

Atualizando usuário

A função de atualizar, acaba sendo a mais complexa, e requer atenção, então leia as anotações !
tudo começa no **serializers.py**

```
class UsuarioUpdateSerializer(serializers.ModelSerializer):

    class Meta:
        model = Usuario
        fields = '__all__'

        nome = serializers.CharField(required=False)
        sobrenome = serializers.CharField(required=False)
        idade = serializers.IntegerField(required=False)
        foto = serializers.ImageField(required=False)
```

Criamos esse novo serializador, para que quando o usuário for atualizar, ele não seja obrigado a atualizar todos os campos!

Em views.py

Adicionamos aos nossos imports esse novo serializador.

```
from .serializers import UsuarioSerializer, UsuarioUpdateSerializer
```

Em views.py:

E agora criamos a nossa VIEW para atualizar o usuário:

```
6  class UpdateUsuario(generics.UpdateAPIView):
7      # Define a consulta para recuperar todos os usuários do banco de dados
8      queryset = Usuario.objects.all()
9
10     # Define o serializador a ser utilizado para serializar e desserializar os dados do usuário
11     serializer_class = UsuarioSerializer
12
13     # Método GET para exibir o formulário de atualização do usuário
14     def get(self, request, *args, **kwargs):
15         # Obtém o ID do usuário a ser atualizado dos parâmetros da URL
16         user_id = kwargs.get('pk')
17         # Obtém o usuário do banco de dados ou retorna um erro 404 se não encontrado
18         usuario = get_object_or_404(Usuario, pk=user_id)
19         # Inicializa o serializador com a instância do usuário
20         serializer = self.serializer_class(instance=usuario)
21         # Renderiza o template 'atualizarusuario.html', passando o usuário e o serializador como contexto
22         return render(request, 'atualizarusuario.html', {'serializer': serializer, 'usuario': usuario})
23
24     # Método POST para processar o formulário de atualização do usuário
25     def post(self, request, *args, **kwargs):
26         # Obtém o ID do usuário a ser atualizado dos parâmetros da URL
27         user_id = kwargs.get('pk')
28         # Obtém o usuário do banco de dados ou retorna um erro 404 se não encontrado
29         usuario = get_object_or_404(Usuario, pk=user_id)
30         # Inicializa o serializador de atualização com a instância do usuário e os dados da requisição
31         serializer = UsuarioUpdateSerializer(instance=usuario, data=request.data, partial=True)
32         # Verifica se os dados do formulário são válidos
33         if serializer.is_valid():
34             # Salva as alterações do usuário no banco de dados
35             serializer.save()
36             # Redireciona para a página de listagem de usuários após a atualização bem-sucedida
37             return redirect('listar')
38         # Renderiza o template 'atualizarusuario.html' com o serializador e o usuário como contexto
39         return render(request, 'atualizarusuario.html', {'serializer': serializer, 'usuario': usuario})
```

Reparam que resgatamos a chave primaria(pk) do usuário, isso será importante na hora de criar nossa url.

Em URLs.PY

```
urlpatterns = [
    path('',views.CadastrarUsuario.as_view(),name='cadastro'),
    path('listar_usuarios',views.ListarUsuario.as_view(),name='listar'),
    path('usuario/<int:pk>/atualizar',views.UpdateUsuario.as_view(),name='update')
```

Na URL que criamos, passamos também a PK do nosso usuário, ela é passada como um inteiro, teriam outras formas de fazer isso como por exemplo, ao invés de usar a PK, usar o ID do usuário, dessa forma: <int:id> ou se preferir usar o nome, ou outra string você pode usar<str:nome> mas para o que estamos fazendo, usaremos a PK

No HTML

```
{% extends 'base.html' %}

{% block dashboard %}
<h1>Atualizar Usuário</h1>
<form method="post" enctype="multipart/form-data" class = 'form-control'>
    {% csrf_token %}
    <center>
        <input type="text" name="nome" value="{{ serializer.nome.value }}" class='form-control' ><br>
        <input type="text" name="sobrenome" value="{{ serializer.sobrenome.value }}" class='form-control'><br>
        <input type="text" name="idade" value="{{ serializer.idade.value }}" class='form-control'><br>
        <br>
        <input type="file" name="foto"><br>
        <button type="submit">Atualizar</button>
    </center>
</form>
<a href="{% url 'listar' %}">Voltar para a Lista de Usuários</a>
{% endblock %}
```

Apenas exibimos um formulário com os campos que o usuário pode atualizar, repare que quando resgatamos a imagem, usamos um **.URL** para a imagem aparecer renderizada, caso fizer o contrario, será resgatado o nome da imagem.

A tela de atualizar :

Atualizar Usuário

usuario	
teste	
20	



Nenhum arquivo escolhido

[Voltar para a Lista de Usuários](#)

Deletando Usuario

Em views.py

```
# Define uma classe de visualização baseada em genéricos para excluir um usuário
class DeleteUsuario(generics.DestroyAPIView):
    # Define a consulta para recuperar todos os usuários do banco de dados
    queryset = Usuario.objects.all()

    # Define o serializador a ser utilizado para serializar e desserializar os dados do usuário
    serializer_class = UsuarioSerializer

    # Método GET para exibir a página de confirmação de exclusão do usuário
    def get(self, request, *args, **kwargs):
        # Obtém o ID do usuário a ser excluído dos parâmetros da URL
        user_id = kwargs.get('pk')
        # Obtém o usuário do banco de dados ou retorna um erro 404 se não encontrado
        usuario = get_object_or_404(Usuario, pk=user_id)
        # Inicializa o serializador com a instância do usuário
        serializer = self.serializer_class(instance=usuario)
        # Renderiza o template 'listarusuario.html', passando o usuário e o serializador como contexto
        return render(request, 'listarusuario.html', {'serializer': serializer, 'usuario': usuario})

    # Método POST para processar a exclusão do usuário
    def post(self, request, *args, **kwargs):
        # Obtém o ID do usuário a ser excluído dos parâmetros da URL
        user_id = kwargs.get('pk')
        # Obtém o usuário do banco de dados ou retorna um erro 404 se não encontrado
        usuario = get_object_or_404(Usuario, pk=user_id)
        # Exclui o usuário do banco de dados
        usuario.delete()
        # Redireciona para a página de listagem de usuários após a exclusão bem-sucedida
        return redirect('listar')
```

A função de deletar herda muita coisa da função de atualizar, porém acaba sendo bem mais simples.

Em urls.py

```
path('usuario/<int:pk>/atualizar',views.UpdateUsuario.as_view(),name='update'),  
path('usuario/<int:pk>/deletar',views.DeleteUsuario.as_view(),name='deletar')
```

Como podem ver, a URL é basicamente a mesma, e não há passos adicionais, como criar um serializador ou algo do gênero, agora só precisamos atribuir a URL ao botão

Passando urls para o html

```
    {{ usuario.nome }} {{ usuario.sobrenome }} idade : {{ usuario.idade }} anos
    <a href="{% url 'update' usuario.id %}" class='btn btn-success'>Editar</a>

    <form method="post" action="{% url 'deletar' usuario.id %}" style="display:inline;" >
        {% csrf_token %}
        <button type="submit" class='btn btn-danger'>Deletar</button>
    </form>
```

URLS que precisam do id, ou PK precisam que eles sejam passados
como é o caso das URLs atualizar e deletar

Passando urls para o html

```
<li class="nav-item active">
|   <a class="nav-link" href="{%url 'cadastro'%}">Cadastrar usuario <span class="sr-only"></span></a>
</li>
<li class="nav-item">
|   <a class="nav-link" href="{%url 'listar'%}">Gerenciar usuarios</a>
</li>
```

Já urls que não precisamo do ID como as de cadastro e listagem, pode ser colocadas apenas nas tags do django : `{%url 'suaURL'%}`