UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

MARIANA APARECIDA DA SILVA DOS SANTOS

PROCESSO PARA CRIAÇÃO DE SERVIÇOS WEB PARA O ENCAPSULAMENTO E ACESSO DE UMA API ESPECÍFICA

QUIXADÁ

2018

MARIANA APARECIDA DA SILVA DOS SANTOS

PROCESSO PARA CRIAÇÃO DE SERVIÇOS WEB PARA O ENCAPSULAMENTO E ACESSO DE UMA API ESPECÍFICA

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Orientador: Prof. Dr. Regis Pires Magalhães

Coorientador: Prof. Me. Luís Gustavo Coutinho

QUIXADÁ

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S236p Santos, Mariana Aparecida da Silva dos.
Processo para criação de serviços web para o encapsulamento e acesso de uma API específica / Mariana Aparecida da Silva dos Santos. – 2018.
37 f. : il.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2018.
Orientação: Prof. Dr. Regis Pires Magalhães.
Coorientação: Prof. Me. Luís Gustavo Coutinho do Rêgo.

1. Serviços da Web. 2. Web Server Gateway Interface. 3. Transferência de Estado Representacional. I. Título.

CDD 005

MARIANA APARECIDA DA SILVA DOS SANTOS

PROCESSO PARA CRIAÇÃO DE SERVIÇOS WEB PARA O ENCAPSULAMENTO E ACESSO DE UMA API ESPECÍFICA

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Aprovada em: ____/____/____

BANCA EXAMINADORA

Prof. Dr. Regis Pires Magalhães (Orientador)
Universidade Federal do Ceará – UFC

Prof. Me. Luís Gustavo Coutinho (Coorientador)
Instituto Federal do Ceará

Prof. Me. Aníbal Cavalcante de Oliveira
Universidade Federal do Ceará - UFC

Aos meus pais, José Ribamar e Franscisca Lúcia. A minha irmã Darliane Aparecida e companheiro Ederson Abreu.

AGRADECIMENTOS

À Universidade Federal do Ceará, pelo apoio financeiro com a manutenção da bolsas de auxílio.

Ao Prof. Dr. Regis Pires Magalhães, pela excelente orientação, paciência e suporte neste período que me acompanhou.

Aos professores participantes da banca examinadora Aníbal Cavalcante de Oliveira e Luís Gustavo Coutinho do Rêgo pelo tempo, pelas valiosas colaborações e sugestões.

Aos amigos e familiares que são base da minha construção profissional e pessoal.

“A mente que se abre a uma nova idéia jamais volta a seu tamanho original”

(Albert Einstein)

RESUMO

Características de execução dos softwares de forma distribuída, rápida, segura, portável e confiável são realidade no cenário atual do desenvolvimento de software. Integrar diferentes tecnologias pode vir a surgir como problema no processo de desenvolvimento abordando essas características, assim como realizar comunicação entre aplicações que utilizam linguagens diferentes mas que precisam trocar informações em alguma etapa do desenvolvimento. Pensando nestes problemas que podem vir a serem enfrentados e diante da necessidade de comunicar aplicações com tecnologias, linguagens diferentes este trabalho propõe um processo para criação de serviços web para acesso e uso genérico de uma *Application Programming Interface* (API) já existente. O processo foi validado através de um estudo de caso em que o framework *Graphast* foi encapsulado e disponibilizado serviços web.

Palavras-chave: Serviços da Web. Web Server Gateway Interface. Transferência de Estado Representacional.

ABSTRACT

Distributed, fast, secure, portable, and reliable software execution features are a reality in today's software development landscape. Integrating different technologies may come as a problem in the development process addressing these characteristics, as well as performing communication between applications that use different languages but that need to exchange information at some stage of development. Thinking about these problems that can be faced and the need to communicate applications with technologies, different languages this work proposes a process for creating web services for access and generic use of an existing *Application Programming Interface* (API). The process was validated through a case study in which the framework *Graphast* was encapsulated and made available web services.

Keywords: Web Services. Web Server Gateway Interface. Representational State Transfer.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
REST	*Representational State Transfer*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
URI	*Uniform Resource Identifier*
WSDL	*Web Services Description Language*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Os sistemas têm propensão a ficarem complexos ao longo do seu desenvolvimento, seja quando surgem novas funcionalidades e precisam ser acrescidas ao sistema, ou com a possibilidade de adicionar novas tecnologias. A dificuldade de integrar diferentes tecnologias é um obstáculo que pode surgir para o desenvolvimento de um sistema. Um outro obstáculo seria a necessidade de fazer a comunicação entre aplicações, no qual as linguagem de programação são diferentes. Problemas como estes estão dentro de um escopo que a solução utilizada para integração de sistemas e na comunicação entre aplicações diferentes é utilização de serviços *web*.

Uma aplicação pode ser criada utilizando uma determinada tecnologia, linguagem que melhor se aplica para resolução do problema mas quando surge a necessidade de se comunicar com outras linguagens, padronizar a comunicação é uma opção para que os sistemas se tornem interoperáveis. Uma abordagem que solucionaria as dificuldades citadas, seria uma aplicação ter seu conjunto de rotinas e padrões, estabelecidos e bem documentados, disponibilizados em forma de API para que outras aplicações possam utilizar suas funcionalidades sem precisar conhecer detalhes de sua implementação, tecnologias, linguagens utilizadas no desenvolvimento. Dessa forma permitirá a interoperabilidade entre sistemas. (SOMMERVILLE, 2011) define *Application Programming Interface* (API) como um conjunto de operações, que permite acesso a uma funcionalidade da aplicação. Isso significa que essa funcionalidade pode ser chamada diretamente por outros programas e não apenas acessada através da interface de usuário. A necessidade de comunicar API‘s foi a motivação deste trabalho em que buscou-se soluções diante do estudo feito.

O problema que este trabalho propõe resolver é a dificuldade de uso de uma API já existente criada em uma tecnologia diferente da que se pretende usar. Uma solução estudada para resolução do prolema é a publicação e documentação de uma API REST disponibilizada através de serviços *web*. Esta API encapsula e provê acesso aos serviços da API subjacente de forma geral e acessível a partir de qualquer linguagem ou tecnologia de desenvolvimento, basta seguir o protocolo de comunicação utilizado. Os serviços se comunicarão através de chamadas remotas, implementadas através da API, utilizando uma arquitetura REST e disponibilizando uma interface pública com serviços baseados no protocolo HTTP. Sendo o HTTP um protocolo padrão web ele permite uma melhor integração com várias tecnologias e linguagens, já que sua comunicação tem contratos claros e bem definidos.

1.1 Objetivo

O objetivo principal é a elaboração de um processo para criação de serviços *web* que permite o acesso e uso genérico de uma API já existente. O estudo do processo será validado através de um estudo de caso.

1.1.1 Objetivo Específicos

- Realizar estudo sobre arquitetura baseada em serviços web;
- Identificar e analisar o *framework* para composição dos serviços web;
- Gerar API para validação do processo;

O trabalho está dividido da seguinte forma, no Capítulo 2 serão estudados os conceitos mais relevantes que embasam este trabalho. No Capítulo 3 são mostrados os principais trabalhos relacionados. No Capítulo 4 é apresentado o processo proposto. No Capítulo 5 será apresentado o estudo de caso que visa encapsular a API do *framework Graphast*[1] e os resultados obtidos.

[1] https://github.com/InsightLab/graphast

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção dissertaremos sobre os principais conceitos para compreensão do processo proposto neste trabalho.

2.1 Arquitetura Orientada a Serviços - SOA

O foco do estudo aqui proposto é serviços *web*, mas não há como falar do mesmo sem contextualizar Arquitetura Orientada a Serviços (em inglês, *Service-Oriented Architecture* (SOA)). A arquitetura orientada a serviços é uma abordagem de design em que vários serviços colaboram para fornecer um conjunto final de recursos (NEWMAN, 2015).

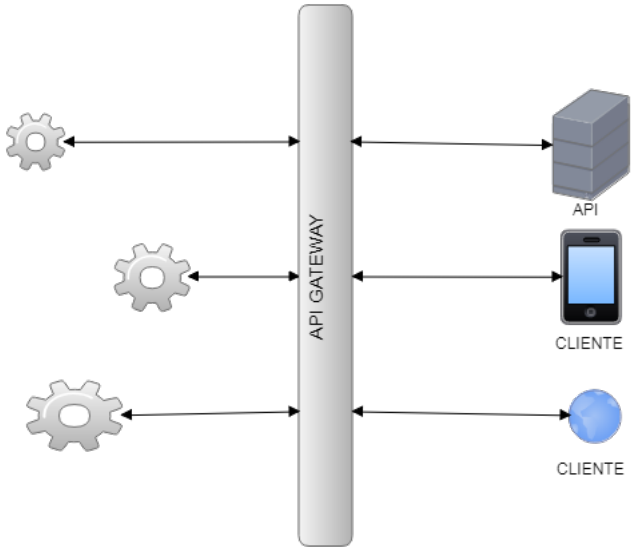
(SOMMERVILLE, 2011) define SOA como uma forma de desenvolvimento de sistemas distribuídos em que os componentes de sistema são serviços autônomos, executando em computadores geograficamente distribuídos. Utilizando protocolos padrão baseados em XML, SOAP e WSDL que foram projetados para oferecer suporte à comunicação de serviço e à troca de informações. Consequentemente, os serviços são plataforma e implementação independentes de linguagem. Os sistemas de software podem ser construídos pela composição de serviços locais e serviços externos de provedores diferentes, com interação perfeita entre os serviços no sistema. Sendo uma forma de criação de aplicações distribuídas usando *web services*.

A arquitetura orientada a serviços neste trabalho foi estudada para compreensão das características de serviços que a solução tem como base.

2.2 Serviços *WEB*

Serviços web são utilizados para transferir dados através de protocolos de comunicação para diferentes plataformas, independentemente das linguagens de programação utilizadas nessas plataformas (OPENSOFT, 2016). Seu funcionamento ocorre disponibilizando seus serviços deixando-os acessíveis ao público que será consumidor, uma aplicação solicita uma dessas operações. O serviço web efetua o processamento e envia os dados para a aplicação que requereu a operação.

Figura 1 – Lista dos serviços web criados



Fonte – Elaborado pelo autor

2.3 API Gateway

API Gateway é um serviço que resolve o problema de ter clientes de naturezas diferentes. É um ponto de entrada único que fornece acesso a muitas API‘s. Uma *API Gateway* fornece unicidades para: publicação de várias API‘s, cada uma dedicada a um conjunto diferente de clientes; e, atualizando o conjunto de APIs publicadas em tempo de execução (MONTESI; WEBER, 2016). *API Gateway* supri o problema de acesso direto a cada *endpoints* da arquitetura, pois é um serviço que os encapsula, tornando-se um único ponto de entrada entre a comunicação de um aplicativo cliente com os demais *endpoints* disponibilizados pela API de cada serviço *web*. Pode-se adicionar funcionalidade por conta da sua posição estratégica na arquitetura de micro serviços, como ter balanceamento de carga, descoberta de serviços, monitoramento e segurança (RIBAS, 2017)

A Figura 1 representa a utilização de um API Gateway na solução de serviços web.

2.4 REST

A sigla *Representational State Transfer* (REST), é um estilo de arquitetura que tem como princípio central a manipulação de recursos. A principal característica dos serviços que usam verdadeiramente este estilo é a exploração rica do protocolo HTTP (LUIZ, 2012). Entender um pouco do funcionamento do protocolo viabiliza a melhor utilização do estilo arquitetural que foi utilizado. Não está no escopo do trabalho fazer todo o estudo do protocolo HTTP, mas uma

condensação do funcionamento para um melhor entendimento do estudo.

O protocolo *Hypertext Transfer Protocol* (HTTP) é utilizado para comunicação na web, baseado em requisição e resposta utilizando a arquitetura cliente servidor, em que o cliente pode ser o navegador que solicita o acesso à determinados recursos. O servidor por sua vez recebe as informações e envia respostas de acordo com o propósito da solicitação. Como em todo protocolo é preciso usar as especificações definidas, terminologias, estrutura, para haver a comunicação, que pode ser encontradas em (NIELSEN et al., 1999).

O protocolo possui alguns aspectos como não possuir estado, funcionalidade de cacheamento da requisição, autenticação, sessões e troca de mensagens dentre outras. O entendimento das mensagens de requisição e respostas são as partes mais importantes no uso do protocolo dentro da solução proposta utilizando serviços *web*, pois toda a comunicação se dá através da troca de informações por via de mensagens e a compreensão da informação pela estrutura pré-definida no protocolo.

As mensagens são dividas em duas, requisição e resposta. As mensagens de requisições são compostas pelos seguintes elementos: verbo que define qual ação o cliente deseja fazer sobre o recurso, o caminho do recurso que será manipulado, versão do protocolo, cabeçalho que contém informações adicionais para os servidores, e podem conter também um corpo de dados que contém o recurso, este corpo haverá dependendo do verbo utilizado. Os métodos HTTP, também denominados verbos, são definidos para se fazer uma requisição e manter a comunicação entre os atores envolvidos, os métodos irão operar no recurso requisitado, pois identificam qual a ação que deve ser executada em um determinado recurso. O Quadro 1 tem uma breve descrição dos métodos mais utilizados.

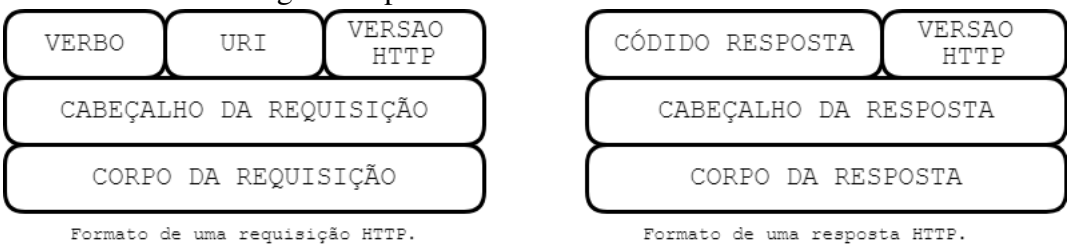
Quadro 1 – Métodos do Protocolo HTTP

MÉTODO	DESCRIÇÃO
GET	Solicita a representação de um recurso específico
POST	Utilizado para submeter uma entidade a um recurso específicos.
PUT	Substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.
PATCH	Utilizado para aplicar modificações parciais em um recurso.
DELETE	Remove um recurso específico.
OPTIONS	Usado para descrever as opções de comunicação com o recurso de destino.
HEAD	Solicita uma resposta de forma idêntica ao método GET, porém sem conter o corpo da resposta.

Fonte – https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods

As mensagens de respostas são compostas pelos elementos: versão do protocolo, um código de status que indica para quem fez à requisição informações sobre o sucesso ou não da

Figura 2 – Formato Mensagens Http



Fonte – Elaborado pelo autor

operação feita e o porque, opcionalmente um corpo de dados do recurso requisitado. A Figura 2 tem uma ilustração da estrutura e do formato das mensagens de requisição e respostas.

API REST tem em sua essência os recursos usufruindo do protocolo HTTP para manuseá-los na comunicação. Tem uma URI base para acessar os recursos, utiliza um tipo de mídia voltado para exibição dos mesmos, as mais utilizados são JSON e o XML, implementa os verbos HTTP padronizando assim a comunicação e usa *links* para referenciar o estado e outros tipos de recursos.

A utilização de uma API REST neste trabalho é para interligar sistemas essencialmente heterogêneos, pois permite realizar a troca de mensagens e informações mantendo a semântica dos dados entre os dispositivos envolvidos, e ainda garante a segurança, integridade e consistência nos dados. Cada serviço *web* poderá ser implementado com tecnologia diferente, ou ser consumido independente da tecnologia. A comunicação ainda vai acontecer pois o padrão REST permitirá a troca de mensagens.

3 TRABALHOS RELACIONADOS

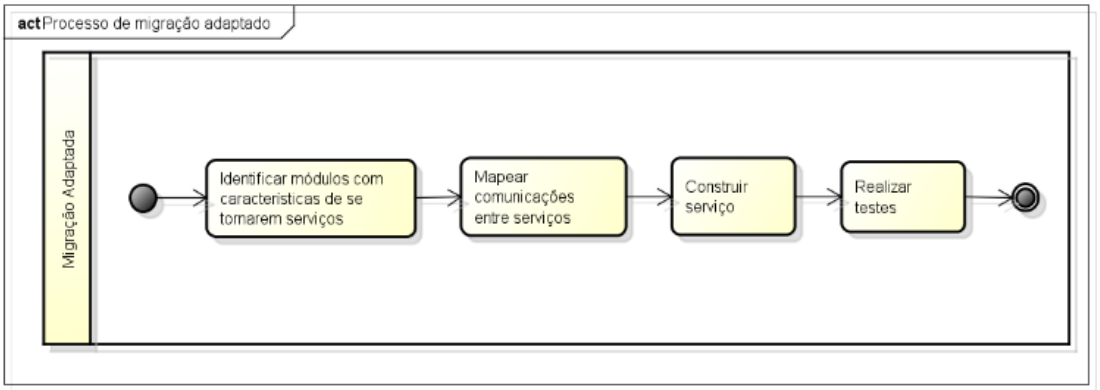
Neste capítulo serão abordados trabalhos, estudos que se relacionam com este, para maior embasamento do processo que será criado, norteando o trabalho.

3.1 Migração de uma arquitetura monolítica para uma arquitetura orientada à serviços

(SÁ, 2016) abordou a migração de um sistema monolítico para um sistema orientado a serviços, com o objetivo de definir um processo e ferramentas para que o *backend* das aplicações sigam o modelo SOA. Em síntese, (SÁ, 2016) dividiu sua pesquisa em três etapas onde a primeira foi feita o levantamento de tecnologias, definindo e validando para o desenvolvimento dos serviços. Na segunda etapa adaptou o processo antigo para o novo, migrando-o assim do sistema monolítico para a arquitetura orientada a serviço. Em sua terceira etapa do modelo proposto, avaliou a migração reiterando a melhoria dos atributos por ele elencados. A figura 3 ilustra o processo feito por (SÁ, 2016) em seu trabalho.

Ambos os trabalhos utilizaram-se do padrão de desenvolvimento REST. Se diferenciando quando em (SÁ, 2016) é criado um processo para seguir o modelo arquitetural SOA com finalidade de migrar um sistema para a arquitetura SOA. Neste trabalho o processo teve como finalidade de integrar uma API deixando-a acessível para uso a partir de outras tecnologias utilizando a solução de serviços web.

Figura 3 – Processo de Migração Adaptado (SÁ, 2016)



Fonte – http://www.repositorio.ufc.br/handle/riufc/25113

3.2 Um método para o desenvolvimento de software baseado em micro serviços

(ROSA, 2016) propôs e avaliou um método para construção de sistemas baseados na arquitetura de micro serviços. O método fornece um conjunto de passos para a construção de software utilizando micro serviços, tendo como objetivo minimizar a necessidade de reimplementação, caso ocorram mudanças. Minimizou por meio da limitação de serviços devidamente coesos com os mecanismos projetados de evolução das interfaces de serviço, envolvendo uma combinação de tecnologias em diferentes áreas da tecnologia da informação, além de suportar o quesito multiplataforma, propendendo à interoperabilidade. Realizou um estudo de caso desenvolvendo um software para suplementar um tratamento medicinal com aplicação de fototerapia.

Assemelha-se com este trabalho em que baseia seu estudo para aplicação dos mesmos para construção de sistemas e na sua utilização na implementação do estudo de caso. Quando em ambas as produções científicas foram padronizadas a comunicação utilizando o padrão REST, compreensão dos conceitos de serviços e arquitetura de software. No estudo de (ROSA, 2016) foi validado o uso de micro serviços propondo e demonstrando um método, este método serviu de base para que neste trabalho fosse definido o processo para criação de serviços web com objetivo de encapsular API.

3.3 Implementação de uma arquitetura baseada em micro serviços

(LIMA, 2015) identificou um ambiente em que aplicações corporativas necessitou da arquitetura de micro serviços para se adaptar à medida que migravam suas soluções para nuvem. Se fazendo importante um estudo das vantagens e desvantagens da arquitetura e propondo uma implementação de uma arquitetura baseada em micro serviços para aplicações corporativas. Com o objetivo de simplificar o ciclo de vida das aplicações, tirando proveito das vantagens que o modelo oferece como fácil manutenção, escalabilidade natural, tolerância à falhas dentre outros. Implementou uma solução com funcionalidades comuns em aplicações modernas, realizando estudos sobre arquiteturas já existentes, processo de negocio que foi decomposto em serviços e realizando um estudo sobre o desempenho da arquitetura proposta no trabalho.

Assemelha-se com o trabalho aqui proposto pois implementa uma solução baseada serviços, mas com objetivos distintos, enquanto em (LIMA, 2015) buscar realizar o estudo encontrando vantagens e desvantagens para o uso de micro serviços, este trabalho busca validar a

Tabela 1 – Tabela comparativa entre os trabalhos relacionados.

AUTOR	WEB SERVICES	ARQUITETURA	ESCOPO
(SÁ, 2016)	REST	Orientada à Serviços	Modelo para migração entre arquiteturas
(ROSA, 2016)	REST	Micro Serviços	Processo para desenvolvimento de projetos de software usando micro serviços
(LIMA, 2015)	REST	Micro Serviços	Uso de micro serviços em sistemas corporativos
TRABALHO PROPOSTO	REST	Orientada à Serviços	Processo para desenvolvimento de api gateway para acesso a micro serviços

Fonte – Produzido pelo autor

interoperabilidade permitida pelo uso da arquitetura orientada à serviços definindo um processo use tais características da mesma. Ambos os trabalhos concluíram que, decompor o negocio ou solução tecnológica para algum problema do mundo real, em serviços menores permite a heterogeneidade tecnológica. Pois pode-se aplicar a tecnologia adequada ao serviço que está sendo desenvolvido ou consumido, que ainda assim haverá a comunicação devido a utilização dos protocolos definidos para comunicação.

A importância do estudo com trabalhos correlatos é para a definição das etapas do processo, compreendendo o que a academia já teria previamente estudado e focando na problemática proposta. A Tabela 1 faz um comparativo entre os trabalhos referenciados nesta seção, com alguns atributos que estão destacados nas linhas.

4 PROCEDIMENTOS METODOLÓGICOS

Neste capítulo é apresentado o processo, definindo suas etapas com intuito de alcançar os objetivos propostos. Para elaboração do processo foi estudado e adaptado o método elaborado por (ROSA, 2016) e o processo feito por (SÁ, 2016). A seguir estão definidas cada etapa do processo e o fluxo pode ser visto na Figura 4.

4.1 Estudo do *framework* a ser encapsulado através de serviços Web

O estudo se faz necessário para conhecer o *framework* que será encapsulado, suas particularidades, funcionalidades e código fonte. Definindo posteriormente as tecnologias adequadas a necessidade dos serviços que serão disponibilizados, para o encapsulamento do *framework* escolhido.

A lista à seguir indica as atividades desta etapa:

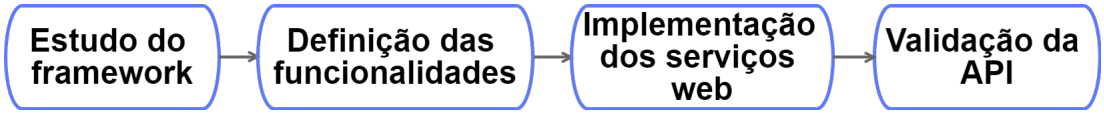
- Estudo do *framework* que pretende ser encapsulado;
- Definição das tecnologias que serão usadas durante a etapa de desenvolvimento dos serviços web;
- Definição da arquitetura de comunicação entre a API os consumidores e o framework.

4.2 Definir as funcionalidades que se tornarão serviços web

Identificar as características do *framework* do estudo de caso, que se tornarão serviços *web* para encontrar contexto limitado e garantir o desacoplamento entre os serviços. Que garanta também a comunicação, com intuito de facilitar a integração das aplicações, permitindo interoperabilidade entre as informações que circulam entre as aplicações.

Na modelagem dos serviços é preciso levar em consideração a representação dos recursos, que é uma característica básica para arquitetura REST. As características recomendadas, à partir deste estudo de acordo com (MOREIRA; BEDER, 2016) para identificação dos recursos são:

Figura 4 – Fluxo das Etapas do Processo



Fonte – Elaborado pelo autor

- Compreensibilidade: Tanto o servidor quanto o cliente devem ser capazes de entender e utilizar a forma de representação do recurso;
- Integralidade: O formato do dado a ser utilizado deve ser capaz de representar um recurso, mesmo nos casos em que um recurso possa ser composto por outros recursos;
- Assincronismo: Cada requisição pode ser tratada independente das outras requisições;

4.3 Implementação dos serviços web

Nesta etapa implementa-se os serviços web construindo a API REST. Está fará a comunicação entre o consumidor da API e o framework encapsulado. Serviços estes que foram descritos na etapa anterior, e que ao fim desta é esperado que esteja disponível para consumo e comunicação com o framework escolhido no estudo de caso.

As atividades desta etapa são implementar cada serviço web definido na seção anterior, utilizando as tecnologia escolhidas e seguindo o padrão REST.

4.4 Validação da API

Nesta etapa certifica-se o sucesso na comunicação entre API criada e o sistema que será encapsulado, com o objetivo de garantir o funcionamento da API. A validação é feita disponibilizando à API e executando seus serviços, testando-os via ferramentas que as consomem.

Atividade desta etapa é realizar testes simulando requisições HTTP para os serviços disponibilizados em *endpoint*'s da arquitetura REST. Validando o funcionamento da API diante das diversas situações possíveis feitas para cada requisição HTTP.

5 RESULTADOS

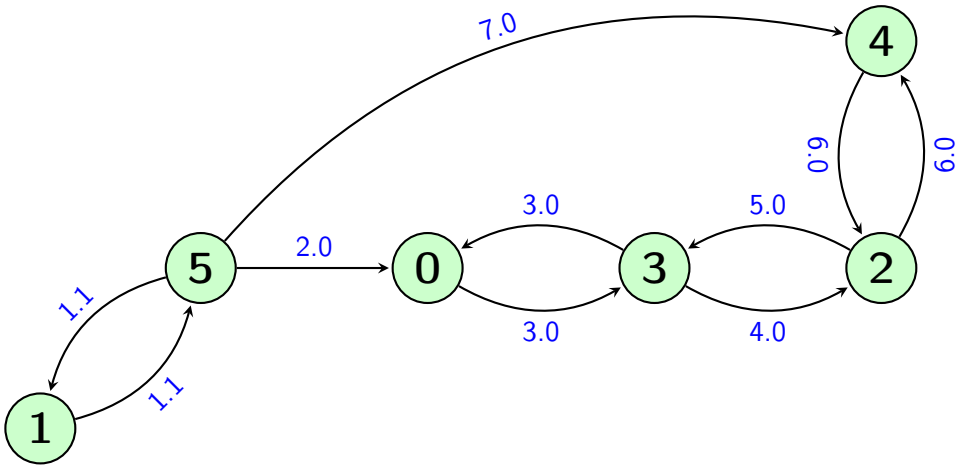
Neste capítulo estão relatados sobre os resultados obtidos durante o desenvolvimento do processo com base no estudo de caso feito.

5.1 Estudo de Caso

O estudo de caso foi feito criando uma API REST para o *framework Graphast*, deixando o mesmo facilmente integrado, aproveitando os benefícios que a arquitetura de serviços web oferece. O *Graphast* é um *framework* para processamento de consultas em redes dependentes de tempo em larga escala (RDT-LE), que provê uma infraestrutura básica para implementação de algoritmos para criação, armazenamento, indexação em redes dependentes de tempo (MAGALHÃES et al., 2015). A Figura 5 representa um grafo direcionado, em que serviços podem ser executados à partir de grafos criados tais como cálculo de custo entre dois pontos, menor caminho entre dois ou mais pontos.

Desenvolvedores que não conhecem a linguagem por trás do *framework* poderão usá-lo, pois com a API pronta os recursos do *Graphast* ficarão disponíveis como serviços *web*, permitindo que possam ser consumidos sem a necessidade de conhecer sobre a tecnologia subjacente utilizada.

Figura 5 – Representação de um grafo direcionado, onde os pesos das arestas indicam o custo para percorrer a aresta na direção indicada.



Fonte – https://github.com/InsightLab/graphast

5.2 Aplicação do Processo

Nesta seção estão exemplificadas as etapas do processo aplicadas ao estudo de caso em que as mesmas foram executadas.

5.2.1 Estudo do framework

Como dito na seção anterior, foi estudado o *framework Graphast* seu código fonte está disponível neste link[1]. Foi analisado e estudado pesquisas relativas ao *framework*, seu código-fonte e posteriormente escolhido suas funcionalidades para serem acessadas pela API desenvolvendo os serviços *web* e validando o processo.

As tecnologias escolhidas para o projeto deste estudo estão na lista à seguir e o código fonte pode ser encontrado através do link[2], que está num repositório público do git:

- O Spring é um framework open source para a plataforma Java, baseado nos padrões de projeto inversão de controle e injeção de dependência. Spring foi escolhido pela alta disponibilidade de documentação, suporte a criação de API REST. Spring Boot é um projeto do Spring que facilita o processo de configuração e publicação de aplicações.
- Postman[3] é uma aplicação que permite realizar requisições HTTP a partir de uma interface simples e intuitiva, facilitando o teste e depuração de serviços REST. Foi escolhida pois será utilizada como aplicação cliente que irá consumir a API .
- Swagger[4] é uma ferramenta utilizada para modelagem, documentação e geração de código para APIs do estilo REST. Foi escolhida pois tem-se a necessidade de ao consumir uma API existente, precisamos conhecer as funcionalidades disponíveis e detalhes de como invocá-las: recursos, URIs, métodos, Content-Types e outras informações. Swagger permite a documentação e publicação dos serviços disponíveis.
- Maven foi escolhido como ferramenta de automação e compilação do projeto.

Utilizado ainda nesta etapa o modelo arquitetural REST e JSON como formatação leve de troca de dados. O encapsulamento do *Graphast* ocorreu adicionando a dependência ao projeto criado, possibilitando a utilização de métodos e classes neste estudo.

[1] https://github.com/InsightLab/graphast
[2] https://github.com/lullabyh/graphast-api.git
[3] https://www.getpostman.com/
[4] https://swagger.io/

5.2.2 Definição dos serviços

Nesta etapa foram definidos os serviços web que serão criados, na próxima seção, à partir do *Graphast*. Foram escolhidas as estruturas primitivas do *framework* para serem acessadas pelos serviços criados, estas estruturas são grafo, nó e aresta. No *Graphsat* é possível manipular informações sobre estas estruturas, como criar um grafo, criar nó, adicionar um nó a um grafo. Além outras funcionalidades como consultas de distância entre nós percorrendo o grafo, cálculo de custos dentre outras.

A escolha foi feita à partir dos testes do projeto *Graphast* e por estarem de acordo com as características descritas no item 4.2. Os desenvolvedores elencaram funcionalidades básicas que foram testadas, facilitando a aplicação do processo e definição dos recursos. A lista à seguir descreve os serviços web criados.

- Criar grafo: gera um novo grafo, com o nome escolhido pelo consumidor da API;
- Recuperar grafo: Recupera o grafo desejado à partir do nome utilizado anteriormente, caso não tenho mensagem de retorno será enviada;
- Criar nó: gera um novo nó e adiciona o nó a algum grafo previamente criado;
- Recuperar nó: recupera um nó criado anteriormente;
- Criar Arestas: cria arestas com pontos definidos nos parâmetros da requisição e adiciona a algum grafo previamente criado;

5.2.3 Implementação dos serviços web

A implementação dos serviços ocorreu utilizando o *framework Spring* aproveitando ao máximo as facilidades e funcionalidades que o mesmo provê. Tem suporte para criação de API‘s REST, é facilmente configurável tornando o *deploy* da aplicação rápido e disponibilizando a API em tempo satisfatório para o trabalho realizado. Os serviços expõe informações via JSON fazendo requisições via protocolo HTTP.

O Código Fonte 1 tem exemplo de um serviço criado, listar grafos. O mesmo contém informações que identificam qual recurso que está sendo acessado via anotação *@RequestMapping*, qual método HTTP é necessário na requisição utilizando também anotação *@GetMapping* que o *framework* Spring provê. É possível ver a praticidade de criação de um serviço e publicação, além da facilidade de acesso via URI, o conhecimento do *framework* é fundamental para melhor utilização na API que será criada.

Código-fonte 1 – Exemplo do endpoint de listar grafos

```
@RestController
@RequestMapping("/graph-generator")
public class GraphResource {

  @GetMapping("/all")
  public ResponseEntity<?> getAllGraphs(){
    return ResponseEntity.ok(GraphRepo.getInstance().getAll
       ());
  }
...
}
```

Fonte – Elaborado pela autora

Para exemplificar na Figura 6 mostra a funcionalidade de listar grafos. Representando o serviço de listar quais grafos já estão cadastrados. A mesma ilustra como está o acesso ao serviços por meio de uma *Uniform Resource Identifier* (URI). Contém o método HTTP utilizado na requisição, formato de dados utilizado na comunicação JSON [5], que se encontra na resposta da requisição e o código de resposta da requisição que indica o *status* sobre a comunicação realizada. Ainda na figura é utilizado o *Postman*.

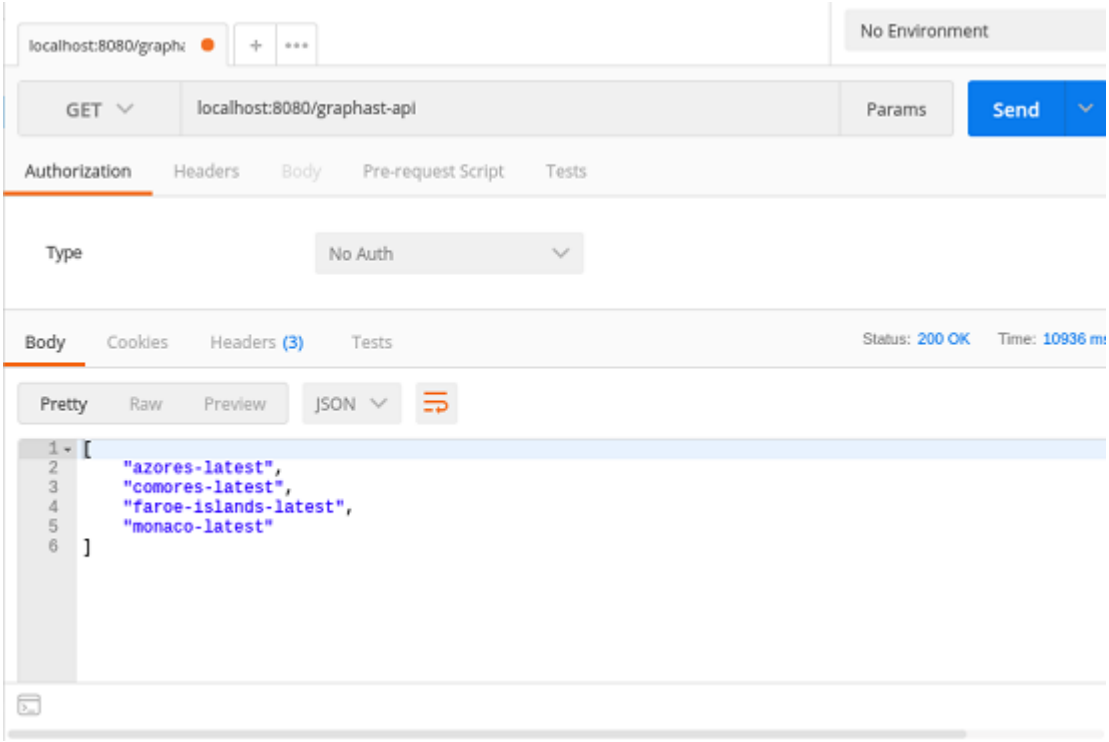
A Figura 7 contêm a lista dos serviços criados, com URI e métodos de acesso à cada serviço web. A imagem foi retirada da ferramenta *Swagger* que foi usada neste processo de desenvolvimento dos serviços web, a documentação da API.

5.2.4 Validação da API

A validação ocorreu testando os serviços web, realizando testes através da ferramenta já citada, o *Postman*. Os testes ocorreram fazendo requisições HTTP aos *endpoint*‘s criados, utilizando JSON e o protocolo HTTP para troca de informações entre as requisições. Os resultados esperados tinham como respostas os códigos de status de sucesso da classe 2xx. O cenário de teste criado foi também utilizando a ferramenta Swagger que informa o contrato da

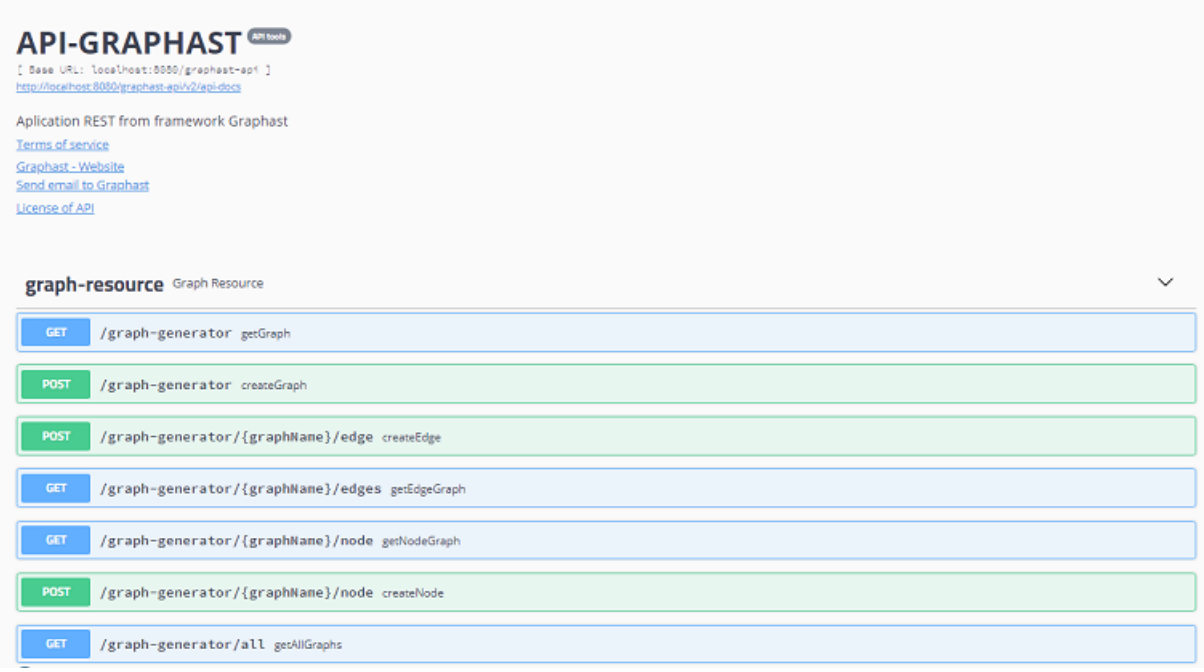
[5] http://www.json.org/

Figura 6 – Exemplo de serviço de listar grafos.



Fonte – Captura de tela de ferramenta Postman

Figura 7 – Lista dos serviços web criados



Fonte – Captura de tela da ferramenta Swagger

API e através dela é possível saber os recursos e corpos de mensagens esperados.

Foram definidos pontos importantes e comportamentos esperados para API criada. A lista à seguir são os pontos que foram definidos para serem validados nos testes executados da API:

- Validar se a resposta está de acordo com os testes contidos no *Graphast*;
- Validar se a estrutura do JSON está correta de acordo com o que é esperado na API;
- Validar se os códigos de respostas estão de acordo com as requisições feitas e de acordo com a especificação (NIELSEN et al., 1999);

O anexo A tem todos os *endpoint*‘s de acesso à API, com um exemplo de teste para cada serviço web criado. No mesmo contém os teste dos serviços, que garante que a API vai funcionar no "caminho feliz"de sua execução e seu fluxo. Validando assim os serviços web criados, certificando que estão de acordo com os testes do *framework Graphast* e os pontos de comportamentos esperados definidos e listados acima.

6 CONCLUSÕES E TRABALHOS FUTUROS

Serviços web podem ser independentes, sua independência traz consigo a possibilidade de mudanças, alterações e adaptações deixando o serviço web não intrusivo e permite que mudanças sejam realizadas gradativamente, adequando-as ao negocio. Permitir o acesso à *frameworks*, aplicações via serviços web no processo proposto provou-se nesse ponto eficiente, pois tornou acessível funcionalidades que podem ser disponibilizadas e acrescidas paulatinamente.

O estudo conseguiu atingir os objetivos traçados, durante o processo aplicado no estudo de caso, percebeu-se e confirmou a viabilidade do processo na integração de API's. Os trabalhos correlatos estudados, nortearam e embasaram o processo na definição de etapas, na escolha de tecnologias e embasamento teórico. As dificuldades encontradas no decorrer do trabalho foi basicamente a curva de aprendizado do *framework* escolhido, justificando novamente o processo criado e viabilizando o uso da API.

Um possível trabalho futuro seria utilizar o processo e adaptar os serviços, criados a partir do *framework Graphast*, para arquitetura de dos micro serviços. (FOWLER; LEWIS, 2014) define micro serviço como abordagem para desenvolver uma única aplicação como uma suíte de serviços, cada um rodando em seu próprio processo e se comunicando através de mecanismos leves, geralmente através de uma API HTTP. O trabalho futuro seria tornar os serviços criados em micro serviços independentes, desacoplados facilitando a escalabilidade e o deploy da aplicação.

REFERÊNCIAS

FOWLER, M.; LEWIS, J. **Microservices a definition of this new architectural term**. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 10 out. 2018.

LIMA, L. T. R. **Implementação de uma arquitetura baseada em microserviços**. Dissertação (Monografia) — Centro Universitário Eurípides de Marília, 2015.

LUIZ, B. **Web services REST**. 2012. Disponível em: <http://www.devmedia.com.br/artigo-java-magazine-56-web-services-rest-uma-abordagem-pratica/8447>. Acesso em: 10 mar 2018.

MAGALHÃES, R. P.; COUTINHO, G.; MACÊDO, J.; FERREIRA, C.; CRUZ, L.; NASCIMENTO, M. Graphast: An extensible framework for building applications on time-dependent networks. In: **Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems**. New York, NY, USA: ACM, 2015. (SIGSPATIAL '15), p. 93:1–93:4. ISBN 978-1-4503-3967-4. Disponível em: <http://doi.acm.org/10.1145/2820783.2820791>.

MONTESI, F.; WEBER, J. Circuit breakers, discovery, and api gateways in microservices. **arXiv preprint arXiv:1609.05830**, 2016.

MOREIRA, P. F. M.; BEDER, D. M. Desenvolvimento de aplicações e micro serviços: Um estudo de caso. **Revista TIS**, v. 4, n. 3, 2016.

NEWMAN, S. **Building microservices: designing fine-grained systems**. 1. ed. [S.l.]: "O'Reilly Media, Inc.", 2015.

NIELSEN, H. F.; MOGUL, J.; MASINTER, L. M.; FIELDING, R. T.; GETTYS, J.; BERNERS-LEE, T. **Hypertext Transfer Protocol – HTTP/1.1**. [S.l.], 1999. (Request for Comments, 2616). Disponível em: <https://rfc-editor.org/rfc/rfc2616.txt>.

OPENSOFT. **Web service: o que é, como funciona, para que serve?** 2016. Disponível em: <https://www.opensoft.pt/web-service/>. Acesso em: 28 jul. 2018.

RIBAS, R. H. **Engenharia de software no contexto de microservices**. Dissertação (Monografia) — Universidade do Vale do Rio dos Sinos, 2017.

ROSA, T. P. **Um método para o desenvolvimento de software baseado em microsserviços**. Dissertação (Monografia) — Universidade Federal do Ceará, Fortaleza, 2016.

SÁ, A. L. F. **Migração de uma arquitetura monolítica para uma arquitetura orientada a serviços**. Dissertação (Monografia) — Universidade Federal do Ceará, Fortaleza, 2016.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. [S.l.]: Pearson Education, 2011.

APÊNDICE A – MODELO DE ACESSO À API

```
URL DOCUMENTAÇÃO:
http://localhost:8080/graphast-api/swagger-ui.html

 - SERVIÇO PARA CRIAÇÃO DE UM GRAFO DISPONÍVEL EM:
URL: localhost:8080/graphast-api/graph-generator

 - MENSAGEM DE REQUISIÇÃO UTILIZANDO PROTOCOLO HTTP:
REQUEST
curl -X POST \
 http://localhost:8080/graphast-api/graph-generator \
  -H 'Content-Type: text/plain' \
  -H 'cache-control: no-cache' \
  -d graph01

 - MENSAGEM DE RESPOSTA ESPERADA UTILIZANDO PROTOCOLO HTTP:
RESPONSE
{
    "duplicatedNodesCounter": 0,
    "edges": [],
    "allComponentNames": null,
    "nodeIterator": [],
    "nodes": [],
    "edgeIterator": [],
    "numberOfNodes": 0,
    "numberOfEdges": 0,
    "allComponents": [],
    "removed": false
}
Status code: 200 ok
```

```
 - SERVIÇO PARA RECUPERAÇÃO DE UM GRAFO DISPONÍVEL EM:
URL: localhost:8080/graphast-api/graph-generator?graphName=graph01

- MENSAGEM DE REQUISIÇÃO UTILIZANDO PROTOCOLO HTTP:
REQUEST
curl -X GET \
'http://localhost:8080/graphast-api/graph-generator?graphName=graph01' \
  -H 'cache-control: no-cache'

- MENSAGEM DE RESPOSTA ESPERADA UTILIZANDO PROTOCOLO HTTP:
RESPONSE
{
    "duplicatedNodesCounter": 0,
    "edges": [],
    "allComponentNames": null,
    "nodeIterator": [],
    "nodes": [],
    "edgeIterator": [],
    "numberOfNodes": 0,
    "numberOfEdges": 0,
    "allComponents": [],
    "removed": false
}
Status Code: 200 Ok

 - SERVIÇO PARA CRIAÇÃO DE UM NÓ EM UM GRAFO DISPONÍVEL EM:
URL: localhost:8080/graphast-api/graph-generator/graph01/node

- MENSAGEM DE REQUISIÇÃO UTILIZANDO PROTOCOLO HTTP:
REQUEST
curl -X POST \
  http://localhost:8080/graphast-api/graph-generator/graph01/node \
```

```
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 65812721-d465-4c97-869e-36a846c60b5c' \
  -H 'cache-control: no-cache' \
  -d '{
  "id": 1.0,
  "lat": 30.0,
  "lng": 40.0
}'
```

- MENSAGEM DE RESPOSTA ESPERADA UTILIZANDO PROTOCOLO HTTP:

```
RESPONSE
Status Code: 200 OK
```

- SERVIÇO PARA RECUPERAÇÃO DE UM NÓ EM UM GRAFO DISPONÍVEL EM:

```
URL: localhost:8080/graphast-api/graph-generator/graph01/node
```

- MENSAGEM DE REQUISIÇÃO UTILIZANDO PROTOCOLO HTTP:

```
REQUEST
curl -X POST \
  http://localhost:8080/graphast-api/graph-generator/graph01/node \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 3d991757-fac0-43fe-a06f-58c305d320a2' \
  -H 'cache-control: no-cache' \
  -d '{
  "id": 2.0,
  "lat": 50.0,
  "lng": 70.0
}'
```

- MENSAGEM DE RESPOSTA ESPERADA UTILIZANDO PROTOCOLO HTTP:

```
RESPONSE
Status Code: 200 OK
```

- SERVIÇO PARA CRIAÇÃO DE UMA ARESTA EM UM GRAFO DISPONÍVEL EM:

URL: localhost:8080/graphast-api/graph-generator/graph01/edge

- MENSAGEM DE REQUISIÇÃO UTILIZANDO PROTOCOLO HTTP:

REQUEST

```
curl -X POST \
  http://localhost:8080/graphast-api/graph-generator/graph01/edge \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 53b33027-e04c-48d8-a513-6c32ec3dcce0' \
  -H 'cache-control: no-cache' \
  -d '{
  "bidirectional": true,
  "from": 1.0,
  "points": [
    {
      "lat": 50.0,
      "lng": 60.0
    }
  ],
  "to": 2.0,
  "weight": 5.0
}'
```

- MENSAGEM DE RESPOSTA ESPERADA UTILIZANDO PROTOCOLO HTTP:

RESPONSE

Status Code: 200 OK

- SERVIÇO PARA RECUPERAÇÃO DE UMA ARESTA EM UM GRAFO DISPONÍVEL EM:

URL: localhost:8080/graphast-api/graph-generator/graph01/edges

- MENSAGEM DE REQUISIÇÃO UTILIZANDO PROTOCOLO HTTP:

```
REQUEST
curl -X GET \
  http://localhost:8080/graphast-api/graph-generator/graph01/edges \
  -H 'Postman-Token: 21d29039-d570-44ff-92dd-443f729c2b45' \
  -H 'cache-control: no-cache'
```

- MENSAGEM DE RESPOSTA ESPERADA UTILIZANDO PROTOCOLO HTTP:

```
RESPONSE
[
    {
        "id": 0,
        "fromNodeId": 1,
        "toNodeId": 2,
        "weight": 1,
        "bidirectional": false,
        "allComponentClasses": [
            ""
        ],
        "removed": false
    }
]
Status Code: 200 OK
```

- SERVIÇO PARA RECUPERAÇÃO DE TODOS OS GRAFOS DISPONÍVEIS EM:

URL: localhost:8080/graphast-api/graph-generator/all

- MENSAGEM DE REQUISIÇÃO UTILIZANDO PROTOCOLO HTTP:

```
REQUEST
curl -X GET \
  http://localhost:8080/graphast-api/graph-generator/all \
  -H 'cache-control: no-cache'
```

- MENSAGEM DE RESPOSTA ESPERADA UTILIZANDO PROTOCOLO HTTP:

RESPONSE


```
{
    "graph01": {
        "duplicatedNodesCounter": 0,
        "edges": [],
        "allComponentNames": null,
        "nodeIterator": [
            {
                "id": 1,
                "allComponentClasses": [
                    ""
                ],
                "removed": false
            }
        ],
        "nodes": [
            {
                "id": 1,
                "allComponentClasses": [
                    ""
                ],
                "removed": false
            }
        ],
        "edgeIterator": [],
        "numberOfNodes": 1,
        "numberOfEdges": 0,
        "allComponents": [],
        "removed": false
    }
}
```