

Antenor Moreira de Barros Leal

**Aplicativo web de auxílio à
navegação aérea**

PROJETO FINAL

DEPARTAMENTO DE INFORMÁTICA
Programa de Graduação em Engenharia da
Computação

Rio de Janeiro
Novembro de 2024



Antenor Moreira de Barros Leal

Aplicativo web de auxílio à 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 título de Bacharel em Engenharia da Computação.

Orientador: Prof. Adriano Francisco Branco

Rio de Janeiro
Novembro de 2024

Todos os direitos reservados. A reprodução, total ou parcial do trabalho, é proibida sem a autorização da universidade, do autor e do orientador.

Antenor Moreira de Barros Leal

Graduando em Engenharia da Computação na PUC - Rio

Ficha Catalográfica

Leal, Antenor Moreira de Barros

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

64 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

Resumo

Leal, Antenor Moreira de Barros; Branco, Adriano Francisco. **Aplicativo web de auxílio à navegação aérea**. Rio de Janeiro, 2024. 64p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

É um aplicativo web de código aberto com o 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 as 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 (.ETAR), 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. Módulos adicionais estão disponíveis como o cálculo da pista em uso (a partir de informações do vento) e do perfil de descida.

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. 64p. 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 don't have access to the tool (Electronic Flight Bag) that an airline pilot would have. When accessing the app, the users encounter 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. Additional modules are available such as calculation of the runway in use (from wind information) and the descent profile.

Keywords

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

Sumário

1	Introdução	12
2	Sistemas Similares	13
3	A Proposta	16
4	Cronograma	18
4.1	Cronograma do Projeto Final I	18
4.2	Cronograma do Projeto Final II	19
5	Princípios norteadores	20
5.1	Correção da Informação Apresentada	20
5.2	Rapidez no Carregamento das Páginas	20
5.3	Facilidade de Uso do Sistema	20
6	Modelo de Dados	21
6.1	Modelo Lógico	21
6.2	Tabelas	21
6.3	Adicionado no Projeto II	29
7	Arquitetura	31
7.1	Docker Network	31
7.2	Docker Secrets	31
7.3	Serviços	32
7.4	FastAPI	32
7.5	Produção	33
7.6	Operações Síncronas e Assíncronas	34
7.7	Diagrama de sequência	35
8	Decodificação do METAR	37
8.1	Exemplo	37
8.2	Algoritmo	39
8.3	Complexidade Temporal	39
9	Decodificação do TAF	41
9.1	Exemplo	41
9.2	Algoritmo	42
9.3	Complexidade Temporal	42
10	Plotagem do METAR Histórico	44
11	Front-end	46
11.1	Minificação	47
12	Rotas do back-end	49
12.1	Rota raiz	49

12.2 Rota: /info/{ICAO}	49
12.3 Rota: /history/{ICAO}	50
12.4 Rota: /taf/{ICAO}	50
13 Conclusão e Próximos Passos	51
14 Apêndice A: Códigos Relevantes	52
15 Referências bibliográficas	63

Lista de figuras

Figura 2.1	Exemplo de um EFB no Flight Simulator 2020 na aeronave A320neo	13
Figura 2.2	AISWEB com informações de pista, frequências de comunicação e navegação para o Santos Dumont	14
Figura 2.3	METAR do Santos Dumont no AISWEB	15
Figura 2.4	Interface gráfica do METAR-TAF	15
Figura 4.1	Cronograma	18
Figura 4.2	Cronograma 2	19
Figura 6.1	Diagrama E/R	22
Figura 7.1	Modelo de Arquitetura	31
Figura 7.2	Uso do sistema em baixa demanda	33
Figura 7.3	Containers Docker em execução	33
Figura 7.4	Diagrama de sequência	36
Figura 10.1	Gráficos do aeroporto Aeroporto Internacional do Recife do dia 28 de agosto de 2024	45

Lista de tabelas

Tabela 4.1	Milestones	18
Tabela 4.2	Milestones	19
Tabela 6.1	City	22
Tabela 6.2	Aerodrome	22
Tabela 6.3	METAR	23
Tabela 6.4	TAF	23
Tabela 6.5	PavementType	24
Tabela 6.6	Runway	25
Tabela 6.7	CommunicationType	26
Tabela 6.8	Communication	26
Tabela 6.9	ILSCategory	27
Tabela 6.10	ILS	27
Tabela 6.11	VOR	29
Tabela 6.12	User	29
Tabela 7.1	Operações assíncronas	34
Tabela 11.1	Com e Sem minification	48

Lista de Abreviaturas

METAR – *ME*Teorological Aerodrome Report (Informe Meteorológico de Aeródromo)

EFB – *Electronic Flight Bag* (Maleta Eletrônica de Voo)

ICAO – *International Civil Aviation Organization* (Organização Internacional da Aviação Civil)

IATA – *International Air Transport Association* (Associação Internacional de Transporte Aéreo)

ATIS – *Automatic Terminal Information Service* (Serviço Automático de Informação Terminal)

PK – *Public Key* (Chave Pública em banco de dados)

FK – *Foreign key* (Chave Estrangeira em banco de dados)

SSH – *Secure SHell* (Shell Segura)

VPS – *Virtual Private Server* (Sevidor Virtual Privado)

ISP – *Internet Service Provider* (Provedor de Serviços de Internet)

API – *Application Programming Interface* (Interface de Programação de Aplicações) REST – *Representational State Transfer* (Transferência de Estado Representacional)

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. Normalmente, o tablet escolhido é um iPad da Apple, mas algumas companhias optaram pelo Microsoft Surface. [1]

O uso do EFB trouxe uma série de benefícios para os pilotos e para as companhias aéreas. Além de reduzir o peso e o volume de documentos físicos a serem transportados, o EFB permite uma rápida atualização das informações, garantindo que os pilotos tenham sempre acesso às versões mais recentes das cartas de navegação. [2]

Além disso, a capacidade de armazenamento do EFB possibilita o acesso a uma vasta quantidade de informações adicionais, como manuais de operação da aeronave, regulamentações atualizadas e até mesmo dados meteorológicos em tempo real, o que contribui para uma tomada de decisão mais informada e segura durante o voo.

2

Sistemas Similares

Os EFBs possuem funções variadas como cálculo de combustível, de performance, etc. 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 [3] 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.

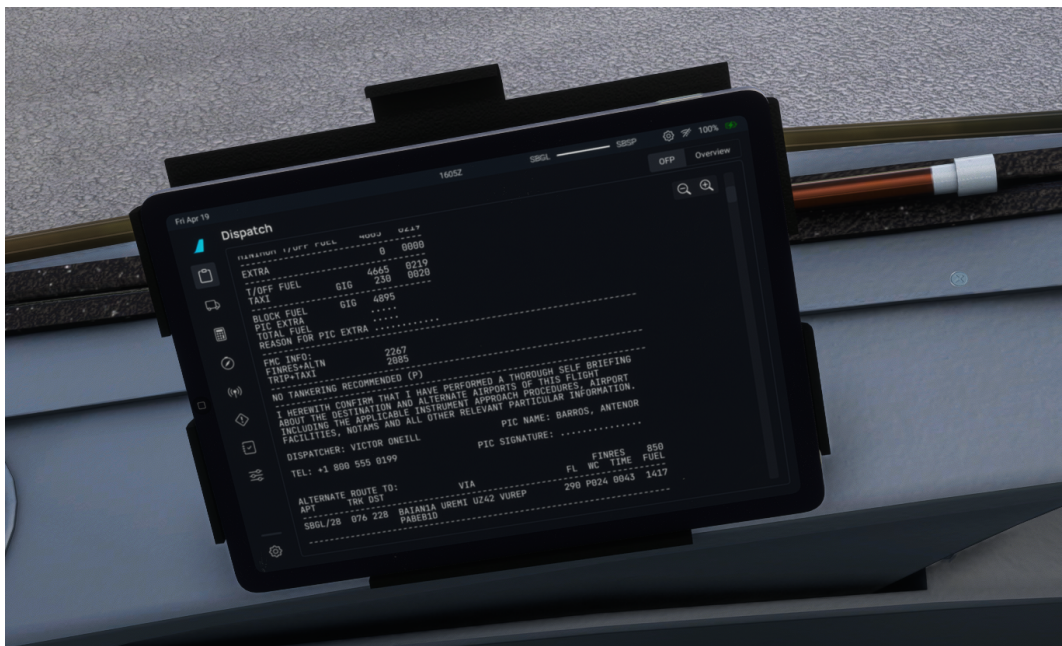


Figura 2.1: Exemplo de um EFB no Flight Simulator 2020 na aeronave A320neo

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.



> Abreviaturas > Central de Ajuda  Português

INÍCIO AERÓDROMOS ▾ ESPAÇO AÉREO ▾ NOTAM

Santos Dumont (SBRJ)

Rio de Janeiro/RJ CIAD: **RJ0002**

[ROTAER](#) [NOTAM](#) [Suplementos AIP](#) [Cartas](#) [Metar/TAF](#) [Rotas Preferenciais](#)

ROTAER D-AMDT 9/24 [Ver mudanças](#)

Santos Dumont (SBRJ) / RIO DE JANEIRO, RJ 22 54 36S/043 09 45W
 AD PUB/MIL INFRAERO 2E UTC-3 VFR IFR L21 , L23 , L26 3 (10)

02L - L9(3.23) [1] [12] [14] , L12 - (1260x30 ASPH 39/F/B/X/T L14 , L15) - L9(3.12) SBCW (CRCEA-SE)
 [11] [14] , L12 - **20R**

02R - L9(3.23) [1] [2] [14] , L12 - (1323x42 ASPH 65/F/B/X/T L14 , L15) - L9(3.12)
 [4] [14] , L12 - **20L**


COM -TORRE RIO [9] [14] 118.700 [10] 121.500 [3]
 SOLO RIO [10] [14] 121.700
 TRÁFEGO RIO [10] [14] 121.050
 ATIS RIO [8] [10] [14] 132.650

CMB- TF [5] **SER** - S1 **RFFS** - CAT CIVIL - 7

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.



METAR
192000Z 17006KT 9999 FEW030 BKN050 23/17
Q1018=

TAF
191500Z 1918/2006 23005KT 9999 FEW020
TX25/1918Z TN22/2006Z BECMG 2000/2002
27005KT FEW030 BECMG 2004/2006 32005KT
SCT017 SCT025 RMK PGY=

Figura 2.3: METAR do Santos Dumont no AISWEB

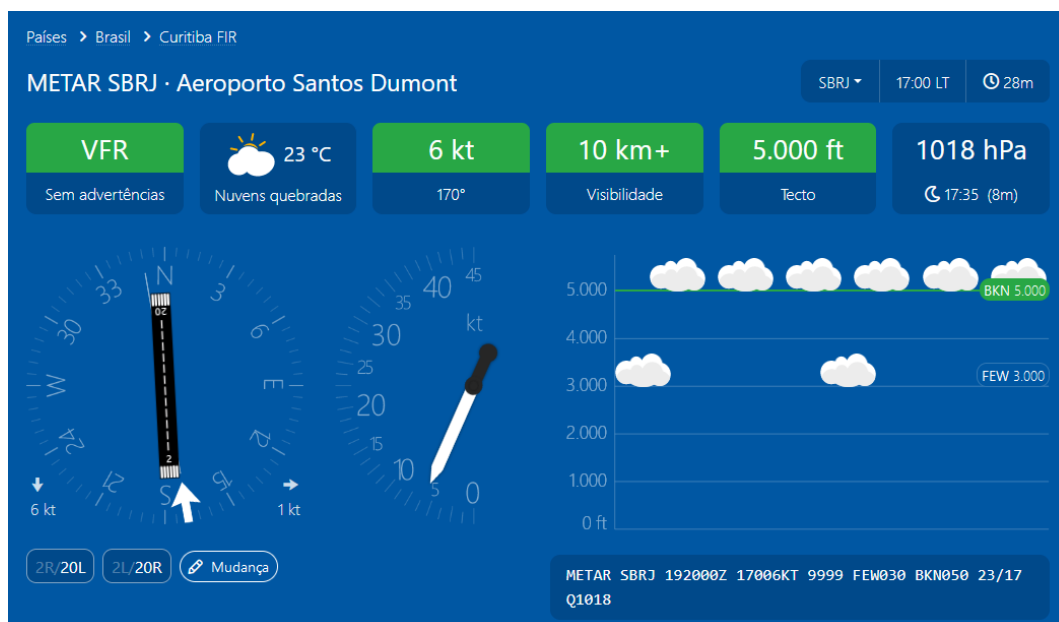


Figura 2.4: Interface gráfica do METAR-TAF

3

A Proposta

A ideia do trabalho seria unir as funcionalidades do METAR-TAF com o AISWEB em uma interface gráfica 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 conterà 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 framework FastAPI. Até a entrega do Projeto I, o backend era feito com Flask. A escolha foi feita pela minha familiaridade com a framework. Porém, depois, descobri a framework FastAPI no meu trabalho. Normalmente, ela não é usada para fullstack, mas sim para fazer APIs REST, que retornam JSON na resposta de uma requisição. Um frontend feito em alguma framework de JavaScript faria a requisição e atualizaria a página.

Contudo, é possível retornar qualquer tipo de dado no FastAPI, inclusive páginas HTML. Como prefiro fazer a renderização de páginas server-side, continuei usando a funcionalidade de templates da biblioteca Jinja2.

A substituição da framework deu-se pelos seguintes motivos:

- Projeto mais maduro, continuamente mantido.
- Documentação bem mais detalhada do que a do Flask, com vários tutoriais sobre assuntos comumente usados, como autenticação e sessão de usuário.
- Servidor embutido (Uvicorn) extremamente fácil de configurar e suficiente para produção [4].
- Assíncrono, permitindo o uso junto com bibliotecas que utilizam asyncio.

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 aeródromos podem, ao longo do tempo, mudarem alguma frequência e outras informações, como o número da pista, que muda a depender da variação do norte magnético, uma ampliação da pista, etc. Atualmente, o código precisa ser alterado para atualizar estas 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

Com o fim de se ter uma direção para o projeto e medição da evolução do projeto, ele foi dividido em várias tarefas com início e duração esquematizados abaixo.

O que está em laranja são grupos de tarefas, em preto são as tarefas em si, as células com fundo em azul são a(s) semana(s) planejadas para realizar cada tarefa. O símbolo de losango mostra quando a tarefa foi de fato feita.

Em vermelho, temos tarefas que não estavam no planejamento.

4.1

Cronograma do Projeto Final I

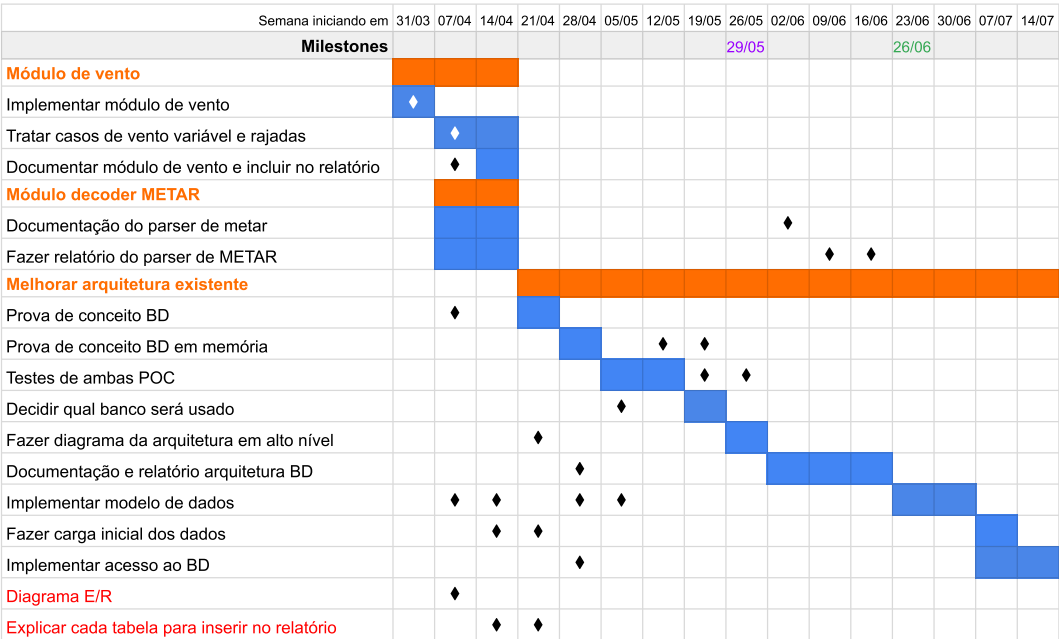


Figura 4.1: Cronograma

Os dias 29/05 e 26/06 são milestones onde ocorrem as entregas.

Tabela 4.1: Milestones

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

Por ter sido meu primeiro grande projeto autogerido, tive um pouco de dificuldade em estimar o tempo real de implementação de cada tarefa.

Portanto, há uma grande diferença entre quando uma tarefa foi estimada para ser feita e quando ela realmente o foi.

As seguintes tarefas são propostas para a segunda parte do projeto.

- Usuários autenticados alterarem informações de um aeroporto;
- Cálculo da pista ativa a partir da direção do vento;
- Cálculo do perfil de descida;
- Consulta de dados históricos do METAR;
- Obtenção de estatísticas históricas para um aeroporto.

4.2

Cronograma do Projeto Final II

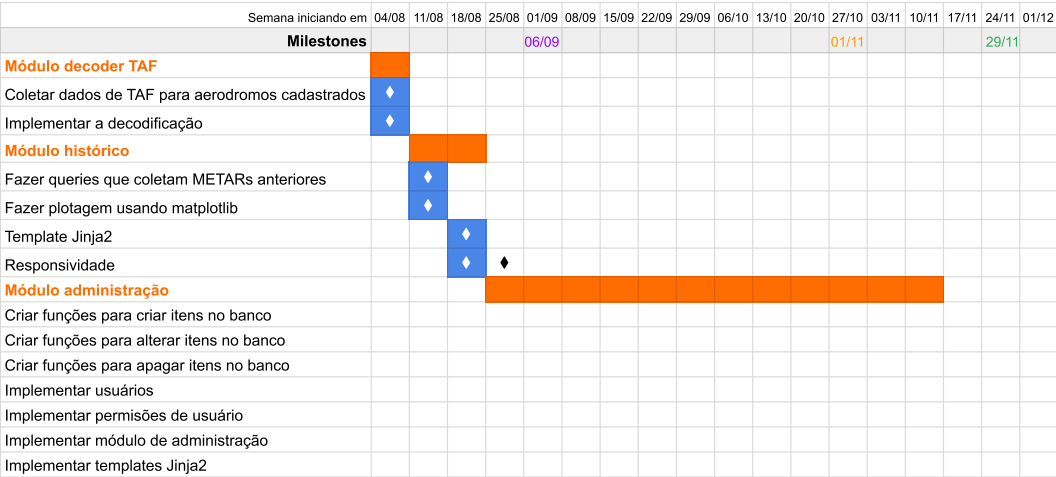


Figura 4.2: Cronograma 2

Tabela 4.2: Milestones

Data	Milestone
06/09	Entrega do formulário de Projeto Final II
01/11	Entrega da proposta de Projeto Final II
29/11	Entrega do relatório de Projeto Final II

5

Princípios norteadores

Para este projeto segui os seguintes princípios por ordem de prioridade começando pela maior prioridade.

5.1

Correção da Informação Apresentada

Não devem haver erros na explicação do METAR, TAF e nas informações do aeródromo. Claro que não é possível aderir a este princípio em 100% dos casos, pois uma informação de aeroporto pode mudar. No entanto, comparo constantemente minhas informações com o AISWeb e, em caso de alterações, é possível fazê-las facilmente pela área restrita do site.

5.2

Rapidez no Carregamento das Páginas

Nenhum arquivo externo como .js, .css, .ttf, etc., é coletado externamente. Os arquivos estáticos estão no servidor. A geração das páginas inclusive dos gráficos é feita no lado do servidor. O que precisa rodar no lado do cliente, como pesquisas e tooltips, é implementado com JavaScript puro.

Os gráficos com informações históricas são construídos de forma assíncrona, e os dados de METAR e TAF são coletados da API do Aviation Weather e inseridos no banco de dados também de forma assíncrona. Assim, quando o usuário carrega a página, essas informações já estão prontas para o envio.

5.3

Facilidade de Uso do Sistema

A diagramação das páginas é feita considerando que o usuário pode não ter um conhecimento avançado em aviação, mas deseja acessar todas as informações de um aeródromo. Essas informações estão divididas em três páginas para cada aeródromo, para que a visualização não fique sobrecarregada.

Por exemplo, para o aeroporto do Galeão no Rio de Janeiro, temos as seguintes páginas:

- <https://aero.a4barros.com/info/SBGL>: Informações de pista, frequências e METAR explicado
- <https://aero.a4barros.com/taf/SBGL>: TAF explicado
- <https://aero.a4barros.com/info/SBGL/history>: Gráficos com informações históricas

6

Modelo de Dados

Nesta seção, explica-se o modelo de dados desenvolvido para o projeto, abordando sua estrutura lógica e os componentes principais. O objetivo deste modelo é garantir flexibilidade para futuras alterações e manutenções, proporcionando uma base sólida e expansível para o aero.

6.1

Modelo Lógico

O modelo de dados foi feito de forma que futuras alterações possam ser absorvidas 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, a sigla PK denota *chave primária* e FK *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.

Note que as tabelas "CommunicationType", "ILSCategory" e "PavementType" poderiam ser substituídas por colunas enum nas tabelas "Communication", "ILS" e "Runway", porém a manutenção seria difícil [5], pois teríamos que alterar a estrutura das tabelas (possivelmente 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.

6.2

Tabelas

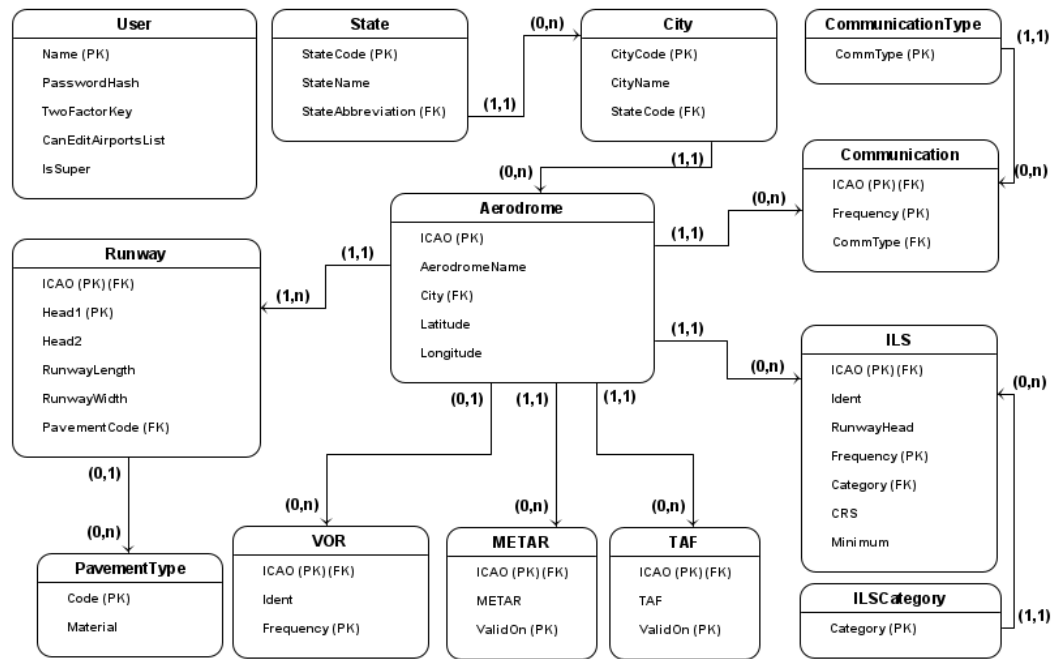


Figura 6.1: Diagrama E/R

Tabela 6.1: City

Nome	Descrição	Tipo
City (PK)	Código do IBGE [6] da cidade	INTEGER
CityName	Nome da cidade escrito em Português com a primeira letra maiúscula	VARCHAR (50)

Tabela 6.2: 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)
Continua na próxima página		

Tabela 6.2 – Continuação da página anterior

Nome	Descrição	Tipo
City (FK)	Chave estrangeira para o código da cidade	INTEGER
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)
Longitude	A longitude do aeroporto, seguindo o mesmo formato da latitude.	DECIMAL(9, 6)

Tabela 6.3: METAR

Nome	Descrição	Tipo
ICAO (PK) (FK)	Chave estrangeira para qual aeródromo este METAR se refere	VARCHAR(4)
METAR	O METAR em si	VARCHAR(100)
ValidOn (PK)	Timestamp com o momento que este METAR é válido. É construído a partir do item do metar "ddhhmmZ" em que "dd" é o dia e "hhmm" é a hora zulu (UTC). O mês e ano são o mês e ano atuais do sistema.	DATETIME

Tabela 6.4: TAF

Nome	Descrição	Tipo
ICAO (PK) (FK)	Chave estrangeira para qual aeródromo este TAF se refere	VARCHAR(4)
Continua na próxima página		

Tabela 6.4 – Continuação da página anterior

Nome	Descrição	Tipo
TAF	O TAF em si	VARCHAR(100)
ValidOn (PK)	Timestamp com o momento que este TAF é válido. É construído a partir do item do metar "ddhhmmZ" em que "dd" é o dia e "hhmm" é a hora zulu (UTC). O mês e ano são o mês e ano atuais do sistema.	DATETIME

Tabela 6.5: 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 6.6: 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 a 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:

6.2.1

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.

6.2.2

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).

6.2.3

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 6.7: 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 6.8: 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 possuem três dígitos 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 haver frequências repetidas.	INTEGER
Continua na próxima página		

Tabela 6.8 – 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 6.9: 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 6.10: 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 6.10 – Continuação da página anterior

Nome	Descrição	Tipo
Ident	Identificação de três letras maiúsculas 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 caindo em desuso.

Tabela 6.11: 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)

6.3

Adicionado no Projeto II

A tabela a seguir foi adicionada na segunda parte do Projeto. Ela serve para a autenticação de usuário com senha e TOTP bem com gerenciar as permissões de cada usuário.

Tabela 6.12: User

Nome	Descrição	Tipo
Name (PK)	Nome do usuário. Usado no login.	VARCHAR(30)
PasswordHash	Hash com salt da senha do usuário. A biblioteca bcrypt é usada para criação do hash e autenticação.	VARCHAR(60)
Continua na próxima página		

Tabela 6.12 – Continuação da página anterior

Nome	Descrição	Tipo
TwoFactorKey	Chave privada para geração de código temporário de 6 dígitos comparado com o código digitado pelo usuário no momento do login. Pode ser nulo, caso não tenha sido cadastrada a autenticação de dois fatores, então essa verificação não é feita.	VARCHAR(32)
CanEditAirportsList	Lista separada por vírgulas dos ICAOs dos aeroportos que este usuário tem permissão de alterar. Entenda "alterar" por criar, editar e apagar informações internas de um aeroporto: pistas, frequências de rádios e de navegação.	VARCHAR(32)
IsSuper	Indica se o usuário pode criar e apagar aeroportos. Se verdadeiro, este usuário pode editar qualquer aeroporto, ignorando a lista CanEditAirportsList.	Boolean

7

Arquitetura

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 Gunicorn, todos os outros serviços usam imagens prontas do Docker Hub, o que oferece mais segurança com as atualizações constantes.

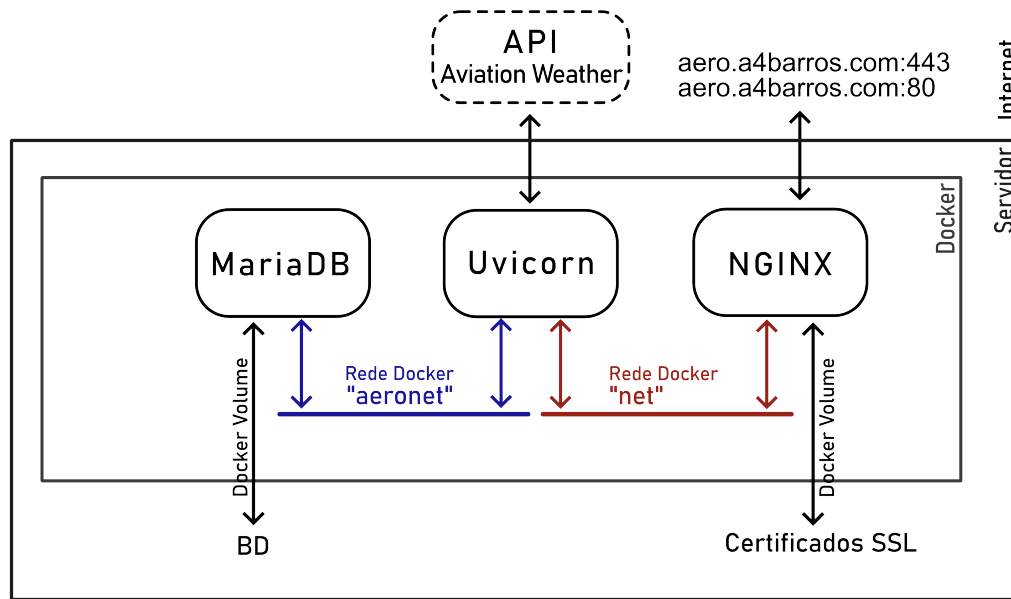


Figura 7.1: Modelo de Arquitetura

7.1

Docker Network

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

7.2

Docker Secrets

Para aumentar a segurança de acesso ao banco, é usada a funcionalidade "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 têm acesso a cada senha. Caso o serviço de banco de dados, por exemplo, tenha acesso a senha `db-password.txt`, será feito um `bind` do arquivo `db-password.txt` no host para o `"/run/secret/db-password.txt"` no `guest`. Tanto os bancos como o servidor Gunicorn usam este método para terem acesso às senhas dos bancos.

7.3 Serviços

7.3.1 MariaDB

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

7.4 FastAPI

O site foi construído com a framework FastAPI tanto para desenvolvimento quanto para produção, utilizando o servidor embutido do FastAPI, o Uvicorn.

Diferentemente do Flask, em que era necessário usar um servidor externo (o Gunicorn no caso deste projeto), o servidor do FastAPI (Uvicorn [7]), com a configuração padrão, é suficiente para produção. Usando o comando `uvicorn server:app --host 0.0.0.0 --port 5000`, o servidor é iniciado em modo de produção.

7.4.1 Proxy NGINX

O NGINX faz o HTTPS funcionar, dá suporte ao HTTP/2 e ao cabeçalho HTTP keep-alive. Quando utilizava o Gunicorn, ele só tinha suporte ao primeiro, e a documentação do Gunicorn não recomendava que ele estivesse diretamente ligado à Internet [8]. O Uvicorn, no momento em que este texto foi escrito, não suporta HTTP/2, mas suporta o keep-alive. No entanto, como tenho outros projetos na mesma máquina, utilizo subdomínios. Todos os subdomínios resolvem para a mesma IP/máquina via "A RECORD", mas na configuração do NGINX, o bloco de servidor com o hostname `aero.a4barros.com` é redirecionado para o endereço interno `"http://aero:5000"`.

7.5 Produção

O site se encontra em produção no endereço <https://aero.a4barros.com>. Ele está hospedado em uma VPS da Oracle Cloud Infrastructure com as seguintes características de hardware:

- **CPU:** AMD EPYC 7551 (2 cores) @ 1.996GHz
- **RAM:** 1GB
- **Armazenamento:** 25GB
- **SO:** Ubuntu 22.04.4 LTS

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
157693	a4	20	0	8704	4352	3328	R	1.3	0.4	0:00.88	htop
1	root	20	0	163M	11372	6380	S	0.0	1.2	1:48.53	/sbin/init
346	root	19	-1	246M	25632	24480	S	0.0	2.6	0:44.56	/lib/systemd/systemd-journald
379	root	RT	0	282M	27392	8960	S	0.0	2.8	3:00.79	/sbin/multipathd -d -s
383	root	20	0	282M	27392	8960	S	0.0	2.8	0:00.00	/sbin/multipathd -d -s
384	root	RT	0	282M	27392	8960	S	0.0	2.8	0:00.00	/sbin/multipathd -d -s
385	root	RT	0	282M	27392	8960	S	0.0	2.8	0:00.00	/sbin/multipathd -d -s
386	root	RT	0	282M	27392	8960	S	0.0	2.8	0:02.73	/sbin/multipathd -d -s
387	root	RT	0	282M	27392	8960	S	0.0	2.8	2:07.28	/sbin/multipathd -d -s
388	root	RT	0	282M	27392	8960	S	0.0	2.8	0:00.00	/sbin/multipathd -d -s
399	root	20	0	26108	6012	4092	S	0.0	0.6	0:07.61	/lib/systemd/systemd-udev
557	systemd-t	20	0	89364	5376	4608	S	0.0	0.6	0:03.79	/lib/systemd/systemd-timesyncd
598	systemd-t	20	0	89364	5376	4608	S	0.0	0.6	0:00.00	/lib/systemd/systemd-timesyncd
616	systemd-n	20	0	16384	5504	4480	S	0.0	0.6	0:16.92	/lib/systemd/systemd-networkd
618	systemd-r	20	0	25672	9440	5120	S	0.0	1.0	0:49.06	/lib/systemd/systemd-resolved
670	messagebu	20	0	9052	4352	3328	S	0.0	0.4	0:32.89	@dbus-daemon --system --address=systemd: --no
671	root	20	0	12924	2360	1664	S	0.0	0.2	1:16.88	/sbin/iscsid
673	root	10	-10	13428	13128	11392	S	0.0	1.4	0:00.00	/sbin/iscsid
677	root	20	0	82700	3200	2944	S	0.0	0.3	1:13.80	/usr/sbin/irqbalance --foreground
678	root	20	0	33088	12800	3712	S	0.0	1.3	0:05.84	/usr/bin/python3 /usr/bin/networkd-dispatcher

Figura 7.2: Uso do sistema em baixa demanda

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

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED	STATUS	PORTS
885061e72965	a4-aero	"python3 -u -m gunic..."	a4-aero-1	3 hours ago	Up About an hour	0.0.0.0:5000->5000/tcp, ::5000-
>5000/tcp						
8bc8d020f125	nginx	"/docker-entrypoint.s..."	a4-a4-1	3 hours ago	Up About an hour	0.0.0.0:80->80/tcp, ::80->80/tc
p, 0.0.0.0:443->443/tcp, ::443->443/tcp						
1e7763d35d44	a4-axia	"unicorn -c gunicor..."	a4-axia-1	3 hours ago	Up About an hour	5002/tcp
4434a160cee2	a4-todo	"unicorn -c gunicor..."	a4-todo-1	3 hours ago	Up About an hour	5001/tcp
c4c10e0c261	redis:alpine	"docker-entrypoint.s..."	a4-redis-1	3 hours ago	Up About an hour	6379/tcp
5ec6396a65b9	mariadb	"docker-entrypoint.s..."	a4-aero-db-1	3 hours ago	Up About an hour	0.0.0.0:3306->3306/tcp, ::3306-
>3306/tcp						
0e52641e9fda	a4-conv	"/docker-entrypoint.s..."	a4-conv-1	3 hours ago	Up About an hour	80/tcp, 4001/tcp

Figura 7.3: Containers Docker em execução

7.6

Operações Síncronas e Assíncronas

Existem operações que ocorrem quando o usuário acessa uma página e outras que ocorrem de tempos em tempos independentemente dos acessos (assíncronas). Fiz esta separação para garantir que as APIs externas (METAR e TAF) sejam acessadas apenas quando necessário sem que uma quantidade grande de acessos sobrecarregue as APIs.

O desenho dos plots com dados histórico é uma operação que demora aproximadamente 3 segundos por aeródromo, então faço esta operação em background e deixo o arquivo .svg já pronto em uma pasta específica.

A seguir esta descrito quando ocorre estas operações assíncronas. Note que elas são feita com mais frequencia que o necessário para ter certeza que uma informação não ficará muito tempo desatualizada caso ocorra algum atraso para atualização do METAR/TAF.

Tabela 7.1: Operações assíncronas

Nome da função	Quando ocorre	Descrição
update_metars	Nos minutos 0, 15, 30 e 45	Normalmente um novo METAR é disponibilizado em cada hora, mas é possível ter um METAR a cada meia hora ou até menos. Para cada ICAO presente na tabela Aerodrome cria um novo registro na tabela METAR com o novo METAR obtido, caso este seja mais novo que o último presente no banco (é verificada a coluna updatedOn).
Continua na próxima página		

Tabela 7.1 – Continuação da página anterior

Nome da função	Quando ocorre	Descrição
update_images	Nos minutos 0, 15, 30 e 45	Criação dos plots. Obviamente precisa ter a mesma frequência da atualização de METAR.
update_tafs	No minuto 5	O mesmo só que para os TAFs. Os TAFs são atualizados com menos frequência que os METARs, Normalmente nos horários 00:00, 06:00, 12:00 e 18:00 UTC.

7.7

Diagrama de sequência

1. O usuário realiza uma requisição para a rota raiz, `"/info/{icao}"`, `"/taf/{icao}"` ou `"/history/{icao}"`
2. Um servidor NGINX funcionando como proxy realiza o limite de requisições por segundo e bloqueia user-agents que aparentem serem robôs. Caso a requisição passe pelo filtro, é realizado um proxy-pass para o servidor Gunicorn.
3. Para a rota raiz, é feito um SELECT no banco para pegar informações de todos os aeroportos. Na rota `"/info/{icao}"`, é feito um SELECT-WHERE na tabela METAR, na rota `"/taf/{icao}"` é feito o mesmo, mas na tabela TAF. Para a rota `"/history/{icao}"` não há acesso ao banco. A ORM é usada para isto, portanto os comandos SQL não aparecem diretamente no código.
4. O banco de dados responde à requisição.
5. O servidor envia as informações necessárias ao Jinja2 para a geração da página.

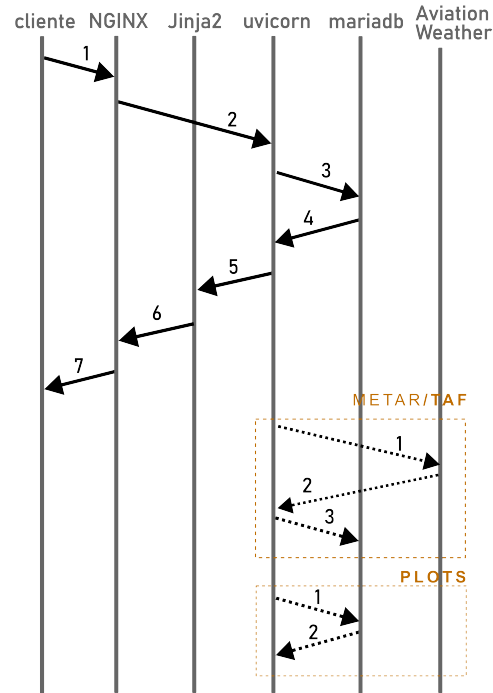


Figura 7.4: Diagrama de sequência

6. A página HTML é gerada.
7. O usuário recebe esta página.

Para atualizar os METARs e TAFs os seguintes passos acontecem.

1. É feita uma requisição para a API do Aviation Weather pedindo o METAR para os aeroportos cadastrados esta API permite informar uma lista de ICAOs sepados por vírgula. O mesmo é feito com outra rota desta API que responde com os TAFs.
2. A API responde.
3. Os METARs e TAFs atualizados são gravados no banco.

Para gerar as plotagens com informações históricas os seguintes passos acontecem.

1. É feito um query para o banco pedir os 10 últimos METARs de cada aeródromo.
2. O banco responde. A biblioteca matplotlib é usada para criar os plots que são salvo como svgs no path /static/plots.

8

Decodificação do METAR

O METAR é um protocolo de transmissão de dados meteorológicos de um aeroporto ou aeródromo. Não se trata de uma previsão do tempo, mas sim de uma visualização atual. O METAR é formado por itens separados por espaço. Cada item corresponde a uma unidade mínima de informação meteorológica. Com os dados de sensores instalados no aeródromo [9], a cada hora é publicado um novo METAR que é válido para aquela hora. Em casos excepcionais, quando as condições de tempo estiverem mudando repentinamente, um METAR pode ser atualizado a cada meia hora [10]. Nas próximas seções será apresentado um exemplo de METAR e sua decodificação bem como uma descrição e complexidade temporal do algoritmo.

8.1

Exemplo

O METAR no aeroporto de Fortaleza [11], no dia 17 de abril de 2024 às 10:54 foi 171300Z 15010KT 9999 BKN019 SCT025 FEW030TCU BKN100 30/25 Q1011.

"SBFZ" se refere ao código ICAO (International Civil Aviation Organization) do aeroporto, não confundir com o código IATA (International Air Transport Association) que é formado por três letras. O aeroporto Pinto Martins possui o código IATA FOR, o Santos Dumont SDU e o Galeão GIG. O público geral parece conhecer mais este código, mas na aviação costuma-se usar mais o código ICAO, pois *todos* os aeródromos possuem um, enquanto o IATA só é presente em aeroportos onde há processamento de bagagem [12] [13].

O ICAO é formado por quatro letras em que a primeira é o prefixo da região. A América do Sul possui o prefixo "S", o Brasil possui o prefixo "SB", por isso que os aeroportos de Fortaleza, Santos Dumont e Galeão possuem os códigos SBFZ, SBRJ e SBGL, respectivamente. Países com muitos aeroportos, apenas uma letra, logo as três últimas letras ficam livres, podendo assim ter mais códigos para uso.

171300Z significa que este METAR se refere ao dia 17 às 13 horas e zero minuto zulu. Horário zulu é simplesmente o fuso horário da longitude de zero grau, chamado de hora UTC ou Coordinated Universal Time [14]. Para que não haja confusão com os horários, a aviação internacionalmente usa o horário

UTC. Este METAR será válido até às 13:59, quando será substituído pelo METAR iniciando com "SBFZ 171400Z".

Note que a seguinte expressão regular com três grupos de captura consegue extrair o dia, a hora e o minuto:

`([0-9]{2})([0-9]{2})([0-9]{2})Z`

Com o METAR supracitado, os grupos de captura serão:

- Grupo 1 (dia): 17
- Grupo 2 (hora): 13
- Grupo 3 (minuto): 00

15010KT se refere à velocidade e direção do vento. Os três primeiros algarismos informam a direção, em graus, de onde o vento sopra, e os últimos dois algarismos informam a velocidade do vento em nós (milhas náuticas por hora). Neste caso, o vento vem da direção 150 graus com velocidade de dez nós. Com a expressão abaixo extraímos essas duas informações:

`([0-9]{3})([0-9]{2})KT`

A informação de vento pode também conter a letra G (gust) para rajadas e a letra V em um item separado para o caso de haver variação de direção. Por exemplo, um METAR com os itens 10016G21KT 080V120 informa que há rajadas de até 21 kt e a direção do vento pode variar de 80 a 120 graus. Existem outros aeroportos que podem usar outras unidades para a velocidade do vento, mas no Brasil só é usado nós (kt). Para obter essas informações usamos o regex `([0-9]{3}[0-9]{2}G[0-9]{2})` e `([0-9]{3}V[0-9]{3})`.

9999 significa visibilidade ilimitada (maior ou igual a 10 km). Se fosse 6000, a visibilidade seria de 6 km. Por ser sempre quatro algarismos, o regex `([0-9]{4})` consegue capturar essa informação.

30/25 Temperatura 30°C e ponto de orvalho 25°C. Caso a temperatura seja negativa, a letra M é adicionada antes do número. M2/M5 significa temperatura -2°C e ponto de orvalho -5°C [15].

Q1012 O altímetro do avião deve ser referenciado para 1012 hectopascal. Também pode ser usada a unidade polegadas de mercúrio (mmHg), mas no Brasil esta não é usada no METAR.

SCT025 Nuvens espalhadas (3/8 a 4/8 do céu com nuvens) em 2500 pés de altitude. 025 se refere ao nível de voo (Flight Level), que é a altitude acima do nível médio do mar com divisão exata por 100.

FEW030TCU Poucas nuvens (1/8 a 2/8 do céu com nuvens) em 3000 pés de altitude. O sufixo TCU significa que há nuvens convectivas significativas [16].

BKN100 Nuvens broken (5/8 a 7/8 do céu com nuvens) em 10000 pés de altitude.

Existe também o tipo OVC (overcast) que se refere a totalmente encoberto.

8.2

Algoritmo

O objetivo do módulo de decoder é dar uma explicação semelhante a esta para qualquer tipo de METAR de aeroportos no Brasil. O módulo usa várias expressões regulares para decodificar uma grande quantidade de informações, porém não é exaustivo; foi dada preferência a fenômenos que podem ocorrer no Brasil [16].

O algoritmo deve separar a string do METAR pelo caractere de espaço. Para cada item separado, cada expressão regular é testada. Caso uma combinação ocorra, os grupos de captura são interpolados em uma string que explica aquele item.

Sendo "\$1"o primeiro grupo de captura e "\$2"o segundo, se o item "27008G16KT"é encontrado pela expressão

$$([0-9]\backslash\{3\}) ([0-9]\backslash\{2\})G([0-9]\backslash\{2\})KT$$

o algoritmo interpola a frase:

Vento \ \$1° com \ \$2 nós e rajadas de até \ \$3 nós

com os grupos de captura do regex supracitado. Então será gerada uma tupla (27008G16KT, Vento 270° com 8 nós e rajadas de até 16 nós). O retorno do algoritmo será uma lista de tuplas que será enviada a ferramenta de templating de página Jinja.

8.3

Complexidade Temporal

Considerando que todas as expressões usadas são simples, isto é, não levam a backtracking, a complexidade temporal para executar a função `re.findall()` do Python é

$$O_{findall}(m + n)$$

m := quantidade de caracteres da expressão regex
 n := quantidade de caracteres da string a ser analisada

Se temos que testar todas as expressões para cada item do METAR, a complexidade será

$$O_{decode}(p * q * (m + n))$$

m := quantidade de caracteres da expressão regex
 n := quantidade de caracteres da string do METAR a ser analisado
 p := quantidade de itens do METAR
 q := quantidade de expressões regex no programa

A maior expressão regex no decoder é

$$([A-Z]\{3\})(\backslash d\{3\})(CB|TCU)*$$

com 26 caracteres. Para efeitos práticos este é um valor muito pequeno então podemos assumir m constante.

Sabemos que o número de expressões é 11, também um valor que pode ser assumindo contante, logo q é igual à 1, portanto.

$$O_{decode}(p * 1 * (1 + n))$$

$$O_{decode}(p * n)$$

Apenas as variáveis " p " e " n " dependem de valores externos.

9

Decodificação do TAF

O TAF (Terminal Aerodrome Forecast) é uma informação meteorológica que fornece previsões para aeroportos, sendo uma ferramenta crucial para a aviação. Diferente do METAR, que relata as condições meteorológicas atuais, o TAF projeta o que se espera para as próximas horas e/ou dias. Ele é composto por diversas seções, cada uma contendo informações específicas sobre as condições previstas para o período. Uma diferença em relação ao METAR é que o TAF se entende por várias linhas. A partir da segunda linha temos os grupos temporais. Tudo que está em uma linha são condições que irão ocorrer dentro de uma faixa de tempo.

Neste capítulo, será apresentado um exemplo de TAF, sua decodificação, e uma análise da complexidade temporal do algoritmo utilizado.

9.1

Exemplo

O TAF emitido para o aeroporto de Tancredo Neves (SBCF) no dia 21 de abril de 2024 às 07:00Z foi:

```
TAF SBCF 210700Z 2112/2212 00000KT 7000 NSC TX31/2119Z TN15/2209Z  
BECMG 2112/2114 11005KT  
PROB30 2117/2121 04005KT  
BECMG 2201/2203 00000KT RMK PGF
```

Vamos decodificar este TAF passo a passo:

Data e Hora de Emissão: 210905Z indica que o TAF foi emitido no dia 21 às 09:05Z.

Período de Validade: 2112/2124 indica que o TAF é válido do dia 21 às 12:00Z até o dia 21 às 24:00Z.

Condições Iniciais: 34005KT 7000 SCT015 descreve as condições previstas para o início do período de validade. 34005KT significa que o vento estará vindo da direção 340° com uma velocidade de 5 nós. 7000 indica visibilidade de 7 km, e SCT015 significa que haverá nuvens dispersas a 1500 pés.

Temperatura Mínima e Máxima: TN21/2112Z TX25/2116Z indica que a temperatura mínima prevista é de 21°C às 12:00Z do dia 21, e a máxima será de 25°C às 16:00Z do dia 21.

A partir de agora começam os grupos separados por linha que comentei na introdução deste capítulo.

Mudança Prevista (BECMG): BECMG 2113/2115 9999 FEW015 indica que entre 13:00Z e 15:00Z do dia 21, a visibilidade aumentará para 10 km ou mais (9999), e haverá poucas nuvens a 1500 pés (FEW015).

Nova Mudança Prevista: BECMG 2116/2118 16010KT CAVOK sugere que entre 16:00Z e 18:00Z do dia 21, o vento mudará para 160° com uma velocidade de 10 nós, e as condições se tornarão "CAVOK"(Ceiling and Visibility OK), indicando que a visibilidade e o teto de nuvens estão dentro dos limites favoráveis para operações de voo.

Observação: RMK PHD adiciona uma observação, é algo específico dos TAFs brasileiros. O trigramma PHD representa a indentificação do previsor que gerou este TAF [17].

9.2

Algoritmo

O algoritmo para lidar com um TAF é semelhante ao utilizado para o METAR, porém adaptado para as especificidades das previsões temporais. O TAF é segmentado em diversas partes, cada uma sendo processada individualmente por expressões regulares para extrair as informações relevantes.

Para cada item do TAF, o algoritmo tenta associar a string correspondente a uma descrição detalhada utilizando expressões regulares pré-definidas. Por exemplo, a expressão `4/44/4` é usada para identificar os períodos de tempo, enquanto a expressão `(3)(2)KT` é utilizada para extrair a direção e velocidade do vento.

Assim como no METAR, as informações são organizadas em tuplas que contêm a string original e sua decodificação correspondente. Essas tuplas são então enviadas para um sistema de templating Jinja2, para a geração da página HTML.

9.3

Complexidade Temporal

A complexidade temporal da decodificação de um TAF segue um padrão similar ao do METAR. A função `re.findall()` é utilizada para identificar e extrair informações, e a complexidade temporal desta operação depende do número de caracteres da expressão regular (m) e da string do TAF a ser analisada (n).

Considerando que as expressões regulares utilizadas no TAF também são simples, a complexidade temporal de decodificação de um TAF pode ser representada por:

$$O_{decode_{TAF}}(p * n)$$

Onde:

p := Quantidade de itens do TAF.

n := Quantidade de caracteres da string do TAF.

Como no caso do METAR, as variáveis " p " e " n " são as que influenciam diretamente a complexidade do algoritmo.

10

Plotagem do METAR Histórico

Para o aeronauta, é fundamental acessar informações meteorológicas históricas, pois essas informações auxiliam na previsão de condições futuras. A meteorologia tende a seguir padrões; por exemplo, uma queda na pressão atmosférica geralmente indica que as temperaturas podem diminuir posteriormente. Assim, visualizar essas informações em gráficos facilita a análise e a tomada de decisões.

Os dados meteorológicos (METARs) coletados de um aeroporto são armazenados em uma tabela dedicada, possibilitando o armazenamento de múltiplos registros para um único aeródromo. A partir dessa tabela, é realizada uma consulta (*query*) que obtém os 10 últimos registros de METAR de cada aeródromo. Desse conjunto de dados, são extraídas seis informações: temperatura, ponto de orvalho, velocidade do vento, direção do vento, ajuste altímetro e visibilidade.

Os resultados da consulta são transformados em uma lista de dicionários, onde cada dicionário contém essas seis informações.

```
metar_data = [  
    {  
        "temperature": 20 ,  
        "dew_point": 15,  
        "wind_speed": 12,  
        "wind_direction": 120,  
        "qnh": 1030,  
        "visibility": 9999,  
    },  
    ...  
]
```

Essa lista é então passada para a biblioteca `matplotlib`, que gera três gráficos em um estilo personalizado para combinar com o tema escuro da página. Cada gráfico combina duas informações. Os gráficos são salvos como arquivos SVG na pasta de arquivos estáticos.

As imagens de plot seguem um padrão, o código ICAO do aeroporto e os dados específicos são utilizados na geração do nome do arquivo SVG ("icao-dado1-dado2.svg"). Ao carregar a página, as imagens são exibidas diretamente por meio de tags ``, que apontam para os gráficos salvos na pasta estática.

Como na consulta do METAR e TAF, estas tarefas são realizadas assincronamente nos minutos 0, 15, 30 e 45 (vide capítulo de arquitetura). Desta forma quando o usuário carrega a página as plotagens já encontram-se prontas.

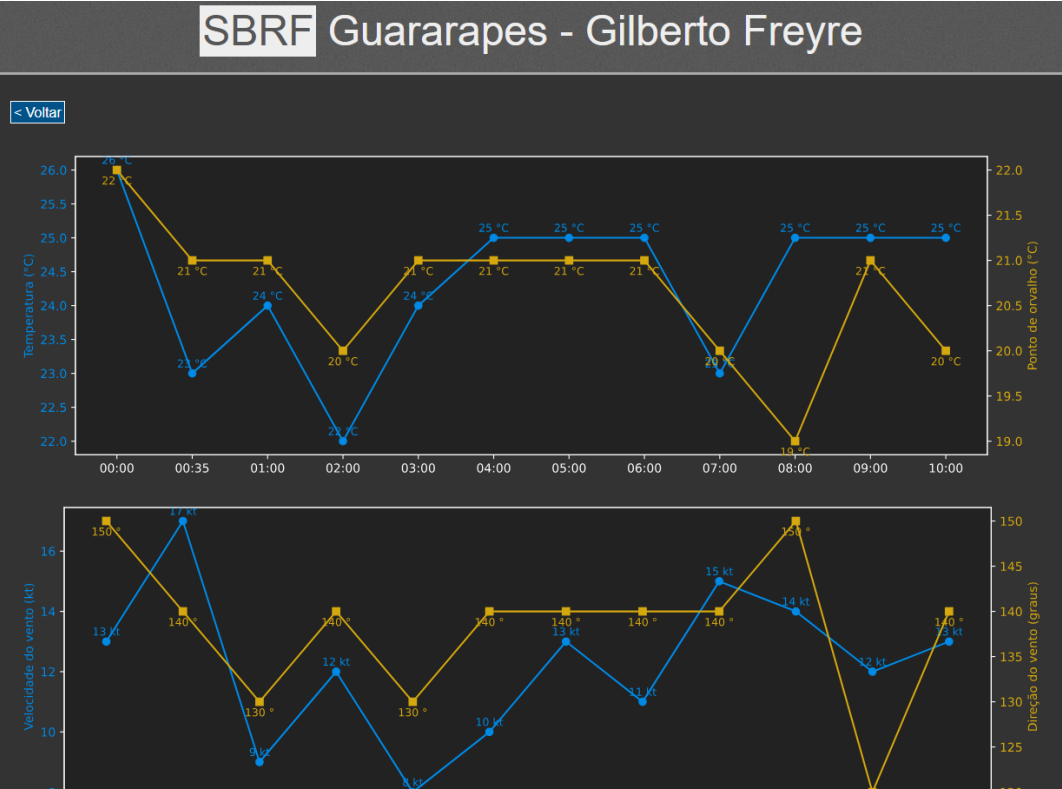


Figura 10.1: Gráficos do aeroporto Aeroporto Internacional do Recife do dia 28 de agosto de 2024

11

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.

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), foi criado o filtro "frequency3" para exibir o número corretamente. Um "filtro" no Jinja é apenas uma função que recebe e retorna uma string. O filtro é "chamado" usando a sintaxe "`{{variavel | funcao}}`". Para os que tem experiência com Linux é parecido com a ideia do operador "pipe".

Código 1: Template de comunicação com o Jinja

```
1 <div class="line">
2     {% for comm in info.communication %}
3     <div class="box-rounded box-outlined">
4         <div class="box-name">{{comm.CommType}}</div>
5         <div class="box-value"><b>{{comm.Frequency | frequency3 }}</b>
6             {% if isAdmin %}
7             <a href="/area/restrita/{{icao}}/communication/{{ comm.
8                 Frequency }}/edit" class="button-admin">Editar</a>
9             {% endif %}
```

```
9         </div>
10     </div>
11     {% endfor %}
12 </div>
```

Note que caso a variável "isAdmin" seja definida, um botão de alterar a frequência aparece. Este e outros botões de adição e edição são mostrados quando o login é feito para que o administrador consiga editar um aeródromo.

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

11.1

Minificação

Removendo os espaços e caracteres de nova linha é possível diminuir o tamanho dos arquivos enviados para o usuário. Após este processo, cada arquivo html, js etc fica com apenas uma linha, os menos linhas no caso dos css, obviamente não é fácil para um humano entender, mas o navegador consegue fazer o parsing sem problemas. Isto é chamado de *minification* ou minificação. Com os arquivos com menor tamanho, a transferência servidor para cliente é terminada mais rápido, logo a experiência para o usuário torna-se mais agradável, já que as páginas carregam mais rápido.

Para a tabela abaixo cada tamanho em kB e cada tempo em ms se refere a média simples dos valores encontrados na aba "rede" das ferramentas de desenvolvedor do navegador Firefox em cinco carregamentos da página. A opção de desabilitar cache foi usada.

Legenda para a tabela abaixo

- **A:** Tamanho do arquivo sem minification
- **B:** Tempo de carregamento deste arquivo
- **C:** Tamanho do arquivo com minification
- **D:** Tempo de carregamento deste arquivo

O "Tamanho" refere-se a quantidade de bytes transferidos (com compactação gzip) e não ao tamanho do arquivo após a compactação já que isto é feito no lado do usuário pelo navegador.

Tabela 11.1: Com e Sem minification

Arquivo	A	B	C	D
SBGR	5,88 kB	130 ms	4,93 kB	57 ms
style.css	3,45 kB	68 ms	2,49 kB	33 ms
tooltip.css	960 B	67 ms	831 B	24 ms
rwyt.css	590 B	36 ms	518 B	38 ms
Total	11.35 kB	363 ms	9.21 kB	196 ms

Fazendo

$$x\% = \frac{x_{\text{initial}} - x_{\text{final}}}{x_{\text{initial}}} \times 100\%$$

É possível ver uma economia de 18,9% na quantidade de informações enviadas. E um tempo de resposta 46,0% menor.

12

Rotas do back-end

12.1

Rota raiz

Página inicial, apresenta a lista de aeródromos para que o usuário escolha um. Internamente, por meio do ORM, 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'),
    ...
]
```

12.2

Rota: `/info/{ICAO}`

Retorna informações de um aeródromo com o ICAO especificado na URL. São exibidos, além da explicação do METAR atual:

- Pista
 - Cabeceiras
 - Comprimento
 - Largura.
- Frequências do aeródromo (nem todos os items a seguir poderão estar disponíveis)
 - Torre
 - Solo
 - Operações
 - Rampa
 - Tráfego
 - ATIS

- Frequências de navegação (nem todos os itens a seguir poderão estar disponíveis)
 - ILS
 - * Qual pista este ILS se refere
 - * Frequência
 - * Direção final de aproximação
 - * Identificador
 - VOR
 - * Frequência
 - * Identificador

12.3

Rota: /history/{ICAO}

Para um aeródromo retorna informações dos dez últimos METARs em três gráficos com o eixo horizontal sendo o tempo.

- Gráfico 1
 - Temperature (graus Célsius)
 - Ponto de orvalho (graus Célsius)
- Gráfico 2
 - Velocidade do vento (milhas náuticas por hora)
 - Direção do vento (graus)
- Gráfico 3
 - Ajuste altímetro (hectopascal)
 - Visibilidade (metro)

12.4

Rota: /taf/{ICAO}

Retorna o próximo TAF válido para este aeródromo com a explicação de cada item.

13

Conclusão e Próximos Passos

O desenvolvimento deste projeto busca juntar as funcionalidades comumente usadas na simulação de voo em uma plataforma acessível e amigável, buscando promover um maior nível de realismo na simulação de voo, aspecto desejado por quem leva a simulação de voo "a sério".

A arquitetura implementada, utilizando Docker e Docker Compose, garante uma implementação modular, segura e, com o uso do Git, de fácil deploy caso seja necessário trocar o ISP no futuro. O uso de um banco de dados como o MariaDB e da funcionalidade Docker Secrets para armazenar as senhas, proporciona um ambiente robusto para o projeto. O uso de uma VPS modesta para hospedagem do projeto em produção demonstra a viabilidade do sistema em ambientes com recursos limitados.

Para o próximo semestre está planejado implementar o módulo que informa a pista ativa. É mais seguro para o voo se a decolagem for realizada a partir da cabeceira em que o vento está soprando contra o sentido da movimentação do avião. Por isso, em um momento os procedimentos estão sendo realizados em um lado da pista e em outro momento pelo outro lado.

Nem sempre a direção do vento estará paralela com a pista, mas, mesmo assim, tenta-se decolar e pousar no sentido que a componente paralela ao sentido do movimento da aeronave fique com sentido contrário.

A pista ativa normalmente é informada pela torre de controle, mas é interessante o piloto se antecipar.

Outro módulo que será implementado é o cálculo do perfil de descida. Para que o avião chegue no procedimento de aproximação com a altitude expressa na carta de aproximação, é necessário calcular uma razão de descida em pés por minuto, a partir da altitude atual, da altitude que se quer chegar e da velocidade atual.

É interessante poder adicionar e editar aeródromos sem precisar estar logado na máquina de produção via SSH. Será feito uma seção do site protegida por nome de usuário e senha que disponibilizará uma interface para estas mudanças.

14

Apêndice A: Códigos Relevantes

Código 2: Código da ORM usando a biblioteca SQLAlchemy

```
1 from sqlalchemy import PrimaryKeyConstraint, create_engine, \
2     Column, Integer, String, ForeignKey, UniqueConstraint, \
3     DECIMAL, DateTime, Boolean, Text
4 from sqlalchemy.orm import declarative_base, relationship,
5     sessionmaker, Session
6
7 Base = declarative_base()
8
9 class State(Base):
10     __tablename__ = 'State'
11
12     StateCode = Column(Integer, primary_key=True)
13     StateName = Column(String(50), nullable=False)
14     StateAbbreviation = Column(String(2), nullable=False)
15
16 class City(Base):
17     __tablename__ = 'City'
18
19     CityCode = Column(Integer, primary_key=True)
20     CityName = Column(String(50), nullable=False)
21     StateCode = Column(Integer, ForeignKey('State.StateCode'),
22         nullable=False)
23
24 class Aerodrome(Base):
25     __tablename__ = 'Aerodrome'
26
27     ICAO = Column(String(4), primary_key=True)
28     AerodromeName = Column(String(50), nullable=False)
29     CityCode = Column(Integer, ForeignKey('City.CityCode'), nullable
30         =False)
31     Latitude = Column(DECIMAL(9, 6), nullable=False)
32     Longitude = Column(DECIMAL(9, 6), nullable=False)
33     __table_args__ = (
34         UniqueConstraint('AerodromeName'),
35     )
36
37     runways = relationship("Runway", backref="aerodrome")
38     ils = relationship("ILS", backref="aerodrome")
39     vor = relationship("VOR", backref="aerodrome")
40     communication = relationship("Communication", backref="aerodrome")
41
42 class METAR(Base):
```

```

41     __tablename__ = 'METAR'
42     ICAO = Column(String(4), ForeignKey('Aerodrome.ICAO'),
         primary_key=True)
43     ValidOn = Column(DateTime(timezone=True), nullable=True,
         primary_key=True)
44     METAR = Column(String(200), nullable=True)
45
46 class TAF(Base):
47     __tablename__ = 'TAF'
48     ICAO = Column(String(4), ForeignKey('Aerodrome.ICAO'),
         primary_key=True)
49     ValidOn = Column(DateTime(timezone=True), nullable=True,
         primary_key=True)
50     TAF = Column(String(3000), nullable=True)
51
52 class PavementType(Base):
53     __tablename__ = 'PavementType'
54
55     Code = Column(String(3), primary_key=True)
56     Material = Column(String(20), nullable=False)
57
58 class Runway(Base):
59     __tablename__ = 'Runway'
60
61     ICAO = Column(String(4), ForeignKey('Aerodrome.ICAO'), nullable=
        False)
62     Head1 = Column(String(3), nullable=False)
63     Head2 = Column(String(3), nullable=False)
64     RunwayLength = Column(Integer, nullable=False)
65     RunwayWidth = Column(Integer)
66     PavementCode = Column(String(3), ForeignKey('PavementType.Code')
        )
67
68     __table_args__ = (
69         PrimaryKeyConstraint('ICAO', 'Head1'),
70         UniqueConstraint('ICAO', 'Head1', 'Head2'),
71     )
72
73     pavement_type = relationship("PavementType")
74
75
76 class CommunicationType(Base):
77     __tablename__ = 'CommunicationType'
78
79     CommType = Column(String(20), primary_key=True)
80
81 class Communication(Base):
82     __tablename__ = 'Communication'
83
84     ICAO = Column(String(4), ForeignKey('Aerodrome.ICAO'), nullable=
        False)

```

```

85     Frequency = Column(Integer, nullable=False)
86     CommType = Column(String(20), ForeignKey('CommunicationType.
           CommType'), nullable=False)
87
88     __table_args__ = (
89         PrimaryKeyConstraint('ICAO', 'Frequency'),
90     )
91
92 class ILSCategory(Base):
93     __tablename__ = 'ILSCategory'
94
95     Category = Column(String(10), primary_key=True)
96
97 class ILS(Base):
98     __tablename__ = 'ILS'
99
100    ICAO = Column(String(4), ForeignKey('Aerodrome.ICAO'), nullable=
        False)
101    Ident = Column(String(3), nullable=False)
102    RunwayHead = Column(String(3), nullable=False)
103    Frequency = Column(Integer, nullable=False)
104    Category = Column(String(10), ForeignKey('ILSCategory.Category'),
        , nullable=False)
105    CRS = Column(Integer, nullable=False)
106    Minimum = Column(Integer)
107
108    __table_args__ = (
109        PrimaryKeyConstraint('ICAO', 'Frequency'),
110    )
111
112 class VOR(Base):
113     __tablename__ = 'VOR'
114
115     ICAO = Column(String(4), ForeignKey('Aerodrome.ICAO'), nullable=
        False)
116     Ident = Column(String(3), nullable=False)
117     Frequency = Column(Integer, nullable=False)
118
119     __table_args__ = (
120         PrimaryKeyConstraint('ICAO', 'Frequency'),
121     )
122
123 class User(Base):
124     __tablename__ = 'User'
125
126     Name = Column(String(30), primary_key=True)
127     PasswordHash = Column(String(60))
128     TwoFactorKey = Column(String(32), nullable=True)
129     CanEditAirportsList = Column(Text, nullable=True)
130     IsSuper = Column(Boolean, default=False)

```

Código 3: a

```

1 import re
2 from datetime import datetime, timedelta
3
4 # Fonte: https://ajuda.decea.mil.br/base-de-conhecimento/como-
   decodificar-o-metar-e-o-speci/
5
6 other_items = {
7     "SH": "Pancada(s) moderada.",
8     "+SH": "Pancada(s) forte.",
9     "-FZ": "Congelante leve.",
10    "FZ": "Congelante moderado.",
11    "+FZ": "Congelante forte.",
12    "-DZ": "Chuveiro leve.",
13    "DZ": "Chuveiro moderado.",
14    "+DZ": "Chuveiro forte.",
15    "-RA": "Chuva leve.",
16    "RA": "Chuva moderada.",
17    "+RA": "Chuva forte.",
18    "-SN": "Neve leve.",
19    "SN": "Neve moderada.",
20    "+SN": "Neve forte.",
21    "-SG": "Grãos de neve leve.",
22    "SG": "Grãos de neve moderado.",
23    "+SG": "Grãos de neve forte.",
24    "-PL": "Pelotas de gelo leve.",
25    "PL": "Pelotas de gelo moderado.",
26    "+PL": "Pelotas de gelo forte.",
27    "-GR": "Granizo leve.",
28    "GR": "Granizo moderado.",
29    "+GR": "Granizo forte.",
30    "-GS": "Granizo pequeno e/ou grãos de neve leve.",
31    "GS": "Granizo pequeno e/ou grãos de neve moderado.",
32    "+GS": "Granizo pequeno e/ou grãos de neve forte.",
33    "-BR": "Névoa úmida leve.",
34    "BR": "Névoa úmida moderada.",
35    "+BR": "Névoa úmida densa.",
36    "-FG": "Nevoeiro leve.",
37    "FG": "Nevoeiro moderado.",
38    "+FG": "Nevoeiro denso.",
39    "-FU": "Fumaça leve.",
40    "FU": "Fumaça moderada.",
41    "+FU": "Fumaça densa.",
42    "-VA": "Cinzas vulcânicas leve.",
43    "VA": "Cinzas vulcânicas moderada.",
44    "+VA": "Cinzas vulcânicas densa.",
45    "-DU": "Poeira extensa leve.",
46    "DU": "Poeira extensa moderada.",
47    "+DU": "Poeira extensa densa.",
48    "-SA": "Areia leve.",
49    "SA": "Areia moderada.",

```



```

50     "+SA": "Areia densa.",
51     "-HZ": "Névoa seca leve.",
52     "HZ": "Névoa seca moderada.",
53     "+HZ": "Névoa seca densa.",
54     "-PO": "Poeira/areia em redemoinhos leve.",
55     "PO": "Poeira/areia em redemoinhos moderada.",
56     "+PO": "Poeira/areia em redemoinhos densa.",
57     "-SQ": "Tempestade leve.",
58     "SQ": "Tempestade moderada.",
59     "+SQ": "Tempestade forte.",
60     "-FC": "Nuvem(ns) funil (tornado ou tromba d'água) leve.",
61     "FC": "Nuvem(ns) funil (tornado ou tromba d'água) moderada.",
62     "+FC": "Nuvem(ns) funil (tornado ou tromba d'água) densa.",
63     "-SS": "Tempestade de areia leve.",
64     "SS": "Tempestade de areia moderada.",
65     "+SS": "Tempestade de areia densa.",
66     "-DS": "Tempestade de poeira leve.",
67     "DS": "Tempestade de poeira moderada.",
68     "+DS": "Tempestade de poeira densa.",
69     "-TS": "Trovoada, Raios e Relâmpagos leve.",
70     "TS": "Trovoada, Raios e Relâmpagos moderada.",
71     "+TS": "Trovoada, Raios e Relâmpagos densa.",
72     "RERA": "Fenômenos meteorológicos recentes.",
73     "WS": "Tesoura de vento (windshear)",
74     "NSC": "No Significant Cloud, podem haver algumas nuvens, mas
           nenhuma está abaixo de 5000 pés ou dentro de 10 quilômetros.
           ",
75     "VCSH": "Chuva leve na vizinhança do aeroporto",
76     "TSRA": "Trovoada com chuva.",
77     "+TSRA": "Trovoada com chuva forte.",
78     "VCTS": "Trovoada na vizinhança"
79 }
80
81 def decode_metar(metar: str) -> dict:
82     #metar = "METAR SBSP 290400Z AUTO 19008KT 160V220 9999 FEW006
83             SCT008 BKN010 16/14 Q1025="
84     #metar = "METAR SBSP 290400Z AUTO VRB08KT 160V220 9999 FEW006
85             SCT008 BKN010 16/14 Q1025="
86     #metar = "METAR SBMN 061300Z 31015G27KT 280V350 5000 1500W -RA
87             BKN010 SCT020 FEW025TCU 25/24 Q1014 RERA WS RWY17 W12/H75="
88
89     try:
90         metar = metar.split(" ")
91         day = int(metar[0][0:2])
92         hour = int(metar[0][2:4])
93         minute = int(metar[0][4:6])
94     except ValueError:
95         return [(" ", "METAR indisponível.")]
96
97 ret = []

```

```

96     ts_utc = datetime(day=day, month=datetime.utcnow().month, year=
        datetime.utcnow().year, hour=hour, minute=minute)
97     ts_local = ts_utc - timedelta(hours=3)
98     ret.append((metar[0], f"METAR válido para dia {ts_local.day} as
        {ts_local.hour}:{ts_local.minute:02d} (hora de Brasília)"))
99
100    metar = metar[1:]
101
102    for item in metar:
103        if item == "AUTO":
104            ret.append((item, "Informação obtida automaticamente."))
105
106        elif item == "9999":
107            ret.append((item, "Visibilidade ilimitada"))
108
109        elif (vis := re.findall("^(\d{4})
        ", item)) != [] and vis != "9999": [vis] = vis
        ret.append((item, f"Visibilidade {vis} metros."))
        elif (vis := re.findall("^(\d{4})
        ", item)) != []:
110
111
112
113     , item)) != []:
114         [(vis, sector)] = vis
115         if sector == "N": sector = "norte"
116         elif sector == "S": sector = "sul"
117         elif sector == "W": sector = "oeste"
118         elif sector == "E": sector = "leste"
119         elif sector == "NE": sector = "nordeste"
120         elif sector == "NW": sector = "noroeste"
121         elif sector == "SE": sector = "sudeste"
122         elif sector == "SW": sector = "sudoeste"
123
124         ret.append((item, f"No setor {sector} do aerodromo,
        visibilidade {vis}m."))
125
126     elif (wind := re.findall("(\d{3})(\d{2})KT", item)) != []:
127         [(direction, speed)] = wind
128         ret.append((item, f"Vento proa <b>{direction}</b>° com
        velocidade <b>{speed}</b> nós (kt)."))
129
130     elif (wind := re.findall("VRB(\d{2})KT", item)) != []:
131         [speed] = wind
132         ret.append((item, f"Vento com direção <b>variável</b> e
        velocidade {speed} nós (kt)."))
133
134     elif (wind := re.findall("(\d{3})(\d+)G(\d+)KT", item)) !=
        []:
135         [(direction, speed, gust)] = wind
136         ret.append((item, f"Vento proa {direction}° com
        velocidade {speed} nós (kt) e <b>rajadas</b> de {
        gust} nós."))

```

```

137
138     elif (wind := re.findall("(\\d{3})V(\\d{3})", item)) != []:
139         [(wind1, wind2)] = wind
140         ret.append((item, f"Vento <b>variável</b> de proa {wind1
141             }° até {wind2}°."))
142
143     elif (cloud := re.findall("([A-Z]{3})(\\d{3})(CB|TCU)*", item
144         )) != []:
145         [(cloud_type, cloud_altitude, formation)] = cloud
146         cloud_altitude = int(cloud_altitude) * 100
147
148         if cloud_type == "OVC": cloud_type = "Totalmente
149             encoberto"
150         elif cloud_type == "BKN": cloud_type = "Nuvens broken
151             (5/8 a 7/8 do céu com nuvens)"
152         elif cloud_type == "SCT": cloud_type = "Nuvens
153             espalhadas (3/8 a 4/8 do céu com nuvens)"
154         elif cloud_type == "FEW": cloud_type = "Poucas nuvens
155             (1/8 a 2/8 do céu com nuvens)"
156
157         extra = ""
158         if formation == "CB":
159             extra = "Atenção: nuvens de tempestade."
160         elif formation == "TCU":
161             extra = "Atenção: nuvens de grande extensão vertical
162                 ."
163
164         ret.append((item, f"{cloud_type} em <b>{cloud_altitude
165             }</b> pés de altitude. {extra}"))
166
167     elif (temp := re.findall("(\\d+)/ (\\d+)", item)) != []:
168         [(temperature, dew_point)] = temp
169         ret.append((item, f"Temperatura <b>{temperature}</b>°C e
170             ponto de orvalho <b>{dew_point}</b>°C."))
171
172     elif (qnh := re.findall("Q(\\d{4})", item)) != []:
173         [qnh] = qnh
174         ret.append((item, f"0 altímetro deve ser ajustado para <
175             b>{qnh}</b> hPa ({convert_to_inhg(qnh)} inHg)"))
176
177     elif item == "CAVOK":
178         ret.append((item, "Ceiling and Visibility OK. Sem nuvens
179             e visibilidade OK."))
180
181     elif (runway := re.findall("RWY(\\d{2}[RLC]*)", item)) != []:
182         [runway] = runway
183         ret.append((item, f"Informação anterior se refere a
184             pista {runway}"))
185
186     elif item in other_items:

```

```

176         ret.append((item, other_items[item]))
177
178     else:
179         ret.append((item, ""))
180
181     return ret
182
183 def convert_to_inhg(hpa):
184     return round(float(hpa) / 33.8639, 2)
185
186
187 def get_wind_info(metar: str) -> dict:
188     if (wind := re.findall("(\\d{3})(\\d{2})KT", metar)) != []:
189         [(direction, speed)] = wind
190         return {"direction": int(direction), "speed": int(speed)}
191     else:
192         raise DecodeError("Could not find wind information.")
193
194 def parse_metar(metar_str: str) -> dict:
195     wind_regex = re.compile(r'(\d{3})(\d{2})KT')
196     temp_regex = re.compile(r'M?(\d{2})/(M?\d{2})')
197     qnh_regex = re.compile(r'Q(\d{4})')
198     vis_regex = re.compile(r'(\d{4})')
199
200     wind_match = wind_regex.search(metar_str)
201     temp_match = temp_regex.search(metar_str)
202     qnh_match = qnh_regex.search(metar_str)
203     vis_match = vis_regex.search(metar_str)
204
205     wind_direction = int(wind_match.group(1)) if wind_match else
        None
206     wind_speed = int(wind_match.group(2)) if wind_match else None
207     visibility = int(vis_match.group(1)) if vis_match else None
208
209     if "CAVOK" in metar_str:
210         visibility = 9999
211
212     if temp_match:
213         temp_str = temp_match.group(1)
214         temperature = -int(temp_str[1:]) if temp_str.startswith('M')
            else int(temp_str)
215         dew_str = temp_match.group(2)
216         dew_point = -int(dew_str[1:]) if dew_str.startswith('M')
            else int(dew_str)
217     else:
218         temperature = None
219
220     qnh = int(qnh_match.group(1)) if qnh_match else None
221
222     return {
223         "wind_direction": wind_direction,

```

```

224         "wind_speed": wind_speed,
225         "temperature": temperature,
226         "dew_point": dew_point,
227         "qnh": qnh,
228         "visibility": visibility
229     }
230
231     class DecodeError(Exception):
232         def __init__(self, message):
233             super().__init__(message)
234
235     if __name__ == "__main__":
236         metar = "METAR SBMN 061300Z 31015G27KT 280V350 5000 1500W -RA -
                DU BKN010 SCT020 FEW025TCU 25/24 Q1014 RERA WS RWY17 W12/H75
                ="
237         print(decode_metar(metar))

```

Código 4: a

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  from matplotlib.dates import DateFormatter
4  import matplotlib.dates as mdates
5  from datetime import timedelta
6  import numpy as np
7
8  from DB.Getter import get_all_icao, latest_n_metars_parsed
9
10
11  def plot_metar_data(icao: str,
12                     metar_data: list[dict],
13                     data_name1: str,
14                     label_name1: str,
15                     unit1: str,
16                     label_color1: str,
17                     data_name2: str,
18                     label_name2: str,
19                     unit2: str,
20                     label_color2: str
21                     ):
22
23     with plt.rc_context({'axes.edgecolor': 'white', 'xtick.color': '
        white', 'ytick.color': 'green', 'figure.facecolor': '#333'}):
24         timestamps = [(data["timestamp"] - timedelta(hours=3)).
25                        strftime("%H:%M") for data in metar_data if data]
26         data1 = [data[data_name1] or np.nan for data in metar_data
27                  if data]
28         data2 = [data[data_name2] or np.nan for data in metar_data
29                  if data]
30
31         fig, ax1 = plt.subplots()

```

```

29     fig.set_size_inches(12, 4)
30     ax1.set_facecolor('#222')
31
32     ax1.set_ylabel(label_name1, color=f'{label_color1}')
33     ax1.plot(timestamps, data1, 'o-', color=f'{label_color1}',
34             label=label_name1)
35
36     ax1.tick_params(axis='y', color='white', labelcolor=f'{
37         label_color1}')
38
39     for i, value in enumerate(data1):
40         ax1.annotate(f'{value} {unit1}', xy=(timestamps[i],
41             value), xytext=(0, 5), textcoords='offset points',
42             ha='center', color=label_color1, fontsize
43                 =9)
44
45     ax2 = ax1.twinx()
46     ax2.set_ylabel(label_name2, color=label_color2)
47     ax2.plot(timestamps, data2, 's-', color=label_color2, label=
48         label_name2)
49     ax2.tick_params(axis='y', color='white', labelcolor=
50         label_color2)
51
52     for i, value in enumerate(data2):
53         ax2.annotate(f'{value} {unit2}', xy=(timestamps[i],
54             value), xytext=(0, -12), textcoords='offset points',
55             ha='center', color=label_color2, fontsize
56                 =9)
57
58     # ax1.grid(True)
59
60     fig.tight_layout()
61     filename = f"static/plots/{icao}-{data_name1}-{data_name2}.
62         svg"
63     plt.savefig(filename, dpi=600)
64     plt.close(fig)
65     print(f"Plot saved as {filename}")
66
67 def plot(icao, metar_data):
68     BLUE = "#008ae6"
69     YELLOW = "#d5a810"
70     plot_metar_data(icao, metar_data, "temperature", "Temperatura (
71         °C)", "°C", BLUE, "dew_point", "Ponto de orvalho (°C)", "°C"
72         , YELLOW)
73     plot_metar_data(icao, metar_data, "wind_speed", "Velocidade do
74         vento (kt)", "kt", BLUE, "wind_direction", "Direção do vento
75         (graus)", "°", YELLOW)
76     plot_metar_data(icao, metar_data, "qnh", "Ajuste altímetro (
77         hectopascal)", "hPa", BLUE, "visibility", "Visibilidade (
78         metros)", "m", YELLOW)

```

```
65
66 def update_images():
67     for icao in get_all_icao():
68         latest = latest_n_metars_parsed(icao=icao, n=12)
69         plot(icao=icao, metar_data=latest)
70         print(icao, latest)
```

Referências bibliográficas

- 1 BORT, J. *Here's The Microsoft Surface 2 Tablet Delta Bought 11,000 Pilots Instead Of iPads*. 2014. Disponível em: <<https://www.businessinsider.com/surface-2-tablet-delta-bought-pilots-2014-1>>. Acesso em: 01 maio 2024.
- 2 MOORMAN, R. *EFBs: More Than Paper Replacers*. 2018. Disponível em: <<https://interactive.aviationtoday.com/avionicsmagazine/gca-link-april-2018/efbs-more-than-paper-replacers/>>. Acesso em: 01 maio 2024.
- 3 PRADZ. *Performance Calculation*. 2024. Disponível em: <<http://perfcalc.pradz.de/index.php>>. Acesso em: 25 abril 2024.
- 4 Sebastián Ramírez. *Run a Server Manually*. 2024. Disponível em: <<https://fastapi.tiangolo.com/deployment/manually/>>. Acesso em: 20 agosto 2024.
- 5 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.
- 6 Instituto Brasileiro de Geografia e Estatística. *Códigos dos Municípios*. 2024. Disponível em: <<https://www.ibge.gov.br/explica/codigos-dos-municipios.php>>. Acesso em: 17 junho 2024.
- 7 Sebastián Ramírez. *Uvicorn*. 2024. Disponível em: <<https://www.uvicorn.org/>>. Acesso em: 20 agosto 2024.
- 8 CHESNEAU, B. *Deploying Gunicorn*. 2024. Disponível em: <<https://docs.gunicorn.org/en/latest/deploy.html>>. Acesso em: 24 abril 2024.
- 9 National Weather Service. *Information Reporting*. 2024. Disponível em: <<https://www.weather.gov/asos/InformationReporting.html>>. Acesso em: 17 abril 2024.
- 10 KOCH, S. *METAR / SPECI*. 2024. Disponível em: <<https://sites.google.com/site/invacivil/meteorologia/metar>>. Acesso em: 08 abril 2024.
- 11 National Weather Service. *Aviation Weather Center*. 2024. Disponível em: <<https://aviationweather.gov/api/data/metar?ids=SBFZ>>. Acesso em: 17 abril 2024.
- 12 Airport Codes. *IATA codes*. 2024. Disponível em: <<https://airportcodes.io/en/iata-codes/>>. Acesso em: 16 abril 2024.

- 13 Airport Codes. *ICAO codes*. 2024. Disponível em: <<https://airportcodes.io/en/icao-codes/>>. Acesso em: 16 abril 2024.
- 14 NIST. *What are International Atomic Time (TAI) and Coordinated Universal Time (UTC)?* 2023. Disponível em: <<https://www.nist.gov/pml/time-and-frequency-division/nist-time-frequently-asked-questions-faq>>. Acesso em: 17 abril 2024.
- 15 College of DuPage Weather Lab. *METAR HELP*. 2024. Disponível em: <<https://weather.cod.edu/notes/metar.html>>. Acesso em: 17 abril 2024.
- 16 Departamento de Controle do Espaço Aéreo. *Como decodificar o METAR e o SPECI?* 2024. Disponível em: <<https://ajuda.decea.mil.br/base-de-conhecimento/como-decodificar-o-metar-e-o-speci/>>. Acesso em: 17 abril 2024.
- 17 Comando da Aeronáutica. *CÓDIGO METEOROLÓGICO*. 2024. Disponível em: <<https://www.abul.org.br/biblioteca/43.pdf>>. Acesso em: 21 agosto 2024.