**Farmalá – API para Pesquisa de Valores de Medicamentos**

**Francisco Q. Rodrigues1, Nei C. pertuis1, Guilherme R. Silva1**

1Universidade Do Grande Rio (Unigranrio)  
Rua Professor José de Souza Herdy, 1160 - Jardim Vinte e Cinco de Agosto, Duque de Caxias - RJ, 25071-202

{fquintas, Nei.pertuis, guilhermer}@unigranrio.br

***Abstract.*** *Nowadays, with more and more attributions and tasks to be carried out every day, and with the familiarization of technological means, many people are looking for tools that facilitate the accomplishment of their tasks, through technology. One of the mandatory tasks for all people, whether routinely or punctually, is the purchase of medication. If we analyze the world ranking of drug consumption, Brazil is in 6th place (Guiadafarmacia, 2020, Inovafarma, 2020, Infograficos, 2020). Farmalá aims to serve this audience, with the availability of the research tool so that developers can create their applications, without the need for all the backend development.*

***Resumo.*** *Atualmente, com cada vez mais atribuições e tarefas a serem realizadas a cada dia, e com a familiarização dos meios tecnológicos, muitas pessoas buscam por ferramentas que facilitem a realização de suas tarefas, através da tecnologia. Uma das tarefas obrigatórias a todas as pessoas, seja rotineira ou pontualmente, é a compra de medicamentos. Se analisarmos o ranking mundial de consumo de medicamentos, o Brasil encontra-se em 6º lugar (Guiadafarmacia, 2020, Inovafarma, 2020, Infograficos, 2020). O Farmalá visa atender a esse público, com uma disponibilização da ferramenta de pesquisa para que desenvolvedores possam criar seus aplicativos, sem a necessidade de todo o desenvolvimento de backend.*

**1. Introdução**

Atualmente, com a atribuição de cada vez mais tarefas diárias, a população precisa de ferramentas facilitadoras, seja em caráter de gestão de suas atividades corporativas, bem como de tarefas pessoais. Com maior acessibilidade e familiarização com os recursos tecnológicos, muito utilizam ferramentas para auxilio a incontáveis atividades. Uma das tarefas que todo cidadão vez por outra precisará realizar é a compra de medicamentos, seja em caráter frequente, devido a tratamentos de saúde, bem como pontualmente ou em caráter de urgência, devido a uma enfermidade específica que surja. Se analisarmos o ranking mundial de consumos de medicamentos, o Brasil encontra-se em 6º lugar (Guiadafarmacia,2020, Inovafarma, 2020, Infograficos, 2020).

Assim, nota-se uma demanda no que se refere a ferramentas que possibilitem auxilio na busca de medicamentos por farmácias e realização de comparação entre os valores disponibilizados; a fim de agilizar a tarefa de consulta, que por vezes é realizada presencialmente através de visitas a diversas unidades de Drogarias, ou por telefone.

O projeto Farmalá visa atender a demanda apresentada; realizando a buscar por medicamentos solicitados e realização de disponibilização de valores para comparação.

Assim como o site de buscas buscapé e zoom (buscapé, 2021, zoom, 2021), que se propõem a fazer a busca por produtos com a opção de exibir uma lista com os menores preços, existem endereços web que disponibilizam a busca por medicamentos, como farmagora (farmagora, 2020) e consultaremedios (consultaremedios, 2020), contudo, não apresentam a disponibilização de unidades de drogarias próximas aos municípios de Duque de Caxias e Rio de Janeiro.

**2. Sistema**

O projeto consiste na criação de uma API que possibilite a busca por medicamentos em uma lista de farmácias disponíveis em seu cadastro, disponibilizando os valores, possibilitando a pesquisa por farmácias mais próximas e disponibilizando link de fácil acesso para compra pelo endereço web das Drogarias.

Em relação ao funcionamento da API em questão, a mesma recebe um arquivo JSON com o nome do remédio desejado e realiza a busca de 3 funções, verificando se a informação sobre o medicamento a ser buscado já consta no banco de dados. Caso as informações já estejam disponíveis, as mesmas serão disponibilizadas. Nos casos em que as informações sobre determinado medicamento não estejam disponíveis no Banco de Dados, será realizada uma busca nos endereços cadastrados, referentes as farmácias específicas, por meio de suas disponibilizações sobre valores na web.

Os dados sobre os valores são por meio de Web Scraping, passando assim a ocorrer atualizações por referência, em relação aos preços cadastrados e armazenados no Banco de Dados.

As informações solicitadas a respeito de determinado medicamento são exibidas para os solicitantes com as referências de valores e farmácia relacionada.

**3. Fundamentação teórica**

Esta seção introduz os conceitos inerentes aWeb Services REST, a prática de API scraping, assim como, técnicas de paralelismo e automação pelo selenium

**3.1. Web Services RESTful**

Web Services REST são aplicações Web construídas sob os princípios de design REST (Representational State Transfer), apresentados na tese de doutorado de Fielding (2000), que descrevem como uma aplicação Web deve responder as requisições de seus clientes, com o objetivo de promover a escalabilidade e robustez de sistemas hypermedia. Diz-se que uma aplicação é RESTful quando cumpre todas as regras (constraints) estabelecidas pelo padrão REST, sendo elas: Interface uniforme, Arquitetura cliente-servidor, Stateless, Cacheabilidade e Sistema em camadas. Por definição, Web Services têm como objetivo prover funcionalidades (ou recursos) a um conjunto de clientes [Booth et al., 2004]. Não obstante, recursos são a base da abstração de informações no estilo de arquitetura REST, sendo que qualquer informação nomeável pode ser considerada um recurso, como: imagens, arquivos ou textos. A identificação de um recurso é feita através de URIs (Uniform Resource Identifiers). No caso de APIs REST acessíveis via Web, URIs são tipicamente URLs (Uniform Resource Locators), chamados também de endpoints. Cada resposta de um servidor RESTful contém uma lista de URIs de recursos acessíveis, indicando quais requisições podem ser feitas pelo cliente a partir daquele momento. Esse comportamento é definido pela constraint conhecida como HATEOAS (Hypermedia As The Engine Of Application State), uma das constraints centrais da arquitetura REST.

**3.2. API Scraping**

API scraping é uma variação do termo Web scraping, utilizado na literatura para definir uma técnica de software para extrair informações de Websites, através da simulação do processo humano de exploração de páginas Web [Vargiu and Urru, 2013]. Segundo Mitchell (2014), trata-se de uma prática tão antiga quanto a própria Internet. As informações extraídas são utilizadas para diversos fins, como agregação de conteúdo e análise de propagandas, dentre outros tipos de processamentos de dados possíveis. Usualmente, essas informações são reunidas em uma nova página Web [Vargiu and Urru, 2013]. No que tange APIs Web especificamente, Jawad (2017) define Web API scraping como o ato de coletar um conjunto substancial de dados de uma API Web sem o consentimento dos fornecedores dessa API. Aplicações que realizem scraping são conhecidas como scraperse funcionam através da implementação de clientes autônomos (bots) das fontes de informação exploradas. A construção de um scraper consiste na engenharia reversa da API Web, a fim de identificar os endpoints que proveem as informações desejadas e quais os mecanismos envolvidos na obtenção da informação, como parâmetros aceitos pela API, tokens solicitados, entre outros. Os conhecimentos obtidos são embutidos em um bot que fará o scraping de fato. Com a popularização das APIs Web, surgiram diversas soluções para a realização de API Scraping [Lospinoso, 2017; Zaslavskiy, 2019; Mitchell, 2014], incluindo soluções pagas [Scraper API, 2019], com o objetivo de facilitar o processo de extração de informações, reduzir o tempo de desenvolvimento de bot se evadir as contramedidas existentes. As contramedidas existentes dividem-se em duas abordagens possíveis: a limitação da quantidade de requisições que um cliente pode fazer a um Web Service em um espaço de tempo (throttling) e a detecção de clientes anômalos. A primeira abordagem pode ser utilizada como estratégia para minimizar os efeitos dos bots é garantir que o Web Service não seja sobrecarregado. Ao atingir o limite máximo pré-estabelecido de chamadas, o cliente é obrigado a aguardar uma janela de tempo definida pelo fornecedor da aplicação para voltar a realizar chamadas e ser atendido [Chandramouli, 2019]. Para a implementação do throttling, é necessário definir duas políticas: como vai se dar a identificação de cada cliente e a política de limitação de acesso ao recurso. Simpson (2019) analisou as políticas de throttling de três grandes mantenedoras de APIs (GitHub, LinkedIn e Bitly) para identificar as práticas adotadas por cada uma. Como resultado, o autor detectou que os três serviços adotam a identificação de clientes por meio de tokens (API Keys). A limitação do máximo de requisições que um cliente pode fazer, no entanto, é feita por hora, para a primeira empresa, por dia, para a segunda, e por minuto, hora e mês, para a terceira. Apesar do uso de tokens ser recomendado como uma medida de controle de acesso a recursos, especialmente contra o abuso de APIs Web, segundo a OWASP, não se pode depender exclusivamente dessa tecnologia, uma vez que tokens podem ser comprometidos com relativa facilidade [OWASP, 2019].Quanto a detecção de bots, Jawad (2017) afirma que pode ser difícil distinguir clientes legítimos de ilegítimos através dos padrões de comportamento, uma vez que podem existir diferentes tipos de clientes legítimos consumindo APIs Web de diferentes formas(navegador, aplicação, mobile, IoT, dentre outros). A autora definiu empiricamente diversas características comportamentais que podem ser utilizadas para identificar um API scraper, como: taxa de requisições constante ou com pequena variação por um longo período de tempo, alta taxa de requisições por segundo, modificações sistemáticas na URI em requisições contínuas, dentre outras. Essas características foram empregadas em três algoritmos de aprendizagem de máquina supervisionado para verificar se era possível detectar bots de API scraping realizando obtenções agressivas de dados. Embora capaz de detectar comportamentos anômalos de clientes, a solução proposta por Jawad (2017) requer a obtenção de conjuntos de dados das requisições dos clientes de determinado Web Service para o treinamento e validação dos algoritmos de aprendizagem de máquina, uma vez que vários padrões de comportamento são inerentes ao serviço em questão. Ademais, quando ocorrerem modificações em endpoints (adição ou exclusão) é necessário realizar um novo treinamento.

**3.3. Paralelismo**

Paralelismo é sobre a execução paralela de tarefas, ou seja, mais de uma por vez (de forma simultânea), a depender da quantidade de núcleos (cores) do processador. Quanto mais núcleos, mais tarefas paralelas podem ser executadas. É uma forma de distribuir processamento em mais de um núcleo.

Também podemos dizer paralelismo é uma forma de atingir concorrência e que cada linha de execução paralela também é concorrente, pois os núcleos estarão sendo disputados por várias outras linhas de execução e quem gerencia o que o núcleo vai executar em dado momento do tempo é o escalonador de processos. Paralelismo implica concorrência, mas o contrário não é verdadeiro, pois é possível ter concorrência sem paralelismo, é só pensar no caso de uso de um único thread gerenciando milhares de tarefas, pausando e resumindo-as, esse é um modelo de concorrência sem paralelismo. Já o scheduler de corrotinas da linguagem Go é um exemplo de scheduler multi threaded, ele paraleliza a execução das corrotinas, ou seja, é um modelo de alta concorrência que faz uso dos núcleos do processador para paralelizar as execuções. [Kennedy Tedesco, 2021]

**3.4. Selenium**

Selenium é um conjunto de ferramentas de código aberto multiplataforma, usado para testar aplicações web pelo browser de forma automatizada. Ele executa testes de funcionalidades da aplicação web e testes de compatibilidade entre browser e plataformas diferentes. O Selenium suporta diversas linguagens de programação, como por exemplo C#, Java e Python, e vários navegadores web como o Chrome e o Firefox. (Marylene Guedes, 2021)

**3.4.1 Selenium WebDriver**

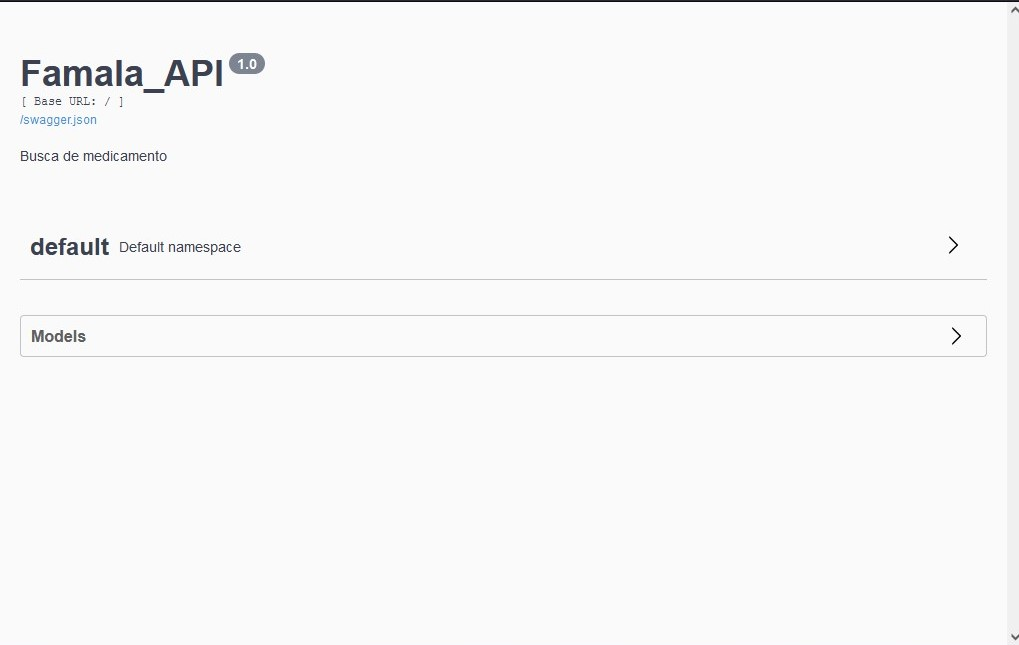
O WebDriver conduz um navegador nativamente, como um usuário faria, localmente ou em uma máquina remota usando o servidor Selenium, marca um salto em termos de automação do navegador. (selenium, 2021)

**4. Figuras e Legendas**

A seguir serão mostradas algumas figuras das telas da documentação da API

**Figura 1.**

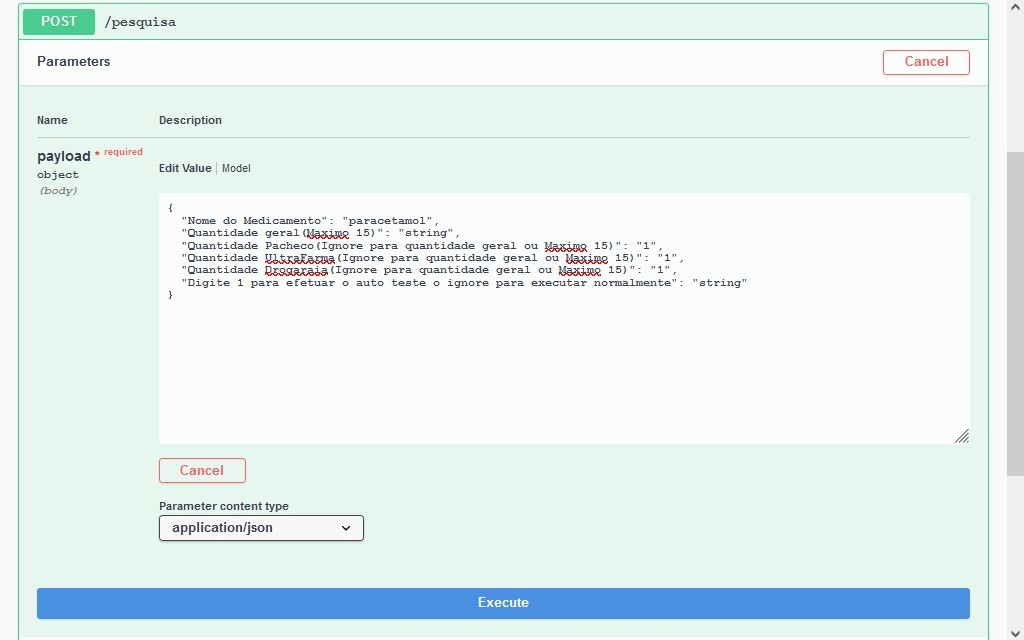
Primeiramente acessamos a tela de documentação que possui as explicação dos métodos utilizados na API e informações sobre como devem ser passados os parâmetros para o funcionamento correto da API.



**Figura 1 – Inicio do uso da API (Documentação Swagger)**

**Figura 2.**

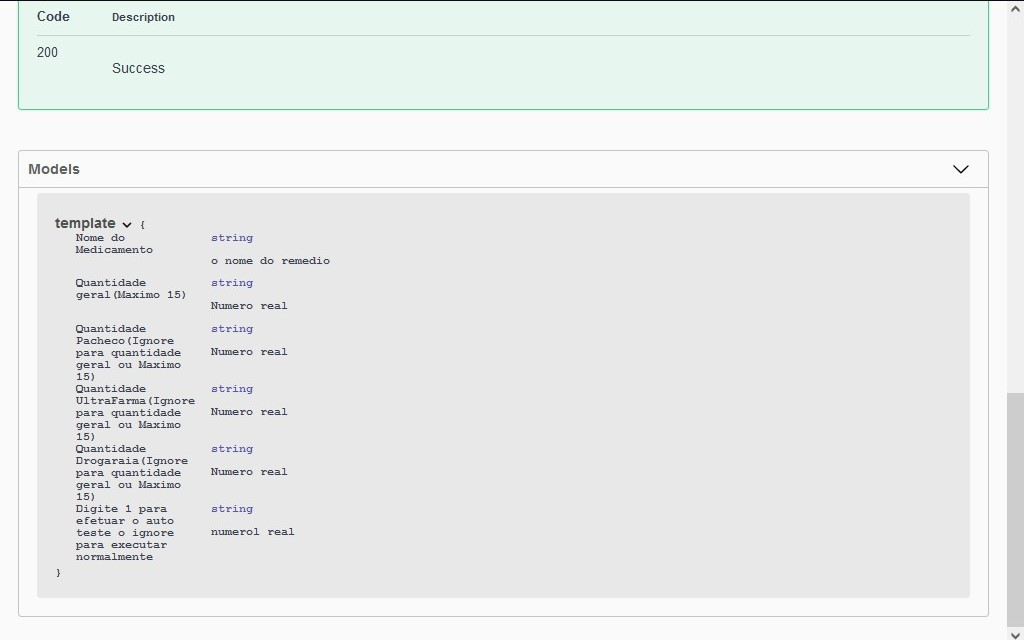
Tela da documentação Swagger com o uso do método POST para pesquisa de farmácias com a utilização correta da passagem de valores por Json.

**

**Figura 2 – Passagem dos Parâmetros para a Execução da API**

**Figura 3.**

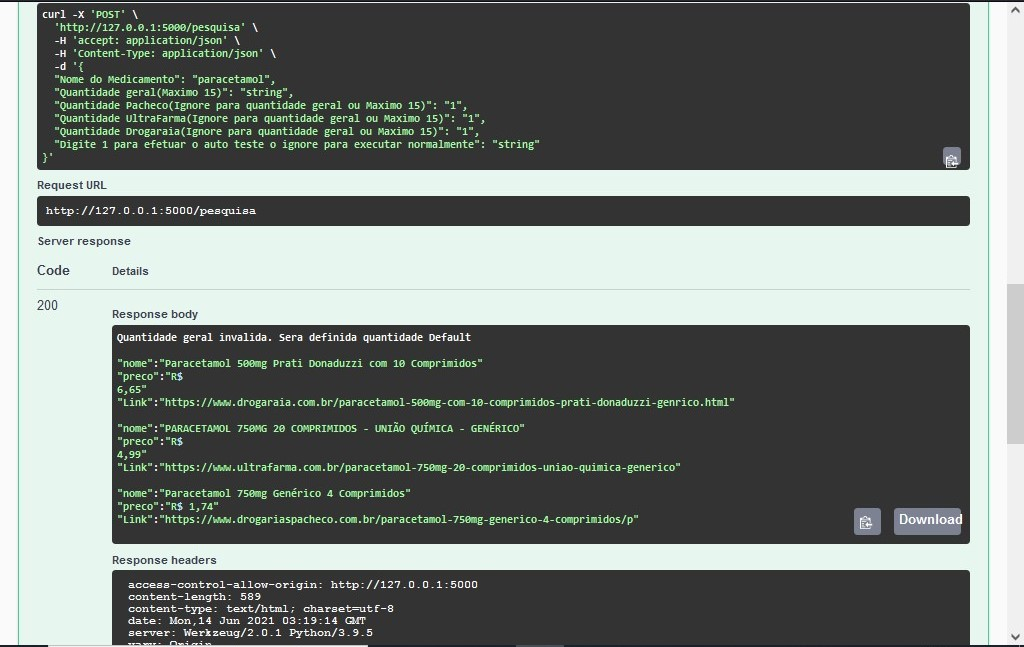
Tela de apresentação do modelo que deve ser usado para a passagem dos parâmetros em Json, com o nome e tipo do parâmetro.

**

**Figura 3 – Dados Necessários para Funcionamento Correto da API**

**Figura 4.**

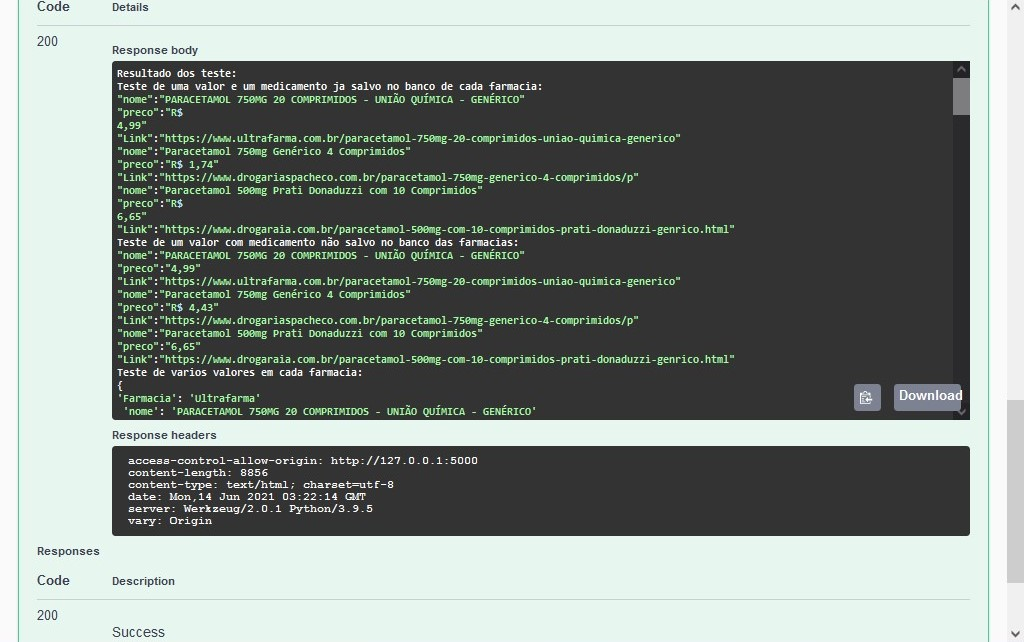
Tela de com todas as informações relacionadas a comunicação entre o servidor e o cliente

**

**Figura 4 – Resposta Retornada pelo Servidor**

**Figura 5.**

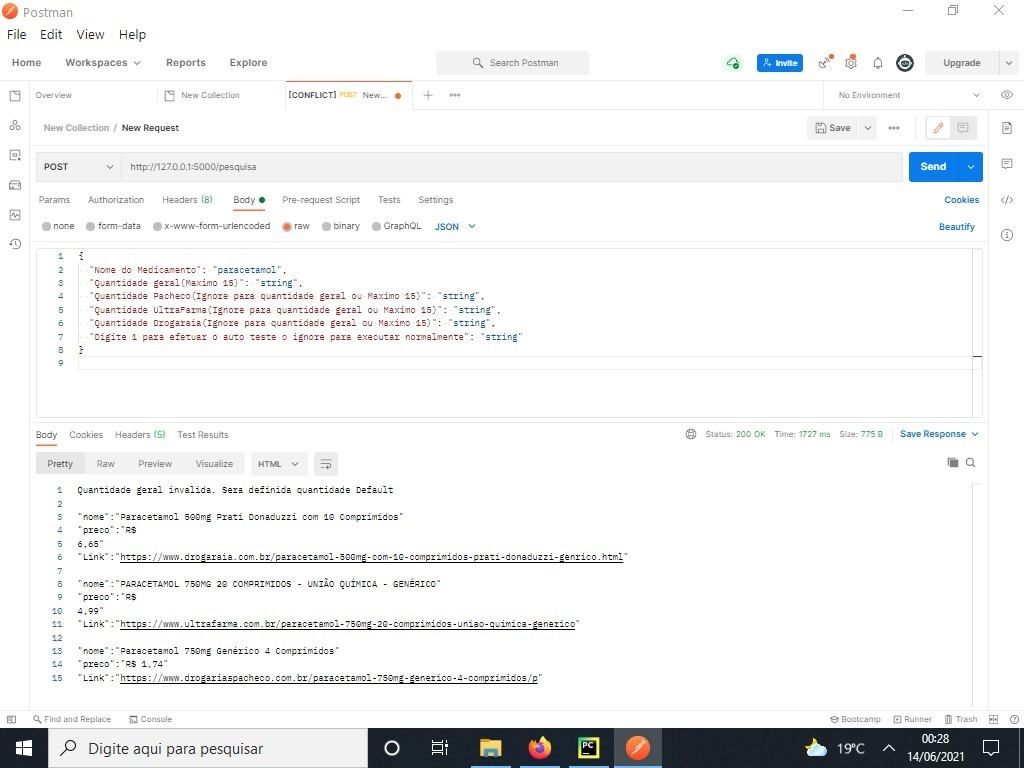
Execução do código automatizado que efetua todos os teste de funcionamento da API

**

**Figura 5 – Resposta Retornada pelo Servidor (Teste)**

**Figura 6.**

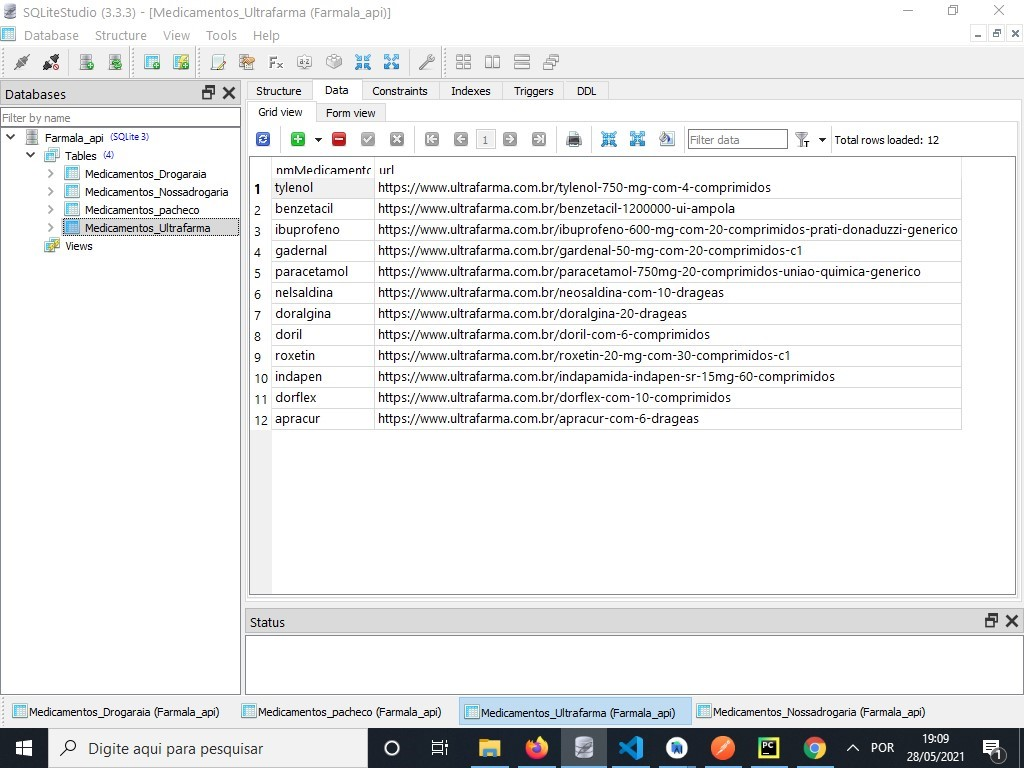
Tela do software Postman, software utilizado para fazer as requisições para o servidor, pesquisa passando parâmetros e a resposta do servidor.

**

**Figura 6 – Execução e Resposta pelo Software Postman**

**Figura 7.**

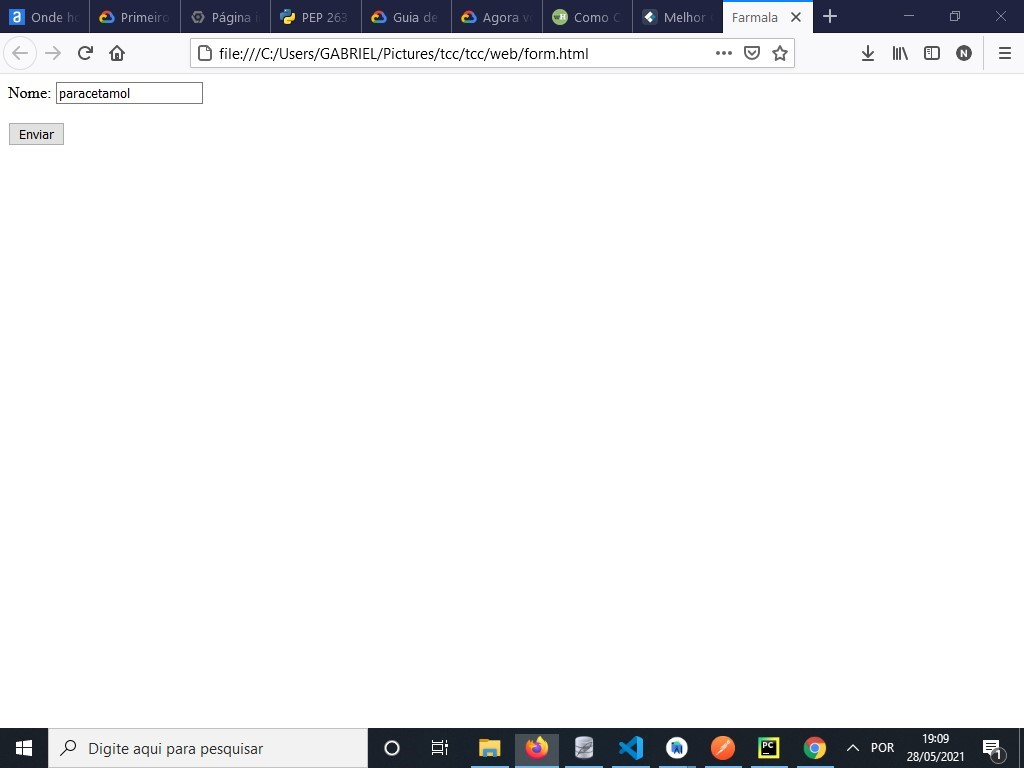
Tela do banco de dados, tabela da farmácia Ultrafarma.

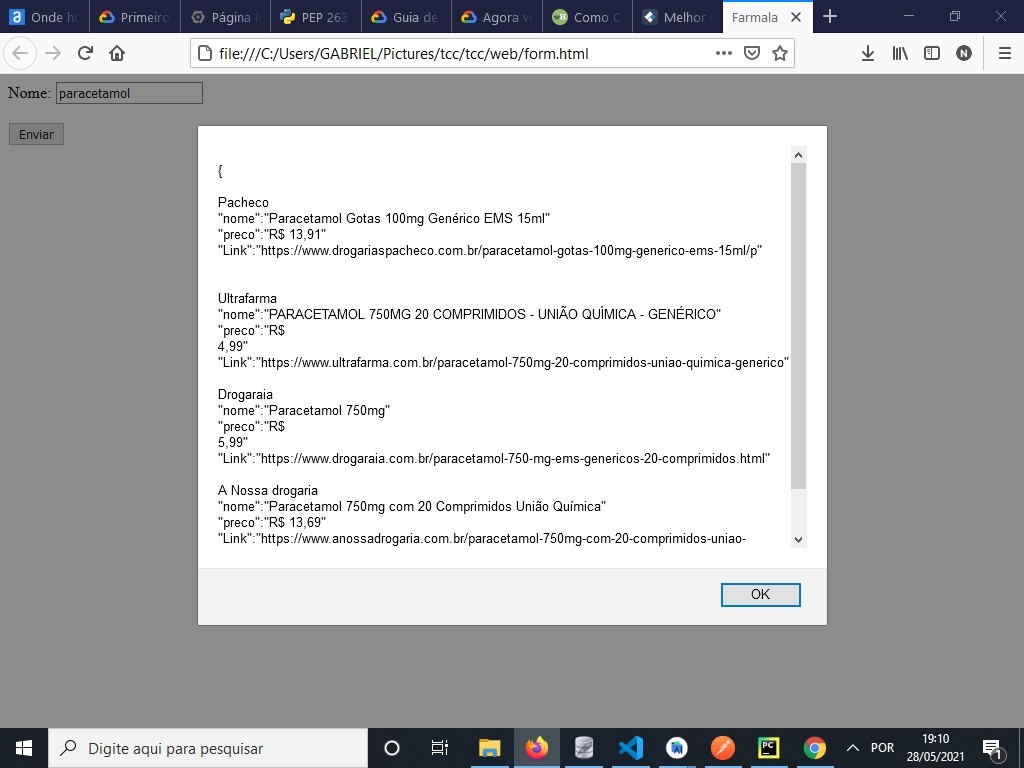


**Figura 7 – Banco de Dados**

**Figura 8 e 9.**

Exemplo de consumo da API numa página simples em html e JavaScript.

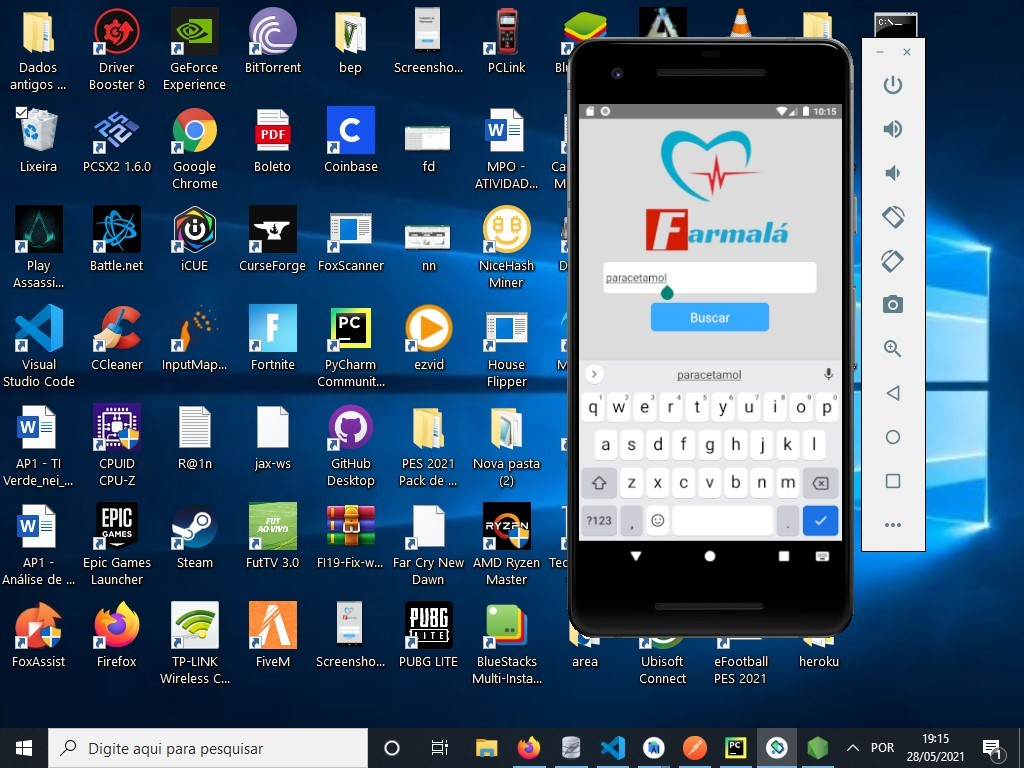
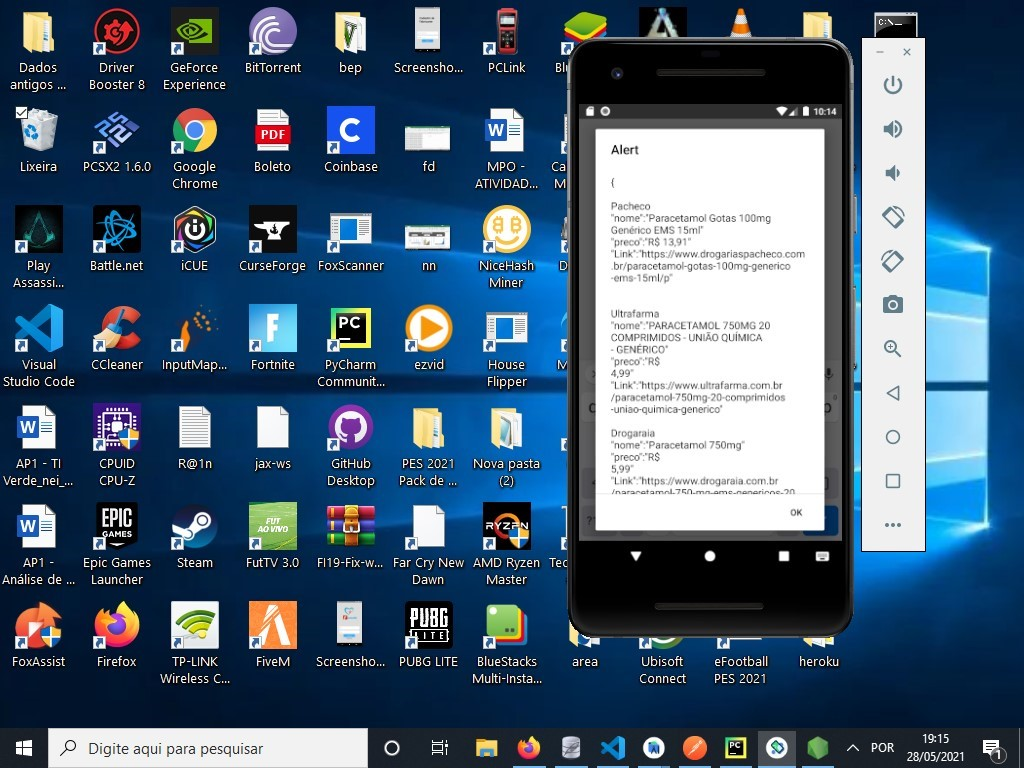




**Figuras 8 e 9– Código Criado em Java Script e HTML para Consumo e Exibição dos Valores Retornados pela API**

**Figuras 10 e 11.**

Exemplos de tela de um app criado em React Native para consumo da API e a exibição dos valores retornados.



**Figuras 10 e 11 – Código Criado em React Native para Consumo e Exibição dos Valores Retornados pela API**

**6. Considerações Finais**

O projeto proposto ressalta o anseio por solucionar as necessidades da população no que tange a pesquisa e indústria farmacêutica, possibilitando melhor qualidade de vida, aliada as novas tecnologias.

Desejamos que com a criação desta API, mais desenvolvedores se habilitem a implementação de aplicativos e integrações para a disponibilização ao público, por meio de dispositivos diversos.

Acreditamos que a iniciativa trará benefícios a comunidade, podendo também haver parcerias dos setores públicos e privados, para aprimoramentos.

**Referências**

Mitchell, R. (2014). Web Scraping with Python. O’Reilly.

Zaslavskiy, A. (2019). API Scraping in the Real World. [Online;https://www.codementor.io/blog/api-scraping-5fq1gtd4ah].

Lospinoso, J. (2017).Abrade, a high-throughput web API scraper.[On-line;https://lospi.net/cpp/developing/software/2017/09/15/abrade-web-scraper.html].

Scraper API (2019). Scraper API. [Online;https://www.scraperapi.com/].

Chandramouli, R. (2019). Security Strategies for Microservices-based Ap-plication Systems.[Online;https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204-draft.pdf].

Simpson, J. (2019). Everything You Need To Know About API RateLimiting|NordicAPIs|.[Online;https://nordicapis.com/everything-you-need-to-know-about-api-rate-limiting/].

OWASP (2019). REST Security Cheat Sheet - OWASP. [Online;https://www.owasp.org/index.php/REST\\_Security\\_Cheat\\_Sheet].

Jawad, D. (2017). Detection of Web API Content Scraping An Empirical Study of Machine Learning Algorithms.[Online;http://www.nada.kth.se/~ann/exjobb/dina\\_jawad.pdf].

Vargiu, E. and Urru, M. (2013). Exploiting web scraping in a collaborative filtering-based approach to web advertising. Artificial Intelligence Research2

Booth, D., Haas, H. and Mccabe, F. (2004). Web Services Architecture.

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based SoftwareArchitectures. University of California, Irvine.

Treinaweb. Disponível em:<https://www.treinaweb.com.br/blog/concorrencia-paralelismo-processos-threads-programacao-sincrona-e-assincrona/> Acessado em: 15-06-2021

Treinaweb. Disponível em:< https://www.treinaweb.com.br/blog/o-que-e-selenium /> Acessado em: 15-06-2021

Treinaweb. Disponível em:< <https://www.selenium.dev/documentation/en/webdriver/>

/> Acessado em: 15-06-2021