	**Descrição**	**Tipo**
ICAO (PK e FK)	O código ICAO do aeródromo ao qual o VOR/DME está associado, utilizado como chave estrangeira referenciando a tabela 'Aerodrome'.	VARCHAR(4)
Frequency (PK)	A frequência de operação do VOR/DME em MHz multiplicada por 10.	INTEGER
Ident	Identificação única do VOR/DME para aquele aeródromo. Forma chave primária junto com ICAO.	VARCHAR(3)

6
Arquitetura

6.1
Introdução

A arquitetura foi pensada para ser implementada com o Docker e Docker Compose. Cada programa que precisa ser executado é rodado em um serviço separado. Exceto, o Guinicorn, todos os outros serviços usaram imagens prontas do Docker Hub o que oferece mais segurança com as atualizações constantes.

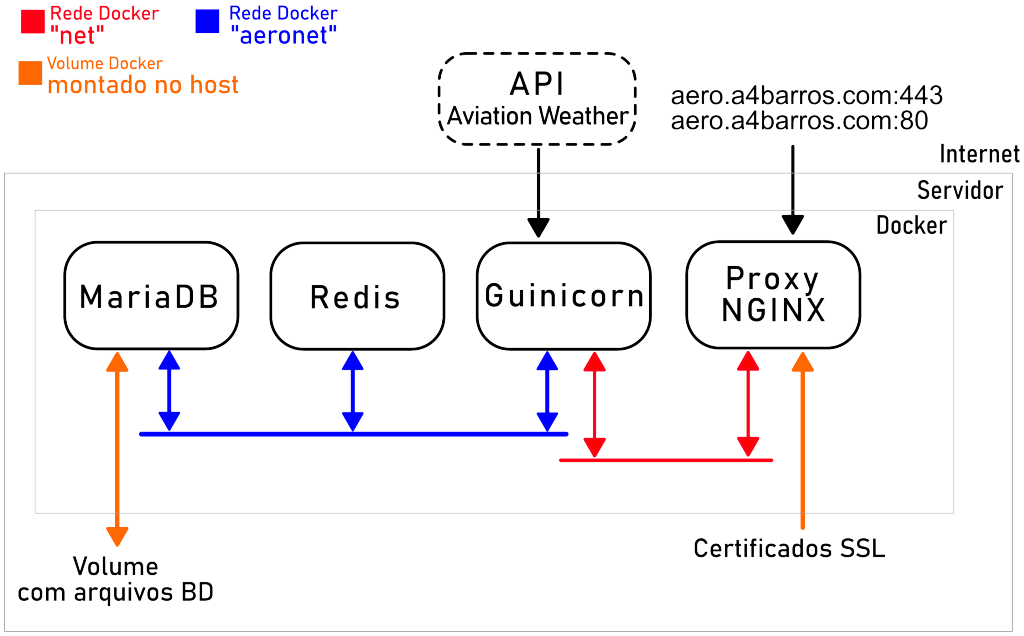

Figura 6.1: Modelo de Arquitetura

6.2
Docker Network

A porta 5000 do Guinicorn não estará disponível para todos os serviços. Por segurança, é usada a função de "network". Observe no diagram que a proxy NGINX compartilha com o Guinicorn a rede "net", para o NGINX, o Guinicorn não pode ser acessado por `localhost:5000` e sim por `http://aero:5000`, "aero"sendo o nome do serviço no Docker Compose.

Já o banco de dados em memória Redis e o banco relacional MariaDB compartilham a rede Docker "aeronet". Apenas eles dois e o Guinicorn possuem

acesso a ela, isto é necessário para que o servidor consiga ler e escrever valores nas tabelas do banco e ler e escrever chaves-valor no Redis. Porém o NGINX não possui acesso aos dois bancos, por mais que os bancos tenham senhas, é uma barreira a mais contra uma invasão.

6.3 Docker Secrets

Para aumentar a segurança de acesso aos dois banco é usada a função "secrets". Nela, no Docker Compose, você informa um arquivo de texto no host onde estará uma senha, uma senha por arquivo. No mesmo Compose, você informa quais serviços tem acesso a cada senha, na execução dos containers, as senhas são guardadas dentro de arquivos montados no caminho "/run/secret/algum-nome.txt"em que "algum-nome"é o mesmo nome do arquivo que estava no host. Tanto os bancos como o servidor Guinicorn usam este método para terem acessos às senhas dos bancos.

6.4 Serviços

6.4.1 MariaDB

Este banco de dados relacional guarda toda a informação mais ou menos fixa sobre os aerodromos conforme explicado no capítulo de modelo de dados.

6.4.2 Redis

O Redis é um banco de acesso mais rápido por estar com as informações salvas em memória em vez de disco rígido/SSD. Ele é um espelho do que existe no banco relacional, claro que com as devidas modificações já que o Redis possui sistema-chave valor, não sendo relacional.

6.4.3 Proxy NGINX

Faz o HTTPS funcionar, dá suporte ao HTTP/2 e ao header HTTP keep-alive. O Guinicorn só tem suporte ao primeiro. De qualquer caso a documentação do Guinicorn não recomenda que ele esteja diretamente ligado a Intenet [3]. Já que tenho outros projetos na mesma máquina, uso subdomínios. Na configuração do NGINX o server block com hostname aero.a4barros.com é redirecionado para o endereço interno "https://aero:5000".

6.5 Flask/Guinicorn

Já que o Guinicorn é o serviço principal: o servidor onde o backend implementado em Flask roda, fiz um Dockerfile próprio iniciando a partir de uma imagem do Alpine, devido ser lightweight. O Python e as dependencias do projeto são instaladas automaticamente pelo Dockerfile e por fim o arquivo de entrada `server.py` é executado usando o servidor Guinicorn. As configurações dele ficam no arquivo `gunicorn_config.py` é apenas configura a porta para 5000 e usa três workers.

A documentação do Guinicorn informa a seguinte fórmula para determinar a quantidade de workers. [4]

$$\begin{aligned} N_{worker} &= 2 * N_{cores} + 1 \\ N_{worker} &= 2 * 1 + 1 = 3 \end{aligned} \tag{6-1}$$

6.6 Produção

O site se encontra no produção no endereço https://aero.a4barros.com. Ele está hospedado em uma VPS com as seguintes características:

- **CPU:** AMD EPYC 7713 (1 core disponível) @ 2GHz
- **RAM:** 1GB
- **Armazenamento:** 25GB
- **SO:** Ubuntu 22.04.4 LTS

Mesmo com uma configuração bastante modesta, o sistema roda sete containers Docker usando aproximadamente metade da memória primária (RAM) em idle.

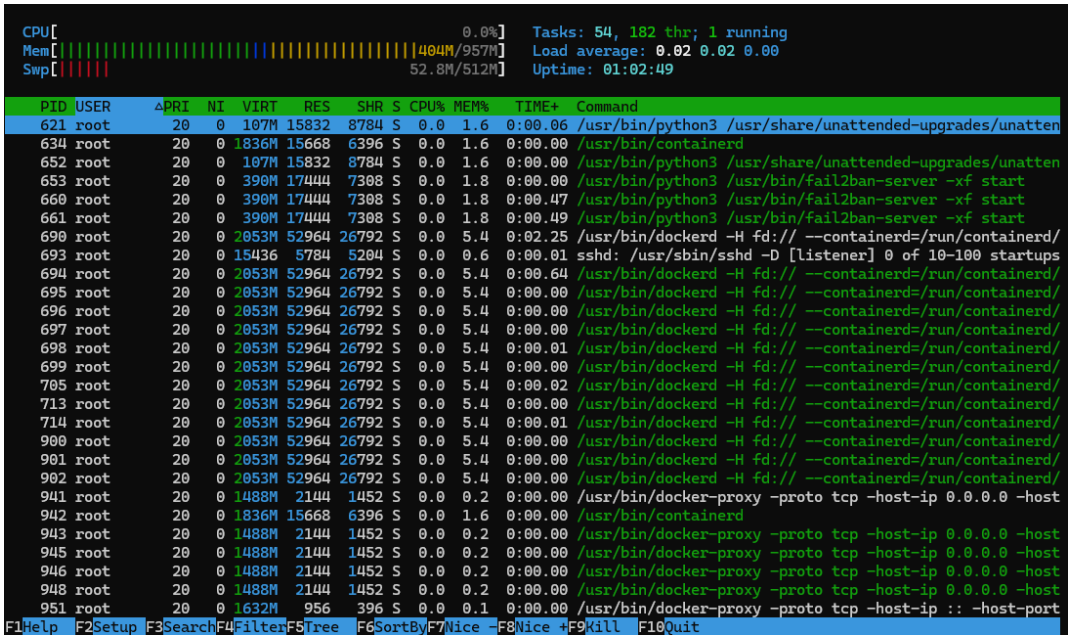

Figura 6.2: Uso do sistema em baixa demanda

Figura 6.3: Containers Docker em execução

7
Front-end

Como falado no capítulo da arquitetura, a página é gerada server-side, então o que é retornado para cada rota é um HTML já pronto. Acredito que, para o meu caso, é mais performático fazer assim do que usar páginas com Javascript que fazem requisição para uma API REST.

De todo modo, no final da execução de uma rota, um dicionário Python é gerado, algo que poderia ser facilmente convertido para um JSON usando a função `dumps()` da biblioteca `json` do próprio Python. No caso desta aplicação, este dicionário é enviado para o template usando a função `render_template()` da biblioteca Jinja2, que recebe o nome da página HTML com o template e um número qualquer de *kwargs* (argumentos nomeados) que podem ter qualquer tipo serializável, incluindo dicionários.

Perceba que não há problema de acoplamento fazendo deste modo, pois não há código HTML sendo escrito dentro do backend. Como já dito, o que é passado para o Jinja é algo equivalente a JSON.

Um página template é um arquivo HTML com *placeholders* que serão substituídos pelos *kwargs* de mesmo nome. O Jinja2 tem estruturas de repetição para que um código HTML possa ser repetido usando valores da lista. E, no caso de dicionários, é fácil acessar os valores. Neste projeto, para exibir a lista de frequência, o seguinte código é usado.

Código 1: Formatação Antiga

```
<div class="line">
    {% for comm in info.communication %}
    <div class="box-rounded box-outlined">
        <div class="box-name">{{comm.CommType}}</div>
        <div class="box-value">
            {{"{:,.3f}".format(comm.Frequency/1000)}}
        </div>
    </div>
    {% endfor %}
</div>
```

Note que é possível fazer operações e formatações simples no Jinja2. Já que a frequência é armazenada no banco como um inteiro de ponto fixo (como dito no capítulo de modelo de dados), aqui eu poderia dividir o valor por mil e formatar com três casas decimais para que, caso uma frequência termine com zeros à direita, sempre tenhamos três dígitos decimais, que é o padrão para frequências de comunicação.

Porém, para não misturar a *view* com o *controller*, deixei este processamento de dados para uma função dentro do controlador.

Código 2: Formatação Atual

```
<div class="line">
    {% for comm in info.communication %}
    <div class="box-rounded box-outlined">
        <div class="box-name">{{comm.CommType}}</div>
        <div class="box-value">
            {{ comm.Frequency }}
        </div>
    </div>
    {% endfor %}
</div>
```

No código do projeto, na pasta 'templates', é possível ver todos os templates usados.

8
Rotas do back-end

8.1
Rota raiz

Página inicial, apresenta a lista de aeródromos para que o usuário escolha um. Internamente, por meio do SQLAlchemy, é feita a seleção dos campos `AerodromeName`, `ICAO` e `City` e o resultado é posto em uma lista para cada um dos aeródromos e enviado para a ferramenta de template criar a página.

Exemplo do resultado do banco:

```
[
    ('Presidente Juscelino Kubitschek', 'SBBR', 'Brasília'),
    ('Tancredo Neves', 'SBCF', 'Belo Horizonte'),
    ('Afonso Pena', 'SBCT', 'Curitiba'),
    ('Pinto Martins', 'SBFZ', 'Fortaleza'),
    ...
]
```

8.2
Rota: /info/{ICAO}

8.3
Rota: /wind/{ICAO}

9
Referências bibliográficas

1 PRADZ. *Performance Calculation*. 2024. Disponível em: <http://perfcalc.pradz.de/index.php>. Acesso em: 25 abril 2024.

2 EMELYANOV, S. *The Pros and Cons of Enum Data Type in Database Design*. 2023. Disponível em: <https://www.linkedin.com/pulse/pros-cons-enum-data-type-database-design-sergey-emelyanov>. Acesso em: 17 abril 2024.

3 CHESNEAU, B. *Deploying Gunicorn*. 2024. Disponível em: <https://docs.gunicorn.org/en/latest/deploy.html>. Acesso em: 24 abril 2024.

4 CHESNEAU, B. *How Many Workers*. 2024. Disponível em: <https://docs.gunicorn.org/en/latest/design.html>. Acesso em: 24 abril 2024.