CURSO DE DJANGO (CURSO ALURA: INTRODUÇÃO AO DJANGO 3: MODELS, ROTAS E VIEWS E OUTROS CURSOS)

PASSO BÁSICO PARA INICIAR UM PROJETO:

1. Crie uma pasta geral para seu projeto, dentro dessa pasta vamos instalar o ambiente virtual utilizando o virtualenv, que pode ser instalado através do pip install virtualenv.

2. Na pasta do projeto, digita no terminal: “virtualenv NomeAmbienteVirtual”, sugestão de nome: 'venv'.

4. Depois de criado o ambiente virtual é necessário ativá-lo, então entra na pasta do ambiente virtual pelo caminho PastaDoProjeto/venv/scripts, depois digitar o comando “./activate”. Pronto, agora o ambiente virtual estará ativado.

OBS: TALVES SEJA PRECISO ENTRAR NO WINDOWS POWERSHELL (NO WINDOWS 10 BASTA PROCURA NA BARRA DE PESQUISA), DIGITAR O COMANDO Set—ExecutionPolicy RemoteSigned, sepois responder o sistema com A ou S.

5. Já dentro do ambiente virtual criado, digita no terminal “pip install Django” para instalar o Django lá no ambiente.

6. Criar novo projeto dentro do ambiente virtual digitando no cmd: “jango-admin startproject NomeDoProjeto”.

7. Ativar o servidor local embutido com o comando no cmd “python manage.py runserver

8. Já é possui acessar alguma coisa do projeto no navegador através do endereço: localhost:8000

9. No Code, abra a pasta onde está o projeto, no arquivo settings.py já podemos alterar a linguagem e o fuso horário do projeto editando os itens LANGUAGE: 'pt-br' e TIME\_ZONE: 'America/Recife'.

10. Para garantir que vamos usar o Django que esta na venv, no CODE digite o Command + Shift + P", e digite: python select interpreter, selecione o que tem no final 'venv': venv.

11. A pasta principal do projeto (no caso AlexReceitas), contém os arquivos que chamamos arquivos de configuração, para criar um aplicativo novo devemos digitar no terminal 'python manage.py startapp receitas'. Com isso criamos arquivos e estruturas de pastas de um novo app.

12. No arquivo settings.py dos arquivos de configurações devemos registrar o aplicativo recém criado, para tanto tente localizar o linha INSTALLED\_APPS e acrescente nosso app Receitas à lista.

13.No app receitas, cria um arquivo chamado urls.py, dentro dele escrevemos os códigos abaixo e salvamos o arquivo:

from django.urls import path

from . import views

from django.urls import path

urlpatterns = [

path('', views.index, name='index')

]

14. No app receitas, cria um arquivo chamado views.py, e escreva a os códigos abaixo e salvamos o arquivo:

from django.shortcuts import render

from django.http import HttpResponse

def index(request):

return HttpResponse('<h1>Receitas</h1>') (apenas um teste, será substituído pela renderização do um arquivo html)

15. No arquivo urls.py das configurações, faça o include das urls do app Receitas com o seguinte código:

a) importe o include: from django.urls import path, include

b) urlpatterns = [

path('', include('receitas.urls')),

path('admin/', admin.site.urls),

]

SALVE O ARQUIVO

16.Dentro do aplicativo Receitas criamos um pasta chamada Templates e dentro desta pasta um arquivo index.html. Dentro do arquivo digitamos html:5, e automaticamente o Djanto cria a estrutura básica de uma página html.

17. Na tag <body> adicionamos a tag <h1>Alura Receitas</h1> e <h2>Bem vindos ao Alura Receitas</h2>. (Poderia digitar qualquer outro texto, pois tem apenas a função de testar a renderização);

18. No arquivo views.py do app Receitas editamos assim:

def index(request):

return render(request, 'index.html')

19. No arquivo index.html, dentro da tag <body> colocamos o seguinte código:

<table>

<thead>

<tr>

<td>Nome da receita</td>

<td>Ingredientes</td>

<td>Modo de preparo</td>

</tr>

</thead>

<tbody>

<tr>

<td>Suco verde</td>

<td>Folhas verdes, 1 maça, 1 cenoura</td>

<td>Bata tudo no liquidificador</td>

</tr>

</tbody>

</table>

20. Precisamos dizer ao Django onde estão os arquivos de template, então no arquivo settings.py dos arquivos de configuração editamos a linha template:

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [os.path.join(BASE\_DIR, 'Receitas/templates')],

},

]

21. No ínicio do arquivo settings.py, escreva: import os.

22. Ao final do arquivo settings.py também acrescentamos, as seguintes linhas de códigos:

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'AlexReceitas/static')

]

23. Dentro da pasta de configurações do projeto, criamos uma nova pasta chamada STATIC. Dentro dela colamos os arquivos "css", "fonts", "img", "js", "scss" e o arquivo "site.css", todos baixados através do link: <https://caelum-online-public.s3.amazonaws.com/1489-django+parte+1/site_receitas.zip>;

24. Digitamos no terminal o comando 'python manage.py collectstatic', isto fará o python copiar os arquivos estáticos para sua memória.

25. A partir dos arquivos baixamos copiamos index.html e receita.html para dentro da pasta templates.

26. Na primeira linha do arquivo index.html acrescente o seguinte código python: {% load static %}, isso fará que o interpretador saiba que existem arquivos estáticos para serem usados.

27. No arquivo index.html onde houve referência a alguma imagem e javascripts, para ser exibido um visual mais agradável, podemos envolver o endereço da imagem ou javascripts com o código python "{% static 'imagem'%}", dessa forma vamos ter uma visualização tratada por arquivo \*.css e \*.js;

28. No arquivo index.html, logo abaixo do <!--Logo-->, devemos envolver o index.html com a tag do django "{% url 'index' %}" dentro da class navbar, para que sejamos redirecionados para o página index.html.

29.Ainda não temos um rota para receitas, para isso adicionamos uma linha no urlpatterns de urls.py do aplicativo receitas, conforme a seguir:

urlpatterns = [

path('', views.index, name='index'),

path('receita', views.receita, name='receita')

]

30. Já que as urls apontam para um caminho "receita", temos que criar a view correspondente dentro do views.py do aplicativo receitas, conforme a seguir:

def index(request):

return render(request, 'index.html')

def receita(request):

return render(request, 'receita.html')

31. No arquivo index.html devemos editar o <!--Nav Start--> com código a seguir:

<div class="classynav">

<ul>

<li><a href="{% url 'index' %}">Home</a></li>

<li><a href="{% url 'receita' %}">Receitas</a></li>

<ul>

</div>

32. Salvar tudo e abrir o localhost:8000 no navegador;

33. Dentro da pasta "templates", criamos um novo arquivo chamado base.html. Em index.html, recortamos todo o intervalo de código desde load static até <body> ficando somente a partir de Preloader, e colamos no novo arquivo. Em seguida, recortamos todo o trecho final de JavaScript em All Javascript Files e colamos em base.html. Deletar os mesmo trechos no arquivo receitas.html;

34. No base.html acrescentamos {% block content %} {%endblock%} abaixo de <body> indicando que naquele espaço poderá inserido partes códigos de outra página.

35. Nos arquivos index.html e receita.html precisamos acrescentar a linha a seguir acima de '{% load static %}' indicando que vamos trabalhar com arquivos estáticos e que vamos extender um arquivo base.html existente.

{% extends 'base.html' %}

{% load static %}

Devemos delimitar dentro dos arquivos as partes que serão utilizadas dentro do base.html, fazemos isso utilizando o código python (tag do python), colocando o conteúdo {% block content %} e {%endblock%}.

36. Dentro da pasta templates criamos um pasta partials e dentro dele criamos dois arquivos 'menu.html' e 'footer.html'. Dentro do menu.html colamos a parte do cabeçario da página lá de index.html ( que começa em <header> e vai até </header>. No arquivo footer.html colocamos a parte do rodapé da página index.html (começa em <footer> e termina em </footer>. No começo de cada arquivo, menu e footer, colamos a tag django {% load static %} e salvamos. Voltando lá em index.html e receita.html apagamos toda a parte do cabeçário e rodapé e no lugar digitamos {% include 'partials/menu.html' %} e {% include 'partials/footer.html' %}.

35. Agora vamos editar o arquivo models.py de tasks escrevendo os códigos/instruções, que através do migration, serão posteriormente transferidos para um banco de dados:

from django.db import models

from datetime import datetime

from django.db.models.fields import DateTimeField

class Receita(models.Model):

nome\_receita=models.CharField(max\_length=200)

ingredientes=models.TextField(default='')

modo\_preparo= models.TextField(default='')

tempo\_preparo=models.IntegerField(default=0)

rendimento=models.CharField(max\_length=100)

categoria=models.CharField(max\_length=100)

date\_receita=models.DateTimeField(default=datetime.now, blank=True)

36. Digite no terminal 'python manage.py makemigrations', depois digite 'python migrate';

37. No arquivo admin.py, edite com o seguinte código:

from django.contrib import admin

from .models import Receita

admin.site.register(Receita)

38. Com o servidor rodando, no navegador no endereço localhost:8000/admin, você terá acesso ao painel administrativo criado automaticamente pelo Django, para criar login e senha, no code digite “python manage.py createsuperuser”. Para acessar seu terminal administrativo digite no navegador: localhost:8000/admin;

39. Agora entrar no localhost:8000/admin e clicar em +adicionar receita, cadastre lá no mínimo 3 receitas simples;

40. No arquivo view.py do aplicativo Receitas, insira a seguinte função:

def index(request):

receitas=Receita.objects.all

dados={

'receitas':receitas

}

return render(request, 'index.html', dados)

41. No terminal execute o comando 'pip install pylint-django' para instalar o pylint. Na pasta vscode das configurações, acréscente a linha a seguir no arquivo settings.json:

"python.linting.pylintArgs": ["--load-plugins=pylint\_django"],

40. No arquivo index.html apague 2 das 3 sequências de código a seguir:

<!-- Single Best Receipe Area -->

<div class="col-12 col-sm-6 col-lg-4">

<div class="single-best-receipe-area mb-30">

<img src="{% static 'img/bg-img/foto\_receita.png'%}" alt="">

<div class="receipe-content">

<a href="{% url 'receita' %}">

<h5>Nome da receita</h5>

</a>

</div>

</div>

</div>

41. Agora edite o arquivo index.html inserindo a tags do Django para fazer ele buscar os dados gravados no nosso banco de dados, acréscente as linhas em vermelho no arquivo index.html:

**{% if receitas %}**

**{% for receita in receitas %}**

<!-- Single Best Receipe Area -->

<div class="col-12 col-sm-6 col-lg-4">

<div class="single-best-receipe-area mb-30">

<img src="{% static 'img/bg-img/foto\_receita.png'%}" alt="">

<div class="receipe-content">

<a href="{% url 'receita' %}">

<h5>{{receita.nome\_receita}}</h5>

</a>

</div>

</div>

</div>

**{% endfor %}**

**{% else %}**

**{% endif %}**

42. Na linha <a href="{% url 'receita' %}"> do código acima acréscente um receita.id, ficando agora assim: <a href="{% url 'receita' receita.id %}">, isso fará com que cada link de receita tenha um id (que será número inteiro);

43. Agora alterar o arquivo urls.py do aplicativo receitas, modificando a url de receita da seginte forma:

path('<int:receita\_id>', views.receita, name='receita');

44. No arquivo views.py do aplicativo receitas alterar a view de receita, simplesmente acrescentando receita\_id no parenteses junto com o request:

def receita(request, receita\_id):

return render(request, 'receita.html')

45. É necessário apagar linha que contem <li><a href="{% url 'receita' %} ">Receitas</a></li> do arquivo menu.html contido na pasta templates/partiais do aplicativo receitas;

46. Agora a def receita do arquivo views.py do aplicativo receitas deve ficar assim:

def receita(request, receita\_id):

receita=get\_object\_or\_404(Receita, pk=receita\_id)

receita\_a\_exibir={

'receita':receita

}

return render(request, 'receita.html', receita\_a\_exibir)

OBS: Importar o 'get\_object\_or\_404' acrescentando esse código na minha de importação no ínicio do arquivo.

47. No nosso arquivo template receita.html devemos procurar dentro do html os nomes dos campos que devemos substituir pelos dados gravados no nosso banco de dados, por exemplo1, onde tem:

<span>Data da Receita</span>

devemos substituir por:

<span>{{receita.date\_receita}}</span>

e exemplo2, onde temos

<h2>Nome da Receita</h2>

devemos escrever:

<h2>{{receita.nome\_receita}}</h2>

Para as tarefas criadas apareçam com seus nomes (titles), será necessário editar o arquivo o arquivo models.py criando uma def \_\_(str)\_\_(self): que diz => return self.title.

48. Editar o arquivo admin.py do aplicativo Receitas para melhorar o grud localhost:8000/admin, até ficar com a seguinte configuração:

from django.contrib import admin

from .models import Receita

class ListandoReceitas(admin.ModelAdmin):

list\_display =('id', 'nome\_receita', 'categoria', 'tempo\_preparo')sou lindo :) codigos grandes

list\_display\_links=('id', 'nome\_receita')

search\_fields=('nome\_receita',)

list\_filter=('categoria',)

list\_per\_page=3

admin.site.register(Receita, ListandoReceitas)

49. Usando o comando python manage.py startapp pessoas criamos um novo aplicativo chamado 'pessoas', que permitir cadatrar os editores de receitas. No aplicativo de configuração (AlexReceitas), arquivo settings.py, procuramos o campo INSTALLED\_APPS e acrescentamos no nosso aplicativo 'pessoas'; (Este app é apenas didático, será deletado no futuro e no seu lugar escrevemos o app usuários.)

50. Agora dentro do aplicativo 'pessoas', editamos o arquivo models.py, da seguinte forma:

from django.db import models

class Pessoa(models.Model):

nome = models.CharField(max\_length=200)

email = models.CharField(max\_length=200)

51. Ainda dentro do aplicativo 'pessoas', agora editamos o arquivo admin.py da seguinte forma:

from django.contrib import admin

from .models import Pessoa

admin.site.register(Pessoa)

52. Depois de salvar as alterações deve preparar as migrações com 'python manage.py makemigrations'. Em seguida fazer a migração de fato com 'python manage.py migrate';

53. Podemos melhorar a estrutura do localhost:8000/admin/pessoas/pessoa/, devemos editar o arquivo admin.py do aplicativo pessoas assim:

from django.contrib import admin

from .models import Pessoa

class ListandoPessoas(admin.ModelAdmin):

list\_display = ('id', 'nome', 'email')

list\_display\_links = ('id', 'nome')

search\_fields = ('nome',)

list\_per\_page = 2

admin.site.register(Pessoa, ListandoPessoas)

50. Vamos agora criar a integração entre os apps 'Receitas' e 'pessoas', para começar vamos apagar todas as receitas cadastradas, para evitar erros;

51. Editar o arquivo models.py do aplicativo 'Receitas', acrescentando as linhas destacadas:

from django.db import models

from datetime import datetime

**from pessoas.models import Pessoa**

class Receita(models.Model):

**pessoa = models.ForeignKey(Pessoa, on\_delete=models.CASCADE)**

nome\_receita = models.CharField(max\_length=200)

ingredientes = models.TextField()

modo\_preparo = models.TextField()

tempo\_preparo = models.IntegerField()

rendimento = models.CharField(max\_length=100)

categoria = models.CharField(max\_length=100)

date\_receita = models.DateTimeField(default=datetime.now, blank=True)

52. Novamente deve ser executado, no terminal, o código 'python manage.py makemigration', em seguida jogar para o banco de dados com o código 'python manage.py migrate', depois salvar o arquivo;

53. Para que a no grud seja exibido a nome da pessoa, devemos editar o arquivo models.py do aplicativo 'pesssoas' com as linha em vermelho:

from django.db import models

class Pessoa(models.Model):

nome = models.CharField(max\_length=200)

email = models.CharField(max\_length=200)

def \_\_str\_\_(self):

return self.nome

54. Na pasta templastes, editar o arquivo receita.html, onde tem o alinha <h6>Por: </h6>, alterar para <h6>Por: {{receita.pessoa}}</h6>;

55. Para poder escolher que receita estão publicadas e quais não estão, devemos acrescentar uma nova linha no arquivo models.py do aplicativo receitas:

publicada=models.BooleanField(default=False )

56. Depois rodar no terminal o: "python manage.py makemigrations" e "python manage.py migrate";

57. Para selecionar e exibir somente as receitas que estão com o campo 'publicada' marcado, devemos editar o arquivo views.py do aplicativo receitas da seguinte maneira:

def index(request):

receitas=Receita.objects.order\_by( '-date\_receita').filter(publicada=True)

dados={

'receitas':receitas

}

return render(request, 'index.html', dados)

\* O 'order\_by('-date\_receita')', ordena as receitas pela data mais recente (sinal de menos);

\* O filter(publicada=True), seleciona as receitas que estão publicadas;

58. Para poder editar a publicação da receita diretamente na lista de receitas, sem precisar entrar em cada receita, basta editar o arquivo admin.py do aplicativo receitas:

class ListandoReceitas(admin.ModelAdmin):

list\_display = ('id', 'nome\_receita', 'categoria', 'tempo\_preparo', 'publicada')

list\_display\_links = ('id', 'nome\_receita')

search\_fields = ('nome\_receita',)

list\_filter = ('categoria',)

list\_editable = ('publicada',)

list\_per\_page = 5

59. Agora queremos que casa receita mostre uma imagem associada a ela. Para começar vamos editar o arquivo models.py do aplicativo receitas, inserindo:

from django.db import models

from datetime import datetime

from pessoas.models import Pessoa

class Receita(models.Model):

pessoa = models.ForeignKey(Pessoa, on\_delete=models.CASCADE)

nome\_receita = models.CharField(max\_length=200)

ingredientes = models.TextField()

modo\_preparo = models.TextField()

tempo\_preparo = models.IntegerField()

rendimento = models.CharField(max\_length=100)

categoria = models.CharField(max\_length=100)

date\_receita = models.DateTimeField(default=datetime.now, blank=True)

foto\_receita = models.ImageField(upload\_to='fotos/%d/%m/%Y', blank=True, null=True, default='0')

publicada models.BooleanField(default=False)

\* Importante que o campo foto\_receita não vai armazenar de fato a foto, mas sim o caminho para chegar até ela

\* O caminho da foto será gravado como nesse exemplo: 'fotos/25/07/2021/sopa.jpg'

59. Salvar o arquivo models.py e no terminal digitar os comandos: "python manage.py makemigrations" e "python manage.py migrate";

60. Necessário fazer alguns ajustes no final do arquivo settings.py do aplicativo de configurações:

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'AlexReceitas/static')

]

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

MEDIA\_URL = '/media/'

61. Agora é necessário acessar o arquivo urls.py do aplicativo de configuração (AlexReceitas) e fazer a seguinte alteração:

from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('', include('receitas.urls')),

path('admin/', admin.site.urls),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

\*Isso acima seria tipo para dizer ao django que pode usar esses arquivos de mídia.

62. Dentro do arquivo index.html escrever um 'if' que vai mostrar a imagem, caso a receita possua, e vai mostrar a imagem padrão caso receita ainda não possua uma imagem:

<!-- ##### Best Receipe Area Start ##### -->

<section class="best-receipe-area">

<div class="container">

<div class="row">

{% if receitas %}

{% for receita in receitas %}

<!-- Single Best Receipe Area -->

<div class="col-12 col-sm-16 col-lg-4">

<div class="single-best-receipe-area mb-30">

{% if receita.foto\_receita == '0' %}

<img src="{% static 'img/bg-img/foto\_receita.png' %}">

{% else %}

<img src="{{ receita.foto\_receita.url }}" alt="">

{% endif %}

<div class="receipe-content">

<a href="receita.html">

<h5>{{ receita.nome\_receita }}</h5>

</a>

</div>

</div>

</div>

{% endfor %}

{% else %}

{% endif %}

</div>

</div>

</section>

63. Agora fazer um 'if' bem parecido dentro do arquivo receita.html:

<!-- Receipe Slider -->

<div class="container">

<div class="row">

<div class='col-12">

<div class "receipe-slider owl-carousel">

{% if receita.foto\_receita == '0' %}

<img src="<% static 'img/bg-img/tomate\_banner.jpg' %}"

{% else %}

<img src="{{ receita.foto\_receita.url }}" alt="">

{% endif %}

</div>

</div>

</div>

</div>

64. Agora vamos implementar um mecanismo de busca de receitas. Já temos o botão buscar que abre um campo de buscar, se digitarmos algo e clicar em buscar vai aparecer uma página de erro porque ainda não programamos essa função. Para a buscar funcionar começamos editando o arquivo urls.py do aplicativo receitas, acrescentando um novo path, conforme a linha destacada:

from django.urls import path

from . import views

urlpatterns = [

path('', views.index, name='index'),

path('<int:receita\_id>', views.receita, name='receita'),

path('buscar', views.buscar, name='buscar')

]

Dessa forma podemos chamar essa url apenas pelo nome 'buscar'

65. No arquivo views.py da mesma pasta, criamos a def 'buscar' para renderizar o template buscar.html:

def buscar(request):

return render(request, 'buscar.html')

66. Dentro da pasta templates criamos um nome arquivo com nome buscar.html, dentro dele copia e colamos todo o conteúdo do arquivo index.html;

67. No arquivo index.html precisamos fazer as seguintes alterações:

<!-- Search Wrapper -->

<div class="search-wrapper">

<!-- Close Btn -->

<div class="close-btn"><i class= "fa fa-times" aria-hidden="true"></i></div>

<div class "container">

<div class="row">

<div class="col-12">

<form action="{% url 'buscar' %}">

<input type="text" name="buscar" placeholder="O que está procurando..."></input>

<button type="submit"><i class="fa fa-search" aria hidden="true"></i></button>

</form>

</div>

</div>

</div>

</div>

68. No arquivo buscar.html devemos procurar o segundo ELSE, em seguida colar o a linha a seguir abaixo do ELSE:

<p>Receita não encontrada</p>

69. Agora continuamos editamos o arquivo views.py conforme a seguir:

def buscar(request):

lista\_receitas = Receita.objects.order\_by('-date\_receita').filter(publicada=True)

if 'buscar' in request.GET:

nome\_a\_buscar = request.GET['buscar']

if nome\_a\_buscar:

lista\_receitas = lista\_receitas.filter(nome\_receita\_\_icontains=nome\_a\_buscar)

dados = {

'receitas' : lista\_receitas

}

return render(request, 'buscar.html', dados)

70. Na página receita.html copiamos as linhas destacadas em vermelho e colamos no mesmo ponto da página index.html:

<!-- Search Wrapper -->

<div class="search-wrapper">

<!-- Close Btn -->

<div class="close-btn"><i class= "fa fa-times" aria-hidden="true"></i></div>

<div class "container">

<div class="row">

<div class="col-12">

<form action="{% url 'buscar' %}">

<input type="text" name="buscar" placeholder="O que está procurando..."> </input>

<button type="submit"><i class="fa fa-search" aria hidden="true"></i></button>

</form>

</div>

</div>

</div>

</div>

71. Agora para finalizar, na pasta templates/partials do aplicativo receitas, criamos um novo arquivo chamado busca.html, dentro desse novo arquivo escrevemos uma tag do django "{% load static %}"e colamos o <!-- Search Wrapper --> que pode ser recortado da página receita.htm, o resultado final deve ser esse:

"{% load static %}

<!-- Search Wrapper -->

<div class="search-wrapper">

<!-- Close Btn -->

<div class="close-btn"><i class="fa fa-times" aria-hidden="true"></i></div>

<div class="container">

<div class="row">

<div class="col-12">

<form action="{% url 'buscar' %}">

<input type="text" name="buscar" placeholder="O que está procurando...">

<button type="submit"><i class="fa fa-search" aria-hidden="true"></i></button>

</form>

</div>

</div>

</div>

</div>

72. No arquivo receita.html, onde tinha o código acima escreve simplesmente:

{% include 'partials/busca.html' %}

73. Nos arquivos index.html e buscar.html, procuramos por '<!-- Search Wrapper -->', ai apagamos tudo e colamos no lugar:

{% include 'partials/busca.html' %}

74. Para finalizar, para ficar mais aderente a nomeclarura utilizada pela maioria dos programadores, alteramos o nome dos arquivos partials, acrescentando um underline antes de cada nome: \_menu.html, \_footer.html e \_buscar.html. Depois procuramos nas páginas html onde há referências as partials e alteramos os nomes dos arquivos partials. (obs: isso pode ser feito logo no início quando começamos usar os partials).

75. Vamos agora criar maneiras do usuário se cadastrar e fazer login na aplicação. A idéia é a pessoa fazer cadastro/login e criar suas próprias receitas e poder visualizá-las na sua própria página de receitas. O admin terá acesso a todas as receitas cadastradas, caso goste de alguma receita e queira publicá-la na sua página principal, basta acessar o /admin/receitas e marcar a caixa publicar aquela receita.

76. Para começar vamos criar um novo app 'usuarios' como comando no terminal 'python manage.py startapp usuarios'. Depois temos que registar o novo aplicativo no settings.py do aplicativo de principal. No settings.py procuramos INSTALLED APPS, e acrescentamos o app 'usuarios' a lista. Salvar e fechar.

77. No novo app 'usuarios', criamos um novo arquivo urls.py. Dentro desse arquivo recém criado colamos o seguinte código:

from django.urls import path

from . import views

urlpatterns = [

path('cadastro', views.cadastro, name='cadastro'),

path('login', views.login, name='login'),

path('dashboard', views.dashboard, name='dashboard'),

path('logout', views.logout, name='logout'),

]

78. Abrir o arquivo urls.py do aplicativo principal e fazer o include nas novas urls criadas a cima:

urlpatterns = [

path('', include('Receitas.urls')),

path('usuarios/', include('usuarios.urls')),

path('admin/', admin.site.urls),

]+ static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

79. Agora no arquivo views.py do aplicativo usuarios, criamos as views referidas no urls acima:

from django.shortcuts import render

def cadastro(request):

pass

def login(request):

pass

def logout(request):

pass

def dashboard(request):

pass

80. Recorte a pasta templates do aplicativo receitas e cole na pasta principal do projeto. Agora acesse o arquivo settings.py e procure TEMPLATES, promova a pequena alteração a seguir:

=> a linha 'DIRS': [os.path.join(BASE\_DIR, 'receitas/templates')], deve ficar a assim:

=> 'DIRS': [os.path.join(BASE\_DIR, 'templates')],

81. Do link a seguir fazer o download dos arquivos cadastro.html e login.html: <https://github.com/guilhermeonrails/material_django2_parte3/archive/aula_1.zip>. Salve os arquivos baixados na pasta templates/usuarios.

82. Na pasta templates/partials, abrir o arquivo menu.html e mudar o texto abaixo de <!-- Nav Start -->:

<div class="classynav">

<ul>

<li><a href="{% url 'index'%} ">Página principal</a></li>

<li><a href="{% url 'cadastro'%} ">Cadastro</a></li>

<li><a href="{% url 'login'%} ">Login</a></li>

</ul>

</div>

83. Agora no arquivo views.py do aplicativo usuarios, devemos mandar o django renderizar as páginas cadastro.html e login.html:

from django.shortcuts import render

def cadastro(request):

return render(request,'cadastro.html')

def login(request):

return render(request, 'login.html')

def logout(request):

pass

def dashboard(request):

pass

84. Salvar todos os arquivos e rodar o servidor: python mange.py runserver e testar todos as funções.

85. Os usuarios que vamos cadastrar serão gravados na tabela auth\_user do nosso banco de dados db.sqlite3.

86. Para criar um nova receita o formulário deverá ser aberto no method POST. Para isso devemos abrir o arquivo cadastro.html e procurar a abertura da tag form (mais ou menos página 27). Editá-lo conforme a seguir:

<form action="{% url 'cadastro' %}" method="POST">

OBS: Feito isso vinculamos o formulário a url 'cadastro' e abrimos ele com method POST.

87. No arquivo views.py do aplicativo usuarios, editamos a def cadastro:

def cadastro(request):

if request.method == 'POST':

return redirect('login')

return render(request, 'usuarios/cadastro.html')

else:

return render(request, 'usuarios/cadastro.html')

88. No arquivo cadastro.html, abaixo da linha onde definimos o method=POST, é necessário acréscentar a linha {% csrf\_token %}.

89. Agora vamos capturar o valores digitados no formulário e atribuí-los a variáveis, essas variáveis posteriormente serão usadas para construir um elemento User, que por último será gravado o banco de dados. Para tanto devemos continuar editando a def cadastro.

def cadastro(request):

if request.method == 'POST':

nome = request.POST['nome']

email = request.POST['email']

password = request.POST['password']

password2 = request.POST['password2']

return redirect('login')

else:

return render(request,'usuarios/cadastro.html')

return render(request,'usuarios/cadastro.html')

90. Para garantir que o usuário só será criado se a pessoa não deixar nome e email em branco e garantir também que segunda senha coincida com a primeira, devemos acrescentar os ifs o seguinte código:

def cadastro(request):

if request.method == 'POST':

nome = request.POST['nome']

email = request.POST['email']

senha = request.POST['password']

senha2 = request.POST['password2']

if not nome.strip():

return redirect('cadastro')

if not email.strip():

return redirect('cadastro')

if password != password2:

return redirect('cadastro')

return redirect('login')

else:

return render(request,'usuarios/cadastro.html')

return render(request,'usuarios/cadastro.html')

91. Finalmente é hora de verificar se o usuário já esta cadastrado e construi um elemento user, usando as variáveis 'nome', 'email' e 'passoword', e depois salvá-lo no banco de dados.

def cadastro(request):

if request.method == 'POST':

nome = request.POST['nome']

email = request.POST['email']

password = request.POST['password']

password2 = request.POST['password2']

if not nome.strip():

return redirect('cadastro')

if not email.strip():

return redirect('cadastro')

if password != password2:

return redirect('cadastro')

if User.objects.filter(email=email).exists():

return redirect('cadastro')

user = User.objects.create\_user(username=nome, email=email, password=senha)

user.save()

return redirect('login')

else:

return render(request,'usuarios/cadastro.html')

92. Agora no templates/usuarios/login.html, procuramos dentro do arquivo, mais ou menos na linha 27, onde esta escrito form action="", entre aspas e usando código python, inserimos a url que vai controlar esse formulário. Em method="" informamos o método "POST", para finalizar a acrescentamos o token de segurança:

<form action="{% url 'login' %}" method="POST">

{% csrf\_token %}

93. Em templates/usuarios devemos criar um novo arquivo e nomeá-lo dashboard.html. Dentro desse arquivo escrever apenas um pequeno html de teste por enquanto:

<h1>Dashboard</h1>

94. No arquivo views.py do aplicativo usuarios, nossas def de login, logout e dashboard deverão ficar assim:

def login(request):

if request.method == 'POST':

email = request.POST['email']

password = request.POST['senha']

if email == "" or senha == "":

return redirect('login')

return redirect ('dashboard')

return render(request, 'usuarios/login.html')

def logout(request):

pass

def dashboard(request):

return render(request, 'usuarios/dashboard.html')

Se o método for POST, as variáveis email e password receberão os valores digitados no formulário de login, haverá um if que vai verificar se algum dos campos esta vazio, se houve um deles vazio, será redirecionado de volta para o login e se não houver vai direcionar para a pagina dashboard, sem não for POST, vai renderizar a página 'usuarios/login.html';

95. Ainda no views.py acima, no topo da página, onde normalmente colocamos as importações, acréscentar na linha: from django.contrib import auth;

96. Na nossa aplicação queremos fazer login com email, mas por padrão o Django utiliza o 'name' como authenticador. Para resolver esse pequeno contratempo acrescentamos o seguinte código na def login:

def login(request):

if request.method == 'POST':

email = request.POST['email']

password = request.POST['password']

if email == "" or password == "":

return redirect('login')

if User.objects.filter(email=email).exists():

nome = User.objects.filter(email=email).values\_list('username', flat=True).get()

return redirect('dashboard')

return render(request, 'usuarios/login.html')

97. Agora vamos usar a função authenticate( ) para fazer login do nosso usuário:

def login(request):

if request.method == 'POST':

email = request.POST['email']

password = request.POST['password]

if email == "" or password == "":

return redirect('login')

if User.objects.filter(email=email).exists():

nome = User.objects.filter(email=email).values\_list('username', flat=True).get()

user = auth.authenticate(request, username=nome, password=password)

if user is not None:

auth.login(request, user)

return redirect('dashboard')

return redirect('dashboard')

return render(request, 'usuarios/login.html')

98. Para ter certeza que o login deu certo, fazer uma pequena mudança no nosso <h1> do templete/usuarios/dashboard.html:

<h1>Dashboard, olá, {{ user.username }} </h1>

99. Agora vamos na página index.html, copiamos todo o seu conteúdo e colamos no dashboard.html. Para que o usuário logado apareça no dashboard basta colar o código a seguir, abaixo das partials \_busca e \_menu:

<div class="contact-area section-padding-0-80">

<div class="container">

<div class="row">

<div class="col-12">

<div class="section-heading">

<h3>Olá {{ user.username }}</h3>

</div>

</div>

</div>

</div>

</div>

100. Alterar a partial \_menu.html para que o menu apresentado seja diferente quando o usuários fizer login:

<ul>

{% if user.is\_authenticated %}

<li><a href="{% url 'index' %}">Página principal</a></li>

<li><a href="{% url 'dashboard' %}">Minhas receitas</a></li>

<li><a href="{% url 'cria\_receita' %}">Criar Receita</a></li>

{% if user.is\_superuser %}

<li><a href="{% url 'admin:index' %}">Admin</a></li>

{% endif %}

<li><a href="{% url 'logout' %}">Logout</a></li>

{% else %}

<li><a href="{% url 'cadastro' %}">Cadastro</a></li>

<li><a href="{% url 'login' %}">Login</a></li>

{% endif %}

</ul>

OBS: o segundo if permite ao usuário superuser logado visualizar um botão que direciona para o localhost:8000/admin.

101. Agora vem a parte mais fácil, fazer logout, no views.py, editar a função logout:

def logout(request):

auth.logout(request)

return redirect('index')

102. Para impedir que usuarios não logados acessem o dashboard, editar a def dashboard com o seguinte código:

def dashboard(request):

if request.user.is\_authenticated:

return render(request, 'usuarios/dashboard.html')

else:

return redirect('index')

103. Agora vamos fazer uma página onde o usuário possa cadastrar suas próprias receitas. Para começar vamos no urls.py do app usuarios e acrescentamos um novo path, conforme a seguir:

urlpatterns = [

path('cadastro', views.cadastro, name='cadastro'),

path('login', views.login, name='login'),

path('dashboard', views.dashboard, name='dashboard'),

path('logout', views.logout, name='logout'),

path('cria/receita', views.cria\_receita, name='cria\_receita'),

]

104. No views.py do mesmo app, escrevemos a função cria\_receita:

def cria\_receita(request):

return render(request, 'usuarios/cria\_receita.html')

105. A função cria\_receita deve renderizar um cria\_receita.html, que ainda não existe, então dentro de templates/usuarios demos da vida a um novo arquivo cria\_receita.html e colar o código existente no link: <https://github.com/guilhermeonrails/material_django2_parte3/blob/aula_4/cria_receita.html>;

106. Alguns passos atrás criamos um vinculação das receitas ao app de pessoas, mas agora queremos que nossas receitas sejam vinculadas ao usuários, para tanto devemos editar o arquivo models.py do app receitas, apagando a importação de pessoas e incluindo importação de User:

from django.db import models

from datetime import datetime

from django.contrib.auth.models import User

from django.db.models.fields import DateTimeField

class Receita(models.Model):

pessoa=models.ForeignKey(User, on\_delete=models.CASCADE)

nome\_receita=models.CharField(max\_length=200)

ingredientes=models.TextField(default='')

modo\_preparo= models.TextField(default='')

tempo\_preparo=models.IntegerField(default=0)

rendimento=models.CharField(max\_length=100)

categoria=models.CharField(max\_length=100)

date\_receita=models.DateTimeField(default=datetime.now, blank=True)

foto\_receita=models.ImageField(upload\_to='fotos/%d/%m/%Y',blank=True, null=True, default='0')

publicada=models.BooleanField(default=False )

def \_\_str\_\_(self):

return self.nome\_receita

107. Para que as mudanças sejam gravadas no Banco de Dados 'python manage.py makemigrations' e 'python manage.py migrate';

108. No views.py do app usuarios, vamos criar algumas variáveis que vão receber os valores digitados no formulário de cria\_receita.html, pegaremos os valores utilizando o método POST, depois as variáveis serão combinadas para criar um elemento receita, antes porém é necessário identificar o usuário que esta criando a receita e para isso usaremos get\_object\_or\_404, que finalmente a receita será gravada no Banco do Dados:

def cria\_receita(request):

if request.method == 'POST':

nome\_receita = request.POST['nome\_receita']

ingredientes = request.POST['ingredientes']

modo\_preparo = request.POST['modo\_preparo']

tempo\_preparo = request.POST['tempo\_preparo']

rendimento = request.POST['rendimento']

categoria = request.POST['categoria']

foto\_receita = request.FILES['foto\_receita']

user = get\_object\_or\_404(User, pk=request.user.id)

receita = Receita.objects.create(pessoa=user, nome\_receita=nome\_receita, ingredientes=ingredientes, modo\_preparo=modo\_preparo, tempo\_preparo=tempo\_preparo, rendimento=rendimento, categoria=categoria, foto\_receita=foto\_receita)

receita.save()

return redirect('dashboard')

else:

return render(request, 'usuarios/cria\_receita.html')

109. Agora precisamos configurar nosso dashboard para exibir as receitas criadas pelo usuário logado:

def dashboard(request):

if request.user.is\_authenticated:

id = request.user.id

receitas = Receita.objects.order\_by('-date\_receita').filter(pessoa=id)

dados = {

'receitas' : receitas

}

return render(request, 'usuarios/dashboard.html', dados)

else:

return redirect('index')

110. Hora que configurar mensagens de alerta para o usuário. Começando pelo arquivo settings.py, escrevemos seguinte código abaixo de MEDIA\_URL:

#Messages

from django.contrib.messages import constants as messages

MESSAGE\_TAGS ={

messages.ERROR: 'danger',

messages.SUCCESS: 'success',

111. Agora vamos criar um novo template/partial chamado \_alertas.html e dentro dele colamos o seguinte código (verificar se há mensagens 'if messages', se houve o 'for' exibe todas a mensagens:

{% if messages %}

{% for message in messages %}

<div class="alert alert-{{message.tags}}" role="alert">

{ message }}

</div>

{% endfor %}

{% endif %}

112. Agora dentro do arquivo login.html de templates/usuarios, colocamos o código a seguir na posição mostrada (+ ou - na linha 24):

<!-- ##### Contact Form Area Start ##### -->

<div class="contact-area section-padding-0-80">

<div class="container">

<div class="row">

<div class="col-12">

<div class="section-heading">

<h2>Login</h2>

</div>

</div>

</div>

{% include 'partials/\_alertas.html'%}

Agora no arquivo cadastro:

<div class="contact-area section-padding-0-80">

<div class="container">

<div class="row">

<div class="col-12">

<div class="section-heading">

<h2>Criar conta</h2>

</div>

</div>

</div>

{% include 'partials/\_alertas.html'%}

113. Agora em views.py do app usuarios acrescentamos as mensagens:

def cadastro(request):

if request.method=='POST':

nome=request.POST['nome']

email=request.POST['email']

password=request.POST['password']

password2=request.POST['password2']

if not nome.strip():

messages.error(request, 'Digite um nome válido para o usuário')

return redirect('cadastro')

if not email.strip():

messages.error(request, 'O email não ser espaço em branco')

return redirect('cadastro')

if password !=password2:

messages.error(request, 'As senhas não são iguais')

return redirect('cadastro')

if User.objects.filter(email=email).exists():

messages.error(request, 'Usuário já cadastrado')

return redirect('cadastro')

if User.objects.filter(username=nome).exists():

messages.error(request, 'Usuário já cadastrado')

return redirect('cadastro')

user=User.objects.create\_user(username=nome, email=email, password=password)

user.save()

return redirect ('login')

else:

return render(request, 'usuarios/cadastro.html')

114. Precisamos agora de melhorar nosso dashboard acrescentando botões para editar e deletar receitas. Começamos editando o template/usuarios/dashboard, acrescentando as linhas a seguir logo abaixo da imagem:

<h5>{{receita.nome\_receita}}</h5>

<a href="{% url 'edita\_receita' receita.id %}" type="button" class="btn btn-info">Editar</a>

<a href="{% url 'deleta\_receita' receita.id %}" type="button" class="btn btn-danger">Deletar</a>

OBS: o receita.id serve para informar ao django qual receita ele deve editar ou deletar.

115. No app de usuarios vamos criar dois novos paths para atender as duas novas urls que digitamos no dashboard.html:

urlpatterns = [

path('', index, name='index'),

path('<int:receita\_id>', receita, name='receita'),

path('busca', buscar, name='buscar'),

path('cria/receita', cria\_receita, name='cria\_receita'),

path('deleta/<int:receita\_id>', deleta\_receita, name='deleta\_receita'),

path('edita/<int:receita\_id>', edita\_receita, name='edita\_receita'),

OBS: <int:receita\_id> serve para informar ao django qual receita será editada ou deletada.

116. Agora vamos criar duas novas defs no views.py do app usuarios, elas ficarão assim:

def deleta\_receita(request, receita\_id):

receita=get\_object\_or\_404(Receita, pk=receita\_id)

receita.delete()

return redirect('dashboard')

def edita\_receita(request, receita\_id):

receita=get\_object\_or\_404(Receita, pk=receita\_id)

receita\_a\_editar={'receita':receita}

return render(request, 'usuarios/edita\_receita.html', receita\_a\_editar)

117. Veja que a def edita\_receita, na última linha, renderiza um edita\_receita, precisamos criar esse arquivo lá em templates/usuarios e nomeá-lo edita\_receita.html. Do link <https://github.com/guilhermeonrails/material_django2_parte3/blob/aula2/edita_receita.html> copiar todo o html e colar no arquivo recém criado;

118. Salvando e rodando, no dashboard do usuário logado, abaixo da imagem aparecem dois botões (azul:editar) e (vermelho:deletar). A parte de deletar esta completa, mas o editar abre o formulario para edição, no entanto o botão "atualizar" ainda precisa de outra url, outra view e outra def para poder atualizar os dados editados, faremos isso no item 119.

119. No arquivo editar\_receita.html de templates/usuarios, procuramos a linha que contém form action (mais ou menos a linha 29) e fazemos a seguinte modificação:

<form action="atualiza\_receita" method="post" enctype="multipart/form-data">

120. Agora temos que adicionar a url atualiza\_receita no urls.py do aplicativo usuarios:

urlpatterns = [

path('', index, name='index'),

path('<int:receita\_id>', receita, name='receita'),

path('busca', buscar, name='buscar'),

path('cria/receita', cria\_receita, name='cria\_receita'),

path('deleta/<int:receita\_id>', deleta\_receita, name='deleta\_receita'),

path('edita/<int:receita\_id>', edita\_receita, name='edita\_receita'),

path('edita/atualiza\_receita', atualiza\_receita, name='atualiza\_receita'),

]

OBS: No curso da alura este último path é "path('atualiza\_receita', atualiza\_receita, name='atualiza\_receita')", mas percebi que para funcionar era preciso acidionar o edita: path('edita/atualiza\_receita', atualiza\_receita, name='atualiza\_receita')

121. Abrir o arquivo views.py do app usuarios e criar a def atualiza\_receita:

def atualiza\_receita(request):

if request.method =='POST':

receita\_id=request.POST['receita\_id']

r=Receita.objects.get(pk=receita\_id)

r.nome\_receita=request.POST['nome\_receita']

r.ingredientes=request.POST['ingredientes']

r.modo\_preparo=request.POST['modo\_preparo']

r.tempo\_preparo=request.POST['tempo\_preparo']

r.rendimento=request.POST['rendimento']

r.categoria=request.POST['categoria']

if 'foto\_receita' in request.FILES:

r.foto\_receita=request.FILES['foto\_receita']

r.save()

messages.success(request, 'Receita editada')

return redirect('dashboard')

OBS: Se o method for POST, pega o id da receita, definimos uma variável r e atribuimos a ela um objeto de receita correspondente ao id. No final salvamos o elemento de receita. Acrescenteu uma message de "Receita Editada" que vai ser mostrada no 'dashboard' depois que o usuario for redirecionado (para funcionar é preciso inserir no dashboard.html o partials: {% include 'partials/\_alertas.html'%}, antes da tag das imagens.

122. Para começar a criar a paginação é bem prático cadastrar e publicar no mínimo 7 receitas, para podermos testar nosso código.

123. A paginação vai começar editando o arquivo views.py do aplicativo do aplicativo receitas, primeiro vamos importar o paginator e alguns complementos:

from django.core import paginator

from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

124. Agora as seguintes alterações na def index:

def index(request):

receitas=Receita.objects.order\_by( '-date\_receita').filter(publicada=True)

paginator=Paginator(receitas, 6)

page=request.GET.get('page')

receitas\_por\_pagina=paginator.get\_page(page)

dados={

'receitas':receitas\_por\_pagina

125. Do link: <https://github.com/guilhermeonrails/material_django2_pt4/blob/aula4/paginacao.html>, devemos copiar o código html e colar no index.html logo abaixo de:

{% endfor %}

{% else %}

{% endif %}

</div>

</div>

</section>

==> COLAR AQUI

126. Agora editamos o html que colamos assim:

<section class="top-catagory-area section-padding-20-0">

<div class="container">

{% if receitas.has\_other\_pages %}

<ul class="pagination">

{% if receitas.has\_previous %}

<li class="page-item">

<a href="?page = {{receitas.previous\_page\_number}}" class="page-link">&laquo;</a>

</li>

{% else %}

<li class="page-item disabled">

<a class="page-link">&laquo;</a>

</li>

{% endif %}

{% for pagina in receitas.paginator.page\_range %}

{% if receitas.number == pagina %}

<li class="page-item active">

<a class="page-link">{{pagina}}</a>

</li>

{% else %}

<li class="page-item">

<a href="?page={{pagina}}" class="page-link">{{pagina}}</a>

</li>

{% endif %}

{% endfor %}

{% if receitas.has\_next %}

<li class="page-item">

<a href="?page={{receitas.next\_page\_number }}" class="page-link">&raquo;</a>

</li>

{% else %}

<li class="page-item disabled">

<a class="page-link">&raquo;</a>

</li>

{% endif %}

</ul>

{% endif %}

</div>

</section>

127. Para melhorar a estrutura do código precisamos reorganizar nossos arquivos de uma maneira mais lógica e intuitiva para facilitar a leitura. Para melhorar nosso projeto vamos excluir um aplicativo que não tem mais utilidade, o aplicativo 'pessoas'. Para fazer isso, no arquivo settings.py do aplicativo de configurações, vamos procurar o INSTALLED APPS, e apagar o 'pessoas' da lista;

128. No app de receitas vamos procurar a pasta migrations e dentro dela o arquivo vamos procurar o arquivo '0002\_receita\_pessoa.py' e dentro dele importar o settings.py com o código 'from django.conf import settings' , perceba que ela faz referência ao aplicativo pessoas que estamos deletando, para evitar erros devemos fazer pequena alteração nesse arquivo:

class Migration(migrations.Migration):

dependencies = [

('Receitas', '0001\_initial'),

]

operations = [

migrations.AddField(

model\_name='receita',

name='pessoa',

field=models.ForeignKey(on\_delete=django.db.models.deletion.CASCADE, to=settings.AUTH\_USER\_MODEL),

),

]

129. Feita a alteração acima é hora de deletar o banco de dados. Após deletar devemos digitar no terminal 'python manage.py migrate' que vai refazer nossas tabelas no banco de dados. Quando deletamos o banco no super user foi junto. Então no terminal digitamos 'python manage.py createsuperuser' e criamos nosso usuário com super permissões.

130. Agora vou refazer minha estrutura de templates, dentro da pasta templates criamos duas subpastas: usuarios e receitas. Na pasta receitas ficam os arquivos buscar.html, cria\_receita.html, edita\_receita.html, index.html e receita.html e na pasta usuarios deixaremos os arquivos cadastro.html, dashboard.html e login.html, o arquivo base.html ficará na pasta principal de templates.

131. Com a estrutura atual provavelmente nossa aplicativo vai da erro, pois quando views.py de receitas for procurar os templates para renderizar nada será localizado. Para resolver isso devemos entrar no views.py do aplicativo receitas e onde fizer referência aos templates devemos alterar o caminho acrescentando o 'receitas/'. Fazer coisa semelhante no arquivo views.py do aplicativo usuarios, mas agora o caminho deverá conter 'usuarios/'. No views.py do app receitas, na def index ficará assim:

return render(request, 'receitas/index.html', dados)

Agora fazer o mesmo para as outras defs do aplicativo receitas e do aplicativo usuarios.

132. Agora vamos criar um estrutura de pastas que contenha nossos apps. Na pasta principal/raiz do projeto criamos a pasta apps e dentro delas colamos as pastas Receitas e usuários. Agora sabemos onde encontrar nossos apps, mas a aplicação esta funcionando.

133. Para corrigir o erro acima precisamos editar o arquivo settings.py, acrescentando o seguinte código no final do arquivo:

PROJECT\_ROOT = os.path.dirname(\_\_file\_\_)

sys.path.insert(0,os.path.join(PROJECT\_ROOT,'apps'))

No início do arquivo devemos importar o ‘sys’ com a linha “import os, sys”;

134. Para explicar a função de cada def, por convenção se usa as “”” “”” (docstrings) que são explicações breves de como funciona aquela função. As docstrings são inseridas sempre abaixo do nome e antes da função em si.

agora vem: REMOVER APP PESSOAS, REORGANIZAR TEMPLATES, REORGANIZAR APPS.

OBS1: para subir o projeto no github, primeiro entrar no github, logar e criar um novo repertório, instalar o aplicativo git no seu sistema operacional, depois digitar a sequência de comandos no terminal:

Criar um novo repertório na sua conta do github (importante, crie o arquivo sem o readme)

criar um arquivo no CODE com nome .gitignore (nele escrevemos os arquivos os pastas que queremos que o git não munitore).

git init

git add . (se for adicionar todos os arquivos)

git commit -m 'primeiro commit'

git remote add origin <https://github.com/alexpb1/AlexReceitas.git> (para subir no repertorio AlexReceitas)

git push -u origin main

**Para somente atualizar as mudanças:**

git add .

git commit -m 'descrição das mudanças'

git push -u origin main

OBS2: Ordem usada no vídeo => Cria URL que aponta para uma view

=> Cria a view

=> Cria a Template