Programação para Internet II
Camila Nascimento de Carvalho
TSIV5A

Atividade 14 - Métodos HTTP

Parte 1: Resumo métodos HTTP

De acordo com o site Mozilla ([2022]), o protocolo HTTP (HyperText Transfer Protocol) que é responsável por transmitir dados na WEB, e que mantém uma comunicação entre cliente-servidor, define um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada para um dado recurso. Cada método implementa uma semântica diferente, mas alguns recursos são compartilhados por um grupo deles, como por exemplo, qualquer método de requisição pode ser do tipo safe, idempotente ou cacheable.

- Safe: se o método HTTP não altera o estado do servidor, ou seja se ele somente realiza a operação de leitura;
- Idempotente: se uma requisição idêntica pode ser realizada uma ou mais vezes em sequência com o mesmo efeito enquanto deixa o servidor no mesmo estado. Posto isso, todos os métodos seguros são idempotentes, mas nem todos os métodos idempotentes são seguros;
- Cacheable: se uma resposta pode ser armazenada em cache, que é armazenada para ser recuperada e usada posteriormente, salvando uma nova solicitação no servidor.

HTTP Verbs	**Descrição**
POST	É utilizado para submeter uma entidade a um recurso específico, o que frequentemente causa mudança no estado do recurso ou efeitos colaterais no servidor. Se der certo a submissão retornará o status 201. Não é idempontente e nem seguro, mas é cacheable.
GET	É utilizado para solicitar a representação de um recurso específico. As requisições com método Get devem retornar apenas dados. No caminho, o Get pode retornar uma representação em XML ou JSON e um código de resposta de 200 (OK), se der certo; em caso

	de erro, pode retornar um 404 (não encontrado) ou 400 (pedido ruim). Pode ser considerado seguro, idempotente e cacheable.
HEAD	Como o método Get, mas apenas transfere a linha de status e seção do cabeçalho. Então pode retornar uma representação em XML ou JSON e um código de resposta de 200 (OK), se der certo; em caso de erro, pode retornar um 404 (não encontrado) ou 400 (pedido ruim).
PUT	Com o método Put é possível substituir/atualizar todas as atuais representações do recurso de destino pela carga de dados da requisição. Ao atualizar com sucesso, retorna 200 (ou 204 se não estiver retornando nenhum recurso no corpo). Se estiver usando o Put para criar, retorna o status 201 ao criar com sucesso. É idempotente, mas não é seguro e nem cacheable.
DELETE	Esse método permite remover um recurso específico (identificado por um URI) requisitado. Na exclusão bem-sucedida, retorna status 201 (OK) junto com um corpo de resposta, ou retorna o status 204 sem corpo de resposta. Pode ser considerado idempontente, mas não é seguro e cacheable.
PATH	Usado para aplicar modificações parciais em um recurso. Se assemelha ao Put, mas o corpo contém um conjunto de instruções que descrevem como um recurso que residem atualmente no servidor deve ser modificado para produzir uma nova versão. Não é seguro nem idempontente, mas cacheable.
OPTIONS	Usado para descrever as opções de comunicação com o recurso de destino. Considerado seguro.
CONNECT	Esse método estabelece uma passagem para o servidor identificado pelo recurso de destino.

Observação: Os métodos mais usados são o POST, GET, PUT, PATH e DELETE, que correspondem às ações de criar, ler, atualizar e deletar (CRUD).

Parte 2: Requisição GET retornando um json

Após começar o projeto biblioteca e criar a aplicação livro, adicionar a aplicação no arquivo *settings.py*, para retornar uma requisição GET foi a realizada a seguinte modificação na views da aplicação, como apresenta a figura 1, logo para chegar na aplicação foi adicionado o caminho da aplicação no arquivo *urls.py*, como mostra a figura 2.

Figura 1- Criando a função de view

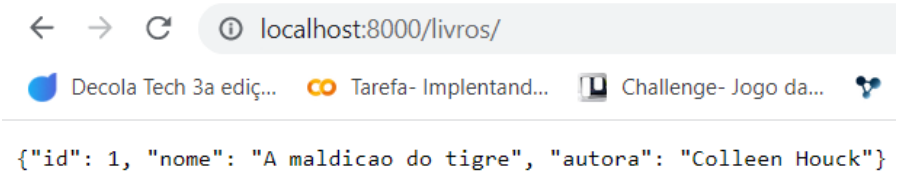
Fonte: Elaborada pela autora, 2022

Figura 2 - Configurando o caminho

Fonte: Elaborada pela autora, 2022

Posteriormente, para testar a aplicação foi startado o servidor através do comando 'python manage.py runserver', e acessada a porta 8000 + o caminho da url, como mostra a figura 3 que retorna um json:

Figura 3 - Acessando a página da aplicação



Fonte: Elaborada pela autora, 2022

- **Parte 3:** Aplicação Crud Biblioteca

Aproveitando o projeto iniciado anteriormente, foi criada uma aplicação de uma biblioteca que apresenta um crud básico, utilizando alguns métodos HTTP. Posto isso, foi criada a tabela livro no arquivo *models.py*, como mostra a figura 4:

Figura 4 - Criando a class Livros

```
from django.db import models

class Livros(models.Model):
    nome = models.CharField(max_length=150)
    autor = models.CharField(max_length=100)
    ano = models.IntegerField()
```

Fonte: Elaborada pela autora, 2022

Logo, foi criado o arquivo *form.py* a fim de pegar os campos do formulário da tabela livro, como ilustra a figura 5:

Figura 5 - Criando a class LivroForm

```
from django.forms import ModelForm
from livro.models import Livros

# Create the form class.
class LivroForm(ModelForm):
     class Meta:
            model = Livros
            fields = ['nome', 'autor', 'ano']
```

Fonte: Elaborada pela autora, 2022

Após isso, as rotas da aplicação foram adicionadas no arquivo urls, como ilustra a figura 6:

Figura 6 - Criando os caminhos no arquivo urls

```
    from django.contrib import admin
    from django.urls import path
    from livro.views import home, form, create, view, edit, update,
delete

    urlpatterns = [
        path('admin/', admin.site.urls),
        path('', home, name='home'),
        path('form/', form, name='form'),
        path('create/', create, name='create'),
        path('view/<int:pk>/', view, name='view'),
        path('edit/<int:pk>/', edit, name='edit'),
        path('update/<int:pk>/', update, name='update'),
        path('delete/<int:pk>/', delete, name='delete')
    ]
```

Fonte: Elaborada pela autora, 2022

Posteriormente, as funções de views foram criadas, como apresenta a figura 7:

Figura 7 - Criando as funções de views

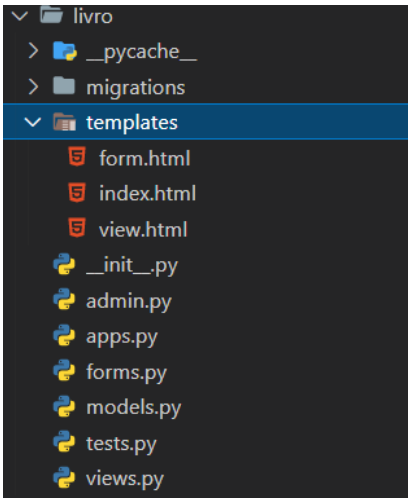
```
    from django.shortcuts import redirect, render
    from livro.forms import LivroForm
    from livro.models import Livros


    def home(request):
        data= {}
        data['db'] = Livros.objects.all()
        return render(request, 'index.html', data)

    def form(request):
         data = {}
         data['form'] = LivroForm
         return render(request, 'form.html', data)


    def create (request):
```

```
        form = LivroForm(request.POST or None)
        if form.is_valid():
            form.save()
            return redirect('home')


    def view(request, pk):
        data ={}
        data['db'] = Livros.objects.get(pk=pk)
        return render(request, 'view.html', data)



    def edit(request, pk):
        data ={}
        data['db']= Livros.objects.get(pk=pk)
        data['form'] = LivroForm(instance=data['db'])
        return render(request, 'form.html', data)

    def update(request, pk):
        data ={}
        data['db']= Livros.objects.get(pk=pk)
        form = LivroForm(request.POST or None, instance=data['db'])
        if form.is_valid():
            form.save()
            return redirect('home')

    def delete(request, pk):
        db = Livros.objects.get(pk=pk)
        db.delete()
        return redirect('home')
```

Fonte: Elaborada pela autora, 2022

E não menos importante, foi criada uma pasta com nome template com propósito de colocar as páginas que seriam renderizadas quando o servidor é iniciado, como mostra a figura 8:

Figura 8 - Criando a pasta template



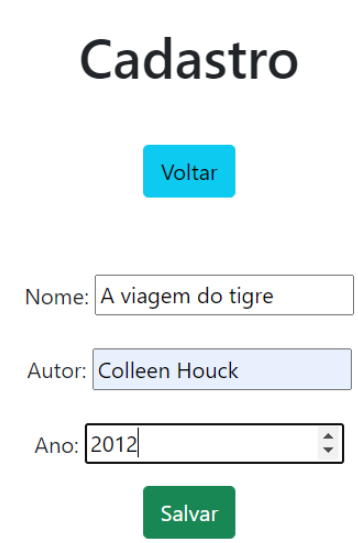
Fonte: Elaborada pela autora, 2022

Posto isso, iniciou-se o servidor através do comando *'python manage.py runserver'* dentro do diretório do projeto, logo, as figuras 9, 10, 11, 12 e 13 apresenta as telas da aplicação com CRUD:

Figura 9 - Buscando os cadastrados

Fonte: Elaborada pela autora, 2022

Figura 10 - Cadastrando um livro

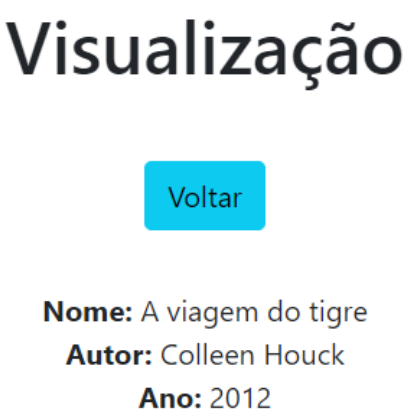


Fonte: Elaborada pela autora, 2022

Figura 11 - Livro cadastrado

Fonte: Elaborada pela autora, 2022

Figura 12 - Visualização do livro cadastrado



Fonte: Elaborada pela autora, 2022

Figura 13 - Livro deletado

Fonte: Elaborada pela autora, 2022

Referências Bibliográficas:

MOZILLA. Methods. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods. Acesso em: 27 de junho de 2022.

Rest API tutorial. Using HTTP Methods for restful services. Disponível em: https://www.restapitutorial.com/lessons/httpmethods.html. Acesso em: 27 de junho de 2022.

LIMA, Thiago. O que você precisa saber sobre os métodos e códigos de status do protocolo HTTP. Disponível em: https://thiagolima.blog.br/api-restful-o-que-voc%C3%AA-precisa-saber-sobre-m%C3%A9todos-e-c%C3%B3digos-de-status-do-protocolo-http-af48d278a2e9. Acesso em: 27 de junho de 2022.