MONOGRAFIA DE PROJETO FINAL DE GRADUAÇÃO

# ESTUDO DE CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA DE REGISTRO ELETRÔNICO DE SAÚDE

**Rodrigo Augusto Rodrigues dos Santos**

Curso Superior de Engenharia de Redes de Comunicação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA
UNIVERSIDADE DE BRASÍLIA

**UNIVERSIDADE DE BRASÍLIA**
Faculdade de Tecnologia

MONOGRAFIA DE PROJETO FINAL DE GRADUAÇÃO

# ESTUDO DE CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA DE REGISTRO ELETRÔNICO DE SAÚDE

**Rodrigo Augusto Rodrigues dos Santos**

*Monografia de Projeto Final de Graduação submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Bacharel em Engenharia de Redes de Comunicação*

Banca Examinadora

Dr. Ricardo Staciarini Puttini, EnE/UnB _____
*Orientador*

Dr. Rafael Timóteo de Sousa Júnior, EnE/UnB _____
*Examinador Interno*

Dr. Fabio Lucio Lopes de Mendonça, EnE/UnB _____
*Examinador Interno*

**FICHA CATALOGRÁFICA**

SANTOS, R.A.R.
ESTUDO DE CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA DE REGISTRO ELETRÔNICO DE SAÚDE [Distrito Federal] 2023.
xvi, 41 p., 210 x 297 mm (ENE/FT/UnB, Bacharel, Engenharia de Redes de Comunicação, 2023).
Monografia de Projeto Final de Graduação - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

| | |
|---|---|
| 1. SOA | 2. Orientação a Serviços |
| 3. Catálogo de Serviços | 4. API REST |
| I. ENE/FT/UnB | II. Título (série) |


**REFERÊNCIA BIBLIOGRÁFICA**

SANTOS, R.A.R. (2023). *ESTUDO DE CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA DE REGISTRO ELETRÔNICO DE SAÚDE*. Monografia de Projeto Final de Graduação, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 41 p.

**CESSÃO DE DIREITOS**

AUTOR: Rodrigo Augusto Rodrigues dos Santos
TÍTULO: ESTUDO DE CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA DE REGISTRO ELETRÔNICO DE SAÚDE.
GRAU: Bacharel em Engenharia de Redes de Comunicação ANO: 2023



Rodrigo Augusto Rodrigues dos Santos
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP: 70919-970 - Brasília-DF - Brasil

*À minha família de sangue e àquela que eu pude escolher: pai, mãe, Tayara e amigos, vocês são meu sustento e inspiração de buscar, a cada dia, ser um pouco melhor do que eu já fui.*

## AGRADECIMENTOS

Terminar um ensino superior em uma universidade federal de tamanho porte como a UnB é de imensa alegria para qualquer pessoa. Sinônimo de oportunidade e realização de um sonho, a conclusão do curso de Engenharia trás muitas lembranças do caminho trilhado até chegar aqui. Sacrifícios, dedicação, alegrias, amizades, noites mal dormidas, ansiedade com provas, mas também a alegria de passar naquela matéria difícil e saber que sou capaz.

Nesse caminho trilhado até esse momento, contei com diversas pessoas que me apoiaram e não desistiram de mim, mesmo muitas vezes eu mesmo desacreditando do meu potencial e capacidade de superação, a todos vocês o meu muito obrigado.

Agradeço, primeiramente, a Deus, por essa conquista tão desejada e que, sustentado e conduzido por Ele, pude alcançar. A todos os meus queridos familiares, em especial meus avôs, que se encontram junto d'Ele, agradeço a intercessão e a presença nesta jornada.

Agradeço aos meus pais, que sempre me deram todo o necessário para construir minha personalidade e correr atrás dos meus sonhos, sem reter nada. Agradeço a minha família, que, sempre unida, incentivou e comemorou todas as minhas conquistas.

Agradeço aos meus amigos, em especial aqueles mais próximos, que acompanharam de perto toda essa jornada e seus momentos desafiantes, e nunca deixaram de me incentivar e acolher quando precisei.

E, claro, agradeço imensamente à minha companheira de vida, que acompanhou essa jornada quase desde o seu começo e me ensinou a ser mais forte, persistente e paciente comigo mesmo e com a vida acadêmica. Tayara Laíse, você foi e sempre será fundamental na trajetória da minha vida e espero fortemente poder dividir contigo essa e todas as outras conquistas que virão.

Por fim, agradeço a meus professores durante o curso, em especial ao meu orientador Prof. Dr. Ricardo Puttini que me ajudou a realizar o sonho de me tornar Engenheiro e conhecer um mundo diferente dentro da tecnologia.

Este trabalho contou com suporte do Laboratório de Tecnologias da Tomada de Decisão da Universidade de Brasília (LATITUDE/UnB), que tem apoio do CNPq - Conselho Nacional de Pesquisa (Outorgas 312180/2019-5 PQ-2 e 465741/2014-2 INCT em Cibersegurança) e da Universidade de Brasília (Outorga FUB/COPEI 7129).

*O caminho se faz ao caminhar...*

## RESUMO

A Arquitetura Orientada a Serviços (*SOA*) trás uma série de paradigmas com o objetivo de implementar as suas características dentro de um contexto de Tecnologia da Informação. O catálogo ou inventário de serviços faz parte de um desses paradigmas, trazendo um conjunto de serviços oferecidos por organizações e empresas de tecnologia, sendo os mesmos agnósticos, ou seja, reutilizáveis e, assim, facilitando o seu uso por demais desenvolvedores e/ou venda de serviços da empresa. Este trabalho visa estudar um catálogo de serviços de uma plataforma de Registro Eletrônico de Saúde, utilizando ferramentas *open source* da WSO2 e documentações *swagger* das APIs/serviços disponíveis para, em um ambiente de testes, montar e entender o funcionamento, características e benefícios da catalogação de serviços. Os resultados se concentram no estudo da disponibilização dos serviços através das ferramentas utilizadas, além de testes de uso e consumo dos serviços publicados.



**Palavras-chave: SOA, Orientação a Serviços, Catálogo de serviços, Swagger, WSO2, API REST.**

## ABSTRACT

Service Oriented Architecture (SOA) brings a series of paradigms with the objective of implementing its characteristics within an Information Technology context. The service catalog or service inventory is part of one of these paradigms, bringing a set of agnostic (reusable) services offered by organizations and technology companies, facilitating their use by other developers and/or sales department. This work aims to study a service catalog of an Electronic Health Record platform, using open source tools from WSO2 and Swagger documentations of the available APIs/services to, in a test environment, assemble and understand the functioning, characteristics and benefits of cataloging services. The results focus on the study of the availability of services through the used tools, in addition to tests of use and consumption of the published services.



**Keywords: SOA, Service Catalog, Service Orientation, REST API, Swagger, WSO2**

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SÍMBOLOS

## Siglas

| | |
|---|---|
| API | *Application Programming Interface* |
| APIM | *API Manager* |
| AWS | *Amazon Web Services* |
| CIAM | *Customer identity access management* |
| COD | *Code on Demand* |
| CORS | *Cross-origin resource sharing* |
| HTTP | *Hypertext Transfer Protocol* |
| IP | *Internet Protocol* |
| IS | *Identity Server* |
| JDK | *Java Development Kit* |
| JSON | *JavaScript Object Notation* |
| RAM | *Random Access Memory* |
| RDS | *Relational Database Service* |
| REST | *Representational State Transfer* |
| SFTP | *SSH File Transfer Protocol* |
| SGPO | *SOA Governance Program Office* |
| SO | Sistema Operacional |
| SOA | *Service Oriented Architecture* |
| SOAP | *Simple Object Access Protocol* |
| UDDI | *Universal Description, Discovery and Integration* |
| VM | *Virtual machine* |
| WSA | *Web Service Architecture* |
| WSDL | *Web Services Description Language* |
| XML | *Extensible Markup Language* |
| YAML | *YAML Ain't Markup Language* |

# 1 INTRODUÇÃO

A Arquitetura Orientada a Serviços (SOA), segundo [Hat 2023], "é um tipo de design de software que torna os componentes reutilizáveis usando interfaces de serviços com uma linguagem de comunicação comum em uma rede".

No atual crescimento e utilização de serviços *web*, componentes reutilizáveis e interfaces com linguagem comum são bastante atrativas e trazem inúmeros benefícios para diversas aplicações.

Um serviço agnóstico e, portanto, reutilizável, economiza tempo e gastos financeiros ao se desenvolver um novo *software* ou plataforma que necessite do mesmo serviço, podendo o mesmo ser utilizado por outros desenvolvedores e plataformas, se assim o for desejado.

Sendo assim, catalogar e disponibilizar serviços ajudam tanto a empresa, em sua organização de serviços oferecidos e possível transmissão de conhecimento a colaboradores, quanto a tecnologia em si, com mais agilidade no desenvolvimento de aplicações que necessitem de serviços já implementados.

Além disso, um catálogo de serviços bem elaborado pode ser útil quanto a parte de vendas da organização. Ao se saber exatamente os serviços disponíveis e o que os mesmos produzem, sabe-se a dor do cliente a ser sanada e, sendo assim, a conversa flui de uma melhor forma.

Cabe trazer a este trabalho um estudo sobre a importância de um catálogo de serviços, especificamente para uma plataforma de registro eletrônico de saúde, mas podendo se estender futuramente a qualquer serviço de interoperabilidade.

Diversos softwares e soluções podem ser utilizados para esta implementação, a solução escolhida é amplamente utilizada no mercado e há muito tempo disponível de forma gratuita e aberta.

É importante, em primeiro lugar, conhecer o inventário de serviços, dentro da arquitetura orientada a serviços, e entender como o mesmo pode contribuir, de diversas formas, no desempenho e alcance de objetivos dentro do contexto de Tecnologia da Informação.

## 1.1 OBJETIVOS

### 1.1.1 OBJETIVOS GERAIS

O objetivo geral deste trabalho é entender a utilização e importância da utilização de um catálogo de serviços dentro de uma plataforma de registro eletrônico de saúde, no contexto da arquitetura orientada a serviços (*SOA)*, através de uma ferramenta *open-source* de exposição e criação do catálogo de serviços.

### 1.1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos tem como teor dar sentido ao objetivo geral deste trabalho. Os requisitos, processos e descrições a seguir visam implementar em um ambiente de testes e analisar a importância de um catálogo de serviços em uma plataforma de registro eletrônico de saúde. Desta forma, objetiva-se:

1.2.2.1 Entender e estudar a importância do catálogo e da catalogação de serviços dentro de uma Arquitetura Orientada a Serviços (*SOA*);

1.2.2.2 Implementar, em um ambiente de testes na núvem, um catálogo de serviços para a plataforma de registro de atendimento eletrônico, utilizando produtos da WSO2;

1.2.2.3 Estudar como o catálogo de serviços se comporta e como pode ser utilizado por um desenvolvedor de fora da plataforma;

1.2.2.4 Analisar o funcionamento do consumo de serviços no portal onde os mesmos estão disponibilizados.

## 1.2 ESTRUTURA DOCUMENTAL

Este documento é um projeto que visa o estudo de um catálogo de serviços de uma plataforma de registro eletrônico de saúde, implementando o mesmo em um ambiente de testes em núvem para analisar o seu comportamento e utilização. O mesmo se encontra segmentado em sete capítulos sendo que:

- O primeiro capítulo é refente à introdução, com o objetivo principal de ralizar uma abordagem introdutória e descritiva das motivações, primeiros estudos, conceitos e objetivos do trabalho;
- O segundo capítulo se refere a fundamentação teórica do trabalho, e tem o objetivo de realizar um estudo mais a fundo dos conceitos utilizados na concepção deste estudo;
- O terceiro capítulo é dedicado a descrição das ferramentas utilizazdas para o desenvolvimento do projeto, visando obter os objetivos descritos anteriormente;
- O quarto capítulo retrata a arquitetura do projeto, com o seu desenho topológico e a ideia por tras das tecnologias e serviços utilizados e implementados para este estudo;
- O quinto capítulo apresenta os testes realizados e os seus respectivos resultados dentro dos objetivos propostos;
- O sexto capítulo apresenta uma análise conclusiva dos resultados obtidos, os relacionando com os objetivos deste estudo, propostos na seção 1.1.2 .
- O sétimo e último capítulo objetiva apresentar sugestões de trabalhos futuros relacionados a este estudo, mas não limitando novas contribuições.
- Além dos sete capítulos apresentados, há ainda um anexo com instruções sobre a instalação e configuração de alguns pré-requisitos dos produtos utilizados;

# 2 FUNDAMENTAÇÃO TEÓRICA

A descrição deste capítulo tem como foco nortear a teoria das tecnologias adjacentes, utilizadas dentro da proposta e dos objetivos do projeto. Dessa forma, visa realizar uma revisão de conceitos, terminologias e fundamentos teóricos necessários para o desenvolvimento e obtenção dos resultados finais.

## 2.1 ARQUITETURA ORIENTADA A SERVIÇOS (SOA)

A Arquitetura Orientada a Serviços define um padrão de projeto ou arquitetura de software, visando fazer os seus componentes reutilizáveis através de interfaces de serviço.

Segundo [Erl 2008] "um dos aspectos mais desafiadores de escrever ou discutir tecnologia é o uso das terminologias pela indústria". A indústria da tecnologia cunha e ressignifica termos a todo momento, devido à sua constante mudança e evolução, o que trás um desafio de os definir sem ambiguidade.

Dessa forma, antes de se aprofundar nos detalhes da Computação Orientada a Serviços, é importante definir alguns conceitos importantes para o seu entendimento, sendo os mesmos citados e explicados a seguir.

- **Característica de Design (*Design Characteristic*)**:

Uma característica é um atributo ou qualidade de algo. [Erl 2008] nos diz que soluções empresariais automatizadas possuem diversas características em sua arquitetura inicial, assim como a Orientação a Serviços, que enfatiza algumas características, enquanto outras são colocadas como menos importantes.

Todos os sistemas possuem características específicas, mas a ideia principal em estudar as características da Arquitetura Orientada a Serviços é compreender e estabelecer as suas características comuns, que se repetem em todos os sistemas com essa arquitetura.

- **Princípio de Design (*Design Principle*)**:

Um princípio é uma prática generalizada, já aceitada e validada pela indústria. Ou seja, é algo que outras pessoas e empresas já fazem e promovem, de acordo com um objetivo comum.

Quando se fala de construir soluções, [Erl 2008] diz que: "um princípio de design representa um guia extremamente recomendado para se desenvolver uma solução lógica em uma forma específica, com certos objetivos em mente". Ou seja, um princípio, nesse caso, é uma forma já aceita e validada pela indústria de se implementar uma característica desejada de uma arquitetura.

- **Paradigma de Design (*Design Paradigm*)**:

O termo “paradigma” tem muitos significados associados, que podem ser desde uma abordagem específica de algo a um conjunto de regras que são aplicadas dentro de um limite especificado.

[Erl 2008] diz que um paradigma de design, no contexto de automação de serviços, geralmente é uma abordagem para se desenvolver uma solução lógica e que, normalmente, consiste em uma série de regras e princípios complementares que são definidos previamente para se obter um resultado desejado.

A Orientação a Objeto é um exemplo de paradigma muito bem difundido e aceito na indústria, e consiste, dessa forma, em uma série de princípios a serem seguidos para se obter uma solução lógica dentro dos objetivos do paradigma.

Sendo assim, a Orientação a Serviços também possui um paradigma, que define alguns princípios a serem seguidos e características a serem implementadas, objetivando a implementação dessa arquitetura.

- ***Design Pattern***:

Nesse ponto, Thomas Erl [Erl 2008] nos diz que a Orientação a Serviço é, então, um paradigma de design, composto por uma série de princípios de design, cada um focado em implantar uma certa característica, utilizando todos os termos definidos anteriormente.

O paradigma em si já parece bastante completo, mas para uma aplicação real da Orientação a Serviço, a teoria não é suficiente, se faz necessário um padrão de design, para que os desafios encontrados na hora de implementar o paradigma sejam melhor superados.

[Erl 2008] diz que um padrão de design descreve um problema comum e, consequentemente, uma solução para o mesmo. Assim, ele documenta um padrão de resolução do problema identificado, de forma genérica, para que o mesmo possa ser aplicado repetidamente.

Os padrões de design são encontrados quando um paradigma bem pensado é colocado em prática, dessa forma, são pensados para o mundo real, com os desafios e erros encontrados nas tecnologias atuais, tornando a implantação do paradigma possível.

- ***Design Standard***:

[Erl 2008] explica que, cada organização tem um conjunto de princípios, estratégias e objetivos na hora de implantar um paradigma e, assim sendo, um conjunto diferente de requisitos precisam ser implementados junto com cada solução.

Dentro desse contexto, ainda nos diz que os padrões de design são convenções, geralmente obrigatórias, usadas para determinar previamente as características do design da solução, dentro dos objetivos e estratégias da empresa.

"É através do uso de padrões de design internos que organizações podem, consistentemente, entregar soluções sob medida para os seus ambientes, recursos, metas e prioridades."[Erl 2008]

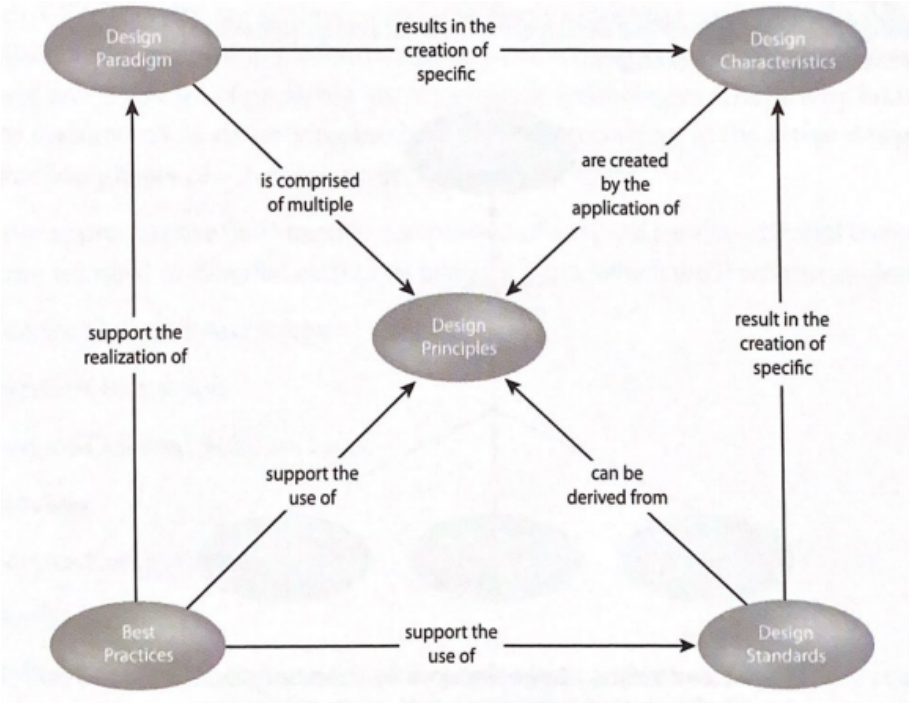



Figura 2.1: Relações entre as partes de um *Design Framework* Fonte: [Erl 2008]

### 2.1.1 Design e Arquitetura

As tecnologias e paradigmas surgem à medida que são encontrados novos desafios e necessidades dentro de um negócio e na sociedade, para ajudar a resolver um problema ou tornar a vida mais fácil, rápida e eficiente.

A Computação Orientada a Serviço resolve problemas e facilita muitos processos dentro de uma organização, representa uma nova geração da computação distribuída e possui diversas características, como o seu próprio paradigma de design, princípios, linguagens, modelo de arquitetura, tecnologias, etc

- ***Service Oriented Architecture***:

"SOA estabelece um modelo de arquitetura que visa aumentar a eficiência, agilidade e produtividade de um empreendimento ao posicionar serviços como meios primários através dos quais a solução lógica é representada, para o cumprimento de objetivos estratégicos associados à computação orientada a serviço."[Erl 2008]

[Erl 2008] diz que a computação orientada a serviço gira em torno do paradigma de design e o seu relacionamento com a arquitetura. O termo que define a arquitetura orientada a serviço (SOA) foi amplamente difundido na indústria como sinônimo da computação orientada a serviço em si, o que trás a importância de se distinguir o que SOA realmente é e como se relaciona com os outros componentes da Computação Orientada a Serviço.

Segundo [Erl 2008]: “como uma forma de arquitetura de tecnologia, uma implementação SOA pode conter uma combinação de diferentes tecnologias”. Ou seja, como uma arquitetura que implementa um

paradigma, são utilizadas diferentes tecnologias disponíveis atualmente para o seu sucesso, como APIs, diferentes ambientes de infraestrutura e outras partes do sistema.

- ***Serviços e Composições de Serviços***:

"Os serviços existem como programas de software fisicamente independentes com características de design distintas que suportam a obtenção dos objetivos estratégicos associados à SOA"[Erl 2008]. Ou seja, serviços são softwares independentes, que executam uma tarefa específica visando alguma característica de design da computação orientada a serviços.

[Erl 2008] diz que cada serviço tem o seu contexto próprio, e está atrelado a certas capacidades relacionadas e esse mesmo contexto. Essas características que o serviço oferece geralmente são expressas através de um contrato de serviço, como uma API tradicional.

Diante da definição de serviço em si, uma composição de serviços será um conjunto agregado e coordenado de serviços entre si. Seria, conforme [Erl 2008], análogo a uma aplicação tradicional, onde o seu escopo funcional é, geralmente, associado com a automação de um processo pai, que controla processos filhos para seu correto funcionamento.

"A aplicação consistente dos princípios de design de orientação a serviços leva à criação de serviços com contextos funcionais que são independentes de qualquer outro processo de negócios". [Erl 2008]

Sendo assim, cada serviço é pensado de forma agnóstica, com funcionamento independente de outro serviço, mas que pode ser agregado e utilizado conjuntamente, formando uma composição de serviços para a implementação de uma arquitetura desejada.

- ***Inventário de Serviços***:

Um inventário de serviços [Erl 2008] é uma coleção de serviços complementares, padronizada de forma independente, que representa um negócio ou parte dele.

Dessa forma, dentro de um empreendimento, o inventário de serviços inclui todos os serviços agnósticos do produto oferecido, que foi implementado com SOA. Um empreendimento pode ter diversos inventários de serviço, para diferentes produtos ou arquiteturas.

"Os inventários de serviço são normalmente criados por meio de processos de entrega top-down, que resultam na definição de esquemas de inventário de serviço. a aplicação subseqüente de princípios de design de orientação a serviço e padrões de design personalizado em todo um inventário de serviço é de suma importância para estabelecer um alto grau de interoperabilidade nativa entre serviços."[Erl 2008]

### 2.1.2 Governança

A expectativa, ao se adotar *SOA* em um empreendimento, é de se utilizar de seus princípios para obter alguns benefícios para o negócio e sua forma de organização. Para se obter esses benefícios, portanto, são necessárias outras abordagens além da implementação técnica dos princípios descritos anteriormente.

Conforme [Erl et al. 2011], uma governança estruturada é necssária para a realização dos compromissos assumidos ao se adotar a arquitetura orientada a serviços. Isso se dá pois ela ajuda as organizações a terem sucesso na implementação SOA, adotando esforços para mitigar riscos por meio de restrições predefinidas, regras e a alocação de autoridades necessárias.

Um sistema de governança existe como um meio de uma organização realizar decisões, dentro de qualquer escopo, inclusive na forma de realizar decisões. Dentro desse contexto, segundo [Erl et al. 2011], um sistema de governança:

- impõe restrições a decisões a serem tomadas;
- determina quem tem a responsabilidade e autoridade para tomar decisões;
- estabelece restrições e paramêtros que controlem, guiem ou influenciem decisões;
- determine as consequências para o não cumprimento das regras definidas;

Um bom sistema de governança permite, portanto, que os membros de uma organização possam tomar e seguir decisões de forma a ajudar a mesma a realizar seus objetivos e concretizar a sua visão de negócio, além de ajudar na melhora de conflitos, ao definir claramente as autoridades de quem deve tomar decisões.

Dentro do contexto de Tecnologia da Informação, um sistema de governança é responsável, conforme [Erl et al. 2011], por "prover, organizar, direcionar e guiar a criação de recursos e ativos de tecnologia".

Um sistema de governança serve, portanto, para mitigar riscos e ajudar uma organização a cumprir suas metas, objetivos, prioridades e visão. Ao se investir em alguma tecnologia ou produto, um dos objetivos de qualquer organização é de obter mais benefícios do que o custo de investimento, o que não é diferente ao se implementar *SOA*.

Sendo assim, o principal objetivo da governança na arquitetura orientada a serviço é garantir que o empreendimento obtenha seus objetivos e que os benefícios obtidos superem os custos de implantação.

"Um sistema de governança *SOA* é um sistema de meta-decisão que uma organização coloca em prática para obter o controle e restrição das responsabilidades de tomada de decisão relativas à adoção e aplicação da orientação a serviço". [Erl et al. 2011]

Há muitas formas de se implementar um sistema de governança *SOA*, como modelos e frameworks, porém seus fundamentos residem em um *SOA Governance Program Office (SGPO)*, que define os modelos e tarefas necessárias para a implementação de uma governança *SOA*.

O *SGPO* define uma equipe responsável pelo sistema de governança *SOA*, garantindo que o mesmo seja devidamente implantado. Algumas das suas diretrizes básicas em uma ambiente de TI são, conforme [Erl et al. 2011]:

- O *SGPO* deve ter a responsabilidade e autoridade para desenvolver e gerenciar o sistema de governança *SOA* e outras equipes devem reconhecer a sua autoridade;
- O *SGPO* deve garantir que o sistema de governança *SOA* esteja alinhado com os sistemas de incentivo e disciplinares da organização;

- O *SGPO* deve desenvolver relações de trabalho colaborativas com outras equipes de governança, que tenham responsabilidades em comum com o *SGPO*;
- O *SGPO* deve garantir que outros sistemas de governança dentro da organização trabalhem em conjunto e evitem conflitos;
- O *SGPO* deve ter acesso a canais de comunicação para disseminar informações sobre os princípios da governança e prover treinamento adequado aos colaboradores que necessitem;

Dessa forma, o *SGPO* deve cooperar e co-existir de forma harmoniosa com outros departamentos de governança da organização.

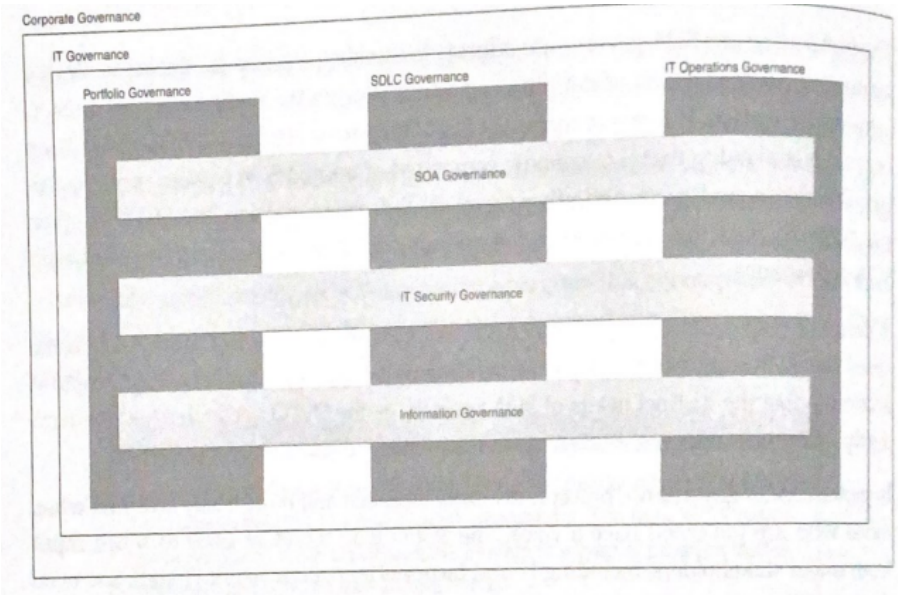



Figura 2.2: A governança *SOA* deve co-existir com outros programas ded governança de TI Fonte: [Erl et al. 2011]

Há vários modelos distintos de oganização do *SGPO*, como o centralizado, federado e independente, que podem ser adotados a depender da estrutura e organização do empreendimento.

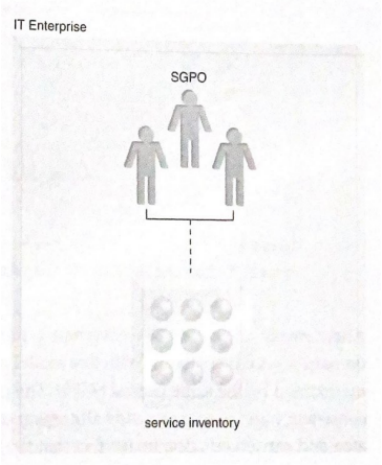



Figura 2.3: Exemplo de *SGPO* para gerenciar um inventário de serviços. Fonte: [Erl et al. 2011]

## 2.2 CATALOGAÇÃO DE SERVIÇOS

Dentro da Arquitetura Orientada a Serviços (*SOA*) existe o inventário de serviços, que consiste em uma coleção de serviços complementares, organizada de forma padronizada, e que represente uma solução oferecida por um negócio.

Segundo [O'Loughlin 2010], [Gama Maria do Mar Rosa 2013] e autores citados, as organizações e empresas de Tecnologia possuem um problema, atualmente, de muitas vezes não conseguir identificar os serviços que as mesmas provem, além de entender os diferentes tipos de serviço disponíveis.

A forma de interação entre empresas e clientes mudou de forma radical ao longo dos anos, acompanhando o avanço tecnológico imenso da atualidade. Um catálogo de serviços se torna, portanto, uma ferramenta útil e importante no relacionamento cliente-empresa, facilitando a organização dos serviços disponíveis e seus casos de uso.

A indústria de software, conforme [O'Loughlin 2010], tem acompanhado essa tendencia, onde diversos sistemas de gerenciamento de serviços trazem, cada vez mais, ferramentas e módulos de catalogação de serviços, dentro da sua oferta principal, ou como um módulo a ser licenciado.

As novas tecnologias e o desenvolvimento tecnológico em si surgem para resolver problemas ou facilitar a vida de quem as usa. Para que isso seja possível dentro do modelo social que vivemos, a Tecnologia da Informação, como diz [O'Loughlin 2010], deve estar devidamente alinhada com a indústria e posicionada estrategicamente como um ponto chave para alcançar resultados satisfatórios e trazer um bom retorno financeiro dentro de uma organização.

Dessa forma, para trabalhar em conjunto, a tecnologia deve resolver um problema, uma dor do consumidor, e a indústria busca sempre estar a frente das necessidades do mesmo, tratando as suas constantes mudanças para conseguir oferecer um serviço útil e que gere resultados positivos para o empreendimento. Sendo assim, segundo [O'Loughlin 2010], o catálogo de serviços fornece à indústria da Tecnologia da Informação a capacidade de mostrar à organização os serviços que a mesma fornece, mas também o processo de negócios e os serviços do cliente suportados e fornecidos pela organização.

"O catálogo de serviços fornece aos usuários e clientes os meios de entender quais serviços eles podem realmente usar. Diferentes visualizações do catálogo de serviços podem fornecer detalhes e informações de serviços em um formato que seja compreendido pelo público relevante. O catálogo de serviços é a única parte do portfólio geral de serviços que pode recuperar custos ou obter lucros". [O'Loughlin 2010]

### 2.2.1 A importância da Catalogação

A catalogação de serviços tem inúmeros objetivos e benefícios dentro de uma organização, desde o melhor conhecimento dos serviços e soluções oferecidos pela empresa a um potencial aumento nos ganhos e lucros da mesma.

Alguns dos benefícios da elaboração de um catálogo de serviços, conforme [O'Loughlin 2010], são:

- aumenta a visibilidade e consciência dos serviços oferecidos;

- melhora a satisfação dos clientes;
- identifica pontos críticos do sistema, otimizando os recursos alocados quando necessários;
- reduz ineficiências e gastos desnecessários;
- permite a usuários e clientes escolherem os serviços realmente necessários à resolução dos seus problemas;
- dentre outros.

Além disso, um catálogo de serviços bem elaborado pode ser consultado e gerenciado pelo time técnico de um negócio, entendendo os pontos críticos e que exigem um maior cuidado quanto à manutenção do serviço, além da sua forma e casos de uso, para uma melhor integração durante o desenvolvimento de alguma aplicação.

Sendo assim, um catálogo de serviço tem benefícios para diversas áreas de um negócio, tanto na parte técnica, com um melhor conhecimento dos seviços oferecidos e suas necessidades, quanto na parte negocial, ao apresentar e disponibilizar os mesmos serviços aos potenciais clientes do empreendimento.

## 2.3 ARQUITETURA REST

### 2.3.1 *APIs*

Uma *Application Programming Interface (API)*, conforme [FERREIRA 2021], consistem em um "conjunto de rotinas e padrões de programação, que possuem o objetivo de acessar aplicativos de software ou plataformas baseadas na web". Ou seja, *APIs* são diversas ferrmantas, inclusive códigos, protocolos e métodos de desenvolvimento que facilitam a comunicação entre programas e aplicações.

As *APIs* não são algo recente na indústria da tecnologia da informação, já existem como interfaces de comunicação entra aplicações e serviços há décadas. Porém, o seu papel tem mudado de forma relevante nos últimos anos.

Segundo [Patni 2017], organizações inovadoras descobriram que as *APIs* podem ser utilizadas como uma interface para negócios, permitindo a monetização de recursos digitais. Isso acontece pois, ao se criar uma *API*, se permite que outras partes, dentro ou fora da organizazção, tenham acesso e possa utilizar de um serviço ou produto em diversas ocasiões.

Dessa forma, segundo [FERREIRA 2021] e autores citados, "uma API permite que sua solução ou serviço se comunique com outros produtos e serviços sem precisar saber como eles foram implementados. Isso simplifica o desenvolvimento de aplicações, gerando economia de tempo e dinheiro. Ao desenvolver novas ferramentas e soluções (ou ao gerenciar aquelas já existentes), as APIs oferecem a flexibilidade necessária para simplificar o design, a administração e o uso, além de fornecer oportunidades de inovação".

Algumas das características de uma API, conforme [FERREIRA 2021], são:

- Pode ser de acesso público, restrito ou privado;

- É baseada em uma arquitetura cliente-servidor e no protocolo HTTP;
- Conecta aplicações e disponibiliza serviços;
- São padrões disponibilizados para desenvolvedores;

### 2.3.2 *Web Services*

*Web Services*, segundo [Subramanian e Raj 2019], são métodos de comunicação entre dois dispositivos ou computadores através da Internet ou, em outras palavras, da *Web*. Essa comunicação ocorre de uma forma padronizada e especificada, para integrações entre aplicações distintas usando *XML/JSON, SOAP, WSDL e UDDI*.

O *XML* ou *JSON* dizem respeito ao formato de dados trocados entre os sistemas, *SOAP* é um protocolo de comunicação, desenvolvido para interligar aplicações heterogêneas. O *WSDL* é usado para definir os serviços disponíveis e que podem ser consumidos e, por fim, o *UDDI* define a lista de serviços disponíveis.

A arquitetura de *Web Services*, conforme [Subramanian e Raj 2019], demanda a presença de algumas características, sendo algumas delas opcionais, ao se desenvolver qualquer *web service*. Essa arquitetura tem três partes principais, que são: o provedor de serviços (*service provider*), o consumidor de serviços (*service consumer*) e o *service broker*.

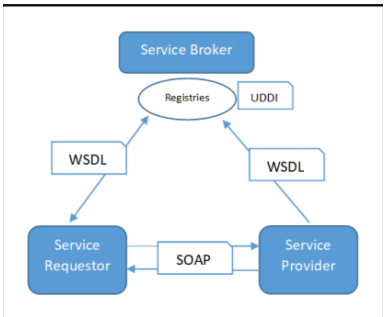



Figura 2.4: Exemplo de arquitetura de *Web Services*. Fonte: [Subramanian e Raj 2019]

Dessa forma, dentro da arquitetura de serviços web funciona com o *service consumer* encontrando o *service provider* através do UDDI e a comunicação entre os dois se dá através de SOAP. O provedor, então, valida a requisição e envia uma resposta em formato XML ou JSON.

Após discutirmos os principais fundamentos da arquitetura de serviços web, cabe definir o conceito de *web API*. Segundo [Subramanian e Raj 2019], uma *Web API* é uma API que funciona para um servidor web ou navegador.

Dessa forma, a API Web é um "conceito ou metodologia para acessar qualquer API (disponível na *web*) através do protocolo HTTP" [Subramanian e Raj 2019]. Existem muitas categorias de APIs, como SOAP e REST, e as mesmas podem ser desenvolvidas em diversas linguagens de programação.

Assim, a API web está disponível tanto do lado do cliente quanto do servidor, podendo ser exposta do lado do cliente como um plugin de navegador, ou um script, e, do lado do servidor, geralmente através de JSON ou XML.

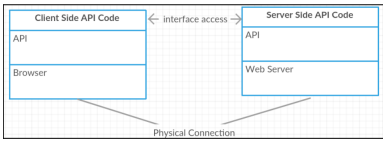



Figura 2.5: Exemplo de arquitetura de *Web API*. Fonte: [Subramanian e Raj 2019]

Ao ser baseada no protocolo HTTP, a API Web pode possuir algumas operações básicas baseadas no mesmo protocolo. Algumas dessas principais operações e uma breve descrição das mesmas estão demonstradas na tabela que segue:

Tabela 2.1: Operações HTTP

| Operação HTTP | Descrição |
|---|---|
| GET | Lê as representações de recursos |
| PUT | Cria um novo recurso |
| DELETE | Deleta um recurso |
| POST | Modifica um recurso |
| HEAD | Meta-informações sobre o recurso |

### 2.3.3 REST

O princípio fundamental da arquitetura REST, conforme [Subramanian e Raj 2019], é utilizar o protocolo HTTP para comunicação de dados entre sistemas distribuídos, o que gira em torno do conceito de recursos, onde todo e qualquer componente disponível é considerado um recurso, e os mesmos são acessados através das interfaces comuns utilizando os métodos HTTP.

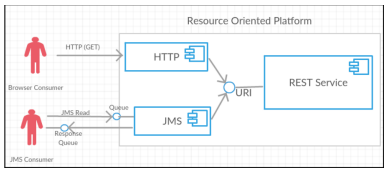



Figura 2.6: Exemplo de serviço REST. Fonte: [Subramanian e Raj 2019]

"REST é um padrão de arquitetura, e não uma linguagem de programação ou tecnologia. Ele fornece diretrizes para sistemas distribuídos se comunicarem diretamente usando os princípios e protocolos existentes na web e nas APIs, sem a necessidade da utilização de SOAP ou qualquer outro protocolo sofisticado". [Subramanian e Raj 2019]

Dessa forma, REST existe como um modelo arquitetural, trazendo uma maior simplicidade no acesso aos recursos disponíveis no modelo web definido anteriormente, com uma melhor comunicação entre sistemas distribuídos e melhor integração de recursos e serviços. Os recursos podem ser representados por diversos tipos diferentes, como XML, JSON, textos, imagens etc.

A arquitetura REST possui algumas restrições, dentro de uma aplicação baseada em web. São elas:

- **Cliente-Servidor (*Client-Server*):** a arquitetura cliente-servidor separa os componentes entre cliente

e servidor, onde o cliente é o componente que requisita um ou vários serviços ao servidor, e o servidor provê serviços aos clientes;

- **Comunicação *Statelessness*:** Uma requisição feita por um cliente a um servidor deve conter, segundo [REST: Princípios e boas práticas], todas as informações necessárias para que o servidor as interprete e as execute corretamente. Os clientes não devem depender de dados previamente armazenados no servidor para processar uma requisição, ou seja, o estado anterior não deve ser guardado;
- ***Cacheable:*** Diz respeito, segundo o livro, à habilidade de guardar dados frequentemente acessados (uma resposta) para poder responder com rapidez aos clientes, e não sendo necessário gerar a mesma resposta mais de uma vez de forma desnecessária;
- **Interface Uniforme:** A arquitetura REST utiliza de um vocabulário comum ao protocolo HTTP e, portanto, a grande parte das comunicações web, mantendo, assim, uma interface uniforme de comunicação juntamente com o HTTP;
- **Sistemas em camadas:** "Em geral, um sistema em camadas consiste em camadas com diferentes unidades de funcionalidade. As características essenciais de uma sistema de camadas é que a comunicação entre as mesmas se dá apenas com uma camada acima ou abaixo da atual. Camadas podem ser adicionadas, removidas, modificadas ou reordenadas de acordo com a evolução da arquitetura"; [Subramanian e Raj 2019]
- **Código por demanda:** Na computação distribuída, conforme [Subramanian e Raj 2019], *code on demand (COD)* é qualquer tecnologia que permita ao servidor enviar códigos como resposta para serem executados no computador do cliente, após uma requisição do software do cliente;

Dessa forma, conforme [Subramanian e Raj 2019], a arquitetura REST traz um conjunto de propriedades que ajudam a estabelecer os seus objetivos, embasados nas suas restrições. As suas propriedades são:

- Performance;
- Escalabilidade;
- Simplicidade;
- Modificabilidade;
- Visibilidade;
- Portabilidade;
- Confiabilidade;
- Testabilidade;

Em suma, REST é uma arquitetura baseada em serviços web que utiliza o protocolo HTTP como interface comum, com uma comunicação baseada em cliente-servidor e que busca simplificar e melhorar a comunicação entre sistemas distribuídos, ao implementar algumas propriedades dentro de suas restrições definidas.

# 3 FERRAMENTAS UTILIZADAS

A descrição deste capítulo objetiva detalhar as ferramentas utilizadas no desenvolvimento deste projeto, visando o entendimento do funcionamento de cada ferramenta e a sua aplicabilidade nos objetivos propostos.

## 3.1 WSO 2

### 3.1.1 *API - MANAGER*

O WSO 2 API Manager é uma ferramenta de código aberto que oferecece uma plataforma completa de gerenciamento de APIs que suporta, segundo [WSO2], o design e publicação de APIs, gerenciamento do ciclo de vida, desenvolvimento de aplicações, segurança de APIs, limitação de taxa, visualização de estatísticas das APIs além da interconexão entre as mesmas, outros produtos e *endpoints*.

Os produtos oferecidos pela WSO2 podem ser utilizados para diversos fins e disponibilizam uma gama enorme de soluções corporativas, dentre elas a de funcionar como uma ferramenta de disponibilização de APIs através de um portal do desenvolvedor, principal função necessária para o desenvolvimento deste trabalho.

Algumas das principais capacidades do produto são, conforme [WSO2]:

- **Desenvolvimento, implantação e gerenciamento de APIs/Produtos de API:** Uma API bem desenvolvida a faz mais fácil de ser utilizada. o WSO2 APIM possui um *publisher* que auxilia em todo o processo de criação e publicação da API, utilizando as suas especificações definidas. Mais informações sobre essa função podem ser encontradas em *Design API Overview*
- **Integração orientada por API:** Uma estratégia de intergração orientada por APÌ pode ser facilmente implementada, utilizando a camada de gerenciamento de API e a de integração da plataforma do produto. *API-led Integration*
- **APIs encontráveis:** Tornar as APIs de um empreendimento encontráveis pode ajudar a aumentar a sua base de clientes. O WSO2 API Manager Publisher pode ser utilizado para criar categorias para as APIs através de *tags*. O portal onde as mesmas ficam expostas possui uma ferramenta de pesquisa de textos, que auxilia na busca de alguma API desejada. Make your APIs Discoverable
- **APIs seguras:** Controle de visibilidade, proteção contra ameaças, validação de *payload* de APIs, políticas de limite de taxas, dentre outros. API Security
- **Informações completas sobre as APIs:** O API Manager conta com uma ferramenta de análise de dados, com *dashboards* customizáveis para acompanhar as informações importantes de uma API. API Manager Analytics

O *API Manager* possui diversas outras funcionalidades. Para os objetivos propostos, o mesmo foi utilizado para um teste de exposição das APIs disponibilizadas pela empresa de forma pública, com o gerenciamento de autenticação de usuários integrado com o *Identity Server*.

### 3.1.2 *IDENTITY SERVER*

O *WSO2 Identity Server* é uma ferramenta de código aberto, parte do pacote da WSO2 de *CIAM (Customer Identity Acess Management)*, que consiste em uma solução de gerenciamento de acesso de usuário, implementado para uma maior segurança dentro de uma organização, com diversas possibilidades de integração.

Os diversos casos de uso e funcionalidades do *WSO2 Identity Server* podem ser consultados em [WSO2]. Dentre elas, a mais importante para o desenvolvimento deste trabalho é:

- **Segurança de APIs:** O *WSO2 Identity Server* cumpre um papel importante como um servidor de autorização que suporta diversos padrões ou perfis relacionados com *OAuth*, além de permitir alta disponibilidade e performance, para uma operação mais fluida;

O objetivo de se utilizar o*Identity Server* é o implementar como um gerenciador de chaves, em conjunto com o *API Manager*, para controlar o acesso e autenticação de usuários ao consumirem as APIs expostas.

## 3.2 CONFLUENCE

O Confluence é uma plataforma de gerenciamento de área de trabalho de equipe, desenvolvido pela Atlassian, e que oferece, segundo [Atlassian], páginas dinâmicas que proporcional um espaço ideal para criações, coletas e colaborações de equipe em projetos ou ideias.

Além disso, ainda oferece espaços para ajudar na estruturação, oganização e compartilhamento dos trabalhos, disponível para que cada membro definido possa acessar as informações e conhecimentos necessários para um melhor desempenho no trabalho.

Os principais termos e ferramentas do Confluence estão descritos a seguir:

- **Página:** Uma página contém os conteúdos criados no site do Confluence, são documentos vivos. Páginas podem ser criadas para quase tudo, como por exemplo planos de projeto, anotações de reuniões, guias de solução de problemas, documentações técnicas, etc. O Confluence possui templates para a criação de páginas de qualquer tipo de conteúdo, mas também permitindo a criação de uma página do zero.
- **Espaço:** As páginas criadas são armazenadas em espaços de trabalho, onde é possível manter todo o conteúdo organizado, análogo aos diretórios dentro de um sistema operacional. Espaços podem ser criados para cada projeto específico, reunindo as informações e documentações do mesmo, permitindo uma melhor organização e compartilhamento de conteúdo.

- **Ramificação de página:** O conteúdo de cada espaço pode ser organizado com uma ramificação de página hierárquica, o que torna o trabalho mais fácil e rápido de ser encontrado dentro da organização.

O Confluence se encaixa dentro do escopo do projeto como auxiliar nas documentações dos serviços e procedimentos executados, tanto na configuração de ambientes quanto na instalação de produtos. Já era um produto amplamente utilizado pela empresa onde o estudo e execução do projeto foram feitas.

## 3.3 SWAGGER

Dentro de uma arquitetura orientada a serviços (*SOA*), alguns dos seus princípios de desenvolvimento e implementação incluem a agnosticidade de um serviço, ou seja, que o mesmo possa ser reutilizado em diversas tarefas sem depender de outro recurso, e a importância da interoperabilidade entre serviços.

Dentro do universo das APIs REST, assim como em tantos outros meios da indústria da tecnologia da informação, as informações compartilhadas e possíveis serviços oferecidos por empresas devem possuir uma linguagem comum, para tornar o seu uso e compreensão mais fáceis e acessíveis, trazendo um grande benefício ao empreendimento que o compartilha.

Uma documentação ruim gera diversos desafios e empecilhos no funcionamento e uso de APIs expostas. Dessa forma, é ideal existir um padrão que facilite e otimize a documentação desses serviços, e é nesse contexto que surge o *Swagger*.

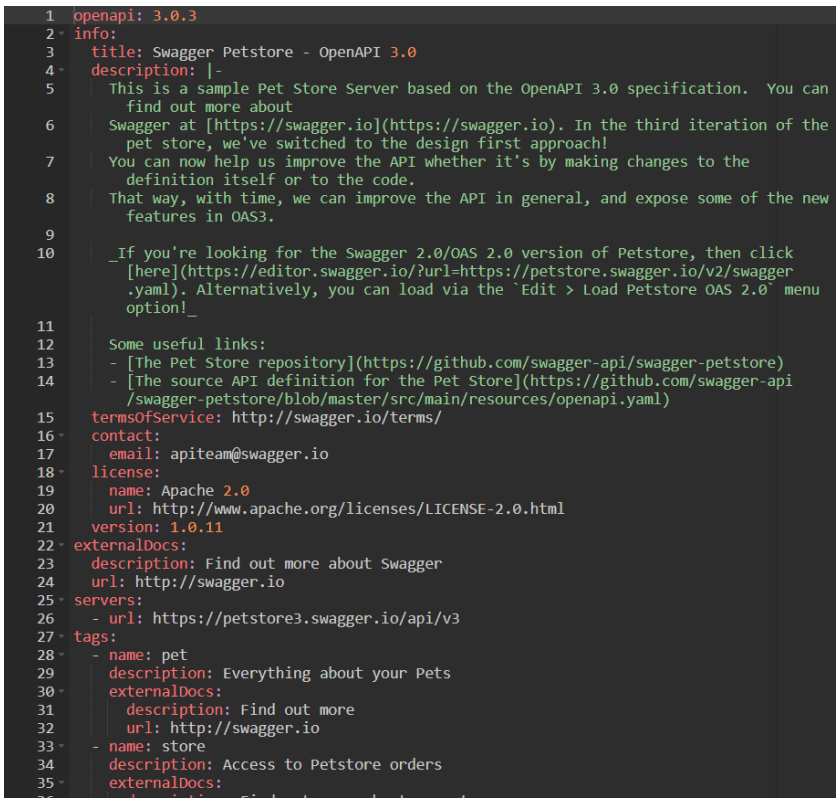
Conforme [IBM], o *Swagger* se trata de uma definição aberta para a definição de APIs REST, sendo um conjunto de ferramentas *Open Source*, que especificam uma lista de recursos disponíveis na API e as operações possíveis com esses recursos.

Os documentos *Swagger* devem estar em qualquer formato JSON, com uma extensão de arquivo .json, ou em formato YAML, com uma extensão .yaml ou .yml.

A especificação do *OpenAPI*, antigamente conhecida como especificação *Swagger*, segundo [SmartBear], permite descrever a API inteira e suas especificações, incluindo:

- *endpoints* disponíveis e operações em cada um deles (GET, POST, etc);
- parâmetros de operações de entrada e as saídas para cada uma delas;
- métodos de autenticação
- informações de contato, licença, termos de uso e outras informações.

Um exemplo de estrutura de documento *Swagger* para uma API REST, e suas operações, se encontram nas imagens a seguir:



```yaml
openapi: 3.0.3
info:
  title: Swagger Petstore - OpenAPI 3.0
  description: |-
    This is a sample Pet Store Server based on the OpenAPI 3.0 specification.  You can find out more about
    Swagger at [https://swagger.io](https://swagger.io). In the third iteration of the pet store, we've switched to the design first approach!
    You can now help us improve the API whether it's by making changes to the definition itself or to the code.
    That way, with time, we can improve the API in general, and expose some of the new features in OAS3.

    _If you're looking for the Swagger 2.0/OAS 2.0 version of Petstore, then click [here](https://editor.swagger.io/?url=https://petstore.swagger.io/v2/swagger.yaml). Alternatively, you can load via the `Edit > Load Petstore OAS 2.0` menu option!_

    Some useful links:
    - [The Pet Store repository](https://github.com/swagger-api/swagger-petstore)
    - [The source API definition for the Pet Store](https://github.com/swagger-api/swagger-petstore/blob/master/src/main/resources/openapi.yaml)
  termsOfService: http://swagger.io/terms/
  contact:
    email: apiteam@swagger.io
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.11
externalDocs:
  description: Find out more about Swagger
  url: http://swagger.io
servers:
  - url: https://petstore3.swagger.io/api/v3
tags:
  - name: pet
    description: Everything about your Pets
    externalDocs:
      description: Find out more
      url: http://swagger.io
  - name: store
    description: Access to Petstore orders
    externalDocs:
```

Figura 3.1: Exemplo de estrutura de documento *Swagger*. Fonte: *Swagger.io* [SmartBear]

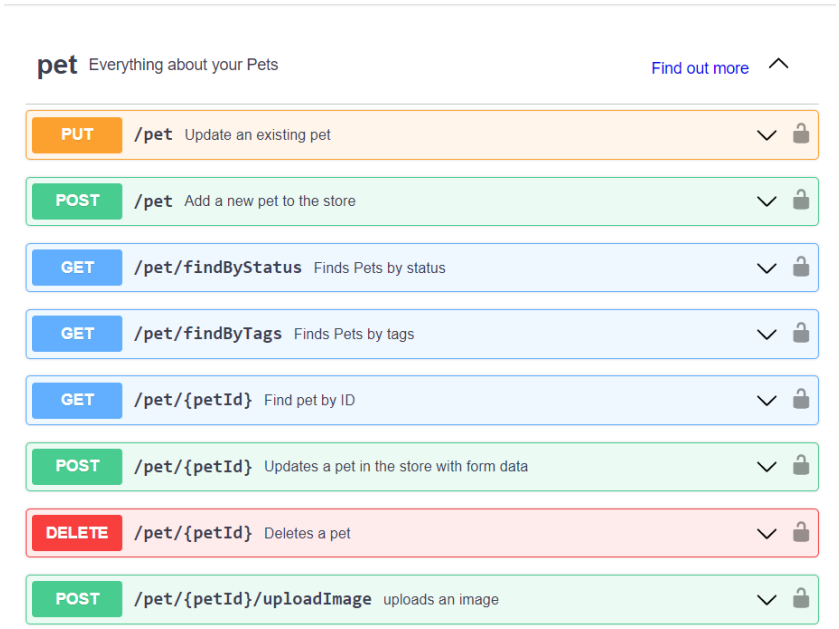



Figura 3.2: Exemplo de operações de uma API de exemplo. Fonte: *Swagger.io [SmartBear]*

O *Swagger*, no escopo deste trabalho, é a ferramenta utilizada pela empresa para a documentação inicial das APIs e suas definições principais. Ele foi estudado para a elaboração de uma documentação adicional de algumas APIs.

## 3.4 AMAZON WEB SERVICES

A Amazon Web Services (*AWS*) é uma plataforma de computação em núvem desenvolvido pela Amazon que oferece diversos serviços tecnológicos sob demanda através da internet.

Organizações de todo tipo usam serviços em núvem para uma grande variedade de casos de uso, que permitem o acesso a servidores, máquinas, bancos de dados e uma variedade de hardwares e ferramentas poderosas a um preço cobrado por uso.

A computação em núvem oferece diversos benefícios para as organizações, como a agilidade de ter acesso a diversas tecnologias de forma rápida, que permitem uma constante inovação, a elasticidade de prover os recursos estritamente necessários para a realização de um projeto, e economia de custos, por serem usados so demanda e de acordo com a necessidade.

A documentação completa de todos os serviços oferecidos pela *AWS* pode ser conferida em docs.aws.amazon.com, porém, alguns dos serviços oferecidos pela plataforma estão citados a seguir, conforme [Amazon 2023]:

- Computação;
- Armazenamento e Banco de Dados;
- Machine Learning;
- Ferramentas de tratamento de dados;
- Inteligência Artificial;
- dentre outros;

Para o escopo deste trabalho, utilizaremos instâncias EC2 da *AWS*, que são máquinas dedicadas em núvem com uma distribuição linux CentOS, configuradas em uma mesma subrede e interconectadas entre si. As instâncias foram utilizadas para a configuração do ambiente utilizado para expor as APIs através dos produtos da WSO2. A arquitetura proposta e detalhes da solução apresentada serão apresentados em um capítulo posterior.

# 4 ARQUITETURA PROPOSTA

Este capítulo visa descrever a arquitetura proposta para o estudo do catálogo de serviços de uma plataforma de registro eletrônico de saúde e os procedimentos para a sua implementação. É possível por meio deste capítulo verificar a topologia e arquitetura utilizadas, em um ambiente controlado, que simula a implementação real e prática do catálogo de serviços.

Por fim, espera-se que a partir do conteúdo apresentado por este, seja possível o entendimento de como as tecnologias usadas se relacionam com os objetivos propostos.

## 4.1 METODOLOGIA

O Projeto tem como finalidade, como citado anteriormente, estudar, implementar e compreender o funcionamento e a importância de um catálogo de serviços de uma plataforma de Registro Eletrônico de Saúde.

O WSO2 APIM é uma ferramenta que permite a exposição de serviços através de um portal, podendo estar acessível pela web, implementando, então, um catálogo de serviços.

O Projeto proposto, portanto, visa utilizar máquinas linux na núvem (*AWS*) para implementar, através das soluções *open-source* da WSO2, e a partir dos documentos *swagger* dos serviços, um catálogo que pode ser exposto através de um portal web para utilização e subscrição nas APIs disponíveis.

A arquitetura proposta utiliza as máquinas do APIM e do IS, em conjunto com um banco de dados comum implementado em outra máquina, além do endpoint das APIs, que já estavam publicadas e em amplo funcionamento.

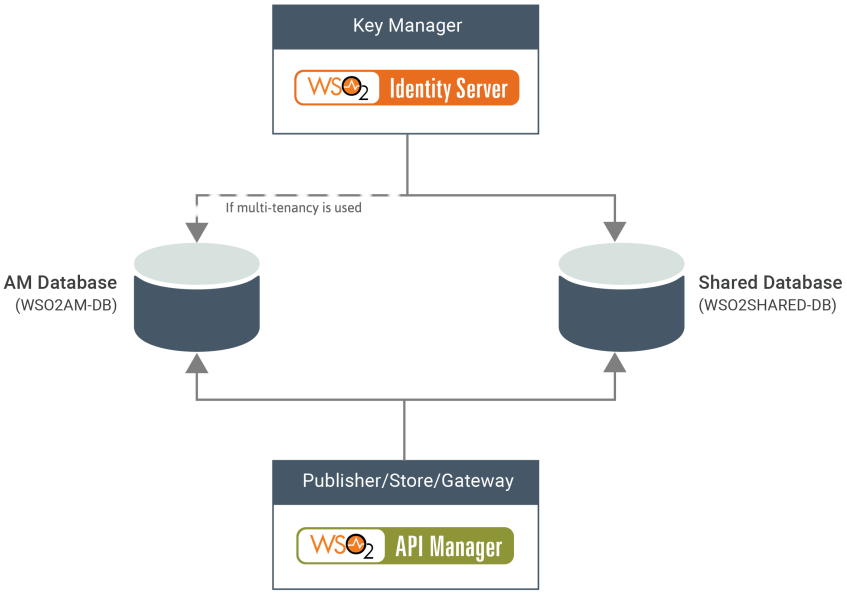



Figura 4.1: Cenário de utilização do Identity Server como *Key Manager* Fonte: WSO APIM Documentation

A implementação e execução do projeto foi confeccionada a permitir um entendimento do passo-a-passo do ambiente montado, como demonstrado a seguir:

- **PASSO 1 → Inicialização das máquinas em núvem;**
- **PASSO 2 → Instalação dos produtos WSO2 (APIM e IS);**
- **PASSO 3 → Configurações dos produtos WSO2 (APIM e IS);**
- **PASSO 4 → Publicação e configuração de APIs a partir do Swagger;**
- **PASSO 5 → Testes de consumo de serviços e geração de tokens;**
- **PASSO 6 → Análise dos resultados.**

## 4.2 CONFIGURAÇÃO E DESENHO DA ARQUITETURA

### 4.2.1 ARQUITETURA DE REDE

A arquitetura de rede da plataforma de registro eletrônico de saúde estudada foi projetada para se obter uma melhor segurança na troca de informações internas. As máquinas se encontram em uma mesma sub-rede na núvem, acessíveis por meio de uma VPN, e toda a arquitetura possui apenas uma superfície exposta à internet por meio de um balanceador de carga (NGINX).

Há cinco camadas diferentes dentro da infraestrutura, sendo elas:

- Camada Pública;
- Camada API;
- Camada Backend;
- Camada DOCs;
- Camada Dados.

O API Manager utilizado e configurado neste trabalho se encontra na camada API, expondo os serviços utilizados na plataforma. Já o Identity Server se encontra na camada de backend.

Outras máquinas e camadas são utilizadas na arquitetura desenvolvida para diversas funcionalidades da plataforma estudada, mas cujo funcionamento detalhado não está dentro do escopo deste projeto. A arquitetura de rede é, portanto, ilustrada na figura 4.2.

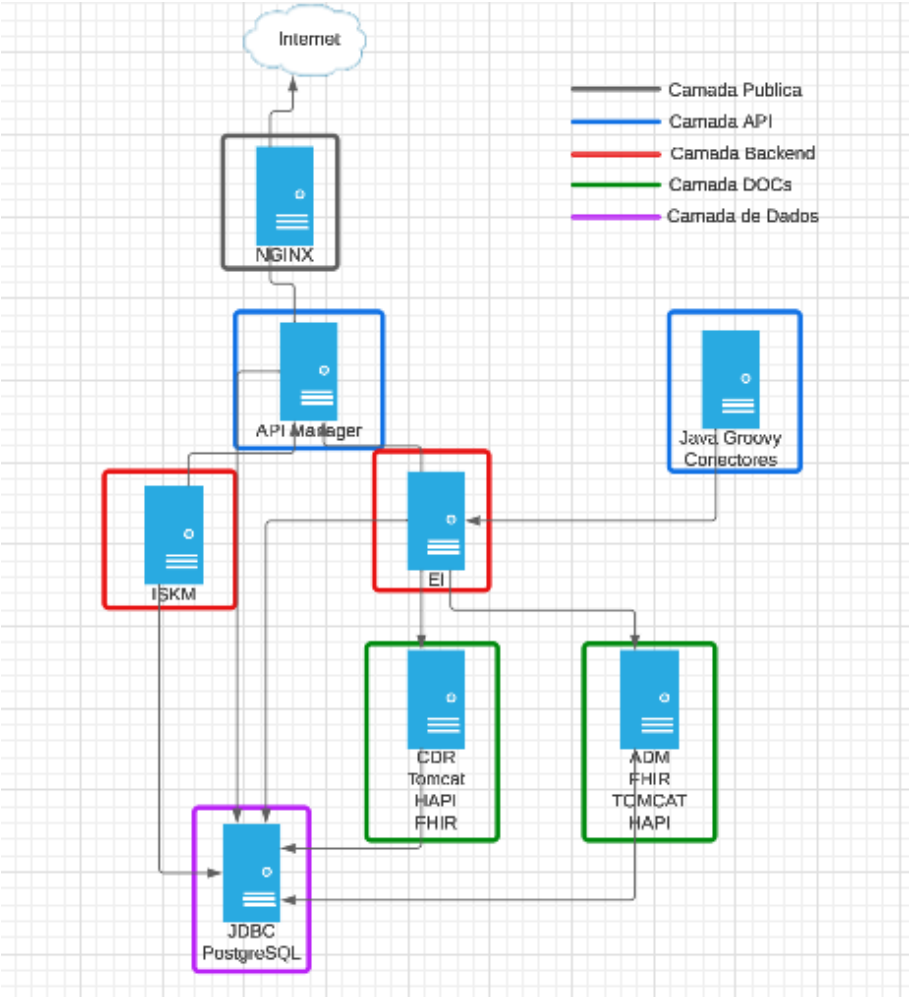



Figura 4.2: Ilustração da Arquitetura de Rede Fonte: autor

### 4.2.2 CRIAÇÃO DAS MÁQUINAS NA *AWS*

A AWS, como descrito pela seção 3.4, é uma plataforma de computação em núvem desenvolvida pela Amazon e que oferece diversos serviços para soluções utilizando a tecnologia da computação em núvem.

Dessa forma, para a arquitetura utilizada nesse projeto, foram utilizadas máquinas linux mantidas na plataforma, para se configurar o ambiente onde o catálogo de serviços a ser estudado seria implementado.

Para as máquinas configuradas, foram escolhidos recursos EC2 da AWS, localizados na região *US East (N. Virginia)*. A máquina escolhida para a implementação do APIM foi a *t3.medium*, enquanto a do IS foi a *t2.small*. Os recursos computacionais de cada máquina são:

Tabela 4.1: Recursos computacionais das máquinas utilizadas na AWS

| **Nome** | **vCPUs** | **RAM(GiB)** | **SO** |
|---|---|---|---|
| *t2.small* | 1 | 2.0 | CentOS 7 |
| *t3.medium* | 2 | 4.0 | CentOS 7 |

### 4.2.3 INSTALAÇÃO DOS PRODUTOS DA WSO2

A WSO2 oferece diversos produtos de código aberto para soluções computacionais. Para o escopo deste trabalho, foram usados o API Manager e o Identity Server. O APIM, como descrito na seção 3.1.1, é uma solução de gerenciamento de APIs, que conta com uma plataforma de exposição das mesmas para a utilização, servindo como um catálogo de serviços.

Já o Identity Server, conforme a seção 3.1.2, será utilizado como uma ferramenta de gerenciamento de chaves, melhorando a segurança no acesso aos serviços disponibilizados através do APIM.

Os produtos da WSO2 necessitam do Java instalado e configurado nas máquinas, além de poderem solicitar alguma outra aplicação. A instalação do Java e respectiva configuração da variável JAVA HOME estão descritos na seção I

#### 4.2.3.1 API Manager

Para a instalação do WSO2 APIM, um arquivo *.zip* deve ser baixado em WSO2 API Manager. A versão utilizada foi a 4.1.0. É necessário instalar o Java Ant como pre-requisito para a utilização do APIM, que pode ser facilmente instalado através do comando:

```
sudo yum install ant
```

Após o download do arquivo, o mesmo foi enviado à VM em execução na AWS, através do protocolo SFTP e descompactado no diretório */opt* da máquina destinada ao funcionamento do APIM.

A execução, com as configurações padrão de instalação, se dá através de um script localizado em */APIM-HOME/bin/api-manager.sh*, da forma:

```
sh /(APIM-HOME)/bin/api-manager.sh
```

O acesso aos portais se dá através da porta 9443, mostrados após a inicialização completa do produto, como na figura que segue:


Figura 4.3: Portais disponíveis após a inicialização do APIM Fonte: autor

#### 4.2.3.2 Identity Server

O Identity Server foi utilizado para o desenvolvimento deste projeto em sua versão 6.0.0. Para a sua instalação, o download do arquivo *.zip* do produto deve ser feito em WSO2 Identity Server.

O arquivo baixado deve, então, ser enviado à máquina que será utilizada. Nesse caso, por ser uma máquina em núvem, foi utilizado o protocolo SFTP para este procedimento.

Após a descompactação, o diretório resultante foi movido para o diretório */opt* da máquina destinada ao funcionamento do Identity Server.

Para a execução do produto, basta se executar um scrip encontrado na pasta */bin* do diretório do produto, chamado *wso2server.sh*, da forma:

```
sh /(IS-HOME)/bin/wso2server.sh
```

As ferramentas disponíveis são as descritas na imagem a seguir:

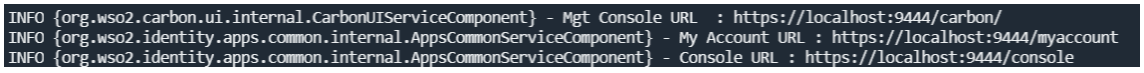



Figura 4.4: Ferramentas disponíveis após a inicialização do IS Fonte: autor

### 4.2.4 CONFIGURAÇÃO DOS PRODUTOS

As configurações dos produtos da WSO2 são feitas através da edição de um arquivo chamado *deployment.toml*, que se encontra em diretórios diferentes, dependendo do produto a ser configurado.

Neste trabalho, o *Identity Server* foi utilizado como um *Key Manager*. Para isso, o mesmo deve ser configurado, juntamente com o APIM, para atuar dessa forma.

Sendo assim, as configurações realizadas no ambiente foram como descritas a seguir.

#### 4.2.4.1 Configurando o offset de portas do Identity Server

Para acessar as ferramentas disponíveis pelo IS, deve-se realizar um offset de portas, para que não haja conflito com as mesmas portas utilizadas pelo APIM no acesso aos portais.

Para tal, o seguinte procedimento é executado:

1. Edição do arquivo <IS-HOME>/repository/conf/deployment.toml com o offset alterado para 1, aplicando a configuração como segue:

```
[server]
offset = 1
```

A configuração ficará como na figura a seguir:

```
[server]
hostname = "localhost"
offset = 1
node_ip = "127.0.0.1"
base_path = "https://$ref{server.hostname}:${carbon.management.port}"
```

Figura 4.5: Configuração de Offset de portas do IS Fonte: autor

Dessa forma, as ferramentas do IS se tornam acessíveis através da porta 9444.

#### 4.2.4.2 Instalação e Configuração dos Bancos de Dados

Para a utilização em conjunto do APIM e IS, temos que seguir uma topologia mostrada na figura 4.1. Para isso, os bancos de dados utilizados pelos produtos utilizados devem ser os mesmos, em sua configuração de banco compartilhado.

Sendo assim, um servidor RDS na AWS foi utilizado para a criação dos bancos a serem utilizados no projeto, criando um banco do tipo PostgreSQL, com o nome de *"WSO2-SHARED-DB"*.

Para se utilizar o banco de dados PostgreSQL, é necessário a instalação de um driver nos produtos utilizados. Os passos necessários para isso são:

1. Download do *driver* em Download | pgJDBC;
2. Colocar o arquivo baixado no diretório *WSO2-HOME/repository/components/lib/*.

Foram criadas as tabelas no servidor de banco de dados para os bancos de nome WSO2 SHARED DB, que diz respeito ao banco compartilhado entre os produtos, e WSO2AM DB, que é o banco de dados do API Manager.

Para alterar o banco de dados H2 padrão para o novo banco PostgreSQL, são necessários os seguintes passos no APIM:

1. Para a mudança do banco de dados pré-definido para o PostgreSQL, deve-se alterar o arquivo *<API-M-HOME>/repository/conf/deployment.toml* em seus elementos de configuração *[database.shared db]* e *[database.apim db]* para:

```
[database.shared_db]

type = "postgre"
url = "jdbc:postgresql://(postgresmachine):5432/wso2 shared db"
username = "(username)"
password = "(password)"
driver = "org.postgresql.Driver"
validationQuery = "SELECT 1""

"[database.apim_db]
type = "postgre"
url = "jdbc:postgresql://(postgresmachine):5432/wso2am db"
username = "(db_username)"
password = "(db_password)"
driver = "org.postgresql.Driver"
validationQuery = "SELECT 1""
```

E, para o identity server, deve-se alterar os elementos contidos no arquivo *deployment.toml*, em *[database.shared db]*, para:

```
[database.shared db]

type = "postgre"
url = "jdbc:postgresql://(postgresmachine):5432/wso2 shared db"
username = "(db_username)"
password = "(db_password)"
driver = "org.postgresql.Driver"
validationQuery = "SELECT 1"
```

As informações de *url*, usuário e senha foram ocultadas por motivos de segurança.

Para o *Identity Server*, deve-se alterar o arquivo *deployment.toml* em seu elemento de configuração [database.shared db] com as seguintes configurações:

```
[database.shared db]

type = "postgre"
url = "jdbc:postgresql://(db.machine)5432/wso2 shared db"
username = "(db.username)"
password = "(db.password)"
driver = "org.postgresql.Driver"
```

```
validationQuery = "SELECT 1""
```

Após realizadas as alterações, os serviços precisam ser reinicializados para que elas ocorram com sucesso.

#### 4.2.4.3 Configuração do IS como Key Manager

Para a correta configuração do Identity Server como um Key manager, juntamente com o APIM, temos as seguintes etapas:

1. Realizar o download do WSO2 IS Connector;
2. Extrair os arquivos e copiar os seguintes arquivos, que estão no diretório <wso2is-extensions-1.4.2>/dropins , para o diretório IS- HOME>/repository/components/dropins:
   - wso2is.key.manager.core-1.4.2;
   - wso2is.notification.event.handlers-1.4.2;
3. Adicionar o arquivo *keymanager-operations.war*, que está no diretório *<wso2is-extensions-1.4.2>/webapps*, ao diretório *<IS-HOME>/repository/deployment/server/webapps*;
4. Configurar os endpoints do traffic manager, adicionando as seguintes configurações ao arquivo *deployment.toml* do IS:

```
[[event_listener]]
id = "token_revocation"
type = "org.wso2.carbon.identity.core.handler.AbstractIdentityHandler"
name = "org.wso2.is.notification.ApimOauthEventInterceptor"
order = 1

[[resource.access_control]]
context = "(.)/keymanager-operations/user-info/claims(.)"
secure = true
http_method = "GET"
permissions = "/permission/admin/manage/identity/usermgt/list"
scopes = "internal_user_mgt_list"

[[resource.access_control]]

context = "(.*)/keymanager-operations/user-info/claims/generate"
secure = true
http_method = "POST"
```

```
permissions = "/permission/admin/manage/identity/usermgt/list"
scopes = "internal_user_mgt_list"

[[resource.access_control]]
context = "(.*)/keymanager-operations/dcr/register"
secure = true
http_method = "POST"
permissions = "/permission/admin/manage/identity/applicationmgt/create"
scopes = "internal_application_mgt_create"

[[resource.access_control]]

context = "(.*)/keymanager-operations/dcr/register(.*)"
secure = true
http_method = "GET"
permissions = "/permission/admin/manage/identity/applicationmgt/view"
scopes = "internal_application_mgt_view"

[[resource.access_control]]

context = "(.*)/keymanager-operations/dcr/register(.*)"
secure = true
http_method = "DELETE"
permissions = "/permission/admin/manage/identity/applicationmgt/delete"
scopes = "internal_application_mgt_delete"

[[resource.access_control]]

context = "(.*)/keymanager-operations/dcr/register(.*)"
secure = true
http_method = "PUT"
permissions = "/permission/admin/manage/identity/applicationmgt/update"
scopes = "internal_application_mgt_update"

[[resource.access_control]]

context = "(.)/keymanager-operations/dcr/register(.)"
secure = true
http_method = "POST"
permissions = "/permission/admin/manage/identity/applicationmgt/update"
scopes = "internal_application_mgt_update"
```

```
[tenant_context.rewrite]

custom_webapps = ["/keymanager-operations/"]
```

5. Configurar o endpoint do event listener, para publicar os eventos de controle no Traffic Manager, adicionando uma configuração, como no exemplo a seguir, ao arquivo *deployment.toml* do IS:

```
[event_listener.properties]
notification_endpoint = https://<tm.wso2.com>:9443/
internal/data/v1/notify"
username = "${admin.username}"
password = "${admin.password}"
'header.X-WSO2-KEY-MANAGER' = "WSO2-IS"
```

No APIM, deve-se apenas alterar o arquivo *deployment.toml*, em suas configurações *[apim.key manager]*, da forma:

```
[apim.key_manager]

service_url = "https://(IS.IP.address):9444/services/"
type = "WSO2-IS"
```

Os serviços devem ser reiniciados após as alterações. Dessa forma, o Identity Server está configurado como um *Key Manager* juntamente com o API Manager.

# 5 TESTES E RESULTADOS

## 5.1 PUBLICAÇÃO DE APIS

Para a publicação de APIs no API Manager 4.1.0 da WSO2, os seguintes passos são necessários:

1. Acessar o portal disponibilizado pela ferramenta, através da url *https://localhost:9443/publisher*, com login e senha definidos no arquivo de configuração. O login e senha padrão é *admin/admin*;

2. Clicar em *Create API* no canto superior esquerdo; na opção *REST API*, clicar em *Import Open API*; na opção *Input Type*, clicar em *OpenAPI FIle/Archive*; em seguida, deve-se subir o arquivo *swagger*; após isso, deve-se preencher Name, Context, Version e Endpoint e clicar OK. Será aberta uma nova página para a API:

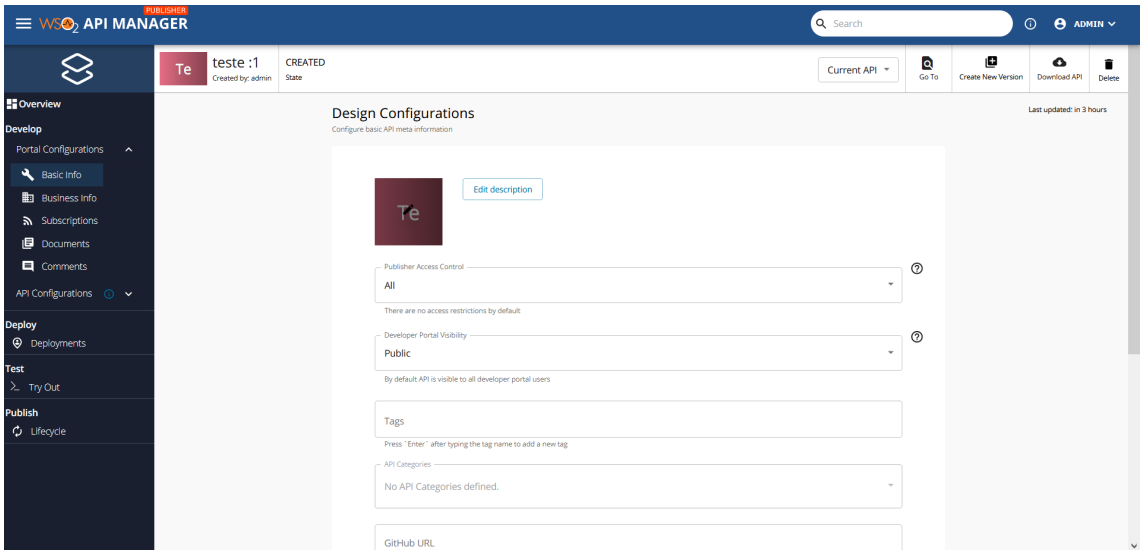



Figura 5.1: Primeira página após a criação da API no APIM. Fonte: autor

3. Na opção *Develop*, clicar em *Portal Configurations*; em seguida, clicar em *Basic Info*; na região do quadrado vermelho, deve-se subir a imagem referente à API (se for o caso) e clicar Save.

4. Na opção *Subscription*, escolher uma ou mais opção conforme for conveniente e clicar Save.

5. Na opção *API Configurations*, clicar em *Runtime*; na opção *Request*, habilitar a opção *CORS Configuration*; abrir a opção *CORS Configuration*; habilitar *Allow All Origins*; preencher o campo Access Control Allow Headers com valores adequados (depende de cada API); preencher o campo *Access Control Allow Methods* com valores adequados (depende de cada API); e clicar Save.

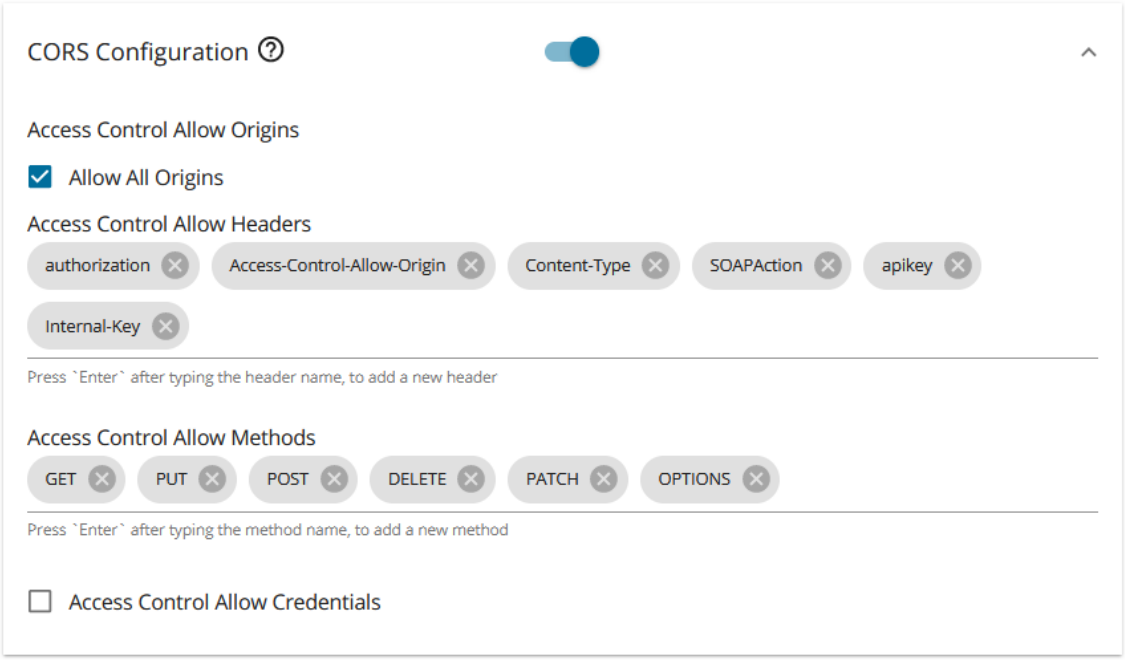



Figura 5.2: CORS configuration no APIM. Fonte: autor

6. Na opção *Policies*, deve-se subir, se for o caso, os arquivos da política de mediação na opção *Add New Policy*; preencher ou selecionar os campos de acordo com a especificidade de cada política; arrastar a potítica de mediação criada para os fluxos adequados dos métodos em questão; clicar Save.

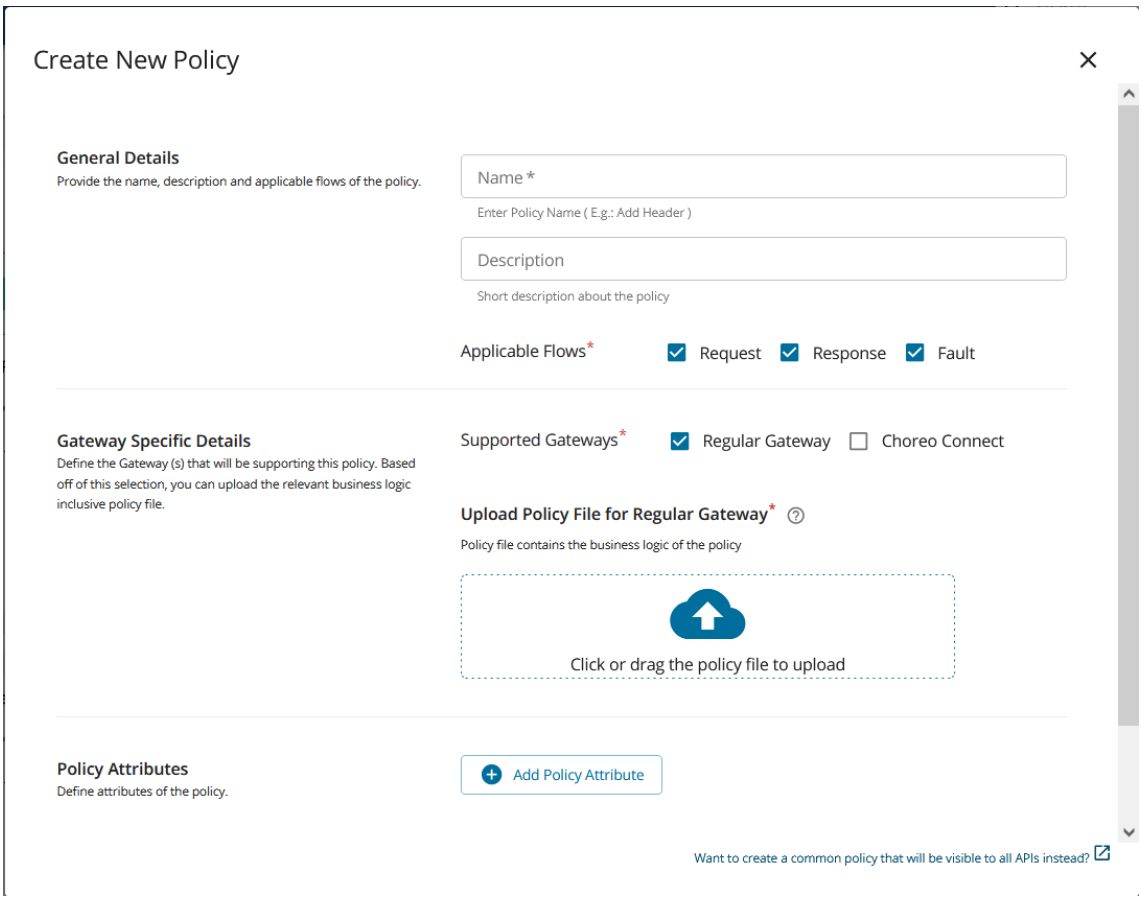



Figura 5.3: Criação de uma política de mediação. Fonte: autor

7. Na opção *Deploy*, clicar em *Deployments*; e, em seguida, clicar em *Deploy* na página que se abriu.
8. Pode-se testar a API por meio da opção *Test*, clicando-se na opção *Try Out*.
9. Na opção *Publish*, clica-se em *Lifecycle*; e, em seguida, deve-se clicar em *Publish* na página que se abriu. Conclui-se, dessa forma, a publicação da API.
10. Pode-se verificar que a API está publicada no devportal em https://localhost:9443/devportal.

Para as APIs da plataforma de registro eletrônico de saúde, os *swaggers* devidamente documentados dos mesmos foram utilizados, com o processo descrito anteriormente de publicação de APIs, para expor todos os serviços necessários para o funcionamento da plataforma.

Após realizado o procedimento para todos os serviços, o *devportal*, onde as APIs são expostas, ficou da seguinte forma:

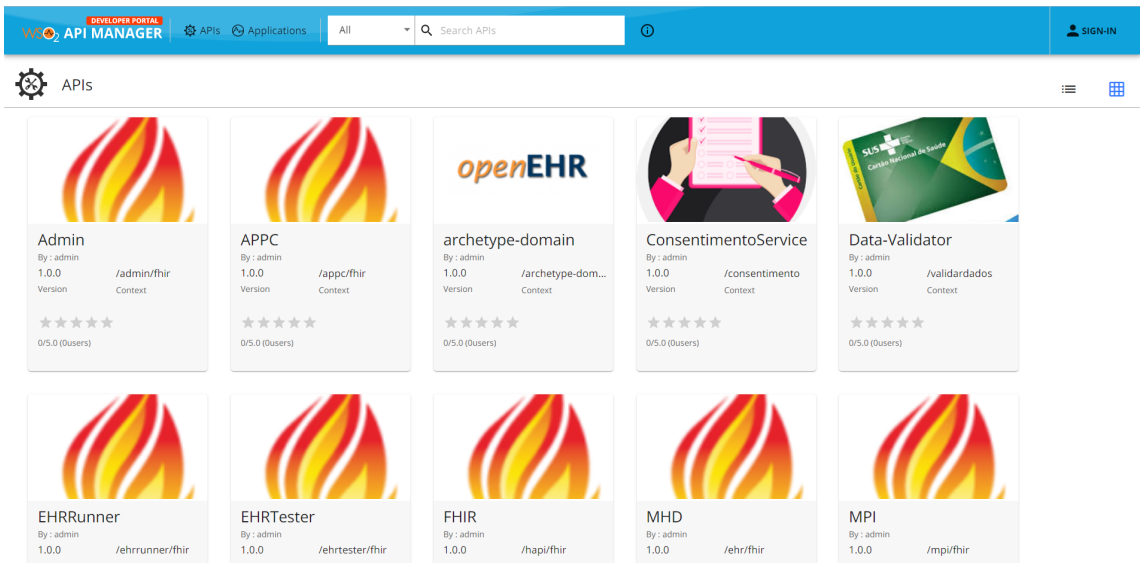



Figura 5.4: DevPortal com as APIs publicadas. Fonte: autor

O DevPortal é o portal onde os serviços ficam disponíveis para serem consumidos e utilizados pela plataforma e, se desejado, por outros produtos ou serviços. Ao se clicar em uma API, algumas informações sobre ela são mostradas, juntamente com a possibilidade de *download* do seu *Swagger*.

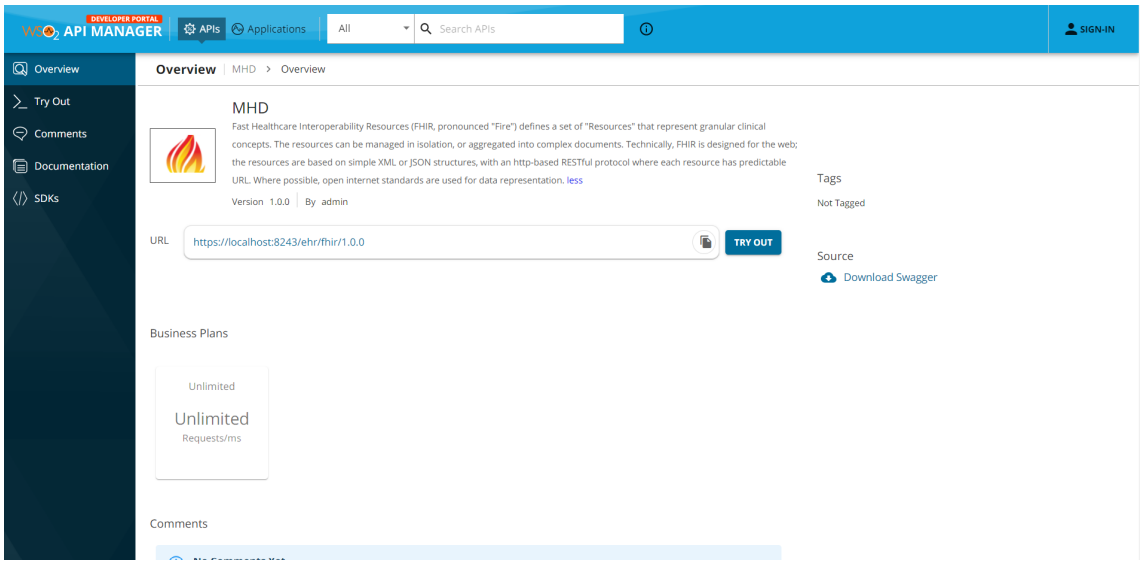



Figura 5.5: Detalhes da API aberta no DevPortal. Fonte: autor

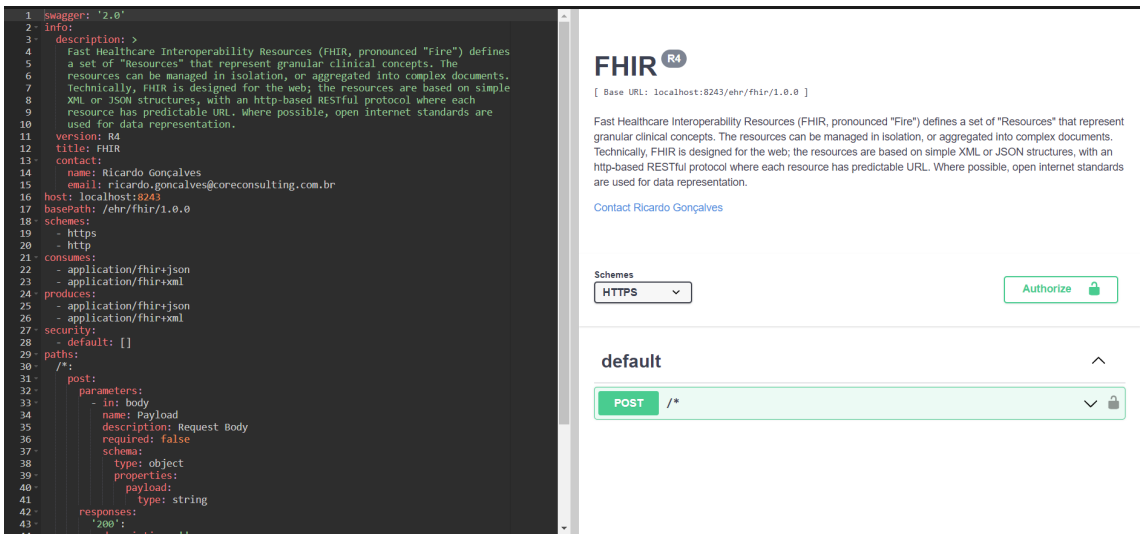



Figura 5.6: Swagger da API baixado através do DevPortal. Fonte: autor

## 5.2 GERAÇÃO DE *TOKEN* E CONSUMO DOS SERVIÇOS

Para o consumo de um serviço, uma aplicação deve ser criada e, a mesma, solicitar um token e consumir a API.

A criação de uma aplicação pode ser feita ao se logar no DevPortal com a credencial configurada no arquivo *deployment.toml* do APIM. O usuário e senha padões são: admin/admin.

Após o login, na aba *applications*, foi criada uma nova aplicação de testes.

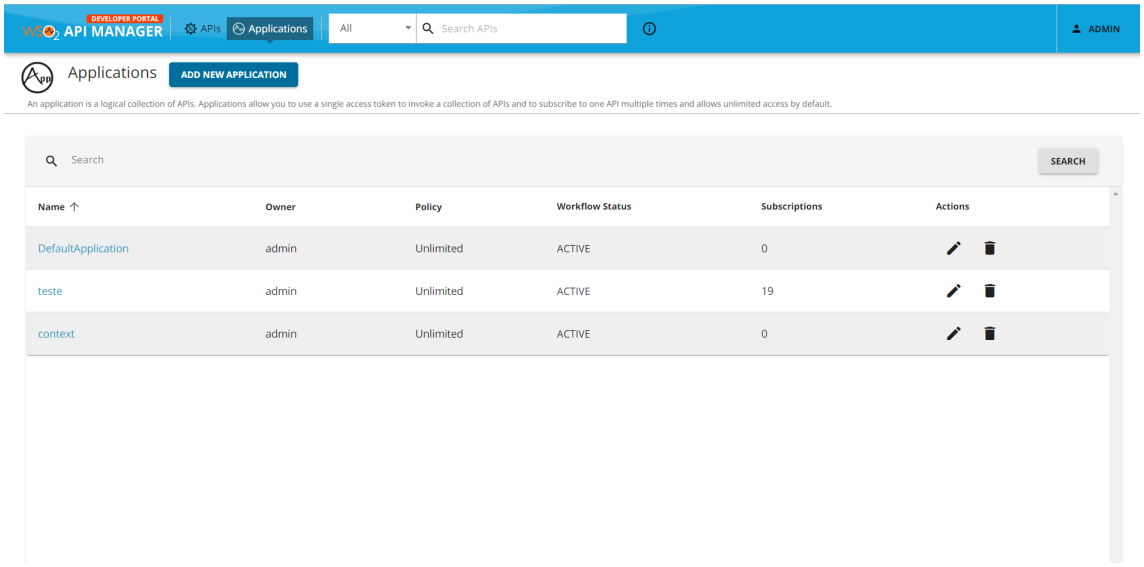



Figura 5.7: Aplicações criadas no DevPortal. Fonte: autor

Com a aplicação criada, pode-se subscrever às APIs disponíveis e testá-las. Alguns testes executados não foram possíveis de serem realizados diretamente no DevPortal devido a como a própria API se

comporta, mas os mesmos puderam ser executados através do comando *curl* no terminal linux.

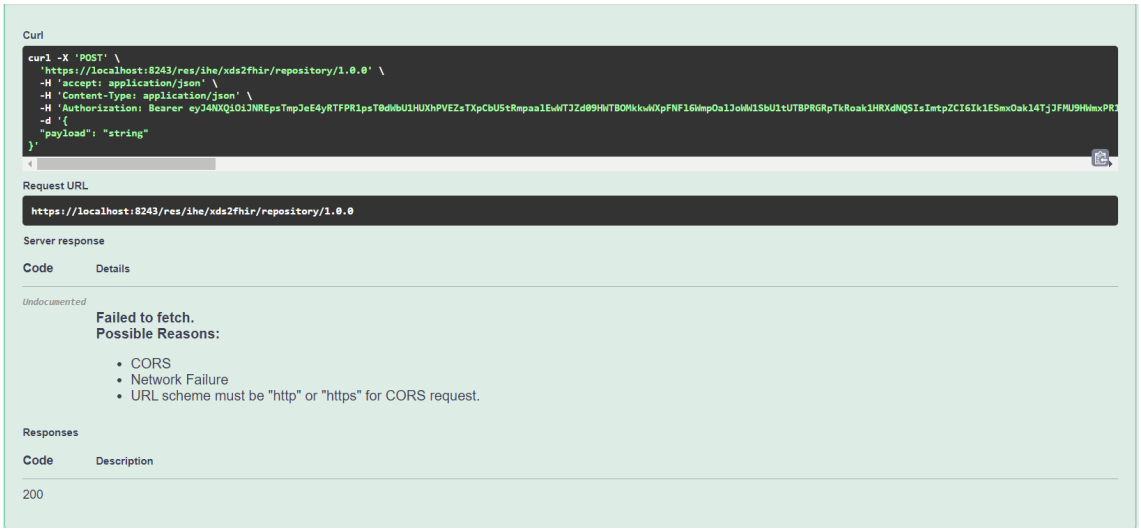



Figura 5.8: Teste de API no DevPortal. Fonte: autor

Mesmo assim, podemos observar o *bearer token* gerado, confirmando o correto funcionamento e integração do Identity Server como um Key Manager.


Figura 5.9: Teste de API no terminal linux. Fonte: autor

Seguindo o mesmo procedimento, todas as APIs foram publicadas, atestando o seu funcionamento completo e, então, disponibilização dos serviços a serem utilizados pela plataforma estudada.

O catálogo fica disponibilizado através do DevPortal, onde aplicações podem ser criadas e subscritas em APIs para o seu consumo. Ao se alterar algo na API, a mesma pode ser republicada através do *publisher*.

O Identity Server está integrado como um Key Manager, gerando *tokens* de acesso ao se consumir um serviço. Com isso, a integração e configuração dos produtos e do catálogo de serviços está completa e em pleno funcionamento.

# 6 CONCLUSÃO

O projeto trouxe um cenário de testes em núvem de forma controlada, para a implementação e estudo do comportamento do catálogo de serviços de uma plataforma de Registro Eletrônico de Saúde. Através de softwares de código aberto da WSO2 foi possível realizar testes de publicação e consumo dos serviços disponíveis através de um portal gerado ao se iniciar o API Manager.

O ambiente foi desenvolvido de forma a integrar as soluções da WSO2 (IS e APIM), utilizando o Identity Server como um Key Manager, para gerenciar *tokens* de usuários e manter a segurança na utilização e consumo dos serviços.

Através de máquinas virtuais utilizadas na núvem (AWS), foi possível demonstrar a implementação do catálogo de serviços utilizando documentações *swagger* das APIs disponibilizadas e utilizadas pela plataforma.

Sendo assim, após configuração e testes, pôde-se observar o comportamento de um inventário de serviços, dentro de uma arquitetura orientada a serviços (SOA), compreendendo o seu comportamento e, dessa forma, pontuando a sua importância dentro da arquitetura da plataforma.

Por fim, pode-se concluir que os objetivos de estudo e observação, em conjunto com a implementação e consumo dos serviços disponíveis, foram realizados, trazendo a importância de uma boa documentação de serviços através de documentos *swagger*, facilitando e ampliando o gerenciamento de serviços da plataforma e a sua possível interconexão e utilização por outros produtos e serviços.

# 7 TRABALHOS FUTUROS

Este capítulo tem como objetivo sugerir trabalhos futuros que continuem, complementem ou sejam inspirados por este estudo. Há diversos produtos que podem ser utilizados e comparados aos utilizados no desenvolvimento deste trabalho, como também outras áreas de estudo que podem ter os seus serviços catalogados e publicados.

Com isso, sugere-se como trabalhos futuros, mas não se limitando aos mesmos:

- Estudo comparativo de desempenho entre outras ferramentas de exposição de serviços com as ferramentas da WSO2 utilizadas neste trabalho;
- Estudo de catálogo de serviços em outras plataformas fora da área da saúde;
- Desenvolvimento das documentações de serviços e sua posterior exposição em alguma ferramenta de uma plataforma que não a possua;
- Testes e comparações de desempenho da arquitetura SOA e exposição de serviços com outras arquiteturas utilizadas atualmente;
- Estudo do uso de serviços pela plataforma de Registro Eletrônico de Saúde e desenvolvedores de fora dela.

# REFERÊNCIAS BIBLIOGRÁFICAS


Amazon 2023 AMAZON. *AWS Documentation*. 2023. <https://docs.aws.amazon.com/?nc2=h_ql_doc_do>. (Accessed on 01/23/2023).

Atlassian ATLASSIAN. *Noções básicas do Confluence*. <https://www.atlassian.com/br/software/confluence/guides/get-started/confluence-overview#about-confluence>. (Accessed on 01/21/2023).

Erl 2008 ERL, T. *SOA: principles of service design*. [S.l.]: Prentice Hall, 2008.

Erl et al. 2011 ERL, T.; BENNET, S. G.; GEE, C.; LAIRD, R.; MANES, A. T.; SCHNEIDER, R.; SHUSTER, L.; TOST, A.; VENABLE, C. *SOA governance: governing shared services on-premise aind in the cloud*. [S.l.]: Prentice Hall, 2011.

FERREIRA 2021 FERREIRA, A. G. *Interface de programação de aplicações (API) e web services.* [S.l.]: Editora Saraiva, 2021.

Gama Maria do Mar Rosa 2013 GAMA MARIA DO MAR ROSA, M. M. d. S. N. It services reference catalog. IFIP/IEEE, 2013.

Hat 2023 HAT, R. *O que é arquitetura orientada a serviços (SOA)?* 2023. <https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-service-oriented-architecture/>. (Accessed on 02/13/2023).

IBM IBM. *Swagger*.

O'Loughlin 2010 O'LOUGHLIN, M. *The Service Catalog - A Practitioner Guide*. [S.l.]: Van Haren Publishing, 2010.

Patni 2017 PATNI, S. *Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS*. [S.l.]: Sanjay Patni, 2017.

REST: Princípios e boas práticas REST: Princípios e boas práticas. Disponível em: <https://www.alura.com.br/artigos/rest-principios-e-boas-praticas>.

SmartBear SMARTBEAR. *What Is OpenAPI?*

Subramanian e Raj 2019 SUBRAMANIAN, H.; RAJ, P. *Hands-On RESTful API Design Patterns and Best Practices*. [S.l.]: Packt Publishing Ltd., 2019.

WSO2 WSO2. *API Manager Documentation 4.1.0*. <https://apim.docs.wso2.com/en/latest/get-started/overview/>. (Accessed on 01/20/2023).

WSO2 WSO2. *Identity Server Documentation*. <https://is.docs.wso2.com/en/latest/get-started/quick-start-guide/>. (Accessed on 01/22/2023).

# ANEXOS

# I. INSTALAÇÃO DO OPENJDK 11 E CONFIGURAÇÃO DA VARIÁVEL JAVA HOME

Ao utilizarmos as máquinas com o sistema operacional CentOS, as instalações de pacotes e aplicações podem ser feitas através do repositório *yum*. Os passos e detalhes das instalações estão descritos a seguir.

### I.0.1 Instalação do OpenJDK 11

Os passos para a instalação do OpenJDK 11 são:

1. Atualizar o repositório de pacotes para ter certeza de que a última versão do software será baixada, através do comando:

   ```
   sudo yum update
   ```

2. Instalar o Java Developer Kit com o seguinte comando:

   ```
   sudo yum install java-11-openjdk
   ```

   Para verificar a instalação do Java na máquina, utilizar o seguinte comando:

   ```
   java -version
   ```

   Se tudo ocorrer como esperado, a resposta do sistema será como na imagem a seguir:

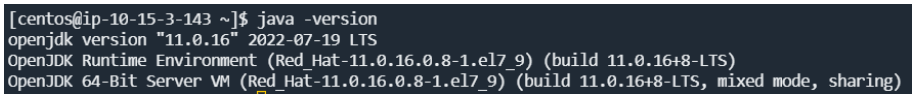



Figura I.1: Java instalado na máquina. Fonte: autor

Caso haja outro Java instalado na máquina, há como decidir qual deles será o utilizado de forma default pelo sistema. Para fazer isso, execute primeiro o comando que mostre as versões instaladas na máquina:

```
sudo alternatives --config java
```

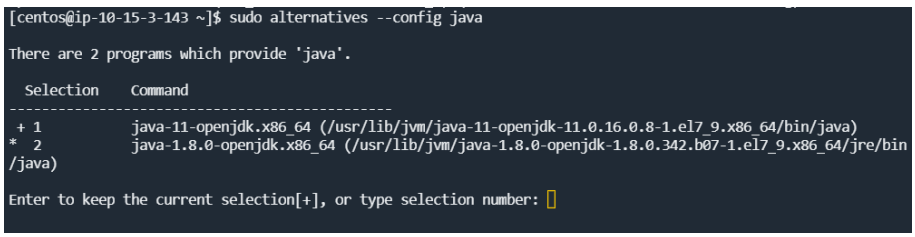



Figura I.2: Versões do Java instalado na máquina. Fonte: autor

A imagem acima mostra que existem duas versões disponíveis no sistema. Pode-se, então, escolher o número 1 (Java 11) e 2 (Java 8 ) digitando a opção escolhida e teclando Enter. A versão escolhida foi a do Java 11.

### I.0.2 Configuração da variável JAVA HOME para todos os usuários

Para a configuração da variável JAVA HOME, necessária para o correto funcionamento dos produtos da WSO2, os seguintes passos são necessários:

1. Editar o arquivo environment com o seu editor de texto de preferência, através dos comandos:

```
touch /etc/environment

nano /etc/environment
```

Após abrir o arquivo, inserir a seguinte linha:

```
JAVA_HOME=/usr/lib/jvm/(java utilizado)
```

Se houverem duas versões do Java disponíveis, configurar para o Java desejado. Nesse caso, temos o Java 11 e a sua configuração na máquina é feita adicionando a seguinte linha ao documento:

```
"JAVA_HOME=/usr/lib/jvm/java-11-openjdk-11.0.16.0.8-1.el7_9.x86_64"
```

2. Verificar a criação da variável de ambiente através do comando:

```
echo $JAVA_HOME
```

```
[centos@ip-10-15-3-143 ~]$ echo $JAVA_HOME
/usr/lib/jvm/java-11-openjdk-11.0.16.0.8-1.el7_9.x86_64/
```

Figura I.3: Variável JAVA HOME configurada. Fonte: autor

3. Editar o arquivo java.sh, utilizando um editor de texto de preferência, da forma:

```
touch /etc/profile.d/java.sh

nano /etc/profile.d/java.sh
```

Adicionando as seguintes linhas ao arquivo aberto:

```
"#!/bin/bash

source /etc/environment

export JAVA_HOME=$JAVA_HOME

export PATH=$PATH:$JAVA_HOME/bin"
```

Após, salvar e executar o arquivo com o seguinte comando:

```
bash /etc/profile.d/java.sh
```

4. Verificar a modificação da variável de ambiente PATH, utilizando o seguinte comando:

```
echo $PATH
```