

Plataforma baseada em microserviços para gerenciamento de transformações de modelos

Samuel Possamai* e Gian Ricardo Berkenbrock†

29 de Julho de 2022

Resumo

Neste trabalho é proposta uma plataforma para conversão entre diferentes tipos de arquivos de modelos. Ela busca simplificar o uso de ferramentas de conversão, integrando elas em um único ambiente, sem a necessidade de instalá-las e configurá-las individualmente. Ela também visa suportar a armazenagem de modelos e permitir a organização deles. Dessa forma, facilitando a integração de modelos em diferentes formatos, permitindo que eles sejam armazenados e convertidos em uma mesma aplicação. Para elaborar a plataforma, no presente texto, foi elaborado requisitos para o projeto e definido uma arquitetura e especificação de um API interna e externa visando cumpri-los.

Palavras-chave: microsserviços, modelos, transformação e requisitos.

1 Introdução

O uso de aplicações de *software* já faz parte do cotidiano de grande parte da população mundial, podendo elas estarem sendo executadas diretamente no computador ou celular do usuário final, ou indiretamente por um servidor na nuvem. Elas têm diversas utilidades que abrangem desde o acesso à informação e notícias até entretenimento e até mesmo o desenvolvimento de outras aplicações ou aperfeiçoar a ela mesma.

No desenvolvimento de software são utilizadas diversas ferramentas que incluem a parte física(*hardware*) e virtual(*software*). Uma das categorias de ferramentas essenciais é as linguagens de programação, elas permitem ao programador dar instruções a máquina, porém existem muitas outras ferramentas que cumprem um papel muito importante para diferentes etapas de um projeto de construção de programas. Essas etapas incluem desde o início do projeto até os momentos finais, como a identificação de falhas, o planejamento da

*possamai.sam@gmail.com

†gian.rb@ufsc.br/Orientador

estrutura e fluxo da aplicação e a entrega automática de *software* ao local adequado para entrar em funcionamento.

Antes de iniciar o desenvolvimento de um programa é vital entender quais funções ele deverá ter, quais serão as principais ferramentas usadas e planejar como a estrutura dele deverá ser e como ele se comportará. Se essa etapa não for executada apropriadamente o desempenho do programa, tanto no momento de seu lançamento ou em atualizações posteriores, poderá ser comprometido. A modelagem de sistemas é um método, que por múltiplas ferramentas, permite ajudar em diversas funções no planejamento e entendimento de um projeto. Ela permite criar uma estrutura e estabelecer o comportamento do *software* em um tempo hábil muito menor do que o necessário para montar a aplicação.

Existem múltiplas linguagens e ferramentas para a modelagem de sistemas e diferentes ferramentas que podem ser escolhidas conforme a necessidade do projeto. Naturalmente a diversidade de opções também pode se tornar um empecilho em alguns casos, considerando que diferentes ferramentas podem utilizar diferentes formatos de armazenamento das informações contidas nos modelos, mas existem, no entanto, algumas outras ferramentas que ajudam na conversão de modelos para diversos formatos.

Esse projeto pretende criar uma plataforma que facilite a utilização dessas ferramentas de conversão, permitindo o uso delas em um único ambiente sem precisar lidar com a instalação e configuração das mesmas. Além disso, ele também visa permitir o armazenamento e organização de modelos nesse mesmo ambiente.

Este trabalho foi feito para descrever as decisões e ações feitas na busca de atingir o objetivo do projeto e quais resultados foram alcançados ao término dele. Para isso ele foi dividido em quatro seções. Em uma primeira seção é descrito o embasamento teórico utilizado para justificar as escolhas do projeto. Em seguida é explicitado a metodologia usada, as ferramentas e arquiteturas escolhidas e suas justificativas. Após as seções anteriores é discutido os resultados obtidos e para finalizar a conclusão obtida é sintetizada.

2 Fundamentação teórica

A escolha da arquitetura tem um impacto profundo no desenvolvimento de software dessa forma é preciso analisar qual arquitetura será mais vantajosa de ser usada para cada aplicação. Algumas delas presentes no mercado são as arquiteturas monolíticas, de microsserviços e orientada a serviços (SOA).

Em uma aplicação monolítica um único código é usado para todos os componentes da aplicação e toda a aplicação está baseada em um único *design*, não permitindo a implantação de componentes individuais (PACHGHARE, 2016).

2.1 Microsserviços

O acoplamento de múltiplos componentes de forma mais solta é muito importante para assegurar a flexibilidade do software. Isso permite um sistema leve e com alta manutenibilidade. Microsserviços permite garantir isso e, simultaneamente, facilitar a integração com novas tecnologias (SELVAKUMAR, 2019).

A arquitetura de microsserviços consiste na decomposição da aplicação em componente menores e modulares. Esses componentes trabalham juntos para exercer a função do todo, oferecendo maior escalabilidade (BANICA et al., 2017).

Esse modelo pode tornar aplicação mais flexível. [Fritzscht et al. \(2019\)](#) afirmam que sistemas monolíticos podem se tornar grandes e complexos após anos ou décadas de crescimento e na busca de maior manutenibilidade e escalabilidade, várias empresas migraram para o uso de microsserviços. Por exemplo,

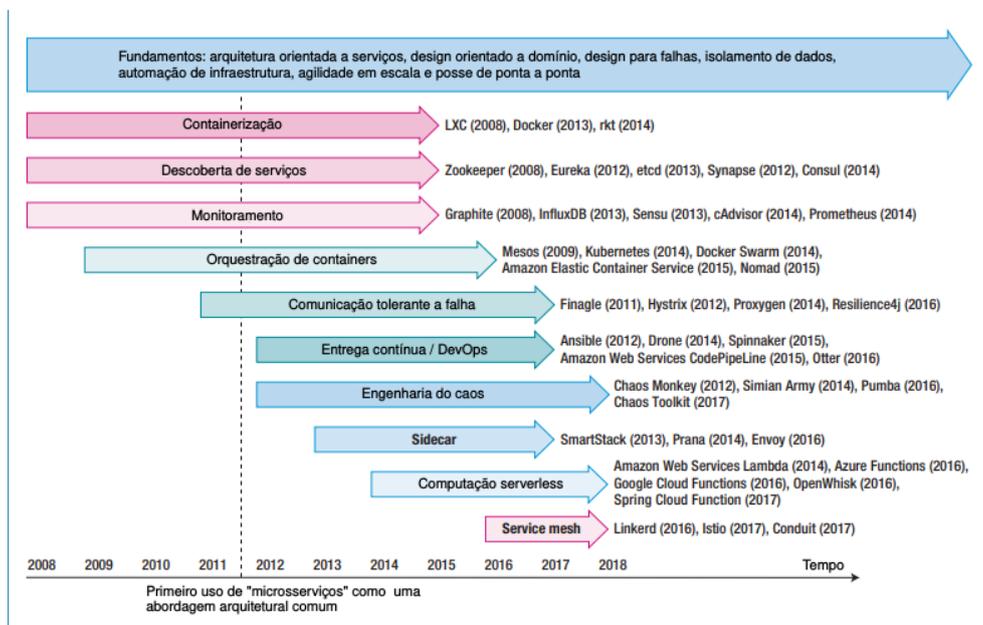
“Várias corporações proeminentes migraram para microsserviços. Netflix, eBay, Amazon, Twitter, Paypal, Sound cloud são algumas delas. Elas experimentaram imensos benefícios em manutenibilidade dos seus sistemas e desenvolvimento contínuo e integração” (selvakumarinternet, tradução nossa).

Além da comparação entre uma arquitetura de microsserviços e uma monolítica clássica é interessante constatar que essa não é a única arquitetura que constitui na divisão da aplicação em partes menores.

[Cerny, Donahoo e Trnka \(2018\)](#) explicam que tanto SOA quanto microsserviços propõem a decomposição da aplicação em múltiplos serviços que cooperam para prover a funcionalidade do todo. SOA sugere a decomposição em serviços simples enfatizando o serviço de integração, em contrapartida, microsserviços sugere a decomposição em serviços inteligentes com mecanismos de roteamento simples.

[Jamshidi et al. \(2018\)](#) classifica a evolução tecnológica, que permitiu a criação e difusão de microsserviços, em dez ondas tecnológicas, sendo que as cinco primeiras delas já existiam antes de o termo microsserviço ter sido cunhado. Elas foram as *containers*, serviços de descoberta de dispositivos, serviços de monitoramento, orquestração de *containers*, bibliotecas de comunicação tolerantes a falhas e latência, *continuous-delivery*, engenharia de caos, *Sidecar*, computação *serverless* e *mesh*. Essa evolução está ilustrada na Figura 1.

Figura 1 – Linha do tempo de tecnologias para microsserviços



Fonte: [Jamshidi et al. \(2018\)](#)

2.2 Comunicação entre microsserviços

Para a comunicação entre os serviços é possível escolher entre comunicação síncrona e assíncrona. A primeira consiste em um dos serviços mandar uma requisição para outro e esperar até este processar o resultado e retornar uma resposta. Um exemplo de ferramenta comumente usada em aplicações é REST API. A segunda forma de comunicação é feita através de um *message broker* intermediário que coordenada a comunicação entre os processos, dessa forma o serviço não espera por uma resposta direta (SHAFABAKHSH; LAGERSTRÖM; HACKS, 2020). Existem múltiplas ferramentas que podem ser usadas para comunicação assíncrona entre serviços, entre eles estão MQTT, AMQP e Kafka.

MQTT é um protocolo de transmissão de dados baseado em *publisher* e *consumer*, ele é um protocolo usado na camada de aplicação do conjunto de protocolos TCP/IP. Nele as partes que se comunicam não precisam conhecer a identidade uma da outra. Nesse modelo de comunicação são considerados dois tipos de agentes: os clientes e o *broker*, sendo os clientes chamados *publisher* ou *consumer*. Um *publisher* pode enviar mensagens, enquanto um *consumer* pode solicitar as informações relacionadas as mensagens e o *broker* reconhece e transmite mensagens entre clientes associados a elas (KASHYAP; SHARMA; GUPTA, 2018).

AMQP é um protocolo de comunicação entre máquinas que suporta tanto *request/response* quanto *publish/subscribe*. O protocolo oferece filas confiáveis, *publish/subscribe* baseado em tópicos, roteamento e transações flexíveis. Ele permite a troca de mensagens em várias formas que incluem mensagens diretas, *fan-out* e por tópico. Confiabilidade é uma de suas principais características e oferece dois níveis preliminares de qualidade de serviço (NAIK, 2017).

Kafka é um sistema de mensagens distribuído. Ele permite distribuir fluxos de mensagens entre múltiplos sub fluxos e pode ser escalável permitindo o uso de múltiplos *brokers*. Diferente da maioria, os *Brokers* Kafka são *stateless*, isso significa que a informação se o dado foi consumido ou não por algum serviço não é mantido no *broker*, mas deve ser mantido no serviço que o utiliza, isso traz vantagens e desvantagens, por um lado o *broker* não sabe se todos os destinatários receberam a mensagem, dificultando a remoção, por outro lado, isso traz mais flexibilidade aos consumidores dos dados (KREPS et al., 2011).

2.3 Tipos de servidores

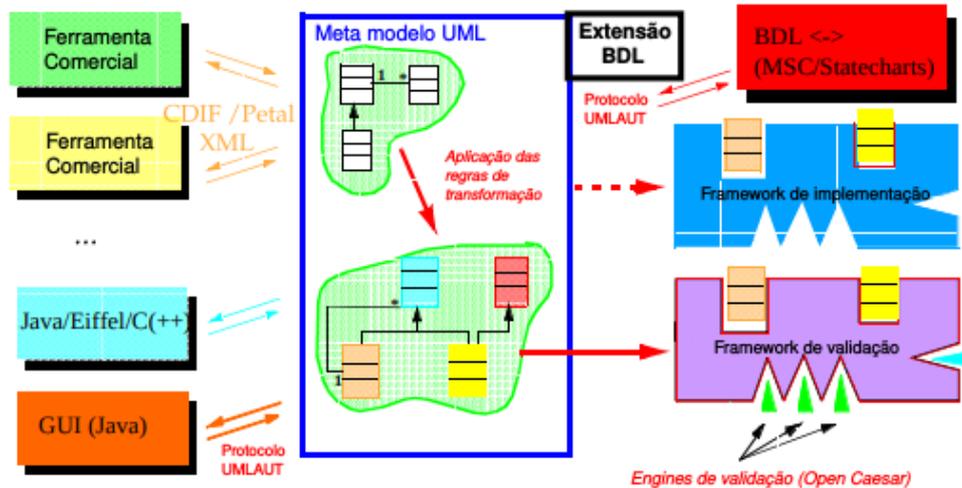
Um servidor pode ser considerado *stateless* ou *stateful*. Segundo Alonso (2015), ser *stateless* significa que cada requisição de um cliente deve conter toda informação necessária para que o servidor consiga interpretar a requisição. Usando esse paradigma dados referentes a sessão devem ser retornados ao cliente e o estado da sessão deve ser mantido por este.

Alonso (2015) afirma também que restringir o sistema a uso de *stateless* melhora a visibilidade, confiabilidade e escalabilidade, porém reduz o desempenho da rede e controle do servidor. Visibilidade é ganha, pois não a necessidade de traçar as requisições anteriores para entender todos os aspectos da requisição atual. Confiabilidade é aperfeiçoada, pois esse protocolo facilita a recuperação de falhas parciais. Não precisar armazenar o estado atual facilita a escalabilidade.

2.4 Trabalhos relacionados

O projeto UMLAUT é um *framework* de transformação de modelos que permite aplicar transformações complexas em modelos UML, na Figura 2 está representado a sua arquitetura. Eles englobam a transformação entre formatos como CDIF e MDL, geração de código em linguagens como Java e Eiffel e transformação de modelos dedicados à validação de sistemas reativos distribuídos (HO et al., 1999).

Figura 2 – Arquitetura do projeto UMLAUT



Fonte: Ho et al. (1999)

A aplicação CROSSMINER busca auxiliar o desenvolvimento de projetos de código aberto através da extensão de um projeto anterior OSSMETER. Nele feito a integração de componentes legados através de uma *API REST* atuando como um *proxy* reverso, essa solução vem de um padrão utilizado em aplicações de microsserviços (BOUDEFFA et al., 2019).

Matthias Wauer¹ e Axel-Cyrille Ngonga Ngomo descrevem GEISER, um projeto de uso de dados de sensores e geoespaciais com intuito de uso em múltiplas aplicações que incluem *geomarketing*. Por exemplo, alertando um restaurante local que uma banda estará tocando próximo, estacionamento inteligente, guiando o usuário para lugares com maior probabilidade de vagas livres e manutenção preditiva.

A meta principal desse projeto é permitir o processamento em massa de dados de sensores e geoespaciais permitindo o uso dos mesmos para aplicações diversas. A arquitetura descrita, ilustrada pela Figura 3, utiliza microsserviços e a ferramenta RabbitMQ usando o protocolo AMQP para comunicação entre serviços e com acesso a aplicação usando uma *API REST* (WAUER; SHERIF; NGOMO, 2018).

3 Materiais e métodos

As ferramentas e estruturas escolhidas para compor uma aplicação podem determinar o cumprimento ou não de qualquer objetivo estabelecido no projeto. Considerando isso é vital o entendimento e planejamento antes de iniciar o desenvolvimento de qualquer sistema. Nessa seção do trabalho é descrito o planejamento e escolhas feitas para a plataforma.

Figura 3 – Arquitetura do projeto Geiser



Fonte: Wauer, Sherif e Ngomo (2018)

3.1 Metodologia

Antes de iniciar o desenvolvimento da aplicação é necessário entender qual o objetivo do software e o que ele teria que ter para cumpri-los. A partir do problema que se busca ser corrigido foi elaborado um conjunto de requisitos funcionais e não funcionais permitindo o melhor entendimento em relação ao que o *software* deverá ou não conseguir fazer.

Após essa etapa foi iniciada a elaboração da arquitetura do software sendo essencial para determinar quais ferramentas e padrões serão utilizados no decorrer do desenvolvimento. Nela, com base nos requisitos dele, foi determinado quais componentes deveriam existir na aplicação e de que forma esses componentes estariam conectados.

Após entender os componentes que devem compor o sistema foi preciso determinar de que forma um usuário poderá se comunicar com eles e de que forma eles devem interagir entre si.

3.2 Definição de problema e requisitos

Na modelagem de sistemas, há a necessidade do uso, intercambiável, de múltiplos formatos de arquivos de modelos em ambientes variados. Um exemplo deles é o ambiente acadêmico, nele a interação entre múltiplos alunos e professores pode resultar em alto número de ferramentas utilizadas, cada uma podendo ter seu próprio formato de armazenamento e representação.

A multitude dos usos de ferramentas de modelagem que se comunicam entre si já pode ser alcançada através do uso de conversores de arquivos, porém isso cria um aumento de complexidade que pode não ser tolerada, levando a instituição de ferramentas e formatos padrões que diminuem a pluralidade e liberdade dos usuários presentes no ambiente em questão.

Esta aplicação busca facilitar esse processo de forma que a diversidade de ferramentas possam ser mantida sem aumentar, significativamente, a complexidade do processo e também busca facilitar o gerenciamento dos modelos, simplificando seu gerenciamento de armazenamento e compartilhamento.

Buscando solucionar os problemas considerados e atingir os objetivos finais foi feito o levantamento dos requisitos funcionais e não funcionais. Dessa forma foi possível entender o que a aplicação, após finalizada, busca atender e permitindo assim decidir quais ferramentas podem melhor acatar as necessidades do projeto.

Os requisitos estabelecidos consideram uma plataforma de conversão de modelos e armazenamento de modelos. Permitindo organização de modelos em grupos e projetos, além de definir permissões de acesso para diferentes usuários.

Os requisitos também incluem a proposta de uma aplicação gráfica que deverá poder se comunicar com a aplicação definida nesse trabalho, porém ela não será tratada nesse trabalho.

3.3 Arquitetura

Como é mostrada na Figura 4, a aplicação disponibiliza uma *API* externa que será usada para a comunicação diretamente com o usuário ou por uma *interface* gráfica externa. Em adição a ela, o sistema é composto de um gerenciador de usuários e permissões, um gerenciador de serviços, um gerenciador de projetos, um gerenciador de modelos, um barramento de mensagens e múltiplos serviços de transformação de modelos.

As setas saindo de *Cliente* para a *Interface* e *API* na imagem representam a possibilidade de um usuário acessar a aplicação através da *API* diretamente ou utilizando alguma outra aplicação como intermediário. Devido a isso, embora o projeto atual não disponibilize uma *interface*

Como pode ser observado na figura a comunicação entre a maioria dos processos é intermediada por um barramento de mensagens, sendo que no caso dos serviços de transformação de modelos, por ter um número variável de aplicações, precisam ser descobertas após o início da aplicação pelo gerenciador de serviços.

Um microsserviço que se destaca na imagem é gerenciador de usuário e permissões, ele estabelece uma comunicação direta com *API*. Ela é utilizada para garantir a validade das requisições que chagam a *API*.

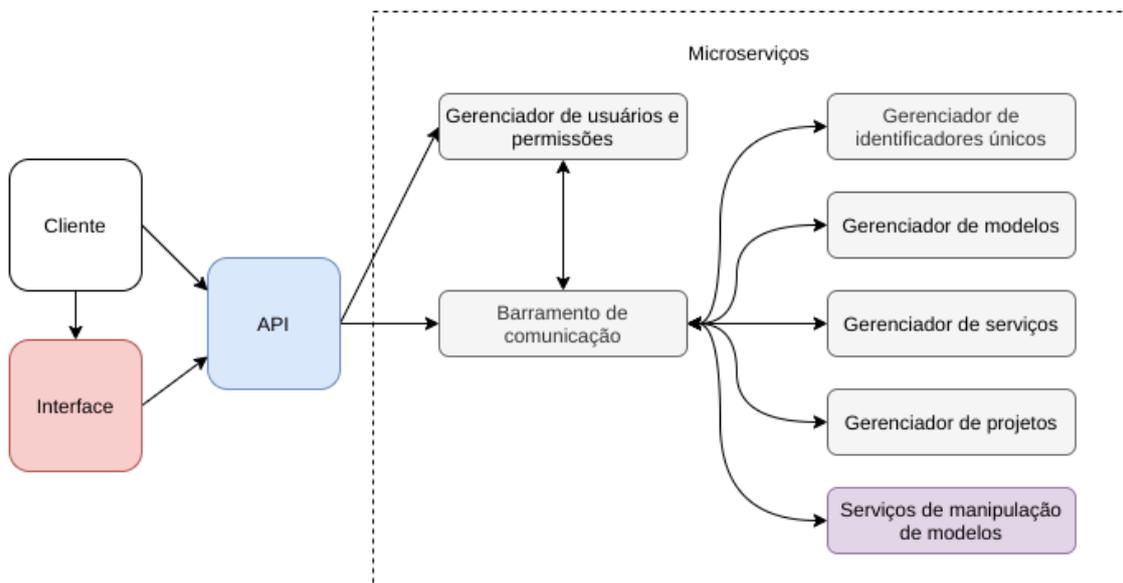
3.3.1 Definição de API

Com base nos requisitos do software foi levantado a lista de quais recursos seriam expostos ao usuário e quais informações estariam contidas neles, utilizando esse conhecimento foi possível definir uma API usando o paradigma REST e o formato JSON para representar todos os recursos que não são arquivos.

O gerenciamento de sessão é feito através de um *token* de autenticação utilizando o formato JWT. Ele poderá ser obtido através de uma rota de autenticação e deverá ser preservado no computador do usuário e enviado junto ao cabeçalho de todas as requisições.

Para descrever a API foi utilizado a especificação OpenAPI na versão 3.0, com ela é possível descrever as rotas de forma padronizada independente da linguagem utilizada

Figura 4 – Arquitetura



Fonte: elaborado pelo autor (2022)

Tabela 1 – Especificação dos métodos de requisição HTTP

Nome do método	Descrição
GET	Transfere a representação atual do recurso alvo
HEAD	Mesmo que GET, porém não transfere o conteúdo da resposta
POST	Executa processamento específico do recurso no conteúdo da requisição
PUT	Substitui toda representação atual do recurso alvo pelo conteúdo da requisição
DELETE	Remove todas as representações do recurso alvo
CONNECT	Estabelece um túnel ao servidor identificado pelo recurso alvo
OPTIONS	Descreve as opções de comunicação para o recurso alvo
TRACE	Executa um teste de retorno pelo caminho até recurso alvo

Fonte: adaptado de Nottingham (2022)

na implementação, além de poder ser utilizada para geração da documentação que pode ser disponibilizada ao usuário.

O protocolo HTTP possui múltiplos métodos de requisição, como pode ser observado na Figura 1 que atribuem diferentes sentidos semânticos a requisição.

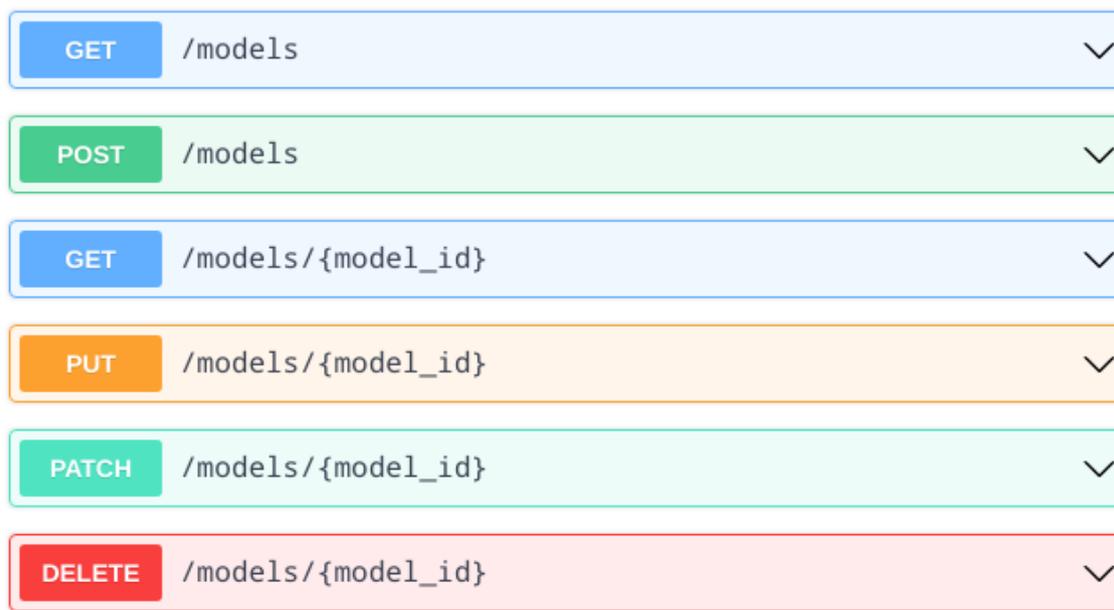
Foram atribuídos, a eles, diferentes significados no escopo da utilização da API. Foi definido que o método *GET* representa requisição de informação, *POST* representando a inserção de recurso, *PUT* indica a substituição total, *PATCH* serve para a atualização parcial dos dados e *DELETE* seria usado para excluir.

Baseando no padrão REST as rotas da *interface* são organizadas com base no conceito de recursos, por exemplo, um modelo é tratado como um recurso e duas rotas são associadas a ele. A primeira delas, utilizando o formato */models/*, contém a representação de coleção de modelos, dessa forma o método *GET* resulta no retorno de múltiplos e o método *POST* na criação de um único. A segunda, utilizando o formato */models/<model_id>*,

representa um único modelo e o método de *GET* retorna informações dele, o *PUT* substitui todos os dados dele, o *PATCH* substitui parte das informações dele e o *DELETE* o deleta.

A Figura 5 mostra a visualização das rotas de modelos através de documentação gerada pelo software Swagger. A Figura 6 mostra os parâmetros para o rota */models* conforme documentação gerada.

Figura 5 – Rotas de modelos na API



GET	/models	▼
POST	/models	▼
GET	/models/{model_id}	▼
PUT	/models/{model_id}	▼
PATCH	/models/{model_id}	▼
DELETE	/models/{model_id}	▼

Fonte: elaborado pelo autor (2022)

Além dos recursos, foi utilizado também a representação de ações, isso foi utilizado para as transformações. Nesse cenário a ação é tratada de forma similar a um recurso sendo definido que deve-se usar o método *PUT* caso a ação não altere o estado da aplicação e *POST* caso ela o faça.

A Figura 7 ilustra um exemplo de uso da *API*. Se usuário quiser obter as informações de um modelo, ele deverá, conforme descrito anteriormente, utilizar a rota */models/<model_id>* e fazer uma requisição com método *GET*, substituindo *<model_id>* pela identificação do modelo desejado.

Para que a requisição tenha sucesso é preciso ter acesso ao modelo, caso o modelo não seja público isso irá requerer a aquisição de um *token JWT*, como mostrado na Figura 7. Para obter o *token* é preciso uma requisição *POST* a rota */auth*. A utilização de *POST* para essa rota implica na criação de um novo *token*. O *token* deve ser encaminhado junto as requisições através do cabeçalho *HTTP Authorization*, passando o valor *Bearer <token>*, substituindo *<token>* pelo valor correspondente.

Na Figura 7 foi mostrada uma requisição a rota */models* antes de acessar */models/<model_id>* com intuito de representar que o *id* associado ao modelo poderá ser encontrado utilizando requisições para rotas que retornem modelos.

Figura 6 – Parâmetro do método *GET* de */models*

public string (query)	Filter for public or private models Available values : true, false <input type="text" value="--"/>
name string (query)	Filter models using their names <input type="text" value="name"/>
limit string (query)	Maximum amount of models returned <input type="text" value="limit"/>
offset string (query)	Offset for pagination <input type="text" value="offset"/>
order_by_column string (query)	Columns used in sorting <input type="text" value="order_by_column"/>
order_by_asc string (query)	Sort using ascending order if true Available values : true, false <input type="text" value="--"/>

Fonte: elaborado pelo autor (2022)

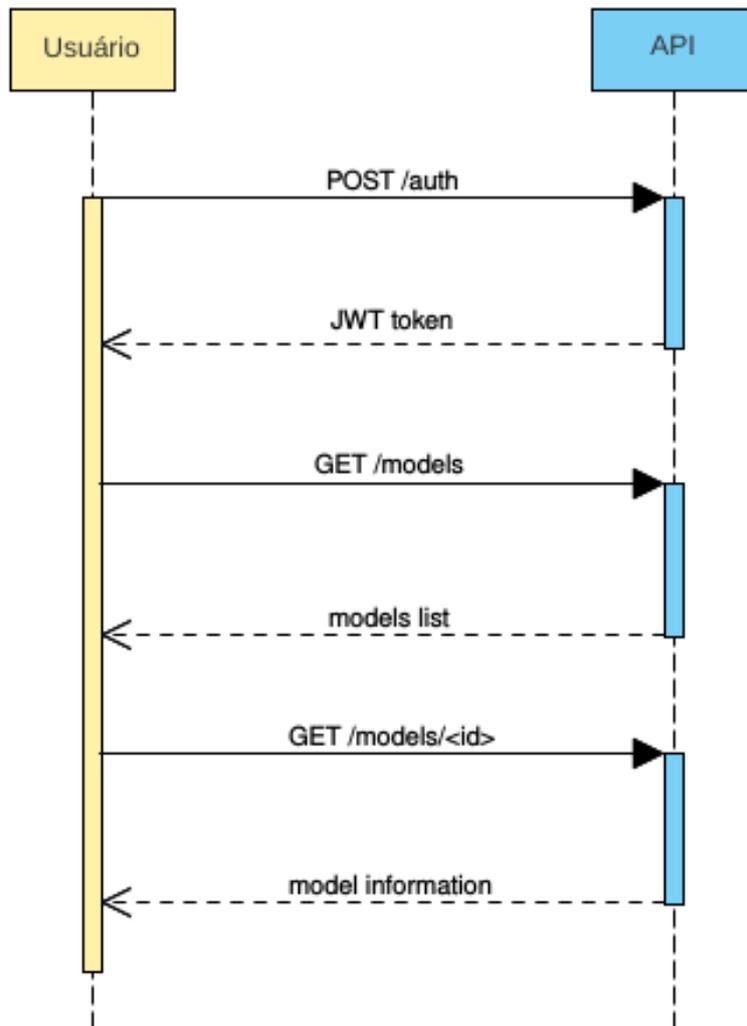
3.3.2 Serviços

A arquitetura de microsserviços foi escolhida para essa aplicação, dessa forma ela abrigará diversos serviços. Para os serviços criados para ela foi escolhida a linguagem de programação Python. Alguns serviços também são utilizados, entre eles MySQL como um banco de dados *SQL*, OpenStack Swift usado para armazenamento de arquivos e RabbitMQ como um barramento de comunicação.

O gerenciador de *tokens* de registros desempenha um papel simples, nele é feita a criação de *tokens* de identificação que visam ser usados para identificar registros tanto para aplicações internas quanto para outros serviços que fazem parte da aplicação, usando um identificador único para aquele recurso que não colide com nenhum outro presente no sistema.

O gerenciador de usuários e permissões é responsável pelo armazenamento das informações associadas aos usuários, através dele é feita a geração e checagem de autenti-

Figura 7 – Exemplo de uso da API



Fonte: elaborado pelo autor (2022)

cidade do *token* JWT. Esse serviço usa um esquema no MySQL dedicado a ela, porém não utiliza o barramento para comunicação, disponibilizando apenas comunicação síncrona através de uma API interna.

Está presente um gerenciador de arquivos servindo de ponte entre o serviço de armazenamento de arquivos escolhido e outros serviços do sistema. Ele usa armazenamento SQL para lidar com a indexação dos arquivos e lidar com a contagem de referências.

O gerenciador de projetos é responsável pelo gerenciamento de todos os dados da estrutura dos grupos, projetos e modelos dos usuários. Nele são geridas a relação de pertencimento desses dados, por exemplo, qual grupo contém o projeto e qual o dono do grupo. Esse serviço também gerencia permissões de nível de grupo, barrando requisições de usuários que não possam ter acesso a um grupo, projeto ou modelo específico com base nas configurações dos mesmos e das informações advindas do gerenciador de usuários e

permissões. As informações são guardadas através do serviço SQL escolhido.

O serviço de *API* serve como uma *interface* para usuários e aplicações externas, servindo como entrada e saída para a aplicação total. Qualquer agente externo deverá poder apenas se comunicar com esse serviço, não sendo possível o acesso direto a qualquer outro.

Cada função de transformação de modelos é lançada como um serviço separado que pode ser integrado ao restante das aplicações através do barramento de comunicação. Elas são organizadas através do gerenciador de serviços que irá conter a fila de requisições de transformação e irá organizar e repassar para o serviço de transformação correspondente qual arquivo ele deverá transformar.

3.3.3 Comunicação entre serviços

Os serviços se comunicam entre si indiretamente por um *broker* RabbitMQ. Esta ferramenta suporta múltiplas formas de comunicação, dentre elas AMQP e MQTT, sendo a escolhida AMQP. Apenas a comunicação entre a API e o gerenciador de usuários e permissões utiliza outro meio de comunicação, o HTTP.

A escolha do protocolo se deve primeiramente pela opção do uso de comunicação assíncrona para a comunicação entre os diversos serviços. Essa escolha foi feita para lidar mais facilmente com a multitude de serviços que compõem a aplicação, considerando a necessidade de extensão do número de serviços conforme o aumento do número de opções de transformação e ter que lidar com a possibilidade de falha deles.

4 Resultados e discussões

Após o planejamento a primeira etapa no desenvolvimento do projeto foi encontrar e testar o serviço de comunicação. Isso foi possível através de uma imagem *docker* (RabbitMQ), que pode ser utilizada através do comando descrito na Figura 9, dessa forma simplificando essa etapa do trabalho, ela também fornece uma *interface* que auxilia em testes durante o desenvolvimento.

A conexão com o servidor de banco de dados foi feita utilizando a biblioteca *SQLAlchemy* em *Python*, usando ela é possível descrever as tabelas utilizadas utilizando o conceito *ORM*, exemplificado na Figura 10. Ele consiste em uma técnica de abstração do *SQL*, uma linguagem utilizada para realizar operações em bancos de dados relacionais, tornando mais simples a interação com os registros. Essa descrição da estrutura das tabelas também é usada pelo sistema de versionamento do banco de dados utilizado, chamado *alembic*, que permite a criação e atualização delas.

A estruturação do código, ilustrada na Figura 11, seguiu uma organização similar para todos os serviços, cada um deles é separado um diretório, dentro dele a pasta chamada *src* contém o código-fonte do programa, o arquivo *requirements.txt* contém as dependências que podem ser instaladas através do gerenciador de pacotes *Pip*, o arquivo *config.py* guarda as configurações e a pasta *migrations* armazena as informações necessárias para o gerenciador de versão do banco de dados.

Na pasta *src* é possível encontrar uma maior diversidade de conteúdo que varia entre os microsserviços, porém é comum estar presente um arquivo `__main__.py` usado para inicializar o programa e uma pasta *models* onde os modelos *ORM* usados estão

Figura 8 – Estrutura da mensagem entre serviços

sender:	api																
receiver:	project_manager																
version:	v0.1.0																
request_id:	0																
message_id:	0																
header:	<table><tr><td>user:</td><td><table><tr><td>id:</td><td>0</td></tr><tr><td>valid:</td><td>0</td></tr><tr><td>authorizations:</td><td></td></tr></table></td></tr></table>	user:	<table><tr><td>id:</td><td>0</td></tr><tr><td>valid:</td><td>0</td></tr><tr><td>authorizations:</td><td></td></tr></table>	id:	0	valid:	0	authorizations:									
user:	<table><tr><td>id:</td><td>0</td></tr><tr><td>valid:</td><td>0</td></tr><tr><td>authorizations:</td><td></td></tr></table>	id:	0	valid:	0	authorizations:											
id:	0																
valid:	0																
authorizations:																	
data:	<table><tr><td>select:</td><td><table><tr><td>limit:</td><td>0</td></tr><tr><td>offset:</td><td>0</td></tr><tr><td>search:</td><td><table><tr><td>name:</td><td>busca</td></tr></table></td></tr><tr><td>orderby:</td><td><table><tr><td>column:</td><td></td></tr><tr><td>asc:</td><td>true</td></tr></table></td></tr></table></td></tr></table>	select:	<table><tr><td>limit:</td><td>0</td></tr><tr><td>offset:</td><td>0</td></tr><tr><td>search:</td><td><table><tr><td>name:</td><td>busca</td></tr></table></td></tr><tr><td>orderby:</td><td><table><tr><td>column:</td><td></td></tr><tr><td>asc:</td><td>true</td></tr></table></td></tr></table>	limit:	0	offset:	0	search:	<table><tr><td>name:</td><td>busca</td></tr></table>	name:	busca	orderby:	<table><tr><td>column:</td><td></td></tr><tr><td>asc:</td><td>true</td></tr></table>	column:		asc:	true
select:	<table><tr><td>limit:</td><td>0</td></tr><tr><td>offset:</td><td>0</td></tr><tr><td>search:</td><td><table><tr><td>name:</td><td>busca</td></tr></table></td></tr><tr><td>orderby:</td><td><table><tr><td>column:</td><td></td></tr><tr><td>asc:</td><td>true</td></tr></table></td></tr></table>	limit:	0	offset:	0	search:	<table><tr><td>name:</td><td>busca</td></tr></table>	name:	busca	orderby:	<table><tr><td>column:</td><td></td></tr><tr><td>asc:</td><td>true</td></tr></table>	column:		asc:	true		
limit:	0																
offset:	0																
search:	<table><tr><td>name:</td><td>busca</td></tr></table>	name:	busca														
name:	busca																
orderby:	<table><tr><td>column:</td><td></td></tr><tr><td>asc:</td><td>true</td></tr></table>	column:		asc:	true												
column:																	
asc:	true																

Fonte: elaborado pelo autor (2022)

Figura 9 – Comando utilizado para usar o docker do RabbitMQ

```
1 docker pull rabbitmq:3.9-management
2 docker run -d -it --restart unless-stopped --name rabbitmq -p 5672:5672 \
3   -p 15672:15672 rabbitmq:3.9-management
```

Fonte: elaborado pelo autor (2022)

estruturados. Assim pode-se encontrar o diretório *routes* para os serviços que exponham rotas HTTP, nele são contidos os códigos chamados quando essas rotas são requisitadas.

A arquitetura de microsserviços permitiu que o desenvolvimento de cada serviço fosse criado de forma individual sem a necessidade do funcionamento completo do todo. E por meio da interação com o barramento de comunicação foi possível emular o envio de mensagens de outros sistemas facilitando o desenvolvimento ao impedir que o mau funcionamento de um deles afete aquele que está sendo desenvolvido e permitindo buscar individualmente em cada um deles caso uma falha no sistema seja detectada. O mesmo comportamento também poderia ser aplicado para as comunicações síncronas que fazem

Figura 10 – Exemplo de representação *ORM*

```
1 class File(Base):
2     __tablename__ = 'files'
3
4     id = Column(Integer(), primary_key=True)
5     filename = Column(String(64), nullable=False)
6     content_type = Column(String(255), nullable=False)
7     id_token = Column(String(20), nullable=False)
8     count = Column(Integer(), nullable=False, server_default=text('1'))
9     deleted = Column(Enum('true', 'false'), nullable=False,
10                      server_default=text("'false'"))
11     created_at = Column(DateTime, nullable=False,
12                        server_default=text('current_timestamp()'))
13     updated_at = Column(DateTime, onupdate=text('current_timestamp()'))
14     storage_idx = Column(ForeignKey('storages.id'), index=True, nullable=True)
15
16     storage = relationship('Storage')
17
18     def put(self, content):
19         self.storage.getInterface().put(self.filename, content)
20
21     def read(self):
22         return self.storage.getInterface().get(self.filename)
```

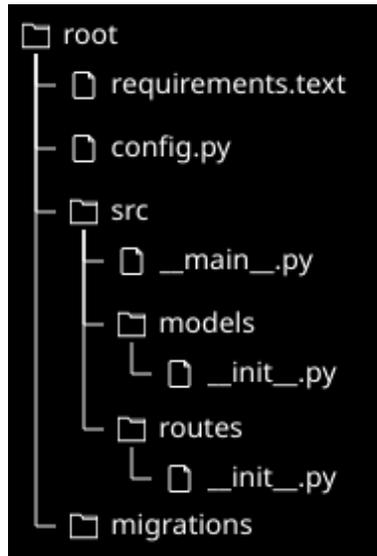
Fonte: elaborado pelo autor (2022)

parte do projeto através de um servidor *dummy*, porém isso foi julgado não ser necessário para a aplicação.

O gerenciador de *tokens* de registros, que está localizado na pasta *unique_tokens*, se mostrou um serviço com menor complexidade de comunicação, através dele é possível fazer apenas duas operações que são: através das informações do registro buscar o *token* correspondente, nesse caso será criado caso ele não exista, e através do *token* obter os dados do registro. Eles são gerados de forma aleatória utilizando a biblioteca *secrets* e para garantir a unicidade deles é verificado se já está em uso, caso já esteja, um novo é gerado até ser encontrado um que esteja livre. As informações de um registro utilizadas pelo gerador são o dono que representa qual serviço detém o registro, o tipo que é um identificador de qual categoria de conteúdo está atrelado a ele, que é, em sua maioria, uma indicação da tabela onde ele está e um identificador local que contém como o serviço o identifica, sendo na prática, normalmente, o *id* no banco de dados.

O diretório *storage* contém o gerenciador de arquivos. Ele gerencia a comunicação com o serviço de armazenamento de arquivos, para isso ele contém uma tabela com as configurações para realizar a comunicação com eles, essa configuração é utilizada para selecionar uma classe de *interface* para com o serviço de armazenamento adequado e

Figura 11 – Estrutura padrão de diretórios



Fonte: elaborado pelo autor (2022)

os parâmetros passados a ela. No momento são suportadas apenas o uso de *Swift* e o sistema de arquivos local, sendo o primeiro a forma recomendada para o uso em produção. As operações permitidas são de inserção, leitura e deleção de arquivos, não permitindo modificar um arquivo já existente.

Através do gerenciador de permissões é possível criar, deletar e alterar usuários, além de gerar e verificar *tokens JWT* usados para validação de autenticidade de requisições. Para manipulá-los, foi utilizado a biblioteca *PyJWT*, a criação é ilustrada na Figura 12. Para isso ele expõe uma *API* de uso interno seguindo o padrão *REST*. Para garantir a segurança do sistema é importante que, ao disponibilizar em produção, a comunicação entre a *API* externa sejam seguras e ele responda apenas requisições provenientes dela, alguns fatores que podem ajudar nisso é garantir que somente a máquina que contém ela possa fazer requisições a ele ou estejam localizados na mesma máquina e que a chave e usuário da autenticação *Basic* utilizada entre os dois tenha as devidas permissões.

A implementação dos diferentes componentes foi iniciada de forma separada, porém a integração deles no atual estado de desenvolvimento é extremamente limitada, dessa forma, atualmente, não é possível utilizar das funções do software, por exceção da autenticação, através da *API*.

4.1 Limitações

Uma limitação do projeto atual é não lidar de forma otimizada com a atualização de modelos já existentes. É possível, no entanto, em uma versão futura, por codificação delta minimizar esse problema, porém não foi considerado em nenhuma etapa de planejamento ou desenvolvimento do trabalho.

Segundo a arquitetura atual, para adicionar novas conversões de arquivos é necessário ter acesso direto ao barramento de comunicação. Devido a isso a adição de novos serviços fica a cargo de usuários ou ferramentas com acesso ao barramento de comunicação, sem ser possível fazer a adição por meio do próprio serviço.

Figura 12 – Código de criação de *token JWT*

```
1 class File(Base):
2     __tablename__ = 'files'
3
4     id = Column(Integer(), primary_key=True)
5     filename = Column(String(64), nullable=False)
6     content_type = Column(String(255), nullable=False)
7     id_token = Column(String(20), nullable=False)
8     count = Column(Integer(), nullable=False, server_default=text('1'))
9     deleted = Column(Enum('true', 'false'), nullable=False,
10                       server_default=text("'false'"))
11     created_at = Column(DateTime, nullable=False,
12                          server_default=text('current_timestamp()'))
13     updated_at = Column(DateTime, onupdate=text('current_timestamp()'))
14     storage_idx = Column(ForeignKey('storages.id'), index=True, nullable=True)
15
16     storage = relationship('Storage')
17
18     def put(self, content):
19         self.storage.getInterface().put(self.filename, content)
20
21     def read(self):
22         return self.storage.getInterface().get(self.filename)
```

Fonte: elaborado pelo autor (2022)

5 Conclusão

No desenvolvimento de software, principalmente em grandes projetos, o planejamento é essencial para obter resultados satisfatórios, devido a isso o maior enfoque no desenvolvimento do presente trabalho foi buscar entender a melhor forma de atender os objetivos estabelecidos.

Para isso foi necessário estabelecer primeiramente quais requisitos a aplicação deveria cumprir para então determinar quais meios e ferramentas poderiam ser utilizadas para assegurar o cumprimento deles.

Também foi estabelecido a definição da API, estabelecendo uma via de entrada e saída da aplicação que pode ser usada para extensão da mesma para disponibilizar uma interface gráfica com o usuário. No período estabelecido foi possível também iniciar a execução da implementação do sistema, estabelecendo a estrutura base do projeto e experimentação de parte das ferramentas consideradas.

Dessa forma, o desenvolvimento desse trabalho resultou na realização de um estudo das necessidades da plataforma e quais ferramentas e métodos podem ser usados para atendê-las, na elaboração de um padrão da *interface* de comunicação externa e interna e

no início do desenvolvimento dos serviços.

Referências

- ALONSO, F. S. Development of a restful api: hateoas & driven api. Turun ammattikorkeakoulu, 2015. Citado na página 4.
- BANICA, L. et al. Leveraging the microservice architecture for next-generation iot applications. *Scientific Bulletin–Economic Sciences*, v. 16, n. 2, p. 26–32, 2017. Citado na página 2.
- BOUDEFFA, A. et al. Integrating and deploying heterogeneous components by means of a microservices architecture in the crossminer project. In: *STAF (Co-Located Events)*. [S.l.: s.n.], 2019. p. 67–72. Citado na página 5.
- CERNY, T.; DONAHOO, M. J.; TRNKA, M. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review*, ACM New York, NY, USA, v. 17, n. 4, p. 29–45, 2018. Citado na página 3.
- FRITZSCH, J. et al. Microservices migration in industry: intentions, strategies, and challenges. In: IEEE. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.], 2019. p. 481–490. Citado na página 3.
- HO, W. M. et al. Umlaut: an extendible uml transformation framework. In: IEEE. *14th IEEE International Conference on Automated Software Engineering*. [S.l.], 1999. p. 275–278. Citado na página 5.
- JAMSHIDI, P. et al. Microservices: The journey so far and challenges ahead. *IEEE Software*, v. 35, n. 3, p. 24–35, May 2018. ISSN 1937-4194. Citado na página 3.
- KASHYAP, M.; SHARMA, V.; GUPTA, N. Taking mqtt and nodemcu to iot: communication in internet of things. *Procedia computer science*, Elsevier, v. 132, p. 1611–1618, 2018. Citado na página 4.
- KREPS, J. et al. Kafka: A distributed messaging system for log processing. In: *Proceedings of the NetDB*. [S.l.: s.n.], 2011. v. 11, p. 1–7. Citado na página 4.
- NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In: IEEE. *2017 IEEE international systems engineering symposium (ISSE)*. [S.l.], 2017. p. 1–7. Citado na página 4.
- NOTTINGHAM, M. Http semantics. 2022. Citado na página 8.
- PACHGHARE, V. K. Microservices architecture for cloud computing. *architecture*, v. 3, p. 4, 2016. Citado na página 2.
- SELVAKUMAR, G. Internet of things and microservices architecture for modern supply chains. 2019. Citado na página 2.
- SHAFABAKHSH, B.; LAGERSTRÖM, R.; HACKS, S. Evaluating the impact of inter process communication in microservice architectures. In: *QuASoQ@ APSEC*. [S.l.: s.n.], 2020. p. 55–63. Citado na página 4.

WAUER, M.; SHERIF, M. A.; NGOMO, A.-C. N. Towards a semantic message-driven microservice platform for geospatial and sensor data. In: *GeoLD-QuWeDa@ ESWC*. [S.l.: s.n.], 2018. p. 47–58. Citado 2 vezes nas páginas 5 e 6.

**SOFTWARE PARA CONVERSÃO DE MODELOS
DEFINIÇÃO DE REQUISITOS**

Versão 1.0

27/05/2022

HISTÓRICO DE VERSÕES

Versão	Implementada por	Data da Revisão	Motivo da Revisão
1.0	Samuel Possamai	19/10/2021	Definição Inicial de Requisitos
1.0	Danilo Silva	29/11/2021	Padronização Inicial de Requisitos
1.0	Danilo Silva	27/05/2022	Revisão dos Requisitos e Inserção da Introdução

Sumário

1. INTRODUÇÃO	4
1.1. PROPÓSITO DA DEFINIÇÃO DE REQUISITOS FUNCIONAIS	4
1.2. PROPÓSITO DA DEFINIÇÃO DE REQUISITOS NÃO-FUNCIONAIS	4
2. RESUMO DE REQUISITOS DE PRODUTO	4
3. REQUISITOS FUNCIONAIS	5
3.1. REQUISITOS FUNCIONAIS DE USO	5
3.2. REQUISITOS FUNCIONAIS DE ACESSO	5
3.3. REQUISITOS FUNCIONAIS DE INTERFACE	6
4. REQUISITOS NÃO FUNCIONAIS	6
4.1. REQUISITOS NÃO FUNCIONAIS DE USO	6

1. INTRODUÇÃO

1.1. PROPÓSITO DA DEFINIÇÃO DE REQUISITOS FUNCIONAIS

Requisitos funcionais tem o propósito de listar os fatores básicos de atendimento do escopo do produto e suas ramificações imediatas, buscando visualizar o essencial para o funcionamento legal, seguro, viável, e útil do produto.

1.2. PROPÓSITO DA DEFINIÇÃO DE REQUISITOS NÃO-FUNCIONAIS

Requisitos não-funcionais tem o propósito de: expandir as ramificações dos requisitos funcionais, definindo os fatores constituintes e/ou apoiadores do essencial; definir a moldura geral de fatores secundários que possuem como objetivo expandir a gama de funcionalidades do produto; e construir a base da existência do funcionamento mínimo do produto.

2. RESUMO DE REQUISITOS DE PRODUTO

As características gerais do produto estão baseadas em um software capaz de cadastrar usuários suportando diferentes níveis de acesso, onde a aplicação deve ser capaz de suportar o carregamento, download, exclusão e conversão de diferentes tipos de modelos. Para ser possível a conversão de modelos o usuário deve utilizar sequências de conversão, de modo que, seja indicado o tipo de modelo de entrada e o tipo de modelo de saída que é suportado na plataforma. Modelos e Sequências de conversão são agrupadas dentro de um projeto, sendo que um projeto, modelo ou sequência de conversão pode ser criado, editado, excluído ou copiado. Projetos são públicos ou privados e podem ser agrupados dentro de grupos onde grupos podem ser compartilhados com outros usuários cadastrados na plataforma.

3. REQUISITOS FUNCIONAIS

3.1. REQUISITOS FUNCIONAIS DE USO

- **RFUS-1.1** - Armazenar modelos no servidor;
- **RFUS-1.2** - Permitir transformação entre os modelos suportados;
- **RFUS-1.3** - Modelos podem ser agrupados para diferentes grupos;
- **RFUS-1.4** - Permitir grupos dentro de grupos;
- **RFUS-1.5** - O sistema permite registro de "Usuário individual";
- **RFUS-1.6** - As restrições de armazenamento e transformação são por projeto;
- **RFUS-1.7** - Um projeto é um conjunto de modelos pertencente a um grupo ou usuário restrito as operações permitidas;
- **RFUS-1.8** - Um projeto pode ser criado a partir de outro projeto desde que o usuário ou grupo tenha acesso a ele;
- **RFUS-1.9** - Um projeto pode ser público ou privado, se for privado e pertencer ao um grupo ele poderá ser acessado pelos membros do grupo e dos subgrupos desde que o usuário responsável permita;
- **RFUS-1.10** - Projetos públicos podem ser consultados por qualquer usuário;
- **RFUS-1.11** - Um projeto pode ser manipulado pelos usuários que tem permissão;
- **RFUS-1.12** - Um projeto tem um usuário responsável;
- **RFUS-1.13** - Permitir visualização de modelos;
- **RFUS-1.14** - Permitir edição de modelos.

3.2. REQUISITOS FUNCIONAIS DE ACESSO

- **RFAC-2.1** - Suportar diferentes tipos de usuários:
 - o **RFAC-2.1.1 - Usuário não registrado:** pode consultar o repositório de modelos públicos;
 - o **RFAC-2.1.2 - Usuário individual:** com as mesmas atribuições do RFAC-2.1.1, porém com acesso ao serviço de armazenamento e transformação;
 - o **RFAC-2.1.3 - Usuário membro de grupo:** com as mesmas atribuições do RFAC-2.1.1, porém com acesso ao serviço de armazenamento e transformação que o "usuário responsável por grupo" permitir;
 - o **RFAC-2.1.4 - Usuário responsável por grupo:** com as mesmas atribuições do RFAC-2.1.2, porém pode definir permissão para usuários do grupo e gerenciar usuários do grupo e grupos;
 - o **RFAC-2.1.5 - Usuário administrador:** com as atribuições dos usuários anteriores, porém pode gerenciar todos os usuários e gerenciar o uso do sistema.
- **RFAC-2.2 - Usuário membro de grupo** poderá escolher uma ou mais sequência de regras de transformações que o responsável pelo grupo permitiu;
- **RFAC-2.3** - Gerenciar usuário consiste em poder criar/alterar/excluir/bloquear e desbloquear usuários;
- **RFAC-2.4** - Gerenciar uso do sistema consiste em verificar a utilização dos serviços de transformação e disponibilizar novos serviços de transformação, cancelar serviços de transformação e acompanhar como os usuários usam o sistema;

-
- **RFAC-2.5** - Mais de um usuário pode manipular o projeto, desde que tenha permissão;
 - **RFAC-2.6** - Um usuário pode convidar outro usuário para participar do projeto;
 - **RFAC-2.7** - O usuário responsável pelo grupo tem acesso irrestrito a todos os projetos do grupo;
 - **RFAC-2.8** - O usuário responsável por um projeto:
 - o **RFAC-2.8.1** - pode definir a permissão de leitura ou leitura/escrita para um usuário convidado;
 - o **RFAC-2.8.2** - pode remover a permissão de um usuário convidado;
 - o **RFAC-2.8.3** - pode transferir um projeto para outro usuário com concordância do usuário destino;
 - o **RFAC-2.8.4** - pode excluir o projeto;
 - o **RFAC-2.8.5** - pode gerenciar um projeto, como nome do projeto e serviços que o projeto vai usar;
 - o **RFAC-2.8.6** - vai poder escolher uma ou uma sequência de regras de transformações a ser aplicados nos modelos dos projetos.
 - **RFAC-2.9** - Número de membros em um grupo pode ser alterado pelo administrador.

3.3. REQUISITOS FUNCIONAIS DE INTERFACE

- **RFIT-3.1** – Suporte para outros idiomas;
- **RFIT-3.2** – Suporte para tema no modo claro e escuro;
- **RFIT-3.2** – Aplicação responsiva em Desktop e Tablet;
- **RFIT-3.3** – Painel de gestão com estatísticas para uso do administrador.

4. REQUISITOS NÃO FUNCIONAIS

4.1. REQUISITOS NÃO FUNCIONAIS DE USO

- **RNFUS-1.1** - Manter a privacidade entre os dados dos usuários;
- **RNFUS-1.2** - Funções devem ser utilizáveis através do site e através de uma API Rest;
- **RNFUS-1.3** - A disponibilização de novos serviços deve ser via interface de administração do sistema;
- **RNFUS-1.4** - Limite de modelos no armazenamento de usuário pode ser imposto por um administrador;
- **RNFUS-1.5** - O usuário responsável pelo grupo pode aplicar permissões a membros, grupo de membros ou a todos os membros;
- **RNFUS-1.6** - O sistema tem uma cota padrão por usuário, gerenciado pelo administrador de 5MB;
- **RNFUS-1.7** - Um grupo pode ter no máximo 250 membros;
- **RNFUS-1.8** - Um grupo pode ter no máximo de 50 subgrupos.
- **RNFUS-1.9** - Evitar perdas na transformação de modelos quando possível;
- **RNFUS-1.10** - Suportar três dos principais formatos de modelos;

-
- **RNFUS-1.11** - Compatibilidade com as últimas versões dos browsers "Google Chrome" e "Mozilla Firefox";
 - **RNFUS-1.12** - Deve ser usado TLS.

APÊNDICE B - DOCUMENTO DE ESPECIFICAÇÃO DA API

```
1 openapi: 3.0.0
2 info:
3   title: API
4   description: Descrição.
5   version: 0.1.0
6 paths:
7   /auth:
8     post:
9       description: Get the authorization token
10      requestBody:
11        required: true
12        content:
13          application/json:
14            schema:
15              type: object
16              properties:
17                email:
18                  type: string
19                  description: User email
20                password:
21                  type: string
22                  description: User password
23      responses:
24        '200':
25          description: Successfully return a list of the models.
26          content:
27            application/json:
28              schema:
29                type: object
30                properties:
31                  auth_token:
32                    type: string
33                    description: JWT token
34        '400':
35          description: Invalid authentication.
36          content:
37            application/json:
38              schema:
39                $ref: '#/components/schemas/error'
40  /profile:
41    get:
42      description: Returns token user information
43      responses:
44        '200':
45          description: Successfully return users information.
46          content:
47            application/json:
48              schema:
49                type: object
50                properties:
51                  email:
52                    type: string
53                    description: User email
54                  first_name:
55                    type: string
56                    description: User first name
57                  last_name:
58                    type: string
59                    description: User last name
60                  locale:
61                    type: string
62                    description: User idiom locale
```

```

63         theme:
64             type: string
65             description: Interface theme
66     '401':
67         description: Invalid credentials.
68         content:
69             application/json:
70                 schema:
71                     $ref: '#/components/schemas/error'
72     security:
73         - bearerAuth: []
74 /models:
75     get:
76         description: Returns all models
77         parameters:
78             - in: query
79               name: public
80               description: Filter for public or private models
81               schema:
82                 type: string
83                 enum:
84                     - 'true'
85                     - 'false'
86                 required: false
87             - in: query
88               name: name
89               description: Filter models using their names
90               schema:
91                 type: string
92                 required: false
93             - in: query
94               name: limit
95               description: Maximum amount of models returned
96               schema:
97                 type: string
98                 required: false
99             - in: query
100              name: offset
101              description: Offset for pagination
102              schema:
103                type: string
104                required: false
105             - in: query
106              name: order_by_column
107              description: Columns used in sorting
108              schema:
109                type: string
110                required: false
111             - in: query
112              name: order_by_asc
113              description: Sort using ascending order if true
114              schema:
115                type: string
116                enum:
117                    - 'true'
118                    - 'false'
119                required: false
120     responses:
121         '200':
122             description: Successfully return a list of the models.
123             content:
124                 application/json:
125                     schema:
126                         type: object
127                         properties:
128                             total_quantity:
129                                 type: integer
130                                 description: Total quantity of models
131                             filtered_quantity:

```

```

132         type: integer
133         description: Quantity of models after filtering
134     data:
135         type: array
136         items:
137             $ref: '#/components/schemas/model'
138     '401':
139         description: Invalid credentials.
140         content:
141             application/json:
142                 schema:
143                     $ref: '#/components/schemas/error'
144     security:
145         - bearerAuth: []
146     post:
147         description: Create a new model
148         requestBody:
149             required: true
150             content:
151                 application/json:
152                     schema:
153                         $ref: '#/components/schemas/model_create'
154     responses:
155         '201':
156             description: Successfully created the model.
157             content:
158                 application/json:
159                     schema:
160                         $ref: '#/components/schemas/model'
161         '400':
162             description: Invalid parameters.
163             content:
164                 application/json:
165                     schema:
166                         $ref: '#/components/schemas/error'
167         '401':
168             description: Invalid credentials.
169             content:
170                 application/json:
171                     schema:
172                         $ref: '#/components/schemas/error'
173     security:
174         - bearerAuth: []
175 /models/{model_id}:
176     get:
177         description: Returns a single model
178         parameters:
179             - in: path
180               name: model_id
181               description: Model id
182               schema:
183                 type: string
184                 required: true
185         responses:
186             '200':
187                 description: Successfully return the model.
188                 content:
189                     application/json:
190                         schema:
191                             $ref: '#/components/schemas/model'
192             '401':
193                 description: Invalid credentials.
194                 content:
195                     application/json:
196                         schema:
197                             $ref: '#/components/schemas/error'
198             '404':
199                 description: User not found.
200                 content:

```

```

201     application/json:
202       schema:
203         $ref: '#/components/schemas/error'
204   put:
205     description: Update or create an model
206     parameters:
207       - in: path
208         name: model_id
209         description: Model id
210         schema:
211           type: string
212           required: true
213     requestBody:
214       required: true
215       content:
216         application/json:
217           schema:
218             $ref: '#/components/schemas/model_create'
219     responses:
220       '200':
221         description: Successfully updated the model.
222         content:
223           application/json:
224             schema:
225               $ref: '#/components/schemas/model'
226       '201':
227         description: Successfully created the model.
228         content:
229           application/json:
230             schema:
231               $ref: '#/components/schemas/model'
232       '401':
233         description: Invalid credentials.
234         content:
235           application/json:
236             schema:
237               $ref: '#/components/schemas/error'
238       '404':
239         description: User not found.
240         content:
241           application/json:
242             schema:
243               $ref: '#/components/schemas/error'
244   patch:
245     description: Partially update an model
246     parameters:
247       - in: path
248         name: model_id
249         description: Model id
250         schema:
251           type: string
252           required: true
253     requestBody:
254       required: true
255       content:
256         application/json:
257           schema:
258             $ref: '#/components/schemas/model_create'
259     responses:
260       '204':
261         description: Successfully updated the model.
262       '401':
263         description: Invalid credentials.
264         content:
265           application/json:
266             schema:
267               $ref: '#/components/schemas/error'
268       '404':
269         description: Model not found.

```

```

270         content:
271             application/json:
272                 schema:
273                     $ref: '#/components/schemas/error'
274 delete:
275     description: Delete a model
276     parameters:
277         - in: path
278           name: model_id
279           description: Model id
280           schema:
281               type: string
282               required: true
283     responses:
284         '204':
285             description: Successfully deleted the user.
286         '401':
287             description: Invalid credentials.
288             content:
289                 application/json:
290                     schema:
291                         $ref: '#/components/schemas/error'
292         '404':
293             description: Model not found.
294             content:
295                 application/json:
296                     schema:
297                         $ref: '#/components/schemas/error'
298 /models/{model_id}/content:
299 get:
300     description: Get the content information of a model
301     parameters:
302         - in: path
303           name: model_id
304           description: Model id
305           schema:
306               type: integer
307               required: true
308     responses:
309         '200':
310             description: Successfully returned the model's content information.
311             content:
312                 application/json:
313                     schema:
314                         type: object
315                         properties:
316                             files:
317                                 type: array
318                                 items:
319                                     type: object
320                                     properties:
321                                         id:
322                                             type: string
323                                             description: Model content file id
324         '401':
325             description: Invalid credentials.
326             content:
327                 application/json:
328                     schema:
329                         $ref: '#/components/schemas/error'
330         '404':
331             description: Model not found.
332             content:
333                 application/json:
334                     schema:
335                         $ref: '#/components/schemas/error'
336 post:
337     description: Insert a new file to a model
338     parameters:

```

```

339     - in: path
340       name: model_id
341       description: Model id
342       schema:
343         type: integer
344         required: true
345     requestBody:
346       required: true
347       content:
348         application/zip: {}
349     responses:
350       '204':
351         description: Successfully created the model file.
352       '400':
353         description: Invalid content.
354         content:
355           application/json:
356             schema:
357               $ref: '#/components/schemas/error'
358       '401':
359         description: Invalid credentials.
360         content:
361           application/json:
362             schema:
363               $ref: '#/components/schemas/error'
364       '404':
365         description: Model not found.
366         content:
367           application/json:
368             schema:
369               $ref: '#/components/schemas/error'
370 /models/{model_id}/content/{file_id}:
371 get:
372   description: Get the content of file of a model
373   parameters:
374     - in: path
375       name: model_id
376       description: Model id
377       schema:
378         type: integer
379         required: true
380     - in: path
381       name: file_id
382       description: Model content file id
383       schema:
384         type: string
385         required: true
386   responses:
387     '200':
388       description: Successfully returned the model's content.
389       content:
390         application/xml: {}
391     '401':
392       description: Invalid credentials.
393       content:
394         application/json:
395           schema:
396             $ref: '#/components/schemas/error'
397     '404':
398       description: Model or file not found.
399       content:
400         application/json:
401           schema:
402             $ref: '#/components/schemas/error'
403 put:
404   description: Update the content of file of a model
405   parameters:
406     - in: path
407       name: model_id

```

```

408     description: Model id
409     schema:
410       type: integer
411     required: true
412   - in: path
413     name: file_id
414     description: Model content file id
415     schema:
416       type: string
417     required: true
418   requestBody:
419     required: true
420     content:
421       application/xml: {}
422   responses:
423     '204':
424       description: Successfully updated the model content.
425     '400':
426       description: Invalid content.
427       content:
428         application/json:
429           schema:
430             $ref: '#/components/schemas/error'
431     '401':
432       description: Invalid credentials.
433       content:
434         application/json:
435           schema:
436             $ref: '#/components/schemas/error'
437     '404':
438       description: Model not found.
439       content:
440         application/json:
441           schema:
442             $ref: '#/components/schemas/error'
443 /projects:
444   get:
445     description: Returns all projects
446     parameters:
447       - in: query
448         name: public
449         description: Filter for public or private projects
450         schema:
451           type: string
452           enum:
453             - 'true'
454             - 'false'
455         required: false
456       - in: query
457         name: name
458         description: Filter projects using their names
459         schema:
460           type: string
461         required: false
462       - in: query
463         name: limit
464         description: Maximum amount of projects returned
465         schema:
466           type: string
467         required: false
468       - in: query
469         name: offset
470         description: Offset for pagination
471         schema:
472           type: string
473         required: false
474       - in: query
475         name: order_by_column
476         description: Columns used in sorting

```

```

477     schema:
478         type: string
479     required: false
480 - in: query
481     name: order_by_asc
482     description: Sort using ascending order if true
483     schema:
484         type: string
485         enum:
486             - 'true'
487             - 'false'
488     required: false
489 responses:
490     '200':
491         description: Successfully return a list of the projects.
492         content:
493             application/json:
494                 schema:
495                     type: object
496                     properties:
497                         total_quantity:
498                             type: integer
499                             description: Total quantity of models
500                         filtered_quantity:
501                             type: integer
502                             description: Quantity of models after filtering
503                         data:
504                             type: array
505                             items:
506                                 $ref: '#/components/schemas/project'
507     '401':
508         description: Invalid credentials.
509         content:
510             application/json:
511                 schema:
512                     $ref: '#/components/schemas/error'
513     security:
514     - bearerAuth: []
515 post:
516     description: Create a new project
517     requestBody:
518         required: true
519         content:
520             application/json:
521                 schema:
522                     $ref: '#/components/schemas/project_create'
523 responses:
524     '201':
525         description: Successfully created the project.
526         content:
527             application/json:
528                 schema:
529                     $ref: '#/components/schemas/project'
530     '400':
531         description: Invalid parameters.
532         content:
533             application/json:
534                 schema:
535                     $ref: '#/components/schemas/error'
536     '401':
537         description: Invalid credentials.
538         content:
539             application/json:
540                 schema:
541                     $ref: '#/components/schemas/error'
542     security:
543     - bearerAuth: []
544 /projects/{project_id}:
545 get:

```

```

546 description: Returns a single project
547 parameters:
548   - in: path
549     name: project_id
550     description: Project id
551     schema:
552       type: string
553       required: true
554 responses:
555   '200':
556     description: Successfully return the project.
557     content:
558       application/json:
559         schema:
560           $ref: '#/components/schemas/project'
561   '401':
562     description: Invalid credentials.
563     content:
564       application/json:
565         schema:
566           $ref: '#/components/schemas/error'
567   '404':
568     description: User not found.
569     content:
570       application/json:
571         schema:
572           $ref: '#/components/schemas/error'
573 security:
574   - bearerAuth: []
575 put:
576 description: Update or create an project
577 parameters:
578   - in: path
579     name: project_id
580     description: Project id
581     schema:
582       type: string
583       required: true
584 requestBody:
585   required: true
586   content:
587     application/json:
588       schema:
589         $ref: '#/components/schemas/project_create'
590 responses:
591   '200':
592     description: Successfully updated the project.
593     content:
594       application/json:
595         schema:
596           $ref: '#/components/schemas/project'
597   '201':
598     description: Successfully created the project.
599     content:
600       application/json:
601         schema:
602           $ref: '#/components/schemas/project'
603   '401':
604     description: Invalid credentials.
605     content:
606       application/json:
607         schema:
608           $ref: '#/components/schemas/error'
609   '404':
610     description: User not found.
611     content:
612       application/json:
613         schema:
614           $ref: '#/components/schemas/error'

```

```

615 patch:
616   description: Partially update an project
617   parameters:
618     - in: path
619       name: project_id
620       description: Project id
621       schema:
622         type: string
623         required: true
624   requestBody:
625     required: true
626     content:
627       application/json:
628         schema:
629           $ref: '#/components/schemas/project_create'
630   responses:
631     '204':
632       description: Successfully updated the project.
633     '401':
634       description: Invalid credentials.
635       content:
636         application/json:
637           schema:
638             $ref: '#/components/schemas/error'
639     '404':
640       description: Project not found.
641       content:
642         application/json:
643           schema:
644             $ref: '#/components/schemas/error'
645 delete:
646   description: Delete a project
647   parameters:
648     - in: path
649       name: project_id
650       description: Project id
651       schema:
652         type: string
653         required: true
654   responses:
655     '204':
656       description: Successfully deleted the group.
657     '401':
658       description: Invalid credentials.
659       content:
660         application/json:
661           schema:
662             $ref: '#/components/schemas/error'
663     '404':
664       description: Project not found.
665       content:
666         application/json:
667           schema:
668             $ref: '#/components/schemas/error'
669 /projects/{project_id}/content:
670 get:
671   description: Get the zipped content of a project
672   parameters:
673     - in: path
674       name: project_id
675       description: Project id
676       schema:
677         type: integer
678         required: true
679   responses:
680     '200':
681       description: Successfully returned the project's content.
682       content:
683         application/zip: {}

```

```

684     '401':
685         description: Invalid credentials.
686         content:
687             application/json:
688                 schema:
689                     $ref: '#/components/schemas/error'
690     '404':
691         description: Project not found.
692         content:
693             application/json:
694                 schema:
695                     $ref: '#/components/schemas/error'
696
697 /projects/{project_id}/models:
698     get:
699         description: Returns models from this project
700         parameters:
701             - in: path
702               name: project_id
703               description: Project id
704               schema:
705                 type: string
706                 required: true
707             - in: query
708               name: public
709               description: Filter for public or private models
710               schema:
711                 type: string
712                 enum:
713                     - 'true'
714                     - 'false'
715                 required: false
716             - in: query
717               name: name
718               description: Filter models using their names
719               schema:
720                 type: string
721                 required: false
722             - in: query
723               name: limit
724               description: Maximum amount of models returned
725               schema:
726                 type: string
727                 required: false
728             - in: query
729               name: offset
730               description: Offset for pagination
731               schema:
732                 type: string
733                 required: false
734             - in: query
735               name: order_by_column
736               description: Columns used in sorting
737               schema:
738                 type: string
739                 required: false
740             - in: query
741               name: order_by_asc
742               description: Sort using ascending order if true
743               schema:
744                 type: string
745                 enum:
746                     - 'true'
747                     - 'false'
748                 required: false
749         responses:
750             '200':
751                 description: Successfully return a list of the models.
752                 content:

```

```

753     application/json:
754       schema:
755         type: object
756         properties:
757           total_quantity:
758             type: integer
759             description: Total quantity of models
760           filtered_quantity:
761             type: integer
762             description: Quantity of models after filtering
763           data:
764             type: array
765             items:
766               $ref: '#/components/schemas/model'
767     '401':
768       description: Invalid credentials.
769       content:
770         application/json:
771           schema:
772             $ref: '#/components/schemas/error'
773     security:
774       - bearerAuth: []
775 /projects/{project_id}/sequences:
776 get:
777   description: Returns sequences from this project
778   parameters:
779     - in: path
780       name: project_id
781       description: Project id
782       schema:
783         type: string
784         required: true
785   responses:
786     '200':
787       description: Successfully return a list of the sequences.
788       content:
789         application/json:
790           schema:
791             type: object
792             properties:
793               total_quantity:
794                 type: integer
795                 description: Total quantity of models
796               filtered_quantity:
797                 type: integer
798                 description: Quantity of models after filtering
799               data:
800                 type: array
801                 items:
802                   $ref: '#/components/schemas/sequence'
803     '401':
804       description: Invalid credentials.
805       content:
806         application/json:
807           schema:
808             $ref: '#/components/schemas/error'
809     security:
810       - bearerAuth: []
811 /sequences:
812 get:
813   description: Returns all sequences
814   parameters:
815     - in: query
816       name: public
817       description: Filter for public or private projects
818       schema:
819         type: string
820         enum:
821           - 'true'

```

```

822         - 'false'
823         required: false
824     - in: query
825       name: limit
826       description: Maximum amount of sequences returned
827       schema:
828         type: string
829         required: false
830     - in: query
831       name: offset
832       description: Offset for pagination
833       schema:
834         type: string
835         required: false
836     - in: query
837       name: order_by_column
838       description: Columns used in sorting
839       schema:
840         type: string
841         required: false
842     - in: query
843       name: order_by_asc
844       description: Sort using ascending order if true
845       schema:
846         type: string
847         enum:
848           - 'true'
849           - 'false'
850         required: false
851 responses:
852   '200':
853     description: Successfully return a list of the sequences.
854     content:
855       application/json:
856         schema:
857           type: object
858           properties:
859             total_quantity:
860               type: integer
861               description: Total quantity of sequences
862             filtered_quantity:
863               type: integer
864               description: Quantity of sequences after filtering
865             data:
866               type: array
867               items:
868                 $ref: '#/components/schemas/sequence'
869   '401':
870     description: Invalid credentials.
871     content:
872       application/json:
873         schema:
874           $ref: '#/components/schemas/error'
875 security:
876   - bearerAuth: []
877 post:
878   description: Create a new sequence
879   requestBody:
880     required: true
881     content:
882       application/json:
883         schema:
884           $ref: '#/components/schemas/sequence_create'
885 responses:
886   '201':
887     description: Successfully created the sequence.
888     content:
889       application/json:
890         schema:

```

```

891         $ref: '#/components/schemas/sequence'
892     '400':
893         description: Invalid parameters.
894         content:
895             application/json:
896                 schema:
897                     $ref: '#/components/schemas/error'
898     '401':
899         description: Invalid credentials.
900         content:
901             application/json:
902                 schema:
903                     $ref: '#/components/schemas/error'
904     security:
905         - bearerAuth: []
906 /sequences/{sequence_id}:
907     get:
908         description: Returns a single sequence
909         parameters:
910             - in: path
911               name: sequence_id
912               description: Sequence id
913               schema:
914                 type: string
915                 required: true
916         responses:
917             '200':
918                 description: Successfully return the sequence.
919                 content:
920                     application/json:
921                         schema:
922                             $ref: '#/components/schemas/sequence'
923             '401':
924                 description: Invalid credentials.
925                 content:
926                     application/json:
927                         schema:
928                             $ref: '#/components/schemas/error'
929             '404':
930                 description: User not found.
931                 content:
932                     application/json:
933                         schema:
934                             $ref: '#/components/schemas/error'
935     put:
936         description: Update or create an sequence
937         parameters:
938             - in: path
939               name: sequence_id
940               description: Sequence id
941               schema:
942                 type: string
943                 required: true
944         requestBody:
945             required: true
946             content:
947                 application/json:
948                     schema:
949                         $ref: '#/components/schemas/sequence_create'
950         responses:
951             '200':
952                 description: Successfully updated the sequence.
953                 content:
954                     application/json:
955                         schema:
956                             $ref: '#/components/schemas/sequence'
957             '201':
958                 description: Successfully created the sequence.
959                 content:

```

```

960     application/json:
961       schema:
962         $ref: '#/components/schemas/sequence'
963     '401':
964       description: Invalid credentials.
965       content:
966         application/json:
967           schema:
968             $ref: '#/components/schemas/error'
969     '404':
970       description: User not found.
971       content:
972         application/json:
973           schema:
974             $ref: '#/components/schemas/error'
975 patch:
976   description: Partially update an sequence
977   parameters:
978     - in: path
979       name: sequence_id
980       description: Sequence id
981       schema:
982         type: string
983         required: true
984   requestBody:
985     required: true
986     content:
987       application/json:
988         schema:
989           $ref: '#/components/schemas/sequence_create'
990   responses:
991     '204':
992       description: Successfully updated the sequence.
993     '401':
994       description: Invalid credentials.
995       content:
996         application/json:
997           schema:
998             $ref: '#/components/schemas/error'
999     '404':
1000       description: Sequence not found.
1001       content:
1002         application/json:
1003           schema:
1004             $ref: '#/components/schemas/error'
1005 delete:
1006   description: Delete a sequence
1007   parameters:
1008     - in: path
1009       name: sequence_id
1010       description: Sequence id
1011       schema:
1012         type: string
1013         required: true
1014   responses:
1015     '204':
1016       description: Successfully deleted the user.
1017     '401':
1018       description: Invalid credentials.
1019       content:
1020         application/json:
1021           schema:
1022             $ref: '#/components/schemas/error'
1023     '404':
1024       description: Sequence not found.
1025       content:
1026         application/json:
1027           schema:
1028             $ref: '#/components/schemas/error'

```

```

1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041 /sequences/{sequence_id}/transformations:
1042   get:
1043     description: Returns all transformations from the sequence
1044     parameters:
1045       - in: path
1046         name: sequence_id
1047         description: Sequence id
1048         schema:
1049           type: string
1050           required: true
1051     responses:
1052       '200':
1053         description: Successfully return a list of the transformations.
1054         content:
1055           application/json:
1056             schema:
1057               type: object
1058               properties:
1059                 total_quantity:
1060                   type: integer
1061                   description: Total quantity of transformations
1062                 filtered_quantity:
1063                   type: integer
1064                   description: Quantity of transformations after filtering
1065                 data:
1066                   type: array
1067                   items:
1068                     type: object
1069                     properties:
1070                       id:
1071                         type: string
1072                         description: Transformation id
1073       '401':
1074         description: Invalid credentials.
1075         content:
1076           application/json:
1077             schema:
1078               $ref: '#/components/schemas/error'
1079     security:
1080       - bearerAuth: []
1081   put:
1082     description: Set the sequence transformations
1083     parameters:
1084       - in: path
1085         name: sequence_id
1086         description: Sequence id
1087         schema:
1088           type: string
1089           required: true
1090     requestBody:
1091       required: true
1092       content:
1093         application/json:
1094           schema:
1095             type: array
1096             items:
1097               type: object

```

```

1098         properties:
1099             id:
1100                 type: string
1101                 description: Transformation id
1102     responses:
1103         '204':
1104             description: Successfully set the transformations sequence.
1105         '400':
1106             description: Invalid parameters.
1107             content:
1108                 application/json:
1109                     schema:
1110                         $ref: '#/components/schemas/error'
1111         '401':
1112             description: Invalid credentials.
1113             content:
1114                 application/json:
1115                     schema:
1116                         $ref: '#/components/schemas/error'
1117     security:
1118         - bearerAuth: []
1119 /transformations:
1120     get:
1121         description: Returns all transformations
1122         responses:
1123             '200':
1124                 description: Successfully return a list of the transformations.
1125                 content:
1126                     application/json:
1127                         schema:
1128                             type: object
1129                             properties:
1130                                 total_quantity:
1131                                     type: integer
1132                                     description: Total quantity of transformations
1133                                 filtered_quantity:
1134                                     type: integer
1135                                     description: Quantity of transformations after filtering
1136                                 data:
1137                                     type: array
1138                                     items:
1139                                         type: object
1140                                         properties:
1141                                             id:
1142                                                 type: string
1143                                                 description: Transformations id
1144                                             from:
1145                                                 type: string
1146                                                 description: Original model kind
1147                                             to:
1148                                                 type: string
1149                                                 description: Model kind of the new model to be created
1150             '401':
1151                 description: Invalid credentials.
1152                 content:
1153                     application/json:
1154                         schema:
1155                             $ref: '#/components/schemas/error'
1156     security:
1157         - bearerAuth: []
1158 components:
1159     securitySchemes:
1160         bearerAuth:
1161             type: http
1162             scheme: bearer
1163             bearerFormat: JWT
1164     schemas:
1165         model:
1166             type: object

```

```

1167 properties:
1168   id:
1169     description: Model id
1170     type: string
1171   name:
1172     description: Model name
1173     type: string
1174   description:
1175     description: Model description
1176     type: string
1177   kind:
1178     description: Model content kind
1179     type: string
1180   empty:
1181     description: Returns if it's empty
1182     type: boolean
1183   tags:
1184     description: Model tags
1185     type: array
1186     items:
1187       type: object
1188       properties:
1189         name:
1190           type: string
1191   project:
1192     description: The models project id
1193     type: string
1194   owner:
1195     description: Owner of this model
1196     type: string
1197   creation:
1198     description: Creation date timestamp
1199     type: integer
1200 sequence:
1201   type: object
1202   properties:
1203     id:
1204       description: Sequence id
1205       type: string
1206     name:
1207       description: Sequence name
1208       type: string
1209     description:
1210       description: Sequence description
1211       type: string
1212     project:
1213       description: The sequences project id
1214       type: string
1215     owner:
1216       description: Owner of this sequence
1217       type: string
1218     creation:
1219       description: Creation date timestamp
1220       type: integer
1221   project:
1222     type: object
1223     properties:
1224       id:
1225         description: Project id
1226         type: string
1227       name:
1228         description: Project name
1229         type: string
1230     description:
1231       description: Project description
1232       type: string
1233     owner:
1234       description: Owner of this project
1235       type: string

```

```

1236     tags:
1237         description: Project tags
1238         type: array
1239         items:
1240             type: object
1241             properties:
1242                 name:
1243                     type: string
1244         creation:
1245             description: Creation date timestamp
1246             type: integer
1247     error:
1248         type: object
1249         properties:
1250             success:
1251                 type: boolean
1252                 description: Always false.
1253             error:
1254                 type: string
1255                 description: Error kind.
1256             error_message:
1257                 type: string
1258                 description: The error message.
1259     model_create:
1260         type: object
1261         properties:
1262             project:
1263                 description: The models project id
1264             name:
1265                 description: Model name
1266                 type: string
1267             description:
1268                 description: Model description
1269                 type: string
1270             kind:
1271                 description: Model content kind
1272                 type: string
1273             tags:
1274                 description: Model tags
1275                 type: array
1276                 items:
1277                     type: object
1278                     properties:
1279                         name:
1280                             type: string
1281         required:
1282             - name
1283             - kind
1284     sequence_create:
1285         type: object
1286         properties:
1287             project:
1288                 description: The sequences project id
1289             name:
1290                 description: Sequence name
1291                 type: string
1292             description:
1293                 description: Sequence description
1294                 type: string
1295         required:
1296             - name
1297
1298     project_create:
1299         type: object
1300         properties:
1301             name:
1302                 description: Project name
1303                 type: string
1304             description:

```

```
1305     description: Project description
1306     type: string
1307   tags:
1308     description: Project tags
1309     type: array
1310     items:
1311       type: object
1312       properties:
1313         name:
1314           type: string
1315   required:
1316     - name
```
