

**UNIVERSIDADE DO GRANDE RIO**

***PROF. JOSÉ DE SOUZA HERDY***

ESCOLA DE CIÊNCIA E TECNOLOGIA

**BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

Matheus de Oliveira Manhães

Felipe Oliveira Barbosa

**Chaos Engineering**

Duque de Caxias

2019



**UNIVERSIDADE DO GRANDE RIO**

***PROF. JOSÉ DE SOUZA HERDY***

ESCOLA DE CIÊNCIA E TECNOLOGIA

**BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

Matheus de Oliveira Manhães

Felipe Oliveira Barbosa

**Chaos Engineering**

Projeto Final de Curso apresentado à Universidade do Grande Rio “*Prof. José de Souza Herdy*” (UNIGRANRIO) como parte dos requisitos para conclusão do curso de Bacharelado em Sistemas de Informação.

Orientador: Prof. Dr. Thiago Silva de Souza

Duque de Caxias

2019

**Chaos Engineering**

Matheus de Oliveira Manhães – 5306315

Felipe Oliveira Barbosa – 5306312

Projeto Final de Curso apresentado à Universidade do Grande Rio “*Prof. José de Souza Herdy*” (UNIGRANRIO) como parte dos requisitos para conclusão do curso de Bacharelado em Sistemas de Informação.

Banca Examinadora:

1. Orientador e Presidente: Prof. Dr. Thiago Silva de Souza
2. Membro interno: Prof. <MEMBRO DA BANCA>
3. Membro interno: Prof. <MEMBRO DA BANCA>

Duque de Caxias

2019

Matheus de Oliveira Manhães  
Felipe Oliveira Barbosa

Chaos Engineering, Duque de Caxias, 2019

<Nº páginas pré-textuais. Exemplo: XIII>, <número total de páginas do texto do projeto> p. 29,7 cm. (Escola de Ciência e Tecnologia, 2019)

Projeto de Final de Curso - Universidade do Grande Rio, Escola de Ciência e Tecnologia.

1. <Assunto 1>
2. <Assunto 2>
3. <Assunto 3>

I. EIN/UNIGRANRIO II. Título (série)

<DEDICATÓRIA DOS ALUNOS>

**AGRADECIMENTOS**

<AGRADECIMENTOS>

“<EPÍGRAFE>”

<AUTOR EPÍGRAFE>

**RESUMO**

O que antes se considerava um tabu em questões de segurança e privacidade, hoje é possível observar que o uso de aplicativos e sistemas tem sido cada vez mais inerente à cultura e à rotina do ser humano. Vemos, por exemplo, o quão comum é uso do Netflix. A plataforma de vídeos sob demanda tem crescido muito e cada vez mais conquista clientes, ou usuários, ao redor do mundo. É possível perceber o quanto é investido para que os serviços da Netflix permaneçam disponíveis para seus usuários, em situações e dispositivos diversos. Estudar a estrutura de grandes sistemas na *World Wide Web* pode auxiliar na maneira como novos aplicativos e sistemas podem ser construídos garantindo qualidade e disponibilidade para os usuários. Uma das formas em que grandes empresas como Netflix, Amazon, Google, Microsoft, Uber e Dropbox conseguem manter seus serviços com qualidade é usando *Chaos Engineering.*

*Chaos Engineering* é o foco deste trabalho, que mostra como o processo de Engenharia do Caos funciona na execução e criação de ataques simulando rotinas de trabalho comuns, na tentativa de gerar turbulência a fim de preparar os engenheiros de software para as situações onde o *software* pode se tornar indisponível, como interrupções de energia, congestionamento de requisições e demora em relação ao tempo de resposta afim de gerar resiliência e descobrir pontos fracos.

Este documento mostra como é feito o processo de Engenharia do Caos através de quatro pequenos sistemas criados para este trabalho: um buscador de repositórios de usuários do GitHub, uma API RESTful de cadastro de termos úteis relacionados a *Chaos Engineering,* uma calculadora comum e um gerenciador de tarefas. A ideia é usar essas aplicações como alvo de nossos experimentos de Caos.

**Palavras-chave:** Aplicativos, Sistemas, Web, Internet, Ataques, Experimento, Netflix, Engenharia do Caos.

**SUMÁRIO**

[1 – INTRODUÇÃO 2](#_Toc25369380)

[1.1 – Motivação 2](#_Toc25369381)

[1.2 – Problema 2](#_Toc25369382)

[1.3 – Hipótese 2](#_Toc25369383)

[1.4 – Objetivos 2](#_Toc25369384)

[1.5 – Organização do Trabalho 2](#_Toc25369385)

[2 – REFERENCIAL TEÓRICO 2](#_Toc25369386)

[2.1 – Netflix 2](#_Toc25369387)

[2.2 – Chaos Engineering 2](#_Toc25369388)

[3 – PROVA DE CONCEITO 2](#_Toc25369389)

[3.1 – Descrição do Problema 2](#_Toc25369390)

[3.2 – Solução Proposta 2](#_Toc25369391)

[3.3 – Arquitetura da Pesquisa 2](#_Toc25369392)

[3.3.1 – Criação das Aplicações 2](#_Toc25369393)

[3.3.1.1 – Calculadora 2](#_Toc25369394)

[3.3.1.2 – Gerenciador de Tarefas 2](#_Toc25369395)

[3.3.1.3 – Buscador de Repositórios 2](#_Toc25369396)

[3.3.1.4 – API REST de Termos Úteis 2](#_Toc25369397)

[3.3.2 – Contêinerização das Aplicações 2](#_Toc25369398)

[3.3.3 – Osquestração dos Contêineres 2](#_Toc25369399)

[3.3.4 – Deploy dos Pods 2](#_Toc25369400)

[3.3.5 – Execução dos Testes 2](#_Toc25369401)

[3.3.5.1 – “Alô Mundo” no Gremlin 2](#_Toc25369402)

[3.3.5.1.1 – CPU Attack 2](#_Toc25369403)

[3.3.5.2 – Gremlin e Kubernetes 2](#_Toc25369404)

[3.3.6 – Criação dos Testes 2](#_Toc25369405)

[3.4 – Tecnologias Utilizadas 2](#_Toc25369406)

[3.4.1 – Chaos Toolkit 2](#_Toc25369407)

[3.4.2 – Gremlin 2](#_Toc25369408)

[3.4.3 – HTML5 2](#_Toc25369409)

[3.4.4 – CSS3 2](#_Toc25369410)

[3.4.5 – JavaScript 2](#_Toc25369411)

[3.4.6 – Node.js 2](#_Toc25369412)

[3.4.7 – Angular 2](#_Toc25369413)

[3.4.8 – Docker 2](#_Toc25369414)

[3.4.9 – Kubernetes Minikube 2](#_Toc25369415)

[3.4.10 – AWS 2](#_Toc25369416)

[4 – CONCLUSÃO 2](#_Toc25369417)

[4.1 – Considerações Finais 2](#_Toc25369418)

[4.2 – Contribuições 2](#_Toc25369419)

[4.3 – Trabalhos Futuros 2](#_Toc25369420)

[REFERÊNCIAS BIBLIOGRÁFICAS 2](#_Toc25369421)

[ANEXO I – Glossário 2](#_Toc25369422)

**LISTA DE FIGURAS**

[Figura 1 - Arquitetura do projeto 2](#_Toc25368623)

[Figura 2 - Calculadora na visão desktop 2](#_Toc25368624)

[Figura 3 - Calculadora na visão mobile 2](#_Toc25368625)

[Figura 4 - Tela Inicial sem tarefas 2](#_Toc25368626)

[Figura 5 - Formulário de inclusão de tarefas 2](#_Toc25368627)

[Figura 6 - Listagem de tarefas salvas no Local Storage 2](#_Toc25368628)

[Figura 7 - Listagem de tarefas 2](#_Toc25368629)

[Figura 8 - Tela de edição de tarefas 2](#_Toc25368630)

[Figura 9 - Remoção de tarefas 2](#_Toc25368631)

[Figura 10 - Sucesso na remoção de tarefas 2](#_Toc25368632)

[Figura 11 - Tela inicial do Buscador de Repositórios 2](#_Toc25368633)

[Figura 12 - Accordion sem usuário na barra de pesquisa 2](#_Toc25368634)

[Figura 13 - Resultado de sucesso no Buscador de Repositórios 2](#_Toc25368635)

[Figura 14 - Accordion em resultado de sucesso no Buscador de Repositórios 2](#_Toc25368636)

[Figura 15 - Resultado de sucesso sobrescrito no Buscador de Repositórios 2](#_Toc25368637)

[Figura 16 - Mensagem de sucesso do Docker 2](#_Toc25368638)

[Figura 17 - Modelo de definição dos termos da API 2](#_Toc25368639)

[Figura 18 - Criação de termos na API com Insomnia 2](#_Toc25368640)

[Figura 19 - Listagem de termos na API com Insomnia 2](#_Toc25368641)

[Figura 20 - Listagem de termos da API no navegador 2](#_Toc25368642)

[Figura 21 - Termo criado na API e persistido no MongoDB 2](#_Toc25368643)

[Figura 22 - Docker exibindo as aplicações do projeto 2](#_Toc25368644)

[Figura 23 - Tela de overview do Kubernetes 2](#_Toc25368645)

[Figura 24 - Resultado do ataque de shutdown do Gremlin nos pods do Kubernetes 2](#_Toc25368646)

[Figura 25 – Console do AWS exibindo o resultado do ataque de CPU do Gremlin 2](#_Toc25368647)

[Figura 26 - Arquitetura do Chaos Tookit 2](#_Toc25368648)

**LISTA DE ABREVIATURAS E SIGLAS**

1. API
2. AWS
3. CSS3
4. CPU
5. FaaS
6. HTML5
7. JSON
8. NoSQL
9. REST

# 1 – INTRODUÇÃO

## 1.1 – Motivação

## 1.2 – Problema

## 1.3 – Hipótese

## 1.4 – Objetivos

## 1.5 – Organização do Trabalho

# 

# 

# 2 – REFERENCIAL TEÓRICO

## 2.1 – Netflix

## 2.2 – Chaos Engineering

# 3 – PROVA DE CONCEITO

Este capítulo está dividido em quatro seções, a seção 3.1 trata da descrição do problema e como isso motiva toda a solução proposta, a seção 3.2 apresenta a solução proposta com o intuito de justificar a pesquisa e o problema, a seção 3.3 apresenta a arquitetura da pesquisa descrevendo o processo de criação e apresentação das aplicações, o *setup* de contêinerização e gestão das aplicações, assim como a execução e criação dos ataques e finalmente a seção 3.4 apresenta as tecnologias utilizadas, ou seja, todos os recursos de software, ferramentas, bibliotecas, *frameworks*, linguagens de programação e tecnologias em geral que foram necessárias para elaboração do projeto.

## 3.1 – Descrição do Problema

Com o avanço da tecnologia o mundo está cada vez mais conectado. Sistemas web estão sendo construídos e novidades em aplicativos estão sendo lançadas frequentemente. O grande público, independente da faixa etária, cada vez mais tem se adaptado às novas tecnologias para se manter atualizado às novidades.

Percebendo isso, os aplicativos e sistemas web provocaram uma enorme transformação na maneira como lidamos com a nossa rotina pessoal, profissional, sentimental e acadêmica. Observando essa realidade, é importante refletir a respeito do que é necessário para garantir que esses aplicativos e sistemas web se mantenham disponíveis sempre que os usuários precisem utilizar, além de garantir que as falhas e turbulências de produção devem ser consideradas e, além disso, serem tratadas da forma correta a fim de não mais ocorrerem ou ocorrerem em menor escala, garantindo a melhor experiência possível para o usuário.

Levando em conta esta problemática, decidiu-se explorar a maneira como é possível garantir disponibilidade e resiliência de aplicações de grandes empresas como Netflix, analisando o seu processo de Engenharia do Caos a fim de gerar conhecimento levando em consideração seus experimentos e como suas falhas podem se tornar aprendizado no contexto de desenvolvimento de software.

## 3.2 – Solução Proposta

Com o objetivo de explorar este problema, foi proposta a simulação de um ambiente de produção onde são servidas aplicações *web* construídas a fim de serem alvos de ataques de experimentos realizados baseados nos princípios da Engenharia do Caos.

## 3.3 – Arquitetura da Pesquisa

A figura 1 representa a arquitetura de todo o projeto, desde a criação das aplicações até a execução e criação dos ataques. O processo é dividido em 5 etapas após a criação das quatro aplicações. 1 – Uso do Docker para configurar as quatro aplicações em contêineres. 2 – Uso do Minikube para instalação do Kubernetes localmente afim de gerir os contêineres Docker. 3 – D*eploy* dos *pods* no Amazon Web Services. 4 – Execução dos ataques de CPU e Shutdown com Gremlin. 5 – Criação e execução de ataques com o Chaos Toolkit.

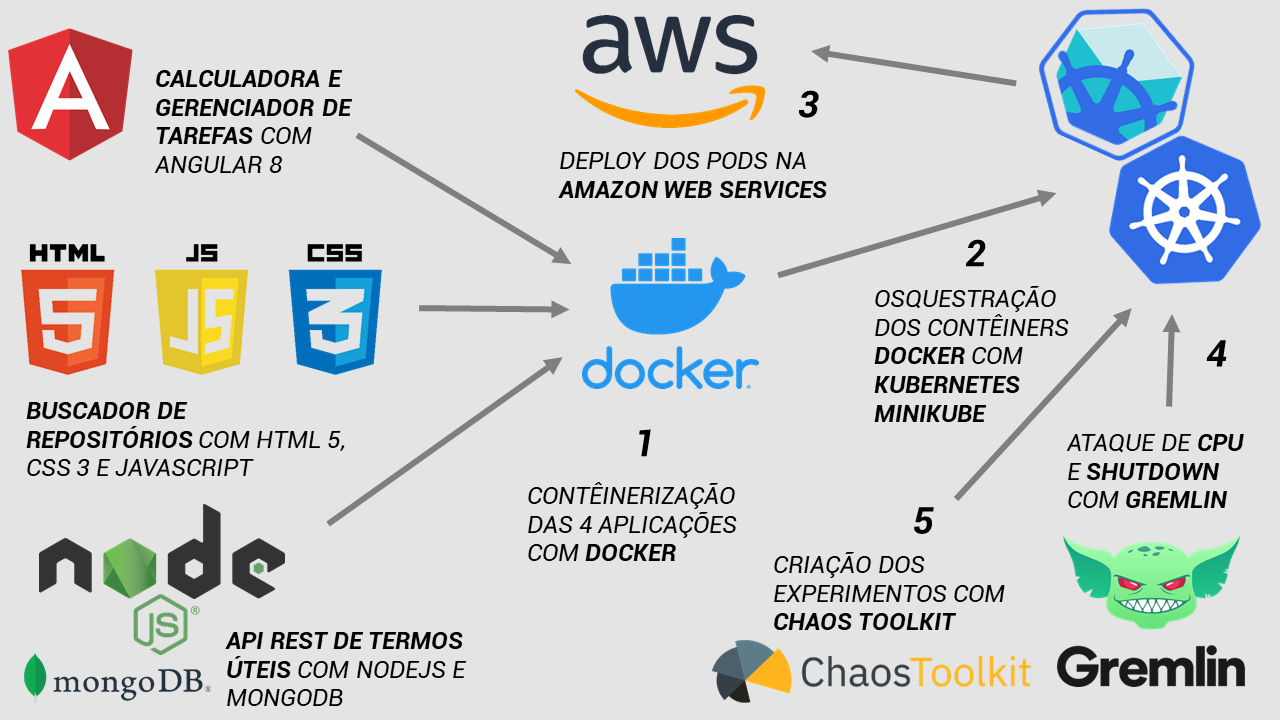


Figura 1 - Arquitetura do projeto

### 3.3.1 – Criação das Aplicações

Nesta seção serão apresentadas as aplicações que foram criadas para simular um ambiente comum de produção a fim de servirem como alvos para os experimentos do Gremlin e os experimentos criados com Chaos Toolkit. Estas aplicações estão divididas em 4 seções: a seção 3.3.1.1 descrevendo a calculadora, 3.3.1.2 descrevendo o gerenciador de tarefas, 3.3.1.3 o buscador de repositórios e finalmente a seção 3.3.1.4 descrevendo a API REST que permite a inclusão, remoção, edição, listagem e pesquisa de termos úteis relacionados a Engenharia do Caos.

##### 3.3.1.1 – Calculadora

Feita com Angular 8, a calculadora é uma aplicação simples que realiza tudo o que uma calculadora comum faz: operações de soma, adição, subtração e divisão. Tem o visual responsivo, ou seja, a interface se adapta independente do dispositivo que o usuário esteja utilizando (computador, TV, celular, tablet etc.). A figura 2 e a figura 3 mostram a única tela que essa aplicação possui, tanto na visão pelo computador, como pelo celular.

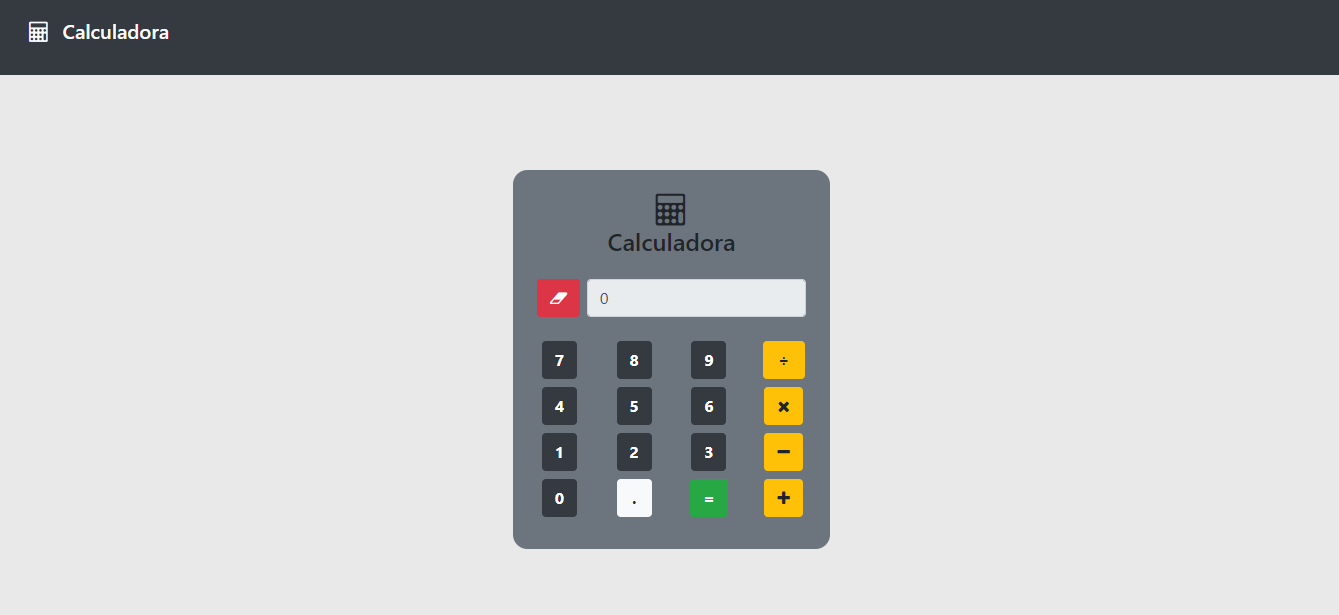


Figura 2 - Calculadora na visão desktop



Figura 3 - Calculadora na visão mobile

##### 3.3.1.2 – Gerenciador de Tarefas

O Gerenciador de Tarefas é uma aplicação que permite o usuário incluir, remover, editar, pesquisar, listar e alterar o status de tarefas. Desenvolvido com o Angular em sua versão 8, tem a interface responsiva e de fácil interação. A figura 4 mostra como é a sua tela inicial sem haver tarefa alguma cadastrada. Acima da tabela que exibe as tarefas, existe um botão chamado “Adicionar Tarefa”. Ao clicar neste botão a tela de formulário de inclusão de tarefas é exibido.

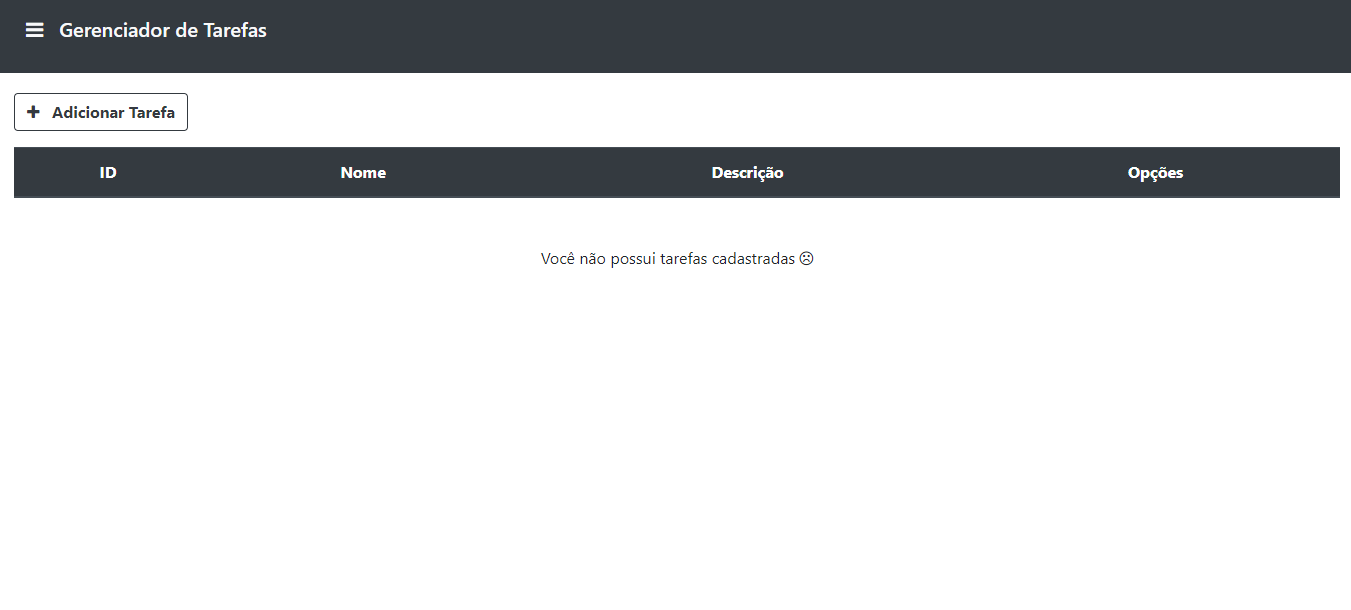


Figura 4 - Tela Inicial sem tarefas

Assim que o botão para adicionar tarefas é selecionado, um formulário que solicita informações como nome e descrição da tarefa é exibido. A figura 5 é apresentada abaixo para ilustrar esta tela.

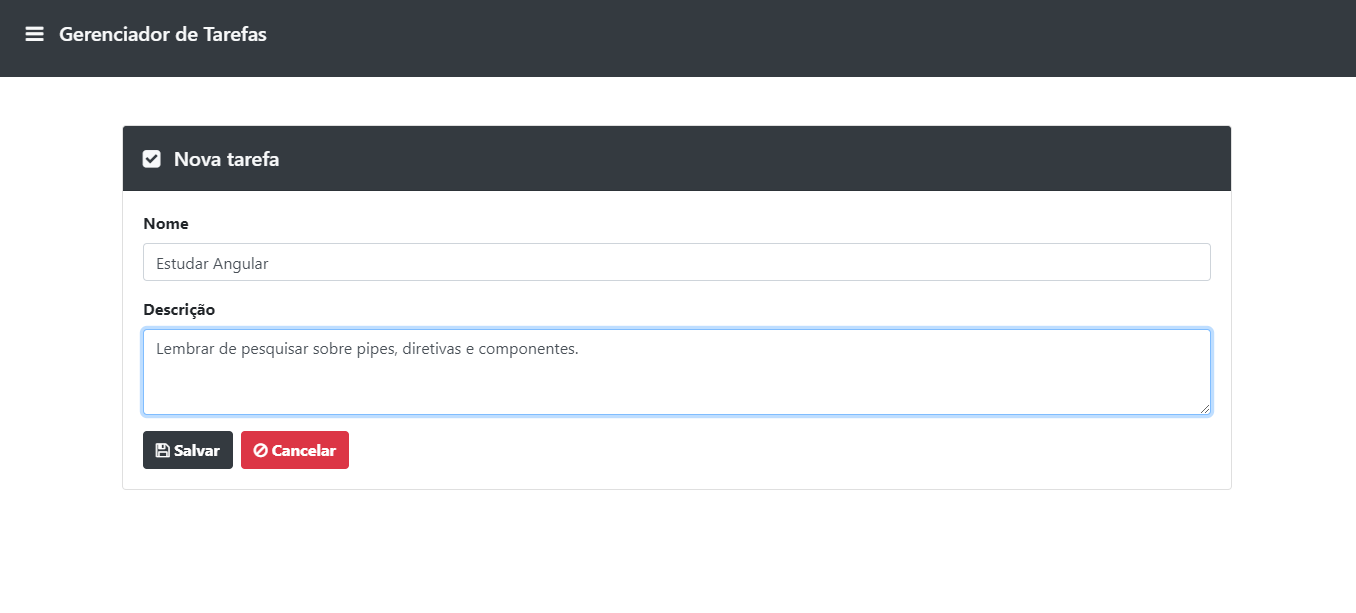


Figura 5 - Formulário de inclusão de tarefas

No formulário de inclusão de tarefas é obrigatório a inclusão do nome e descrição da tarefa. Ao clicar no botão de cancelar, todos os dados do formulário são perdidos e o usuário é conduzido pelo fluxo da aplicação até a tela de listagem de tarefas. Se o usuário clicar no botão de salvar a tarefa, os dados do formulário são salvos no *local storage,* onde ficam salvos no lado do cliente (navegador), não expiram, só são removidos por ação do cliente e são exibidos formatados e estilizados na tela de listagem de tarefas, conforme mostra a figura 6 e a figura 7 abaixo.

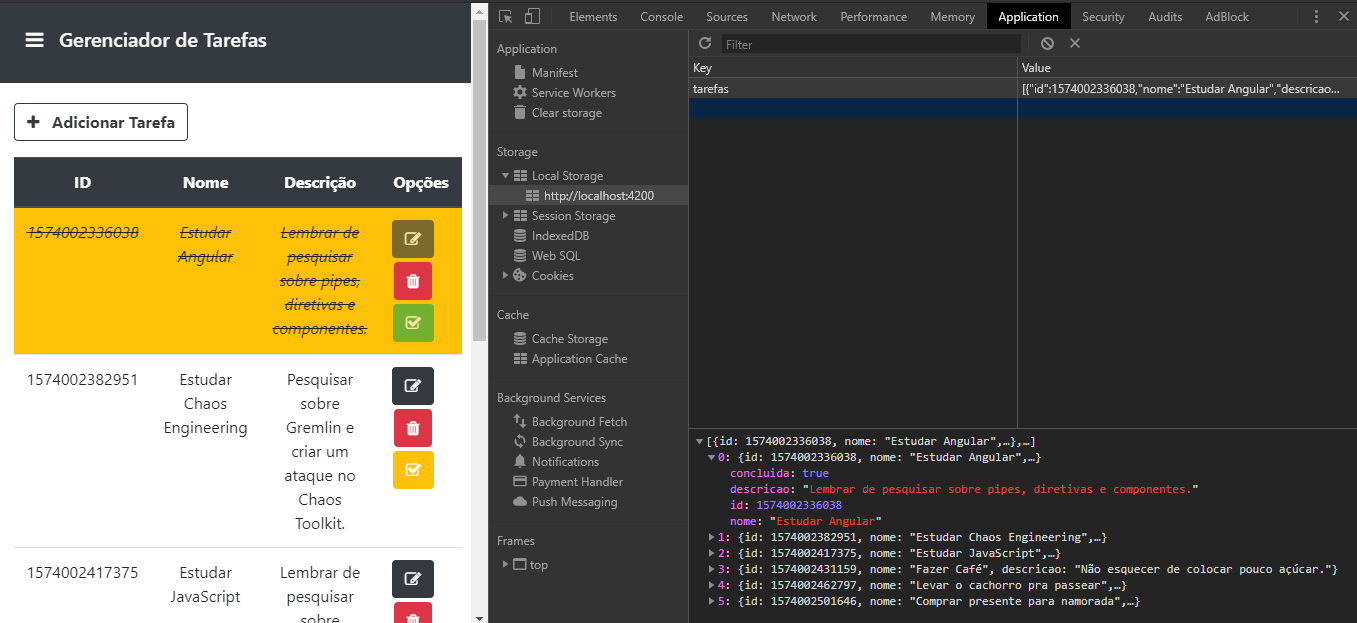


Figura 6 - Listagem de tarefas salvas no Local Storage



Figura 7 - Listagem de tarefas

A figura 7 acima ilustra a tela de exibição das tarefas criadas em uma lista. É possível perceber que a tabela que faz a listagem de tarefas possui quatro colunas: a primeira exibe os IDs gerados automaticamente para cada tarefa, a segunda exibe os nomes escolhidos pelo usuário no formulário de inclusão de tarefas, assim como na terceira coluna que exibe a descrição. A quarta coluna exibe três botões para que seja possível, nesta ordem, editar, excluir e alterar o status de uma tarefa para marcá-la como pronta. É possível perceber na figura 7 acima que algumas tarefas foram marcadas como prontas, ou seja, tiveram seus status alterados, a partir dessa condição, a cor de fundo desta tarefa é alterada para amarelo, o texto também tem seu estilo alterado para que fique em itálico e taxado. Além disso, é possível perceber que o botão de editar e alterar status ficam desabilitados, somente dando a opção de o usuário excluir aquela tarefa, partindo do pressuposto de que se o usuário concluiu tal tarefa, a mesma não pode ser editada nem ter seu status alterado novamente.

Caso o usuário pressione o botão para editar uma tarefa, o fluxo da aplicação conduz o usuário para o formulário de inclusão de tarefas, porém como se trata de uma edição, o formulário é exibido preenchido com os dados fornecidos pelo usuário previamente para que seja necessário somente a alteração que o usuário deseja. A figura 8 abaixo ilustra este caso.

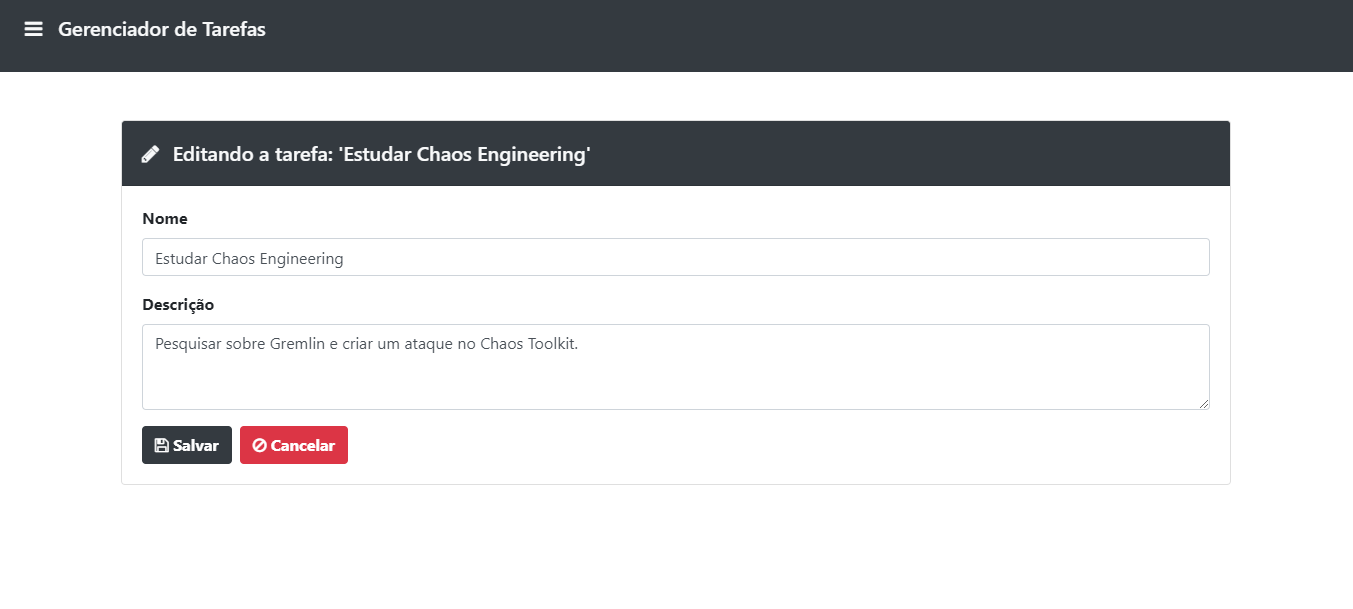


Figura 8 - Tela de edição de tarefas

A figura 8 acima mostra como a tela de edição de tarefas exibe uma tarefa previamente cadastrada para que seja possível a alteração de seu conteúdo. Caso o usuário clique em “Salvar”, todas as alterações são salvas novamente no *local storage* e a tarefa é exibida atualizada na lista de tarefas, mas caso o cliente clique em “Cancelar” nada é feito e a tarefa volta a ser exibida como estava previamente.

Conforme a figura 9 abaixo é possível perceber que aplicação também disponibiliza que o usuário faça a exclusão de uma tarefa. É exibido um modal de alerta para que o usuário se certifique de que a tarefa será excluída.

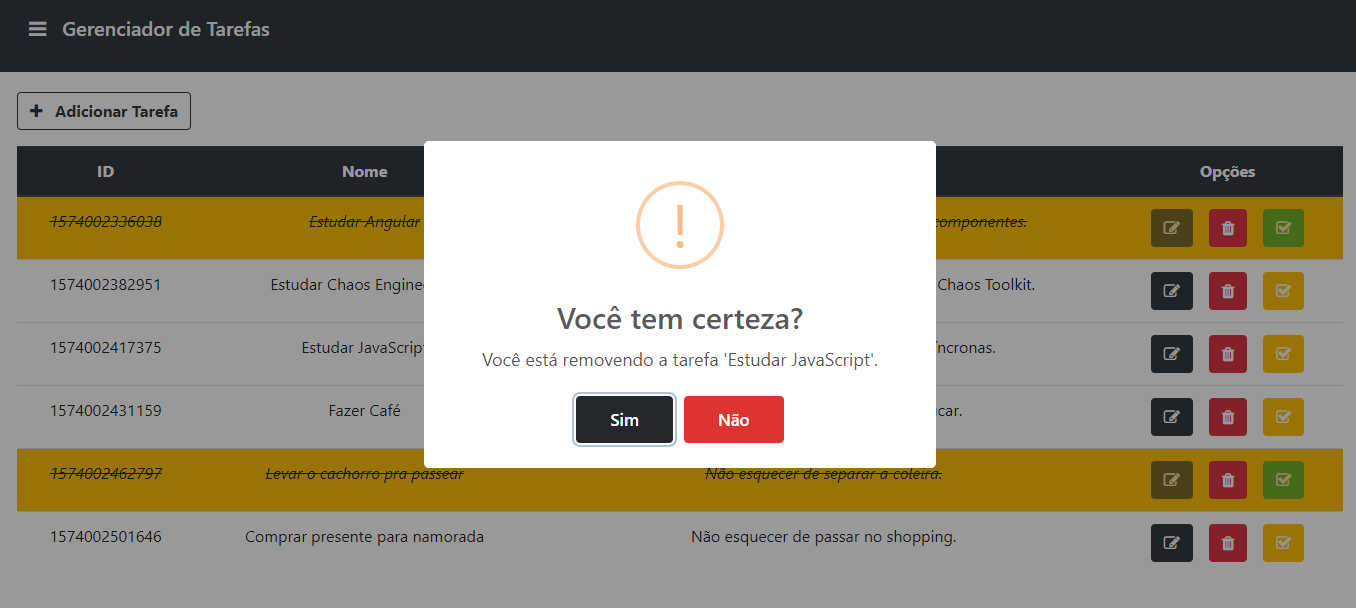


Figura 9 - Remoção de tarefas

Caso o usuário clique em “Cancelar”, o processo de exclusão da tarefa é revertido e nada acontece, porém, clicando em “Sim” o usuário remove a tarefa da lista de tarefas e também da lista no *local storage.* A figura 10 abaixo mostra como o modal muda seu comportamento exibindo uma mensagem de sucesso para o usuário caso a opção “Sim” seja escolhida. Clicando em “Ok” o usuário é redirecionado para a tela de listagem de tarefas.

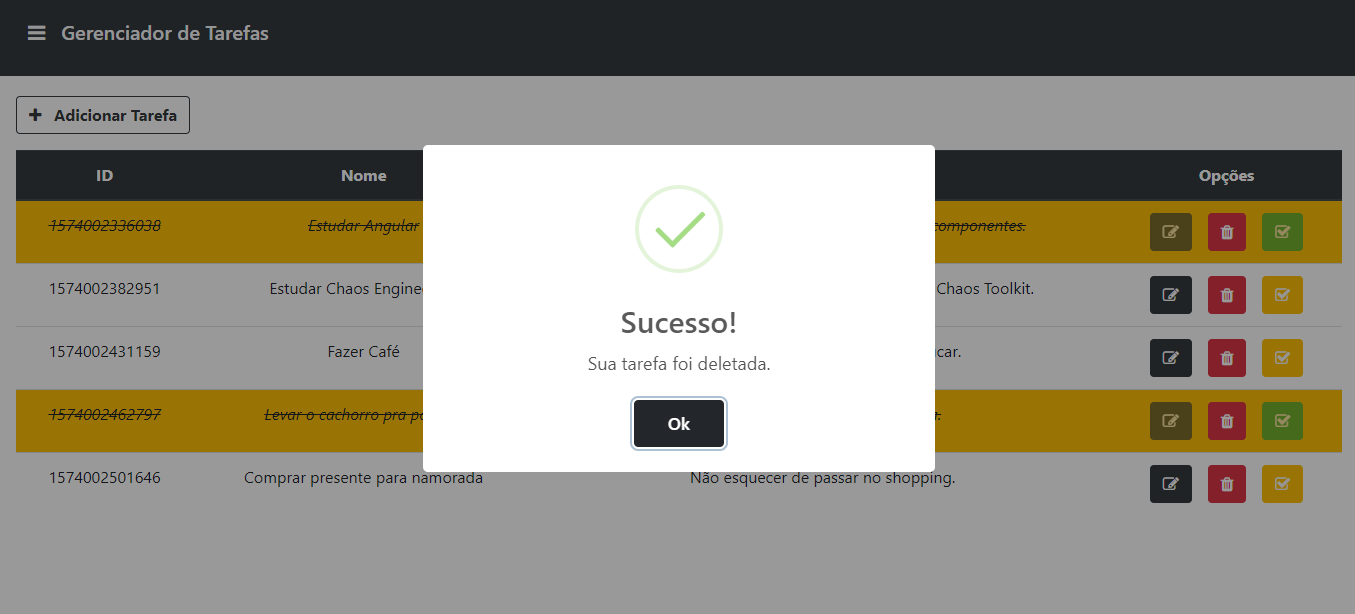


Figura 10 - Sucesso na remoção de tarefas

##### 3.3.1.3 – Buscador de Repositórios

O Buscador de Repositórios é uma aplicação web que permite que o usuário digite o nome de um usuário qualquer do GitHub e clique no Botão “Pesquisar”. A aplicação faz uma requisição na API do GitHub para obter algumas informações úteis a respeito do usuário pesquisado. O perfil deste usuário é exibido na tela contendo seu nome, localidade e avatar (foto cadastrada no GitHub), além de possuir um *link* que funciona como um *accordion* permitindo o usuário visualizar todos os repositórios públicos deste membro do GitHub pesquisado. Desenvolvido com HTML, CSS3 com Bootstrap e JavaScript, tem a interface responsiva e de fácil interação. A figura 11 abaixo mostra como é a tela inicial sem haver usuário algum pesquisado.

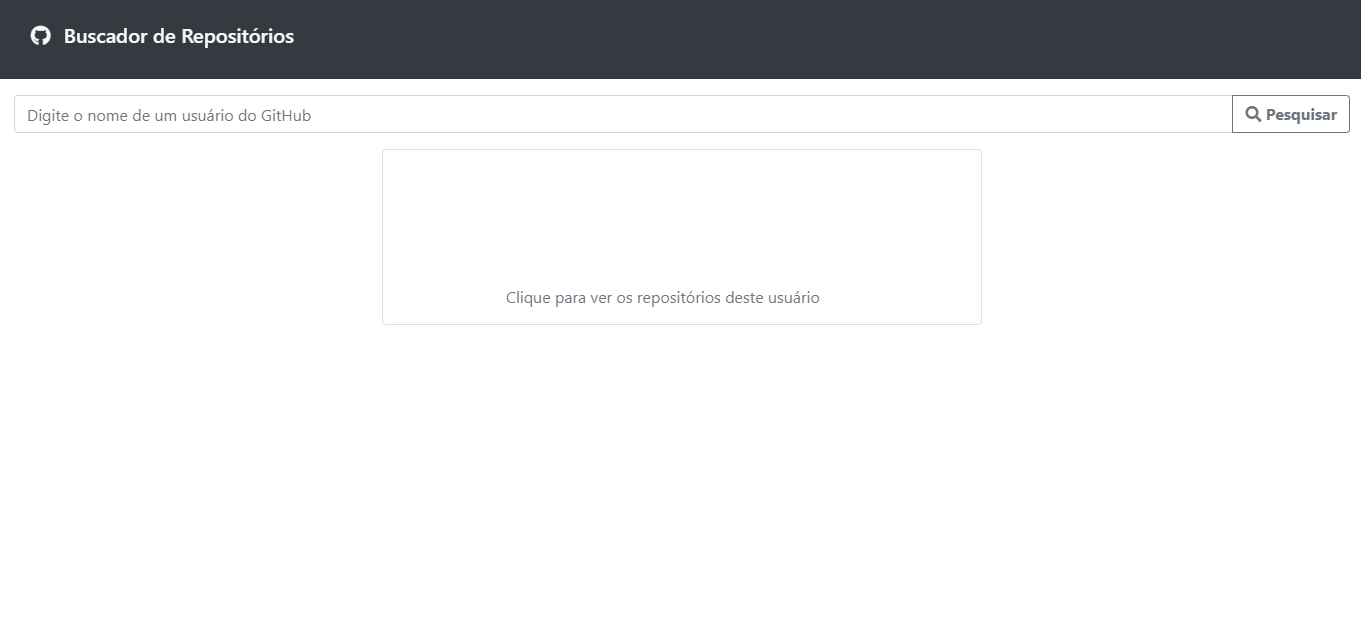


Figura 11 - Tela inicial do Buscador de Repositórios

Abaixo do *card* que exibe os usuários do GitHub, existe um *link* chamado “Clique para ver os repositórios deste usuário”. Ao clicar neste *link* a tabela de repositórios do usuário é exibida. Caso o usuário não tenha digitado na barra de pesquisa, a mensagem “Primeiro digite o nome do usuário” é exibida. A figura 12 abaixo mostra este comportamento.

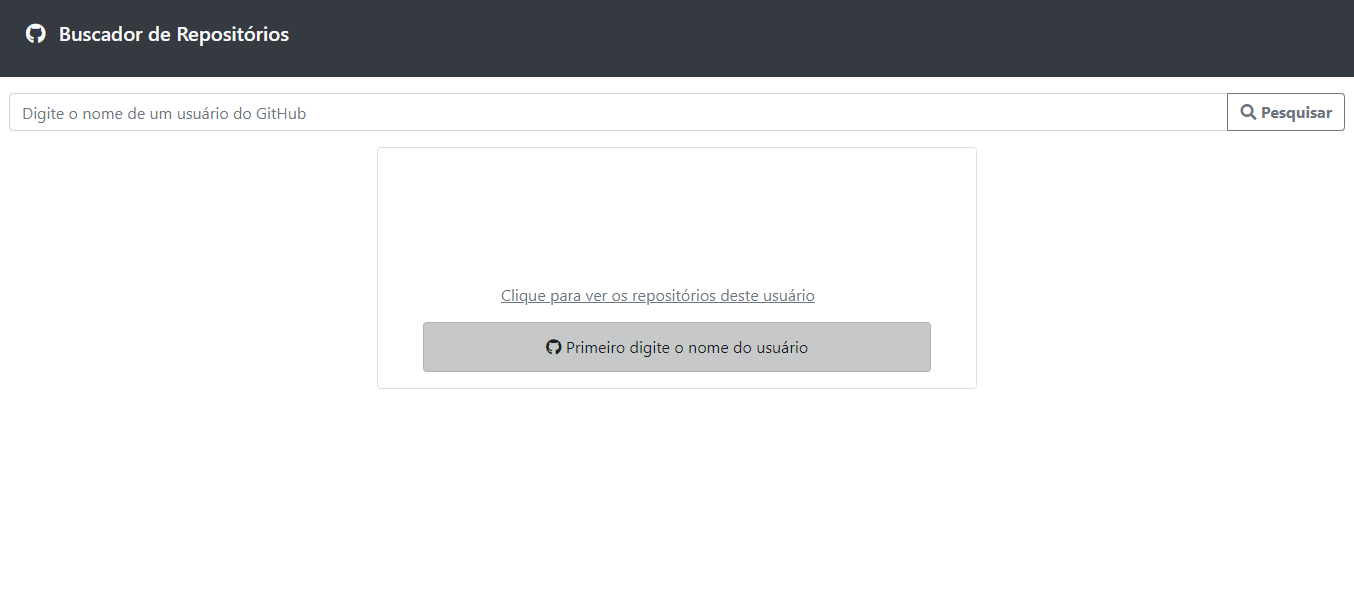


Figura 12 - Accordion sem usuário na barra de pesquisa

Digitando “momanhaes” na barra de pesquisa é possível perceber o resultado da pesquisa. Uma requisição GET é feita à API do GitHub utilizando a biblioteca Axios passando o usuário “momanhaes” digitado na barra de pesquisa como parâmetro. Caso o usuário não exista, o erro 404 é exibido com um tratamento personalizado para que a mensagem seja mais amigável para o usuário. Se o campo para inserir o usuário não for preenchido e o botão para pesquisar for clicado, um erro é exibido na tela para que o usuário digite um usuário válido do GitHub. É possível perceber um resultado de sucesso analisando a figura 13 abaixo.

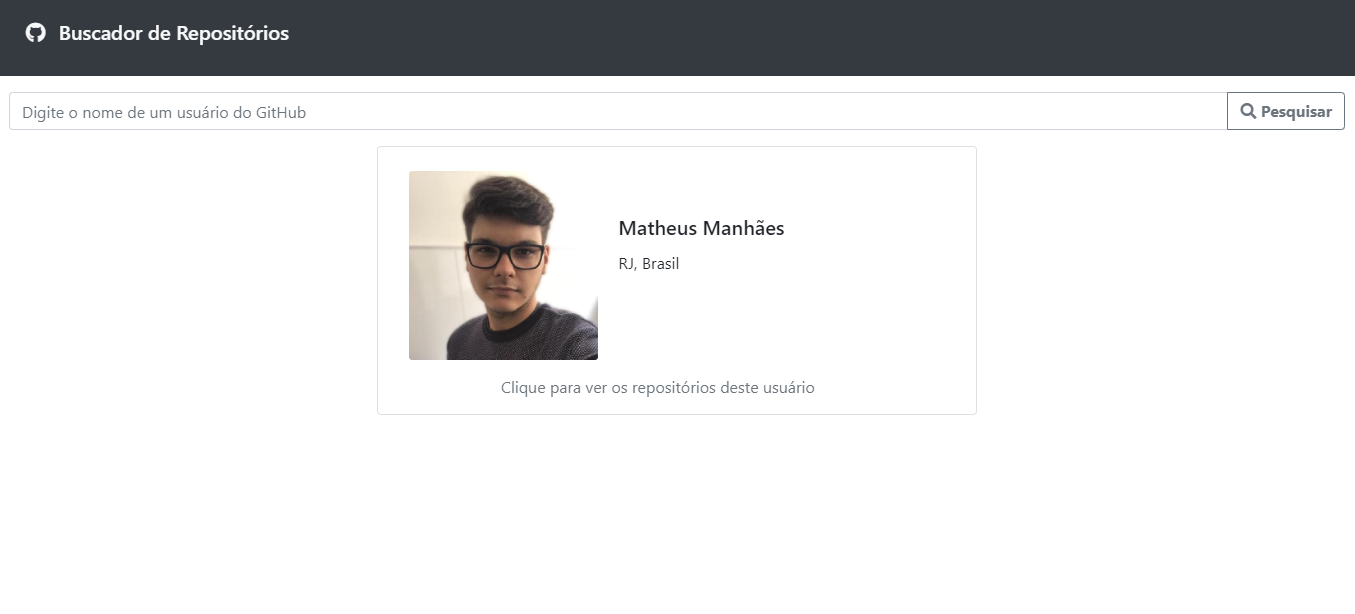


Figura 13 - Resultado de sucesso no Buscador de Repositórios

É possível, analisando a figura 13 acima, que a aplicação trouxe à tela principal o avatar do usuário “momanhaes”, assim como sua localização e seu nome. Clicando no link abaixo de sua foto é possível verificar os repositórios públicos e *forks* deste usuário. Na figura 14 abaixo é possível analisar este resultado.



Figura 14 - Accordion em resultado de sucesso no Buscador de Repositórios

Caso seja necessário buscar por outro usuário do GitHub na aplicação, todos os dados da pesquisa anterior são descartados e os dados do novo usuário são preenchidos. É possível perceber este comportamento na figura 15 abaixo. Com os dados preenchidos anteriormente como resultado da pesquisa do usuário “momanhaes”, foi pesquisado pelo usuário “felipe-b-oliveira” e o resultado dessa nova pesquisa sobrescreveram o resultado da pesquisa anterior.

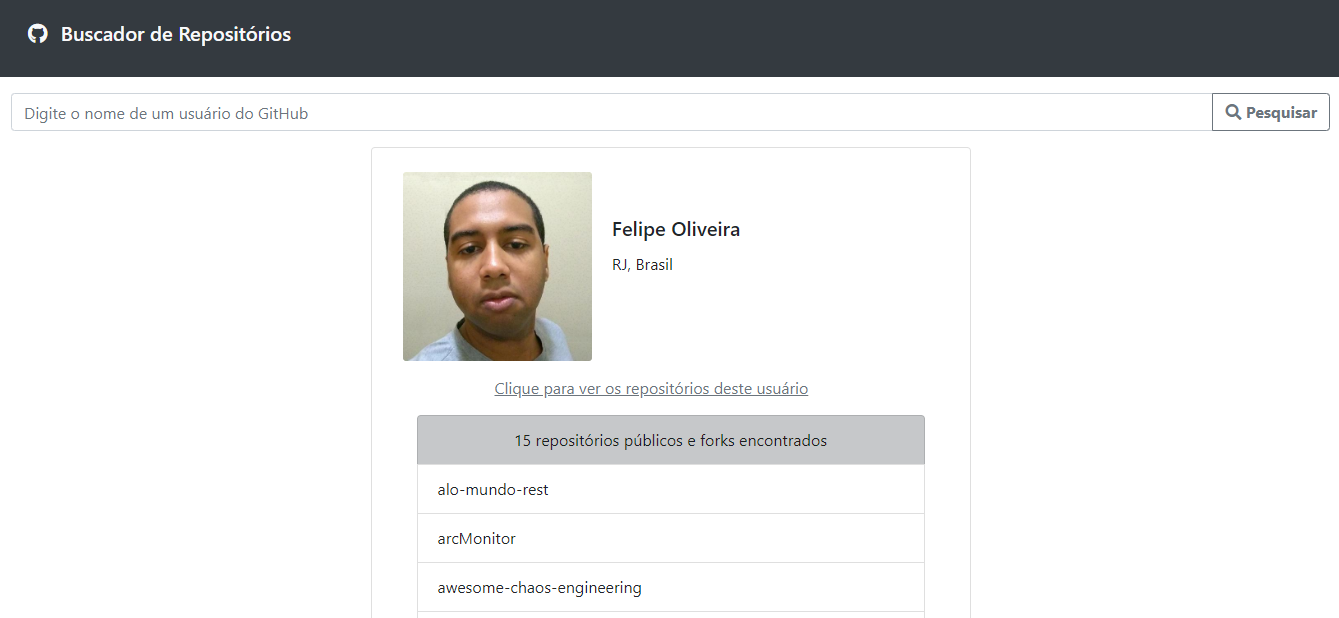


Figura 15 - Resultado de sucesso sobrescrito no Buscador de Repositórios

##### 3.3.1.4 – API REST de Termos Úteis

A API de Termos Úteis é um servidor RESTful onde são usados os verbos HTTP para disponibilizar os recursos do servidor, sendo possível que uma aplicação *front-end* da camada de interface interaja com esse servidor a fim de consumir seus serviços. Os recursos usados nesta API são termos úteis da disciplina de Chaos Engineering. É possível inserir um novo termo, listar, editar, excluir e buscar um termo pelo seu ID. Os verbos utilizados neste servidor são: GET, POST, PUT, DELETE. Onde GET será usado exclusivamente para busca de termos. POST será utilizado para a criação de novos termos. PUT será utilizado para realizar alterações em termos já existentes. DELETE será usado para a remoção de termos.

Esta API foi desenvolvida com a linguagem de programação JavaScript, utilizando a plataforma Node.js. O servidor foi escrito usando Express, um *framework* utilizado para desenvolvimento de aplicações *web* e que facilita várias rotinas e disponibiliza alguns recursos. Os dados são persistidos no MongoDB, um software de banco de dados gratuito e NoSQL. O MongoDB é orientado a documentos, o que significa que os registros guardados no banco seguem uma estrutura flexível e adaptável. Seus documentos são semelhantes ao formato JSON. O MongoDB não foi instalado em uma máquina local, mas em um contêiner Docker, a fim de não causar uma dependência dos dados persistidos estarem em uma máquina física.

Após fazer toda a configuração para baixar o contêiner do MongoDB, definição do seu seu nome e redirecionamento de porta é necessário verificar quais imagens estão disponíveis através do comando docker ps –a, verificar quais imagens estão rodando através do comando docker ps e ir até o navegador e digitar localhost:27017. Caso a imagem do MongoDB não esteja rodando, é necessário executar o comando docker start mongo. Se o banco de dados estiver disponível, deverá ser exibida uma mensagem no navegador partindo do Docker, conforme ilustra a figura 16 abaixo. A mensagem diz: “*It looks like you are trying to access mongoDB over HTTP on the native driver port*”.

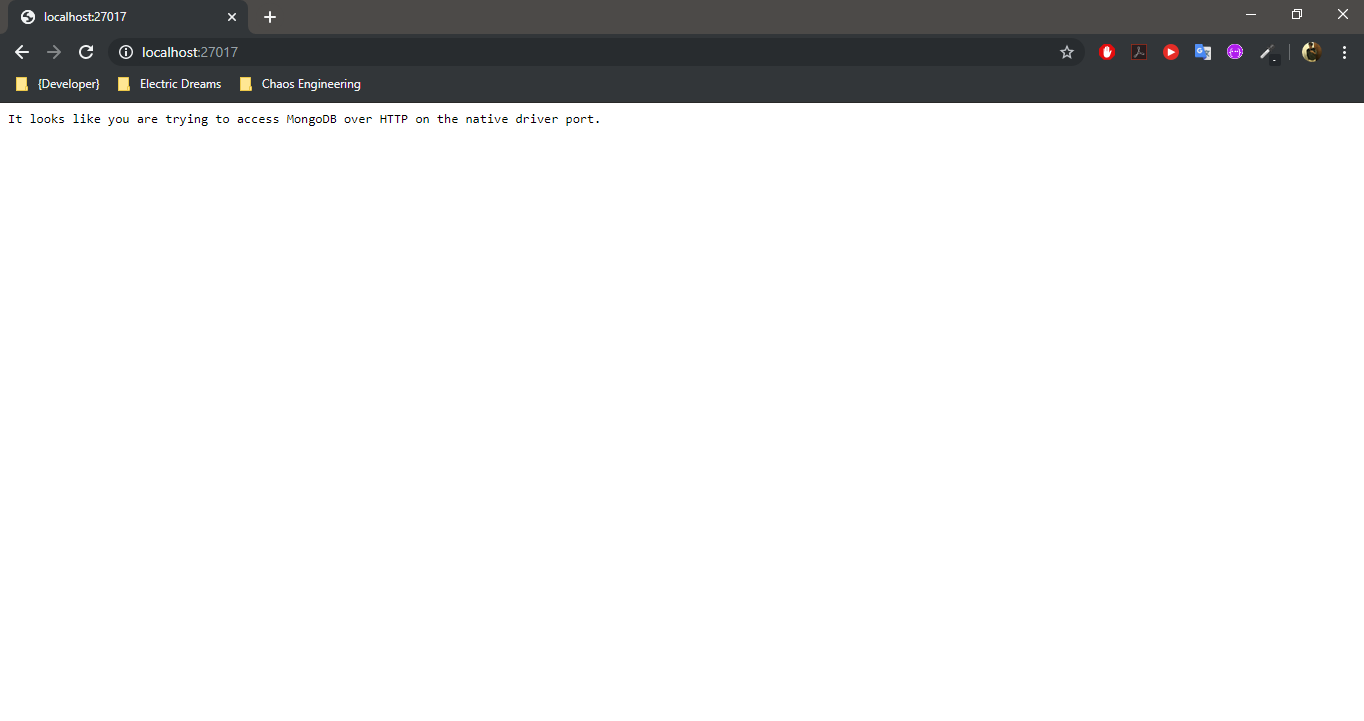


Figura 16 - Mensagem de sucesso do Docker

Para pode interagir com os termos da API, foi definido um *body*, que também serve de *schema* para o banco de dados, para padronizar a forma como os dados vão ser inseridos e editados na API e consequentemente persistidos no bando de dados. Conforme a figura 17 abaixo ilustra, os termos terão um nome, uma descrição e um *log* de registro que será definido automaticamente a cada inserção, a fim de mostrar a data e hora que cada registro foi criado.

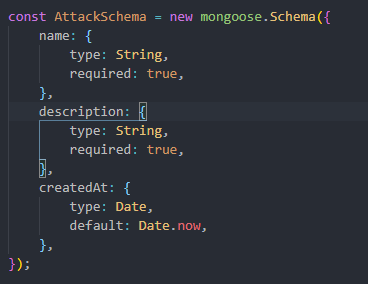


Figura 17 - Modelo de definição dos termos da API

Foram criadas rotas na API para que seja possível interagir e consumir seus recursos. Todo o contexto da aplicação como rotas, esquemas e nomenclatura de arquivos tiveram o nome “*attack*” usado como prefixo, pois ataque é algo muito comum na disciplina de *Chaos Engineering*. Portanto, por questões de preferência visual, sem afetar absolutamente em nada o funcionamento da API, foi escolhido o prefixo “*attack*” ao invés de “*terms*”, pois “*attack*” harmoniza-se melhor com a disciplina de Engenharia do Caos, enquanto “*terms*” tem uma contextualização mais genérica.

As rotas e seus respectivos verbos HTTP são as seguintes:

**GET** /attacks – rota utilizada para a listagem de termos.

**GET** /attacks/:id – rota utilizada para listar o termo que possui o ID recebido como parâmetro, ou seja, irá listar um termo específico de acordo com o ID recebido pelo usuário.

**POST** /attacks – rota utilizada para a criação de um novo termo.

**PUT** /attacks/:id – rota utilizada para alterar dados do termo que possui o ID recebido como parâmetro, ou seja, irá editar um termo específico de acordo com o ID recebido pelo usuário.

**DELETE** /attacks/:id – rota utilizada para excluir o termo que possuir o ID recebido como parâmetro, ou seja, irá excluir um termo específico de acordo com o ID recebido pelo usuário.

Usando o Insomnia, um software responsável por armazenar, organizar e executar requisições de APIs REST, é possível testar cada uma das rotas especificadas acima, permitindo assim, através do Insomnia, criar, listar, editar e excluir termos da disciplina de Engenharia do Caos.

Na figura 18 abaixo é possível analisar um exemplo de criação de termo com o Insomnia, realizando uma requisição POST para criar um novo termo.

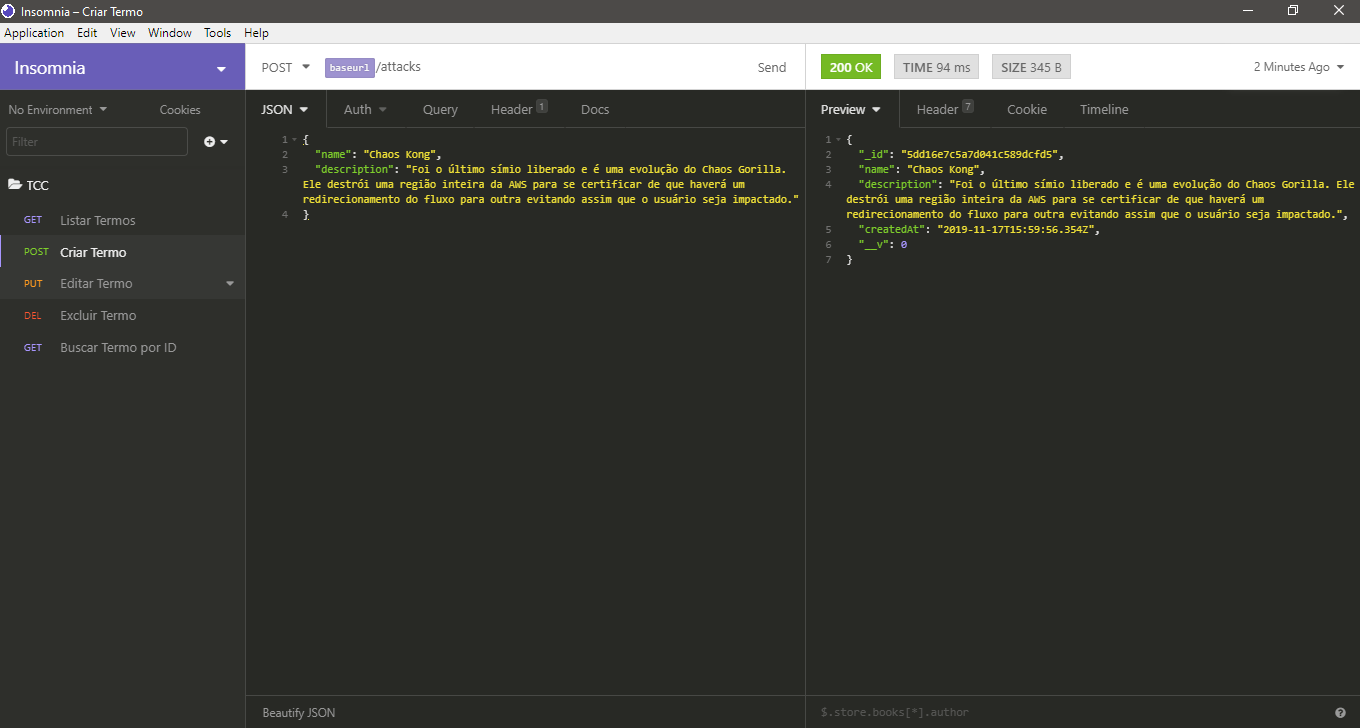


Figura 18 - Criação de termos na API com Insomnia

Na figura 19 abaixo é possível analisar, em outra requisição e desta vez um GET para listar os termos, o termo anteriormente criado sendo exibido na listagem, também utilizando o Insomnia.

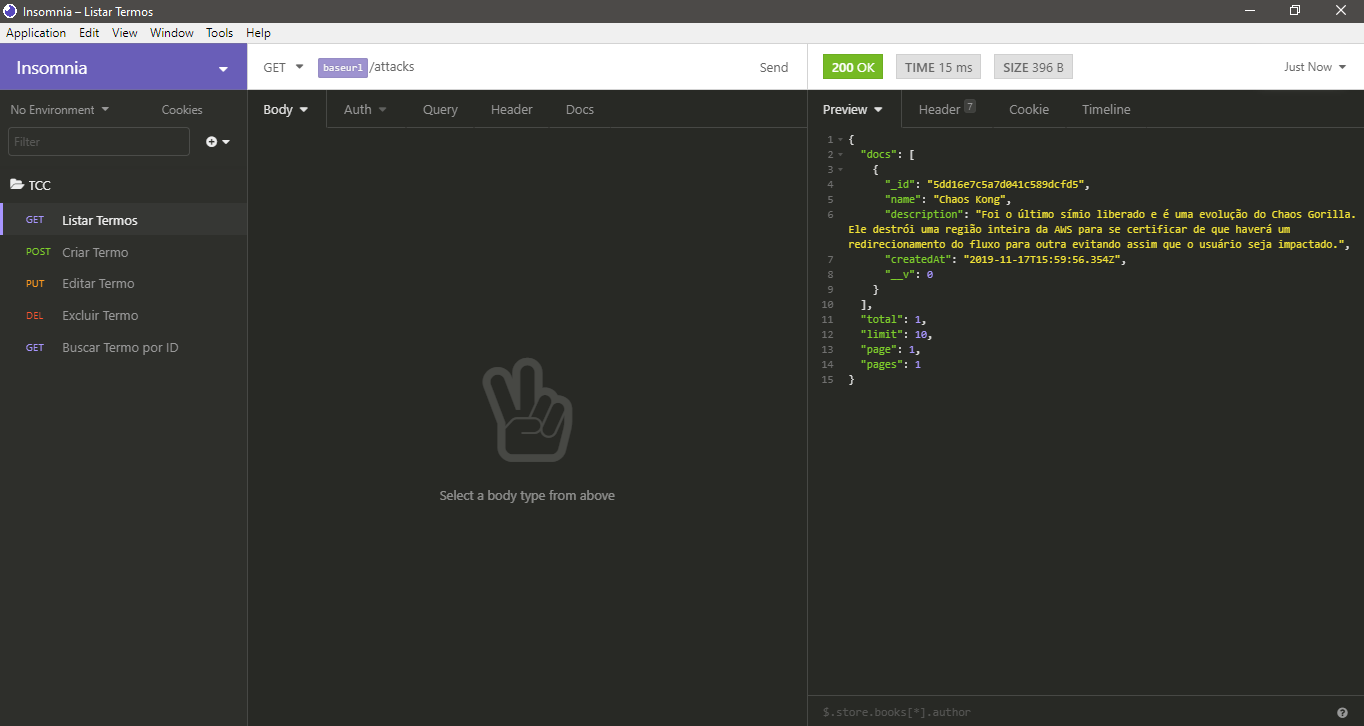


Figura 19 - Listagem de termos na API com Insomnia

É possível perceber na imagem que, com o Insomnia, também é possível editar, excluir e buscar um termo pelo ID.

Na figura 20 abaixo também é possível perceber o resultado da criação do termo com Insomnia fazendo uma requisição no navegador.

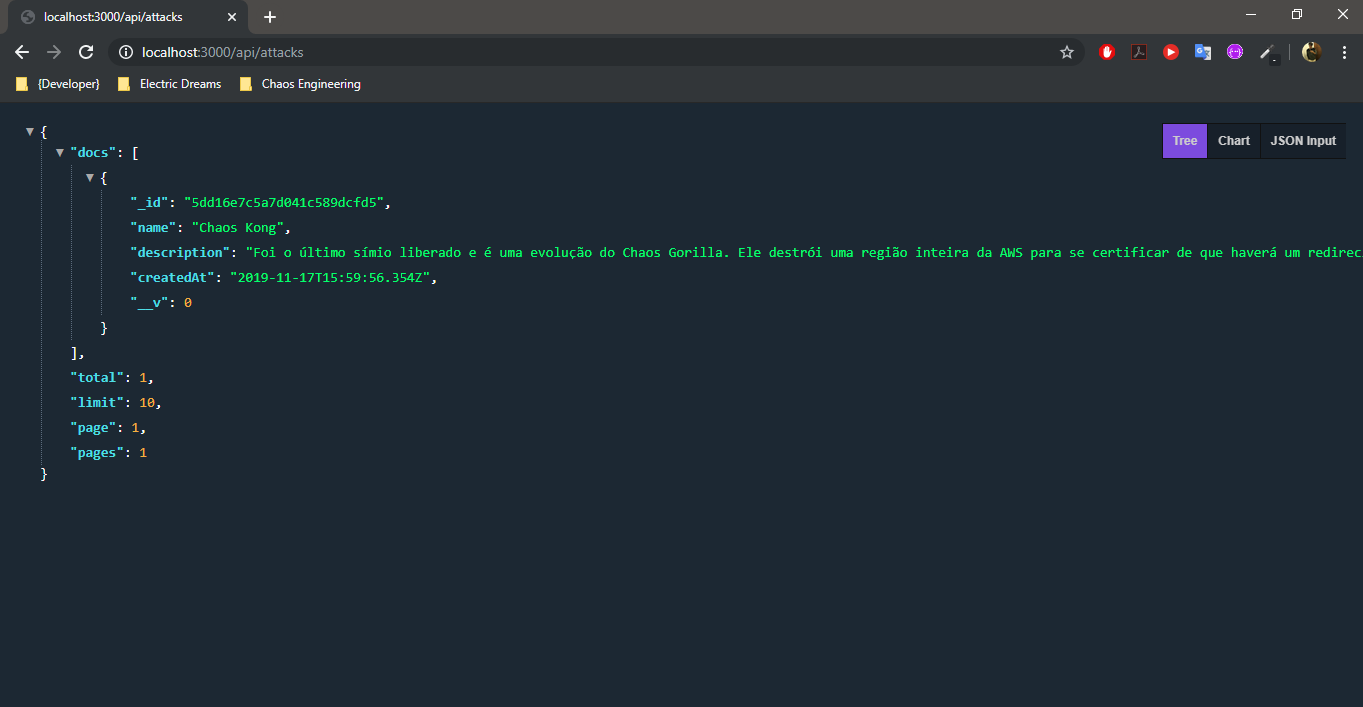


Figura 20 - Listagem de termos da API no navegador

Também é possível perceber como é o comportamento no MongoDB utilizando o software Robo3T, uma ferramenta que reproduz graficamente os dados do MongoDB, visto que toda interação feita é consequentemente replicada no banco de dados. A figura 21 abaixo mostra como os dados estão sendo persistidos no Mongo.

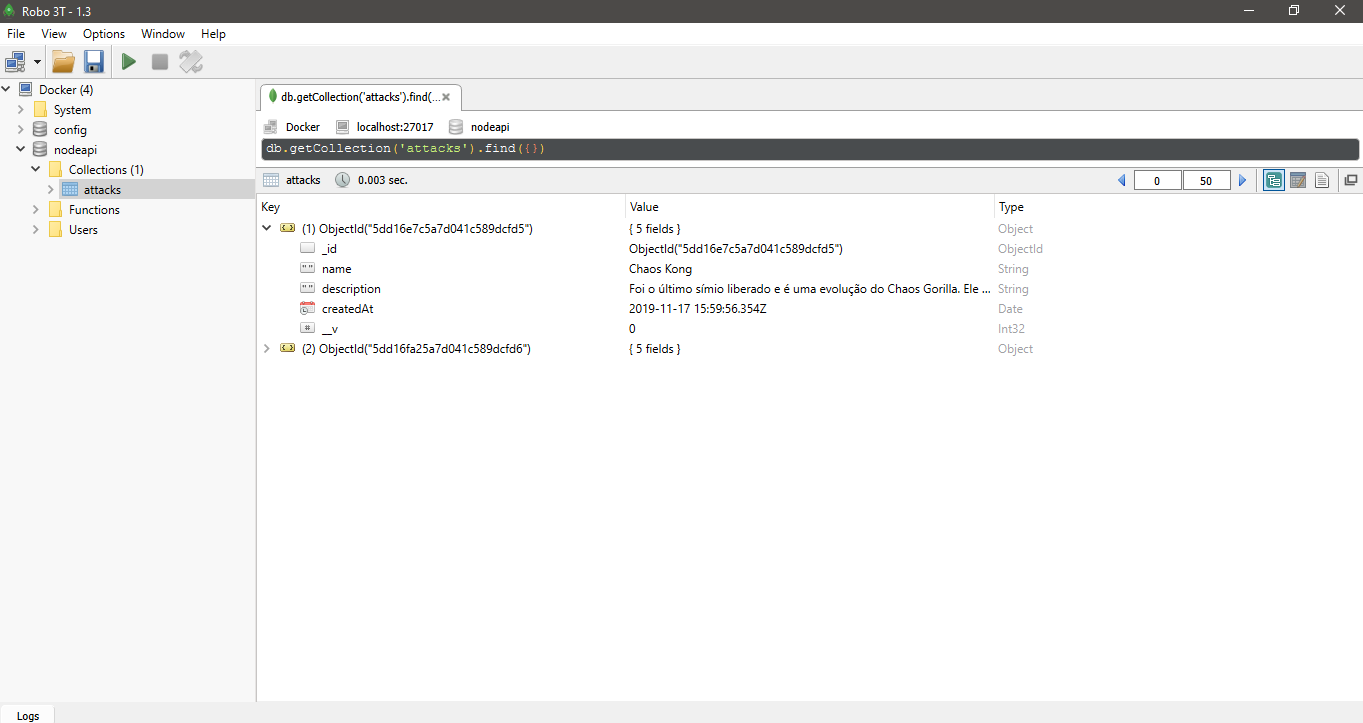


Figura 21 - Termo criado na API e persistido no MongoDB

### 3.3.2 – Contêinerização das Aplicações

A figura 22 abaixo apresenta o painel do Docker chamado Docker Hub. Nele é possível ver todos os repositórios criado a fim de armazenarem um contêiner. É possível perceber na imagem que as 4 aplicações estão listadas.

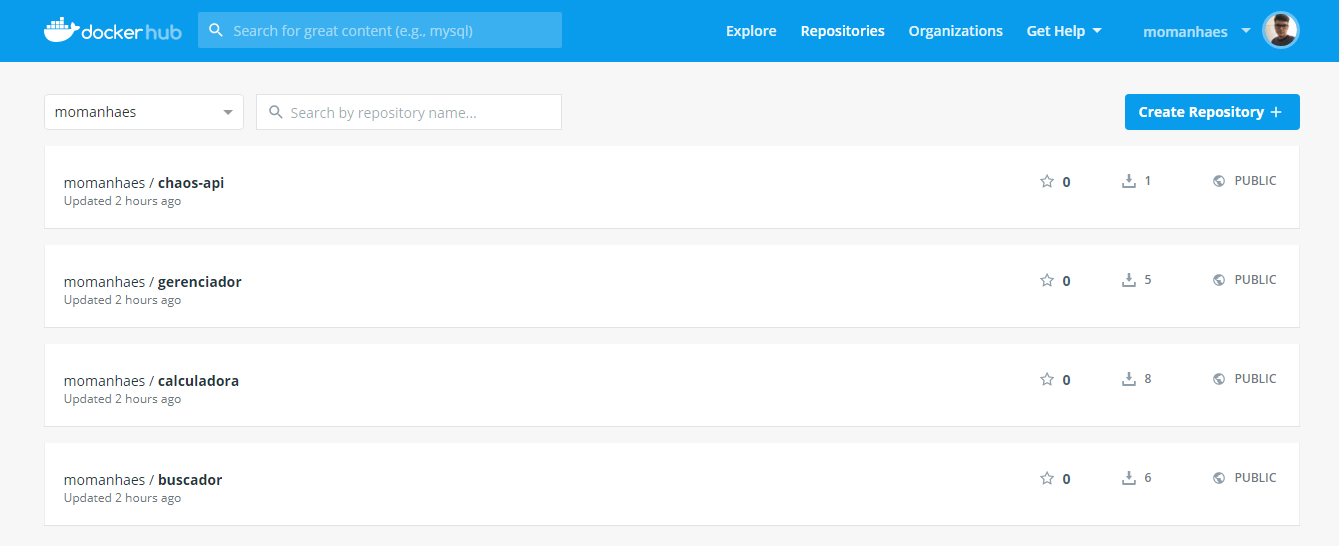


Figura 22 - Docker exibindo as aplicações do projeto

### 3.3.3 – Osquestração dos Contêineres

Utilizando o Minikube para gerar os *pods* e fazer o *deploy* das aplicações no Kubernetes, é necessário verificar se o serviço do Kubernetes está habilitado no painel do Docker Desktop. Com alguns pequenos comandos e passos de configuração é possível ter acesso ao *dashboard* do Kubernetes e verificar os *status* das aplicações, verificar como elas estão sendo servidas e qual porta deve-se usar para acessar cada uma delas. A figura 23 abaixo mostra como é exibido este *dashboard* na tela de overview do painel do Kubernetes.

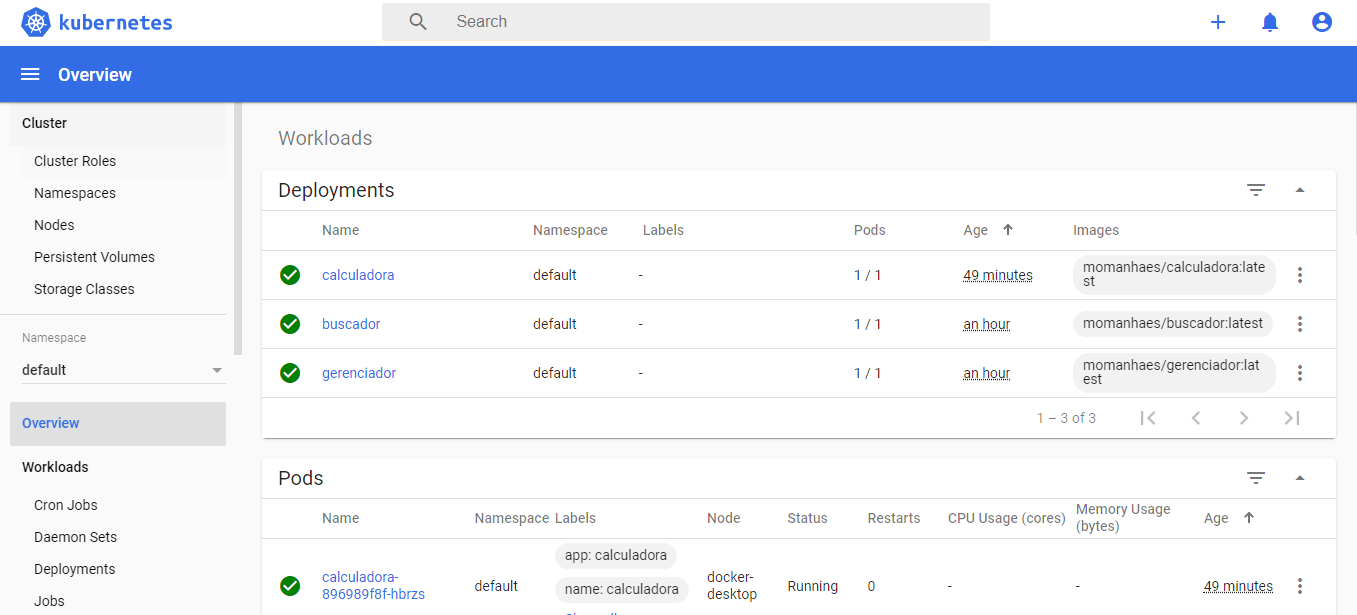


Figura 23 - Tela de overview do Kubernetes

### 3.3.4 – Deploy dos Pods

*Pod* é o nome dado pelo Kubernetes para um grupo de um ou mais contêineres que são geridos pelo seu serviço. Como boa prática de uso do Kubernetes, é necessário que seja utilizado um *pod* por aplicação, ou seja, a calculadora, o gerenciador de tarefas, o buscador de repositórios e a Chaos API terão um *pod* separado para cada, por mais que seja possível colocar as 4 aplicações em um só *pod.* Todos os 4 *pods* foram configurados nos serviços da Amazon, o AWS. O AWS faz o *deploy* dos 4 *pods* e oferece uma máquina virtual para que os 4 *pods* sejam servidos e estejam disponíveis para os usuários utilizarem, sem a necessidade de separar uma máquina física para este fim. O próprio AWS se encarrega de gerar uma URL para que seja possível acessar, através dela, as aplicações que estão nos *pods* do Kubernetes.

### 3.3.5 – Execução dos Testes

Esta seção irá descrever como testes de CPU e *Shutdown* foram configurados e executados utilizando o Gremlin.

##### 3.3.5.1 – “Alô Mundo” no Gremlin

Neste projeto o Gremlin foi a primeira ferramenta utilizada para gerar conhecimento da disciplina de Engenharia do Caos. Embora ele seja limitado em sua versão *free,* é possível realizar alguns experimentos que serão bem úteis para analisar o comportamento dos alvos do serviço do Gremlin e o processo de geração de falhas nas aplicações.

Antes de utilizarmos como alvo do Gremlin uma das aplicações criadas, ou até mesmo todas, foi resolvido reproduzir seus ataques em uma máquina virtual criada especialmente para este fim. Foi utilizado o software Oracle VirtualBox para criarmos uma máquina virtual utilizando o sistema operacional Linux em sua distribuição Ubuntu.

Realizando um cadastro no *site* do Gremlin é possível acessar o seu painel a fim de estudar a documentação da ferramenta, configurar os ataques, criar grupos de ataques e acompanhar o resultado de alguns ataques.

Utilizando um terminal e executando o comando apt-transport-https é possível fazer a instalação do Gremlin no sistema operacional Ubuntu, previamente instalado na máquina virtual criada pelo VirtualBox. É necessário executar também os comandos abaixo substituindo a palavra “usuario” pelo usuário criado no site do Gremlin e a palavra “ip\_servidor” pelo IP do servidor que se deseja realizar o ataque: ssh usuario@ip\_servidor e echo "deb https://deb.gremlin.com/ release non-free" | sudo tee /etc/apt/sources.list.d/gremlin.list.

Para importar a chave GPG é necessário executar o comando abaixo no terminal:

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys C81FC2F43A48B25808F9583BDFF170F324D41134 9CDB294B29A5B1E2E00C24C022E8EF3461A50EF6

É necessário a execução do comando sudo apt-get update && sudo apt-get install -y gremlin gremlind para instalar o *client* e o *daemon* do Gremlin.

Após ter criado uma conta no site do Gremlin, um *e-mail* de verificação será enviado com um link para configuração da mesma. Nessa configuração, será enviado um *Team ID* e uma *Secret Key* para garantir que o ambiente em que os testes estão sendo executados estão isolados. Estas duas chaves também são necessárias para adicionar novos *hosts* que serão alvos dos ataques. É importante ressaltar que cada usuário de um mesmo time possui uma *Secret Key* diferente.

Após passar por esse processo de configuração, utilizando o comando gremlin init é possível criar ataques usando o Gremlin.

###### 3.3.5.1.1 – CPU Attack

Fazendo o *login* no site Gremlin e clicando em *Create Attack* é possível ser redirecionado para o fluxo de criação de um ataque.

Na guia *Choose the Targets* deve ser escolhido o host que foi configurado para ser atacado. É possível marcar a opção *Random* para uma escolha aleatória, caso estejam cadastrados vários *hosts*.

Na guia *Choose a Gremlin* é possível escolher uma de três categorias disponíveis. São elas *Resource*, *State* e *Network*. Cada categoria possui uma lista de ataques de acordo com cada perfil. Na versão *free* (versão que está sendo utilizada) estão disponíveis dois ataques: CPU na categoria *Resource* e *Shutdown* na categoria *State*. É necessário que seja escolhido CPU da categoria *Resource*.

Na opção *Length* é possível definir quanto tempo (em segundos) a CPU será atacada. Em *Cores* é possível definir a quantidade de núcleos da CPU que serão atacados. Em *CPU Capacity* é possível definir a porcentagem de consumo de CPU em cada núcleo. Será definido 120 para *Length* e 1 para *Cores*, visto que a opção CPU *Capacity* não está disponível na versão grátis.

Na guia *Run the attack* é possível executar o ataque ou agendá-lo. É necessário clicar em *Unleash Gremlin* para executar o ataque. Após ter programado o ataque no console do Gremlin no site, é possível verificar o impacto no processador causado pelo Gremlin no Linux Ubuntu usando o comando top.

##### 3.3.5.2 – Gremlin e Kubernetes

Após realizar o experimento “Alô Mundo” na sessão anterior disparando um ataque à CPU de uma máquina virtual Linux, foi realizado o mesmo procedimento de configuração nesta seção, só que dessa vez os alvos são os *pods* que estão servindo as aplicações calculadora e buscador de repositórios. Foi criado um ataque de *shutdown,* que tem por objetivo desligar os *pods* alvos e um ataque de CPU, que tem por objetivo aumentar o consumo de CPU dos *pods* alvos. Conforme representa a figura 24 abaixo, é possível verificar no console do computador o impacto causado pelo ataque de *shutdown* nos pods usando o comando top.

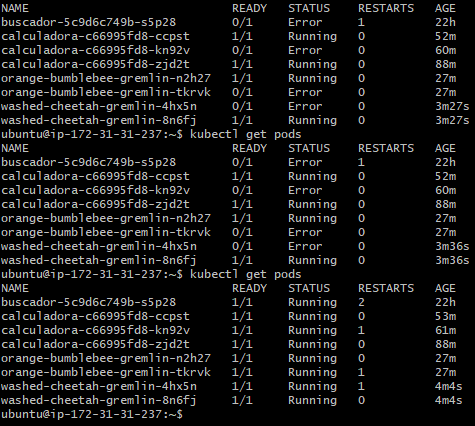


Figura 24 - Resultado do ataque de shutdown do Gremlin nos pods do Kubernetes

Analisando a figura 24 é possível verificar que os *pods* foram desligados algumas vezes e o Kubernetes se encarrou de reiniciá-los. Também é possível verificar no console do AWS, conforme ilustra a figura 25 abaixo, o impacto causado pelo ataque de CPUnos *pods*. É possível perceber que no momento em que os testes são disparados, o gráfico exibe alguns picos no consumo de CPU, revelando que o teste criado no painel do Gremlin obteve sucesso.

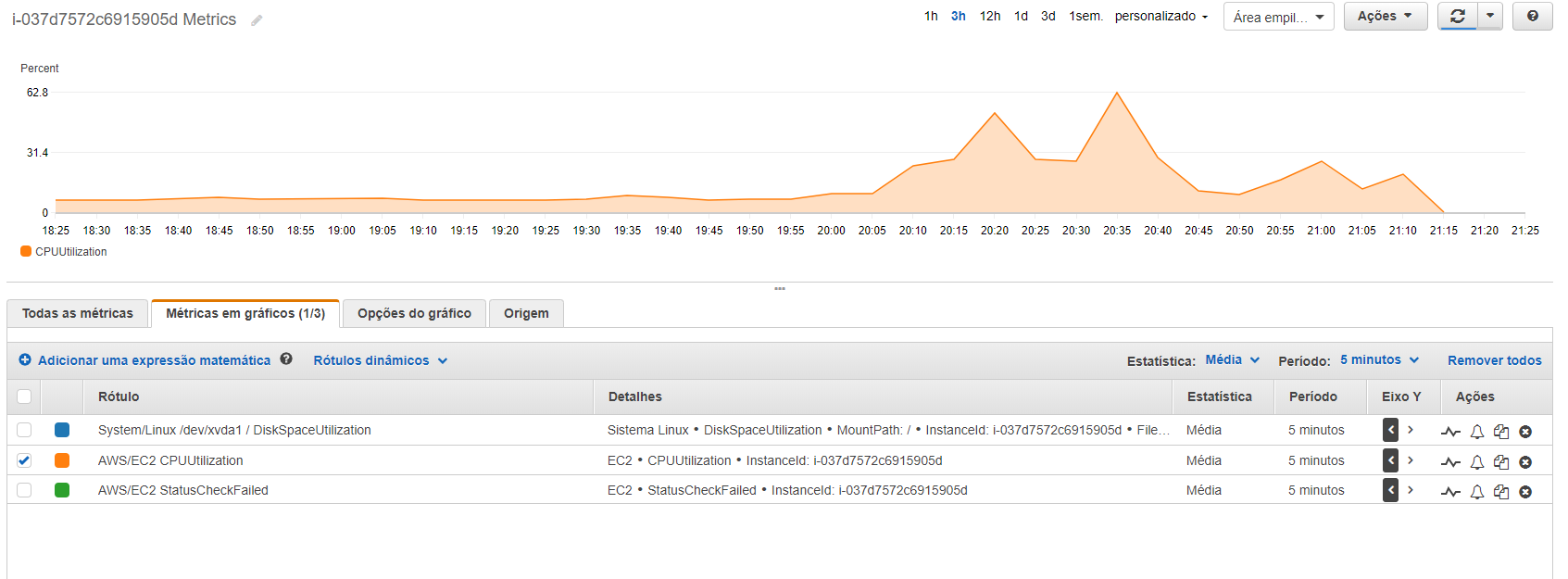


Figura 25 – Console do AWS exibindo o resultado do ataque de CPU do Gremlin

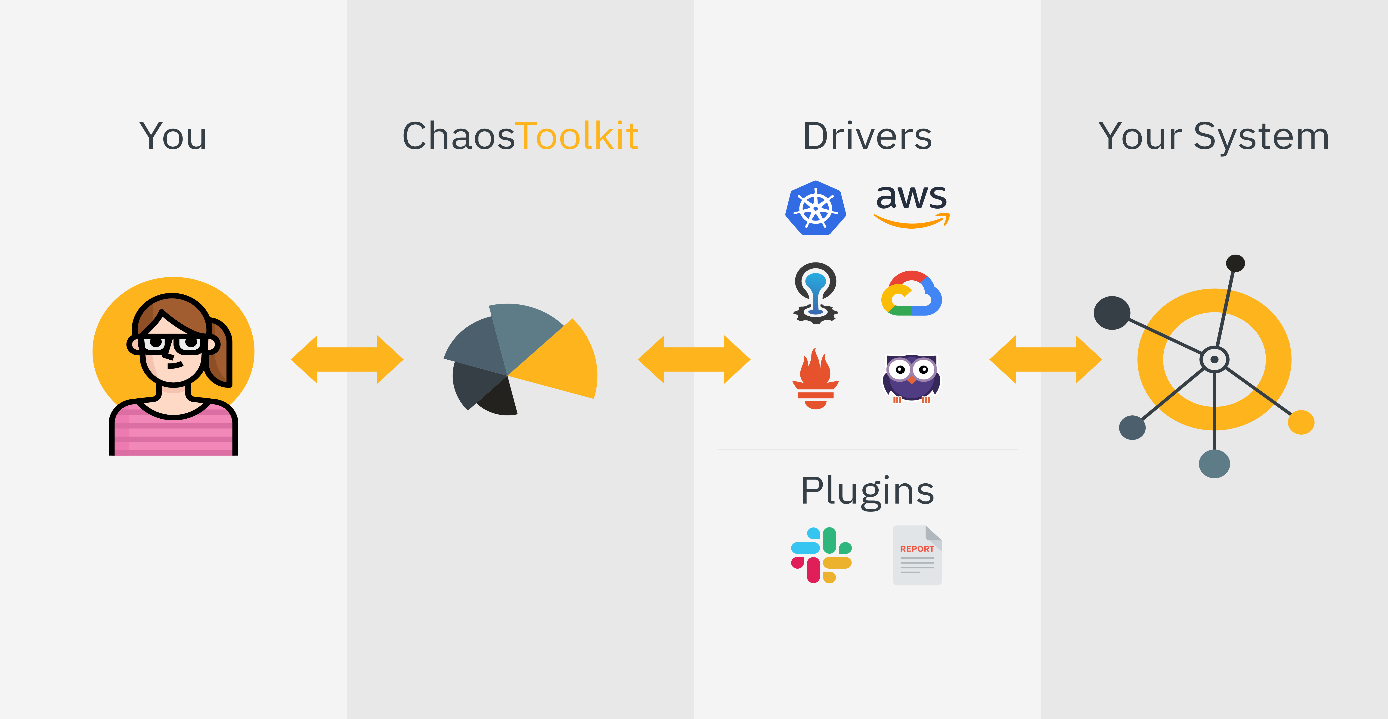
### 3.3.6 – Criação dos Testes

## 3.4 – Tecnologias Utilizadas

Nesta seção serão descritas as tecnologias Chaos Toolkit, Gremlin, HTML5, CSS3, JavaScript, NodeJS, Angular, Docker, Kubernetes Minikube e AWS.

### 3.4.1 – Chaos Toolkit

Chaos Toolkit é um framework de código aberto que disponibiliza um kit de ferramentas para execução e criação de experimentos de Chaos Engineering. O framework tem dois principais propósitos: prover uma ferramenta completa, porém simples a fim de fornecer um ponto de partida fácil para a aplicação da disciplina (*Chaos Engineering*) e disponibilizar uma API aberta para a comunidade para que qualquer experimento de Caos possa ser executado consistentemente usando integrações com outras ferramentas que estão surgindo. A figura 5 mostra como esse processo é simples.



**Figura 26 - Arquitetura do Chaos Tookit**

Fonte: Documentação do Chaos Tookit

O Chaos Toolkit visa tornar simples e direto, seguindo os Princípios da Engenharia do Caos, a execução de experimentos contra o seu sistema ativo, a fim de criar confiança em seu comportamento e aprender sobre possíveis pontos fracos.

Foi escolhida esta ferramenta para poder escrever casos de testes similares ao Gremlin. O uso do Chaos Toolkit permite a criação dos experimentos, execução dos mesmos e integração com os outros serviços como o Gremlin, Docker, Kubernetes e AWS.

### 3.4.2 – Gremlin

O Gremlin é uma ferramenta paga, mas oferece alguns recursos grátis, porém limitados. Criada a fim de “Transformar o fracasso em resiliência”, como diz em sua documentação, o Gremlin oferece uma estrutura a fim simular interrupções reais com segurança e com uma crescente biblioteca de ataques. Gremlin se encaixa na categoria FaaS, onde oferece uma plataforma para os usuários encontrarem pontos fracos em seus sistemas antes que eles possam causar problemas.

Foi escolhido o Gremlin, pois é possível realizar alguns experimentos que estão divididos pelas categorias “Resource”, “State” e “Network”. Cada categoria possui uma lista de ataques de acordo com cada perfil. Na versão grátis estão disponíveis dois ataques: CPU na categoria “Resource” e “Shutdown” na categoria “State”. Escolhendo o Gremlin como ferramenta deste trabalho é possível executar experimentos de aumento do consumo de CPU do servidor assim como desligá-lo programadamente com alguns passos de configuração.

### 3.4.3 – HTML5

HTML é sigla para HyperText Markup Language, que significa Linguagem de Marcação de Hipertexto. HTML é a linguagem de marcação padrão para páginas Web. Com ela é possível criar páginas web estáticas e dinâmicas. HTML descreve a estrutura de uma página da Web que consiste em uma série de elementos que informam ao navegador como exibir o conteúdo a quem está acessando a página.

Os elementos HTML são representados por *tags* que rotulam partes do conteúdo, como "cabeçalho", "parágrafo", "tabela" e assim por diante. Os navegadores não exibem as tags HTML, mas as usam para exibir o conteúdo da página.

Para este projeto escolhemos utilizar o HTML5 que é a versão mais atual da linguagem que possui novos elementos, atributos e comportamentos. Com ela, segundo sua documentação na página do Mozilla, é possível descrever mais precisamente o nosso conteúdo por conta da inserção das *tags* semânticas, além de permitir que páginas web armazenem dados localmente no lado do cliente (navegador)entre outras muitas funcionalidades que auxiliam e muito na rotina do programador.

### 3.4.4 – CSS3

CSS é uma sigla para Cascading Style Sheets, Folhas de Estilo em Cascata traduzindo para a Língua Portuguesa. CSS é a linguagem que define o estilo (formatação, cores, espaçamento, posição de elementos etc.) de um documento HTML, ou seja, descreve como os elementos HTML devem ser exibidos.

O estilo da página com CSS pode ser feito de várias maneiras: usando uma *tag* de estilono documento inteiro, usando uma propriedade de estilo para cada *tag* ou criando um *link* para um arquivo *.css* externo ao documento de conteúdo (HTML).

Neste projeto três aplicações utilizam páginas *web* feitas com HTML, então foi decidido utilizar CSS3 que é a versão mais atual da ferramenta junto com o framework do Twitter chamado Bootstrap, que é um conjunto de estilos previamente definidos e configurados onde podemos ter acesso usando sua documentação na internet. Optamos por utilizar CSS3 com Bootstrap pelo fato de ser uma biblioteca muito utilizada em muitos sites renomados e por possuir estilos criativos e funcionais.

### 3.4.5 – JavaScript

Uma linguagem de programação que surgiu em 1995 que tinha como objetivo ser linguagem de programação para o lado do cliente (navegador), algo que nos últimos anos vem sendo resignificado desde o surgimento do Node.js, um interpretador javascript para o lado do servidor.

JavaScript é uma linguagem multi-paradigma de tipagem dinâmica que não faz uso de compilador por ser uma linguagem interpretada, portanto a maioria de seus erros podem apenas ser descobertos em tempo de execução.

Foi o escolhido o JavaScript como linguagem de programação, pois faz a integração perfeita com as tecnologias HTML5 e CSS3. JavaScript é a linguagem mais utilizada na web para fazer a interação dos elementos HTML e estilo CSS como botões, barras de navegações, fundos animados e muitas outras possibilidades.

### 3.4.6 – Node.js

É um interpretador de código que tem como propósito interpretar o JavaScript no lado do servidor, coisa que só era possível no lado do cliente (navegador). Isso é possível hoje, pois Ryan Dahl, engenheiro por trás do Node.js, utilizou o motor *open-source* do Google Chrome, chamado V8, no lado do servidor, assim sendo capaz de interpretar código JavaScript não somente no lado do cliente, como era até 2009.

Foi o escolhido o Node.js como tecnologia para o desenvolvimento da API de termos úteis de Chaos Engineering pela sua simplicidade para “subir” um servidor e, como a lógica desta API é algo simples, ou seja, somente um CRUD de termos úteis com ID, título e descrição, foi escolhido o Node.js, junto com a biblioteca Express e o banco de dados não relacional MongoDB para construção.

### 3.4.7 – Angular

Angular é um framework usado para construção da interface de aplicações Web usando HTML, CSS e, principalmente JavaScript. O framework é mantido pela Google e a sua primeira versão chamada AngularJS foi totalmente reescrita e passou, a partir da versão 2, a ser chamado somente de “Angular”. A partir da versão 2 fizeram um framework totalmente novo. Entre as muitas mudanças do AngularJS para as versões superiores, destaca-se a linguagem padrão de desenvolvimento do Angular que passou a ser o TypeScript, um superconjunto da Microsoft que insere tipagem no JavaScript.

Para este projeto foi escolhido o uso do Angular para o desenvolvimento da calculadora e do gerenciador de tarefas, pois é um framework mantido por uma empresa de alta credibilidade, além de ser um framework excelente para criação de SPAs, ou Single Page Applications.

### 3.4.8 – Docker

O Docker é uma plataforma aberta de desenvolvimento, envio e execução de aplicações. Fornece a capacidade de empacotar e executar um aplicativo em um ambiente pouco isolado chamado contêiner. O isolamento e a segurança permitem executar muitos contêineres simultaneamente em um determinado host.

Foi escolhido o Docker para este projeto, pois com o Docker é possível criar imagens (containers prontos para *deploy*) a partir de arquivos de definição chamados Dockerfiles com facilidade, além do Docker utilizar como *backend* *default* o LXC, sendo possível definir limitações de recursos por container (memória, CPU e I/O, por exemplo). Sendo assim, o Docker “caiu como uma luva” para que pudéssemos “subir” as quatro aplicações em contêineres, visto que também seria fácil a integração com o Kubernetes e o AWS, que serão tratados nos próximos tópicos.

### 3.4.9 – Kubernetes Minikube

O Kubernetes, também chamado de k8s, é uma plataforma de código aberto que automatiza as operações dos containers Linux. Essa plataforma elimina grande parte dos processos manuais necessários para implantar e escalar as aplicações em containers. O Kubernetes oferece um serviço para que os *pods* (grupo de um ou mais contêineres) sejam facilmente gerenciados.

Já o Minikube é uma ferramenta que facilita a execução local do Kubernetes. O Minikube executa um cluster Kubernetes de nó único dentro de uma Máquina Virtual (VM).

Esse projeto faz uso do Kubernetes com Minikube, pois ele tem a responsabilidade de gerenciar as aplicações com facilidade e eficiência. O Kubernetes foi feito para gerenciar *clusters* que podem incluir hosts em *clouds* públicas, privadas ou híbridas. Por isso, o Kubernetes é a plataforma ideal para hospedar aplicações nativas em *cloud* que exigem escalabilidade rápida, algo que é essencial para o projeto, visto que serão realizados testes e experimentos de Chaos Engineering, onde será necessário, por exemplo, “subir” serviços que foram derrubados, analisar indisponibilidade dos serviços e configurar redirecionamento de fluxo.

### 3.4.10 – AWS

Amazon Web Services, também conhecido como AWS, é uma plataforma de serviços de computação em nuvem. É a maior empresa fornecedora de serviços em nuvem, oferecendo cerca de 165 serviços completos incluindo série de aplicativos, incluindo computação, armazenamento, bancos de dados, redes, análises, machine learning e inteligência artificial (IA), Internet das Coisas (IoT), segurança e desenvolvimento, implantação e gerenciamento de aplicativos.

Foi escolhido o uso do AWS para este projeto para não precisar fazer o *setup* e o *deploy* de toda a configuração e recursos do projeto em uma máquina física. Seria extremamente complicado ter que deixar um computador disponível e ligado para que pudéssemos realizar as atividades deste trabalho, além de termos a possibilidade de analisarmos os gráficos de monitoramento do serviço da Amazon, visto que o AWS oferece serviços *cloud* de maneira excepcional, onde grandes empresas utilizam de seus recursos, além de oferecer serviços grátis com fins experimentais.

# 4 – CONCLUSÃO

## 4.1 – Considerações Finais

## 4.2 – Contribuições

## 4.3 – Trabalhos Futuros

# REFERÊNCIAS BIBLIOGRÁFICAS

**Links Acessados**

Chaos Toolkit. **Chaos Tookit Site.** Disponível em: <https://docs.chaostoolkit.org/>. Acesso em: 15 de novembro de 2019.

GitHub Chaos Toolkit. **Chaos Tookit Documentation.** Disponível em: <https://github.com/chaostoolkit/chaostoolkit/>. Acesso em: 15 de novembro de 2019.

Gremlin. **Gremlin Documentation.** Disponível em: <https://www.gremlin.com/docs/>. Acesso em: 15 de novembro de 2019.

W3Schools. **HTML Tutorial.** Disponível em:

<https://www.w3schools.com/html/>. Acesso em: 15 de novembro de 2019.

MDN Web Docs. **HTML5 Documentation.** Disponível em:

<https://developer.mozilla.org/pt-BR/docs/Web/HTML/HTML5/>. Acesso em: 15 de novembro de 2019.

W3Schools. **CSS Tutorial.** Disponível em:

<https://www.w3schools.com/css/>. Acesso em: 15 de novembro de 2019.

Bootstrap. **Bootstrap Documentation.** Disponível em:

<https://getbootstrap.com/docs/4.3/getting-started/introduction/>. Acesso em: 15 de novembro de 2019.

MDN Web Docs. **Documentação JavaScript.** Disponível em:

<https://developer.mozilla.org/pt-BR/docs/Aprender/JavaScript/>. Acesso em: 15 de novembro de 2019.

Node.js. **Documentação Node.js.** Disponível em:

<https://nodejs.org/pt-br/about/>. Acesso em: 15 de novembro de 2019.

Angular. **Documentação Angular.** Disponível em:

<https://angular.io/docs/>. Acesso em: 16 de novembro de 2019.

Docker. **Documentação Docker.** Disponível em:

<https://docs.docker.com/>. Acesso em: 16 de novembro de 2019.

Mundo Docker. **O que é Docker?** Disponível em: <https://www.mundodocker.com.br/o-que-e-docker/>. Acesso em: 16 de novembro de 2019.

Kubernetes Concepts. **O que é um pod?** Disponível em: <https://kubernetes.io/docs/concepts/workloads/pods/pod/>. Acesso em: 16 de novembro de 2019.

Red Hat. **O que é Kubernetes?** Disponível em: <https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes/>. Acesso em: 16 de novembro de 2019.

Kubernetes. **Site Oficial.** Disponível em: < https://kubernetes.io/pt/docs/home/>. Acesso em: 16 de novembro de 2019.

Kubernetes. **Getting Started – Installing Kubernetes with Minikube.** Disponível em: < https://kubernetes.io/docs/setup/learning-environment/minikube/>. Acesso em: 16 de novembro de 2019.

AWS. **Documentação da AWS.** Disponível em: <https://docs.aws.amazon.com/index.html?nc2=h\_ql\_doc\_do\_v />. Acesso em: 16 de novembro de 2019.

Site Oficial do AWS. **O que é AWS?** Disponível em: <https://aws.amazon.com/pt/what-is-aws/>. Acesso em: 16 de novembro de 2019.

# ANEXO I – Glossário