Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

API REST na Plataforma A-CDM

Helena Schubert I. L. Silva 10/0012311

Projeto Final apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Li Weigang

Brasília
2019

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

API REST na Plataforma A-CDM

Helena Schubert I. L. Silva 10/0012311

Projeto Final apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Li Weigang (Orientador)
CIC/UnB

Prof. José Edil Guimarães de Medeiros
Coordenador do Curso de Engenharia da Computação

Brasília, 2 de julho de 2019

Dedicatória

Dedico esse trabalho e minha resiliência primeiramente a Deus e ao meu anjo da guarda. Só Deus sabe o que Ele e meu anjo guardião fazem por mim.

Dedico, em segundo lugar, à minha mãe e ao meu pai. Minha mãe, por ser meu exemplo de coragem, persistência, esforço e amor incondicional. Sem ela, eu não seria. A força que ela me ofereceu em meus momentos difíceis faz parte de quem eu sou hoje, enquanto que o carinho que ela sempre me deu é simplesmente meu maior patrimônio. Meu pai que, do seu jeito, sempre fez tudo por mim e me ensinou a ter generosidade e que continua sempre me apoiando e me ajudando a crescer. Nada do que eu fizer estará a altura do que vocês fizeram e fazem por mim, mesmo assim dedico este trabalho a vocês.

Dedico ainda a todos meus avós. As lembranças mais doces que eu possuo foram na casa dos meus avós, pais da minha mãe, Casimiro, meu finado avô, e Justina, minha avó. Essas lembranças serão meus tesouros eternos. E minha vó Santa, que sempre me dá tanto carinho e meu finado avô Elesbão, um dos maiores orgulhos que eu tenho. Dedico ainda à minha finada tia Irene, que cuidou de mim e de meu pai.

Dedico também às minhas amigas, que são algumas das melhores pessoas do mundo e nem sei se mereço tê-las como amigas: Luana e Isadora. Obrigada, Luana, por ter me ajudado com a revisão de português deste trabalho e sempre me escutar quando eu preciso! Obrigada, Isadora, por estar há tantos anos na minha vida, crescendo comigo!

Dedico, por último, a todos os colegas de curso e de área da UnB. Os que vêm e vão e os que ficam. Obrigada a todos que tornaram meus dias mais leves!

Agradecimentos

Agradeço a todos os professores e aos fundadores da minha instituição favorita no mundo, a Universidade de Brasília. Obrigada por fornecerem tantos desafios e sentido aos meus caminhos.

Agradeço, em particular, à equipe do Translab e ao meu orientador, Prof. Li. Tive muitas experiências enriquecedores graças a eles.

Gostaria de agradecer ainda a todos os que já se esforçaram para investigar, criar, descobrir e se surpreender. Principalmente, aqueles que formaram os alicerces das conquistas científicas e de engenharia da humanidade no passado e que mantiveram sua fé no fato de que o Conhecimento transforma o mundo.

Agradeço também a todas as mulheres acadêmicas na área de engenharia e ciência da computação, mesmo as que não foram minhas professoras, mas que me serviram de referência e exemplo. O mesmo vale a todos os acadêmicos brasileiros, cujo objetivo de chegar mais longe costuma ir contra a correnteza. Todos são inspiradores em suas próprias maneiras.

Resumo

Neste trabalho, uma Interface de Programação de Aplicação REST é desenhada e especificada para que um dos sistemas da plataforma do A-CDM, o A-CDM Information Sharing, possa ser futuramente usado por outras aplicações. Os pontos principais do estilo arquitetural REST e de práticas de design são explicitados e investigados para tanto. A importância do trabalho está em provar o uso de tal tecnologia para um domínio que ainda está em desenvolvimento, que é o uso de Tecnologia da Informação nos processos operativos aeroportuários que integrem diferentes parceiros em torno de uma só base de dados. No futuro, pode-se usar essa tecnologia para satisfazer todos os requisitos possíveis da plataforma A-CDM.

Palavras-chave: API, REST, A-CDM, information sharing, engenharia de software

Abstract

A *REST Application Program Interface was designed and especified in order to be used in the future by a system of the A-CDM plataform, namely the A-CDM Information Sharing. For this reason, the main points of the REST architetural style and the design process were investigaded and made explicit. The relevance of this work is its proof that the REST tecnology is a valid solution for the Information Tecnology in airport operational process, an area still developing and that aims that all the airport's partners work together, to have better decisions. In the future, one may use a REST API to satisfy all the needs of the A-CDM plataform.*

Keywords: API, REST, A-CDM, information sharing, software engineering

Sumário

1 Introdução **1**
1.1 Motivação . . . 2
1.2 Objetivo . . . 2
1.3 Metodologia . . . 3
1.4 Organização do trabalho . . . 3

2 API REST **5**
2.1 Nomeclatura - API REST e API RESTful . . . 5
2.2 Arquitetura Orientada a Serviço . . . 5
2.3 Definição de API . . . 6
2.4 REST versus SOAP . . . 7
2.5 Definição de REST . . . 7
2.5.1 Condições de Fielding . . . 8
2.6 Componentes de uma API RESTful . . . 9
2.6.1 HTTP . . . 9
2.6.2 URI . . . 10
2.6.3 *Hypermedia* . . . 11
2.6.4 Tipo de Relação de Link de Perfil . . . 12
2.6.5 Documentos de Representação . . . 12
2.7 Design de *frontend* . . . 13
2.7.1 Experiência do Desenvolvedor e Usabilidade . . . 13
2.7.2 Design de URI . . . 16
2.7.3 Procedimentos de design . . . 17
2.8 Decisões de Design de Backend . . . 18
2.8.1 *Backend* . . . 18
2.8.2 Transformações . . . 18
2.9 Requisitos não funcionais . . . 19
2.10 Processos de software . . . 19

3 A-CDM e SWIM **20**
3.1 A-CDM . . . 20
3.1.1 Parceiros A-CDM . . . 21
3.1.2 Conceitos chaves . . . 21
3.1.3 Objetivos dos parceiros . . . 22
3.1.4 ACIS: A-CDM *Information Sharing* - Compartilhamento de Informações 22
3.2 SWIM . . . 23
3.2.1 Objetivos . . . 24
3.2.2 Definição . . . 24
3.2.3 SWIM-Registry . . . 25
3.2.4 SOA e SWIM . . . 25

4 Requisitos e Projeto **27**
4.1 Decisões de Projeto . . . 27
4.2 Conceitos importados do A-CDM . . . 28
4.3 Requisitos funcionais . . . 28
4.4 Requisitos de interface . . . 29
4.5 Decisões de design . . . 29
4.5.1 Definições para a prova de conceito . . . 31

5 Implementação e Resultados **33**
5.1 Aplicações . . . 33
5.2 Módulos . . . 34
5.3 Autenticação e autorização . . . 36
5.4 Validação . . . 36
5.5 Provas de conceito . . . 36
5.5.1 Raiz da API . . . 37
5.5.2 Requisitando coleções . . . 37
5.5.3 *Hypermedia* nas representações . . . 38
5.5.4 Filtragem de resultados . . . 39
5.5.5 Enviando uma nova instância . . . 39
5.5.6 Deletando uma instância existente . . . 40
5.6 Documentação . . . 40
5.7 Repositório do trabalho . . . 40

6 Conclusão **41**
6.1 Experiência de desenvolvimento da API REST . . . 41
6.2 Considerações finais . . . 41

6.3 Trabalhos futuros 42

Referências **43**

Lista de Figuras

2.1 Esquema da relação entre cliente e servidor com interface REST. 8

3.1 Objetivos A-CDM. Figura retirada do The Manual Airport CDM Implementation cápitulo 2, página 5 [1]. 23
3.2 Funções do Compartilhamento de Informações. Figura retirada do The Manual Airport CDM Implementation cápitulo 3, página 2 [1]. 24
3.3 Estruturas que compõem o SWIM. Figura retirada do MANUAL ON SYSTEM WIDE INFORMATION MANAGEMENT (SWIM) CONCEPT, capítulo 2, página 2 [2]. 25

4.1 Descritores da API REST do ACIS. 30
4.2 Escopo da prova de conceito contendo os descritores dos recursos. 30

5.1 Esquema da disposição dos módulos segundo o *framework* adotado. 35

Lista de Abreviaturas e Siglas

A-CDM Tomada de Decisão Colaborativa de Aeroporto (*Airport-Collaborative Decision Making*).

ACIS Compartilhamento de Informações de A-CDM (*A-CDM Information Sharing*).

API Interface de Programa de Aplicação (*Application Program Interface*).

DECEA Departamento de Controle do Espaço Aéreo.

HATEOAS Hipermídia como Mecanismo de Estado de Aplicação (*Hypermedia as the Engine of Application State*).

HTTP Protocolo de Transferência de Hipertexto (*HyperText Transfer Protocol*).

JSON Notação de Objeto *JavaScript* (*JavaScript Object Notation*).

REST Transferência de Estado Representacional (*Representational State Transfer*).

SOA Arquitetura Orientada a Serviço (*Service-Oriented Architecture*).

SOAP Protocolo de Acesso a Objeto Simples (*Simple Object Access Protocol*).

SWIM Gerenciamento de Informações Vastas de Sistema (*System Wide Information Management*).

TCP Protocolo de Transmissão de Controle (*Transmission Control Protocol*).

UDP Protocolo de Datagrama de Usuário (*User Datagram Protocol*).

URI Identificador de Recurso Uniforme (*Uniform Resource Identifier*).

URL Localizador Uniforme de Recursos (*Uniform Resource Locator*).

URN Nome de Recurso Uniforme (*Uniform Resource Name*).

XML Linguagem de Marcação Extensível (*eXtensible Markup Language*).

Capítulo 1

Introdução

Desde seu advento, a Internet vem unindo produtores e consumidores de informação e, em diversas ocasiões, um mesmo agente é capaz de consumir e produzir dados. Novos serviços são propostos e, com isso, são criadas necessidades inéditas. Para que a comunicação continue possível, padrões são estipulados, uniformizando o intercâmbio de dados e serviços. A tecnologia Transferência de Estado Representacional (*Representational State Transfer*) (REST), propõe um estilo de desenvolvimento de softwares adaptado para um meio interconectado por redes de computadores, sanando uma dessas necessidades que surgiram com a Internet.

Este trabalho apresenta uma solução que se estabelece sob o estilo arquitetural REST. Uma Interface de Programa de Aplicação (*Application Program Interface*) (API) é projetada seguindo as condições e regras que a tornam *RESTful*, a fim de solucionar um problema acerca de um conceito do universo aeroportuário e de transporte aéreo: o A-CDM (Airport-Collaborative Decision Making).

O contexto do A-CDM engloba os aeroportos, e trata-se de um conceito operacional. Da mesma forma, há o conceito de Gerenciamento de Informações Vastas de Sistema (*System Wide Information Management*) (SWIM), que é um esforço para padronizar e facilitar a comunicação dos agentes envolvidos com atividades de gerenciamento de tráfego aéreo. A organização intergovernamental europeia Eurocontrol e o órgão governamental estadunidense FAA têm realizado pesquisas na área e desenvolvido esses conceitos. O A-CDM já está efetivamente implantado em 28 aeroportos europeus [3].

Com objetivo de automatizar processos e permitir a comunicação de agentes ligados a um determinado aeroporto, o Tomada de Decisão Colaborativa de Aeroporto (*Airport-Collaborative Decision Making*) (A-CDM) deve possuir uma plataforma de cooperação entre diversos sistemas, que incluem o Compartilhamento de Informações de A-CDM (*A-CDM Information Sharing*) (ACIS). Tal compilado de softwares tem uma vasta gama de objetivos, mas a meta do sistema ACIS, em particular, é reunir e distribuir as informações

dos parceiros A-CDM e demais sistemas.

1.1 Motivação

Os últimos anos têm mostrado que há um crescimento contínuo no tráfego aéreo brasileiro, acompanhando a tendência mundial. Segundo projeções disponibilizadas pelo Ministério de Transportes de 2017, a demanda de passageiros no Brasil deve aumentar em 99% até o ano de 2037, em relação a 2017. No mesmo período, a movimentação de passageiros, isto é, o embarque e desembarque deve aumentar em 67%, quase o mesmo que a quantidade de carga transportada por vias aéreas em toneladas, que tem projeção de crescer 65% durante esse período. Essas projeções consideram que a rede de aeroportos atual se mantenha a mesma [4].

Evidentemente, quanto mais intenso é o fluxo de passageiros em aeroportos, maior é a necessidade de organizar os processos operacionais e de tráfego de forma a suprir as necessidades de segurança e eficiência, além de seguir as normas das autoridades. Para amparar os desafios do crescimento, nos últimos anos têm surgido conceitos como A-CDM e SWIM, que procuram otimizar as atividades envolvidas com o tráfego aéreo.

No Brasil, o Departamento de Controle do Espaço Aéreo (DECEA) junto à administração do aeroporto de Guarulhos, GRU Airport, firmaram um acordo para a implementação do A-CDM no dito aeródromo. Sendo o mais movimentado aeroporto do Brasil, com uma média de 744 voos diários, o aeroporto de Guarulhos foi definido como o primeiro do país a ter o A-CDM implementado, seguindo o padrão estudado e desenvolvido pela Eurocontrol, para que haja mais pontualidade e otimização nas operações [5].

Para que mais aeroportos no Brasil possam aderir aos conceitos de A-CDM e SWIM, e desse modo aumentem a eficiência dos processos operativos aeroportuário e de tráfego aéreo, sugere-se a criação de ferramentas cruciais para a adoção desses conceitos.

Após a criação do SWIM Registry por um dos laboratórios do departamento de `UnB-CIC` da Universidade de Brasília, o Translab, seria conveniente que fosse desenvolvido uma interface para a utilização do sistema ACIS, e que pode vir a ser um dos serviços disponíveis no sistema supracitado. Dessa maneira, haveria aumento do portfólio de serviços disponíveis no SWIM Registry e uma etapa para a primeira ferramenta para a implementação do A-CDM já estaria pronta.

1.2 Objetivo

O objeto desse trabalho é uma API REST para o sistema ACIS. O objetivo é provar que o estilo arquitetural REST é compatível com as necessidades e requisitos do sistema em

questão. Outro objetivo adjacente, é propor a API com o intuito de vir estabelecer mais um serviço que possa ser ofertado pelo SWIM *Registry* a quem se interessar.

Os objetivos finais podem ser divididos em subobjetivos, que seriam:

1. Pesquisar os requisitos do sistema ACIS;
2. Especificar a API web para esse sistema;
3. Implementar a API;
4. Validar com um caso de uso para cada funcionalidade.

1.3 Metodologia

Para o entendimento e o desenvolvimento do projeto descrito neste trabalho, a seguinte metodologia foi empregada:

1. **Revisão de Literatura**: artigos e outras obras sobre a temática do trabalho e auxiliares são pesquisados e estudados para maior elucidação da técnica, das vantagens e desvantagens, entre outros tópicos;
2. **Levantamento de requisitos:** junto à parte dos requisitos que já foi devidamente documentada em manuais, como o manual do A-CDM, há uma análise dos requisitos para especificar a API;
3. **Definição de arquitetura:** a partir dos requisitos e da literatura, a arquitetura do projeto é definida, especificando elementos técnicos como módulos de software e suas relações, escolha da linguagem de programação e eventuais *frameworks*;
4. **Implementação:** a API é desenvolvida seguindo a arquitetura anteriormente citada;
5. **Validação:** cenários de testes são feitos para a validação do projeto, onde cada funcionalidade é apresentada e verificada;
6. **Conclusão:** os resultados do projeto são então analisados e possíveis trabalhos futuros são descritos.

1.4 Organização do trabalho

Esta monografia está organizada do seguinte modo:

- O capítulo 1, que é este que contém a introdução, a motivação, os objetivos e a estrutura do trabalho;

- O capítulo 2 destina-se à revisão bibliográfica técnica, apresentando os conceitos relevantes à API;
- O capítulo 3 aprofunda-se nos conceitos de A-CDM e SWIM, detalhando seus contexto e o objetivo do projeto;
- O capítulo 4 trata dos requisitos e do projeto em alto nível do *design* da API para o ACIS;
- O capítulo 5 apresenta a arquitetura usada na implementação, assim como os resultados escolhidos para validar os conceitos;
- Por último, o capítulo 6 contém as conclusões e prováveis trabalhos futuros.

Capítulo 2

API REST

Nesse capítulo encontra-se a fundamentação teórica necessária ao desenvolvimento da solução de uma Interface de Programa de Aplicação (*Application Program Interface*) (API) para o sistema ACIS. Explicações sobre o funcionamento do estilo arquitetural são providas, assim como a base para futuras decisões de projeto.

Para que se possa desenvolver uma API REST, é necessário entender seu funcionamento, suas bases e como projetar um *design* para essa. As bases históricas desse estilo arquitetural foram exploradas, assim como as tecnologias que antecederam e possibilitaram o conceito REST. Também é preciso saber quais são as técnicas para se desenvolver uma API REST, e por isso também foram explorados tópicos sobre *design*.

2.1 Nomeclatura - API REST e API RESTful

É importante notar que o termo "API RESTful"está tão correto quanto o termo "API REST". Enquanto o primeiro, a palavra "RESTful"é um adjetivo que modifica o termo "API", no segundo, "REST"assume um papel de substantivo assim como "API". Ambos os nomes são encontrados na literatura e são gramaticalmente corretos, apesar de que em inglês, a ordem deve ser invertida, deixando o termo "API"aparecer depois do nome ou substantivo modificador.

2.2 Arquitetura Orientada a Serviço

Com o intuito de reusar uma enorme sorte de ferramentas já existentes, às vezes de diferentes idades e provedores, a indústria de Tecnologia da Informação (TI) desenvolveu soluções que fornecessem meios para a cooperação entre estas ferramentas. Um dos conceitos que emergiram foi o de "serviço". O serviço pode ser visto como determinado código em uma caixa-preta, disponível por meio de uma interface. Junto do serviço, há o

provedor e o cliente desse, um localizador do provedor e possivelmente um intermediário [6][7].

Com a Arquitetura Orientada a Serviço (*Service-Oriented Architecture*) (SOA), torna-se possível utilizar recursos existentes que provêm serviços para terceiros, em vez de se criar um novo artefato desde o início. Também é esperado que haja facilidade em integrar diferentes recursos, sendo esta facilidade um dos requisito da SOA. Tal aspecto permite que outros serviços que usam a SOA como base sejam desenvolvido com rapidez.

É notável a utilização da SOA em ambientes web, sendo esse heterogêneo e distribuído. Algumas tecnologias foram desenvolvidas para tanto, como o Linguagem de Marcação Extensível (*eXtensible Markup Language*) (XML), Protocolo de Acesso a Objeto Simples (*Simple Object Access Protocol*) (SOAP) e REST[7].

Outra característica da SOA é sua relativa independência dos serviços dos softwares. Mesmo o software da lógica interna do serviço prestado deve ser fracamente acoplado, possibilitando que, em caso de mudanças internas, pouca ou nenhuma mudança do serviço seja requerida [8].

2.3 Definição de API

Interface de Programa de Aplicação (*Application Program Interface*) (API) é uma solução para conectar componentes de sistemas distribuídos com acoplamento fraco entre si, disponibilizando um conjunto de assinaturas que forneçam serviços e/ou dados[9][10]. Com uma API, é possível que um sistema se integre com outro, de tal forma que um sistema cliente pode receber serviços e trocar dados com um sistema servidor.

Para que a API de uma aplicação servidora possa ser usada por aplicações clientes, é primordial que os dados de entrada e saída, as funcionalidades e sua utilização sejam especificados e tenham uma assinatura clara e que evite compreensões errôneas, sendo direta e de fácil entendimento. Além disso, deve haver uma semântica sobre o comportamento do programa [10]. Isto decorre do fato de que um(a) programador(a) da aplicação cliente provavelmente não fez parte da implementação da aplicação servidora, sendo ambas as partes geralmente independentes ou distantes geograficamente e/ou temporalmente.

Uma API também pode ser vista como uma evolução da interoperação entre ferramentas que trocam apenas arquivos entre si, mesmo que sejam arquivos de determinado Formato de Troca Padrão (FTP). Uma API deve fornecer os meios necessários para que sejam desenvolvidos *scripts* robustos e simples, que facilitem o consumo de informação por uma aplicação [11].

Um exemplo de API é quando um sistema usuário requisita determinada informação a um banco de dados, e esse pedido é acionado por uma chamada de API (API *Call*)

que acessa *queries* em SQL previamente implementadas. Assim, o sistema cliente ignora qualquer interno código do sistema servidor, precisando conhecer apenas a API [12].

2.4 REST versus SOAP

Ambas tecnologias, REST e SOAP, são opções para se desenvolver uma API. Em termos gerais, porém, SOAP difere de REST por ser um protocolo padronizado, enquanto REST é um estilo arquitetural. Apesar disso, SOAP pode usar vários protocolos da camada de aplicação, como Protocolo de Transferência de Hipertexto (*HyperText Transfer Protocol*) (HTTP), Protocolo de Transmissão de Controle (*Transmission Control Protocol*) (TCP) ou Protocolo de Datagrama de Usuário (*User Datagram Protocol*) (UDP), enquanto REST é baseado no HTTP. Entretanto, SOAP se restringe ao uso de recursos em Linguagem de Marcação Extensível (*eXtensible Markup Language*) (XML), enquanto é possível utilizar diversos recursos de dados com REST [13].

SOAP fornece dados na forma de serviços, enquanto REST os fornece como recursos. De modo geral, SOAP entrega mais dados, sendo mais pesado do que REST, pois seus dados estão estruturados de forma abrangente e com considerável meta-dados. Em contrapartida, REST costuma granularizar os dados, enviando-os em pacotes menores. Devido a esses fatores, torna-se mais cansativo aos desenvolvedores aprender uma API em SOAP do que em REST.

Por outro lado, SOAP é geralmente usado em empresas, por fornecer um contrato de software formal entre servidor e cliente. SOAP também fornece mais segurança, pois essa está dentro do protocolo, enquanto a segurança do REST se reduz a criptografia do protocolo de aplicação [7][9].

2.5 Definição de REST

REST, termo originário de *Representational State Transfer*, criado por Roy Fielding no ano 2000, pode ser tido como um conjunto de acordos, ou condições, de design focado no desenvolvimento de sistema com hipermídia, podendo também ser referenciado como um estilo arquitetural. Entretanto, é diferente de uma arquitetura, que possui uma conotação mais ampla, ou de um *framework*. Uma API implementada totalmente de acordo com as condições do modelo REST é chamada de API RESTful [9][14][15].

Um sistema RESTful é constituído de diferentes partes, como um servidor, clientes e *proxies*, que funcionam independentemente uns dos outros. Para que possam se comunicar, um protocolo da camada de aplicação é escolhido e usado por todos. Nesse trabalho, tal protocolo é o HTTP, uma vez que o modelo REST é otimizado para esse protocolo.

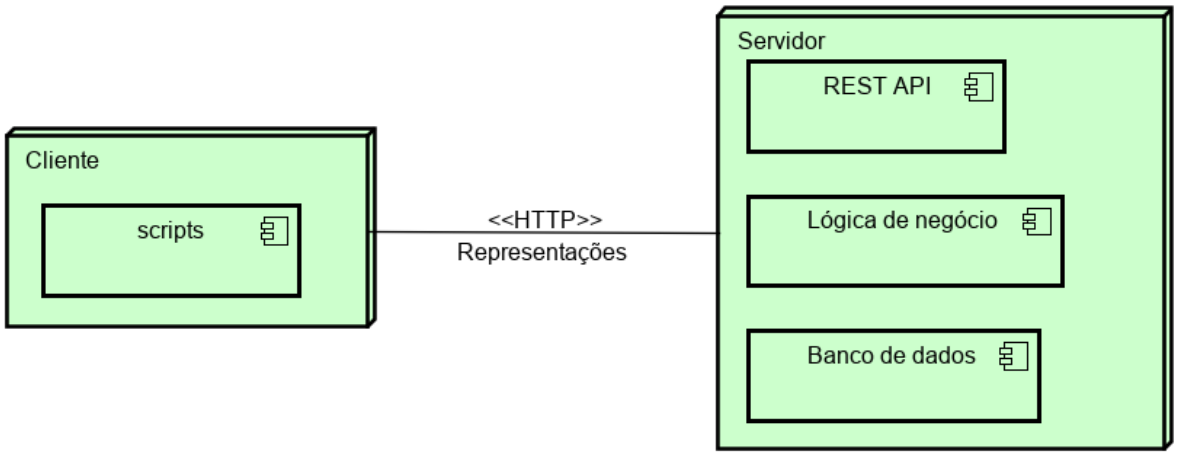

Figura 2.1: Esquema da relação entre cliente e servidor com interface REST.

2.5.1 Condições de Fielding

Fielding propôs o estilo arquitetural REST com base no entendimento de um ambiente distribuído de hipermídia, com a intenção de se servir do contexto, assim como proporcionar os melhores princípios de desenvolvimento. Com isso, as seguintes condições foram identificadas [14]:

- Cliente-servidor - a arquitetura amplamente usada na Web, que possui separação e independência entre as ações de um cliente e de um servidor;
- Sem estado - o servidor não guarda nenhuma informação de estado da sessão de um cliente. Se o cliente precisar, as informações de estado devem ser armazenadas nesse;
- Cache - Um resposta de uma requisição deve ser rotulada como passível ou não de ser guardada em cache, e quando for possível, pode ser assim guardada, eventualmente diminuindo a interação cliente-servidor, substituindo por uma mais rápida;
- Interface uniforme - a interface entre os componentes é uniforme;
- Sistema em camadas - existem componentes que se comunicam entre si, mas que não conhecem o funcionamento interno um do outro e que atuam em uma mesma direção, como cliente-cache-servidor;

- Código por demanda - novos programas ou *scripts* podem ser usados pelo cliente conforme a necessidade.

Além dessas condições anteriormente citadas, condições herdadas do HTTP devem ser usadas no estilo arquitetural REST, como os métodos e códigos de retorno próprios do HTTP. Por já serem adotados como padrão dentro do protocolo HTTP, os métodos e códigos internos desse devem ser também adotados por uma API RESTful, possuindo os mesmos significados tanto no protocolo quanto na API.

2.6 Componentes de uma API RESTful

Alguns componentes são essenciais para o funcionamento de uma API RESTful. O protocolo HTTP está no cerne do conceito de REST, assim como alguns outros conceitos utilizados na web.

2.6.1 HTTP

O Protocolo de Transferência de Hipertexto (*HyperText Transfer Protocol*) (HTTP) é um protocolo de nível de aplicação comumente usado em sistemas distribuídos e colaborativos devido a suas propriedades genéricas e sem estado. Esse protocolo tem sido usado pela *World-Wide Web* desde 1990, e atualmente é usado na versão HTTP/1.1 [16].

Algumas terminologias referentes ao HTTP são:

- Conexão - refere-se ao circuito criado entre dois programas distribuídos, mantido pela camada de transporte;
- Mensagem - a unidade básica de comunicação do HTTP, que é transmitida pela conexão;
- Requisição - mensagem pedida pelo programa cliente ao servidor;
- Resposta - mensagem enviada pelo programa servidor ao cliente como resposta à requisição.

Esquema do HTTP

Para endereçar um recurso do protocolo HTTP, o seguinte esquema deve ser seguido no URL:

http://<hospedeiro>:<porta>/<caminho>?<parte buscada>

onde "hospedeiro"é o nome do domínio; "porta"é a porta lógica que o protocolo deve se comunicar, mas que pode ter um valor *default*; "caminho"são os descritores dos recursos; "parte buscada"é a palavra que deve ser buscada no caminho daquele domínio. Ambos, caminho e parte buscada são opcionais no esquema do HTTP, e podem utilizar os caracteres "/", ";"e "?"como caracteres reservados [17].

Métodos HTTP

Os métodos HTTP, também chamados de verbos HTTP, fazem parte das informações do cabeçalho de uma requisição e são utilizados para informar qual é a intenção dessa requisição HTTP em relação a um determinado recurso. Existem vários métodos HTTP, mas os mais utilizados em um servidor API RESTful são:

- GET - usado para pedir o envio de um determinado recurso;
- DELETE - usado para apagar determinado recurso;
- POST - criar um novo recurso, baseado em um recurso anterior;
- PUT - atualizar um recurso.

Além desses métodos, os seguintes métodos costumam ser usados principalmente por um cliente de uma API:

- HEAD - pede o cabeçalho que viria com a representação pedida, mas sem a representação;
- OPTIONS - questiona a quais métodos o recurso responde.

Também é recomendado o uso do método PATCH, que não faz parte da versão padrão do HTTP, mas sim de um suplemento do RFC 5789. Parecido com o PUT, o PATCH modifica uma representação de um recurso, mas envia apenas a parte modificada.

2.6.2 URI

O Identificador de Recurso Uniforme (*Uniform Resource Identifier*) (URI), e é utilizado para endereçar um recurso, podendo ser um Localizador Uniforme de Recursos (*Uniform Resource Locator*) (URL) ou Nome de Recurso Uniforme (*Uniform Resource Name*) (URN).

Dentro do contexto de URI, dois conceitos são importantes a serem ressaltados:

- Recurso - a entidade endereçada por uma URI, como uma *home page*, por exemplo;

- Representação - o documento que o cliente recebe ao pedir um recurso identificado com um URI. Um recurso pode ter mais de uma representação, onde cada uma possui uma extensão diferente (HTML, JSON etc.).

O URI é uma sequência de caracteres, sendo possível usar letras de "a"a "z", números, sinais "+", "."e -", e os interpretadores de URI devem tratar letras maiúsculas transformando-as em minúsculas. Também é possível e comum representar octetos com "%"seguido de dois números hexadecimais. Outro aspecto importante, é que o sinal "/"é usado para separar a hierarquia dos componentes de uma URI [17].

No domínio das APIs, será preferencialmente usado o conceito de URL, que identifica um recurso existente e disponível em uma rede, e que pode ser diferenciado de outro. Isso é justificável pois, diferentemente do URL, o conceito de URI não garante que haja uma representação endereçado por esse, isto é, o URI representa um endereço, mas que não esteja necessariametne atrelado a um recurso existente. Apesar disso, ambos, URI e URL, seguem as mesmas normas do padrão RFC 3986 [18].

2.6.3 *Hypermedia*

Hypermedia pode ser entendido como a apresentação de um número finito de recursos em dada representação. A partir dessas opções, é possível escolher uma, que por sua vez contém outros hypermedias, traçando um grafo de caminho possíveis, a partir de um nó (representação) inicial.

Nelson propôs o conceito de *hypertext*, no qual um texto pode ser vinculado a outros textos por meio de links (elos) instanciados de diferentes formas, como mapas ou sumários. Segundo Nelson, uma das aplicações do *hypertext* é a aprendizagem, onde um estudante adentra a textos relacionados ou forma um novo relacionamento entre diferentes textos. O mesmo conceito se estende ao *hypermedia*, sendo que o último considera outros meios de comunicação, como imagens e vídeos [19].

A *World Wide Web* recorre ao conceito de *hypermedia* para conectar representações de diferentes formas. Quando um documento HTML, por exemplo, apresenta uma referência a uma URL com o sinal <a>, esse documento contém um link a outro recurso presente na rede, requisitado com método GET do HTTP. Isto é, logicamente, se esta URL está correta.

Hypermedia as the Engine of Application State

Hipermídia como Mecanismo de Estado de Aplicação (*Hypermedia as the Engine of Application State*) (HATEOAS) é tido como uma extensão do estilo arquitetural REST,

segundo M. Biel, em API Design [11], mas é dito como uma das condição de REST, segundo o site especializado e baseado na tese de Fielding [20].

Os links de *hypermedia* são usados para acessar recursos relacionados a um recurso acessado. Isso permite que uma API não necessite de descrições externas, pois os próprios recursos indicam onde achar demais recursos por meio de seus URLs. Assim, inicialmente uma ferramenta cliente só precisa receber o URL raiz para poder navegar entre os recursos. HATEOAS é uma forma de se modelar uma máquina de estados em um meio sem estado [9].

Para utilizar o HATEOAS em um determinado recurso, um *script* deve achar os meta dados que referenciam a *hypermidia* para outro recurso.

2.6.4 Tipo de Relação de Link de Perfil

O tipo de relação de link de perfil pode ser definido como uma semântica adicional a alguma representação de um recurso. Essa semântica pode incluir condições, extensões e convenções, mas não deve alterar as regras de semântica original, de tal forma que um cliente que consuma a representação do recurso que tenha um link de perfil não dependa desse perfil, mas sim seja auxiliado por esse [21].

Um exemplo de perfil de seria a documentação legível por seres humanos de determinada API [18].

2.6.5 Documentos de Representação

Os documentos de representação devem ser padronizados em uma sintaxe determinada, para que clientes e servidores, ou mesmo pares de clientes possam se comunicar inequivocamente. Para poderem entender os dados, tanto o remetente de uma representação quanto o destinatário devem ter o conhecimento necessário para ler ou escrever a semântica dos dados presentes na documentação. Dois exemplos de tipos de documentos de representação são o XML e o Notação de Objeto *JavaScript* (*JavaScript Object Notation*) (JSON).

JSON

Notação de Objeto *JavaScript* (*JavaScript Object Notation*) (JSON) trata-se de uma formatação para troca de dados, utilizado como recurso na web. Entre as vantagens da utilização está a independência da linguagem de *backend*, a facilidade de humanos e máquinas lerem, interpretarem e escreverem os dados e a possibilidade de estruturá-los [22].

É possível organizar o JSON em duas estruturas: um par, constituído de um nome e um valor, usualmente chamado de objeto; uma lista ordenada de valores, chamada vetor. Um objeto é localizado entre chaves, e o nome é separado do valor pelo sinal de ":", dois pontos. No caso do vetor, os valores ficam dentro de colchetes, com os valores separados por vírgulas.

2.7 Design de *frontend*

O design de software é um artefato produzido por um processo de desenvolvimento de software, possuindo ciclo de desenvolvimento próprio e sendo aplicável a diversos atributos do software, como componentes e interfaces. O design é reavaliado diversas vezes, para garantir que está de acordo com os requisitos do software. Ao final, o design de determinado componente mostra como está organizada a arquitetura desse [10].

No contexto de uma API, o design do *frontend*, que são os componentes visíveis ao usuário final, possui requisitos focados nesse, que é o(a) desenvolvedor(a) que utilizará esta API em projetos posteriores. Por esta razão, são analisadas a experiência de desenvolvedores e a usabilidade quanto a uma API.

2.7.1 Experiência do Desenvolvedor e Usabilidade

Existem duas perspectivas para a experiência do desenvolvedor que utiliza uma API de terceiros: desenvolvedor como usuário da ferramenta, cujo foco é a percepção de um usuário, como utilidade, valor e apelo, em relação a determinado produto; a (), que visa o engajamento de desenvolvedores à atividade de criação de software, em vez de apenas consumi-lo.

DEx

A importância da DEx vem do fato de que ambientes de desenvolvimento de softwares que causem confiança, boa comunicação e clareza à comunidade de desenvolvedores influencia positivamente na produtividade e no sucesso de projetos [23].

Sabe-se que o desenvolvimento de softwares é uma atividade intelectual, e em razão disso está sujeito à questões cognitivas, de afetos e conativos (vontades, impulsos). Fagerholm e Münch dividem a experiência de desenvolvimento conforme essas entidades psicológicas, tendo assim três partes:

- Infraestrutura de desenvolvimento - bibliotecas, *frameworks*, ambientes de desenvolvimento entre outras ferramentas;

- Afeto sobre o trabalho - como o(a) desenvolvedor(a) se sente em relação ao trabalho, se sente respeito, pertencimento;
- Valor da própria contribuição - se o significado e objetivo do projeto está de acordo com as visões de quem o desenvolve.

Todos esse fatores, se estiverem positivos segundo o(a) desenvolvedor(a), aumentam a chance de sucesso em um projeto [23].

Usabilidade no consumo de APIs

Há perspectivas que veem os desenvolvedores que utilizam uma API como usuários dessa, como McLellan et al. [24]. Para o usuário, um aspecto importante é a usabilidade de um produto, sendo que a usabilidade é um atributo de qualidade que mede a facilidade e o quão agradável é a um usuário usar determinada interface [25]. Diferentemente da utilidade de um produto, que se refere às funções que o produto exerce, a usabilidade trata-se de um atributo subjetivo. Também é importante frisar que muitos dizem que a usabilidade é dependente do contexto de uso [26].

Como apontado por Mosqueira-Rey et al. a literatura sobre usabilidade de APIs é heterogênea e mais focada em aspectos mensuráveis e objetivos do que na experiência humana e subjetiva em relação à usabilidade. Outro aspecto sobre esta literatura, é que grande parte dos guias sobre desenvolvimento de APIs se foca em recomendação, e não em um *framework* bem estabelecido. Transformar essas recomendações em um *framework* requer esforços adicionais, além de que é possível que diferentes autores façam recomendações incompatíveis entre si. Mosqueira-Rey et al. também apontam que o contexto de APIs não é levado em conta, na literatura e que determinar se uma API possui uma boa usabilidade é uma tarefa ainda complicada [22].

Alonso-Ríos et al. desenvolveram um taxonomia de usabilidade com intuito de ser abrangente e estruturar hierarquias de propriedades de produtos e que pode ser usado tanto na concepção de uma API que visa a usabilidade, quanto a qualquer produto. Segundo eles, a usabilidade está atrelada aos seguintes atributos:

- Conhecimento - abrangendo clareza, consistência, memorabilidade;
- Operabilidade - abrangendo corretude, universalidade, precisão, flexibilidade;
- Eficiência - tanto em tempo de execução de máquina quanto de esforço humano, economia de custos;
- Robustez - a erros, usos indevidos, abusos de terceiros;
- Segurança - segurança de usuários, segurança contra terceiros;

- Satisfação subjetiva - interesse, estética.

Recomendações de design

Como mencionado, há diversas recomendações na literatura sobre API. Para facilitar o trabalho de quem irá usar a API, é essencial que a API seja desenhada de modo intuitivo, simples e bem documentado, principalmente em relação aos URLs disponíveis para o usuário.

O URL deve ser auto-descritivo, para facilitar o uso e não gerar a necessidade de verificar a documentação exaustivamente. Termos dos descritores devem ser concretos e intuitivos. Também deve haver apenas um caminho para efetuar uma ação [12].

Um recurso, endereçado pelo URL, é requisitado pelo cliente ao servidor. Trata-se de um componente chave na API e, sendo difícil de se modificar depois de uma API pronta, procura-se investir esforço para se decidir como serão os recursos. Uma forma de se pensar em recursos para melhor entendimento é vendo-os como uma linha em uma tabela de banco de dados.

Os recursos podem ser divididos em

- Recurso de coleção - recurso que é uma lista de recursos do mesmo tipo;
- Recurso de instância - representada por um único objeto de negócio;
- Recurso de controle - executa alguma funcionalidade fora os métodos padrões do HTTP.

Boas práticas em design de recurso

Para melhorar a experiência do desenvolvedor, as seguintes práticas de design de recursos devem ser **evitadas**:

- Redundância - os mesmos dados estão ligados a mais de um recurso;
- Exposição de dados internos de implementação;
- Recursos compostos - vários recursos combinados.

Ordem dos recursos

A ordem dos recursos pode ser vista em formato de árvore a partir da hierarquia presente nos endereço URL. Um exemplo seria:

http://aeroportojk/acis/parceiros/10342

Neste exemplo, até o sub-recurso "parceiros", tem-se o endereço de um recurso de coleção, enquanto o sub-recurso incluindo o numeral é um recurso de instância do parceiro de identificador 10342.

Granularidade

Uma prática recomendada de design é a granularidade dos recursos. Os recursos devem ser divididos para facilitar o entendimento para o desenvolvedor usuário da API. Dado isso, é importante julgar se os modelos de dados usados no *backend* devem ser os mesmos do *frontend* da API. Recursos associados a muitos dados podem usar banda em demasia, enquanto recursos pequenos demais podem levar à iteração de requisições. Isso ocorre, pois um cliente pode requisitar um recurso inteiro ou uma página de determinado recurso; quanto mais dados tiver este recurso ou esta página, mais banda do meio de transmissão (geralmente Internet) será usada. Entretanto se um recurso ou página for pequeno demais, o cliente pode não encontrar os dados que procura e ter que fazer uma nova requisição para um recurso ou uma página.

Para contornar esse problema, é mais eficiente disponibilizar recursos pequenos e que possuem endereços para outros recursos associados a eles. Como exemplo, é possível separar dados de diferentes tipos de uma mesma entidade e colocar cada tipo em um recurso diferente e associá-los a endereços dos recursos dos demais dados dessa entidade. Uma das formas de fazê-lo é ordenando os recursos [9]. Com isso, diminui-se a quantidade de dados no tráfego entre cliente e servidor.

2.7.2 Design de URI

Os identificadores de recurso uniforme, cujo termo URI é geralmente alternado com o termo URL, apesar de URI ser um termo mais genérico, está intimamente ligado ao recurso. As recomendações para o uso das URI são:

- letras minúsculas;
- não usar a barra "/"no final;
- usar hífen em vez de linha baixa para separar palavras;
- não usar extensão de arquivos;
- nomes de recursos devem estar no plural;
- não usar sinal de espaço.

Outros padrões são indicados na construção de uma API:

- No caso de coleções de recursos, o substantivo do recurso deve estar no plural;
- O URI não deve passar de 2000 caracteres, por limitações de navegadores;
- Não se deve usar a extensão do arquivo da representação, pois o HTTP possui o campo *Content-Type* em seu cabeçalho para isso.

Modelo de URI

Em uma API, geralmente o URI aparece seguindo o seguinte padrão:

/<versão-da-API>/<espaço-de-nome>/<recurso>/<recurso-id>/<sub-recurso>/<sub-recurso-id>

Recomenda-se que o número de sub-recursos não passe de dois, totalizando uma profundidade de três níveis em uma árvore hierárquica.

No caso de uma *query*, o padrão é:

<recurso>?<parâmetro-1>=<valor-1>&<parâmetro-2>=<valor-2>

Em que um sinal de interrogação separa os parâmetros a serem buscados, cujo valor aparece após um sinal de igualdade, e que podem ser um ou mais, unidos por um "e"comercial entre cada.

URIs devem manter-se as mesmas ao longo do tempo, pois atualizá-las pode causar inconsistência nos componentes do cliente.

2.7.3 Procedimentos de design

Para compor um recurso ou um atributo desse recurso, como um objeto e suas características, são criados nomes chamados de descritores semânticos. O cliente pode ler e requisitar esses descritores, assim como o servidor pode organizá-los, criá-los e fornecê-los.

A escolha desses descritores faz parte de um processo de design de qualquer API, e tem como âmbito manter padrões e facilitar o entendimento do usuário cliente. Além da escolha dos descritores, o design também pode estar associado a uma documentação sobre a utilização da API. Dessa forma, são estipuladas as seguintes etapas de design [18]:

1. Listar as informações que um cliente possa requerer da API ou enviá-las para a API, e cujas entidades formarão os descritores;
2. Conceber um esquemático para a API, em que cada módulo seja uma representação e as setas as quais os interligam sejam novas requisições HTTP;
3. Tentar adaptar os descritores escolhidos com semânticas já existentes e padronizadas;
4. Escolher um formato de documento de representação que seja compatível;

5. Conceber um perfil que servirá de documentação para a API e que explique os descritores semânticos;
6. Desenvolver um servidor HTTP que forneça as representações segundo o esquemático criado;
7. Publicar a API, junto a opcionais documentações e exemplos.

2.8 Decisões de Design de Backend

É possível distinguir a abordagem para o desenvolvimento de *backends* de APIs de duas formas: abordagem de legado e abordagem de campo verde (*green field*). Enquanto a primeira pressupõe que uma API será desenvolvida para usar um sistema de *backend* com lógica de negócio e lógica de dados previamente desenvolvidas, a segunda abordagem considera que uma lógica de negócios será desenvolvida como parte da API, a partir das necessidades desta [9]. Desse modo, percebe-se que a abordagem de legado é mais modularizada do que a de campo verde, tendo uma separação mais evidente entre a API e a camada de negócio.

2.8.1 *Backend*

É comum que o sistemas de *backend* possuam tanto a lógica de negócio quanto a lógica de dados, contendo as usuais camadas de persistência. O *backend* provê dados para as camadas de cima, como o *frontend*. Em oposição a uma API, que deve fornecer dados organizados em uma representação para facilitar o consumo pelo cliente, as camadas de *backend* podem fornecer outras estruturas de dados, podendo ser inadequadas para o envio para quem os consome pela API REST.

2.8.2 Transformações

Uma camada de transformações pode ser necessária para transformar objetos provindos dos sistemas de *backend* para a camada da API, em que uma representação daquele objeto será requisitada pelo cliente, assim como pode ser necessário transformar uma representação recebida do cliente para o *backend*. Pode ser responsabilidade da camada de transformação possuir mecanismos de segurança, para que um cliente externo não acesse a camada de persistência indiscriminadamente.

2.9 Requisitos não funcionais

Entre os possíveis requisitos não funcionais de uma API, podemos citar:

- Segurança;
- Disponibilidade;
- Performance;
- Capacidade de evoluir.

2.10 Processos de software

Apesar de haver recomendações sobre como desenvolver uma API REST na literatura especializada, pode-se ver tal API como um artefato de software. Logo, as boas práticas da engenharia de software também têm efeito positivo no desenvolvimento de uma API REST, podendo encontrar os mesmos objetivos de qualidade de software se aplicadas.

No intuito de se conceber um software com qualidade, várias atividades são planejadas e organizadas em um processo de software. Os processos podem seguir vários modelos estipulados, como o Processo em Cascata ou o Desenvolvimento Incremental, mas todos esses modelos devem possuir as seguintes etapas [27]:

- Especificação de software: determina as funcionalidades que o software deve prover e as suas restrições;
- Projeto e implementação: onde os desenvolvedores criam o software com base nas especificações;
- Validação de software: o software é avaliado conforme as demandas do cliente;
- Evolução de software: eventuais ajustes são feitos, conforme as necessidades que o cliente achou durante a validação.

Capítulo 3

A-CDM e SWIM

Neste capítulo, os conceitos relevantes no contexto do trabalho são apresentados. Trata-se do domínio de operações aeroportuárias e do gerenciamento de tráfego aéreo, ainda que o último tenha uma relação mais distante com os conceitos apresentados neste capítulo.

A partir dos conceitos discutidos neste capítulo, tem-se as bases para entender em qual contexto a API REST desenvolvida neste trabalho será aplicada. Tal base facilitará o processo de levantamento de requisitos, explorado no próximo capítulo.

3.1 A-CDM

Como expresso anteriormente, o processo de A-CDM é um novo paradigma operacional nos aeroportos, cujo objetivo é aumentar a eficiência, a previsibilidade e a pontualidade das atividades aeroportuárias, fornecendo o ambiente e a cultura necessários para que os atores envolvidos com o aeroporto em questão, e com o tráfego aéreo de modo geral, tomem decisões colaborativamente e de modo distribuído. O A-CDM é focado principalmente nas atividades táticas e pré-táticas [3].

Um dos grandes méritos do A-CDM é a distribuição e o fornecimento de informações de alta qualidade e no tempo certo aos seus interessados, a fim de que esses possam tomar as suas decisões. Pode-se dizer que os estados de todos os parceiros envolvidos ficam visíveis aos demais. Um exemplo de aplicação é o aumento da ciência de um operador de aeronave sobre o estado e a localização de sua aeronave e das demais. Com esse tipo de informação, é possível reduzir o tempo que uma aeronave aguarda na pista para levantar voo, o que economiza combustível.

Em termos gerais, pode-se dizer que o A-CDM é uma distribuição melhor dos dados gerados por e para os parceiros aeroportuários. É necessário que todos esses contribuam para o processo, disponibilizando suas informações em tempo hábil. Por último, há outros atores que influenciam as atividades de um aeroporto, além dos operadores de nave e de

aeroporto, como os operadores de rede ou questões nos arredores de um aeroporto, como congestionamento em tráfego que impede a chegada de passageiros ao aeroporto, clima etc., cujas informações podem ser relevantes para o A-CDM [1].

O A-CDM é um conceito desenvolvido pela Eurocontrol, sendo uma das ideias propostas para a aviação do século XXI, prevendo aumento na demanda por essa atividade, apesar disso, o conceito de CDM já foi desenvolvido previamente, ligado, principalmente, a questões de negócios inteligentes.

3.1.1 Parceiros A-CDM

A maior motivação do A-CDM é que os parceiros acessem e forneçam os dados uns aos outros. Os principais parceiros identificados do A-CDM são

- Operadores de aeronaves;
- Operadores aeroportuários;
- Serviços de solo;
- Provedor de serviço de navegação aérea;
- Operador de rede;
- Serviços de suporte.

Outros parceiros podem contribuir ou acessar informações relevantes aos processos do A-CDM, como passageiros, serviços de imigração e serviços de previsão meteorológica.

3.1.2 Conceitos chaves

Alguns conceitos foram criados para facilitar a divisão das etapas durante os processos do A-CDM. Esses conceitos devem ser disponibilizados como um software e são eles [28]:

- **Compartilhamento de informações:** ferramenta essencial que precisa ser implementada antes das demais;
- **Abordagem por marco histórico:** usar eventos bem definidos para conscientizar os parceiros dos acontecimentos de um voo desde o seu planejamento até a decolagem;
- **Tempo de taxiamento variável:** predizer uma decolagem em um espeço de tempo variável;

- **Sequência de pré-decolagem colaborativa:** definir uma sequência de decolagem de acordo com a preferência dos parceiros e demais condições;
- **CDM em condições adversas:** administrar o CDM durante períodos de menor capacidade do aeroporto;
- **Gerência colaborativa de atualizações de voos:** aumentar a acurácia de informações sobre decolagens e pousos entre os operadores de rede e os do aeroporto.

3.1.3 Objetivos dos parceiros

Enquanto o objetivo genérico do A-CDM é aumentar a predicabilidade e a eficiência das operações de um determinado aeroporto, os parceiros envolvidos com o A-CDM podem ter objetivos diferentes e mesmo contraditórios. Os processos do A-CDM preveem essa possibilidade. A figura a seguir identifica alguns desses diversos objetivos.

3.1.4 ACIS: A-CDM *Information Sharing* - Compartilhamento de Informações

O sistema do A-CDM *Information Sharing* (ACIS) é uma das partes que compõem a plataforma de software do A-CDM. Esse aspecto do A-CDM não é apenas o primeiro a ser considerando, como também o objetivo principal deste trabalho. Os objetivos principais desse conceito são:

- Agir como ponto comum entre os diversos sistemas de processamento de dados dos parceiros (em especial, do controle de tráfego aéreo, dos serviços de aeroporto e operadores de aeronaves);
- Uniformizar os dados descritivos de intenções e estados de um voo;
- Prover uma interface; interativa para o compartilhamento de informações;
- Informar previsões e estados;
- Criar sistema de alerta;
- Salvar dados para análises futuras.

A figura 3.2 apresenta um esquema dos dados de entrada e saída do A-CIS.

Evidentemente, o compartilhamento de informações torna-se um sistema grande e complexo, no qual componentes de cálculo, estatística e interface humano-computador devem ser estudados e levados em consideração.

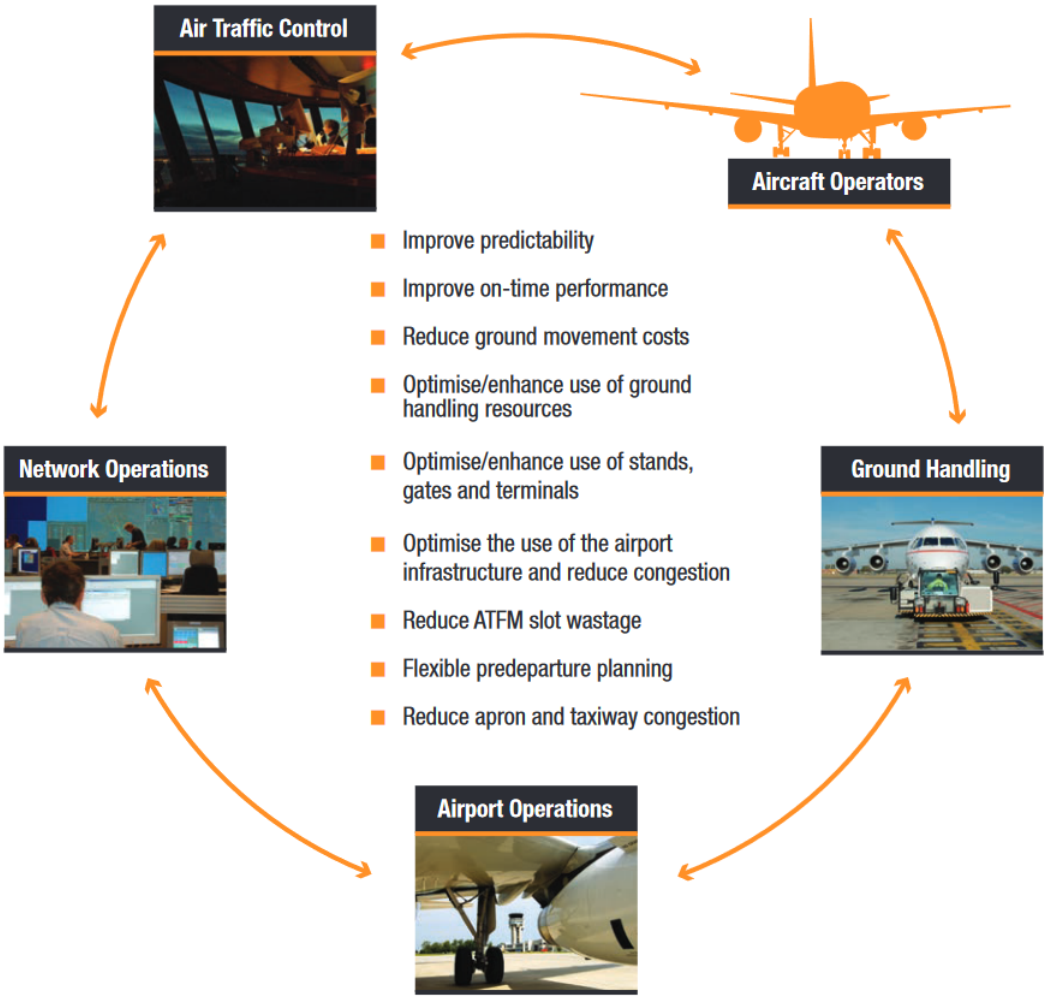

Figura 3.1: Objetivos A-CDM. Figura retirada do The Manual Airport CDM Implementation cápitulo 2, página 5 [1].

3.2 SWIM

Enquanto o A-CDM é pensado para que os diversos parceiros envolvidos com um aeroporto específico tomem suas decisões colaborativamente, o SWIM é um conceito o qual abrange entidades que surpassam um único aeroporto, podendo abranger diversos aeródromos, controle de tráfego aéreo, organizações governamentais, estações meteorológicas entre outros. Isto ocorre, pois o SWIM visa conectar os sistemas de informação e protocolos de todos os envolvidos no gerenciamento de tráfego aéreo (ATM).

Em outras palavras, pode-se afirmar que o SWIM pretende apresentar uniformidade entre as comunicações do ATM, ao estabelecer padrões de estrutura, protocolos e gover-

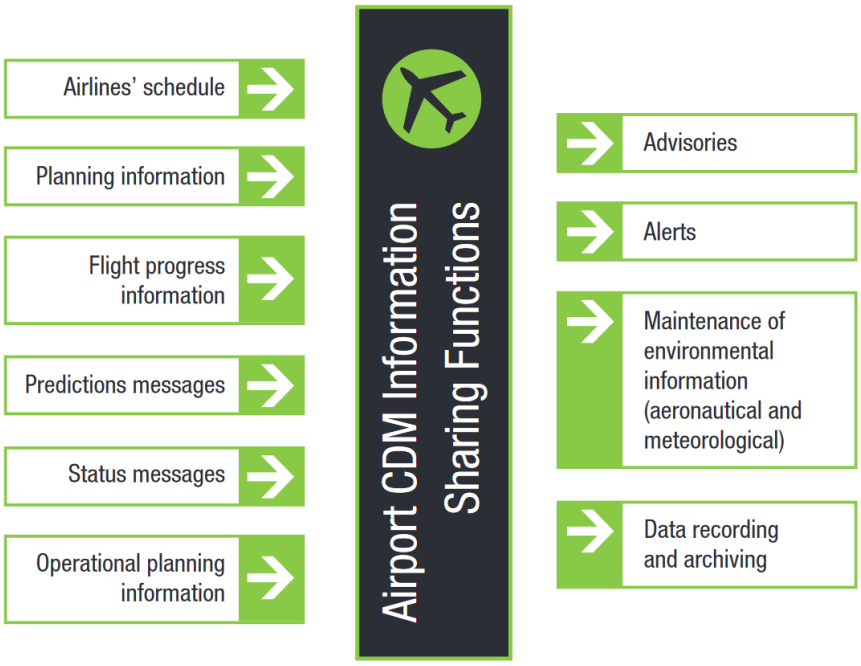

Figura 3.2: Funções do Compartilhamento de Informações. Figura retirada do The Manual Airport CDM Implementation cápitulo 3, página 2 [1].

nança proveem melhor interoperabilidade entre sistemas [29]. A figura 3.3 mostra como a arquitetura do SWIM é estipulada.

3.2.1 Objetivos

De maneira geral, pode-se dizer que o SWIM tem como objetivo interligar variados sistemas. As metas podem ser definidas como:

- Estabelecer parâmetros comuns entre os interessados;
- Facilitar a interoperabilidade entre dois ou mais sistemas distintos;
- Melhorar os processos de tomadas de decisão entre os interessados;
- Baratear e flexibilizar o acesso à informação entre os *stakeholders*.

3.2.2 Definição

O SWIM é um conceito que abrange padrões, infraestrutura e regulamentos para disponibilizar seus objetivos de interoperabilidade entre sistemas. Os sistemas que utilizam o SWIM para operar são as aplicações baseadas no SWIM, e não fazem parte do núcleo

do SWIM, mas fornecem ou consomem serviços de informação SWIM de acordo com os padrões desse último.

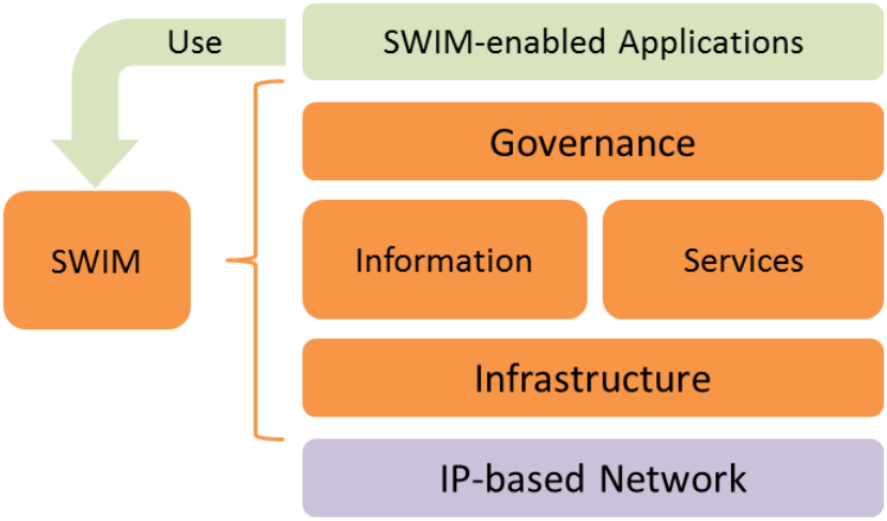

Figura 3.3: Estruturas que compõem o SWIM. Figura retirada do MANUAL ON SYSTEM WIDE INFORMATION MANAGEMENT (SWIM) CONCEPT, capítulo 2, página 2 [2].

3.2.3 SWIM-Registry

O registro do SWIM é uma ou mais plataforma desenvolvida para que os artefatos habilitados pelo SWIM sejam devidamente cadastrados e provejam as instruções de consumo de seus dados e serviços.

O SWIM *Registry* facilita o acesso à informação, em razão de que centraliza o acesso a diferentes serviços habilitados pelo SWIM, como os padrões de troca de mensagem (AIXM, FIXM, por exemplo), descrição de documentos, participantes do SWIM entre outros. Geralmente, há uma autoridade que gere o SWIM *Registry*, permitindo o cadastro dos participantes [30].

O Translab desenvolveu o primeiro SWIM *Registry* do Brasil, e atualmente contém o serviço meteorológico cadastrado.

3.2.4 SOA e SWIM

A SOA é uma solução para quando se há diferentes provedores de serviço descentralizados e possivelmente de diferentes donos, mas que se organizam para prover e consumir esses serviços, como já foi explicado mais detalhadamente no capítulo 2. Dessa forma, a SOA é

ideal para ser adotada pelo SWIM, cuja natureza é a mesma. O SWIM *Registry* é o meio pelos quais esses serviços apresentam suas formas de consumo.

De acordo com o manual do SWIM ([2]), o SOAP é usado como solução padrão na troca de mensagens. O próprio manual aponta, porém, que também é possível utilizar o REST, principalmente devido a sua facilidade de utilização.

Capítulo 4

Requisitos e Projeto

O Compartilhamento de Informações do A-CDM (Airport CDM Information Sharing, ACIS) é tido como o primeiro sistema entre os que constituem a plataforma completa do A-CDM. Esse sistema tem como função integrar os parceiros e os demais sistemas em um só ambiente, provendo as funcionalidades necessárias para tanto.

O ACIS pode ser visto como um sistema no qual os diversos parceiros A-CDM e outros sistemas da plataforma para o A-CDM consomem e disponibilizam informações entre si. Não obstante, o ACIS é passível de ser disponibilizado como um serviço ao SWIM, sendo cadastrável no registro desse. Dessa forma, é evidente a necessidade de se desenvolver interfaces aos usuários do ACIS. Por isso, soluções relativas à SOA podem ser empregadas, trazendo todos os benefícios dessa arquitetura para as interfaces operáveis do sistema ACIS.

Espera-se que haja sistemas já desenvolvidos entre os parceiros A-CDM. Entretanto, esses não precisem ser substituídos, mas sim atualizados para que possam disponibilizar e receber informações de outros sistemas, os quais incluem a plataforma do A-CDM.

4.1 Decisões de Projeto

Como mencionado no capítulo 2, não existe apenas uma forma de se criar um *design* de API. Pelo contrário, há uma vasta gama de opções de recomendações diferentes. Pode-se ver uma API como um artefato de *software* e utilizar as estratégias comuns de Engenharia de *Software* para desenvolvê-la. Existem também recomendações que são próprias para o desenvolvimento de uma API.

Para este trabalho, optou-se pela literatura especializada. As etapas de desenvolvimento encontrada nesta literatura, em especial os livros RESTful Web API, da editora O'Reilly (referência [18]) e API Design, da API University Press (referência [9]) foram as adotadas neste trabalho. A escolha se justifica, pois tais fontes se mostraram as mais deta-

lhadas e completas em relação ao *design* de APIs REST particularmente em comparação com as outras partes da literatura exploradas.

4.2 Conceitos importados do A-CDM

Alguns conceitos do A-CDM serão utilizados para compor o ACIS. Esses são:

- Evento: uma ocorrência específica no planejamento ou nas operações de um voo, que uma pessoa ou sistema percebe e responde de maneira determinada;
- Marco Histórico: tipo de evento com significado importante que ocorre em um planejamento ou operação de voo. Um marco histórico completado com sucesso acionará um processo de decisão de evento subsequentes e que influenciam no progresso do voo e na acurácia com que o progresso pode ser previsto;
- Alerta: uma mensagem automática do sistema que ocorre em caso de evento irregular, e que precisa de um ou mais parceiros para ser atendida;
- Tempo Alvo: tempo dentro do qual um Marco Histórico precisa ser completo.

4.3 Requisitos funcionais

Alguns requisitos da implementação do ACIS são encontrados em um documento da ETSI [28]. Esses são mostrados resumidamente:

- Mostrar informação idêntica a todos usuários;
- Identificar usuário;
- Validar usuário;
- Autorizar usuário;
- Prover mensagens de alerta;
- Gerenciar mensagens de alerta;
- Prover entrada de dados manualmente;
- Prover modificação de dados manualmente;
- Distribuir informações de voos;
- Acesso a informações aeronáuticas relevantes;

- Acesso a informações meteorológicas relevantes;
- Salvar dados;
- Filtrar informações de acordo com a origem;
- Filtrar informações de acordo com o horário de criação.

4.4 Requisitos de interface

No mesmo documento citado na seção anterior, há os requisitos de interface do ACIS:

- Identificação única do voo de cada mensagem;
- Formato de mensagem de acordo com as especificações do ICAO e do IATA;
- Identificação dos originadores e de modificadores de quaisquer dados;
- Permitir troca de mensagens entre parceiros;
- Identificar eventos e seus horários, datas, formato e qualidade de serviço;
- Identificação do tempo mínimo estimado de parâmetros de duração;
- Identificação dos recursos do aeroporto;
- Identificação de alerta de A-CDM.

4.5 Decisões de design

A partir dos requisitos supracitados, e seguindo o processo de design para a elucidação dos descritores, pode-se considerar que as seguintes entidades podem ser traduzidas em recursos para a API do ACIS:

- usuário;
- alerta;
- evento (marco histórico e tempo alvo são atributos de evento);
- informações de estado de voos;
- plano de voo;
- informações aeronáuticas relevantes;
- recursos de aeroporto;

- mensagens entre parceiros.

Todas essas entidades são passíveis de se tornarem um descritor semântico de uma coleção, como na figura a seguir, na qual cada módulo é um recurso de coleção e as setas representam *hyperlinks* com as referências a outros recursos.

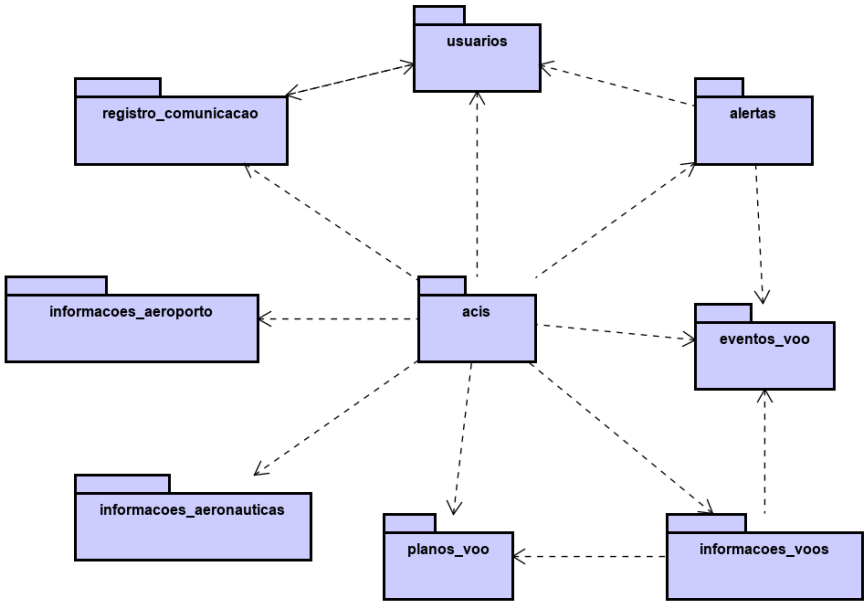

Figura 4.1: Descritores da API REST do ACIS.

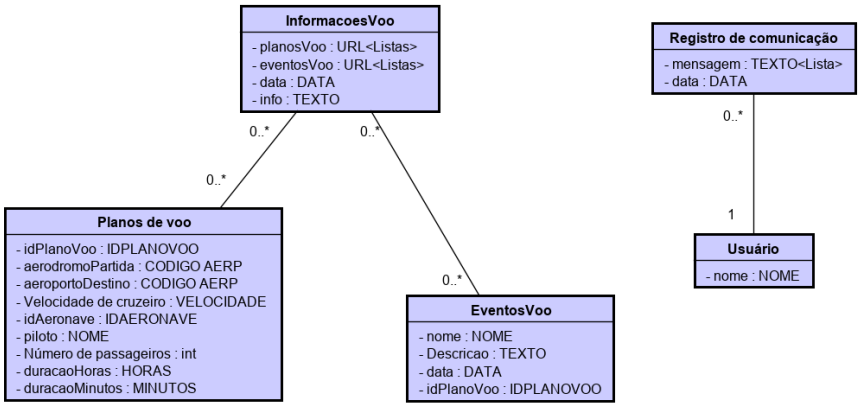

Figura 4.2: Escopo da prova de conceito contendo os descritores dos recursos.

4.5.1 Definições para a prova de conceito

O esquema da figura 4.2 será usado neste trabalho como prova de conceito. Essa subestrutura de recursos da API está baseada em duas entidades principais: planos de voo e informações de voo.

O plano de voo tratado neste projeto é a parte frontal do plano de voo simplificado fornecido pelo DECEA [31]. Cada plano de voo deve ter uma única rota. Cada rota é um conjunto de pontos do mapa aéreo e pode conter um ou mais planos de voo, pois mais de um voo pode passar pela mesma rota em diferentes horários.

Já as informações de voos podem ser diversas e, diferentemente do modelo de plano de voo, devem ter maior preocupações relativas aos aeroportos de destino e de partida do que relativas à rota de um voo. Cada instância de uma informação de voo deve possuir um plano de voo e pode possuir zero ou vários eventos e zero ou vários registros de comunicação. Os eventos de voo serão aqueles relevantes ao ACIS, enquanto que o registro de comunicação é qualquer mensagem enviada por um parceiro A-CDM que seja atrelada àquele voo em questão.

Planos de voo

O plano de voo é uma entidade que tem maior relação com o ATM do que com o A-CDM em si, pois o conjunto inteiro de informações é de maior relevância no âmbito estratégico do que tático e pré-tático, como rotas e aeródromos alternativos. Entretanto, as informações de planos de voo devem ser compartilhadas com o meio do A-CDM por serem completas, trazendo detalhes que podem ser relevantes aos parceiros do A-CDM.

Informações de voo

A definição da entidade de informações de voo é ser um conjunto de diversas informações que podem vir a ser relevantes aos parceiros de modo generalizado. Por isso, cada entidade de informações de voo pode estar relacionada a mais de um voo, isto é, a mais de uma plano de voo e a mais de um Evento de Voo, além de conter uma caixa de texto, dando liberdade a quem instanciar uma entidade Informações de Voo de preenchê-la. É interessante notar que mais de um voo ou evento de voo podem compartilhar um mesmo estado, como, por exemplo, "cancelados devido ao mau tempo".

Evento de Voo

A entidade evento de voo é pensada para conter dados padronizados do A-CDM. A princípio, porém, pode-se escrever um texto nesta, além de poder associá-la a um identificador de Plano de Voo.

Registro de Comunicação

Como os parceiros podem ter decisões, perguntas e comentários a serem declarados aos outros parceiros do A-CDM, eles têm essa possibilidade de fazê-lo por meio da entidade registro de comunicação, que contém um texto livre para a escrita da mensagem e relaciona-se ao parceiro.

User

Destinada a representar um usuário parceiro do A-CDM, a entidade *user* não possui uma representação na API. Entretanto, espera-se que o usuário seja previamente cadastrado no sistema para poder acessar e modificar os dados do ACIS. Desse modo, também é provido o conceito de segurança, exigindo autenticação e autorização para que um usuário possa usar a API.

Capítulo 5

Implementação e Resultados

Neste capítulo, são apresentadas questões da arquitetura usada na implementação, assim como alguns exemplos que mostram o funcionamento da API REST conforme o projeto descrito no capítulo anterior. Os exemplos estão focados principalmente nas funcionalidades da API e em como fazer uma requisição a essas funcionalidades.

Com o intuito de provar o conceito de uma API *RESTful* aplicada ao funcionamento do sistema do A-CDM, ACIS, algumas decisões foram tomadas. A primeira foi a de utilizar o *framework* em *python* Django 2.2, que oferece uma estrutura em alto nível para programação *web*. A linguagem *python* utilizada foi a de versão 3.6.7. A segunda decisão foi a de usar o *framework* para desenvolvimento de APIs *RESTful* no Django, o Django REST *framework*. De acordo com os *frameworks*, o sistema foi dividido "verticalmente"em aplicações e "horizontalmente"em módulos, de forma que cada uma das aplicações tenham os mesmos tipos de módulos em sua implementação e aquelas são independente entre si.

5.1 Aplicações

Seguindo os padrões do *framework* usado, foi desenvolvido um sistema dividido em aplicações. Cada aplicação é um diretório e todas as aplicações contêm basicamente os mesmos módulos.

A grande vantagem de dividir o sistema como um todo em diversas aplicações, é que cada aplicação fica responsável por apenas um modelo de dado. O modelo de dados, o qual será discutido, pode ser visto como uma tabela em um banco de dados relacional, que é uma entidade. Desse modo, pode-se gerir cada entidade independentemente das outras, facilitando o entendimento do código. Um aspecto negativo é o retrabalho que há em cada aplicação, apesar de isso ser algo diminuto.

As aplicações são:

- ACIS - raiz do sistema, relaciona o URL de cada aplicação aos seus respectivos módulos (*ViewSets*, no caso);
- eventos_voo - responsável pela entrada e saída de dados da entidade de eventos sobre voos;
- informacoes_voo - responsável pela entrada e saída de dados da entidade sobre informações genéricas sobre voos;
- planos_voo - responsável pela entrada e saída de dados da entidade de planos de voo;
- registro_comunicacao - responsável pela entrada e saída de dados da entidade de mensagens trocadas pelos usuários do sistema.

5.2 Módulos

Cada módulo tem sua função bem definida dentro da arquitetura do sistema, contendo, geralmente, uma classe. Para se comunicarem uma com a outra, a classe de um módulo é importada para os módulos que dependerem dela.

Model

Model é o módulo que se comunica com o banco de dados. Uma classe entidade é declarada, instanciando a classe do *framework* "models.Model". Essa classe recebe os atributos que serão da entidade, assim como alguns argumentos desses atributos.

Serializers

Para transformar a entidade em uma representação, é feito o módulo *serializer*. Neste trabalho, o formato das representações é JSON. Além disso, o módulo em questão também transforma um texto de entrada de representação em um formato correto para ser guardado em um banco de dados. Nem todos os atributos de uma entidade precisam fazer parte da representação desta, e no *serializer* tais atributos são escolhidos.

ViewSets

Como supõe o nome, *ViewSets* é um conjunto *views*. As *views* são funções que recebem uma requisição de um cliente como parâmetro e retornam-lhe uma resposta. Dentro do contexto de uma API no Django REST *framework*, o módulo viewsets proporciona algumas ações que vinculam uma representação do *serializer* ao método HTTP enviado pelo cliente. São as ações:

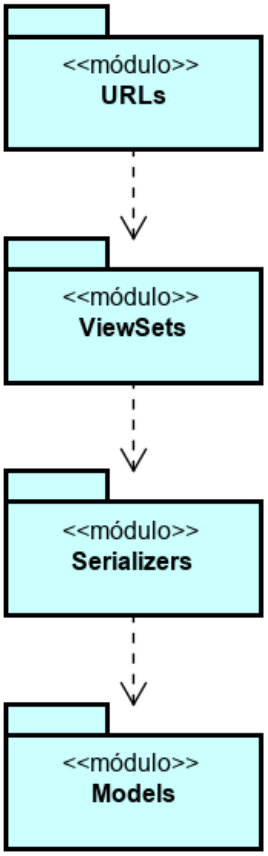

Figura 5.1: Esquema da disposição dos módulos segundo o *framework* adotado.

- *list*;
- *create*;
- *retrieve*;
- *update*;
- *partial_update*;
- *destroy*;

URLS

Para criar um recurso e associá-lo à classe do *viewsets*, há o módulo URL. O Django REST *framework* proporciona uma classe que registra todas as *viewsets*, chamada de *router*. Tal classe configura as questões do URL automaticamente, carecendo apenas do descritor semântico que deve ser associado àquele recurso.

5.3 Autenticação e autorização

Alguns dos requisitos não funcionais estipulados foram levados em consideração. O principal trata-se da segurança, em que há a preocupação de autenticar e autorizar um usuário para que ele tenha os dados disponibilizados pela API.

O "Django adim"é uma aplicação que provê a possibilidade de cadastrar um novo usuário como *user* e essa entidade faz parte do processo para realizar uma autenticação. Outra etapa é a geração de um *token*, associado ao *user*, disponibilizado pelo Django REST *framework*. Quando o cliente adiciona um cabeçalho informando que deseja se autenticar com um *token* de um usuário válido, torna-se possível acessar os dados da API. Se um usuário não informar esses dados, a API retorna o erro 401 do HTTP, informando a necessidade de autenticar. Já se o usuário informar um *token* inválido, a API responde com o erro 403 do HTTP, informando o usuário do erro.

5.4 Validação

A fim de se provar que a tecnologia REST poderia suprir as demandas que o sistema ACIS venha a ter em relação a sua API, foi feito um teste com cada uma das demandas, varrendo as possíveis funcionalidades e obtendo seus resultados. Considerando as possíveis trocas de dados entre máquinas clientes e servidoras, a API deve ser capaz de enviar uma resposta a uma requisição de coleções de dados; uma resposta a uma requisição com filtro de busco; aceitar uma nova instância em um a entidade; deletar uma instância existente. Também deve ser possível navegar entre as coleções e as instâncias por meio de *hyperlinks*.

Como cada uma das aplicações tem a mesma estrutura divisível em módulos e são todas passíveis das mesmas propriedades, neste trabalho, cada teste das funcionalidades só foi mostrado com uma aplicação, quando possível. Desse modo, foi feita uma varredura das funcionalidades da API REST nas aplicações desenvolvidas.

5.5 Provas de conceito

Como já foi afirmado anteriormente, uma API deve servir para que uma máquina se comunique com outra. Porém, é possível visualizar as ações de uma API por meio de uma interface gráfica ou de um programa apropriado. Neste trabalho, foi usado o programa *Postman* com tal intenção.

Para apresentar o funcionamento, o servidor do sistema foi rodado em um computador pessoal, de tal forma que o endereço do domínio e sua porta são "http://localhost:8000/". Após acessar qualquer URL do sistema enviando uma requisição HTTP, o cliente recebe

de volta uma resposta HTTP apropriada e, se não houver exceções, uma representação em JSON.

5.5.1 Raiz da API

Para localizar todas as representações disponíveis por meio de *hyperlinks*, há a raiz da API, informada pelo recurso http://localhost:8000/api/. A resposta é então obtida com uma representação JSON como mostrado a seguir:


```
{ "informacoesvoo": "http://localhost:8000/api/informacoesvoo/",
"planosvoo": "http://localhost:8000/api/planosvoo/",
"registrocomunicacao": "http://localhost:8000/api/registrocomunicacao/",
"eventosvoo": "http://localhost:8000/api/eventosvoo/"}
```


5.5.2 Requisitando coleções

O método HTTP GET é vinculado à ação de entregar dados ao cliente, como já discutido previamente. Pelo exemplo a seguir, tem-se o resultado de uma requisição HTTP com o método GET ao *hyperlink* forncecido pela raiz "http://localhost:8000/api/planosvoo/", que é um *link* para uma coleção da entidade planos de voo. A representação requisitada pelo cliente contém uma página com até dez entidades, em que cada atributo dessa entidade tem seu nome à esquerda de seu valor. Os atributos vistos não são todos os de um plano de voo, mas foram aqueles selecionados pelo módulo *serializer*.


```
{ "count": 3,
"next": null,
"previous": null,
"results": [
{
"idPlanoVoo": "TAM8005",
"data": "2019-06-26",
"idAeronave": "111111A",
"aerodromoPartida": "SABE",
"aerodromoDestino": "SBGR",
"piloto": "PILOTO 1",
"duracaoHoras": "2",
"duracaoMinutos": "20"
},
{
"idPlanoVoo": "TAM8009",
```

```
"data": "2019-06-30",
"idAeronave": "222222B",
"aerodromoPartida": "SABE",
"aerodromoDestino": "SBGR",
"piloto": "Piloto 2",
"duracaoHoras": "2",
"duracaoMinutos": "20"
},
{
"idPlanoVoo": "GOL9094",
"data": "2019-06-20",
"idAeronave": "333321",
"aerodromoPartida": "BSB",
"aerodromoDestino": "POA",
"piloto": "Piloto 3",
"duracaoHoras": "2",
"duracaoMinutos": "20"
} ] }
```


5.5.3 *Hypermedia* nas representações

A entidade informações de voo possui referências para as entidades planos de voos e evento de voo. Desse modo, deve ser possível chegar às instâncias associadas por uma determinada instância de informações de voo, por meio de uma ligação de *hypermedia*, ou *hyperlink*. Quando se faz uma requisição com o método GET ao URL "http://localhost:8000/api/planosvoo/3/", de uma única instância, o servidor envia a seguinte resposta:


```
{
"data": "2019-06-17",
"Info": "Neblina, aviões atrasados",
"planosVoo": [
"http://localhost:8000/api/planosvoo/1/",
"http://localhost:8000/api/planosvoo/2/"
],
"eventosVoo": [
"http://localhost:8000/api/eventosvoo/1/"
] }
```

Percebe-se que os atributos "planosVoo"e "eventosVoo"possuem como valor um vetor de URLs para as suas instâncias associadas àquela instância específica de informações de voo.

5.5.4 Filtragem de resultados

É possível fazer busca em resultados com valores específicos, tal qual foi dito no capítulo 2. O método GET e a URL "http://localhost:8000/api/informacoesvoo/?data=20%2F06%2F2019", cujo descritor final possui a sintaxe para fazer buscas, em que a data deve ser "20-06-2019", filtram as instâncias da informações de voo. Desse modo, o resultado para esta requisição é:

```
{ "count": 2,
"next": null,
"previous": null,
"results": [
"data": "2019-06-20",
"Info": "Avião à espera",
"planosVoo": [
"http://localhost:8000/api/planosvoo/3/"],
"eventosVoo": [] ,

"data": "2019-06-20",
"Info": "Avião abastecendo",
"planosVoo": [
"http://localhost:8000/api/planosvoo/3/"],
"eventosVoo": []
] }
```

5.5.5 Enviando uma nova instância

Para que um cliente use a API para enviar uma nova instância de alguma entidade, usa-se o método HTTP POST. Dessa forma, uma nova representação JSON deve ser enviada ao URL "http://localhost:8000/api/informacoesvoo/", por exemplo, com um cabeçalho HTTP com o método supracitado e um documento JSON apropriado. Ao recebê-la, o módulo *model* fará as validações dos dados. Então, o servidor envia ao cliente uma resposta contendo um código de HTTP. Se o código for 200, por exemplo, isso significa que os novos dados foram inseridos com sucesso no banco de dados.

5.5.6 Deletando uma instância existente

Para se deletar determinada instância, é necessário passar seu identificador. O URL "http://localhost:8000/api/planosvoo/2/", por exemplo, refere-se à instância com identificador de número "2"da coleção de planos de voo. Ao se mandar uma requisição com o método HTTP DELETE, o sistema entende como um pedido para ação de deletar aquela instanciação específica. Novamente, se o sistema obteve êxito em deletar, esse retorna um código HTTP 200 para o cliente.

5.6 Documentação

Segundo a literatura, existem mais de uma maneira de se documentar uma API para que um(a) desenvolvedor(a) a utilize em seus sistemas. Umas das formas é utilizando um esquemático, como apresentado nas figuras 4.1 e 4.2 do capítulo anterior; outra forma seria fazer um *link* de perfil para a documentação. Como um dos objetivos de uma API é que seja rapidamente entendida, foi optado por usar o esquemático para que se possa navegar entre os recursos. Além disso, também é necessário informar ao cliente o endereço inicial da API.

Por meio de *hyperlinks* entre os recursos, procura-se diminuir o esforço de um(a) desenvolvedor(a) de ter que checar em um documento à parte quais URLs um *script* deve acessar. Em vez disso, o *script* pode navegar entre as representações apenas analisando e acessando novas referências presentes em uma representação inicial. Além disso, a resposta a uma requisição GET de uma coleção já contém em seu cabeçalho as opções de métodos HTTP disponíveis para aquela entidade.

Uma última consideração é que o *framework* utilizado usa códigos padrões do HTTP para sucesso e diversos tipos de erros. Esses códigos são bem documentados e devem ser reconhecidos pelas aplicações clientes.

5.7 Repositório do trabalho

A fim de fazer versionamento, manter cópias de segurança e divulgar o código do trabalho a terceiros, este código foi disponibilizado no GitHub. A partir do endereço "https://github.com/HelenaSILS/ACIS_api", é possível acessar o repositório do projeto.

O repositório está dividido entre as aplicações, e cada aplicação em seus módulos. Todos os módulos referentes à API de cada aplicação estão em suas respectivas pastas com nome de "api".

Capítulo 6

Conclusão

6.1 Experiência de desenvolvimento da API REST

De forma geral, a experiência de desenvolver a API descrita nesse trabalho é dependente dos *frameworks* utilizados. Entretanto, não foram realizadas métricas objetivas em relação ao uso dessas ferramentas.

O *framework* Django possui uma vasta documentação, além de diversas opções de tutoriais feitos por terceiros. Trabalhar com esse *framework* torna-se gratificante, pela facilidade e flexibilidade provida. Já o *framework* REST possui consideravelmente menos documentação, e requer um esforço mais para ser dominado pelo desenvolvedor por isso. Esse também possui menos tutoriais de terceiros. Ainda assim, é um *framework* flexível e com várias opções de manipulação.

6.2 Considerações finais

REST é um estilo arquitetural intuitivo em parte por fazer compor diversos sistemas *web*, de modo que é um estilo familiar a qualquer usuário da internet. Diversos sítios na internet possuem recursos que seguem essa tecnologia, baseando-se no protocolo HTTP e em *hyperlinks*. É importante ressaltar, porém, que nem todo sítio que utilize HTTP e *hyperlinks* está de acordo com o estilo arquitetural REST.

A grande diferença entre um sítio com interface gráfica visualmente amigável e uma API é que a última é feita com o intuito de ser consumida por alguma máquina, em vez de seres humanos. Por isso, suas representações são mais simples, auxiliando ainda na economia de banda consumida na rede.

Sabe-se que a tecnologia REST é capaz de fornecer todos os requisitos funcionais propostos. Existem, entretanto, algumas limitações em relação à segurança. Apesar de ser possível transmitir uma representação criptografada pelo HTTP e ser possível autenticar

e autorizar um usuário, algumas aplicações podem exigir um grau maior e mais específico de segurança.

Uma API REST é simples e rápida de se implementar e de se consumir, utiliza pouca banda, pelo tamanho de suas representações, que podem, inclusive, ser paginadas. Por essas razões, uma API REST pode ser uma opção plausível para a plataforma do A-CDM, principalmente em relação a informações que não são de segurança crítica.

6.3 Trabalhos futuros

O primeiro trabalho futuro que se enxerga é terminar todo o escopo do ACIS. Além das outras entidades que devem ser desenvolvidas, há a necessidade de se fazer também uma interface gráfica. Relações entre as entidades também devem ser discutidas. Apesar de haver alguns documentos sobre os requisitos funcionais e não funcionais, há carência de documentos e nem todos os dados estão disponíveis. Um dos prováveis motivos para tal é que esses devem ser documentos privados de corporações.

Outro trabalho vislumbrado é a comparação entre as tecnologias que permitem desenvolvimento de API. Comparar os resultados da API REST e de uma API SOA pode gerar conclusões interessantes e abrir espaço para otimizações de serviços.

Por último, é interessante discutir a possibilidade de se associar o ACIS e o restante da plataforma do A-CDM a um aeroporto e disponibilizar esse serviço. Com o sistema pronto, é possível também cadastrar esse sistema no SWIM Registry, para que outros parceiros interessados no domínio do ATM possam consumir e fornecer informações ao aeroporto com A-CDM.

Referências

[1] Team, EUROCONTROL Airport CDM: *THE MANUAL AIRPORT CDM IMPLEMENTATION*. EUROCONTROL. x, 21, 23, 24

[2] ORGANIZATION, INTERNATIONAL CIVIL AVIATION: *MANUAL ON SYSTEM WIDE INFORMATION MANAGEMENT (SWIM) CONCEPT*. ICAO. x, 25, 26

[3] EUROCONTROL: *Airport collaborative decision making (a-cdm) eurocontrol*. https://www.eurocontrol.int/node/10666/library, 2018. 1, 20

[4] MINISTÉRIO DOS TRANSPORTES, PORTOS E AVIAÇÃO CIVIL SECRETARIA NACIONAL DE AVIAÇ
btxfnamespacelong AO CIVIL: *ProjeÇÃo de demanda da aviaÇÃo civil 2017-2037*. http://www.transportes.gov.br/images/AEROPORTOS/ProjDemandaPress.pdf, 2017. 2

[5] DECEA: *Cooperação operacional – decea, gru airport e empresas aéreas celebram acordo para otimização das operações em guarulhos*. https://www.decea.gov.br/?i=midia-e-informacao&p=pg_noticiamateria=cooperacao-operacional-decea-gru-airport-e-empresas-aereas-celebram-acordo-para-otimizacao-das-operacoes-em-guarulhos, 2017. 2

[6] Allen, Paul e Stuart Frost: *Component-based development for enterprise systems: applying the SELECT perspective*. Cambridge University Press, 1998. 6

[7] Mark Endrei, Jenny Ang, Ali Arsanjani Sook Chua Philippe Comte Pål Krogdahl Min Luo Tony Newling: *Patterns: Service-Oriented Architecture and Web Services*. IBM Redbooks publication, 2004. 6, 7

[8] Center, IBM Knowledge: *Service-oriented architecture (soa)*. https://www.ibm.com/support/knowledgecenter/en/SSMQ79_9.5.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html/. 6

[9] Biehl, M: *API Design*, volume 2016. API University Press, 2016. 6, 7, 12, 16, 18, 27

[10] Bourque, Pierre e Richard Fairley: *SWEBOK Guide to the Software Engineering Body of Knowledge*, volume 3. IEEE, 2014. 6, 13

[11] Sim, Susan: *Next generation data interchange: tool-to-tool application programinterfaces*. Reverse Engineering, 2000, 2000. 6

[12] Gilchrist, Alasdair: *REST API Design Control and Management: DevOp Series (English Edition)*, volume 2015. RG Consulting, 2015. 7, 15

[13] Monus, Anna: *Soap vs rest vs json comparison.* https://raygun.com/blog/soap-vs-rest-vs-json/, 2018. 7

[14] Fielding, R.T: *Architectural styles and the design of network-based software architectures.* páginas 94–123, 2000. 7, 8

[15] Dambal, Vedesh: *Rest, web services, rest-ful services.* https://www.ibm.com/developerworks/library/ws-RESTservices/, 2010. 7

[16] Fielding, R, J Gettys, J Mogu, H Frystyk, L Masinter, P Leach e T Berners-Lee: *Hypertext transfer protocol – http/1.1.* https://tools.ietf.org/html/rfc2616, 1999. 9

[17] Berners-Lee, T, L Masinter e M McCahill: *Uniform resource locators (url).* https://tools.ietf.org/html/rfc1738, 1994. 10, 11

[18] Richardson, Leonard, Sam Ruby e Mike Amundsen: *RESTful Web APIs*, volume 2013. O'Reilly Media, 2013. 11, 12, 17, 27

[19] Nelson, T H: *Complex information processing: a file structure for the complex, the changing and the indeterminate.* ACM '65 Proceedings of the 1965 20th national conference, páginas 84–100, 1965. 11

[20] Tutorial, REST API: *Hateoas driven rest apis.* https://restfulapi.net/hateoas/. 12

[21] Wilde, E: *The 'profile' link relation type.* https://www.rfc-editor.org/rfc/rfc6906.txt, 2013. 12

[22] Mosqueira-Rey, E, D Alonso-Ríos, V Moret-Bonillo, I Fernández-Varela e D Álvarez Estévez: *A systematic approach to api usability: Taxonomy-derived criteria and a case study.* Information Software Technology, 2017. 12, 14

[23] Fagerholm, F e J Münch: *Developer experience: Concept and denition.* ICSSP '12 Proceedings of the International Conference on Software and System Process, páginas 73–77, 2012. 13, 14

[24] McLellan, S.G.and Roesler, A.W., J.T. Tempest e C.I. Spinuzzi: *Building more usable apis.* Reverse Engineering, 2000, 2000. 14

[25] Nielsen, J: *Introduction to usability.* https://www.nngroup.com/articles/usability-101-introduction-to-usability/, 2012. 14

[26] Scheller, T e E Kühn: *Automated measurement of api usability: the api concepts framework.* Inf. Softw. Technol., 61:145–162, 2015. 14

[27] Sommerville, Ian: *Engenharia de Software*, volume 2011. PEARSON, SP, 2011. 19

[28] ETSI: *Airport collaborative decision making (a-cdm); community specification for application under the single european sky interoperability regulation ec 552/2004, draft etsi en 303 212 v1.1.1.* https://www.etsi.org/deliver/etsi_en/303200_303299/303212/01.01.01_20/en_303212v010101c.pdf, 2009-08. 21, 28

[29] ORGANIZATION, INTERNATIONAL CIVIL AVIATION: *2013-2028 Global Air Navigation Capacity and Efficiency Plan.* ICAO. 24

[30] Barbosa, I, L Monteiro, Li Weigang, J Fregnani, I Oliveira e G Balvedi: *A proposal of a swim registry in brazil.* 25

[31] DECEA: *Ica 100-11 plano de voo.* https://publicacoes.decea.gov.br/?i=publicacao&id=4589, 2018. 31