CURSO DE DJANGO (Disponivel em <https://youtu.be/6EpHpUoBQDc>)

PASSO BÁSICO PARA INICIAR UM PROJETO:

1 . Para usar ambiente virtual instalar o virtualenv pelo cmd com o “comando pip install virtualenv”.

2. Crie uma pasta geral para seu projeto, dentro dessa pasta vamos instalar o ambiente virtual, essa pasta pode ser criada pelo cmd, utilizando o mkdir, mas não tem problema criar pelo Windows.

3. Na pasta do projeto, digita no cmd: “virtualenv NomeAmbienteVirtualASerCriado”

4. Depois de criado o ambiente virtual é necessário ativá-lo, então entra na pasta do ambiente virtual pelo cmd utilizando o comando cd, entra na pasta “scripts”, estando nessa pasta digitar o comando “./activate”. Pronto, agora o ambiente virtual estará ativado. E o no cmd aparecerá entre parentes, no começo da linha, o nome do ambiente virtual com você esta trabalhando.

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 cmd “pip install Django” para instalar o Django lá no ambiente.

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

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. Abra o Code e selecione no menu “abrir pasta”, e abra a pasta onde está o projeto criado.

10. Pelo terminal, pode ser do próprio code, digita o comando: “django-admin startapp tasks”, depois digita “django-admin startapp about” e finalmente digita: “django-admin startapp users”. Feito isso você criou 3 aplicativos dentro do projeto: tasks, about e users.

11. No code, clica no arquivo settings.py e procura dentro do texto onde está escrito INSTALLED\_APPS, vai aparecer uma lista de apps padrão do Django, basta você acrescentar os 3 apps criados no final da lista:

‘tasks’

‘about’

‘users’

12.No app tasks, cria um arquivo chamado urls.py, dentro dele cole o mesmo texto (a partir da linha 17) do arquivo urls.py que tem dentro do app principal do projeto (no nosso caso o projeto chama-se “ToDo”) e escreva a os códigos conforme o vídeo 3 do vídeo.

13. No app tasks, cria um arquivo chamado views.py, e escreva a os códigos conforme o vídeo 3.

14. No app urls.py do projeto ToDo, faça o include das urls conforme o vídeo 3.

15. Criando agora primeira página funcional do projeto, primeiramente vamos criar a URL, a VIEW e o TEMPLATE. No arquivo urls.py do app tasks, acrescente as paths conforme o vídeo 4.

16. No arquivo views.py do app tasks criar a view taskList conforme o vídeo 4.

17. Clique no aplicativo tasks com botão direito do mouse e escolha new folder com nome “templates”, dentro da nova pasta templates, crie outra nova pasta chamada tasks e dentro de dessa última pasta crie um arquivo chamado list.html.

18. Digite um html em list.html, pode ser: <h1> Lista de Tarefas </h1>

19. Agora dentro da pasta templates do projeto crie um arquivo chamado base.html que servirá de base para as outras paginas do projeto. Editar o base.html conforme vídeo 5 (a partir de 2:00)

20. Editar os arquivos list.html e yourname.html conforme vídeo 5 ( a partir de 3:40);

21. Dentro do app tasks crie uma pasta chamada static e dentro dela crie duas subpastas ‘css’ e ‘js’. Dentro da pasta css crie um arquivo chamado styles.css e dentro da pasta js crie um arquivo chamado scripts.js.

22. Edite o arquivo styles.css e scripts.js conforme o vídeo 6 (a partir de 2:00), apenas para fazer um teste do styles.css.

23. Abra o arquivo base.html e acrescente as linhas conforme vídeo 6 (a partir de 2:15)

24. Adicionar bootstraps (são tipo modelos prontos de códigos que economizam tempo de digitação e pesquisa). O tipo usado no projeto é bootsstrapsCDN, disponível em <https://www.bootstrapcdn.com/>.

25. No nosso projeto vamos copiar o código disponível no link acima, na opção CSS:” <link rel="stylesheet"href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css" integrity="undefined" crossorigin="anonymous">” e colamos dentro do arquivo arquivo base, conforme vídeo 6 ( a partir de 5:14). Da mesma forma copiamos o link: <https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/js/bootstrap.esm.min.js> disponível em JavaScript .

26. No arquivo base.html adicionar uma div conforme mostra o vídeo 6 (a partir de 6:40), a div esta disponível em <https://getbootstrap.com/docs/5.0/layout/containers/>.

27. copie o código disponível na página <https://getbootstrap.com/docs/5.0/components/navbar/> Dentro do arquivo base.html, conforme vídeo 6 ( a partir de 7:55), fazendo isso você está criando uma barra de navegação.

28. Na linha 13 do arquivo base.html, acrescente ‘-fluid’, deve ficar assim: <div class='container-fluid'>.

29. Fazer mudanças na linha 37,conforme vídeo 7 (a partir de 1:20);

30. Editar aquivo style.css de acordo com vídeo 7|(a partir de 1:48);

31. Editar o arquivo list.html de acordo com vídeo 7|(a partir de 5:20);

32. No arquivo base.html você deve colar embaixo do bootstrap (linha 12), conforme vídeo 7 ( a partir de 7:00), o código disponível no link: ‘<https://fontawesome.com/account/cdn>’ (trata-se de um bootstrap que vai aproveitar o icons de um repositório online.

33.Continuar construindo o list.html com base no vídeo 7.

34. Do item 28 até 33 fizemos apenas a estilização do projeto através do arquivo style.css, com exceção da construção do list.html no final do vídeo 7. (Obs1: o navegador pode gravar em cache a estilização anterior e por esse motivo não mostrar quando fizermos mudanças, então é sempre bom esvaziar o cache quando perceber que as mudanças não estão sendo notadas. Obs2: Quando estiver editando o arquivo style.css não colocar espaços entre os números e as unidades de medida, ERRADO 10 px, CORRETO 1Opx.

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. Editar o arquivo conforme vídeo 8 (a partir de 4:10)(CORRIGIR O COMANDO MOSTRADO AOS 7:00, RETIRAR OS NÚMEROS DA CONSTANTE STATUS, SUBSTITUINDO-0S POR NÚMEROS, EM VEZ DE (1, ‘done’), (2, ‘Doing’), fica correto com (‘Done’, ‘Done’), (‘Doing’, ‘doingo’), alterar o max\_length da linha 13 de ‘1’ para ‘5’). Depois executar o “python manage.py makemigrations” (será criado o arquivo 0001\_initial.py que contém as intruções para criação de uma tabela no banco de dados), em seguida executar o comando “python manage.py migrate”. Então será criado uma tabela no banco de dados.

36. 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”.

37. Para ter acesso ao arquivos de dados (model task) pelo múdulo administrativo tem que abrir o arquivo admin.py do app tasks e registrar o model que deve ser mostrado, conforme mostra o vídeo 8 (a partir de 14:10). Só salvar e verificar lá no localhost:8000/admin.

38. 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.

39. No arquivo views.py do app tasks cria uma def chamada taskList que atribuir à variável tasks=Task.objects.all (pega todos os objetos de task), conforme vídeo 9 (a partir de 1:00);

40. No arquivo list.html criar um {% for task in tasks%}, conforme vídeo 9 (a partir de 3:30). O for vai percorrer todos os registros e mostrar os títulos na página list.html;

41. No arquivo urls.py do aplicativo tasks acrescentar uma path que vai direcionar cada tarefa para o caminho task/”id da tarefa”, conforme vídeo 9 (a partir de 7:15). (NO VÍDEO É COMETIDO UM ERRO, COLOCOU UMA BARRA ANTES DE TASK NA LINHA 8, NÃO PODE CONTER ESSA BARRA, TEM QUE DELETÁ-LA.);

42. Voltando no arquivo views.py cria uma view chamada taskView conforme vídeo 9 (a partir de 8:15);

43. Na pasta templates do app tasks cria um novo arquivo chamado task.html, editar o arquivo conforme vídeo 9 (a partir de 10:05);

44. No arquivo base.html, na linha “<a class="navbar-brand" href="#">ToDo</a>, precisa trocar o ‘#’ por ‘barra’. Agora já é possível acessar cada tarefa e ver sua descrição e voltar a página principal clicando no Todo.

45. No arquivo urls.py editar acrescentando um novo path newtask/ que vai apontar para uma nova view newtask, conforme video 10 (a partir de 1:00).

46. No template list.html editar para deixar o botão 'adicinar tarefa' funcional, basta envolver o botão pelo html <a id=add-link' href='newtask/'></a>. Esse add-link chama a configuração do nosso arquivo styles.css, portanto devemos editá-lo para incluir a referência a esse add-link. No video 10 a edição do list.html acontece a partir de 1:25;

47.No app tasks criar um arquivo chamado forms.py. Dentro desse arquivo devemos criar uma classe para cada form que desejamos criar, o form usa os model salvos no arquivo models.py para criar os formulários automaticamente. A edição do arquivo forms.py começa a partir de 2:55 do video 10.

48. No arquvo views do app task, criar a nova view chamada newtask, que vai pegar o formulário taskform atribuilo a uma variável form e mandá-lo imprimir no template, no vídio 10 a partir de 5:08.

49. Na pasta templates/tasks criar um novo arquivo com nome addtask.html, para economizar tempo compia conteudo do arquivo task.html. Manda o django imprimir o formulário através do method POST. Editar conforme video 10 a partir de 6:30.

50. No arquivo urls.py criar uma nova path('edit/<int:id>', views.editTask, name='edit-task'), esse caminho é que vai direcionar para a tarefa que será editada (identificada pelo id chamao no ínicio).

51. No arquivo views.py criar a view editTask, conforme video 11 a partir de 1:15.

52. Na pasta templates/task adicione um arquivo edittask.html conforme video 11 a partir de 4:15.

52. No arquivo list.html edite a linha que se refere ao botão edit, a que tem a class='fas fa-edit', no meu projeto a essa altura do campeonato é a linha (procure por:<span><i class ='fas fa-edit'></i></span>), envolver esse código com um <a href='edit/{{task.id}}> </a>, isso fará com que o icone seja link para o formulário de edição de uma determinada tarefa.

53. Voltando ao arquivo de views.py, agora editar o arquivo conforme video 11 a partir de 7:04.

54. Agora vamos fazer um meio de deletar uma tarefa, a primeira coisa é criar uma url por onde a gente poderá acessar a view onde poderemos deletar a tarefa, para isso criamos um novo path no arquivo urls.py, conforme video 12 a partir de 0:32.

55. No arquivo views.py cria a função deletetask(request, id): achamos a tarefa a ser deletada setando uma variável task = get\_object\_or\_404(task, pk=id), depois disso basta task.delete. Escrevendo no final da view 'return redirect('/') volta pra lista de tarefas após deletar a tarefa selecionada.

56. Depois a fazer o ícone da lata de lixo funcionar, basta adicionar a tag de áncora no arquivo list.html, envolvendo a linha <span><i class ='fas fa-delete</span> com um <a href='delete/{{task.id}}'> <a/>.

57. Agora será programada uma mensagem de aviso quando uma tarefa for apagada. No arquivo de view.py acrescentar um código na view deletetask, conforme video 12 a partir de 3:47. Depois editar o arquivo list.html conforme video 12 a partir de 4:54. Editar o arquivo styles.css conforme video 12 a partir de 6:28. Depois que editamos o arquivo css é importante limpar o cache do navegador para as mudanças poderem ter efeito.

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

=> Cria a view

=> Cria a Template