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

Pontifícia Universidade Católica
do Rio de Janeiro

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

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.

82 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

Agradecimentos

Este trabalho final não seria possível sem a ajuda dos meus pais Antenor e Joilma que possibilitaram que eu estudasse na PUC-Rio, meu irmão George com quem sou muito próximo. Meu orientador Adriano Branco que me deu o norte para o trabalho e sugestões de ajustes. Meus amigos de curso que me deram sugestões sobre a implementação: Daniel Guimarães e Miguel Garcia. Meu amigo do tempo de escola Luiz Carlos que tenho a amizade a quase uma década que, com sua experiência profissional, deu sugestões em relação a arquitetura.

Resumo

Leal, Antenor Moreira de Barros; Branco, Adriano Francisco. **Aplicativo web de auxílio à navegação aérea**. Rio de Janeiro, 2024. 82p. 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 (METAR), obtidas a cada hora de uma API externa, como em um texto em linguagem natural para melhor entendimento do usuário. Um usuário com permissão de administrador pode adicionar e editar aeroportos. Módulos adicionais estão disponíveis como o cálculo das componentes de vento em cada pista 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. 82p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

It is an open-source web application designed to assist flight simulator users who do not have access to the tool (Electronic Flight Bag) that an airline pilot would have. When accessing the application, the user is presented with a list of registered airports and, after choosing one, the runway information, airport frequencies (tower, ground, ATIS, etc.), and navigation frequencies (ILS, VOR, etc.) are displayed. Information on the current weather conditions of the aerodrome (wind, visibility, temperature, etc.) is also presented, both in the official format (METAR), obtained every hour from an external API, and in a text in natural language for better understanding by the beginner player. A user with administrator permission can add and edit airports. Additional modules are available, such as calculation of the wind components on each runway and the descent profile.

Keywords

Aviation; Navigation; Application; Algorithm; Web; Internet.

Sumário

Lista de figuras

Lista de tabelas

Lista de Abreviaturas

METAR – *METeorological 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* (Servidor 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 surgimento dos tablets, começaram a ser desenvolvidos programas que substituem partes ou todos estes documentos, é a chamada maleta de voo eletrônica, mais conhecida pela sigla em Inglês EFB (*Electronic Flight Bag*).

Atualmente existem hardwares 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. [2]

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. [3]

Além disso, o EFB possibilita o acesso a uma vasta quantidade de informações adicionais, como manuais de operação da aeronave, cálculo de consumo de combustível, cálculo de velocidades de decolagem e informações de peso e balanceamento, o que contribui para uma tomada de decisão mais informada e segura durante o voo.

2
Sistemas Similares

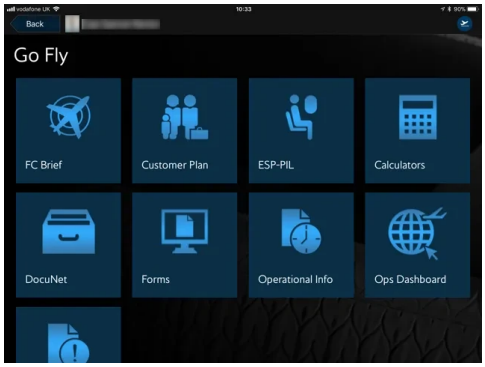
Os EFBs possuem funções variadas como cálculo de combustível, gasto de combustível em voo, velocidades de decolagem, etc. [1] Para aeronaves mais novas, como o Airbus A320neo (New Engine Option), é difícil implementar o cálculo de performance e combustível, pois não é disponibilizado ao público como este cálculo é feito. Ferramentas encontradas na Internet [4], normalmente usam engenharia reversa, e, portanto, podem apresentar resultados diferentes de um cálculo oficial feito no aplicativo de tablet.

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.

Uma das informações importantes para a realização de um voo é o METAR. Trata-se de uma string codificada com as condições meteorológicas atuais de um aeródromo. Todos os pilotos aprendem a ler um METAR.

Contudo, o METAR do aeródromo não se encontra disponível no EFB.

(a) EFB no Flight Simulator 2020 na aeronave A320neo



(b) Interface de um EFB real [1]

(c) EFB real em um Cessna Caravan

Figura 2.1: Exemplos de Electronic Flight Bag (EFB)

É possível usar o computador de bordo da aeronave (FMC) e realizar um "WX Request"para conseguir esta informação. Também, é possível sintonizar na frequência do ATIS, mas isto só funcionará se o avião estiver perto do aeródromo.

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

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

É um site extremamente completo e usado em operações reais, mas para o usuário iniciante seria de valia uma interface mais simples. 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. Como o nome deste sugere, também é disponibilizado o TAF que é parecido com o METAR, mas o TAF é uma previsão das condições.

REDEMET

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

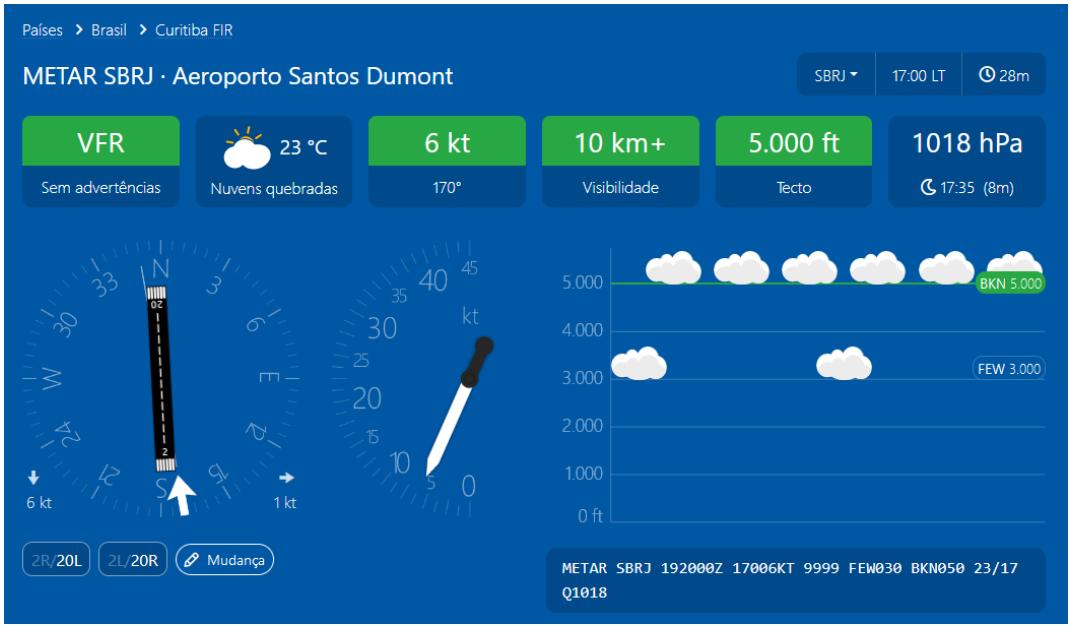


Figura 2.4: Interface gráfica do METAR-TAF

3
A Proposta

A ideia do trabalho é unir as funcionalidades do METAR-TAF com o AISWEB em uma aplicação web que o usuário, mesmo que leigo em aviação, consiga usar sem dificuldades.

No segundo semestre de 2023, o projeto começou a ser desenvolvido inicialmente para uso próprio do autor. O código está disponível em `https://github.com/antenor-z/aero`. Atualmente, o projeto funciona razoavelmente bem, mas a arquitetura foi feita sem muito planejamento, as informações do aeroporto fazem parte do código.

Pelo fato da aviação necessitar ser um ambiente seguro e bastante regulado, considerando que o projeto é apenas um protótipo, prefere-se 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, existe um aviso **"APENAS PARA USO EM SIMULADOR"**.

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 é desejado armazenar estas informações em um banco de dados relacional com uma arquitetura bem planejada. São feitos testes de desempenho simulando uma alta taxa de acesso.

Os aeródromos podem, ao longo do tempo, mudar alguma frequência e outras informações, como o número da pista, que muda a depender da variação do norte magnético, uma ampliação da pista faz a informação de comprimento precisar ser alterada...

Na versão inicial do projeto, o código precisava ser alterado para atualizar estas informações. É implementado um sistema administrativo no site, com uma autenticação por senha e TOTP, para que seja possível mudar qualquer informação no banco mesmo por pessoas que não saibam programar.

Esta parte administrativa, possui controle de permissões. Ao criar uma conta, informa-se quais aeroportos esta conta pode alterar. Também pode ser informado que esta é uma conta "super"que pode alterar todos os aeroportos e também criar e apagar aeroportos.

4
Cronograma

Para ter uma direção do projeto e medição da evolução, 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

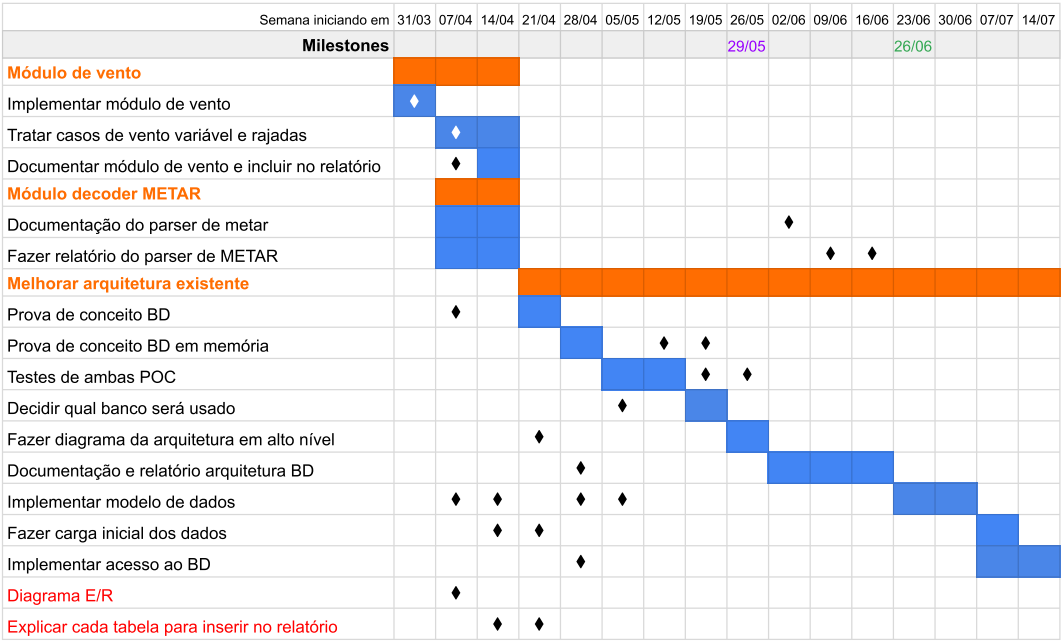


Figura 4.1: Cronograma I

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 o primeiro grande projeto autogerido do autor, teve-se 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 das componentes do vento;
- Cálculo do perfil de descida;
- Geração de gráficos históricos para um aeroporto;
- Visualização das posições dos aeroportos em um mapa.

4.2 Cronograma do Projeto Final II

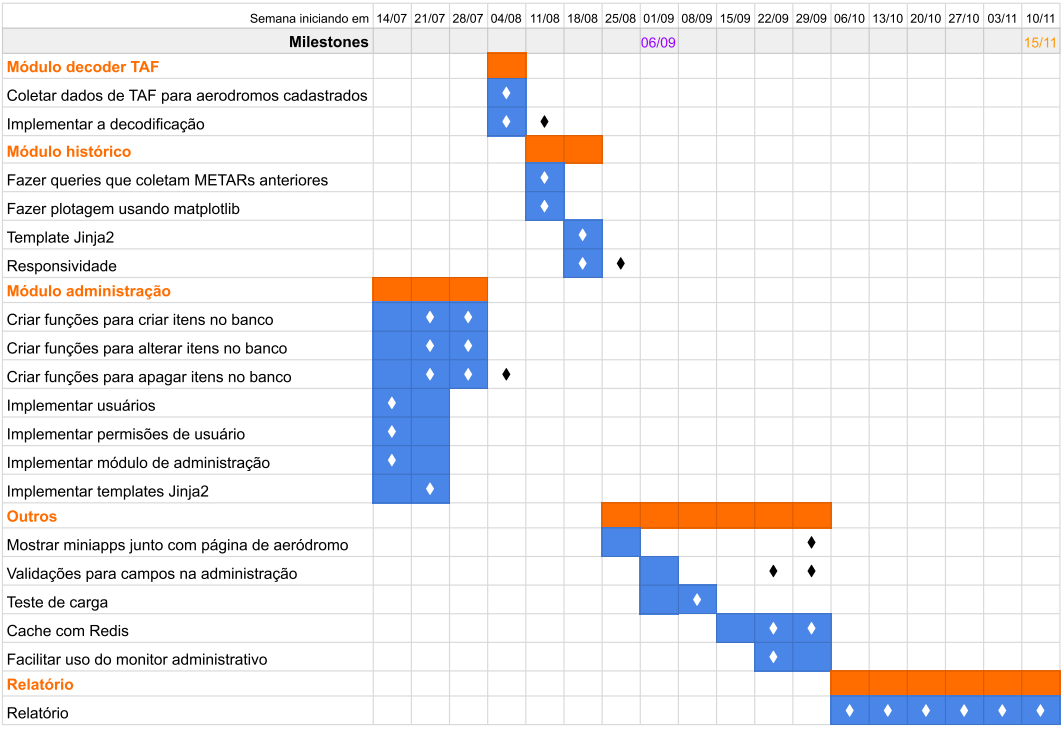


Figura 4.2: Cronograma II

Tabela 4.2: Milestones

Data	Milestone
06/09	Entrega do formulário de Projeto Final II
15/11	Entrega do Projeto Final II
25/11 a 29/11	Bancas Projeto Final II

5
Princípios norteadores

Para este projeto seguem-se os seguintes princípios por ordem de prioridade começando pela maior prioridade.

5.1
Exatidão da Informação Apresentada

Não deve 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, são comparadas constantemente as informações locais com o AISWeb e, em caso de alterações, é possível alterá-las facilmente pela área restrita do site. Normalmente, estas informações não mudam com tanta frequência. Um aeroporto pode passar anos ou até décadas sem nenhum dado ser alterado.

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.

Como será visto no capítulo de arquitetura existe uma proxy reversa com NGINX. Esta proxy faz cache das páginas. Então se uma página é recarregada muitas vezes seguidas o NGINX entrega a página salva no cache. O cache está configurado para ser invalidado em um minuto, então, no pior caso, as informações ficam com um minuto de atraso. A informação que atualiza com mais frequência é o METAR. A atualização ocorre de hora em hora, então o atraso de um minuto, no máximo, é aceitável.

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 infor-

mações de um aeródromo. Essas informações estão divididas em três páginas para cada aeródromo, para que a visualização não fique sobrecarregada.

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 projeto.

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 do nome 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 e uma pista pertence a um único aeródromo.

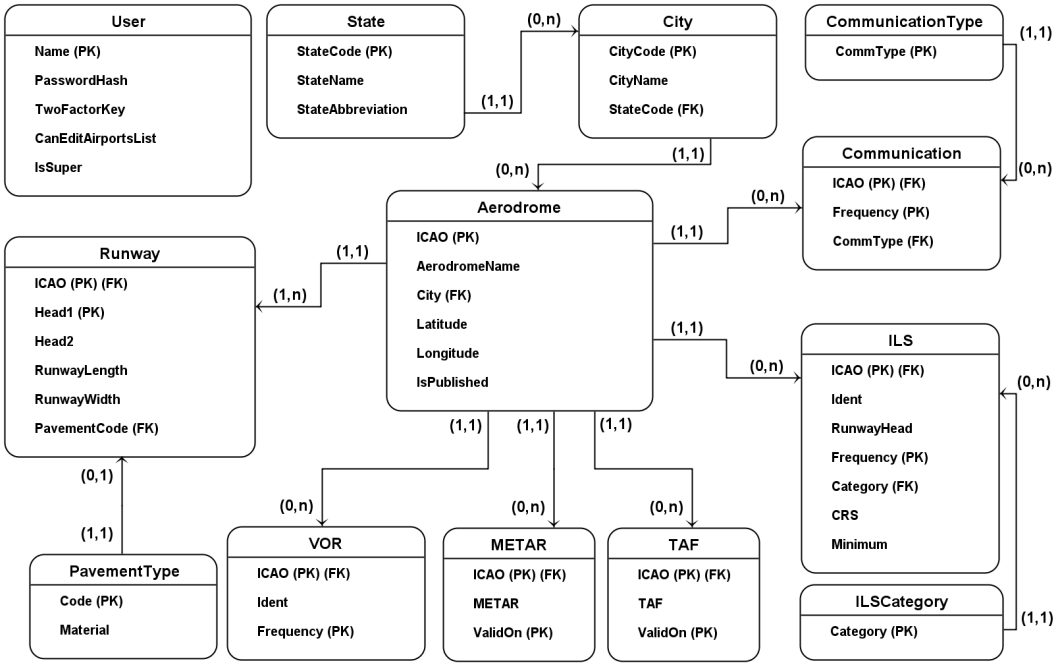


Figura 6.1: Diagrama lógico

Note-se 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 Aero 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 Noções sobre o Funcionamento de um Aeródromo

Antes de discutir as tabelas, é interessante compreender algumas regras e características básicas do funcionamento de um aeródromo. Isto certamente facilitará o entendimento das colunas das tabelas.

Um aeródromo é a área destinada às operações de pouso, decolagem, e taxiamento de aeronaves, abrangendo pátio, taxiways (vias de taxi) e pistas. O terminal de passageiros, no entanto, não faz parte do aeródromo em si. Por isso, todos os aeroportos são aeródromos, mas nem todo aeródromo é um aeroporto, pois nem todos possuem infraestrutura para o atendimento de passageiros.

6.2.1 Pistas

Para o entendimento do funcionamento das pistas de pouso e decolagem, é importante conhecer algumas de suas características essenciais, como com-

primento e largura, que impactam diretamente as operações das aeronaves.

O comprimento da pista, por exemplo, influencia a quantidade de frenagem necessária para uma aeronave parar com segurança após o pouso. Em pistas curtas, certos modelos de aeronaves não podem pousar com segurança. Já a largura da pista limita a envergadura máxima das aeronaves que podem operar nela. Em pistas muito estreitas, uma aeronave de grande porte, como o Boeing 747, pode ter dificuldades adicionais: os motores mais externos podem ultrapassar a área pavimentada, expondo-os a materiais no gramado, o que aumenta o risco de ingestão de objetos.

No caso de cabeceiras paralelas, ou seja, pistas que apontam para a mesma direção, temos as seguintes configurações:

6.2.1.1 Pista Única

A numeração da cabeceira da pista é determinada pela sua orientação magnética, dividida por 10 e arredondada. Por exemplo, em Fortaleza, a cabeceira com orientação de 126 graus é numerada como 13 (126 dividido por 10 é igual a 12,6, que arredondamos para 13).

6.2.1.2 Pista Dupla

Para pistas paralelas duplas, são usadas as letras "L"(Left, esquerda) e "R"(Right, direita) para distinguir as cabeceiras. No aeroporto Santos Dumont, por exemplo, temos as cabeceiras 02L (esquerda) e 02R (direita), quando o observador está de costas para o Pão de Açúcar.

6.2.1.3 Pista Tripla

Embora no Brasil não haja aeroportos com três pistas paralelas, em outros países, onde existem pistas triplas, são utilizadas as letras "L"(left/esquerda), "C"(center/central) e "R"(right/direita) para identificação das cabeceiras.

6.2.2 Comunicação

As frequências de comunicação em aeródromos utilizam modulação de amplitude (AM), com banda de 118 MHz a 137 MHz. Cada aeródromo possui frequências específicas para cada fase do voo, como autorização, taxi, decolagem, aproximação e pouso.

Os tipos de frequência achadas em aeroportos do Brasil são:

1. Tráfego
2. Torre
3. Solo
4. Rampa
5. ATIS
6. Operações

Nesta trabalho, para cada aeroporto, são mostradas as frequencias usadas com seus tipos.

6.2.3 Navegação

As frequências de navegação são utilizadas pelos sistemas de localização do avião, não transmitindo áudio, mas sim dados, dentro da faixa de 108 a 117.95 MHz. Esses sistemas ajudam a determinar a posição e a trajetória da aeronave com precisão.

- VOR (Very High Frequency Omnidirectional Range): Um sistema de navegação baseado em uma antena no solo, o VOR permite que a aeronave determine a direção que deve seguir para passar diretamente sobre a antena, indicando essa direção em um instrumento específico na cabine. Na aviação isto é chamado de bloqueio de VOR. [6]
- ILS (Instrument Landing System): Sistema de aterrissagem por instrumentos que transmite informações de orientação lateral (alinhamento com o eixo da pista) e orientação vertical (ângulo de descida) específicas para cada cabeceira da pista. Essas informações permitem uma aproximação precisa, mesmo em condições de baixa ou nenhuma visibilidade. O ILS pode ser usado tanto para pouso manual, onde o piloto controla a aeronave seguindo as indicações, quanto para pouso automático (auto land), caso a aeronave possua esta capacidade. [7]

Neste trabalho são mostradas as frequências de ILS e VOR para cada aeroporto.

6.3 Dicionário de Dados

Tabela 6.1: State

Nome	**Descrição**	**Tipo**
StateCode (PK)	Código do IBGE [8] do estado	INTEGER
StateName	Nome do estado escrito em Português com a primeira letra maiúscula	VARCHAR (50)
StateAbbreviation	Abreviatura de duas letras do estado. Ambas as letras maiúsculas.	VARCHAR(2)

Tabela 6.2: City

Nome	**Descrição**	**Tipo**
CityCode (PK)	Código do IBGE [8] da cidade	INTEGER
CityName	Nome da cidade escrito em Português com a primeira letra maiúscula	VARCHAR (50)
StateCode (FK)	Chave estrangeira para o estado	INTEGER

Tabela 6.3: 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)	Chave estrangeira para o código da cidade	INTEGER

Continua na próxima página

Tabela 6.3 – Continuação da página anterior

Nome	**Descrição**	**Tipo**
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)
IsPublished	Se for falso o aeródromo não aparece no site. Serve para que os aeródromos recém-criados não apareçam enquanto outras informações ainda não foram cadastradas.	Boolean

Tabela 6.4: 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 6.5: TAF

Nome	**Descrição**	**Tipo**
ICAO (PK) (FK)	Chave estrangeira para qual aeródromo este TAF se refere	VARCHAR(4)
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 TAF "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 6.6: 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
```

Tabela 6.7: 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	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)

Tabela 6.8: 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(20)

Tabela 6.9: 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-se 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
CommType (FK)	O tipo de comunicação, chave estrangeira para '*CommunicationType*'.	VARCHAR(20)

As duas tabelas a seguir listam as diferentes categorias de Sistema de Pouso por Instrumentos (Instrument Landing System). Para as cabeceiras com este sistema é possível pousar mesmo sem ter a pista no visual.

Tabela 6.10: 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 6.11: 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
Ident	Identificação de três letras maiúsculas única do ILS para aquele aeródromo. Aparece na carta aérea do procedimento ILS.	VARCHAR(3)
RunwayHead	Para qual cabeceira este ILS se refere.	VARCHAR(3)
Category (FK)	A categoria do ILS, referenciando a tabela '*ILSCategory*'.	VARCHAR(10)

Continua na próxima página

Tabela 6.11 – Continuação da página anterior

Nome	**Descrição**	**Tipo**
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 6.12: 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)
Continua na próxima página		

Tabela 6.12 – Continuação da página anterior

Nome	**Descrição**	**Tipo**
Frequency (PK)	A frequência de operação do VOR/DME em MHz multiplicada por 10. Forma chave primária junto com ICAO.	INTEGER
Ident	Identificação única do VOR/DME para aquele aeródromo.	VARCHAR(3)

6.4 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 como gerenciar as permissões de cada usuário.

Tabela 6.13: User

Nome	**Descrição**	**Tipo**
Name (PK)	Nome do usuário. Usado no login.	VARCHAR(30)
PasswordHash	*Hash* com *salt* da senha do usuário. O padrão *bcrypt* é usado para criação do *hash* e autenticação.	VARCHAR(60)
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)

Continua na próxima página

Tabela 6.13 – Continuação da página anterior

Nome	**Descrição**	**Tipo**
CanEditAirports List	Lista separada por vírgulas dos ICAOs dos aeroportos que este usuário tem permissão de alterar. Entenda-se "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*. Apenas este tipo de usuário pode apagar o aeródromo inteiro ou criar um novo.	Boolean

7
Arquitetura

O sistema possui backend escrito na linguagem Python, fazendo uso da framework FastAPI. Até a entrega do Projeto I, o backend era feito em Flask. A escolha foi feita pela familiaridade com a framework. Porém, depois, descobriu-se 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 a partir dos dados recebidos.

Contudo, é possível retornar qualquer tipo de dado no FastAPI, inclusive páginas HTML. Como se preferiu fazer a renderização de páginas no lado do servidor, continuou-se o uso da funcionalidade de templates do Jinja2.

A substituição da framework deu-se pelos seguintes motivos:

1. Projeto mais maduro, continuamente mantido;
2. Documentação bem mais detalhada do que a do Flask, com vários tutoriais sobre assuntos comumente usados, como autenticação e validação de dados;
3. Pensado para validação de dados, se integra muito bem ao Pydantic;
4. Servidor embutido (Uvicorn) extremamente fácil de configurar e suficiente para produção [9]. No Flask é necessário configurar alguns parâmetros para o Guinicorn via o arquivo "gunicorn_config.py";
5. Assíncrono, permitindo o uso junto com bibliotecas que utilizam asyncio; No Flask, para paralelismo, é necessário usar uma biblioteca como o Gevent.

A arquitetura foi pensada para ser implementada com o *Docker* e *Docker Compose*. Cada programa executado é rodado em um serviço separado.

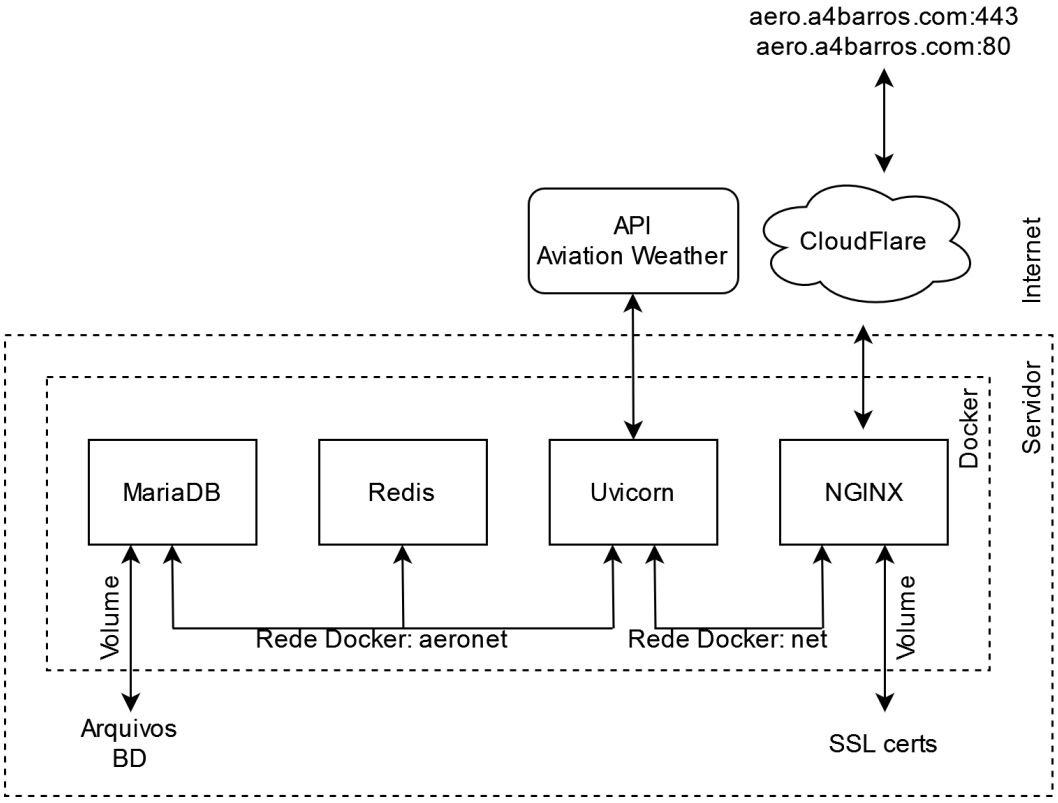


Figura 7.1: Modelo de Arquitetura

7.1 Docker Network

A porta 5000 do Uvicorn 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 Uvicorn não pode ser acessado por `localhost:5000`, e sim por `http://aero:5000`, "aero"sendo o nome do serviço no Docker Compose. Na máquina host e em containers que não tenham a rede configurada para estas, não conseguem ver o Uvicorn.

7.2 Docker Secrets

Para aumentar a segurança de acesso ao banco, é usada a funcionalidade "secrets". Nela, no Docker Compose, é informado um arquivo de texto no host, onde estará uma senha, uma senha por arquivo. No mesmo Compose, informa-se 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 Uvicorn 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 fixa sobre os aeródromos e as dinâmicas como o METAR e TAF conforme explicado no capítulo de modelo de dados. É um banco tradicional da indústria e extremamente confiável.

7.3.2 Redis

Para um banco gravado em memória secundária, o tempo de acesso pode prejudicar a performance do site durante picos de acessos. Para diminuir o tempo de resposta das funções que usam o banco de dados foi implementado um decorator Python para salvar as respostas em um banco em memória, no caso o Redis. Mais informações no capítulo de teste de carga.

7.3.3 FastAPI

O site foi construído com a framework FastAPI. Tanto para desenvolvimento quanto para produção, é utilizado o servidor embutido do FastAPI, o Uvicorn. A configuração padrão funcionou bem ao longo do tempo. Como explicado com mais detalhes do capítulo "Proposta", apesar do FastAPI ser usado normalmente para a construção de APIs, ele pode retornar qualquer tipo de dado, inclusive imagens, vídeos e páginas HTML. Para páginas HTML precisa ser informado no decorator que a classe de resposta é "HTMLResponse".

Código 1: Exemplo de Decorator

```
1 @app.get("/taf/{icao}", response_class=HTMLResponse)
2 async def info_taf(request: Request, icao: str):
3     ...
```

7.3.4 Proxy NGINX

O NGINX realiza o SSL, 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 [10]. O Uvicorn, no momento em que este texto foi escrito, não suporta HTTP/2, mas suporta o keep-alive. No entanto, como há outros projetos do autor na mesma máquina, ultilizam-se subdomínios. Todos os subdomínios resolvem para a mesmo 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".

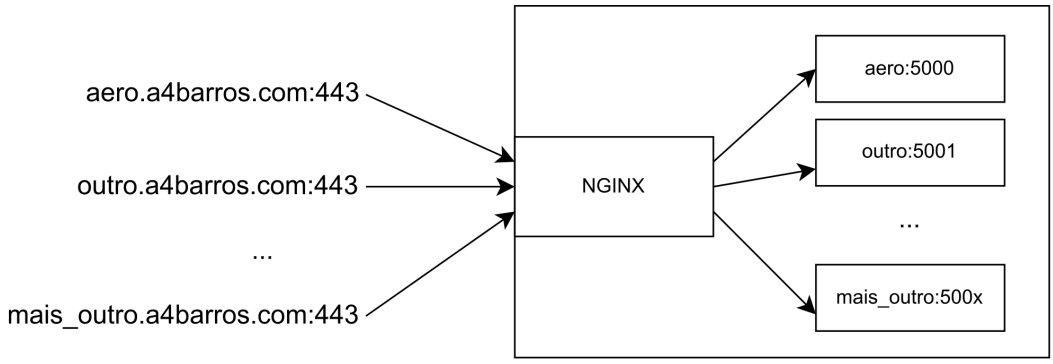


Figura 7.2: Diagrama do funcionamento de vários subdomínios resolvendo no mesmo IP. Com o NGINX funcionando como proxy.

7.4 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

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

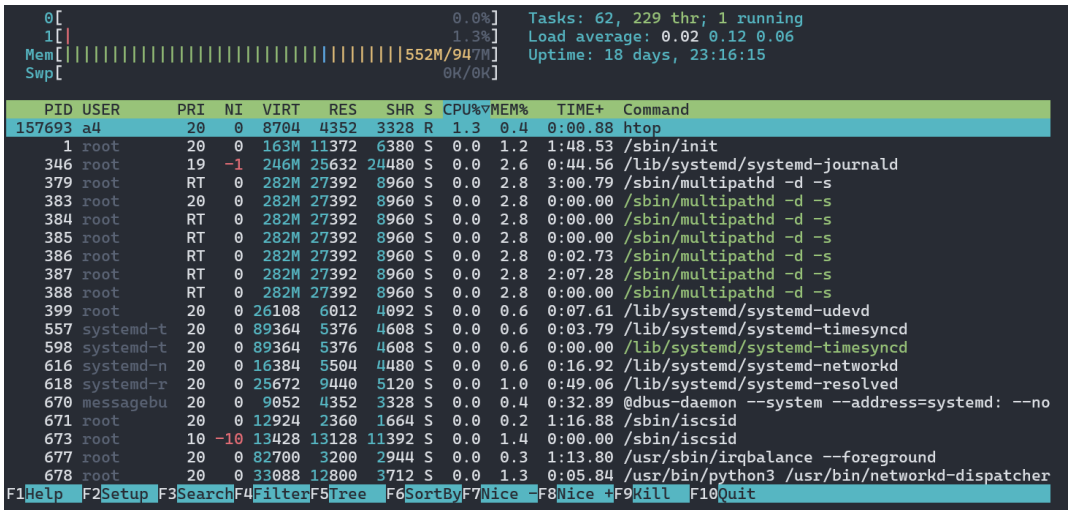


Figura 7.3: Uso do sistema em baixa demanda

Figura 7.4: Containers Docker em execução

7.5 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). Fez-se tal separação para garantir que as APIs externas (METAR e TAF) sejam acessadas apenas quando necessário sem que uma quantidade grande de acessos ao site sobrecarregue estas APIs.

O desenho dos plots com dados históricos é uma operação que demora aproximadamente 3 segundos por aeródromo, então faz-se esta operação em background e deixa-se o arquivo .svg já pronto em uma pasta específica.

A seguir, está descrito quando ocorrem estas operações assíncronas. Note-se que são feitas com mais frequência que o necessário, para se ter certeza que uma informação não ficará muito tempo desatualizada caso ocorra algum atraso para atualização do METAR/TAF.

O jitter de 30 segundos serve para adicionar um offset aleatório no

tempo para que a atualização não ocorra exatamente no segundo zero onde provavelmente o servidor do Aviation Weather estará sobre uma carga maior.

Tabela 7.1: Operações assíncronas

Nome da função	**Quando ocorre**	**Descrição**
update_metars	De vinte em vinte minutos com jitter de 30 segundos	Normalmente, um novo METAR é disponibilizado a cada hora, mas é possível ter um METAR a cada meia hora ou até menos. Para cada ICAO presente na tabela Aerodrome, cria-se 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).
update_tafs	De três em três horas	Os TAFs são atualizados com menos frequência que os METARs, normalmente nos horários 00:00, 06:00, 12:00 e 18:00 UTC.
update_images	De vinte em vinte minutos com jitter de 30 segundos	Criação dos plots. Obviamente precisa ter a mesma frequência da atualização de METAR.

7.6 Diagrama de sequência

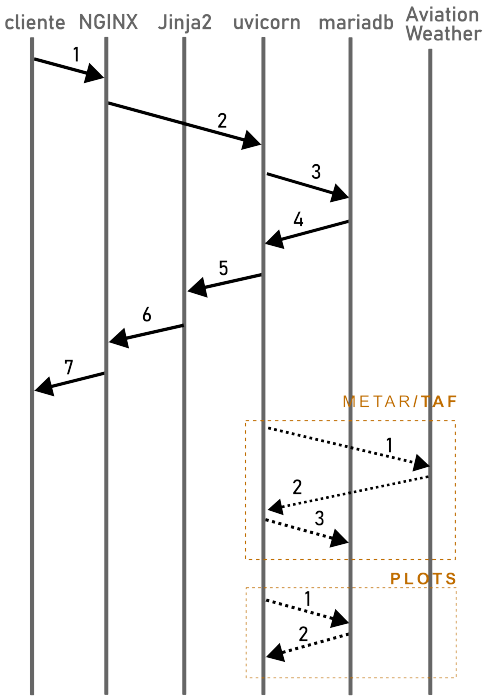


Figura 7.5: Diagrama de sequência

Para não poluir o diagrama não foi incluído o Redis, mas ele ficaria entre o Uvicorn e o MariaDB. Primeiro é tentando o Redis caso a chave exista, o banco não é acessado. Existem operações que invalidam o cache como será mostrado ainda neste capítulo.

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 aparentam ser robôs. Existe um limite de 3 requisições por minuto para a rota de login. Caso a requisição passe pelos filtros, é realizado um proxy-pass para o servidor Uvicorn.

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, ILS, Communication, Runway... 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.
6. A página HTML é gerada.
7. O usuário recebe esta página.

No momento de 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 separados 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;
4. O cache para todos os ICAOs são invalidados.

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

1. É feito um query para o banco pedir os 12 últimos METARs de cada aeródromo;
2. O banco responde. A biblioteca matplotlib é usada para criar os plots que são salvos 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 [11]. Não se trata de uma previsão do tempo, mas sim de uma visualização atual. Para previsões existe o TAF que será explorado no próximo capítulo. 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 [12], 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 [13] ou menos, o autor já viu o METAR do aeroporto do Galeão sendo atualizado na hora cheia e depois no minuto 16. Nas próximas seções, será apresentado um exemplo de METAR e sua decodificação bem como uma explicação do algoritmo.

8.1
Exemplo

O METAR no aeroporto de Fortaleza (Pinto Martins)[14], no dia 17 de abril de 2024 às 10:54 foi

`SBFZ 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 leigo 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 [15] [16].

– **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 [17]. Para que não haja confusão com fusos horários, a aviação usa o horário UTC. Em tempo estável, este METAR será válido até às 13:59, quando será substituído pelo METAR iniciando com "SBFZ 171400Z". Pode haver alteração antes em caso de mudança repentina de condições atmosféricas.

Note-se 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 na bússola, 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 (variable) 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})KT` 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 [18]. O regex usado é

  ```
  (M?[0-9]{2})/(M?[0-9]{2})
  ```

  Observe o M opcional. É necessário depois fazer a substituição do M para o sinal de menos.

– **Q1011** O altímetro do avião deve ser ajustado para 1012 hectopascal. Também pode ser usada a unidade polegadas de mercúrio (mmHg), mas no Brasil esta não é usada no METAR. O regex é`(Q[0-9]{4})`

– **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 [19].

– **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. O regex usado é `([A-Z]{3})([0-9]{3})(TCU)?`. O primeiro grupo de captura é comparado com "FEW", "BKN", "SCT"e "OVC".

8.2 Algoritmo

O objetivo do módulo de decoder é dar uma explicação automática 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 [19].

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

8.3 Exemplo

Sendo "$1"o primeiro grupo de captura, "$2"o segundo e "$3"o terceiro, o item "27008G16KT", por exemplo é comparado com todas as expressões regex e ocorre o match com apenas a seguinte:

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

que é mapeada para 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 enviada à ferramenta de templating de página Jinja.

A seguir um exemplo para esta lista de tuplas:

```
[
    ('251600Z', 'METAR válido para dia 25 as 16:00 (UTC)'),
    ('22011KT', 'Vento proa <b>220</b>° com velocidade <b>11</b>
    nós (kt).'),
    ('9999', 'Visibilidade ilimitada'),
    ('SCT017', 'Nuvens espalhadas (3/8 a 4/8 do céu com nuvens) em
    <b>1700</b> pés de altitude. '),
    ('BKN023', 'Nuvens broken (5/8 a 7/8 do céu com nuvens) em
    <b>2300</b> pés de altitude. '),
    ('OVC035', 'Totalmente encoberto em <b>3500</b> pés de altitude.
        '),
    ('21/16', 'Temperatura <b>21</b>°C e ponto de orvalho <b>16
    </b>°C.'),
    ('Q1022', 'O altímetro deve ser ajustado para <b>1022</b> hPa
    (30.18 inHg)')
]
```

E como é mostrado na página:

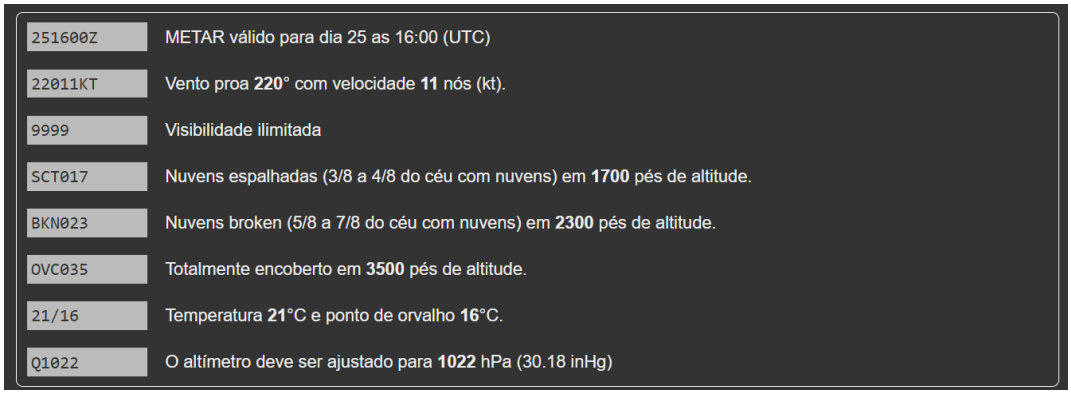


Figura 8.1: METAR do aeroporto do Galeão dia 30 de agosto de 2024

9
Decodificação do TAF

O TAF (*Terminal Aerodrome Forecast*) é um protocolo que fornece previsões para aeroportos. Diferente do METAR, que relata as condições meteorológicas atuais, o TAF projeta o que se espera para as próximas horas e/ou dias. Uma diferença em relação ao METAR é que o TAF se entende por várias linhas. A partir da segunda linha, temos os chamados grupos temporais. Tudo que está em uma linha são condições que irão ocorrer dentro de uma faixa de tempo.

Um grupo temporal pode começar com BECMG ou TEMPO.

1. BECMG: Indica uma mudança gradual e definitiva nas condições meteorológicas;

2. TEMPO: Indica uma mudança temporária nas condições.

Neste capítulo, será apresentado um exemplo de TAF, sua decodificação, e uma explicação do algoritmo utilizado.

9.1
Exemplo

O TAF emitido para o aeroporto de Pinto Martins (SBFZ) no dia 31 de agosto de 2024 às 09:00Z foi:

```
310900Z 3112/0112 15010KT CAVOK TX31/3115Z TN24/0108Z
  BECMG 3115/3117 11015KT FEW025
  BECMG 0104/0106 15010KT RMK PHD
```

- **310900Z** significa que este TAF foi gerado dia 31 às 09:00 UTC. 3112/0112 indica o período válido desta informação: do dia 31 às 12 horas até dia primeiro às 12 horas. 15010KT indica um vento de 10 nós vindo da direção 150 graus. Não há nuvens e a visualidade é máxima. É usado o acrônimo CAVOK (Ceiling Altitude and Visibility OK, Teto e Visibilidade OK) para indicar isto.
- **TX31/3115Z TN24/0108Z** A temperatura mínima será de 24 graus dia 01 às 08:00 (UTC) e a máxima será de 31 graus no dia 31 as 15:00 (UTC).

  A seguir, vêm os grupos de previsão, cada um em uma linha.

– **BECMG 3115/3117 11015KT FEW025**: Dia 31 das 15 às 17 horas UTC terá vento de 15 nós com 110 graus. E haverá poucas nuvens na altitude no nível 025 ou 2500 pés.

– **BECMG 0104/0106 15010KT RMK PHD** : dia primeiro de setembro das 04:00 as 06:00, o vento será de 10 nós com 150 graus.

RMK adiciona uma observação (remark). O trigrama PHD representa a identificação do previsor que gerou este TAF. É algo específico do Brasil [16].

É claro que há muitos outros itens possíveis dentro da especificação. PROB30 em um grupo de previsão, por exemplo, informa que as informações nesta linha tem 30% de chance de ocorrer.

O objetivo do decodificador é explicar corretamente qualquer TAF brasileiro.

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 com expressões regulares.

Para cada item do TAF, o algoritmo tenta associar a string correspondente a uma descrição. Por exemplo, a expressão

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

Sendo:

– $1: Dia início

– $2: Hora início

– $3: Dia fim

– $4: Hora fim

é usada para identificar os períodos de tempo, enquanto a expressão

```
TX(M?[0-9]{2})/([0-9]{2})([0-9]{2})Z
```

Sendo:

– $1: Temperatura máxima (M opcional indicando negativa)

– $2: Dia em que a temperatura máxima vai ocorrer

– $3: Hora em que a temperatura máxima vai ocorrer

é utilizada para extrair a temperatura máxima em um dia e hora.

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.

Perceba no exemplo de resposta a seguir que os grupos temporais são agrupados, isto serve para que no front end possamos ocultar cada grupo.

9.3 Exemplo de Resposta

```
["251600Z",
    "Disponibilizado em 16:00Z no dia 25"],
    ["2518/2624",
    "Válido do dia 25 as 18:00Z até dia 26 as 24:00Z"],
    ["23012KT",
    "Vento proa <b>230</b>° com velocidade <b>12</b> nós (kt)."],
    ["8000",
    "Visibilidade 8000 metros"],
    ["BKN020",
    "Nuvens broken (5/8 a 7/8 do céu com nuvens) em <b>2000</b>
    pés de altitude. "],
    ["BKN030",
    "Nuvens broken (5/8 a 7/8 do céu com nuvens) em <b>3000</b>
    pés de altitude. "],
    ["TN16/2609Z",
    "A temperatura mínima é de 16°C prevista de ocorrer dia 26
    as 09:00 (UTC)"],
    ["TX20/2615Z",
    "A temperatura máxima é de 20°C prevista de ocorrer dia 26
    as 15:00 (UTC)"],
    [
        ["TEMPO 2518/2521",
        "Condições temporárias previstas do dia 25 as 18:00 (UTC)
        até dia 25 as
        21:00 (UTC)"],
        ["23012G25KT",
        "Vento proa 230° com velocidade 12 nós (kt) e <b>rajadas
        </b> de 25 nós."],
        ["5000",
        "Visibilidade 5000 metros"],
        ["DZ",
        "Chuvisco moderada."],
        ["BR",
        "Névoa úmida moderada."],
        ["BKN015",
        "Nuvens broken (5/8 a 7/8 do céu com nuvens) em <b>1500</b>
```

```
37          pés de altitude. "],
38          ["BKN020",
39          "Nuvens broken (5/8 a 7/8 do céu com nuvens) em <b>2000</b>
40          pés de altitude. "]
41      ],
42      [
43          ["TEMPO 2521/2524",
44          "Condições temporárias previstas do dia 25 as 21:00 (UTC)
45          até dia 25 as 24:00
46          (UTC)"],
47          ["25010KT",
48          "Vento proa <b>250</b>° com velocidade <b>10</b> nós (kt)."
                ],
49          ["4000",
50          "Visibilidade 4000 metros"],
51          ["RA",
52          "Chuva moderada."],
53          ["BR",
54          "Névoa úmida moderada."],
55          ["BKN010",
56          "Nuvens broken (5/8 a 7/8 do céu com nuvens) em <b>1000</b>
57          pés de altitude. "],
58          ["BKN020",
59          "Nuvens broken (5/8 a 7/8 do céu com nuvens) em <b>2000</b>
60          pés de altitude. "]
61      ]
```

10
Plotagem do METAR Histórico

Para o piloto, é importante ter 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. Visualizar essas informações em gráficos facilita a análise e a tomada de decisões.

Os dados meteorológicos (METARs) coletados de um aeroporto são armazenados na tabela METAR (vide capítulo de Modelo de Dados) com chave estrangeira ICAO, possibilitando o armazenamento de múltiplos registros para um único aeródromo. A partir dessa tabela, é realizada uma query que obtém os 12 últimos registros de METAR de cada aeródromo. Desse conjunto de dados, usando regex, de forma semelhante ao Decodificador de Metar, 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,
    },
    {
        "temperature": 21 ,
        "dew_point": 13,
        "wind_speed": 5,
        "wind_direction": 129,
        "qnh": 1028,
        "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. Temos dois eixos Y e eixo X compartilhado. Os gráficos são salvos como 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 `<img>`, que apontam para os gráficos salvos na pasta estática.

Como na consulta do METAR, esta tarefa é realizada assincronamente nos minutos 0, 20, 40 (vide capítulo de arquitetura). Quando o usuário carrega a página, as plotagens já se encontram prontas seguindo o segundo princípio norteador.

11
Front-end

Como mencionado no capítulo da arquitetura, a página é gerada no lado do servidor. Então o que é retornado para cada rota, é um HTML já pronto. O autor decidiu fazer desta forma como experimentação da funcionalidade de template e porque acredita ser mais performático fazer desta forma.

As *frameworks* atuais de frontend como React e Vue pela experiência profissional do autor são lentas para fazer a build e lentas para carregar a página. Isto parece ser devido as muitas dependências com outras bibliotecas. O objetivo para este projeto conforme o segundo princípio norteador é o carregamento rápido.

Além disso, o desenvolvimento é mais rápido, já que não é necessário codificar o front-end. Para um projeto individual isto tem funcionado bem, mas claro que, para um projeto mais complexo com mais pessoas envolvidas, seria melhor fazer uma API REST já teríamos um time para back-end e outro para front-end.

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 mistura entre controlador e visão. Pois não há HTML sendo escrito dentro do backend. Como já dito, o que é passado para o Jinja é algo equivalente a JSON. Transformar este backend em uma API REST não seria muito difícil.

Uma 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 e listas, é fácil acessar os valores. Neste projeto, para exibir a lista de frequência, o seguinte código é usado.

Código 2: 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.
                    Frequency }}/edit" class="button-admin">Editar</a>
8               {% endif %}
9           </div>
10      </div>
11      {% endfor %}
12  </div>
```

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}}".

Note-se 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 existe uma seção de usuário aberta para que o administrador consiga editar um aeródromo.

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

11.1 Imagens da Interface Gráfica

`https://aero.a4barros.com/`

Figura 11.1: Página Inicial

`https://aero.a4barros.com/info/SBGL`

Figura 11.2: Dados do aeroporto e METAR

https://aero.a4barros.com/taf/SBGL

Figura 11.3: TAF

https://aero.a4barros.com/history/SBGL

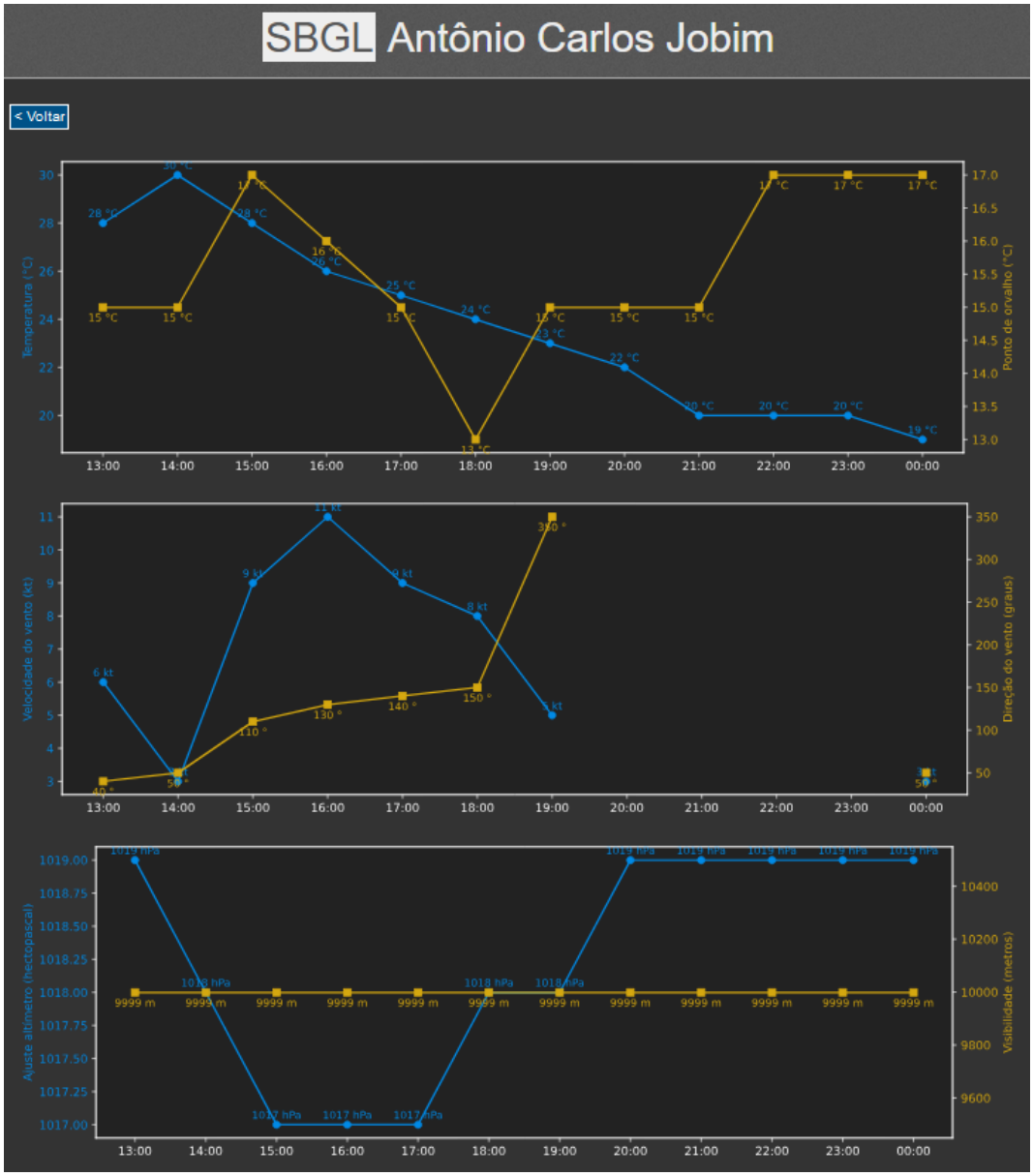


Figura 11.4: Plotagem das informações históricas. O buraco nas informações de vento é devido ao fato que os METARs publicados nestes horários não tinham o item de vento

As rotas a seguir são utilitários de apoio que podem ser abertos clicando no botão de elipsis na página de qualquer aeroporto.

Figura 11.5: Acesso aos mini-utilitários

Figura 11.6: Calculadora das componentes de vento

Figura 11.7: Vento atual para cada cabeceira de um aeroporto

Figura 11.8: Calculadora de perfil de descida

12
Rotas do back-end

A seguir são listadas e explicadas todas as rotas públicas do backend. As rotas de administração são explicadas na seção "Interface de Administração".

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` dos aeródromos com isPublished como verdadeiro e o resultado é posto em uma lista de tuplas que é enviada para o template.

Para um usuário logado também são mostrados os aeródromos não publicados.

Exemplo do resultado enviado à ferramenta de template:

Código 3: Resposta da root

```
{
    "airports": [
        [
            "Presidente Juscelino Kubitschek",
            "SBBR",
            "Bras\u00edlia"
        ],
        [
            "Tancredo Neves",
            "SBCF",
            "Belo Horizonte"
        ],
        [
            "Afonso Pena",
            "SBCT",
            "Curitiba"
        ],
  ...
    ]
}
```

12.2
Rota: /info/{ICAO}

Retorna informações do aerodrómo e seu metar decodificado. A seguir um exemplo de resultado enviado ao *Jinja2*.

Código 4: Resposta da rota info

```
{
    "info": {
        "AerodromeName": "Pinto Martins",
        "Latitude": -3.7764,
        "Longitude": -38.532041,
        "IsPublished": true,
        "runways": [
            {
                "Head1": "13",
                "Head2": "31",
                "RunwayLength": 2755,
                "RunwayWidth": 45,
                "PavementCode": "ASP"
            }
        ],
        "ils": [
            {
                "Ident": "IFO",
                "RunwayHead": "13",
                "Frequency": 1103,
                "Category": "I",
                "CRS": 126,
                "Minimum": 200
            }
        ],
        "vor": [
            {
                "Ident": "FLZ",
                "Frequency": 1141
            }
        ],
        "communication": [
            {
                "Frequency": 121500,
                "CommType": "Torre"
            },
            {
                "Frequency": 121950,
                "CommType": "Solo"
            },
            {
                "Frequency": 122500,
                "CommType": "Operações"
            },
            {
                "Frequency": 127700,
                "CommType": "ATIS"
            }
        ]
    },
```

```
51      "icao": "SBFZ",
52      "metar": "050100Z 10010KT 9999 BKN020 26/22 Q1014",
53      "go_back": "/",
54      "decoded": [
55          // Ver capítulo de Decodificação do METAR
56      ]
57      }
```

12.3 Rota: /history/{ICAO}

É uma página estática com os gráficos históricos para os doze últimos METARs. Mais informações do capítulo "Plotagem do METAR Histórico".

12.4 Rota: /taf/{ICAO}

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

Exemplo do resultado enviado à ferramenta de template para o aeroporto do Galeão.

Código 5: Resposta da rota tag

```
1 {
2      "icao": "SBGL",
3      "taf":"251600Z 2518/2624 23012KT 8000 BKN020 BKN030 TN16/2609Z
           TX20/2615Z \n
4      TEMPO 2518/2521 23012G25KT 5000 DZ BR BKN015 BKN020 \n  TEMPO
           2521/2524 25010KT
5      4000 RA BR BKN010 BKN020 \n  TEMPO 2600/2612 22008KT 3100 RA BR
           BKN007 BKN015 \n
6      BECMG 2612/2614 26007KT 7000 BKN016 BKN023 \n  BECMG 2618/2620
           32005KT BKN017
7      RMK PGA","decoded": [
8          // Ver capítulo de Decodificação do TAF
9      ]
10 }
```

12.5 Rota: /descent

Página estática onde é possível calcular o perfil de descida. Dada a altitude inicial, altitude final, velocidade e razão de descida (graus ou pés por minuto) é retornado a distância horizontal requerida e o tempo requerido para fazer esta descida. O cálculo é feito apenas com JavaScript na própria página.

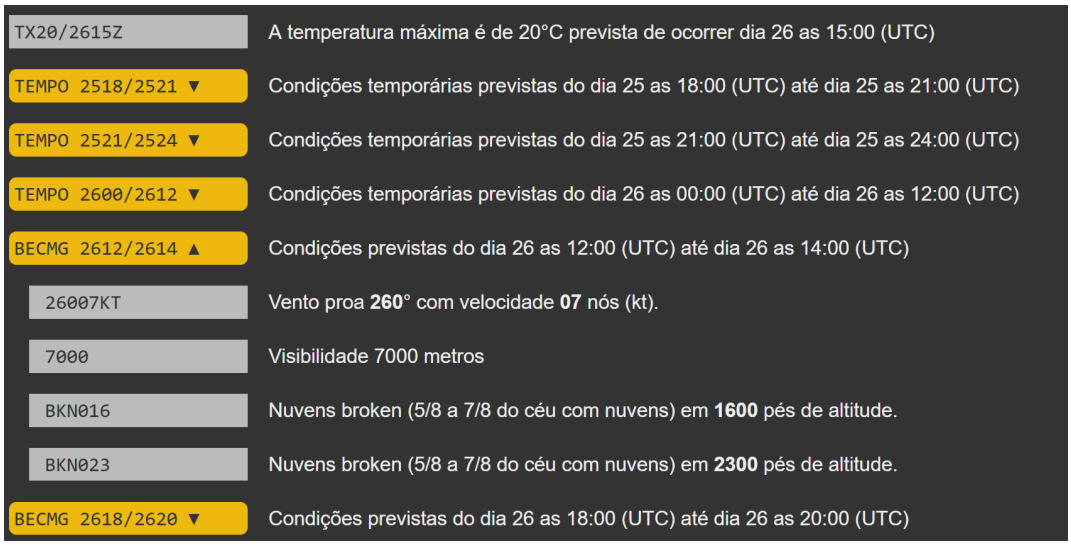


Figura 12.1: Grupo BECMG exibido

12.6 Rota: /wind

Dada a direção da pista, a direção do vento e velocidade do vento, é calculada a componente paralela e perpendicular do vetor de vento em relação à aeronave. Consulte a capítulo de front end para ver imagens da interface desta e das outras rotas.

12.7 Rota: /windcalc

Usada pela rota anterior para calcular as componentes do vento. Para a chamada `/windcalc/?wind_dir=100&wind_speed=3&runway_head=90` a resposta será

```
{"cross":0.52,"head":2.95,"angle":10.0}
```

Os valores retornados serão mostrados para o usuário como números e também serão usados para modificar uma imagem SVG que mostra graficamente a influência das componentes de vento.

12.8 Rota: /wind/{icao}

Parecida com a rota anterior, mas considera cada cabeceira do aeródromo selecionado e o vento atual.

13
Interface de Administração

O Aero possui uma interface administrativa com login em `/area/restrita`. Para se autenticar, o usuário deve digitar seu nome de usuário, senha e código TOTP (a autenticação de dois fatores por tempo). A senha é armazenada no banco com hash e salt usando o padrão bcrypt. Para o TOTP é utilizada a biblioteca pyOTP.

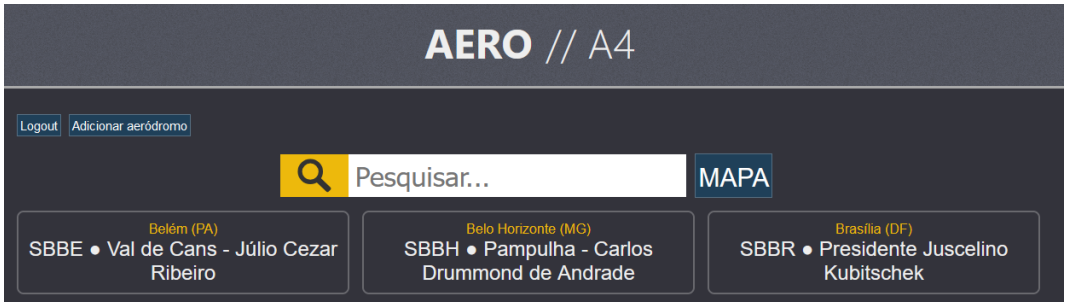


Figura 13.1: Tela inicial da área logada

Para criar e alterar as contas usa-se o script "monitor.py"executado dentro de um SSH. O script monitor.py permite fazer diversas operações de máximo privilégio incluindo criar, alterar senha e TOTP e apagar usuários. Forçar a atualização de METARs e TAFs também é possível. Tais tarefas não são feitas com frequência. O que é feito com maior frequência é a adição de aeroportos e alteração deles.

Figura 13.2: monitor.py

Um usuário com o `isSuper` em verdadeiro pode adicionar e remover aeródromos bem como editar e remover qualquer aeródromo. Um usuário normal pode apenas alterar aeródromos com ICAO fazendo parte da lista autorizada para ele no momento da criação de sua conta.

O login é mantido via cookie assinado com padrão JWT de forma que não é custoso (não envolve acesso ao banco) verificar se está logado. É usada a biblioteca padrão do *fastAPI* já que ela assina os cookies automaticamente. Ao estar logado, na tela inicial, dois botões são mostrados, o de logout e o de adicionar aeródromo. Caso o usuário não seja super o segundo botão não aparece e, obviamente, a rota GET e POST de adicionar aeródromo retorna um erro.

Ao entrar em um aerodromo, caso tenha permissão de altera-lo, a seguinte tela é mostrada.

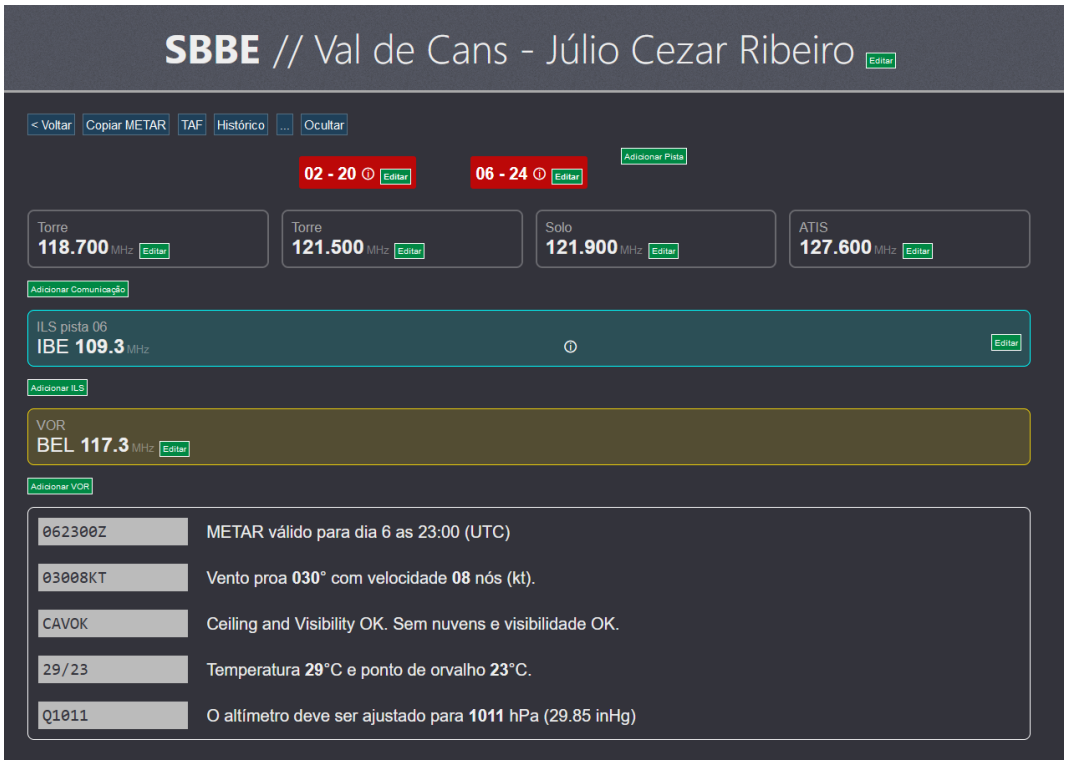


Figura 13.3: Tela de aeródromo da área logada

Pressionando nos botões verdes, é possível criar e alterar qualquer dado do aeroporto. Dentro da edição de um aeródromo é possível, apagá-lo.

13.1 Adicionar/editar aeródromo

Caso tenha permissão a rota `/area/restrita/add` permite adicionar um aeródromo.

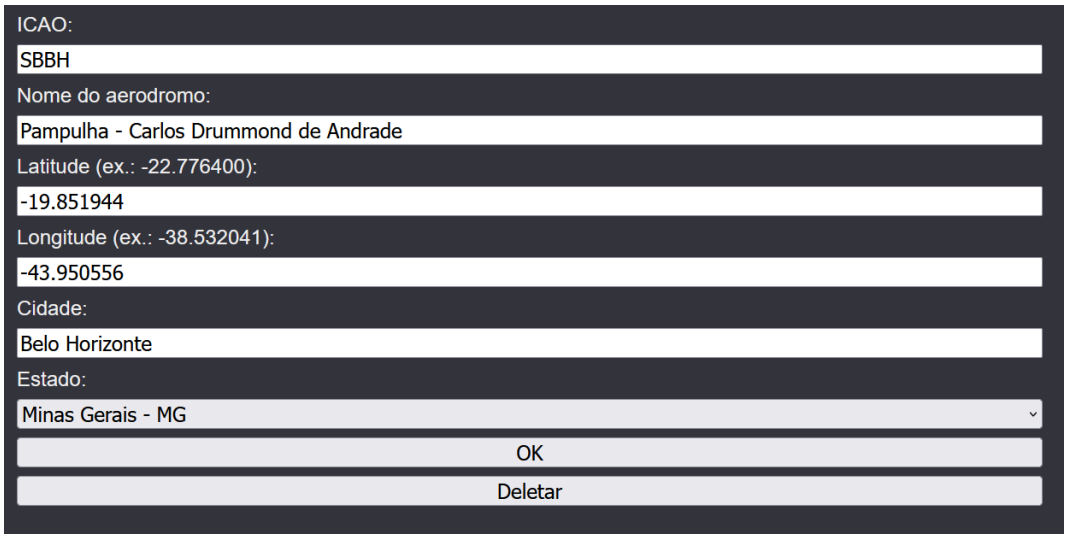


Figura 13.4: Tela de adicionar / editar um aeródromo

Obviamente o botão de apagar aparece apenas na tela de edição.

Todos os campos possuem algum tipo de validação tanto via HTML quanto via servidor. Caso o HTML fosse alterado via developer tools, mesmo que rotas de administração são acessíveis apenas por usuários autorizados, ainda assim não seria possível para um bad actor adicionar no banco um record inválido.

As validações para esta rota são:

Tabela 13.1: Rotas: /area/restrita/<icao>/edit e /area/restrita/add

Campo	**Frontend**	**Backend**
ICAO	Regex: `[A-Z]{4}`	Regex: `[A-Z]{4}`
Nome do aerodromo	Type: Text	String
Latitude	Regex: `-?[0-9]*\.[0-9]{6}`	Regex: `-?[0-9]*\.[0-9]{6}`
Longitude	Regex: `-?[0-9]*\.[0-9]{6}`	Regex: `-?[0-9]*\.[0-9]{6}`

Continua na próxima página

Tabela 13.1 – Continuação da página anterior

Campo	**Frontend**	**Backend**
Cidade	Type: Text	Verificar se a cidade existe no estado via API do IBGE
Estado	Menu dropdown (select/option)	Existe na lista de estados do Brasil

13.2 Adicionar/editar pista

Figura 13.5: Tela de adicionar / editar uma pista

As validações para esta rota são:

Tabela 13.2: Adicionar/editar informações pistas

Campo	**Frontend**	**Backend**
Cabeçeira 1	Regex: `^\d{2}[LRC]?$` (dois números mais R, L ou C opcionalmente)	Regex: `^\d{2}[LRC]?$`, os números de cabeceiras opostas tem que somar 18 e caso haja letra uma precisa ser R, a outra L ou ambas C.
Continua na próxima página		

Tabela 13.2 – Continuação da página anterior

Campo	**Frontend**	**Backend**
Cabeçeira 2	O mesmo da cabeceira 1	O mesmo da cabeceira 1
Comprimento	De 100 até 5000 (ambos inclusivos)	De 100 até 5000 (ambos inclusivos)
Largura	De 20 até 80 (ambos inclusivos)	De 20 até 80 (ambos inclusivos)
Tipo do pavimento	Menu dropdown (select/option)	Existe na lista de pavimentos salva do banco

13.3 Adicionar/editar frequências de comunicações

São as frequências que devem ser usadas na comunicação da aeronave possuem uma frequência em MHz e um tipo que pode ser: torre, operações, solo, rampa, ATIS e tráfego.

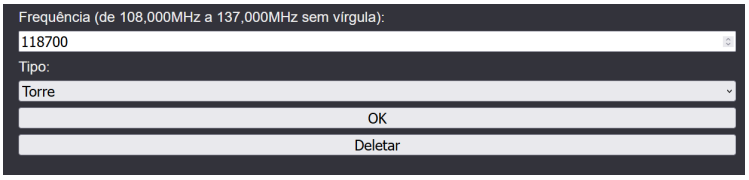


Figura 13.6: Tela de adicionar / editar uma frequência de comunicação

As validações para esta rota são:

Tabela 13.3: Editar comunicações

Campo	**Frontend**	**Backend**
Frequência	Deve ser de 108000 até 137000 (ambos inclusivos) (não deve ter vírgula. Por exemplo, a frequência 108,000MHz é digitada como 108000)	Deve ser de 108000 até 137000 (ambos inclusivos)
Tipo	Menu dropdown (select/option)	Existe na lista de tipos de comunicações

13.4 Adicionar/editar frequências de VOR

São as frequências usadas para navegação, ao sintonizar na frequência de uma antena VOR o avião pode voar diretamente para ela. O nome de um localizador VOR é sempre formado por três letras maíusculas.

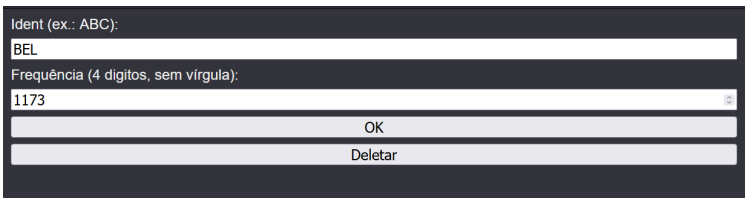


Figura 13.7: Tela de adicionar / editar um VOR

As validações para esta rota são:

Tabela 13.4: Adicionar/editar frequências de VOR

Campo	**Frontend**	**Backend**
Frequência	Deve ser de 1080 até 1180 (ambos inclusivos)	Deve ser de 1080 até 1180 (ambos inclusivos)
Ident	`^[A-Z]{3}$`	`^[A-Z]{3}$`

13.5 Adicionar/editar frequências de ILS

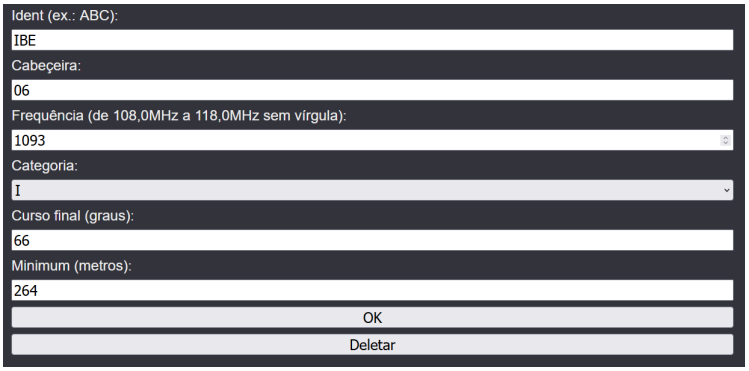


Figura 13.8: Tela de adicionar / editar um ILS

As validações para esta rota são:

Tabela 13.5: Adicionar/editar frequências de ILS

Campo	**Frontend**	**Backend**
Ident	Regex: `^[A-Z]{3}$`	Regex: `^[A-Z]{3}$`
Cabeçeira	Regex: `^\d{2}[LRC]?$`	Regex: `^\d{2}[LRC]?$`
Frequência	De 1080 até 1180 (ambos inclusive)	De 1080 até 1180 (ambos inclusive)
Categoria	Menu dropdown (select/option)	Existe na lista de tipos de ILS
Curso final	Type: Number	Inteiro
Minimum	Type: Number	Inteiro

Um campo existente na tabela de aeródromo é o isPublished. Ele é usado para, no ato de criação, o aeródromo não aparecer vazio para o usuário final. Enquanto este campo está em false apenas usuários logados conseguem ver. Ao serem adicionadas as pistas, comunicações, VOR e ILS o botão de publicar pode ser pressionado fazendo esta variável ir para true, mostrando o aeródromo para todos que acessarem. Quando um aeródromo é adicionado seu METAR e TAF não são atualizados automaticamente. É necessário que os trabalhos agendados pelo APS (Advanced Python Scheduler) atualizem estas informações. A função de atualização de gráficos (update_images()) também é

executada. Mesmo que, no início, exista apenas um METAR, o gráfico pode ser gerado sem problemas. Mas quando adiciono um aeródromo costumo esperar um tempo até publicá-lo, para termos vários METARs salvos deixando o gráfico com vários pontos no tempo.

14
Disponibilidade do METAR

Durante as enchentes de grandes proporções no Rio Grande do Sul ocorridas em maio de 2024, grande parte da capital Porto Alegre ficou alagada. O Aeroporto Internacional Salgado Filho fechou. Já que grande parte do pátio e da pista ficaram debaixo de água estavam impossibilitados pousos e decolagens. Algumas aeronaves particulares foram danificadas pela água.

Com o aeroporto fechado, o METAR não era mais emitido. Porém, a API do Aviation Weather não respondia com um código HTTP de erro ou de não encontrado, mas sim com código de sucesso (200) e uma string vazia como resposta. Com isso, a decodificação do METAR falhava. Para este aeroporto, a seguinte tela aparecia:

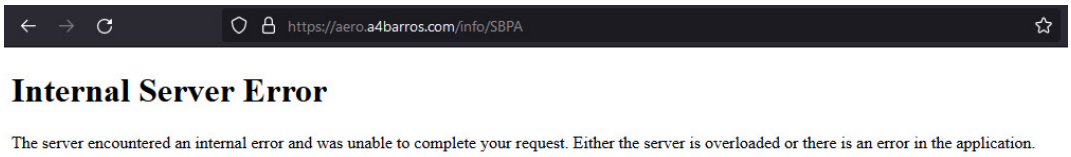


Figura 14.1: Página com o erro

Quando o módulo de decodificação do METAR tentava buscar o dia e hora na string do METAR, esta não existia. Então, quando tentava buscar o índice nos grupos de captura, este não existia. Então a exceção IndexError era levantada.

Já que todo METAR deve possuir o item com o dia e a hora zulu, caso este não exista, a função de decodificação do METAR retorna o seguinte valor:

```
[(" ", "METAR indisponível.")]
```

Este valor consegue ser interpretado pelo template.

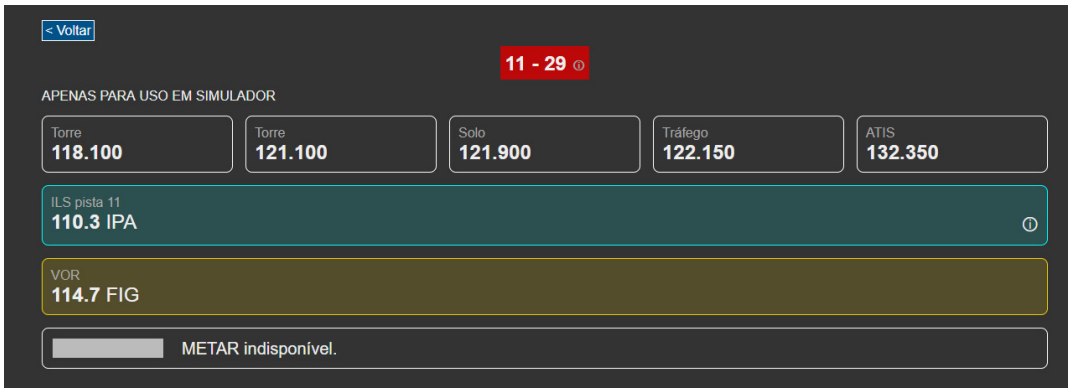


Figura 14.2: Página com o fallback

15
Observabilidade

Com o aumento da complexidade do projeto, é importante saber quando algum usuário teve problemas ao tentar acessar alguma rota que não funcionou corretamente. Também é interessante ver quais URLs robôs estão tentando acessar para fechar uma possível falha de segurança.

O NGINX possui um arquivo de logs localizado em `/var/log/nginx/access.log`. No entanto, com o aumento do número de acessos, esse arquivo se torna cada vez mais difícil de interpretar.

A solução é utilizar uma ferramenta que processe esse arquivo de log e exiba os dados em uma interface gráfica. Para isso, usou-se o GoAccess [20]. Ele é um programa para Linux que, ao receber o arquivo de log, gera um arquivo HTML como saída.

Código 6: Comando para executar o GoAccess

```
1 goaccess /var/log/nginx/nginx-access.log -o /var/log/goaccess/report
      .html --log-format=COMBINED
```

Figura 15.1: Dashboard do GoAccess

Observe que o arquivo de log que estou acessando é `/var/log/nginx/nginx-access.log`. Isso foi necessário porque a imagem *Docker* do NGINX que uso, por padrão, cria um link simbólico de `/var/log/nginx/access.log` para `/dev/stdout`, o que impossibilitava o acesso ao arquivo. Para contornar esse problema, precisou-se configurar o arquivo de configuração do NGINX para que os logs de acesso fossem gravados também em outro local.

Também é importante destacar que o GoAccess não serve o arquivo HTML gerado automaticamente, apenas o cria. A solução foi utilizar o próprio NGINX para servir esse arquivo na rota `a4barros.com/private/report`. Essa URL está protegida por autenticação HTTP básica. O arquivo HTML gerado é estático e não é atualizado automaticamente. Existe, contudo, a opção de usar o parâmetro `-real-time-html`, que abre um websocket na porta 7890 e adiciona um código JavaScript à página HTML para atualizações em tempo real. No entanto, como não deseja-se manter essa porta aberta, optou-se por criar um script shell que recria essa página a cada 3 minutos.

Código 7: Dockerfile do container do GoAccess

```
1  FROM ubuntu:latest
2
3  RUN apt-get update && \
4      apt-get install -y goaccess && \
5      rm -rf /var/lib/apt/lists/*
6
7  RUN echo '#!/bin/bash\n\
8  while true; do\n\
9    goaccess /var/log/nginx/nginx-access.log -o /var/log/goaccess/
         report.html --log-format=COMBINED\n\
10   sleep 180\n\
11 done' > /usr/local/bin/run-goaccess.sh
12
13 RUN chmod +x /usr/local/bin/run-goaccess.sh
14
15 CMD ["/usr/local/bin/run-goaccess.sh"]
```

Em média, o GoAccess consegue processar 101.245,3 linhas de log por segundo [21], então executá-lo a cada três minutos para a quantidade de acessos médios (aproximadamente 4640 requisições de 48 IPs únicos por dia) não é problema.

16
Teste de Carga

Para avaliar a confiabilidade do projeto em cenários extremos com muitos acessos simultâneos, utilizou-se a framework de testes Locust. A vantagem dessa ferramenta é que ela é uma biblioteca Python, permitindo que, com código Python, seja possível escrever as chamadas "tasks"[22], que são as requisições a serem feitas. Cada task é uma função com um decorator Python específico.

Para aumentar a confiança nos testes, as rotas são chamadas com ICAOs aleatórios.

Para garantir que os testes consultassem exclusivamente o meu servidor, coloquei o Cloudflare no modo de desenvolvedor, o que desativa a função de cache.

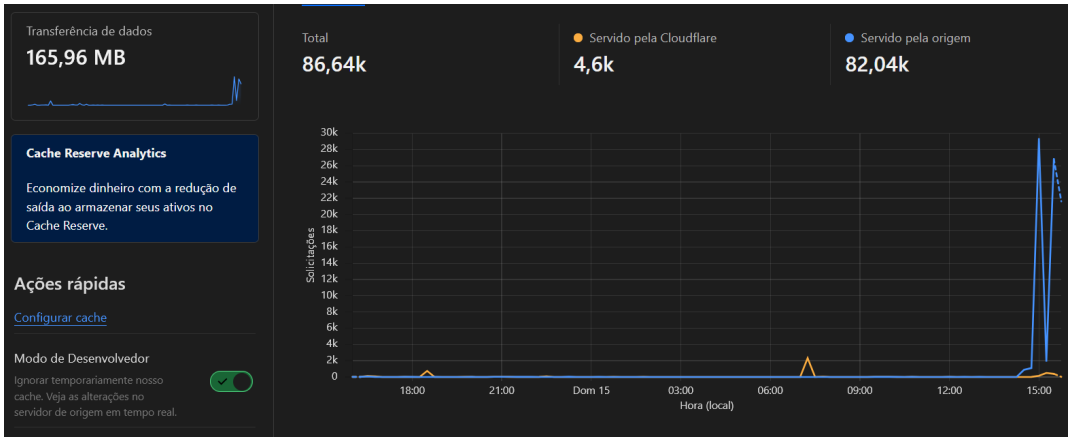


Figura 16.1: Cache no Cloudflare desativado. Note que a linha azul está bem acima da linha laranja, mostrando que meu servidor de origem está servindo as requisições

O maior gargalo deste projeto era o acesso ao banco de dados. Ao testar com 500 usuários simultâneos, no pior caso, o tempo de resposta ultrapassava 15 segundos, o que é totalmente inaceitável para um código de produção.

Uma ideia é utilizar um banco de dados em memória, como o Redis, para fazer cache das respostas. Fiz um decorator Python chamado cache_it() que verifica se já não existe uma reposta no Redis.

O código do decorator é o seguinte:

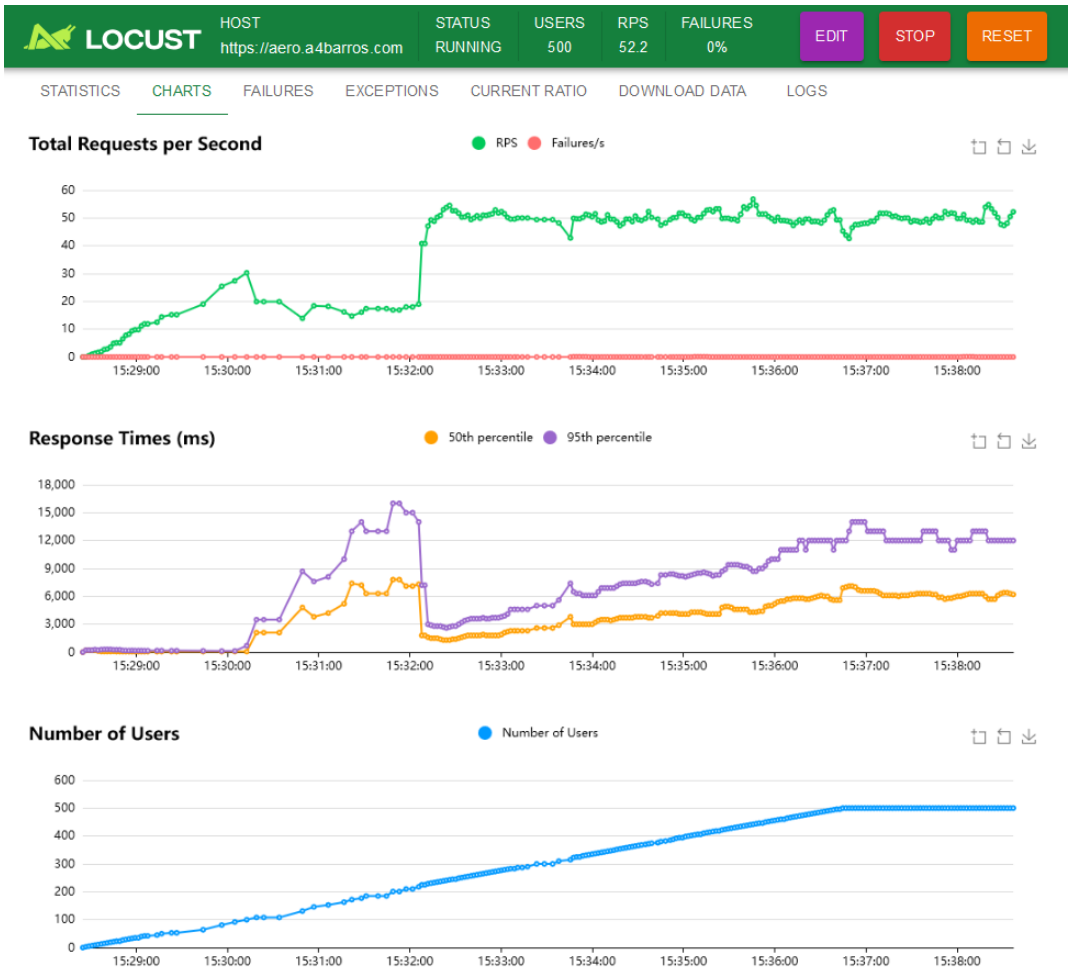


Figura 16.2: Teste de carga no Locust

Código 8: Código do decorator de cache

```
def cache_it(func):
    async def wrapper(*args, **kargs):
        try:
            icao = kargs.get("icao") or args[0]
        except IndexError:
            icao = "default"
        key = f'{icao}:{func.__name__}'
        print(key, end=" :: ")
        cached = json.loads(await client.get(key) or "null",
            object_hook=datetime_parser)
        if not cached:
            cached = await func(*args, **kargs)
            await client.set(key, json.dumps(cached, default=
                datetime_serializer), ex=3600)
            print("miss")
        else:
            print("hit")
        return cached
    return wrapper
```

Ele é aplicado nas funções:

- async def get_info(icao):
- async def get_metar(icao: str) -> tuple[str, str]:
- async def get_taf(icao: str) -> tuple[str, str]:

Note que são funções que apenas recebem icao como parâmetro, então fazendo a chave do Redis ser `<ICAO>:<NOME_DA_FUNCAO>` conseguimos a unicidade. Cada chave é configurada apenas expirar em uma hora, mas normalmente, antes disso a função trash_it(icao) é chamada.

Ela remove todos os registros com chave iniciando no icao informado no parâmetro. Todas as funções que atualizam o metar e o taf chamam a trash_it(). Já que estas funções atualizam informações para todos os aeródromos, todo o cache é invalidado. Todas as funções de administração também invalidam o cache, mas apenas para o aeródromo alterado. Se um ILS de SBCT, por exemplo, é alterado, todo o cache para SBCT é invalidado.

Contudo, o teste no Locust revelou apenas uma pequena melhora principalmente no início.

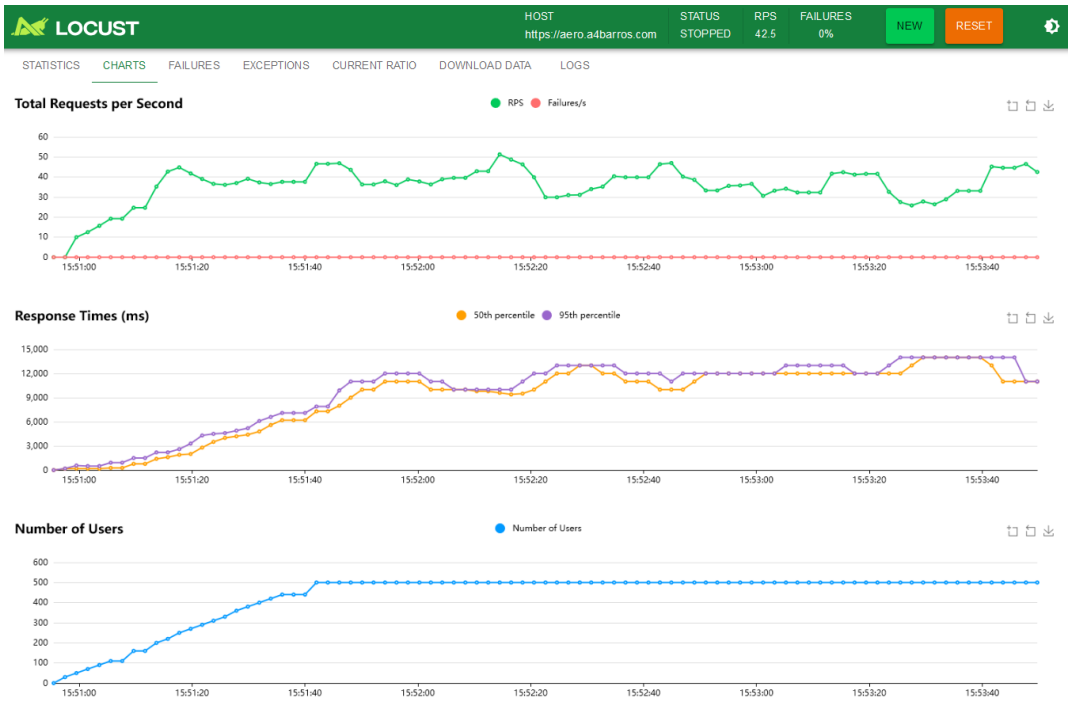


Figura 16.3: Teste de carga no Locust

Devido a isto, pensou-se em uma forma de fazer cache da resposta completa. Existem bibliotecas que se integram com o *FastAPI* como middleware, e, ao adicionar um decorador antes da definição da rota, é possível configurar

um cache com o tempo definido pelo programador. Contudo, essas bibliotecas não estão sendo mantidas, o que traz o risco de pararem de funcionar sem aviso.

Devido a este risco, optou-se por uma solução mais declarativa, em vez de implementar algo próprio. Também queria-se ter que não depender de um servidor Python, que, por usar uma linguagem interpretada, será mais lento que algo executado nativamente.

Como já existe o NGINX funcionando como proxy reverso, utilizou-se o próprio NGINX para realizar o cache das respostas. Configurou-se o cache com um tempo de um minuto, ou seja, no pior caso, a informação ficará desatualizada por até 1 minuto. Como o METAR é atualizado a cada hora, no pior cenário, alguém acessou a página com o METAR às 17:59:59, e essa informação seria armazenada no cache. Às 18:00:00 sairia um novo METAR, e apenas às 18:00:59 a informação seria atualizada, o que é aceitável, considerando o ganho de escala.

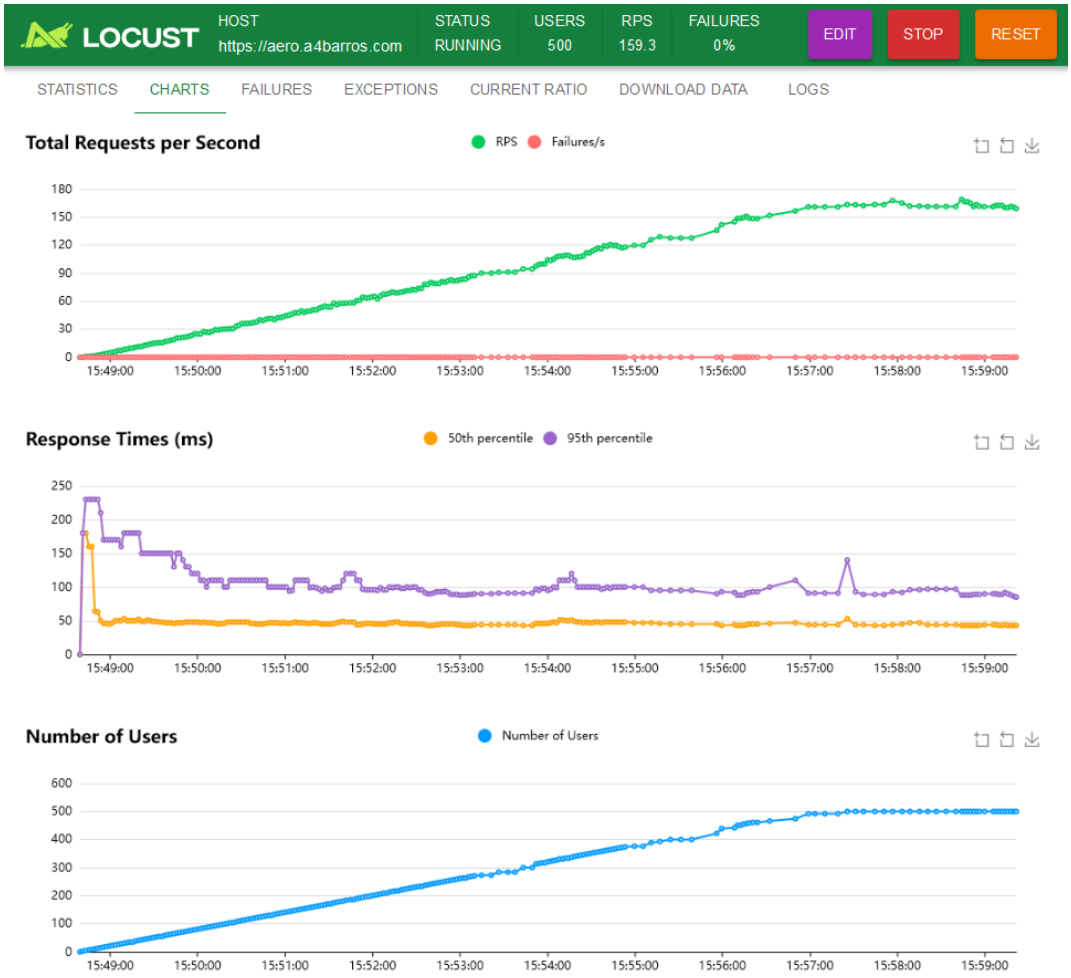


Figura 16.4: Teste de carga no Locust com cache do NGINX

17
Conclusão

Como conclusão, pode-se dizer que foi um projeto muito interessante de se fazer já que o autor tem interesse tanto da área de programação para web como a de aviação. Trata-se de um primeiro projeto open-source grande e os utilizadores podem melhorar o produto mesmo sendo leigos em programação já que na parte inferior da página principal há um e-mail de contato para que me sejam enviados elogios, críticas e sugestões.

Foi explorado, também, um pouco do frontend, área que que o autor não tinha tanta familiaridade.

Acredito que o objetivo do projeto foi alcançado. O Aero busca juntar as funcionalidades comumente usadas na simulação de voo em uma plataforma acessível e amigável, trazendo um maior nível de realismo na simulação de voo, aspecto desejado por quem leva a simulação de voo "a sério".

Vários assuntos que não conhecia aprendi com este projeto incluindo:

- Fazer uma página de observabilidade
- Fazer um teste de carga
- Usar o *Docker* Secrets para armazenar senhas ao invés do ".env"
- Usar cache direto no NGINX para evitar que o site saia do ar em períodos de muitos acessos
- Armazenar hash de senha com salt o padrão bcrypt
- Implementar autenticação de dois fatores TOTP
- Traduzir mensagens de erros do Pydantic para o Português para que possam ser mostrados para o usuário.

Acredito fortemente que este foi um projeto muito proveitoso devido aos conhecimentos de programação web que adquiridos que serão de grande valia no meu trabalho que é o desenvolvimento do backend de um PWA. Além disso, já que o site está público servirá como uma vitrine para outras potenciais oportunidades de carreira.

17.1 Código completo

- O código fonte na íntegra do Aero pode ser visto acessando o seguinte link: `https://github.com/antenor-z/aero`
- O código LaTeX que gerou este documento está disponível em: `https://github.com/antenor-z/aero-relatorio`
- Por fim, o código da apresentação de slides feita com o Beamer está disponível em: `https://github.com/antenor-z/presentation`

18
Referências bibliográficas

1 Jack Stewart. *The Surprisingly Simple iPad Apps Pilots Use to Make Your Flight Better*. 2018. Disponível em: <https://www.wired.com/story/pilot-ipad-apps/>. Acesso em: 15 outubro 2024.

2 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.

3 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.

4 PRADZ. *Performance Calculation*. 2024. Disponível em: <http://perfcalc.pradz.de/index.php>. Acesso em: 25 abril 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 EBIANCH. *Bloqueando um VOR*. 2021. Disponível em: <https://www.youtube.com/watch?v=fwdXY3l2lW4>. Acesso em: 01 novembro 2024.

7 EBIANCH. *Treinamento ILS em Simulador*. 2020. Disponível em: <https://www.youtube.com/watch?v=UHkSsp1iKa4>. Acesso em: 01 novembro 2024.

8 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.

9 Sebastián Ramírez. *Run a Server Manually*. 2024. Disponível em: <https://fastapi.tiangolo.com/deployment/manually/>. Acesso em: 20 agosto 2024.

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

11 ORGANIZATION, W. M. *Aerodrome reports and forecasts*. 2022. Disponível em: <https://library.wmo.int/records/item/30224-aerodrome-reports-and-forecasts>. Acesso em: 30 setembro 2024.

12 National Weather Service. *Information Reporting*. 2024. Disponível em: <https://www.weather.gov/asos/InformationReporting.html>. Acesso em: 17 abril 2024.

13 KOCH, S. *METAR / SPECI*. 2024. Disponível em: <https://sites.google.com/site/invacivil/meteorologia/metar>. Acesso em: 08 abril 2024.

14 National Weather Service. *Aviation Weather Center*. 2024. Disponível em: <https://aviationweather.gov/api/data/metar?ids=SBFZ>. Acesso em: 17 abril 2024.

15 Airport Codes. *IATA codes*. 2024. Disponível em: <https://airportcodes.io/en/iata-codes/>. Acesso em: 16 abril 2024.

16 Airport Codes. *ICAO codes*. 2024. Disponível em: <https://airportcodes.io/en/icao-codes/>. Acesso em: 16 abril 2024.

17 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.

18 College of DuPage Weather Lab. *METAR HELP*. 2024. Disponível em: <https://weather.cod.edu/notes/metar.html>. Acesso em: 17 abril 2024.

19 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.

20 Gerardo O. *goaccess*. 2024. Disponível em: <https://goaccess.io/get-started>. Acesso em: 15 setembro 2024.

21 Gerardo O. *goaccess*. 2024. Disponível em: <https://goaccess.io/faq>. Acesso em: 15 setembro 2024.

22 HEYMAN, J. *What is Locust?* 2024. Disponível em: <https://docs.locust.io/en/stable/what-is-locust.html>. Acesso em: 30 setembro 2024.