

# Sistema Gerenciador de Estágio do IFSP Hortolândia com arquitetura de microsserviços

<sup>1</sup> Lenilson T. Salvino, <sup>1</sup> Michele Cristiani Barion

<sup>1</sup> Campus Hortolândia - Instituto Federal de Educação, Ciência e Tecnologia de São Paulo  
(IFSP) 113183-250 - Hortolândia - SP - Brasil  
lenilsons@gmail.com; michele\_barion@hotmail.com

**Abstract.** *The purpose of this work was to develop a internship management system for the Hortolândia Federal Institute of São Paulo campus, using the microservices architecture. The project consists of seven microservices, a cache server, a distributed messaging processing platform, two NoSql database servers, three relational database servers, and a load balancer. The technologies used in development and the end result will be presented in this article. The main purpose was to acquire new knowledge and improve the technical knowledge related to the concepts and technologies that involve the microservices architecture.*

**Resumo.** *A finalidade deste trabalho foi desenvolver um sistema gerenciador de estágios para o Instituto Federal de São Paulo campus Hortolândia, utilizando a arquitetura de microsserviços. O projeto é composto por sete microsserviços, um servidor de cache, uma plataforma distribuída para processamento de mensageria, dois servidores de banco de dados NoSql, três servidores de banco de dados relacionais e um balanceador de carga. As tecnologias utilizadas no desenvolvimento e o resultado final serão apresentadas neste artigo. O intuito principal foi de adquirir novos conhecimentos e aprimorar os conhecimentos técnicos relacionados aos conceitos e tecnologias que envolvem a arquitetura de microsserviços.*

## 1. Introdução

Definir a arquitetura de uma aplicação não é uma tarefa simples e existem vários padrões que podem ser seguidos, sendo difícil escolher o que melhor se adequa ao propósito da aplicação (AMARAL, 2017).

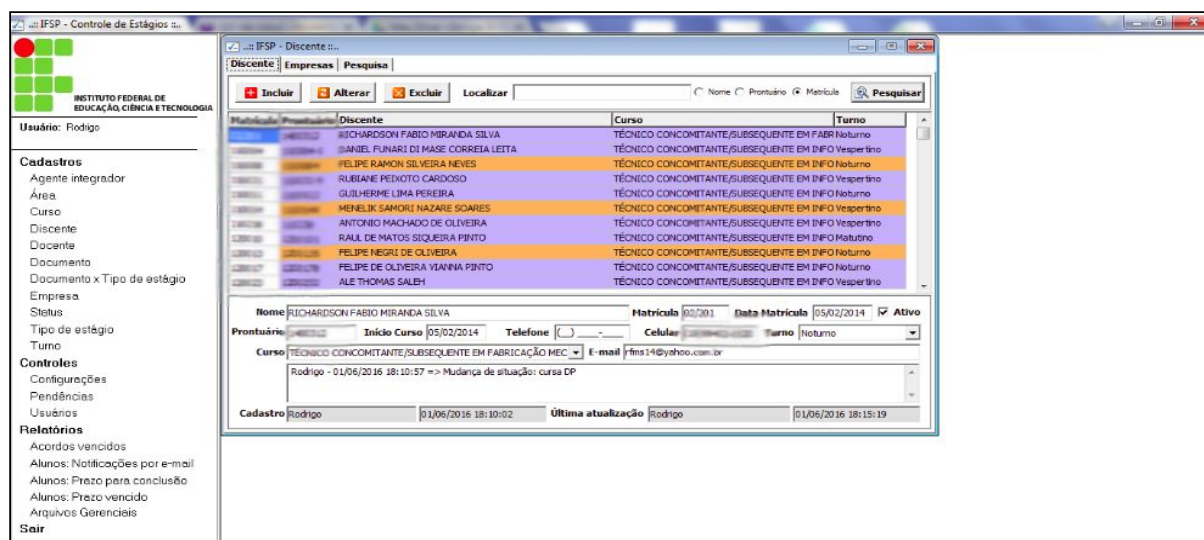
A arquitetura monolítica é um padrão amplamente usado para o desenvolvimento de aplicações corporativas. Esse padrão funciona razoavelmente bem para pequenas aplicações, pois o desenvolvimento, testes e implantação de pequenas aplicações monolíticas são relativamente simples (FOWLER; LEWIS, 2014).

Segundo Rosa (2016) dentre as seguintes dificuldades enfrentadas com o crescimento de aplicações monolíticas podemos destacar:

- A grande base de código monolítico torna o entendimento complexo;
- Dimensionamento da aplicação torna-se um desafio;
- Integração contínua e implantação tornam-se complexas e extensas;
- Sobrecarga da IDE em virtude da grande base de código.

Após levantamento realizado por demandas e melhorias de sistemas no IFSP Campus Hortolândia, foi verificado que o sistema de gerenciamento de estágios utilizado pela Coordenadoria de Estágio não estava atendendo as necessidades do setor por conta de alguns

problemas, tais como: monousuário, travamentos, perda de dados. A Figura 1 apresenta uma captura da tela do cadastro de alunos desse sistema.



**Figura 1. Tela referente ao cadastro de alunos**

**Fonte: Captura de tela do sistema de gerenciamento de estágio, realizada pelo autor**

O objetivo do presente trabalho é o desenvolvimento de um sistema de gerenciamento de estágios utilizando a arquitetura de microserviços, consistindo em um sistema para auxiliar os usuários da Coordenadoria de estágios do IFSP Hortolândia no processo de gerenciamento de contratos de estágios, relatórios de atividades e consulta de informações estatísticas relacionadas ao processo de estágios.

O intuito principal é a aquisição e o aprimoramento de conhecimentos técnicos relacionados aos conceitos e tecnologias que envolvem a arquitetura de microserviços.

A seção 1.1. apresenta dois trabalhos similares à proposta, destacando as características de cada um.

### 1.1. Trabalhos correlatos

O “Sistema de Gerenciamento de Estágios da POLI-UPE” foi desenvolvido por Barros (2008) com a finalidade de facilitar o gerenciamento das tarefas realizadas no decorrer do processo de estágio supervisionado dos alunos da Escola Politécnica de Pernambuco, pelo fato de que as atividades eram realizadas manualmente. O desenvolvimento desse sistema ofereceu uma série de melhorias no processo de estágio, dentre elas a diminuição da carga de trabalho do coordenador de estágio supervisionado, maior participação dos alunos no processo, facilidade na comunicação entre as pessoas envolvidas, dentre outras.

O projeto “CEFET-CONNECTIONS (aplicação da arquitetura de microserviços em uma rede social)” foi desenvolvido por Viana (2017), o mesmo teve como objetivo implementar uma arquitetura com funcionalidades comuns em aplicações baseadas em microserviços a fim de explorar as vantagens, desvantagens e dificuldades que o modelo arquitetural oferece. Para esse estudo de caso foi desenvolvido uma rede social utilizada pelos alunos do CEFET-RJ. O projeto é composto por oito microserviços, dois banco de dados não relacionais, isolamento de serviços e processamento de mensageria.

Além da introdução, este trabalho está organizado em 7 seções, sendo na ordem: fundamentação teórica; metodologia adotada para desenvolvimento da proposta; implementação do trabalho; resultados obtidos; conclusões e possível continuidade do

trabalho, finalizando com a apresentação das referências que foram fundamentais no decorrer de todo o desenvolvimento.

## 2. Fundamentação teórica

Nesta seção são apresentadas as referências teóricas que serviram de base para a realização do trabalho.

### 2.1. Modelo Arquitetural *REST*

*REST* é um termo definido por Roy Fielding (2000) em seu mestrado no qual ele descreve um estilo de arquitetura de software sobre um sistema operado em rede. *REST* é um acrônimo para "Transferência de Estado Representacional" (*Representational State Transfer*).

O *REST* trabalha em conjunto com o protocolo HTTP aproveitando os recursos do mesmo, tais como seus métodos: *GET*, *POST*, *PUT*, *DELETE* e *PATCH*. A Figura 2 apresenta a definição de cada método:

MÉTODO	EXPLICAÇÃO
GET	Buscar recursos
POST	Criar um novo recurso
PUT	Atualizar um recurso existente
PATCH	Atualizar parcialmente um recurso existente
DELETE	Remover um recurso

**Figura 2. Métodos HTTP utilizados pelo modelo arquitetural REST**

Fonte: elaborada pelos autores

### 2.2. Arquitetura Monolítica

Richardson (2014) menciona que a arquitetura monolítica pode ser então definida como aquela que mantém todos os componentes de uma aplicação em uma única unidade de implantação. Esse modelo de arquitetura é representado na Figura 3:



**Figura 3. Representação de uma aplicação baseada na arquitetura monolítica**

Fonte: elaborada pelos autores

### 2.3. Arquitetura de Microserviços

O termo microserviços surgiu nos últimos anos para descrever uma maneira peculiar de desenvolver aplicações de software independentemente implementáveis através de diferentes linguagens de programação, possibilidade de evolução, gerenciamento de equipes distintas, dentre outras vantagens.

Fowler e Lewis (2014) definem que o estilo arquitetural de microserviços aborda como desenvolver uma única aplicação como um conjunto de pequenos serviços, cada um executando em seu próprio processo e utilizando mecanismos leves de comunicação.

Newman (2015) considera microserviços como serviços pequenos e autônomos que funcionam em conjunto.

Segundo Martin (2017), cada serviço deve ser pequeno em razão do princípio de responsabilidade única, mantendo agrupado o conjunto de funcionalidades que sofrem alterações em conjunto e separando as demais funcionalidades independentes.

Thones (2015) define microserviços como uma aplicação que pode ser implantada, dimensionada e testada de maneira independente.

### 2.4. Histórias de Usuário

Histórias de usuário é uma maneira de expressar os requisitos de um *software* em *Extreme Programming*, que é uma metodologia ágil para o desenvolvimento de *software* (SOMMERVILLE, 2011). Cohn (2004) afirma que histórias de usuário são descrições curtas de uma funcionalidade vista pela perspectiva de um usuário do sistema.

A Figura 4 apresenta um exemplo de história de usuário:

	<b>Título: Pagamento com Cartão de Crédito</b>	<b>Prioridade: ?</b>
●	<b>Quem ?</b> como um cliente	
●	<b>O que ?</b> preciso de uma interface de pagamento por cartão de crédito que seja intuitiva e fácil de usar.	
●	<b>Por que ?</b> Com objetivo de facilitar os pagamentos.	
		<b>Pontos: 8</b>

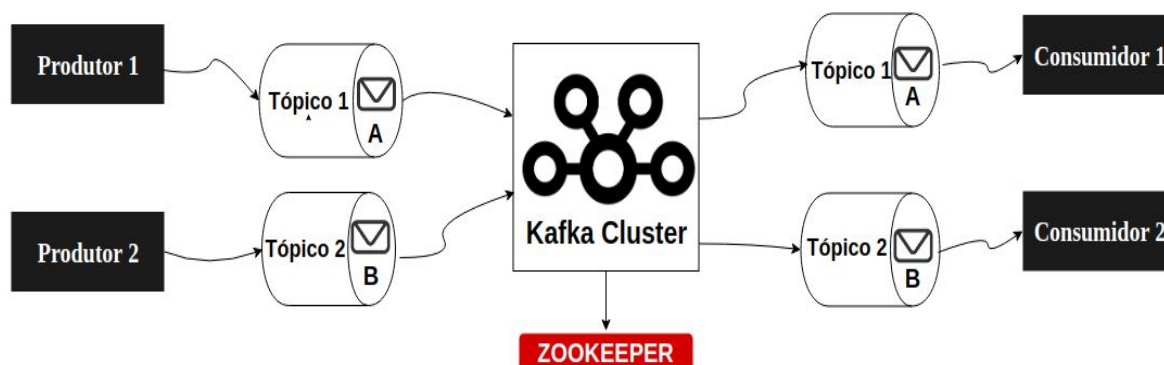
Figura 4. Exemplo de uma história de usuário (Levison 2017)

### 2.5. Apache Kafka

É definido como um sistema de processamento de dados em tempo real, o mesmo é considerado como um sistema de mensageria que atua como um mediador para orquestrar as mensagens processadas.

O propósito do Kafka é fornecer uma plataforma de sistema unificada de alta performance, baixa latência e alta disponibilidade para processamento da dados em tempo real (APACHE KAFKA, 2018).

A Figura 5 ilustra o funcionamento do Kafka com base num cenário típico de atuação:



**Figura 5. Funcionamento do Apache Kafka**  
**Fonte: elaborada pelos autores**

Na ilustração acima, a mensagem A é transmitida pelo Produtor 1 para o Tópico 1, e a mensagem B é transmitida pelo Produtor 2 no Tópico 2. O Consumidor 1 por estar inscrito no Tópico 1 recebe a Mensagem A e o Consumidor 2 por estar inscrito no Tópico 2 recebe a mensagem B. Como intermediador o *ZooKeeper* é um serviço de coordenação de informações para auxiliar o gerenciamento de distribuição de mensagens.

Segundo publicação no site da *RedHat* o *Apache Kafka* é incorporado a pipelines de transmissão que compartilham dados entre sistemas e aplicações que consomem esses dados, sendo compatível com vários casos de uso, em que alta produtividade e escalabilidade são fatores essenciais.

### 3. Metodologia

Os requisitos foram coletados através de entrevistas com os servidores responsáveis pela Coordenadoria de Estágio do IFSP Hortolândia. Após o levantamento de requisitos, o próximo passo foi idealizar e planejar o desenvolvimento do sistema. Na fase de idealização do sistema foram elaboradas algumas representações gráficas, tais como: representação de módulos, componentização de serviços e visão arquitetural. Na fase de planejamento, ocorreu a descrição de histórias de usuários com objetivo entender as perspectivas dos usuários do sistema e também a definição das tecnologias a serem utilizadas.

#### 3.1. Materiais utilizados

A figura 6 apresenta a listagem de tecnologias que foram utilizadas no desenvolvimento do projeto juntamente com um breve descrição:



**Figura 6. Listagem de tecnologias utilizadas no desenvolvimento do projeto**  
**Fonte: elaborada pelos autores**

A descrição da utilização de ferramentas e tecnologias são apresentadas na Tabela 1.

**Tabela 1. Descrição da utilização de ferramentas e tecnologias**

Ferramenta/Tecnologia	Versão	Utilização
Docker	18.09.1	Implantação e execução dos componentes e aplicações em ambientes isolados.
Portainer	1.22.0	Gerenciamento e monitoramento dos contêineres.
Apache Kafka	5.3.0	Plataforma de processamento mensageria responsável por realizar a distribuição de mensagens entre os microserviços: MS-Contratos, MS-Relatórios, MS-Notificador e <i>MS-Analytics</i> .
HAProxy	1.6.7	Balanceador de carga e servidor proxy.
Prometheus	2.11.1	Monitoramento e coleta de métricas das aplicações.
Grafana	6.2.5	Análise das métricas coletadas dos microserviços através dos recursos fornecidos pelo <i>Prometheus</i> .
Postman	7.3.4	Testar os serviços <i>RESTful</i> por meio do envio de requisições <i>HTTP</i> .
Redis	5.0.3	Solução de banco de dados em memória para armazenar as informações consumidas através do <i>Apache Kafka</i> pelo microserviço <i>MS-Analytics</i> .



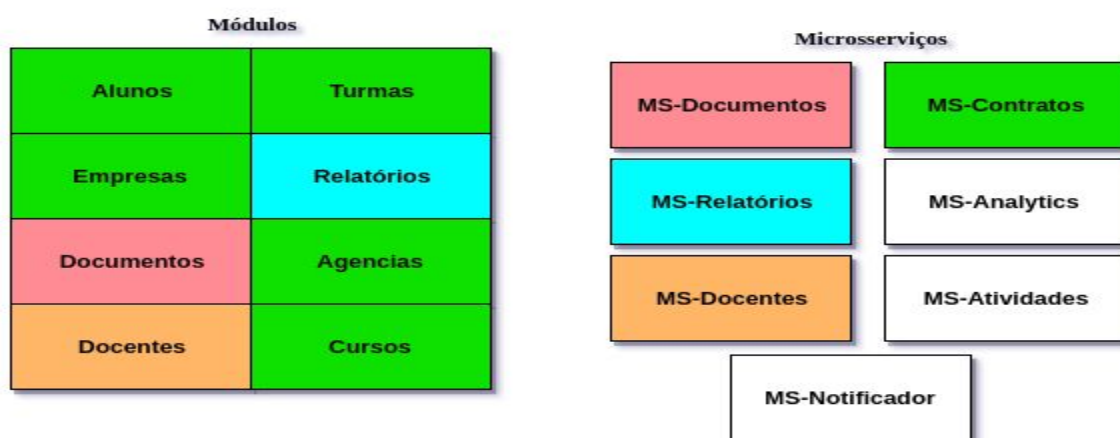
MySQL	5.7	Solução de banco de dados relacional dos microsserviços: MS-Contratos e MS-Docentes.
PostgreSQL	9.4	Solução de banco de dados relacional do microsserviço MS-Documentos.
MongoDB	3.6.4	Solução de banco de dados não relacional dos microsserviços: MS-Relatórios e MS-Atividades.

## 4. Desenvolvimento

O primeiro passo para o desenvolvimento foi entender o funcionamento do sistema monolítico, dado o contexto geral. O desenvolvimento no projeto se baseou nesse sistema levando em consideração suas funcionalidades.

### 4.1. Separação dos módulos em microsserviços

O sistema proposto foi dividido em sete microsserviços, a Figura 7 exibe uma representação gráfica de como foi realizada a separação dos módulos do sistema monolítico em microsserviços:



**Figura 7. Representação da divisão de módulos**

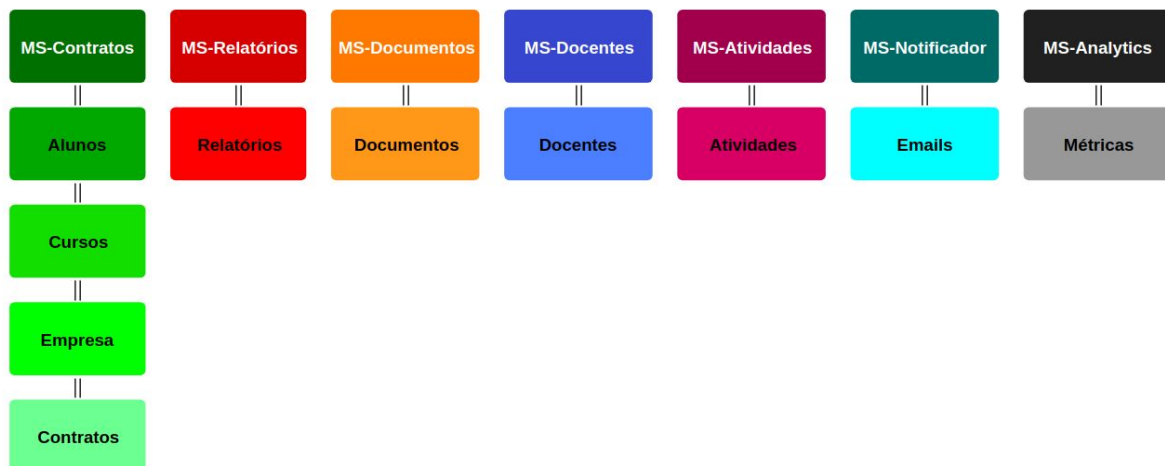
**Fonte: elaborada pelos autores**

Como medida de padronização para facilitar a leitura adiante, cada microsserviço será representado com o prefixo MS representando o termo microsserviço neste artigo.

Os módulos representados pela cor verde foram agrupados como componentes (APIs REST) do microsserviço MS-Contratos, pois a principal responsabilidade em comum desses módulos está ligada diretamente na composição de informações pertinentes ao contrato de estágio. Nos demais módulos, ocorreu o processo de isolamento conforme representação de cores distintas. Os módulos representados pela cor branca (MS-Analytics, MS-Notificador e MS-Atividades) surgiram por meio de demandas de novas funcionalidades.

## 4.2. Componentização

A Figura 8 apresenta uma representação gráfica da componentização do projeto dividida por microsserviço:



**Figura 8. Representação da componentização de serviços**  
Fonte: elaborada pelos autores

## 4.3. Estruturação

Esta subseção apresenta uma breve descrição do objetivo principal de cada microsserviço e também a composição de APIS REST pertinentes a cada um deles.

A Tabela 2 apresenta a estruturação do MS-Contrato.

**Tabela 2. Estruturação do MS-Contratos**

MS-Contratos		
<i>Base URL: http://localhost:8080/ms-contratos/v1</i>		
Responsável por gerenciar os processos relacionados aos contratos de estágio, tais processos podem ser definidos como: criação, atualização, exclusão e pesquisa de contratos.		
API DE ALUNOS		
Método	Recurso	Operação
GET	/alunos	Retorna todos os alunos cadastrados.
GET	/alunos/{id}	Busca um aluno existente com base no identificador.
GET	/alunos/{prontuario}	Busca um aluno existente com base no prontuário.
GET	/alunos/{nome}	Busca um aluno existente com base no nome.
POST	/alunos	Cria um novo aluno.



PUT	/alunos/{id}	Atualiza um aluno existente com base no identificador.
DELETE	/alunos{id}	Remove um aluno existente com base no identificador.
<b>API DE CURSOS</b>		
<b>Método</b>	<b>Recurso</b>	<b>Operação</b>
GET	/cursos	Retorna todos os cursos cadastrados.
POST	/cursos	Cria um novo curso.
PUT	/cursos/{id}	Busca um curso existente com base no identificador.
DELETE	/cursos{id}	
<b>API DE EMPRESAS</b>		
<b>Método</b>	<b>Recurso</b>	<b>Operação</b>
GET	/empresas	Retorna todas as empresas cadastradas.
GET	/empresas/{id}	Busca uma empresa existente com base no identificador.
GET	/empresas/{codigo}	Busca uma empresa existente com base no código.
POST	/empresas	Cria um nova empresa.
PUT	/empresas/{id}	Atualiza uma empresa existente com base no identificador.
DELETE	/empresas/{id}	Remove uma empresa existente com base no identificador.
<b>API DE CONTRATOS</b>		
<b>Método</b>	<b>Recurso</b>	<b>Operação</b>
GET	/contratos	Retorna todos os contratos cadastrados.
GET	/contratos/metricas	<p>Retorna as seguintes métricas:</p> <ul style="list-style-type: none"> <li>● Total de contratos;</li> <li>● Total de alunos;</li> <li>● Total de empresas;</li> <li>● Total de cursos;</li> <li>● Total de contratos de estágio por empresa;</li> <li>● Total de contratos de estágio por curso.</li> </ul> <p>Todas essas informações estão relacionadas no processo de gerenciamento de contratos de estágios.</p>
POST	/contratos	Cria um novo contrato.

PUT	/contratos/{id}	Atualiza um contrato existente com base no identificador.
DELETE	/contratos/{id}	Remove um contrato existente com base no identificador.

A Tabela 3 apresenta a estruturação do MS-Relatórios.

**Tabela 3. Estruturação do MS-Relatórios**

MS-Relatórios		
<b>Base URL:</b> <a href="http://localhost:8083/ms-relatorios/v1">http://localhost:8083/ms-relatorios/v1</a>		
Responsável por gerenciar relatórios mensais, o mesmo possui dois processos configurados mediante agendamento de execução ( <i>crontab</i> ) interligado ao framework <i>Spring</i> . O primeiro processo é responsável criar automaticamente relatórios mensais a cada mês, por padrão a execução desse processo está atualmente configurada para ser executada no final de cada mês. O segundo processo é responsável por identificar relatórios não entregues conforme verificação da data atual com a data da entrega, caso encontrado algum relatório, o mesmo encaminha o relatório para o MS-Notificador.		
API DE RELATÓRIOS		
Método	Recurso	Operação
GET	/relatorios	Retorna todos os relatórios cadastrados.
PUT	/relatorios/{id}	Atualiza um relatório existente com base no identificador.
DELETE	/relatorios/{id}	Remove um relatório existente com base no identificador.

A Tabela 4 apresenta a estruturação do MS-Documents.

**Tabela 4. Estruturação do MS-Documents**

MS-Documents		
<b>Base URL:</b> <a href="http://localhost:8085/ms-documentos/v1">http://localhost:8085/ms-documentos/v1</a>		
Responsável por manter e processar informações relacionadas aos documentos necessários para o processo de contratação/encerramento de contratos de estágio.		
API DE DOCUMENTOS		
GET	/documentos	Retorna todos os documentos cadastrados.
POST	/documentos	Cria um novo documento.

PUT	/documentos/{id}	Atualiza um documento existente com base no identificador.
DELETE	/documentos/{id}	Remove um documento existente com base no identificador.

A Tabela 5 apresenta a estruturação do MS-Docentes.

**Tabela 5. Estruturação do MS-Docentes**

MS-Docentes		
<b>Base URL:</b> http://localhost:8086/ms-docentes/v1		
Responsável por manter e processar informações relacionadas aos docentes.		
API DE DOCUMENTOS		
GET	/docentes	Retorna todos os docentes cadastrados.
POST	/docentes	Cria um novo docente.
PUT	/docentes/{id}	Atualiza um docente existente com base no identificador.
DELETE	/docentes/{id}	Remove um docente existente com base no identificador.

A Tabela 6 apresenta a estruturação do MS-Atividades.

**Tabela 6. Estruturação do MS-Atividades**

MS-Atividades		
<b>Base URL:</b> http://localhost:8081/ms-atividades/v1		
Responsável por manter e processar informações relacionadas às atividades de estágio.		
API DE RELATÓRIO DE ATIVIDADES		
GET	/atividades	Retorna todas as atividades cadastradas.
GET	/atividades/agrupadas	Retorna agrupamento de atividades.
POST	/atividades	Cria uma nova atividade.
PUT	/atividades/{id}	Atualiza uma atividade existente com base no identificador.
DELETE	/atividades/{id}	Remove uma atividade existente com base no identificador.

A Tabela 7 apresenta a definição do MS-Notificador.

**Tabela 7. Definição do MS-Notificador**

MS-Notificador
<b>Base URL:</b> <a href="http://localhost:8084/ms-notificador/v1">http://localhost:8084/ms-notificador/v1</a>
Responsável por notificar os alunos que ainda não efetuaram a entrega do relatório mensal, esse processo de notificação é estabelecido com base nas informações coletadas dos relatórios enviados pelo MS-Relatórios.

A Tabela 8 apresenta a estruturação do MS-Analytics.

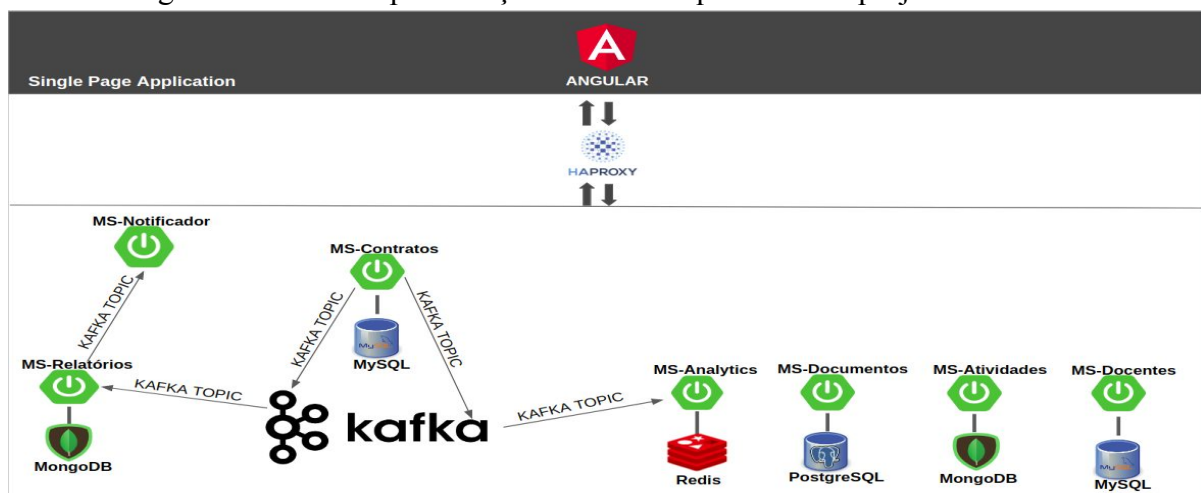
**Tabela 8. Definição do MS-Analytics**

MS-Analytics		
<b>Base URL:</b> <a href="http://localhost:8082/ms-analytics/v1">http://localhost:8082/ms-analytics/v1</a>		
Responsável por centralizar métricas fornecidas pelo MS-Contratos e MS-Relatórios.		
API DE MÉTRICAS		
Método	Recurso	Operação
GET	/metricas/{id}	Retorna todas as métricas salvas no <i>Redis</i> .

#### 4.4. Visão Arquitetural

Com base na visão arquitetural deste projeto, é possível evidenciar alguns princípios fundamentais fornecidos pela arquitetura de microsserviços, tais como: serviços independentes com responsabilidade única, escalabilidade, possibilidade de adoção de diversas tecnologias, múltiplos servidores de bancos de dados, dentre outras possibilidades.

A Figura 9 ilustra a representação da visão arquitetural do projeto.





MS-CONTRATOS-DATA e MS-CONTRATOS-METRICS, esse fluxo é representado pelo passo 1. Todas mensagens enviadas para o tópico MS-CONTRATOS-DATA no formato JSON cuja mensagem possui dois objetos: Header e Payload.

O objeto header contém o status da ação executada e o objeto payload contém encapsulado as informações do contrato. Os dois objetos são mesclados em um único objeto no formato *JSON*, conforme apresentado na Figura 11.

```
{
  "header": {
    "action": "CREATE"
  },
  "payload": {
    "numeroContrato": "27cf8c5f-f76e-42f5-ae48-e240ac0205b5",
    "aluno": {
      "nome": "Lenilson Teixeira Salvino",
      "email": "lenilsonts@gmail.com",
      "telefone": "19981187565",
      "prontuario": "A1420411"
    },
    "dataInicial": 1565395200000,
    "dataFinal": 1597017600000,
    "situacaoContrato": "ATIVO"
  }
}
```

**Figura 11. Exemplo de mensagem encaminhada para tópico MS-CONTRATOS-DATA**

**Fonte: elaborada pelos autores**

As mensagens enviadas para o tópico MS-CONTRATOS-DATA são encaminhadas diretamente para o MS-Relatórios, que por sua vez executa uma ação com base no tipo de ação (criação, atualização ou exclusão) contido no conteúdo da mensagem, especificamente no header. Ainda no passo 1, o tópico MS-CONTRATOS-METRICS recebe a seguinte mensagem: {"payload": "OK"} que logo em seguida é encaminhada para o MS-*Analytics*, O intuito de notificar a ocorrência de um novo evento relacionado ao MS-Contratos ou MS-Relatórios. A cada ocorrência de um novo evento, o MS-*Analytics* executa uma requisição no recurso de métricas do MS-Contratos e MS-Relatórios para obter as métricas atualizadas e centralizar o armazenamento das mesmas em sua própria base de dados, esse fluxo é ilustrado no passo 2.

O processo de execução de agenda do MS-Relatórios, quando executado realiza uma verificação por todos relatórios cadastrados, caso encontre algum relatório com atraso de entrega encapsula as seguintes informações: nome do aluno, prontuário, email, data da entrega do relatório dentro do objeto payload e envia a mensagem para o tópico MS-RELATORIOS-DATA que em seguida encaminha a mensagem para o MS-Notificador, esse fluxo é apresentado no passo 3.

Uma das vantagens em relação a abordagem de distribuição e processamento de mensageria é a garantia de um melhor desempenho dos serviços transmissores, permitindo que estes não fiquem bloqueados durante a concorrência de processamento, esse mecanismo

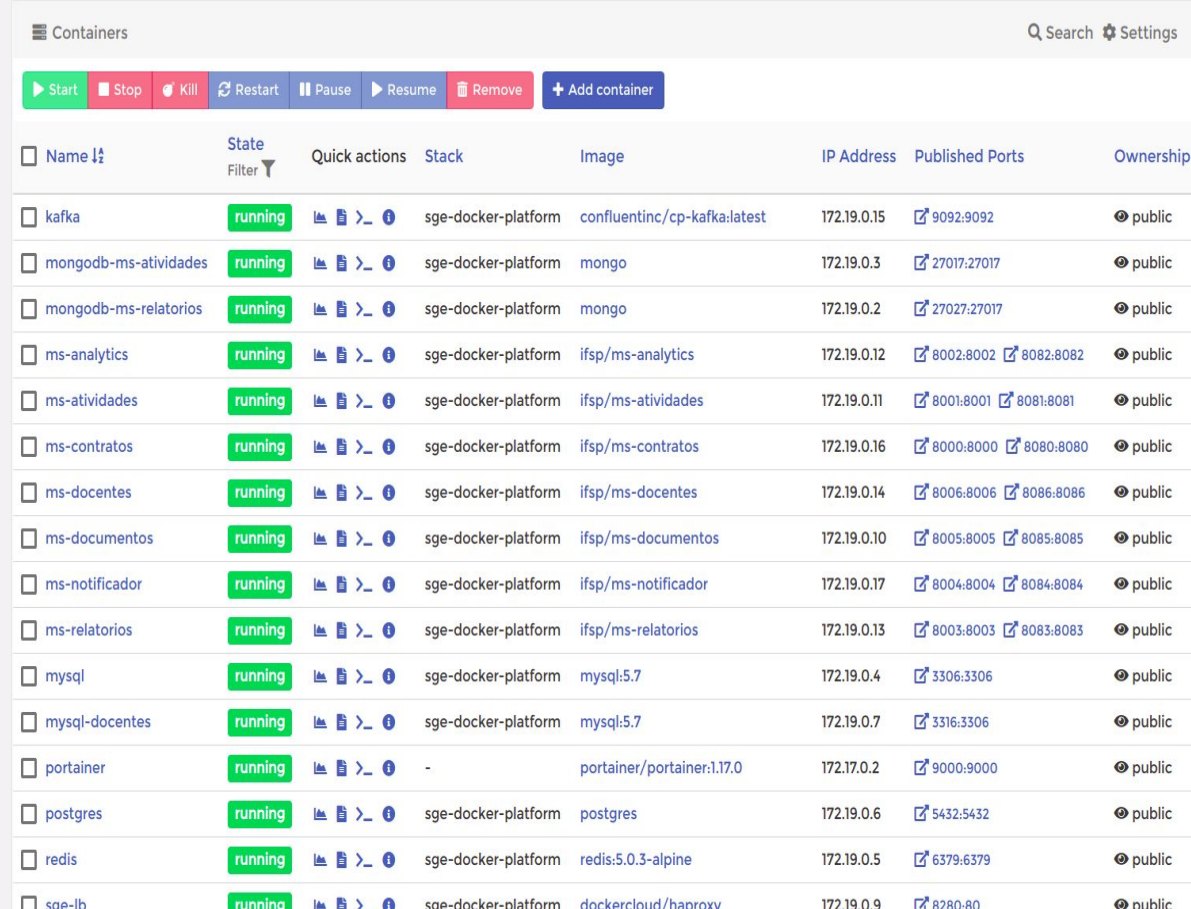
também garante uma maior escalabilidade, permitindo que filas de processamento sejam implementadas com objetivo de facilitar a replicação de serviços ouvintes em cenários de alta demanda.

## 5. Resultados

Nesta seção são apresentados os resultados do desenvolvimento do sistema de gerenciamento de estágios.

Todos os microsserviços e demais componentes da infraestrutura foram isolados em imagens gerenciáveis pelo Docker.

Por meio da interface *web* do *Portainer* a Figura 12 apresenta a listagem de todos os componentes que fazem parte da infraestrutura do projeto.



Name	State	Quick actions	Stack	Image	IP Address	Published Ports	Ownership
kafka	running		sgc-docker-platform	confluentinc/cp-kafka:latest	172.19.0.15	9092:9092	public
mongodb-ms-atividades	running		sgc-docker-platform	mongo	172.19.0.3	27017:27017	public
mongodb-ms-relatorios	running		sgc-docker-platform	mongo	172.19.0.2	27027:27017	public
ms-analytics	running		sgc-docker-platform	ifsp/ms-analytics	172.19.0.12	8002:8002  8082:8082	public
ms-atividades	running		sgc-docker-platform	ifsp/ms-atividades	172.19.0.11	8001:8001  8081:8081	public
ms-contratos	running		sgc-docker-platform	ifsp/ms-contratos	172.19.0.16	8000:8000  8080:8080	public
ms-docentes	running		sgc-docker-platform	ifsp/ms-docentes	172.19.0.14	8006:8006  8086:8086	public
ms-documentos	running		sgc-docker-platform	ifsp/ms-documentos	172.19.0.10	8005:8005  8085:8085	public
ms-notificador	running		sgc-docker-platform	ifsp/ms-notificador	172.19.0.17	8004:8004  8084:8084	public
ms-relatorios	running		sgc-docker-platform	ifsp/ms-relatorios	172.19.0.13	8003:8003  8083:8083	public
mysql	running		sgc-docker-platform	mysql:5.7	172.19.0.4	3306:3306	public
mysql-docentes	running		sgc-docker-platform	mysql:5.7	172.19.0.7	3316:3306	public
portainer	running		-	portainer/portainer:1.17.0	172.17.0.2	9000:9000	public
postgres	running		sgc-docker-platform	postgres	172.19.0.6	5432:5432	public
redis	running		sgc-docker-platform	redis:5.0.3-alpine	172.19.0.5	6379:6379	public
sgc-lb	running		sgc-docker-platform	dockercloud/haproxy	172.19.0.9	8280:80	public

**Figura 12. Listagem de todos os componentes que compõem a infraestrutura do projeto**

**Fonte: elaborada pelos autores**

Por meio da tela de cadastro de contratos, é possível gerenciar todas as informações referente ao contrato de estágio. Especificamente nesta tela, para auxiliar os usuários, foi desenvolvido um mecanismo de auto preenchimento. O sistema exige que o usuário insira somente o prontuário do aluno e para empresa é necessário inserir somente o código para que todas informações sejam carregadas, essa tela é apresentada na Figura 13.



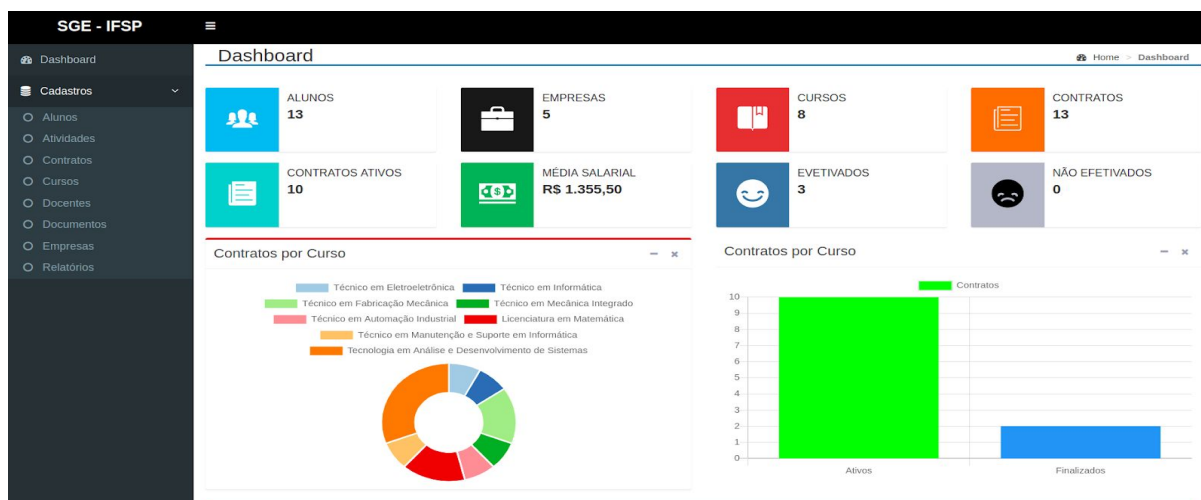
**Figura 13. Tela de cadastro de contratos**  
**Fonte: elaborada pelos autores**

A Figura 14 apresenta a tela de listagem de alunos, por meio desta dela é possível listar, cadastrar, editar e excluir um aluno:

Nome	Email	Telefone	Prontuário	Curso	Ação
Lenilson Teixeira Salvino	lenilsonts@gmail.com	19981187565	A1420411	Tecnologia em Análise e Desenvolvimento de Sistemas	
Lilian Teixeira	lilian.teixeira@gmail.com	34996683137	A1420413	Licenciatura em Matemática	
Guilherme Ricardo Alves Teixeira	guilherme@gmail.com	19981187566	A1421412	Técnico em Informática	
Luane Carla Teixeira Salvino	luane.carla@gmail.com	34996682488	A1420414	Técnico em Fabricação Mecânica	
Gabriel Lira	gabrielira@gmail.com	19987786534	A1420415	Tecnologia em Análise e Desenvolvimento de Sistemas	
Eder Antônio da Silva	eder@hotmail.com	19998765431	A1420416	Tecnologia em Análise e Desenvolvimento de Sistemas	
Michael Pablo	michael@outlook.com	19998787645	A1420417	Tecnologia em Análise e Desenvolvimento de Sistemas	
Lindomar Teixeira	lindomar@yahoo.com	34996681234	A1420419	Técnico em Automação Industrial	
Eduardo Gabriel Marques	eduardo.gabriel@gmail.com	3499872146	A1420420	Técnico em Fabricação Mecânica	
Valdivino Teixeira	valdivino@gmail.com	3432687048	A1420421	Técnico em Eletroeletrônica	

**Figura 14. Tela de listagem de alunos**  
**Fonte: elaborada pelos autores**

Com o desenvolvimento da funcionalidade de *dashboard*, é possível visualizar as seguintes informações: quantidade de alunos, total de empresas, total de cursos, total de contratos, média salarial, quantidade de alunos efetivados e não efetivados após encerramento do contrato. Todas informações apresentadas nesta tela são fornecidas pelo MS-*Analytics*. A tela de dashboard é apresentada na Figura 15.



**Figura 15. Tela de *dashboard***  
**Fonte: elaborada pelos autores**

Com intuito de apresentar *dashboard* de monitoramento desenvolvido com a ferramenta Grafana, foram realizadas durante 15 minutos várias chamadas simultâneas no recurso de consulta de alunos do MS-Contratos, com o objetivo de elevar o processamento da aplicação somente para exibir algumas informações (métricas), tais como: número de *threads* concorrentes, processamento de CPU e quantidade de conexões HTTP ativas. A Figura 16 apresenta o dashboard com base nas métricas coletadas pelo Prometheus no MS-Contratos:



**Figura 16. Dashboard de monitoramento do MS-Contratos**  
**Fonte: Captura de tela de monitoramento do Grafana, realizada pelo autor.**

## 6. Conclusões

A arquitetura de microsserviços fornece diversas vantagens, tais como: possibilidade de escalabilidade sem afetar toda a aplicação, gerenciamento de serviços por equipes distintas, heterogeneidade tecnológica, resiliência, dentre outras vantagens. Por outro lado o desenvolvimento torna-se mais complexo, exigindo implementações de mecanismos de monitoramento, recuperação de falhas, replicação de dados, comunicação, dentre outros mecanismos.

Importante enfatizar que a arquitetura de microsserviços não resolve todos os problemas relacionados ao contexto arquitetural de desenvolvimento de *software*, cada cenário possui suas particularidades, sendo assim, é importante analisar o melhor modelo arquitetural caso a caso.

Como contribuição, todo o código fonte do projeto está disponível no Github através do link: <https://github.com/LenilsonTeixeira/SGE-IFSP> podendo ser consultado, modificado e expandido.

O estudo e a implementação do projeto colaboraram para o aprendizado de diversas técnicas e tecnologias que podem ser utilizadas em futuros projetos.

Todos os conhecimentos adquiridos ao longo da jornada do curso de Análise e Desenvolvimento de Sistemas foram de grande importância e extremamente válidos para o desenvolvimento do projeto.

## 7. Trabalhos Futuros

Esta seção apresenta sugestões de trabalhos futuros que estão relacionadas à integração de novas tecnologias, possíveis melhorias e escalabilidade da rede de microsserviços que podem ser seguidas por alunos que tenham interesse em expandir o projeto. Abaixo algumas sugestões de melhorias:

- Desenvolvimento de um microsserviço de autenticação.
- Desenvolver a tela de controle de relatórios
- Adicionar um mecanismo de *API Gateway* na infraestrutura do projeto para estabelecer um ponto único de entrada.
- Adicionar mecanismos de tolerância a falha.
- Estabelecimento de um processo de integração contínua por meio de pipelines automatizados.
- Implantação do *Kubernetes* para prover escalabilidade e controle de serviços.
- Adicionar a ferramenta *Swagger* em todos os projetos para documentação das *APIs REST*.

## Referências

Amaral, O. (2017). “Arquitetura de Micro Serviços: uma Comparação com Sistemas Monolíticos”.

Barros, M. P. (2008). “Sistema Web para Gerenciamento do Processo de Estágio Supervisionado na POLI-UPE”.

Cohn, M. (2004). “Advantages of user stories for requirements.”, Disponível em:

<<https://www.mountaingoatsoftware.com/articles/advantages-of-user-stories-for-requiremen>

ts> Acessado em Outubro de 2018.

Fielding, T. (2000) “Architectural Styles and the Design of Network-based Software Architectures”.

Fowler ,M.; Lewis, J. “Microservices”, Disponível em:

<<http://martinfowler.com/articles/microservices.html>> Acessado em Setembro de 2018.

Kafka. “A Distributed streaming platform”. Disponível em:

<<https://kafka.apache.org/>> Acessado em Novembro de 2018.

Levison, M. (2017) “Definition of done vs. user stories vs. acceptance criteria.”, Disponível em:<<https://agilepainrelief.com/notesfromtooluser/2017/05/definition-of-done-vs-user-stories-vs-acceptance-criteria.html>> Acessado em Setembro de 2018.

Martin, R. (2017) "Clean Architecture: A Craftsman's Guide to Software Structure and Design".

Newman, S. (2015) “Building Microservices”. 1 ed. O'Reilly Media, Inc.

RedHat "O que é Apache Kafka". Disponível em: <<https://www.redhat.com/pt-br/topics/integration/what-is-apache-kafka>> Acessado em: Junho de 2019.

Richardson, C. (2014) "Microservices: Decomposing applications for deployability and scalability".

Rosa, T.P (2016) "Um método para o desenvolvimento de software baseado em microserviços".

Sommerville, I. (2011) “Software Engineering”, Addison-Wesley, 9ª edição

Thones, J. (2015) "Microservices".

Viana, D. D. S. (2017) "CEFET-CONNECTIONS: Aplicando uma arquitetura de microserviços em uma rede social".

# Documento Digitalizado Restrito

## Versão Final TCC

**Assunto:** Versão Final TCC  
**Assinado por:** Michele Barion  
**Tipo do Documento:** Projeto  
**Situação:** Finalizado  
**Nível de Acesso:** Restrito  
**Tipo do Conferência:** Documento Original e Cópia

Documento assinado eletronicamente por:

■ **Michele Cristiani Barion, PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 26/08/2019 09:23:39.

Este documento foi armazenado no SUAP em 26/08/2019. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifsp.edu.br/verificar-documento-externo/> e forneça os dados abaixo:

**Código Verificador:** 223177

**Código de Autenticação:** a4e11eb409

