# Testando APIs REST com Postman e Rest Assured: Um Relato de Experiência com o Sistema EducAPI

**Fernando Ribeiro de Souza**

Departamento de Ciências Exatas (DCX) –
Universidade Federal da Paraíba (UFPB) - Campus IV
Cep 58297-000 – Rio Tinto – PB – Brasil

fernando.ribeiro@dcx.ufpb.br

***Abstract.*** *Developing tests for REST APIs is not an easy task, specially for developers starting in the area of automated tests. Besides, it is challenging to develop tests that are easy to maintain and to understand. In this context, this work focuses on the problem of implementing tests with quality for a system that provides a REST API. Faced with this problem, this article aims to describe the experience in developing tests for the EducAPI System using Postman and Rest Assured tools and the lessons that were learned through this process.*

***Resumo.*** *Desenvolver testes para APIs REST não é uma tarefa fácil, principalmente para desenvolvedores iniciando na área de automatização de testes. Além disso, um desafio é desenvolver testes que sejam fáceis de se manter e que sejam fáceis de se compreender. Nesse contexto, este trabalho se foca na problemática de implementar testes com qualidade para um sistema que disponibiliza uma API REST. Diante dessa problemática, este artigo visa descrever a experiência obtida durante a implementação de testes para o Sistema EducAPI utilizando as ferramentas Postman e Rest Assured e as lições que foram aprendidas nesse processo..*

## 1. Introdução

O processo de desenvolvimento de software envolve uma série de atividades. Apesar das técnicas, métodos e ferramentas nelas empregados, erros no produto ainda podem ocorrer [Barbosa 2000]. Segundo Barbosa (2000), dentre essas técnicas está a atividade de testes, que [1]consiste de uma análise dinâmica do produto e é uma atividade relevante para a identificação e eliminação de erros que persistem.

Conforme destaca Pressman (2016), o software é testado com o objetivo de revelar erros que foram cometidos inadvertidamente durante o seu projeto e construção. Então, uma das formas para se ter uma garantia de qualidade de software seria integrar os testes ao desenvolvimento do software desde seu início, e ajudar os desenvolvedores a desenvolverem um software com mais qualidade e a encontrar desde o início os erros que viriam a afetar as funcionalidades do sistema.

Conforme destacam Bernardo e Kon (2008), "a execução de testes manuais de um caso de teste pode ser rápida e efetiva, mas a execução e repetição de vasto conjunto de testes

[1] "Trabalho de conclusão de curso, sob orientação do professor <Ayla Rebouças> submetido ao Curso de Licenciatura em Ciência da Computação do Centro de Ciências Aplicadas e Educação (CCAE) da Universidade Federal da Paraíba, como parte dos requisitos necessários para obtenção do grau de LICENCIADO EM CIÊNCIA DA COMPUTAÇÃO."

manualmente é uma tarefa muito dispendiosa e cansativa. Essa forma de testes acaba trazendo prejuízo para as equipes de desenvolvimento que perdem muito tempo para identificar e corrigir os erros e também prejuízo para o cliente que, entre outros problemas, sofre com constantes atrasos nos prazos e com a entrega de um software de qualidade duvidosa." Os testes automatizados podem ser mais eficazes e de baixo custo de implementação e manutenção e funcionam como um bom mecanismo para controlar a qualidade do sistema [Bernardo 2011].

Os autores Bernardo e Kon (2008) afirmam que "a grande vantagem dos testes automatizados é que todos os casos de teste podem ser facilmente e rapidamente repetidos a qualquer momento e com pouco esforço". Estes autores também reforçam que "a reprodutibilidade dos testes permite simular identicamente e inúmeras vezes situações específicas, garantindo que passos importantes não sejam ignorados por falha humana e facilitando a identificação de um possível comportamento não desejado".

Em geral, ao testar um software, busca-se executar o programa ou modelo utilizando algumas entradas em particular e verificar se seu comportamento está de acordo com o esperado [Bertolino 2007, Delamaro 2007]. Testes têm o intuito de melhorar a qualidade do software que está sendo testado. Um bom planejamento dessas atividades pode significar economia para um projeto, visto que a identificação de defeitos no início do ciclo de desenvolvimento do produto pode reduzir os custos da sua correção, além de sua confiabilidade [Pontes 2009].

No entanto, o uso de testes pode ser bem custoso. Considerando esse aspecto, o emprego de testes automatizados contribui significativamente com a redução de custos e tempo de projeto durante o processo de desenvolvimento [Myers et al. 2004]. Segundo Rafi et al. (2012), alguns dos benefícios do teste automatizado relatados na literatura são: aumento da qualidade do produto final, alto índice de cobertura, redução do tempo de teste, aumento da confiança, diminuição de esforços humanos e redução de custos.

O foco deste trabalho será em testes de APIs REST. Uma API (*Application Programming Interface*) ou Interface de Programação de Aplicações é o conjunto de rotinas, padrões e instruções de programação que permite que os desenvolvedores criem aplicações que possam acessar determinado serviço na internet [Dos Santos 2013]. APIs formam um componente integral do ecossistema de software [Manikas 2016]. Esses ecossistemas de software se tornaram uma forma ideal de construir grandes soluções de software em cima de uma plataforma de tecnologia comum [Ofoeda 2019].

REST (*Representational State Transfer*) é um estilo de arquitetura de software que define um conjunto de restrições a serem usadas para a criação de serviços web (Web Services). Os serviços web que estão em conformidade com o estilo arquitetural REST, fornecem interoperabilidade entre sistemas de computadores na internet. Os serviços web RESTful permitem que os sistemas solicitantes acessem e manipulem representações textuais de recursos da Web usando um conjunto uniforme e predefinido de operações sem estado [Wikipédia].

Segundo Masse (2011), uma API expõe um conjunto de dados e funções para facilitar as interações entre programas de computador e permitir a troca de informações. A API REST é a interface de um serviço web, ouvindo e respondendo diretamente às requisições de clientes, ela consiste em um conjunto de recursos interligados. Quando se está produzindo uma API, é fundamental a realização de testes.

Podemos dizer que os testes destas APIs são tão importantes como qualquer outro tipo de teste de software. Através de testes podemos ver se o que foi desenvolvido segue o que foi

proposto, ou seja, se as funcionalidades atendem às expectativas dos clientes, e se requisitos como confiabilidade, desempenho (*performance*) e segurança são atendidos.

Algumas das ferramentas que podem ser usadas para testes de API são o Postman[2] e o Rest Assured[3].

Diante deste contexto, este trabalho teve como objetivo desenvolver testes para a API Rest do Sistema EducAPI [Ludgério et al. 2021] e identificar os passos para realizar esses testes utilizando as ferramentas Postman e Rest Assured e as principais lições aprendidas com base na experiência obtida. Foram desenvolvidos testes para a API REST do sistema EducAPI [Ludgério et al. 2021], um sistema construído utilizando o framework Spring.

Para isso, será descrito o processo de testes utilizando o Postman e utilizando o Rest Assured e serão mostrados os principais passos para construir testes da API do sistema EducAPI . As demais seções deste artigo estão organizadas conforme descrito a seguir. Na Seção 2 serão apresentados conceitos e ferramentas utilizados neste trabalho de forma a facilitar a sua compreensão. A Seção 3 apresenta a metodologia utilizada no trabalho. A Seção 4 apresenta as soluções que foram desenvolvidas utilizando as ferramentas REST Assured e Postman para a realização dos testes. Na Seção 5 serão apresentadas as principais lições aprendidas com a experiência. Por fim, a Seção 6 apresenta as conclusões e propostas de trabalhos futuros.

[2] "Ferramenta Postman -Postman API Platform" https://www.postman.com/ -Accessado em 15 set.. 2021.
[3] "Ferramenta REST Assured - REST Assured" https://rest-assured.io/ - Acessado em 15 set.. 2021.

## 2. Fundamentação Teórica

Nessa seção serão apresentados conceitos e ferramentas utilizados neste trabalho de forma a facilitar sua compreensão.

### 2.1 Testes de API

Existem duas questões básicas associadas aos testes de softwares: (1) custo; e (2) tempo. Em relação ao custo, pode-se afirmar que o emprego do teste traz a necessidade de contratação de equipes especializadas (testadores e projetistas de teste) e aquisição de licenças de ferramentas específicas, elevando significativamente os custos de desenvolvimento, dependendo da complexidade do sistema em desenvolvimento [Myers et al. 2004]. Tal acréscimo do tempo de projeto é consequência do não uso de técnicas e critérios de testes adequados, da reexecução dos dados de teste após cada correção de inconsistências detectadas e, principalmente, da falta de automatização [Bertolino, 2007].

Considerando que qualidade é um fator essencial no desenvolvimento de software, muito se tem investido em pesquisas na área de teste de software [Delamaro et al. 2007b]. Pesquisadores procuram definir técnicas, critérios e ferramentas que possibilitem a aplicação de tais atividades de maneira sistemática, com alta qualidade e custo reduzido.

O teste automatizado de software é um processo no qual as ferramentas de software executam testes pré-programados em uma ferramenta de software antes de serem liberados para produção. O objetivo do teste automatizado é simplificar ao máximo o esforço de teste e isso pode se dar utilizando scripts que possam ser rapidamente e até automaticamente executados [Tecnisys].

Existem duas abordagens para os testes de software: uma utiliza a técnica de teste estrutural; a outra, a técnica de teste funcional. A técnica estrutural recebe o nome de teste de caixa-branca. Isso acontece porque nela é necessário o conhecimento da lógica interna do sistema, para se desenvolverem os casos de testes [Perry, 2006]. Nesta técnica, os casos de teste são construídos a partir do código fonte do programa [Chen; Poon 2004]. A técnica de teste funcional é conhecida como técnica de teste de caixa-preta. Ela recebe esse nome porque não é necessário nenhum conhecimento da lógica interna do sistema para se construírem os casos de testes [Perry 2006]. Testes caixa-preta visam verificar a funcionalidade e a aderência aos requisitos, em uma ótica externa ou do usuário, sem se basear em qualquer conhecimento do código e da lógica interna do componente testado [Delamaro 2013]. Nessa técnica, os casos de teste são construídos a partir de uma especificação do programa [Chen; Poon 2004].

Há vários níveis de testes. Segundo o ISTQB (*International Software Testing Qualifications Board*), os níveis de teste são grupos de atividades de teste que são organizados e gerenciados juntos. Cada nível de teste é uma instância do processo de teste. Alguns desses níveis são Testes de integração, unidade, sistema, aceitação, etc.

Para este trabalho nos concentraremos em Testes de Sistema, que são testes onde o sistema já está completamente integrado e onde são verificados seus requisitos em um ambiente similar ao ambiente de produção. O objetivo de um teste de sistema é avaliar as especificações do sistema de ponta a ponta. Os testes de sistema podem ser baseados em especificações de riscos e/ou de requisitos, processos de negócios, casos de uso, dentre outras descrições de alto nível do comportamento, interações e recursos do sistema. Em particular, este trabalho tem como foco testes de sistema em APIs REST. Uma API é criada quando uma empresa de software tem a intenção de que outros criadores de software desenvolvam produtos associados ao seu serviço [CanalTech].

APIs são como pontes integradoras de softwares, representando um acordo entre duas partes interessadas, onde cada parte só pode se comunicar se seguir as regras deste acordo e enviar uma solicitação estruturada para a outra parte.

A seguir serão apresentados alguns detalhes sobre a comunicação de um software com uma API. Esta comunicação se baseia em “métodos” e “requisições”, onde os métodos indicam a ação a ser realizada no recurso ou função especificada. Já as requisições são de fato a solicitação que o cliente fará para ter acesso a determinado recurso ou função [OneDayTesting].

Aqui podemos ver os principais métodos de APIs e sua aplicação:

**Quadro 1: Métodos de APIs**

| | |
|---|---|
| GET | Obter os dados de um recurso |
| POST | Criar um novo recurso |
| PUT | Substituir completamente os dados de um determinado recurso |
| PATCH | Atualizar parcialmente um determinado recurso |
| DELETE | Excluir um determinado recurso |
| HEAD | Similar ao GET, mas utilizado apenas para se obter os cabeçalhos de resposta, sem os dados em si |
| OPTIONS | Obter quais manipulações podem ser realizadas em um determinado recurso |

Uma das formas de documentar APIs é por meio da ferramenta Swagger. O Swagger é um conjunto de ferramentas de código aberto construído em torno da especificação OpenAPI que pode ajudá-lo a projetar, construir, documentar e consumir APIs REST [Swagger specification].

Para realizar testes de API utilizamos a documentação do Swagger. Os testes de API são extremamente importantes, pois ajudam a garantir o funcionamento, o desempenho e a confiabilidade da sua aplicação. O Swagger oferece ferramentas para testes manuais, automatizados e de desempenho.

No Swagger, existem ferramentas para vários tipos de tarefas relacionadas ao desenvolvimento da API de um serviço WEB. Uma dessas tarefas é a especificação da API, que consiste em determinar os modelos de dados que serão entendidos pela API e as funcionalidades presentes na mesma. Para cada funcionalidade, é preciso especificar o seu nome, os parâmetros que devem ser passados no momento de sua invocação e os valores que irão ser retornados aos usuários da API. Entre estas ferramentas, podemos citar o OpenAPI Specification. Após especificar a API, o Swagger facilita a implementação da API, pois com a ferramenta Swagger Codegen é possível montar o código inicial automaticamente nas principais linguagens de programação. Para auxiliar na utilização da API, o Swagger dispõe de ferramenta para deixar a visualização mais intuitiva [TerraLab], conforme ilustrado pela Figura 1, que mostra um exemplo de API REST e que corresponde à API do serviço EducAPI.

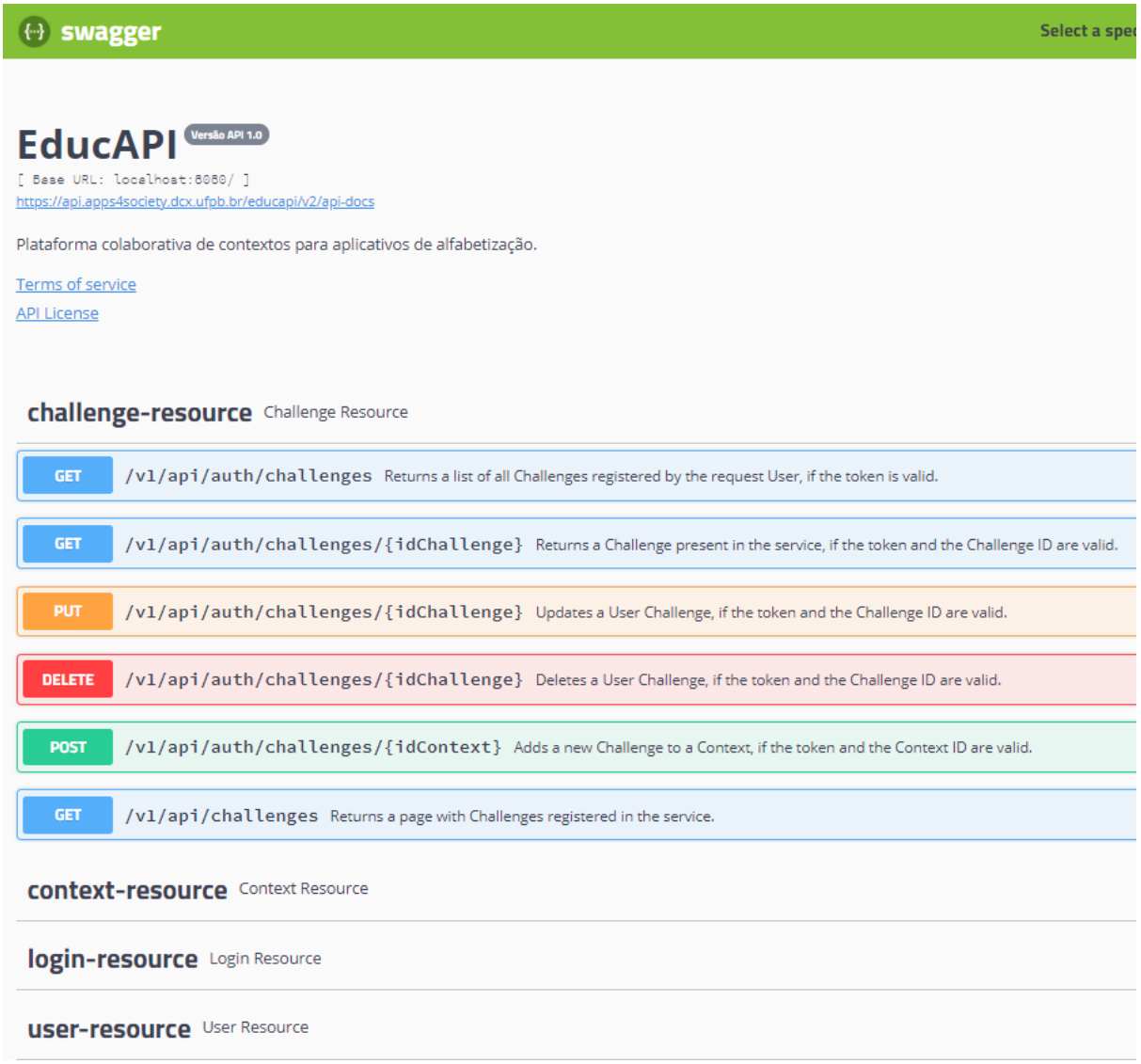


**Figura 1. Interface do Swagger para a API do EducAPI**

## 2.2 A API do Sistema EducAPI

O sistema EducAPI [Ludgério et al. 2021] é um sistema que gerencia uma base de dados multimídia colaborativa dinâmica, onde os usuários podem cadastrar e consultar desafios (ou palavras) agrupadas em contextos (ou temas).

> Pode-se ter, por exemplo, um tema como "NATUREZA" e desafios como "ÁRVORE", "RIO", "MAR", etc, relacionados a este contexto/tema. Cada tema e desafio está associado no sistema a imagens, áudios e vídeos que os representem. A ideia é dar suporte a um processo de aprendizagem em que se tenta explorar com aprendizes temas e recursos multimídia que façam sentido em suas vidas e relacionados aos contextos em que vivem ou que lhe são apresentados em sala de aula [Ludgerio et al. 2021].

Um professor cadastrado no sistema pode fazer cadastros de palavras e imagens, editar, deletar, consultar desafios e contextos. Esses desafios e contextos que foram cadastrados pelo usuário, podem ser acessados tanto por ele, quanto por outros usuários que fazem uso do sistema.

A Figura 2 é uma representação da abordagem proposta para a utilização do sistema. Nela se ilustram professores utilizando uma interface de software para gerir os contextos e que se comunica com o serviço provido pelo EducAPI, que é responsável por disponibilizar e armazenar as informações relativas a contextos e desafios cadastrados por diferentes usuários. A Figura também ilustra que usuários de diferentes perfis (e.g. crianças ou adultos) podem acessar o sistema através de softwares que utilizam os serviços do EducAPI para as mais diversas finalidades, como por exemplo, atividades de associação de imagens a palavras, jogos da forca, jogos de memorização, dentre outros [Silva 2021].

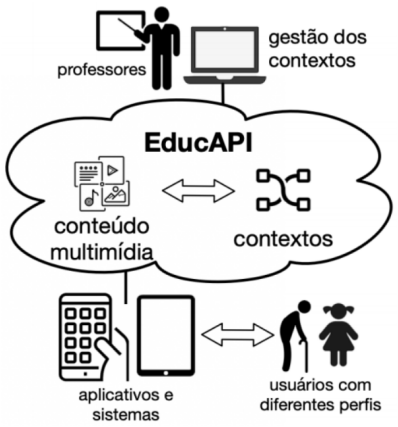


**Figura 2: Abordagem Proposta com o EducAPI**

O sistema, desenvolvido em Java, fornece uma API REST para permitir o acesso dos usuários aos seus serviços [Silva 2021]. Essa API permite o cadastro e consulta de palavras agrupadas em contextos e ilustradas por recursos multimídia [Ludgerio et al. 2021].

No entanto, para que o serviço possa ser utilizado na prática, a ideia é que existam sistemas e aplicativos para permitir que professores cadastrem palavras (ilustradas por imagens/vídeos/áudios) e que façam sentido para os aprendizes em processo de alfabetização, agrupando tais palavras em contextos ou temas.

## 2.3 A Ferramenta REST Assured

Rest-Assured é uma ferramenta que foi desenvolvida para facilitar a criação de testes automatizados para APIs REST. Esta oferece suporte para validar requisições HTTP utilizando JSON [OneDayTesting]. O JSON (*Java Script Object Notation*) é utilizado para estruturar dados em formato de texto e permitir a troca de dados entre aplicações de forma simples e rápida.

O Rest Assured interage com a API Rest em um modo cliente e oferece também extensas opções de validações das requisições que são enviadas nos serviços REST, tais como: código de estado (*Status Code)*, cabeçalhos (*Headers*) e também elementos do corpo da requisição (*Body*). Dessa forma, ela é considerada extremamente flexível para utilizar na criação de testes automatizados de API [ONeDayTesting].

A ferramenta nos permite também testar serviços RESTful em Java de um jeito muito prático. Basicamente, ela nos provê uma maneira de criar chamadas HTTP, como se fôssemos

um cliente acessando a API [Amaral 2019]. Além disso, ela suporta os métodos POST, GET, PUT, DELETE, OPTIONS, PATCH e HEAD e pode ser usada para validar e verificar a resposta dessas solicitações [Amaral 2019].

Segundo a documentação do Rest Assured [OneDayTesting], estas são as dependências que devem ser importadas ao desenvolver um projeto usando esta ferramenta [OneDayTesting]:

- **Rest-Assured:** Dependência que importa a ferramenta Rest Assured. Versões acima do Java 9 importam todas as outras dependências que iremos comentar automaticamente.
- **Json-Path**: Dependência que contém funções semelhantes ao XPath. JSONPath é usado para selecionar e extrair os valores de propriedade de um documento JSON.
- **Xml-Path:** Dependência que facilita a conversão de respostas em XML.
- **Json-Schema-Validator**: Permite realizar validações com arquivos em Schemas JSON.
- **AssertJ**: Dependência que permite realizar as assertivas (*assertions*) na aplicação.

O Rest Assured utiliza a sintaxe da linguagem GHERKIN (*Given, When, Then*), amplamente utilizada em práticas como BDD. Essa sintaxe é utilizada para dividir o código em seções para demonstrar seu comportamento. A seguir serão apresentadas resumidamente algumas construções utilizadas em testes com Rest Assured:

- **Given**: Define o contexto da aplicação, como por exemplo destacar quais headers, parâmetros, portas e tokens que a aplicação deve ter para ser inicializada.
- **When**: Este trecho identifica a proposta de seu cenário. “Quando” você insere um dado, algo deve acontecer.
- **Method**: Este campo logo após o “when” define o método que será executado na aplicação, podendo ser um POST, PATCH, GET, entre outros.
- **Then**: Será o resultado a ser exibido. Nele você pode validar qual foi a resposta da aplicação. [OneDayTesting]

Um exemplo de teste utilizando essas construções é apresentado no trecho de código da Listagem 1

```
@Test
public void testCriarUsuario(){
   //Configurar caminho de acesso a API REST
   baseURI = "http://localhost";
   port = 8080;
   basePath = "v1/api";
//Requisição para criar usuario
given().body("{\n" +
              "\"email\":\"fernando.ribeiro@dcx.ufpb.br\",\n"
+
               "  \"name\": \"Nando Ribeiro\",\n" +
               "  \"password\": \"12345678\"\n" +
               "}")
```

```
        .contentType(ContentType.JSON)
    .when()
        .post("/users")
    .then()
        .log().all();
}
```

**Listagem 1. Exemplo de criação de usuário usando o Rest Assured**

O teste apresentado na Listagem 1, é uma requisição seguindo algumas construções utilizadas em testes no Rest Assured. Este é um teste de criação de um usuário. Nele é configurado o caminho que irá dar acesso à API REST em uso. Em seguida, é montado o corpo da requisição utilizando o **given**, **when** e **then**. No exemplo da Listagem 1 se especifica que dado que (**given**) seja passado um corpo da requisição com os dados indicados, quando (**when**) for enviado via **post** (**method**) para o "/users", então (**then**) deverá ser exibida a resposta da requisição com o usuário criado, ou seja, o (**log().all()**) exibirá no console as informações do usuário que foi criado, assim como todas as informações que foram passadas na requisição. Então a requisição cria um usuário para que ele possa se logar e acessar as funcionalidades do sistema EducAPI.

## 3. Metodologia

Foi realizado inicialmente um levantamento bibliográfico, com o propósito de compreender os principais conceitos e ferramentas necessários para a realização deste trabalho, tomando por base artigos e livros que apresentem os fundamentos teóricos sobre a área de testes, APIs e automação de testes.

Conforme previamente apresentado, este trabalho pretende realizar testes de uma API do sistema EducAPI, utilizando duas ferramentas para testes, a ferramenta Postman e a Rest Assured e relatar a experiência obtida e algumas lições aprendidas.

Para dar início aos testes, primeiramente foi necessário colocar em execução localmente a API REST do EducAPI. Com ela já rodando, pudemos fazer algumas requisições para melhor compreender a API e realizar configurações iniciais nas ferramentas de teste, montando a estrutura básica das principais requisições feitas à API REST e identificando operações onde eram necessárias autenticações ou autorizações, como operações para editar, consultar, deletar dados. Posteriormente, foram desenvolvidos testes utilizando o Postman e o Rest Assured e documentados os passos e lições obtidas nesse processo de desenvolvimento.

Os testes da API REST do sistema EducAPI que foram desenvolvidos neste trabalho estão disponíveis em https://github.com/FernandoSouzaa/TesteEducAPI.git. Alguns destes testes podem ser encontrados nos Apêndices A e B. As funcionalidades testadas foram as de cadastrar, consultar, alterar e excluir contextos, desafios e usuários. Os testes foram implementados e executados localmente, utilizando a mesma máquina onde o serviço estava rodando.

## 4. Testes Automáticos para o EducAPI com PostMan e Rest Assured

### 4.1 Solução de Testes de API com PostMan e Swagger

Para testar a API do EducAPI usando PostMan foi necessário entender melhor o serviço a ser testado e os principais tipos de requisições em serviços REST.

A seguir serão apresentados exemplos de alguns dos métodos que são mais utilizados no envio de requisições HTTP e da interface do Postman nesses tipos de requisição.

O método **GET** serve para requisitar um recurso específico. Requisições utilizando o método devem retornar apenas dados. Fazer uma consulta no sistema e ver um recurso específico. Neste caso não é necessário fazer o preenchimento do corpo da requisição (body). A Figura 3 ilustra o uso do Postman para realizar a requisição GET à URI http://localhost:8080/v1/api/auth/users. Para essa requisição são passados dados de login do usuário por meio do cabeçalho da requisição (headers) e são recebidos os dados referentes ao usuário solicitado.

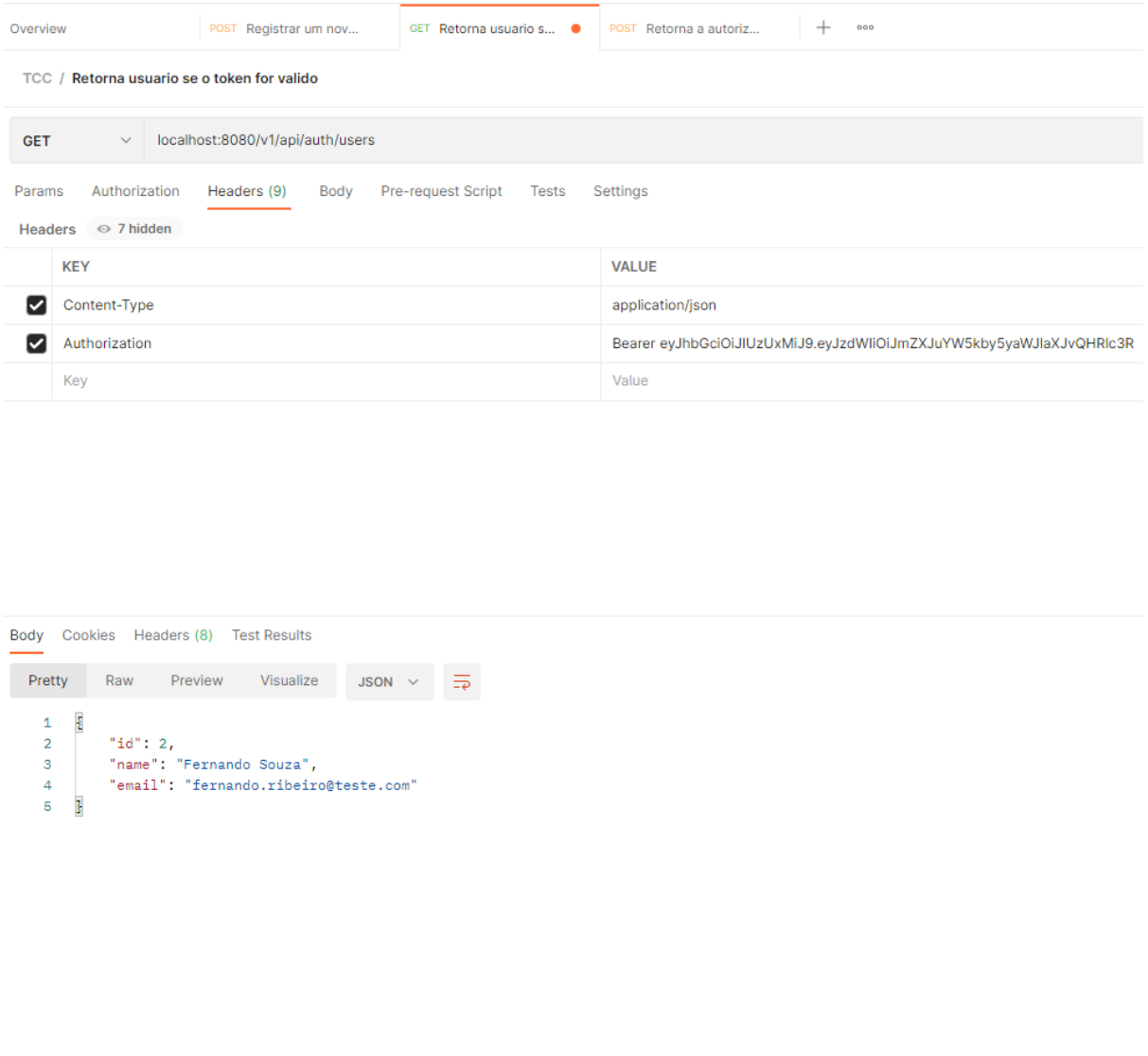


**Figura 3 - Exemplo de requisição GET através do Postman**

O método **POST** é utilizado para enviar dados relativos a um recurso específico. Por exemplo, pode ser utilizado para fazer um cadastro ou criar um recurso. No POST é necessário fazer o preenchimento do body, passando as informações no corpo. A Figura 4 ilustra um exemplo de requisição do tipo POST ao serviço via URI http://localhost:8080/v1/api/users onde se passam os dados de cadastro do usuário.

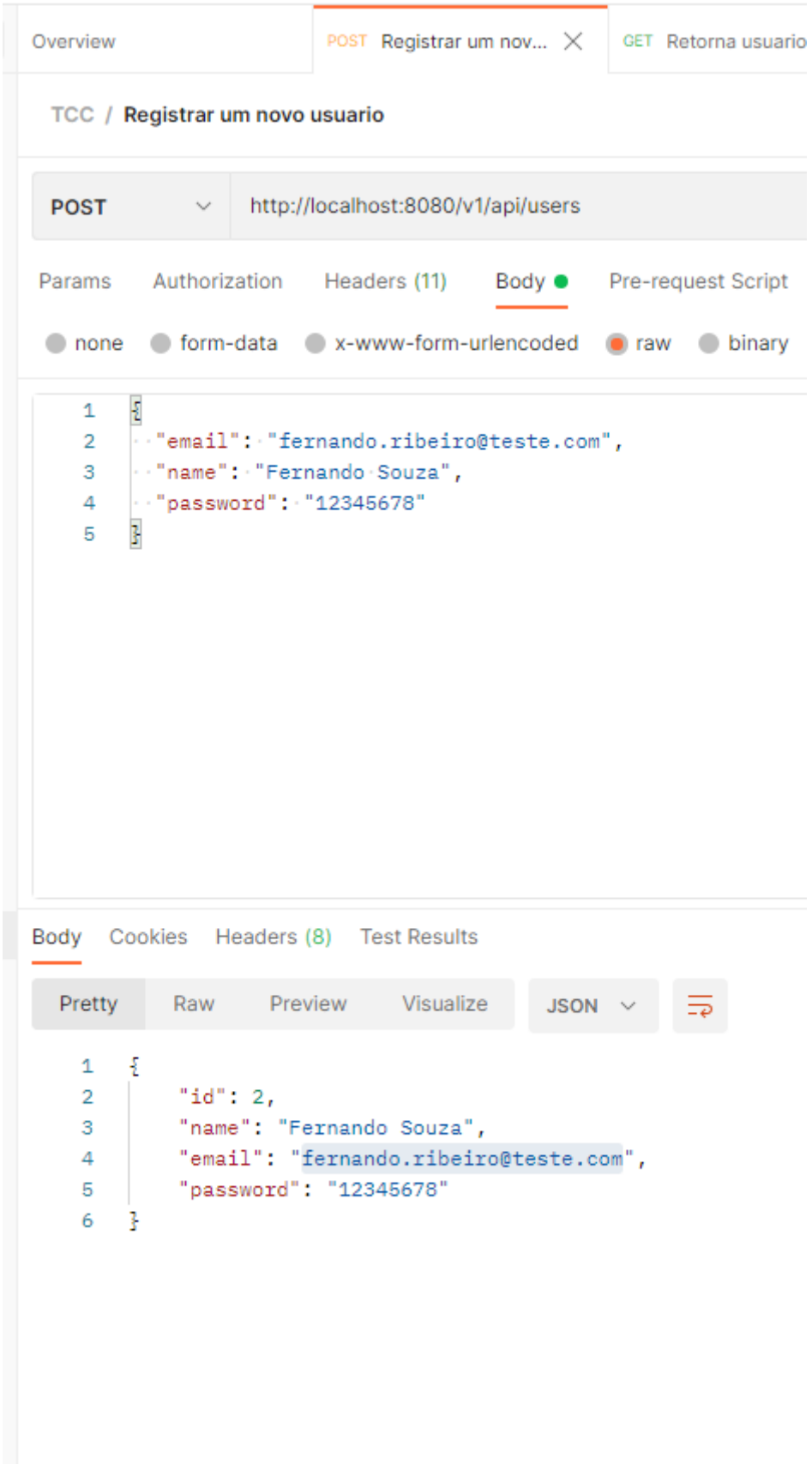


**Figura 4 - Exemplo de requisição POST através do Postman**

O método **PUT** serve para atualizar os dados de um recurso com os dados passados na requisição. Por exemplo, pode ser utilizado para fazer uma alteração de uma informação. Para que seja realizado, é necessário informar o código do recurso e informar no *body* os campos que deseja alterar. A Figura 5 ilustra o uso do Postman para realizar a requisição PUT à URI http://localhost:8080/v1/api/auth/users. Para essa requisição são passados dados de login do usuário por meio do cabeçalho da requisição (*headers*) e o corpo (*body*) da requisição para fazer a alteração das informações.

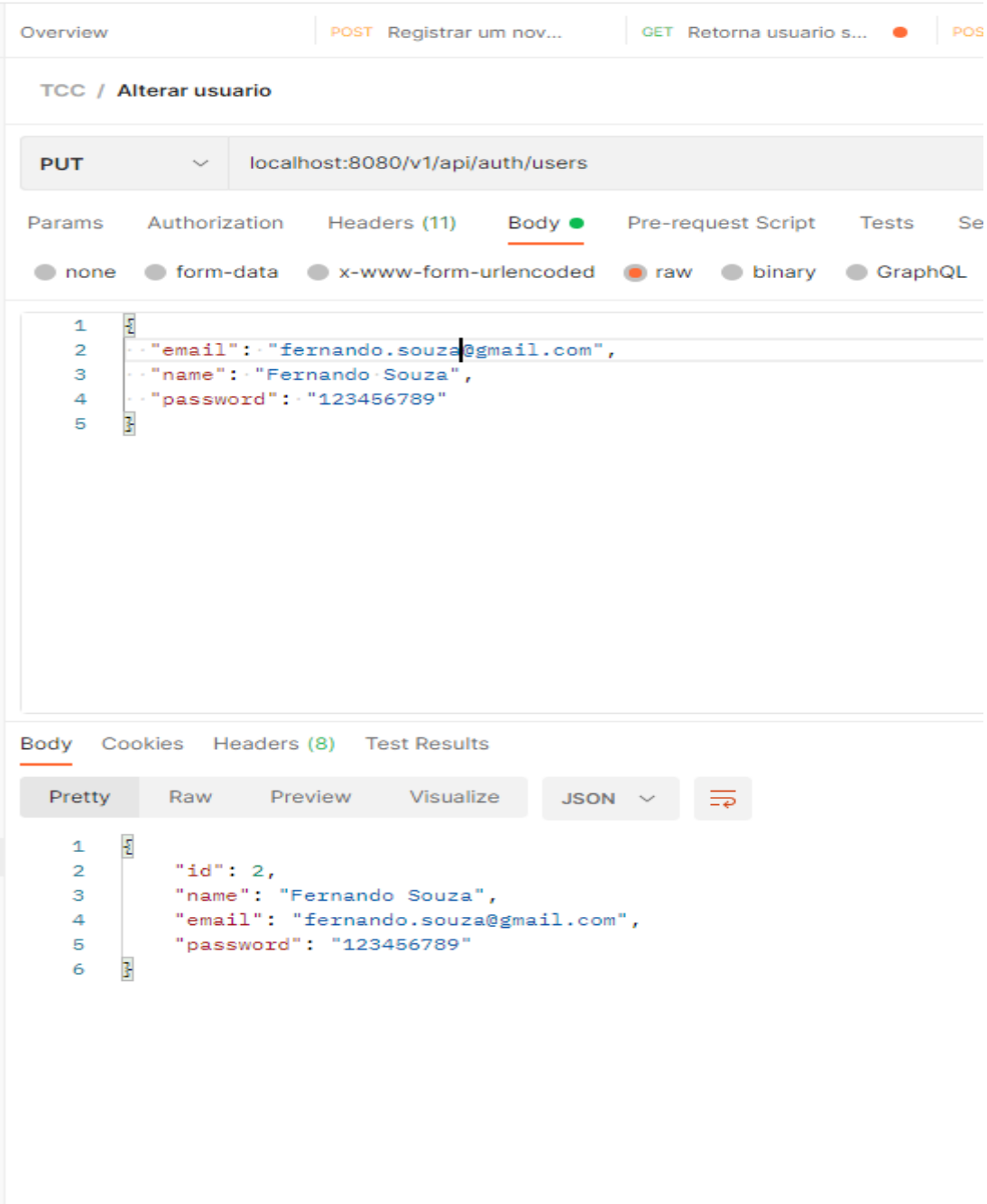


**Figura 5 - Exemplo de requisição PUT através do Postman**

O método **DELETE** remove um recurso específico. Para realizar essa requisição, é necessário informar o código do recurso e não é necessário o preenchimento do *body*. A Figura 6 ilustra o uso do Postman para realizar a requisição PUT à URI

http://localhost:8080/v1/api/auth/users. Para essa requisição são passados dados de login do usuário por meio do cabeçalho da requisição (*headers*) e os dados referentes ao usuário passado serão excluídos.

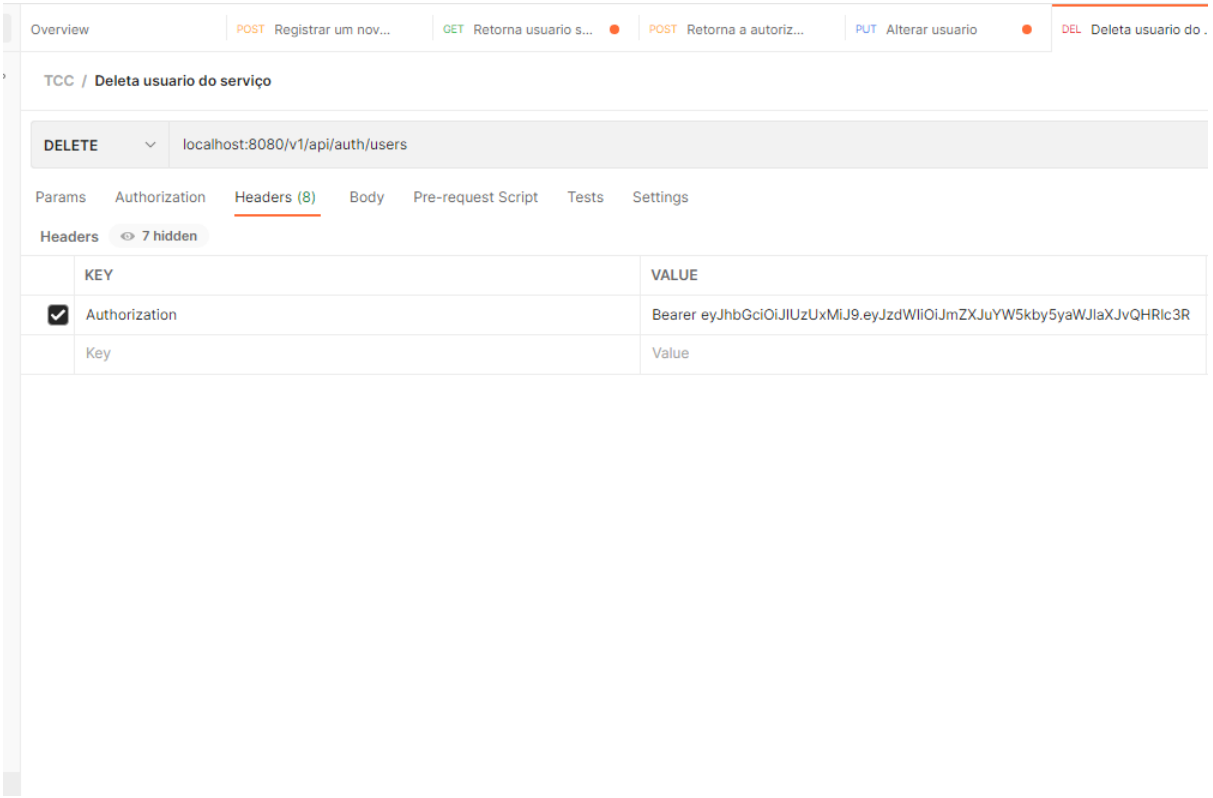


**Figura 6 - Exemplo de requisição DELETE através do Postman**

A Figura 7 ilustra o uso de uma asserção do status code. Para essa requisição vamos em “Tests” e estando nele, vamos fazer o nosso teste. Nesse teste é verificado se o status code está de acordo com o da documentação após o usuário ser excluído:

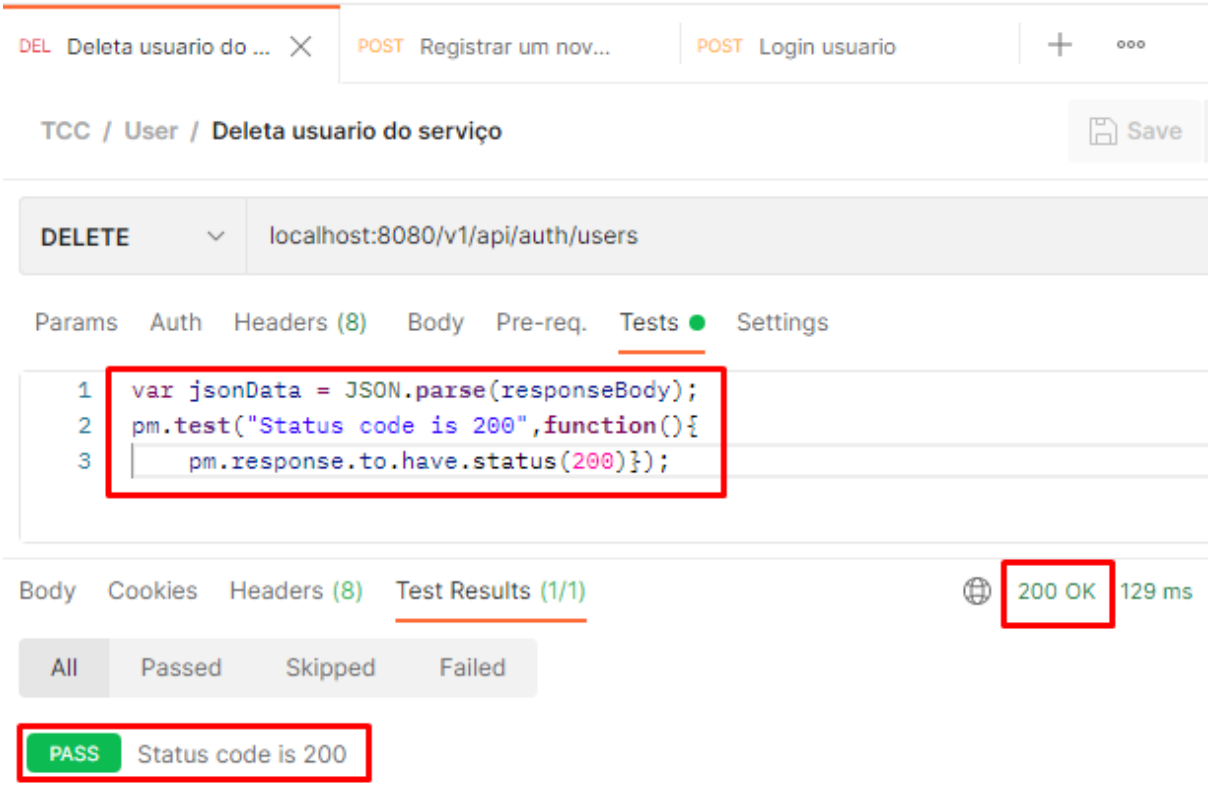


**Figura 7 - Exemplo de asserção de status code com o Postman**

Atualmente na empresa Phoebus Tecnologia e mais especificamente na equipe cujo trabalho motivou este artigo, um dos tipos de testes a fazer são os testes de API. Estes testes são semi-automáticos e visam verificar por meio do Postman se aquela API está funcionando corretamente, além de observar aspectos como confiabilidade, desempenho (*performance*) e segurança da aplicação.

Para a execução dos testes no Postman é criado de início uma coleção (*collection*), que é um grupo de solicitações salvas que podem ser organizadas em pastas. Dentro dessa coleção é criado uma requisição (*request)*. Com a requisição criada, definimos o método a ser usado POST/GET/PUT/ e logo em seguida definimos a URL do *endpoint*, adicionamos os *Headers* necessários. Caso se esteja fazendo uma operação de inclusão ou de edição de dados, deve ser preenchido o *body*. Com isso pronto, podemos salvar e enviar a requisição, aguardar alguns segundos e obter as respostas esperadas daquela requisição verificando de forma manual se os resultados da requisição são os esperados de acordo com os casos de teste da empresa. A Figura 8 ilustra algumas áreas da interface do Postman, explicadas a seguir:

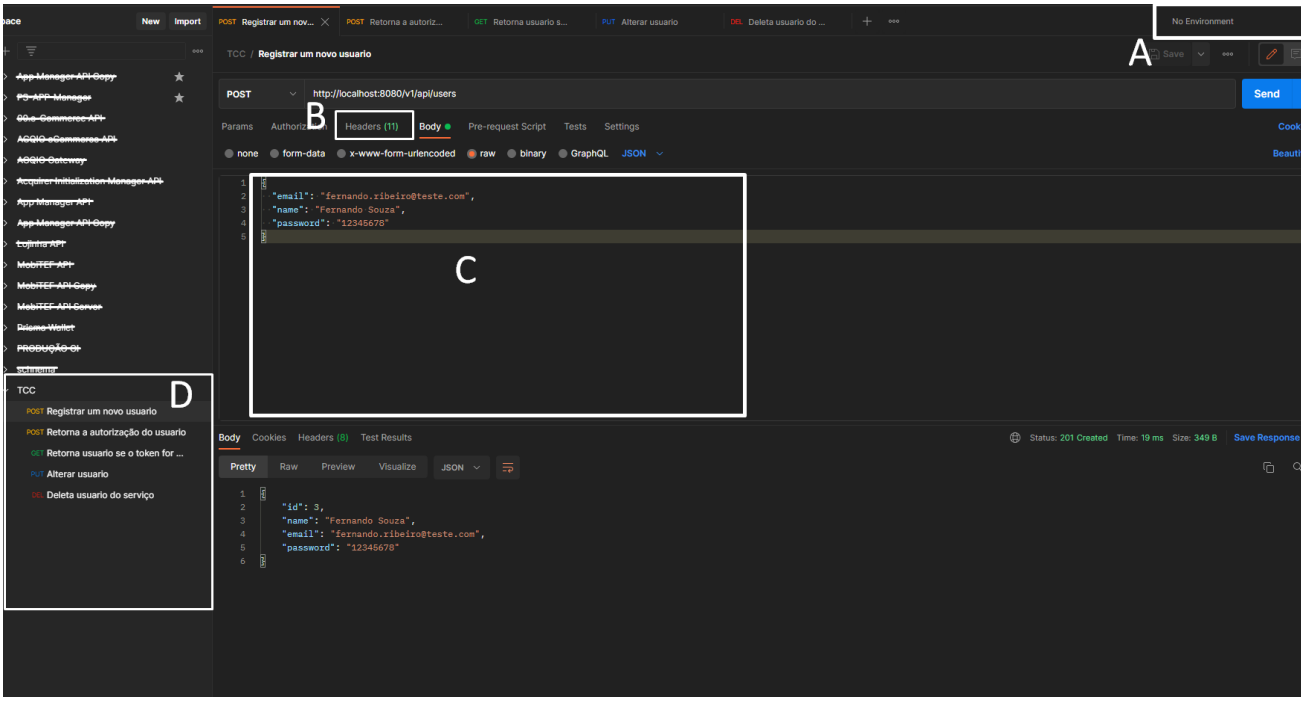


**Figura 8 - Áreas de interface do Postman**

- **A-Environment (Ambiente):** Ambiente para onde serão direcionadas as requisições. Nesta área que serão definidos dados de:
  - **MerchantId** - Identificador de sua loja nas APIs.
  - **MerchantKey** - Chave de segurança da sua loja nas APIs.
  - **URL do POST/PUT** - Endpoint para criar ou editar transações.
  - **URL do GET** - Endpoint para consulta de transações.
- **B-Header** Aqui existem o MerchantId/MerchantKey, que por padrão usam os mesmos dados registrados em environment.
- **C-Body:** É o conteúdo das Requisições. Aqui é onde você pode alterar ou criar exemplos para a API e validar o conteúdo do seu POST/GET/PUT
- **D - Collection (Coleções):** Local que contém todos os exemplos e códigos que podem ser utilizados na API. Aqui existem as criações de transações, consultas e outras funcionalidades. O número de coleções é ilimitado, ou seja, você pode criar várias coleções para se adequar ao seu estilo de uso do Postman.

Para iniciar os testes com o Postman, precisamos inicialmente baixar e instalar esta ferramenta. O download do Postman pode ser feito através do link: https://www.postman.com/downloads/?utm_source=postman-home. Com o Postman já instalado, vamos agora precisar subir a API do sistema EducAPI na máquina para ter acesso a ela e as suas funções. Com ela já rodando, vamos abrir o aplicativo e criar uma coleção que será utilizada para guardar as requisições. Quando o software estiver aberto, clique no botão "New" e depois em "Collection", conforme ilustrado pela Figura 9:

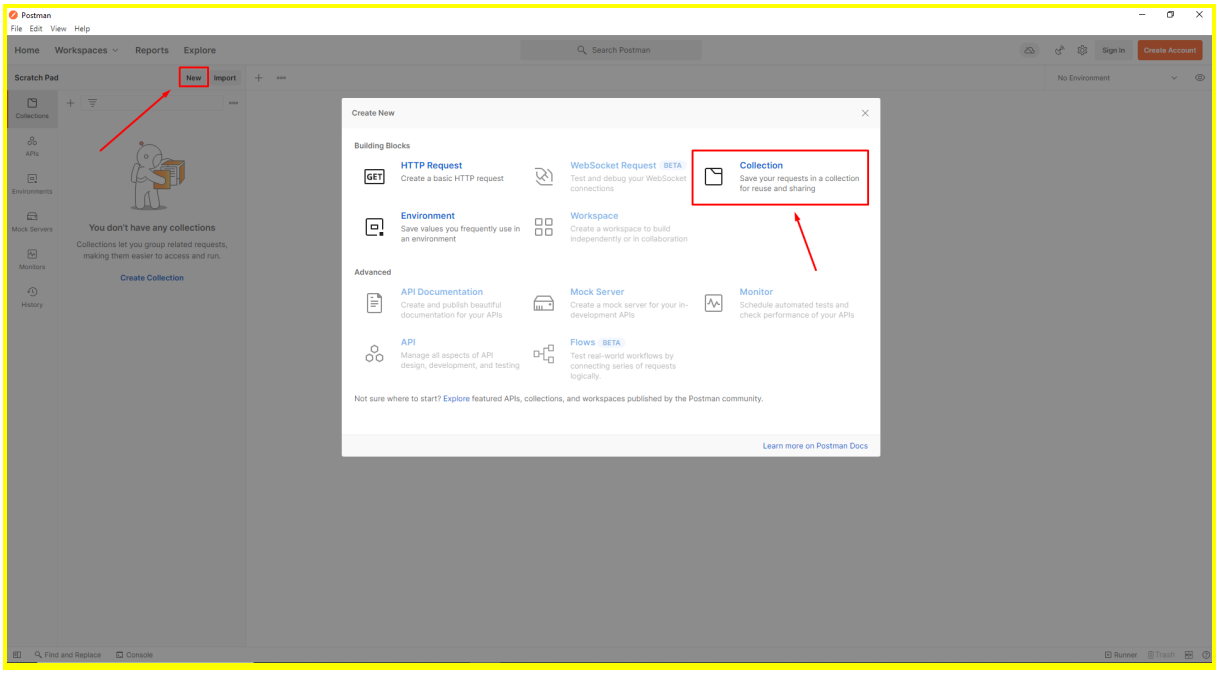


**Figura 9 - Criando uma Coleção (*Collection*)**

Dê um nome a sua *Collection*. Para fazer isso, clique na *Collection* criada e depois clique em editar e dê um nome para essa coleção. A nossa, ilustrada na Figura 10, irá se chamar EducAPI:

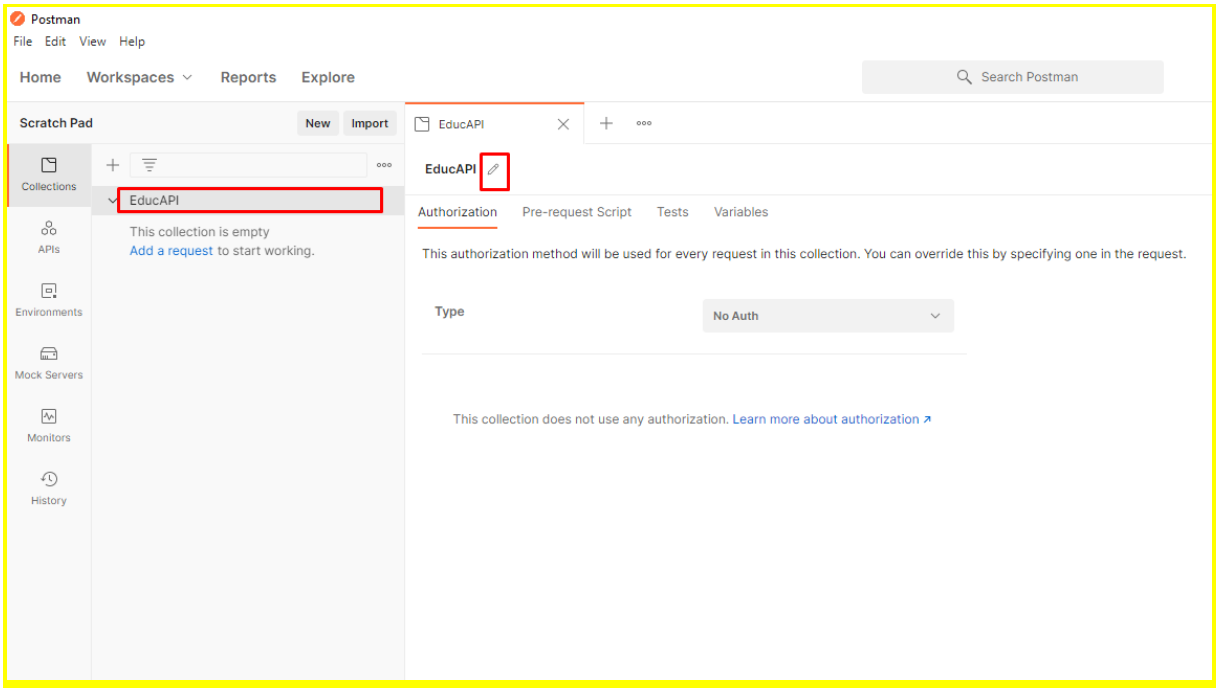


**Figura 10 - Nomeando a coleção (Fonte: O Autor)**

Agora, para criar uma *Request* (requisição ao serviço), clique nos 3 (três) pontos e depois clique em "*Add Request*", conforme ilustrado pela Figura 11:

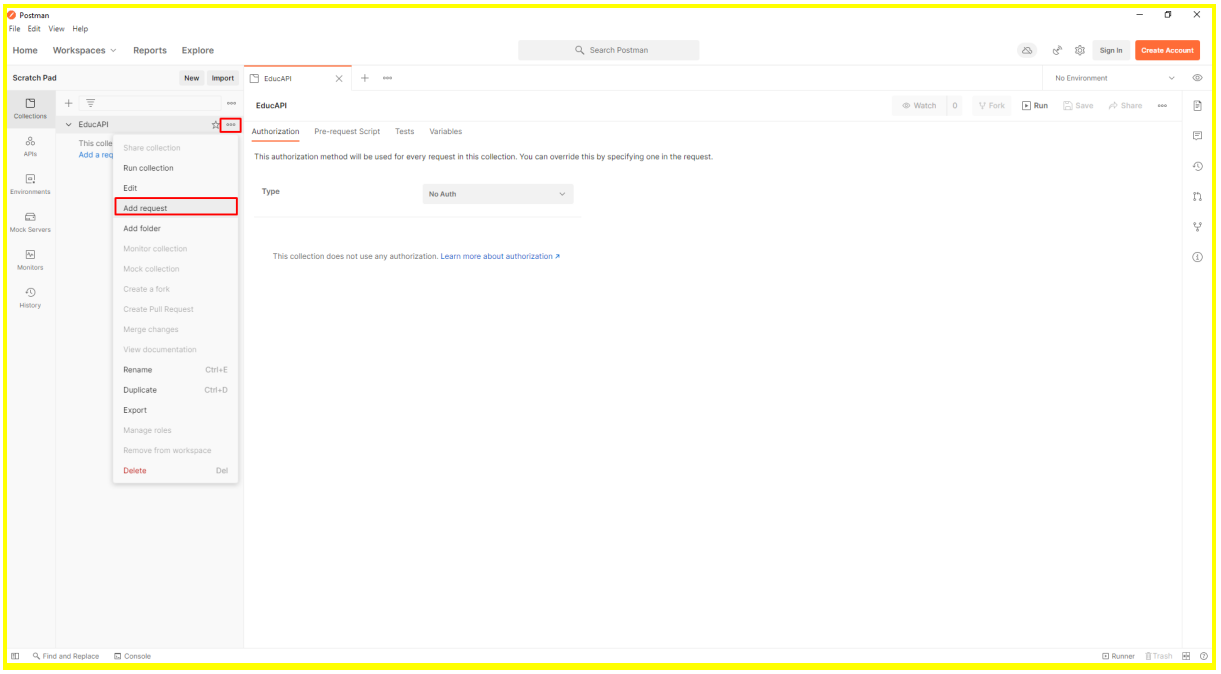


**Figura 11 - Criando uma *Request* (requisição)**

Deveremos dar um nome a nossa requisição. Geralmente uma requisição recebe o nome do que ela faz de fato. No exemplo ilustrado pela Figura 12, a requisição criada irá fazer um cadastro de um contexto no EducAPI e por isso foi chamada de "Cadastrar Contexto":

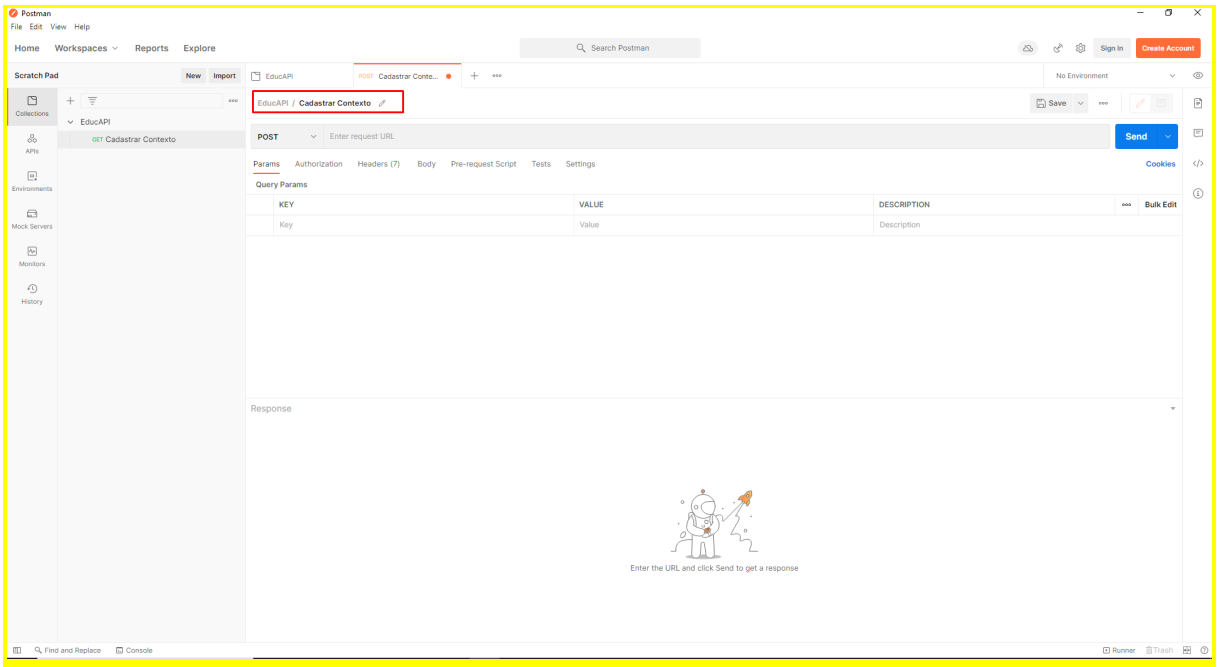


**Figura 12 - Nomeando uma requisição (Cadastrar Contexto)**

Com a requisição já criada, vamos agora passar no corpo da requisição a url do serviço, escolher qual será o tipo de requisição enviada, clicar em "*raw*" (modo bruto) e

escolher o tipo “JSON”. Exemplos de informações que podem ser passadas no corpo da requisição estão ilustrados pela Figura 13:

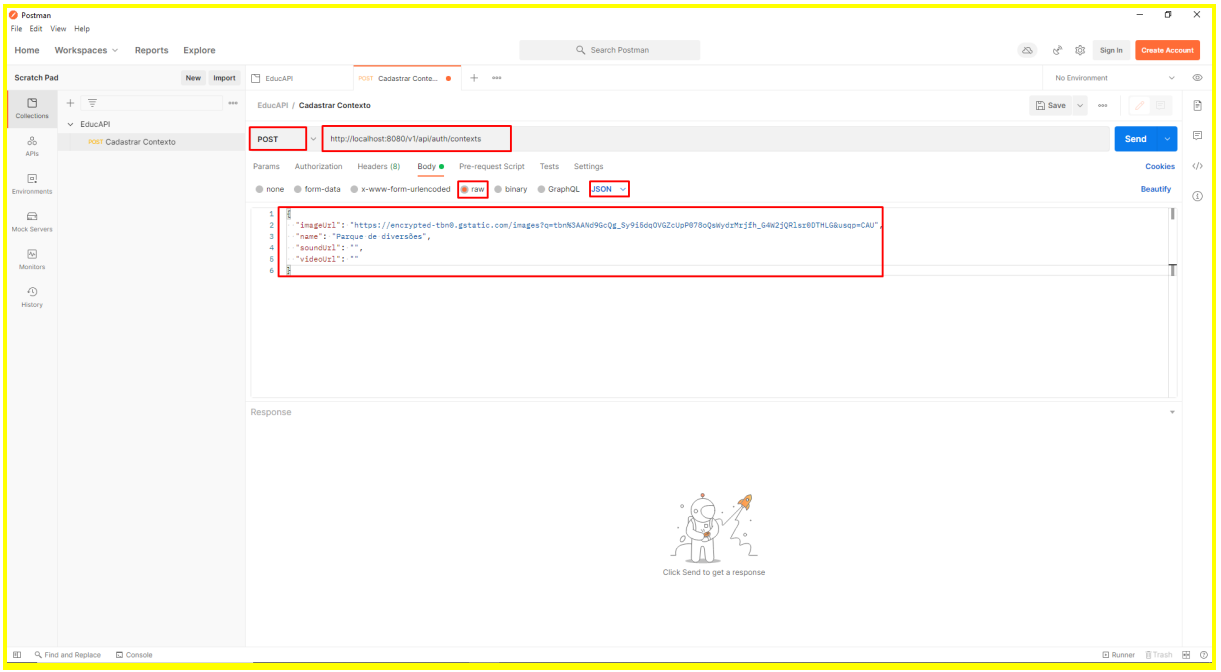


**Figura 13 - Passando as informações no corpo da requisição**

Para criar um contexto no EducAPI, é preciso passar no cabeçalho (*header*) uma autorização (*Authorization*) com o token que foi gerado quando o usuário fez seu login no Sistema. No corpo (*body*) da requisição devem ser passados os dados do usuário e após clicar em “*Send*”, o Postman envia a requisição e na resposta a essa requisição poderemos ver que o contexto foi cadastrado com sucesso, conforme ilustrado pela Figura 14:

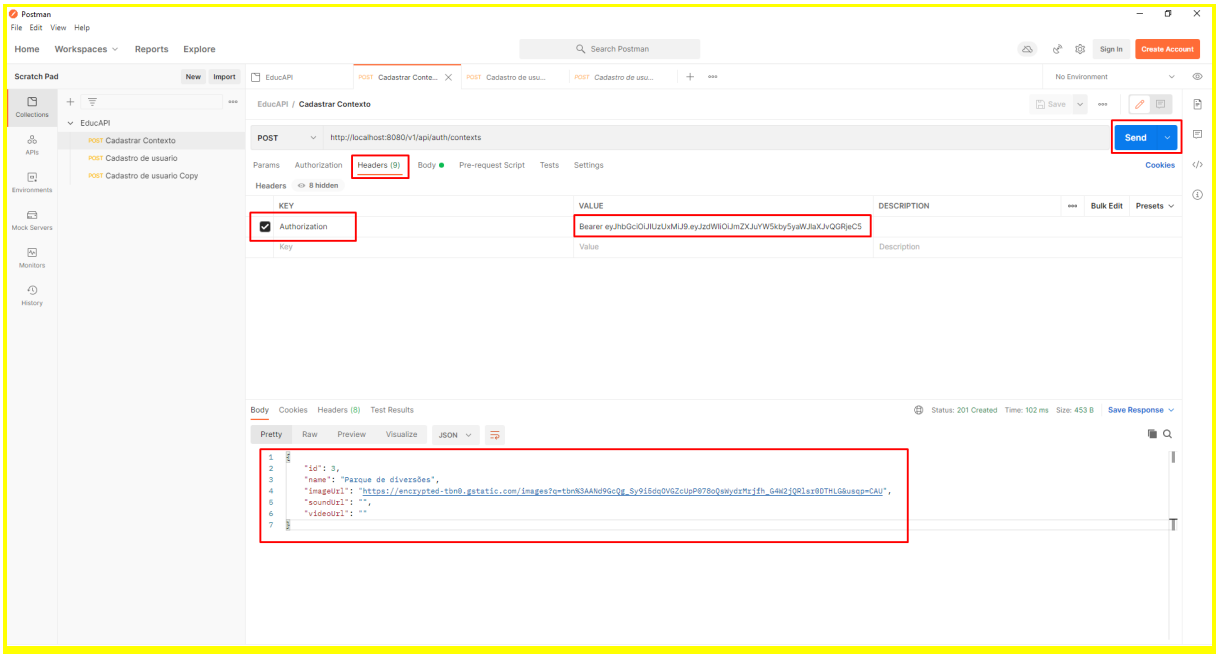


**Figura 14 - Criação de Contexto no EducAPI usando o Postman**

Para fazer o cadastro de um desafio, basta seguir os passos anteriores, como foi feito com o cadastro de um contexto. Sendo assim, vamos passar no cabeçalho (header) a autorização (Authorization) e passar o token, conforme ilustrado pela Figura 15:

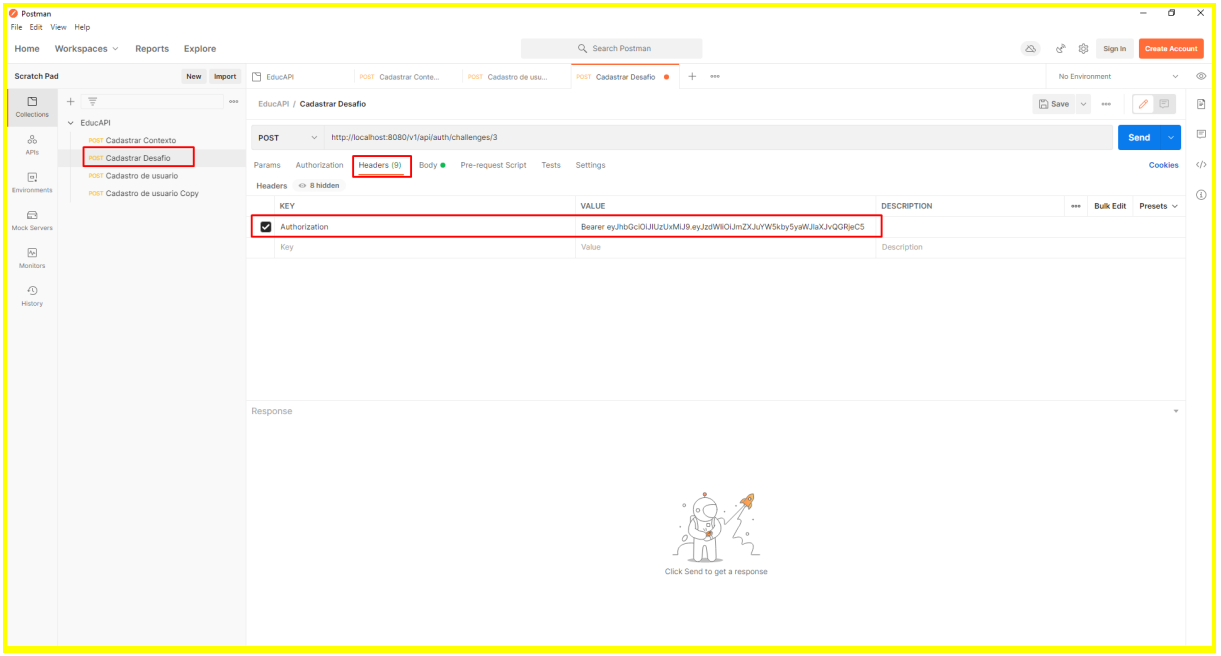


**Figura 15 - Cadastro de desafio através do Postman (Fonte: O autor)**

Além disso, vamos passar no corpo (*body*) os dados do cadastro de um Desafio. Na url devemos passar o identificador do contexto cadastrado e clicando em "*Send*" será enviada a requisição. Veremos então a resposta do cadastro do desafio, conforme ilustrado pela Figura 16:

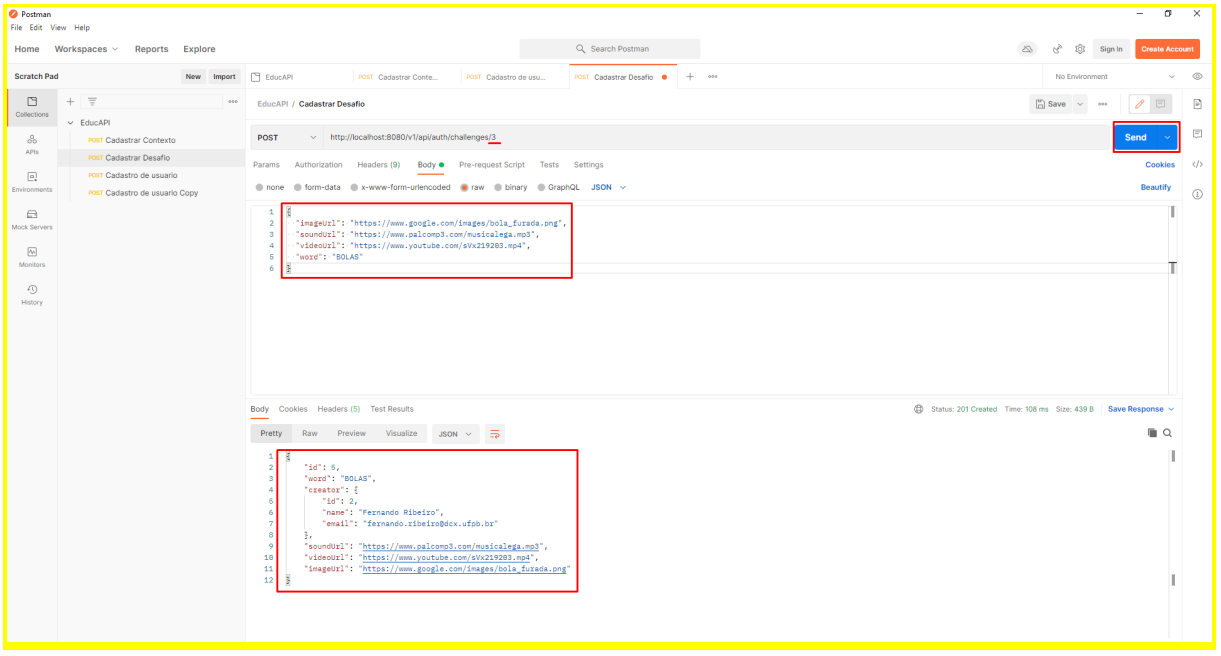


**Figura 16 - Cadastro de Desafio utilizando o Postman (Fonte: O autor).**

Para consultar um contexto, basta mudar o tipo da requisição para “GET” e informar na url o identificador do contexto. Quando isso é feito, será retornado o contexto que foi cadastrado correspondente ao identificador passado, conforme mostrado pela Figura 17:

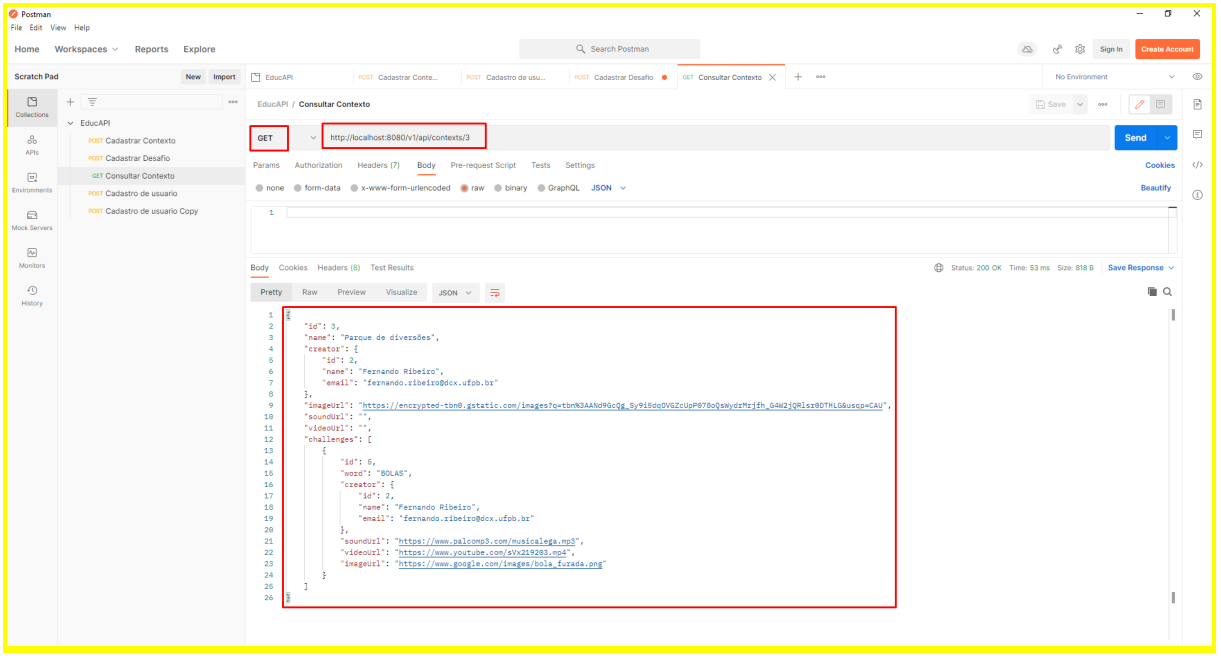


**Figura 17 - Consulta de um Contexto utilizando Postman (Fonte: O autor)**

Para fazer consulta de um desafio, basta seguir os passos da consulta anterior, acrescentando a autorização e o token, conforme mostrado na Figura 18:

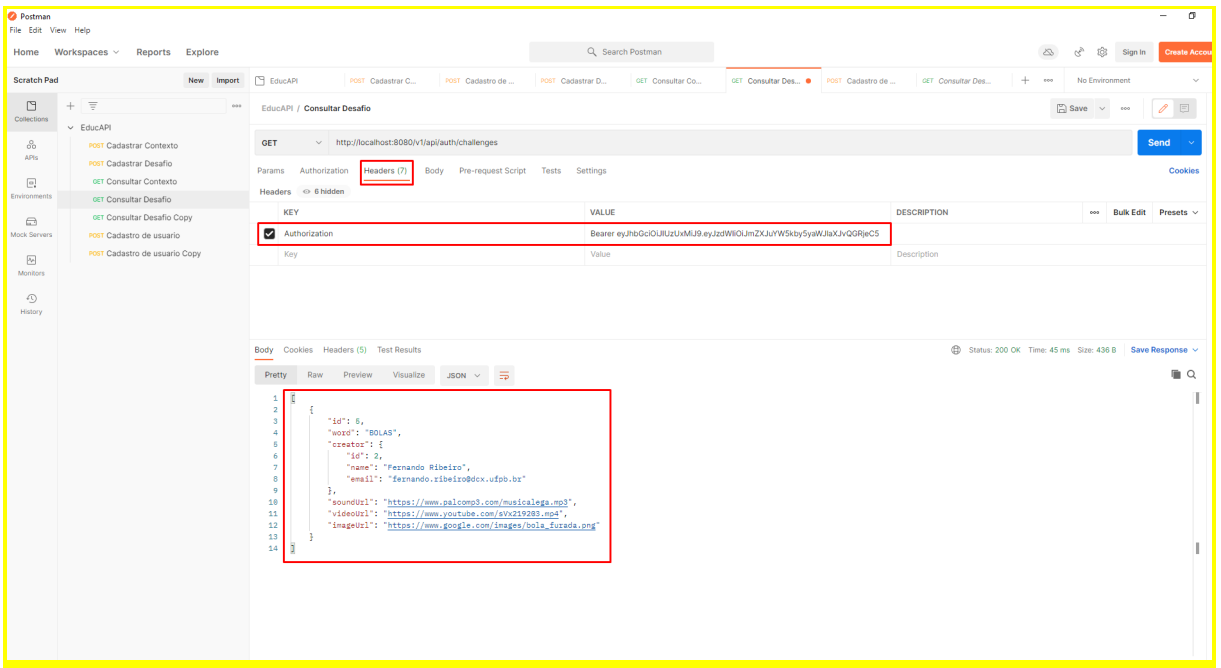


**Figura 18 - Consultar um Desafio usando o Postman (Fonte: O autor)**

## 4.2 Testes do Sistema EducAPI usando Rest Assured

Os primeiros passos para começar a realizar testes com o Rest Assured são os seguintes:

1. Adicionar as dependências do Rest Assured e JUnit ao arquivo POM do projeto. Para os testes do EducAPI será usada a versão 3.0.0 do Rest Assured que pode ser encontrada em https://mvnrepository.com/artifact/io.rest-assured/rest-assured/3.0.0 e a versão 4.12 do JUnit que pode ser encontrada em https://mvnrepository.com/artifact/junit/junit/4.12,. A Listagem 2 ilustra a configuração feita no arquivo pom.xml do projeto.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <version>3.0.0</version>
  <scope>test</scope>
</dependency>
```

**Listagem 2 - Adicionando dependências**

2. Com as dependências do projeto já adicionadas, precisamos subir a API do sistema EducAPI na máquina e com ela já rodando, damos início à criação da estrutura dos pacotes de testes. No exemplo mostrado na Figura 19 é ilustrada uma estrutura onde temos *challenge*, *context* e *users* em pacotes separados para deixar os testes mais organizados:

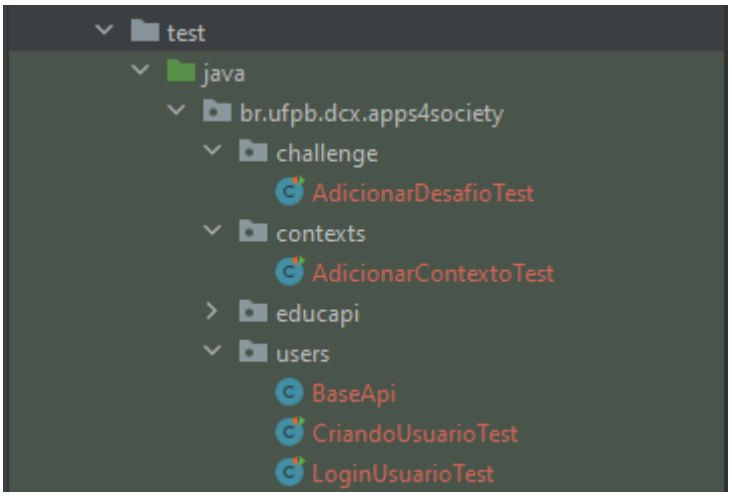


**Figura 19 - Criar estrutura dos pacotes de testes**

3. O próximo passo é fazer as importações estáticas para poder realizar verificações automáticas nos testes (asserções). Quando começamos a digitar os códigos de testes do Rest Assured, automaticamente a IDE irá sugerir os *imports*, mas através da documentação do Rest Assured recomenda-se fazer as importações estáticas ilustradas pela Listagem 3:

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;
```

**Listagem 3 - Fazendo as importações**

Após os primeiros passos, inicia-se a programação dos testes. Para isso é importante que o desenvolvedor tenha uma noção de programação para dar início aos seus testes Para isso, inicialmente deve ser obtido um "*token*" para permitir que se possa chamar métodos que exigem autenticação.

Para obter o *token*, no exemplo ilustrado pelo código da Listagem 4, será feita uma requisição onde são enviados os dados do usuário através do corpo (*body*) ao fazer o post. O *token* obtido será extraído e armazenado na variável token, para utilização nos testes.

```
public void testSistemaEducAPI(){

   //Configurar caminho de acesso a API REST
   baseURI = "http://localhost";
   port = 8080;
   basePath = "v1/api";

   //Login do usuario
   //a varivel token serve para guardar o token do usuario
após ele ter si logado no sistema
   String token = given()
           .body("{\n" +
                   "  \"email\":
\"fernando.ribeiro@dcx.ufpb.br\",\n" +
                   "  \"password\": \"12345678\"\n" +
                   "}")
           .contentType(ContentType.JSON)
           .when().post("/auth/login")
           .then().log().all()
           .extract().path("token");
```

**Listagem 4 - Obtendo token**

Após a obtenção do *token*, ele pode ser utilizado nos testes, como ilustrado no código da Listagem 5. Neste teste, no *given()*, estamos passando o *token* que foi extraído anteriormente para obter a autorização de cadastro de um contexto. Esse *token* está sendo

passado diretamente no corpo da requisição. No *when()* está sendo chamado o *endpoint* a ser testado, que no caso é o "auth/contexts". Por fim, no *then()* está sendo verificado se o código de status (*statusCode())* é 200.

```
//Adicionar novo contexto
given().header("Authorization", "Bearer "+token)
       .body("{\n" +
       "  \"imageUrl\":
\"https://encrypted-tbn0.gstatic.com/" +
"images?q=tbn%3AANd9GcQg_Sy9i5dqOVGZcUpP078oQsWydrMrjfh_" +
               "G4W2jQRlsr0DTHLG&usqp=CAU\",\n" +
       "  \"name\": \"Parque de diversões\",\n" +
       "  \"soundUrl\": \"\",\n" +
       "  \"videoUrl\": \"\"\n" +
       "}")
   .contentType(ContentType.JSON)
       .when().post("auth/contexts")
   .then()
       .statusCode(200);
```

**Listagem 5 - Passando token de Autorização**

Um outro caso de teste semelhante ao previamente discutido é um que recebe o id do contexto que foi cadastrado (3) anteriormente, para poder fazer o cadastro de um desafio, conforme ilustrado pela Listagem 6.

```
//Adicionar desafio a um contexto
/*given()
       .header("Authorization", "Bearer "+token)
       .body("{\n" +
               "  \"imageUrl\":
\"https://www.google.com/images/bola_furada.png\",\n" +
               "  \"soundUrl\":
\"https://www.palcomp3.com/musicalega.mp3\",\n" +
               "  \"videoUrl\":
\"https://www.youtube.com/sVx219203.mp4\",\n" +
               "  \"word\": \"BOLAS\"\n" +
               "}")
       .contentType(ContentType.JSON)
           .when().post("/auth/challenges/5")
       .then()
       .statusCode(200);
```

**Figura 6 - Recebendo o id do contexto**

Um outro teste realizado, ilustrado pela Listagem 7, mostra uma consulta de um desafio cadastrado passando para isso seu id (6) para identificação do desafio. No teste verifica-se se o *statusCode* retornado é 200.

```
//Retornar um challenge cadastrado
given()
        .headers("Authorization", "Bearer "+token)
    .when()
        .get("auth/challenges/6")
    .then()
        .statusCode(200);
```

**Listagem 7 - Fazendo consulta de um desafio**

Um outro teste realizado refere-se à atualização de um desafio de usuário. No teste é passado o token que foi extraído anteriormente, passando no endpoint o id que identifica o desafio que o usuário cadastrou. Esse teste também verifica o *statusCode()* e o retorno da resposta para ver se o desafio foi atualizado, conforme ilustrado pela Listagem 8:

```
//Atualizar desafio de um usuario
given()
        .header("Authorization", "Bearer "+token)
    .contentType(ContentType.JSON)
        .body("{\n" +
                "  \"imageUrl\":
\"https://cec-a.akamaihd.net/img-prod/images/standard/po
rta-pivotante-em-madeira-macica-quartier-eucalipto-210x1
00cm-natural-cruzeiro-1309907-foto-20180405173925121_225
172_A.png\",\n" +
                "  \"soundUrl\":
\"https://cec-a.akamaihd.net/img-prod/images/standard/po
rta-pivotante-em-madeira-macica-quartier-eucalipto-210x1
00cm-natural-cruzeiro-1309907-foto-20180405173925121_225
172_A.png\",\n" +
                "  \"videoUrl\": \"teste\",\n" +
                "  \"word\": \"Porta\"\n" +
                "}")
        .when()
            .put("auth/challenges/6")
        .then().statusCode(200)
            .log().all();
}
```

**Listagem 8 - Atualização de desafio**

Com o Rest Assured ainda podem ser feitas várias validações com base nas respostas obtidas durante o envio da requisição. Os exemplos apresentados são só alguns exemplos de como começar a fazer testes com a ferramenta.

## 5. Lições Aprendidas

Nessa seção serão relatadas algumas lições aprendidas ao implementar testes usando o Postman e o REST Assured para o sistema EducAPI.

Uma lição mais geral é a de que para fazer teste de sistemas REST é importante ter bem documentada a interface do sistema a ser testado pois só assim se poderá tentar garantir que todas as funcionalidades desenvolvidas estejam de acordo com o que foi proposto e documentado na interface. Essa lição foi obtida ao trabalhar com a documentação Swagger do EducAPI e perceber que para testar algumas funcionalidades não muito bem documentadas era bastante desafiador.

Uma outra lição aprendida é que é necessário também o conhecimento da ferramenta de teste que está sendo utilizada para desenvolver os testes para o sistema, pois com o conhecimento sobre a ferramenta, seus testes irão se tornar mais legíveis para outras pessoas que irão ler o código ou olhar sua coleção (no caso do Postman) ou suíte de testes (no caso do Rest Assured). No desenvolvimento dos testes para o sistema EducAPI com as ferramentas REST Assured e com o Postman, foram aprendidas algumas lições com o passar do desenvolvimento.

Ao desenvolver os testes do sistema EducAPI com a ferramenta Rest Assured, analisando o desenvolvimento dos testes, algumas lições aprendidas foram as seguintes:

- O Rest Assured usa o formato BDD (*Behavior Driven Development)* que é uma técnica de desenvolvimento que visa integrar regras de negócios com a linguagem de programação, focando no comportamento do software e utilizando no desenvolvimento dos testes a sintaxe *given, when* e *then*, o que pareceu ser bem interessante*;*
- Ao utilizar essa sintaxe nos testes deve-se no *given* passar as pré-condições, como *token*, e *body*; no *when*, devemos chamar os métodos e passar os parâmetros quando preciso; e no *then* deveremos trabalhar com os retornos das requisições, extraindo as respostas e efetuando algumas assertivas (assertions);
- É sempre bom fazer as validações de mensagens de erro e de *status code* e no Rest Assured fazemos isso usando a sintaxe *then,* como foi mostrado em exemplos acima de validações de status code;
- Temos que estar sempre atentos à documentação, verificando se é necessário passar informações no cabeçalho ou no corpo da requisição, e usando para isso o *given* para poder passar essas informações;
- É importante estar atento para o tipo de requisição (PUT/GET/POST) utilizado em cada teste;
- Temos de desenvolver testes de uma forma organizada para que outros desenvolvedores tenham uma boa compreensão do que está sendo testado e possam manter esses testes.

Com o desenvolvimento dos testes do sistema EducAPI utilizando a ferramenta Postman, analisando o desenvolvimento dos testes, algumas lições aprendidas foram as seguintes:

- O Postman é um Cliente de APIs que facilita a criação, compartilhamento, testes e documentação de APIs (por meio da criação de coleções, por exemplo);
- Para iniciar o desenvolvimento dos testes com o Postman, começamos criando uma coleção, onde serão guardadas todas as requisições que serão feitas ao sistema sob teste (que em nosso caso foi o EducAPI);
- Dentro de cada coleção são criadas as requisições que serão usadas para fazer os testes e também se deve nomear essa requisição de acordo com o que ela faz para ajudar a equipe de desenvolvimento a saber a que se refere;
- É importante extrair da documentação da API todas as informações necessárias para se saber o que irá ser passado no corpo, no cabeçalho e quais os métodos que serão utilizados em cada requisição;
- Faz-se necessário criar variáveis de ambiente para guardar informações que serão usadas em seus testes, como url, token e informações que não irão mudar ao decorrer dos testes.

## 6. Conclusões e Trabalhos Futuros

Este artigo teve como objetivo identificar os passos para testar sistemas usando o Postman e Rest Assured, e as principais lições aprendidas, com base na experiência obtida ao testar o sistema EducAPI, um sistema desenvolvido utilizando o framework Spring.

Considerando tudo o que foi apresentado ao longo do artigo, acreditamos que o objetivo principal foi atingido, uma vez que descrevemos o processo de desenvolvimento de testes com foco no sistema EducAPI através das ferramentas apresentadas de forma que os passos possam ser seguidos por desenvolvedores fazendo testes neste e em outros serviços. Além disso, foram destacadas as lições que foram aprendidas ao decorrer do processo de criação dos testes.

Este relato contribui também para um direcionamento para a implementação dos testes com as ferramentas que foram adotadas para este trabalho e de uma certa forma dá uma direção para desenvolvedores que queiram começar a desenvolver testes utilizando uma das ferramentas para os mais diversos sistemas. Além disso, todo o código e coleções que foram criadas ao longo da escrita deste artigo estão disponíveis no repositório: https://github.com/FernandoSouzaa/TesteEducAPI.git.

Foram desenvolvidos um total de 19 (dezenove) testes utilizando a ferramenta Rest Assured e 5 (cinco) classes contendo os testes para cada tipo de requisição que seria feita ao sistema. Utilizando o Postman, foram produzidos 17 (dezessete) casos de testes para cada requisição a ser feita ao sistema, 1 (um) collection e dentro dela existem mais 3 (três) collections, onde ficaram todas as requisições que farão as chamadas à API do sistema EducAPI.

Como trabalhos futuros, pretendemos melhorar ainda mais os testes que foram desenvolvidos para este trabalho, trazendo uma melhor qualidade para o código e coleções desenvolvidas para o sistema. E além disso, esperamos que outros desenvolvedores possam contribuir para a melhoria da qualidade do sistema EducAPI e dos testes produzidos neste trabalho.

## Referências

Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. In 2007 Future of Software Engineering, pages 85–103.

Amaral, Rafael. (2019) Medium. Escrevendo os primeiros testes de API em Java usando o Rest Assured. Disponível em: https://medium.com/qa-sampa-meeting/primeiros-testes-api-rest-assured-2f4397bad355. Acesso em: 24 de agosto de 2021.

Antunes, Flavio (2021). Zup. Diferença entre fases de teste, tipos de teste e formas de execução. Disponível em: https://www.zup.com.br/blog/fases-de-teste-tipos-de-teste. Acesso em: 9 de agosto de 2021.

Barbosa, Ellen Francine et al. (2000) Introdução ao teste de software. Minicurso apresentado no XIV Simpósio Brasileiro de Engenharia de Software (SBES 2000).

Bernardo, Paulo Cheque; Kon, Fabio (2008). A importância dos testes automatizados. Engenharia de Software Magazine, v. 1, n. 3, p. 54-57.

Bernardo, Paulo Cheque (2011). Padrões de testes automatizados. Tese de Doutorado. Universidade de São Paulo.

Canaltech. O que é API. Disponível em: https://canaltech.com.br/software/o-que-e-api/. Acesso em: 06 de setembro de 2021.

Chen, T. Y. Poon, P. L. (2004) Experience with teaching black-box testing in a computer science/software engineering curriculum. IEEE Transactions on Education, v. 47,n. 01, p. 42-50.

Delamaro, Marcio; Jino, Mario; Maldonado, Jose (2013). Introdução ao teste de software. Elsevier Brasil.

Dos Santos, Márcio Carneiro (2013). Conversando com uma API: um estudo exploratório sobre TV social a partir da relação entre o twitter e a programação da televisão. Revista GEMInIS, v. 4, n. 1, p. 89-107.

Rafi, D. M.; Moses, K. R. K. Peterson, K. and Mantyla, M. V. (2012). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In Proceedings of the 7th International Workshop on Automation of Software Test, pages 36–42.

Myers, G. J.; Sandler, C.; Badgett, T.; and Thomas, T. M. (2004) The Art of Software Testing. Business Data Processing: A Wiley Series. Wiley. ISBN 9780471678359.

Khan, Mohd Ehmer et al. (2012) A comparative study of white box, black box and grey box testing techniques. Int. J. Adv. Comput. Sci. Appl, v. 3, n. 6.

Ludgério, Raimundo. et al. EducAPI: Um Sistema Colaborativo para Apoiar a Alfabetização. In: Congresso sobre Tecnologias na Educação (CTRL+E), 6. , 2021. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2021.

Manikas, K. (2016). Revisiting software ecosystems Research: A longitudinal literature study. Journal of Systems and Software, 117, 84–103. doi:10.1016/j.jss.2016.02.003.

Masse, Mark. (2011) REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. " O'Reilly Media, Inc.".

Mesquita, Gabriel. OneDayTesting. Teste de integração com Rest Assured. Disponível em: https://blog.onedaytesting.com.br/testes-de-integracao-com-rest-assured/. Acesso em: 24 de agosto de 2021.

Delamaro, M. E.; Maldonado, J. C. and Jino, M. (2007). Introdução ao teste de software. In M. E. Delamaro, M. Jino, and J. C. Maldonado, editors, Introdução ao Teste de Software, pages 1 – 7. Elsevier.

Ofoeda, Joshua; Boateng, Richard; Effah, John (2019). Application programming interface (API) research: A review of the past to inform the future. International Journal of Enterprise Information Systems (IJEIS), v. 15, n. 3, p. 76-95.

Perry, W. E. (2006) Effective methods for software testing. Indianápolis, Indiana: Third Edition, Wiley.

Pontes, Melissa Barbosa (2009). Introdução a testes de software. Engenharia de Software Magazine. Ano, v. 1, n. 11.

Rest. Wikipédia, 04 de setembro de 2021. Disponível em: https://pt.wikipedia.org/wiki/REST. Acessado 22 de setembro de 2021.

Rodrigues, Flávia. OneDayTesting. Entendendo a importância de testar sua API. Disponível em: https://blog.onedaytesting.com.br/teste-de-api/. Acesso em: 07 de setembro de 2021.

Pressman, Roger S. (2016). Engenharia de software: Uma abordagem profissional. AMGH Editora.

Silva, Enos F. T. (2021). Implementação de Testes Unitários para o Sistema EducAPI: Um Relato de Experiência. Trabalho de Conclusão de Curso. 2021.

Swagger (2021). O que é swagger. Disponível em: https://swagger.io/docs/specification/about/. Acesso em: 06 de setembro de 2021.

Teste de Sistema. Wikipédia, 04 de fevereiro de 2020. Disponível em: https://pt.wikipedia.org/wiki/Teste_de_sistema . Acesso em: 20 de outubro de 2021.

## Apêndice A - Exemplo de requisição usando o Rest Assured


ild Run Tools Git Window Help EducAPI - CriandoUsuarioTest.java
users CriandoUsuarioTest testCriarUsuario
README.md pom.xml (EducAPI) CriandoUsuarioTest.java
@Test
public void testCriarUsuario(){
//Configurar caminho de acesso a API REST
baseURI = "http://localhost";
port = 8080;
basePath = "v1/api";
//Requisição para criar usuario
given() RequestSpecification
.body("{\n" +
"  \"email\": \"fernando.ribeiro@dcx.ufpb.br\",\n" +
"  \"name\": \"Nando Ribeiro\",\n" +
"  \"password\": \"12345678\"\n" +
"}")
.contentType(ContentType.JSON)
.when()
.post( s: "/users") Response
.then() ValidatableResponse
.log().all();
}
@Test
public void testLoginUsuario(){
//Configurar caminho de acesso a API REST
baseURI = "http://localhost";
port = 8080;
basePath = "v1/api";
String token = given() RequestSpecification
.body("{\n" +
"  \"email\": \"fernando.ribeiro@dcx.ufpb.br\",\n" +
"  \"password\": \"12345678\"\n" +
"}")
.contentType(ContentType.JSON)
.when().post( s: "/auth/login") Response
.then().log().all() ValidatableResponse
.extract().path( s: "token");
System.out.println(token);
}

## Apêndice B - Exemplo de requisição usando o Postman