

Django

(Telusko)

Project - Travello

* (we got the 'Travello' related templates/files from Telusko - Django Tutorial)

(01:04:02)

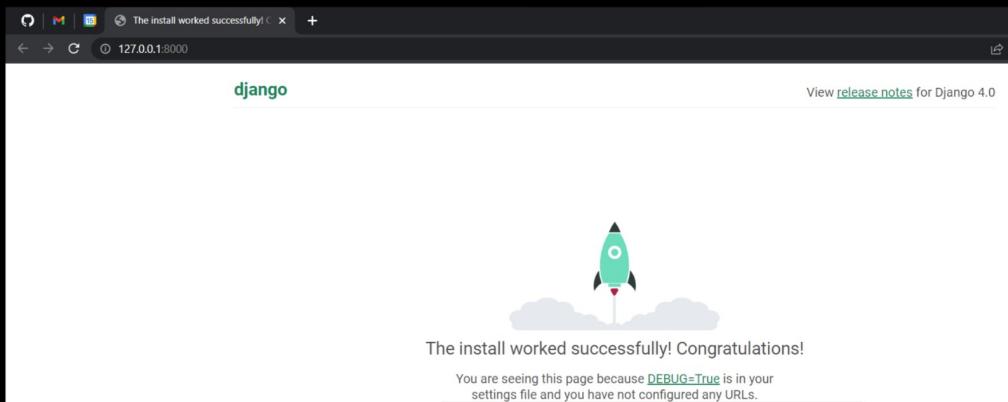
① (django telusko) D:\Edu\ Django telusko>django-admin startproject travello → created a project
(django telusko) D:\Edu\ Django telusko>

Note: Refer to Django_inNeuron_Telusko.pdf before starting this file.

(django telusko) D:\Edu\ Django telusko> travello>dir
Volume in drive D is Data
Volume Serial Number is B49E-AE1F
Directory of D:\Edu\ Django telusko>travello
15-04-2022 05:03 PM <DIR> .
15-04-2022 05:03 PM <DIR> ..
15-04-2022 05:03 PM 686 manage.py
15-04-2022 05:03 PM 1 File(s) 686 bytes
3 Dir(s) 944,756,375,552 bytes free
(django telusko) D:\Edu\ Django telusko>travello>

↑
Saved them in
travello-site-files)

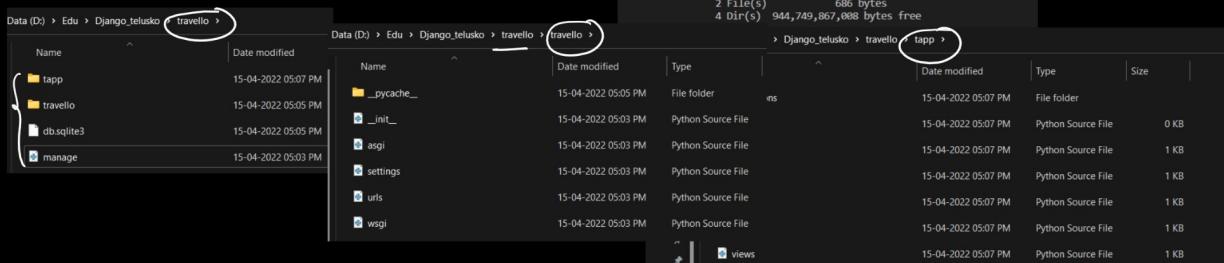
(django telusko) D:\Edu\ Django telusko>travello>python manage.py runserver → run server
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
April 15, 2022 - 17:06:06
Django version 4.0.4, using settings 'travello.settings'.
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.



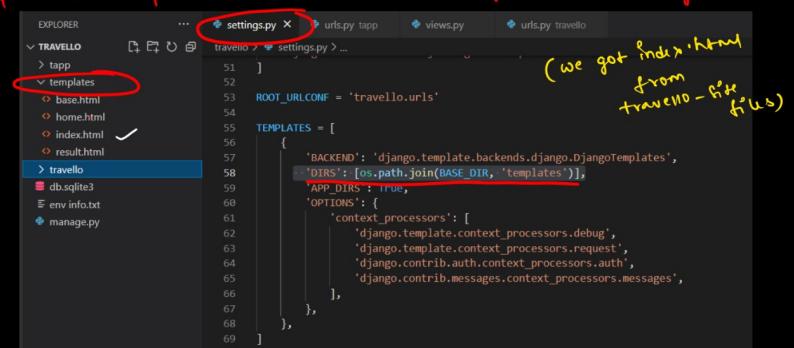
(django telusko) D:\Edu\ Django telusko>travello>python manage.py startapp tapp
(django telusko) D:\Edu\ Django telusko>travello>

② Created app

(django telusko) D:\Edu\ Django telusko> travello>dir
Volume in drive D is Data
Volume Serial Number is B49E-AE1F
Directory of D:\Edu\ Django telusko>travello
15-04-2022 05:07 PM <DIR> ..
15-04-2022 05:05 PM 0 db.sqlite3
15-04-2022 05:03 PM 686 manage.py
15-04-2022 05:07 PM <DIR> tapp
15-04-2022 05:05 PM <DIR> travello
2 File(s) 686 bytes
4 Dir(s) 944,749,867,088 bytes free



③ *→ created 'templates' folder & added its path in the 'DIRS' of Templates of settings.py



④

→ create urls.py (in the app)

settings.py urls.py tapp views.py urls.py travello

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

EXPLORER ... settings.py urls.py tapp views.py u...
travello > views.py ...
1 from django.shortcuts import render, HttpResponseRedirect
2 from django.http import HttpResponseRedirect
3
4 # Create your views here.
5
6 def index(request):
7 return render (request, "index.html")
8
9
10

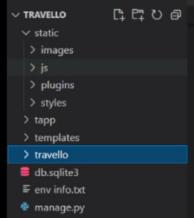
travello > urls.py ...
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5 path('', views.index, name='index'),
6]

travello > urls.py ...
14 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15
16 from django.contrib import admin
17 from django.urls import path, include
18
19 urlpatterns = [
20 path('', include('tapp.urls')),
21 path('admin/', admin.site.urls),
22]

passing static files:

→ we get this because we don't have the images and other related files of travello-site

⑤ → Create a folder 'static' for all the static files (Copy the static folder from travello-site-files)



So if we get the same page without any layout/images.

→ This is because we have the files in 'static' folder but we didn't map them.

map them.

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.0/howto/static-files/
STATIC_URL = 'static/' → In settings.py we have
# Default primary key field type
# https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

field.

Creating
static files
directly

```
114
115
116 # Static files (CSS, JavaScript, Images)
117 # https://docs.djangoproject.com/en/4.0/howto/static-files/
118
119 STATIC_URL = 'static/' → with this we are
120 STATICFILES_DIRS = [ informing Django that
121     os.path.join(BASE_DIR, 'static'), all the static files will
122 ] be present in the
123 STATIC_ROOT = os.path.join(BASE_DIR, 'assets')
```

we can give any name 'Base-DIR / static' directory.

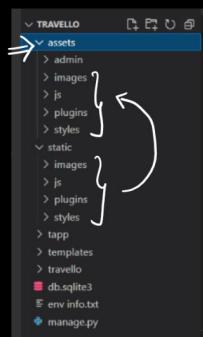
what happens here is, to improve the process, Django will collect all the files in our 'static' folder & keeps in an other

Django folder ('assets')

→ we need to run a command so that Django will create a folder with our static files.

⑥ ⇒ python manage.py collectstatic

```
(django_telusko) D:\Edu\DJango_telusko\travello>python manage.py collectstatic ✓
253 static files copied to 'D:\Edu\DJango_telusko\travello\assets'.
(django_telusko) D:\Edu\DJango_telusko\travello>
```



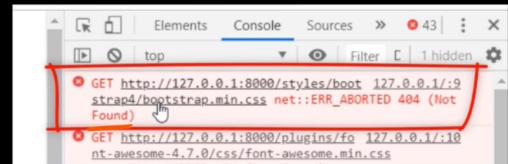
Once we run the command, Django will create a folder ('assets') & will collect all the files from 'static' folder in the 'assets' folder.

→ Still will get the page with no images/ layout.

In the browser console we see error that the url is NOT found.

→ This is because in the index.html we have all the links to static pages but these won't be understandable by Django

```
5 <!DOCTYPE html>
6 <html lang="en">
7 <head>
8   <meta charset="UTF-8">
9   <meta name="viewport" content="width=device-width, initial-scale=1">
10  <title>Travello</title>
11  <link rel="stylesheet" href="styles/bootstrap4/bootstrap.min.css" type="text/css">
12  <link rel="stylesheet" href="plugins/fontawesome-4.7.0/css/font-awesome.min.css" type="stylesheet" rel="stylesheet" type="text/css">
13  <link rel="stylesheet" href="plugins/OwlCarousel2-2.2.1/owl.carousel.css" type="text/css">
14  <link rel="stylesheet" href="plugins/OwlCarousel2-2.2.1/owl.theme.default.css" type="text/css">
15  <link rel="stylesheet" href="plugins/OwlCarousel2-2.2.1/animate.css" type="text/css">
16  <link rel="stylesheet" href="styles/main_styles.css" type="text/css">
17  <link rel="stylesheet" href="styles/responsive.css" type="text/css">
```



⑧ To make it understand to Django, we need to enclose the urls in `static ' ' .` which says that all the files of these urls are coming from 'static' folder.

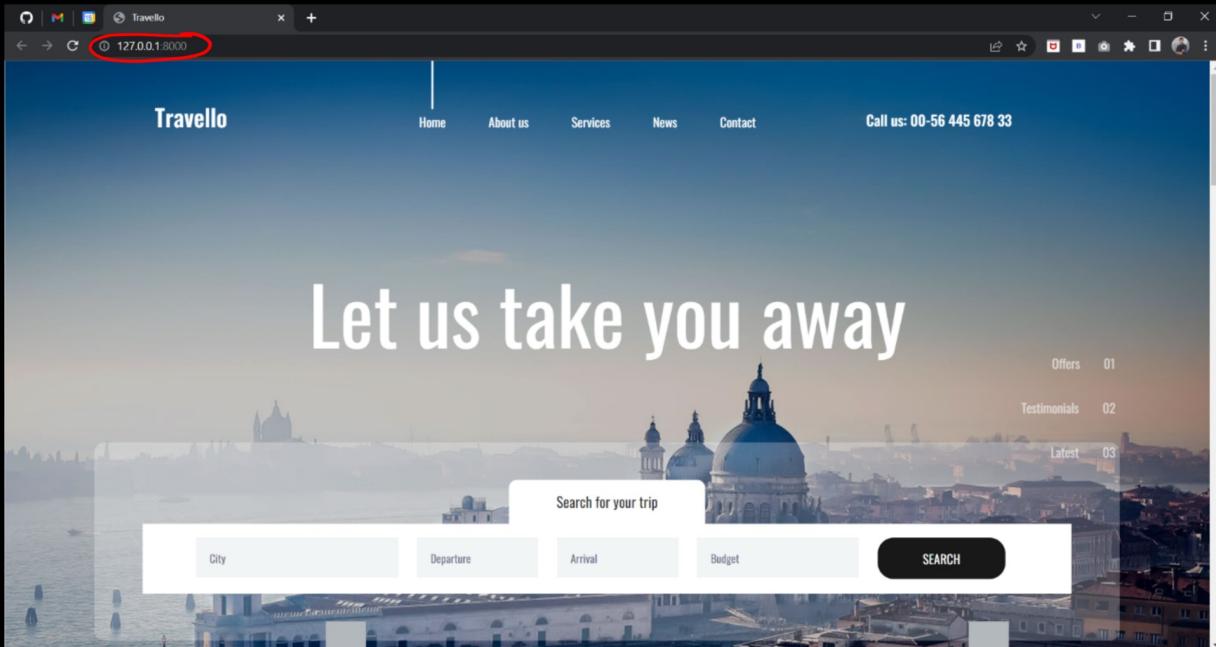
```
5 <!DOCTYPE html>
6 <html lang="en">
7 <head>
8   <title>Travello</title>
9   <meta charset="UTF-8">
10  <meta name="viewport" content="width=device-width, initial-scale=1">
11  <link rel="stylesheet" href="{% static 'styles/bootstrap4/bootstrap.min.css' %}" type="text/css">
12  <link rel="stylesheet" href="{% static 'plugins/fontawesome-4.7.0/css/font-awesome.min.css' %}" type="stylesheet" rel="stylesheet" type="text/css">
13  <link rel="stylesheet" href="{% static 'plugins/OwlCarousel2-2.2.1/owl.carousel.css' %}" type="text/css">
14  <link rel="stylesheet" href="{% static 'plugins/OwlCarousel2-2.2.1/owl.theme.default.css' %}" type="text/css">
15  <link rel="stylesheet" href="{% static 'plugins/OwlCarousel2-2.2.1/animate.css' %}" type="text/css">
16  <link rel="stylesheet" href="{% static 'styles/main_styles.css' %}" type="text/css">
17  <link rel="stylesheet" href="{% static 'styles/responsive.css' %}" type="text/css">
```

We need to load the static

```
1 <% load static %>
2 <% static "images" as baseUrl %>
3 <!DOCTYPE html>
4 <html lang="en">
5 <head>
6   <title>Travello</title>
7   <meta charset="UTF-8">
8   <meta name="viewport" content="width=device-width, initial-scale=1">
9   <link rel="stylesheet" href="{% static 'styles/bootstrap4/bootstrap.min.css' %}" type="text/css">
10  <link href="{% static 'plugins/fontawesome-4.7.0/css/font-awesome.min.css' %}" rel="stylesheet" type="text/css">
11  <link rel="stylesheet" href="{% static 'plugins/OwlCarousel2-2.2.1/owl.carousel.css' %}" type="text/css">
12  <link rel="stylesheet" href="{% static 'plugins/OwlCarousel2-2.2.1/owl.theme.default.css' %}" type="text/css">
13  <link rel="stylesheet" href="{% static 'plugins/OwlCarousel2-2.2.1/animate.css' %}" type="text/css">
14  <link rel="stylesheet" href="{% static 'styles/main_styles.css' %}" type="text/css">
15  <link rel="stylesheet" href="{% static 'styles/responsive.css' %}" type="text/css">
```

(django telusko) D:\edu\ Django_telusko\travello>python manage.py collectstatic
253 static files copied to 'D:\edu\ Django_telusko\travello\assets'.
(django telusko) D:\edu\ Django_telusko\travello>python manage.py runserver
watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
April 15, 2022 - 18:16:54
Django version 4.0.4, using settings 'travello.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

⑨



For those who couldn't figure out how to put `{% static ' ' %}` in the background images!
When you type `{% static 'image-name' %}`, you are essentially just adding the base directory of the 'Static' folder we created before the image location. You can type `/static/` before the image location and it will come out the same. It should look like: `style="background-image:url('/static/images/travello.jpg')".`

passing Dynamic Data in HTML:

02 JUNE
Best tips to travel light
LIFESTYLE & TRAVEL
Pellentesque sit amet elementum cumsan sit amet mattis eget, tristique at leo. Vivamus massa. Tempor massa et laoreet.

01 JUNE
Best tips to travel light
LIFESTYLE & TRAVEL
Tempor massa et laoreet malesuada. Pellentesque sit amet elementum cumsan sit amet mattis eget, tristique at leo.

→ Initially the discount in the page is 20%, but I want to change it.
Not directly from the html page.

⇒ Let make it dynamic.

01 JUNE
Best tips to travel light
LIFESTYLE & TRAVEL
Tempor massa et laoreet malesuada. Pellentesque sit amet elementum cumsan sit amet mattis eget, tristique at leo.

29 MAY
Best tips to travel light
LIFESTYLE & TRAVEL
Vivamus massa. Tempor massa et laoreet malesuada. Aliquam nulla nisl, accumsan sit amet mattis.

Get a 20%
Discount
Buy Your Vacation Online Now

⇒ We need to define a parameter for discount in views.py & call it in index.html.
→ In this way, we are making the discount values as dynamic, because we are not hardcoding the values in html, we are changing in views.py

```
tapp > * views.py ...
1  from django.shortcuts import render, HttpResponseRedirect
2  from django.http import HttpResponseRedirect
3
4
5  # Create your views here.
6
7  def index(request):
8      return render (request, 'index.html',{'discount':'10%'})
```

index.html

```
<div class="background_image" style="background-image:url({% static 'images/travello.jpg' %});>
```

Screenshot of the travel website showing the same two blog posts. A red annotation on the right side highlights a promotional banner with 'Get a 10% Discount Buy Your Vacation Online Now'.

Travello

Home About us Services News Contact Call us: 00-56 445 678 33

SIMPLY AMAZING PLACES

Popular Destinations

Mumbai
The City That Never Sleeps
From \$700

Hyderabad
First Biryani, Then Sherwani
From \$650

Bengaluru
Beautiful City
From \$750

⇒ We want to make the fields (marked above) dynamic.

for that, create a class & declare variables. As 'models.py' deals with data we need to create a class in models.py then give the logics in views.py i.e., instantiate the class (create object) & pass values.

```
urls.py app models.py x views.py urls.py travelo settings.py
tapp > models.py > Destination
1 from django.db import models
2
3 # Create your models here.
4 # Models are used to connect with databases and data.
5
6 class Destination:
7     id : int
8     name : str
9     img : str
10    desc : str
11    price : int
```

```
urls.py app models.py x views.py urls.py travelo settings.py index.html
tapp > views.py > index
1 from django.shortcuts import render, HttpResponseRedirect
2 from django.http import HttpResponseRedirect
3 from . import models
4 from .models import Destination
5
6
7 # Create your views here.
8
9 # def index(request):
10 #     return render (request, 'index.html',{'discount':'10%'})
11
12 def index(request):
13
14     dest1 = Destination()
15     dest1.name = 'Mumbai'
16     dest1.desc = 'The city That Never Sleeps'
17     dest1.img = 'destination_1.jpg'
18     dest1.price = 700
19
20     dest2 = Destination()
21     dest2.name = 'Hyderabad'
22     dest2.desc = 'First Biryani, Then Sherwani'
23     dest2.img = 'destination_2.jpg'
24     dest2.price = 650
25
26     dest3 = Destination()
27     dest3.name = 'Bengaluru'
28     dest3.desc = 'Beautiful City'
29     dest3.img = 'destination_3.jpg'
30     dest3.price = 750
31
32     dests = [dest1, dest2, dest3]
33
34     return render(request, "index.html", {"dests": dests})
```

objects → values

Now we need to mention these values in index.html

(dest1.name, dest1.desc, dest1.price)

but as we have multiple obj, we create a list of objects & pass it in index.html

```
index.html x models.py views.py
templates > index.html
1 [% load static %]
2 [% static "images" as baseUrl %] → we are defining a variable 'baseUrl' for 'images' present in 'static' folder
3 <!DOCTYPE html>
4 <html lang="en">
5
6     <head>
```

```
index.html x models.py views.py
templates > index.html
267 <!-- Destinations -->
268
269     <div class="destinations" id="destinations">
270         <div class="container">
271             <div class="row">
272                 <div class="col text-center">
273                     <div class="section_subtitle">simply amazing places</div>
274                     <div class="section_title">
275                         <h2>Popular Destinations</h2>
276                     </div>
277                 </div>
278             </div>
279             <div class="row destinations_row">
280                 <div class="col">
281                     <div class="destinations_container item_grid">
282
283                         [% for dest in dests %]
284                             <!-- destination -->
285                             <div class="destination_item">
286                                 <div class="destination_image">
287                                     
289                                 <div class="spec_offer text-center"><a href="#">Special Offer</a></div>
290                                 <div class="destination_content">
291                                     <div class="destination_title"><a href="destinations.html">{{dest.name}}</a></div>
292                                     <div class="destination_subtitle">
293                                         <p>{{dest.desc}}</p>
294                                     </div>
295                                     <div class="destination_price">From ${{dest.price}}</div>
296                                 </div>
297                             <% endfor %>
298                         </div>
299
300
301
302
303             </div>
304         </div>
305     </div>
```

Creating a for loop & passing the values as dynamic content

→ In this way we get the values/content from the logic written in views.py (or) from database.

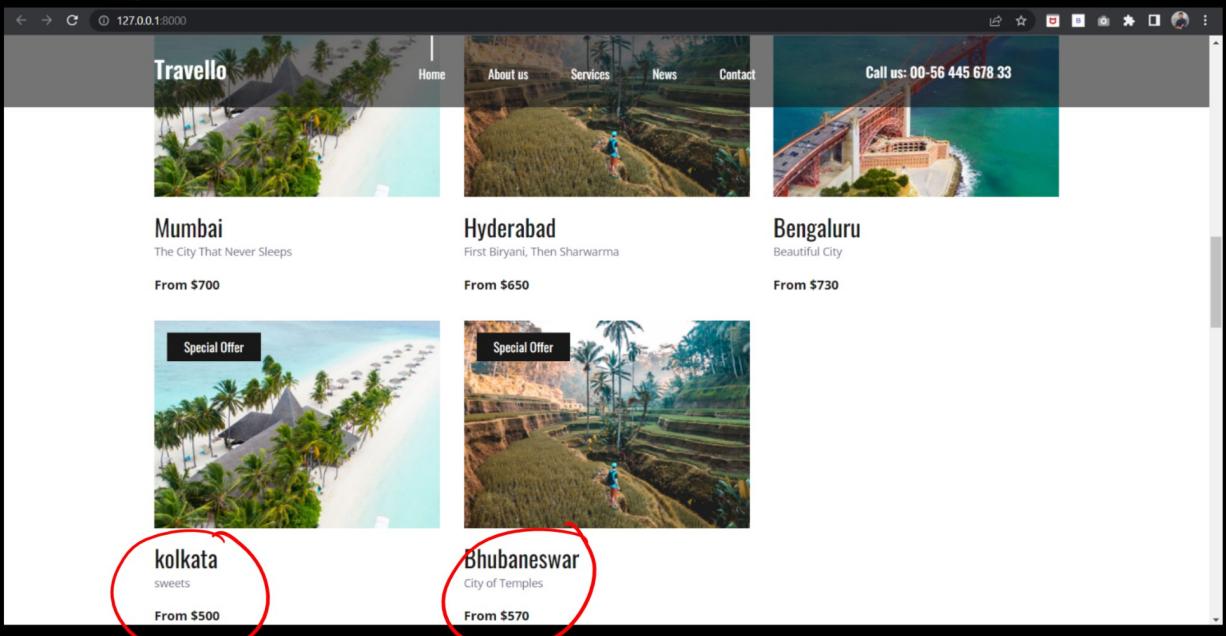
⇒ The page is now dynamic.
i.e., if we want to add new places, add the details in views.py -- Job done.

```
index.html   models.py   views.py x
tapp > views.py > index

12 def index(request):
13
14     dest1 = Destination()
15     dest1.name = 'Mumbai'
16     dest1.desc = 'The City that Never Sleeps'
17     dest1.img = 'destination_1.jpg' #we need to mention the name of image which is in the images folder od static
18     dest1.price = 700
19
20
21     dest2 = Destination()
22     dest2.name = 'Hyderabad'
23     dest2.desc = 'First Biryani, Then Sharwama'
24     dest2.img = 'destination_2.jpg'
25     dest2.price = 650
26
27
28     dest3 = Destination()
29     dest3.name = 'Bengaluru'
30     dest3.desc = 'Beautiful city'
31     dest3.img = 'destination_3.jpg'
32     dest3.price = 730
33
34
35     dest4 = Destination()
36     dest4.name = 'Kolkata'
37     dest4.desc = 'sweets'
38     dest4.img = 'destination_1.jpg'
39     dest4.price = 500
40
41
42     dest5 = Destination()
43     dest5.name = 'Bhubaneswar'
44     dest5.desc = 'City of Temples'
45     dest5.img = 'destination_2.jpg'
46     dest5.price = 570
47
48
49
50     dests = [dest1, dest2, dest3, dest4, dest5]
51
52
53     return render(request, 'index.html', {'dests': dests, 'discount': '10%'})
```

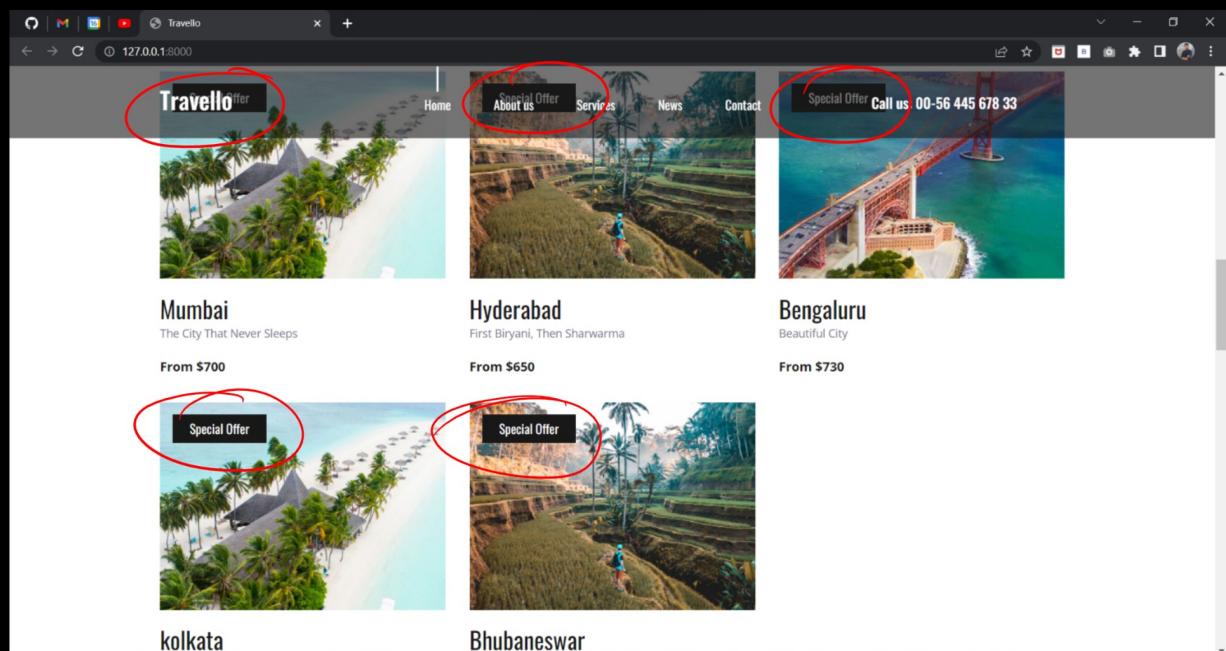
} Details added in views.py
the content will be displayed in page => The page is now

Details added in views.py
&
the Content will be displayed in page \Rightarrow The page is now dynamic.



If statement in Django: (for the dynamic Content)

→ here we see that special offer for all places, but that shouldn't be the case.
we should special offer based on some condition.



So, to give offer to any particular place we need to have some condition.
⇒ we are defining a variable (`offer:bool`) - This accepts True/false. when it is True, the offer will be applicable for that place.

models.py:

```
1 from django.db import models
2
3 # Create your models here.
4 # Models are used to connect with databases and data.
5
6 class Destination:
7     id : int
8     name : str
9     img : str
10    desc : str
11    price : int
12    offer : bool
```

views.py:

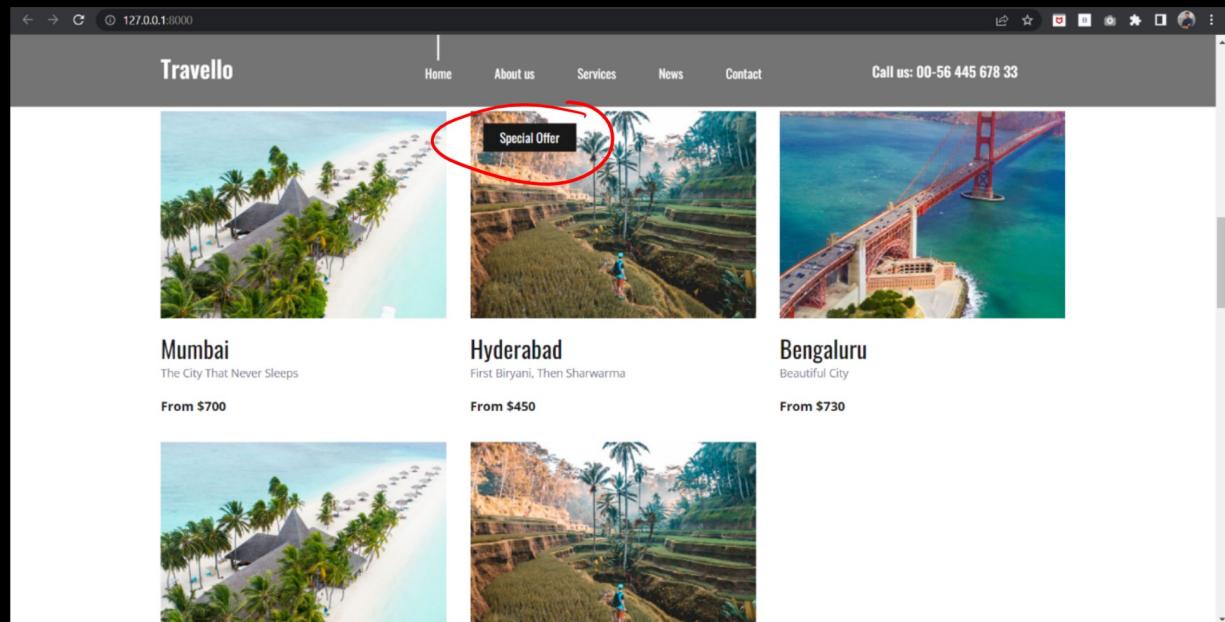
```
14 dest1 = Destination()
15 dest1.name = 'Mumbai'
16 dest1.desc = 'The City That Never Sleeps'
17 dest1.img = 'destination_1.jpg' #we need to mention the name of image which is in the images folder od static
18 dest1.price = 700
19 dest1.offer = False
20
21 dest2 = Destination()
22 dest2.name = 'Hyderabad'
23 dest2.desc = 'First Biryani, Then Sharwarma'
24 dest2.img = 'destination_2.jpg'
25 dest2.price = 450
26 dest2.offer = True
27
28 dest3 = Destination()
29 dest3.name = 'Bengaluru'
30 dest3.desc = 'Beautiful City'
31 dest3.img = 'destination_3.jpg'
32 dest3.price = 730
33 dest3.offer = False
34
35 dest4 = Destination()
36 dest4.name = 'Kolkata'
37 dest4.desc = 'Sweets'
38 dest4.img = 'destination_1.jpg'
39 dest4.price = 580
40 dest4.offer = False
41
42 dest5 = Destination()
43 dest5.name = 'Bhubaneswar'
44 dest5.desc = 'City of Temples'
45 dest5.img = 'destination_2.jpg'
46 dest5.price = 570
47 dest5.offer = False
48 dests = [dest1, dest2, dest3, dest4, dest5]
49
50 return render(request, 'index.html', {'dests': dests, 'discount': '10%'})
```

index.html:

```
267 <!-- Destinations -->
268
269 <div class="destinations" id="destinations">
270   <div class="container">
271     <div class="row">
272       <div class="col text-center">
273         <div class="section_subtitle">simply amazing places</div>
274         <div class="section_title">
275           <h2>Popular Destinations</h2>
276         </div>
277       </div>
278     </div>
279     <div class="row destinations_row">
280       <div class="col">
281         <div class="destinations_container item_grid">
282
283           <% for dest in dests %>
284             <%-- destination -->
285             <div class="destination_item">
286               <div class="destination_image">
287                 
288               <% if dest.offer %>
289                 <div class="spec_offer text-center"><a href="#">Special Offer</a></div>
290               <% endif %>
291             </div>
292             <div class="destination_content">
293               <div class="destination_title"><a href="destinations.html">{{dest.name}}</a></div>
294               <div class="destination_subtitle">
295                 <p>{{dest.desc}}</p>
296               </div>
297               <div class="destination_price">From ${{dest.price}}</div>
298             </div>
299           </div>
300
301         <% endfor %>
302       </div>
303     </div>
304   </div>
```

Annotations:

- 'If block' (highlighted in green) → we kept the offer statement in the 'If block'
- Only when `dest.offer = True`, the 'Special Offer' will be displayed.
- ⇒ 'offer' should be applicable only for Hyderabad.



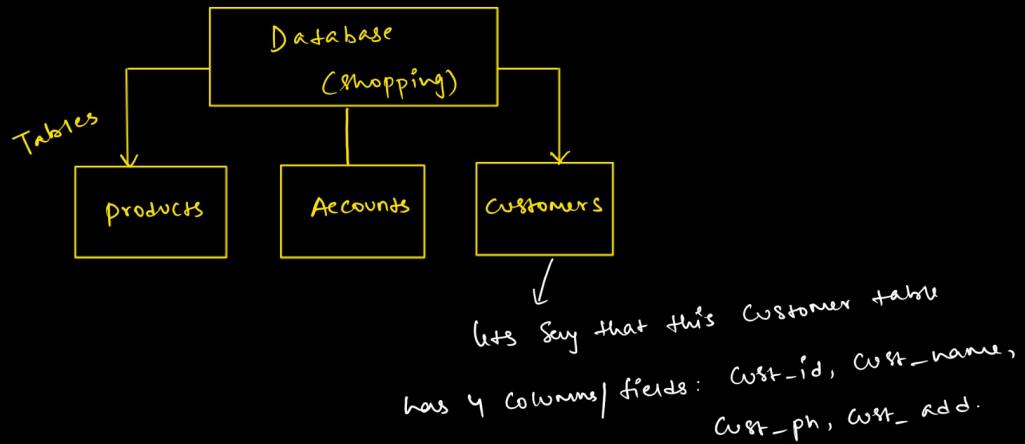
Now we want to fetch the data from the database.

→ Migration is creating the tables in the database with the help of 'models'.

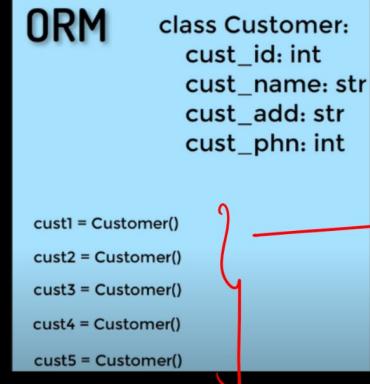
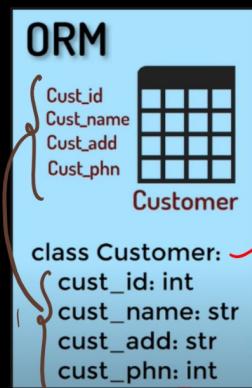
Object Relational Mapper (ORM) Theory



User can see the data in the database with the help of an application.

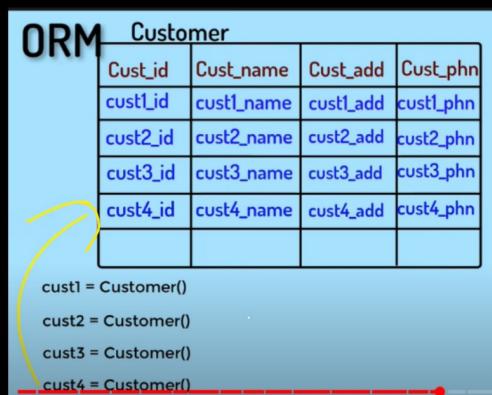


⇒ Now to define this table in the application - we will create a class.



① for the customer table, we create a 'Customer' class.
& the fields/columns will be the class variables/properties.

② Now we can have multiple customers,
so we create one object for each customer.
⇒ Every object will have different values
(name, id, add, ph)

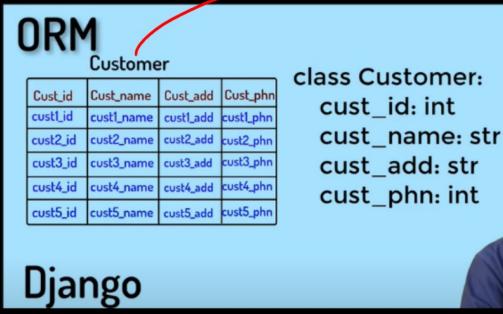


** So, when we map the data/to from the database.
→ One object = One row of the table.

⇒ We need to write the SQL queries to insert the data in the database.table from the objects.

But what if we don't want to use SQL queries ??
we just want to focus on application.

→ So, we have the 'Customer' class, we don't even want to create the table manually.



⇒ The table should be created automatically.

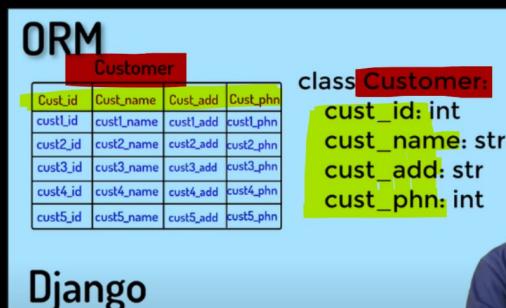


Is it possible? → YES

The framework that we use (Django) has the special power to look at the class and says, 'Hey I (django) need to create a table for the class.'

⇒ Class name = Customer, so django will create a table - 'Customer'.

Class has 4 properties, so django will create 4 columns in the table



→ If we create a new object in the class, a row will be added automatically in the table of database ⇒ That is the power of ORM.

⇒ ORM → we can directly create tables with the help of the classes which we create and the data will be added into the tables with the help of the objects we create in the class.

ORM - Object Relational Mapper

Class → database/Table
Object → data in table
mapping

mapping of the objects to the relational databases.

Postgres and PgAdmin Setup:

We have many RDBMS - MySQL, PostgreSQL etc... to work with Python.

→ We will use PostgreSQL here.

→ We need to download the PostgreSQL Software

The screenshot shows the EDB website with the URL enterpriseedb.com/downloads/postgres-postgresql-downloads. The page title is "Download PostgreSQL". It features a navigation bar with links like "Why EDB?", "Cloud PostgreSQL", "PostgreSQL Software", "Services & Support", "Resources", and "Plans". A prominent orange "Get started" button is at the top right. Below the title, it says "Open source PostgreSQL packages and installers from EDB". A table lists PostgreSQL versions for different platforms: 14.2 (Linux x86-64, Linux x86-32, Mac OS X, Windows x86-64, Windows x86-32). The Windows x86-64 button is circled in red.

→ for the UI, we need other tool PgAdmin. It is PostgreSQL tool.

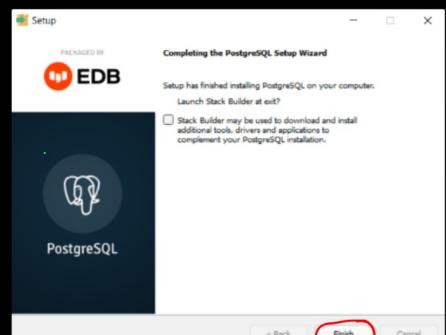
The screenshot shows the pgAdmin 4 (Windows) download page with the URL pgadmin.org/download/pgadmin-4-windows/. The page has a sidebar with "Quick Links" for "Online Demo", "Download", "FAQ", and "Support". The main content area is titled "pgAdmin 4 (Windows)". It includes a "Download" section with a "Download" button. Below it, it says "Maintainer: pgAdmin Development Team" and provides compatibility information for Windows 7 SP1, 2008R2, 8, and 2012. It also mentions support for v5.0 and later on Windows 8/2012 and v4.29 on 32-bit Windows. A list of packages is shown, with "pgAdmin 4 v6.8 (released April 7, 2022)" highlighted with a red arrow pointing to the "Download" link.

⇒ PostgreSQL Installation:

keep all as mentioned then give the password for the database superuser - 'postgres'.



→ Port number - 5432
(fixed by default)



→ Now we need to install PgAdmin.

keep all the default values &
click next > Install > Finish.

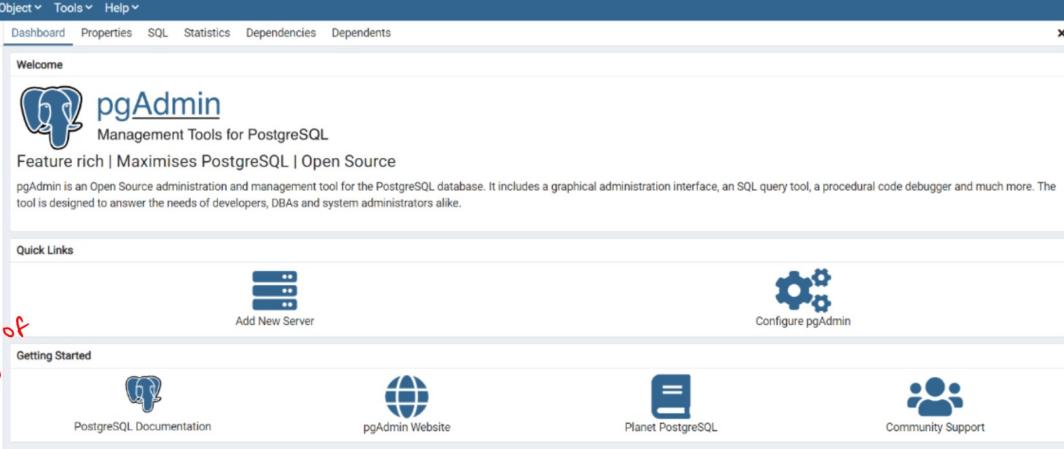
→ Open the pgAdmin in Administrator Mode.



pgAdmin 4

Servers

Click on Servers & enter the password which we configured while setup of PostgreSQL.

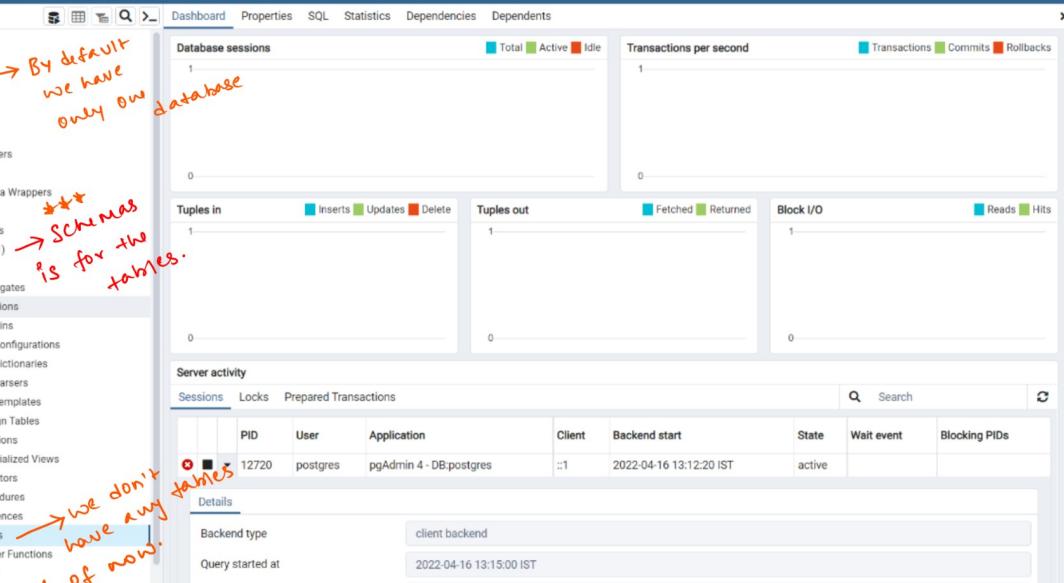


The screenshot shows the pgAdmin 4 interface. The top menu bar includes File, Object, Tools, and Help. Below it is a toolbar with icons for Browser, Dashboard, Properties, SQL, Statistics, Dependencies, and Dependents. The main area has a 'Welcome' header with the pgAdmin logo and a brief description. It features a 'Quick Links' section with 'Add New Server' and 'Configure pgAdmin' buttons, and a 'Getting Started' section with links to PostgreSQL Documentation, pgAdmin Website, Planet PostgreSQL, and Community Support.

By default we have only one database.

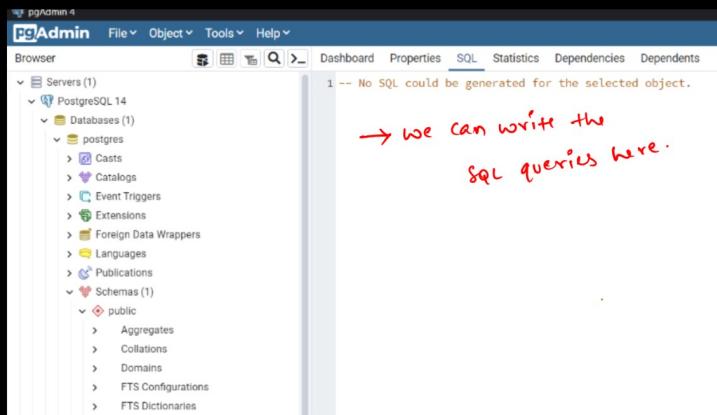
Schemas is for the tables.

We don't have any tables now.



The screenshot shows the pgAdmin 4 interface with the browser tree expanded to show a single PostgreSQL 14 server with one database named 'postgres'. The database tree shows various objects like casts, catalogs, event triggers, extensions, foreign data wrappers, languages, publications, and schemas. The main pane displays three monitoring dashboards: 'Database sessions', 'Transactions per second', and 'Server activity'. The 'Database sessions' dashboard shows session details for 'postgres' on port ':1'. The 'Transactions per second' dashboard shows transaction metrics. The 'Server activity' dashboard shows server statistics and a table of active sessions.

→ we can write the SQL queries here.

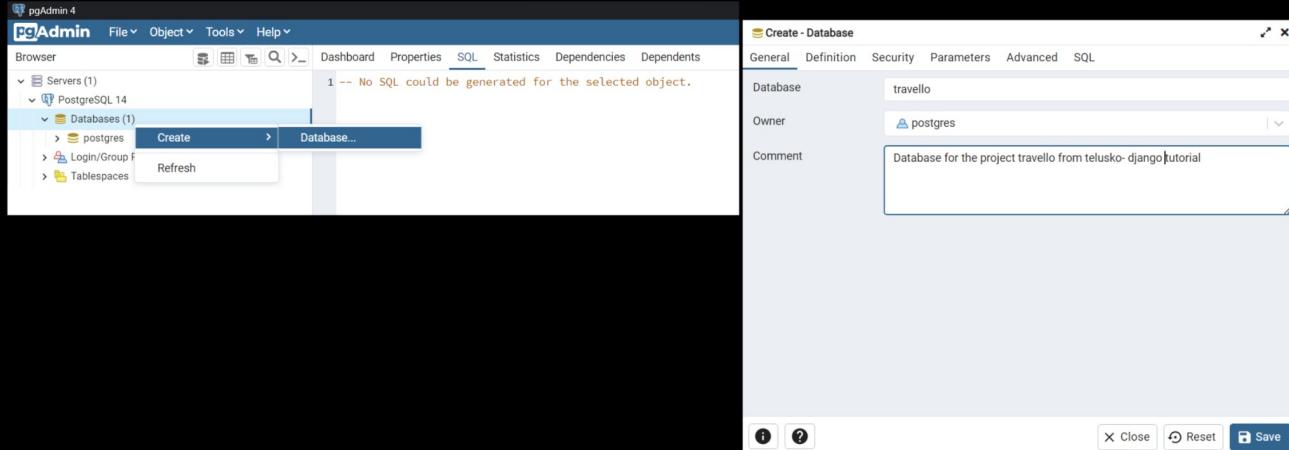


The screenshot shows the pgAdmin 4 interface with the browser tree expanded to show a single PostgreSQL 14 server with one database named 'postgres'. The main pane displays a SQL query editor window with the message '1 -- No SQL could be generated for the selected object.' A red annotation points to the bottom right of the editor with the text '→ we can write the SQL queries here.'

Models & Migrations

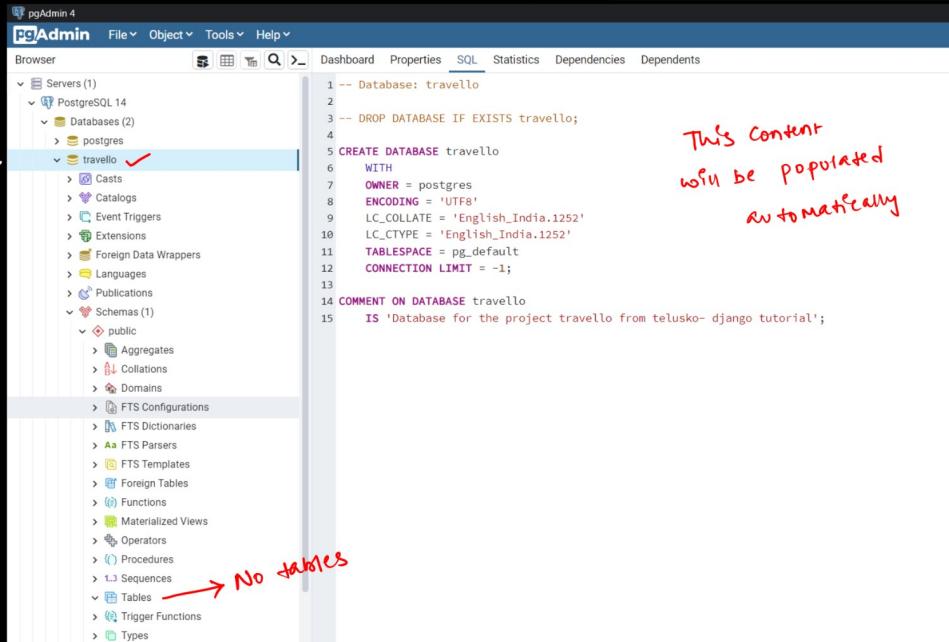
①

→ Create a database



This content
will be populated
automatically

Database created



★ ② → Now we need to connect the Postgres database to our application in our project.

```

    WSGI_APPLICATION = 'travello.wsgi.application'
    # https://docs.djangoproject.com/en/4.0/ref/settings/#databases
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': BASE_DIR / 'db.sqlite3',
        }
    }
    # Password validation
    # https://docs.djangoproject.com/en/4.0/ref/settings/#auth-password-

```

by default django takes
SQLite3 database. we need to
change it to PostgreSQL.

```

    WSGI_APPLICATION = 'travello.wsgi.application'
    # https://docs.djangoproject.com/en/4.0/ref/settings/#databases
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.postgresql',
            'NAME': 'travello', # database name
            'USER': 'postgres', # database user
            'PASSWORD': '...', # database password
            'HOST': 'localhost', # database host
        }
    }

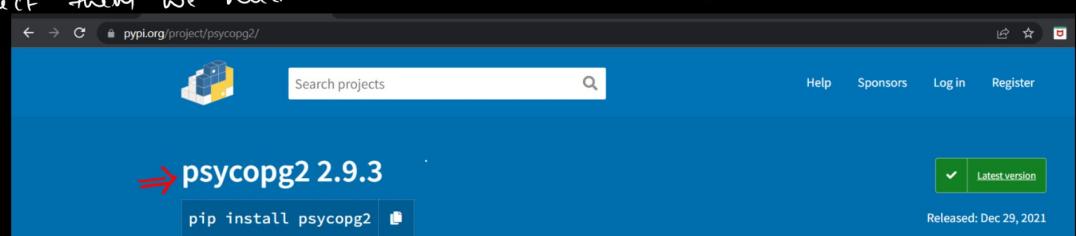
```

Connecting to postgresql

(we need to give the database
name that we had already
create in PgAdmin)

★ ③ → Django and PostgreSQL are different softwares.

so, to connect them we need a connector. → psycopg2



Released: Dec 29, 2021

Psycopg is the most popular PostgreSQL database adapter for the Python programming language.

→ pip install psycopg2

```
(django telusko) D:\Edu\ Django telusko\travello>pip install psycopg2 ✓
Collecting psycopg2
  Downloading psycopg2-2.9.3-cp310-cp310-win_amd64.whl (1.2 MB)
    1.2 MB 3.2 MB/s
Installing collected packages: psycopg2
Successfully installed psycopg2-2.9.3 ✓
(django telusko) D:\Edu\ Django telusko\travello>
```

→ The ORM feature of Django will create the database tables for us.

The tables creation depends on our model (because we define the classes in it).

④ ***

→ In models.py, we have a simple class.
we need to convert it into model.

```
tapp > models.py > Destination
1 from django.db import models
2
3 # Create your models here.
4 # Models are used to connect with databases and data.
5
6 class Destination:
7     id : int
8     name : str
9     img : str
10    desc : str
11    price : int
12    offer : bool
```

→ Converting class to model:

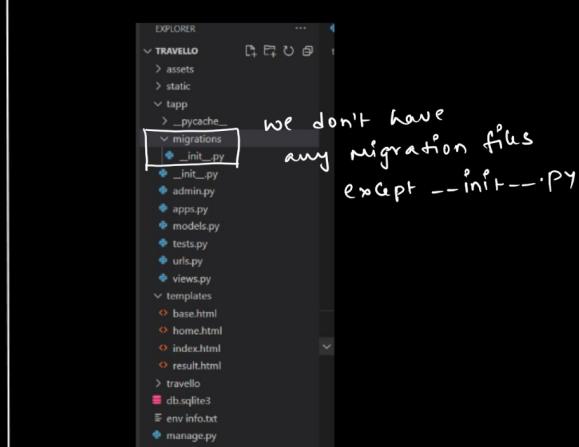
```
tapp > models.py > Destination
1 from django.db import models
2
3 # Create your models here.
4 # Models are used to connect with databases and data.
5
6 class Destination(models.Model): # we are inheriting the feature of models. ✓
7     # https://docs.djangoproject.com/en/4.0/ref/models/fields/#autofield
8     name = models.CharField(max_length=100) ←
9     img = models.ImageField(upload_to='pics') # basically we upload the images to a specific folder and we need to give that folder name here.
10    desc = models.TextField()
11    price = models.IntegerField()
12    offer = models.BooleanField(default=False)
```

defining the datatypes of the fields/columns
(we get the datatypes info in
django model fields documentation)

(Class name = Destination)

⑤ ***

Now if we want to create a table, we need to migrate our models to the database.



```
tapp > settings.py > ...
28 ALLOWED_HOSTS = []
29
30 # Application definition
31
32 INSTALLED_APPS = [
33     'tapp.apps.TappConfig', ←
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40 ]
41
42
43
```

appname
tapp.apps.TappConfig
we need to mention the app
on which we are working.
in the 'Installed-Apps' of
settings.py

To migrate the command is: python manage.py makemigrations

```
(django telusko) D:\Edu\ Django telusko\travello>python manage.py makemigrations
SystemCheckError: System check identified some issues:
ERRORS:
tapp.Destination.img: (fields.E210) Cannot use ImageField because Pillow is not installed.
  Hint: Get Pillow at https://pypi.org/project/Pillow/ or run command 'python -m pip install Pillow' (Error)
```

Collecting Pillow
 Downloading Pillow-9.1.0-cp310-cp310-win_amd64.whl (3.3 MB)
 3.3 MB 6.8 MB/s
Installing collected packages: Pillow
Successfully installed Pillow-9.1.0

⑦ installing pillow

To work with images
we need the library
(pillow)

Migration file

```

models.py views.py settings.py 0001_initial.py index.html
...
1 # Generated by Django 4.0.4 on 2022-04-16 09:53
2
3 from django.db import migrations, models
4
5
6 class Migration(migrations.Migration):
7     initial = True
8
9     dependencies = [
10         ...
11     ]
12
13     operations = [
14         migrations.CreateModel(
15             name='Destination',
16             fields=[
17                 ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
18                 ('name', models.CharField(max_length=100)),
19                 ('img', models.ImageField(upload_to='pics')),
20                 ('desc', models.TextField()),
21                 ('price', models.IntegerField()),
22                 ('offer', models.BooleanField(default=False)),
23             ],
24         ),
25     ]
26

```

PROBLEMS OUTPUT DEBUG CONSOLE

(django_telusko) D:\Edu\ Django_telusko\travello>python manage.py makemigrations
Migrations for 'tapp':
 tapp\migrations\0001_initial.py - Create model Destination

(django_telusko) D:\Edu\ Django_telusko\travello>

⑧ we had made/ created the migration file now we need to migrate it to

Create the table.

⇒ python manage.py sqlmigrate tapp 0001
app name ↑
migration file name ↑

This query will be executed

```

(django_telusko) D:\Edu\ Django_telusko\travello>python manage.py makemigrations
Migrations for 'tapp':
    tapp\migrations\0001_initial.py - Create model Destination

```

⑨ → The actual migration ⇒ python manage.py migrate

```

(django_telusko) D:\Edu\ Django_telusko\travello>python manage.py makemigrations
Migrations for 'tapp':
    tapp\migrations\0001_initial.py - Create model Destination

```

```

(django_telusko) D:\Edu\ Django_telusko\travello>python manage.py sqlmigrate tapp 0001
BEGIN;
-- Create model Destination
CREATE TABLE "tapp_destination" ("id" bigserial NOT NULL PRIMARY KEY, "name" varchar(100) NOT NULL, "img" varchar(100) NOT NULL, "desc" text NOT NULL, "price" integer NOT NULL, "offer" boolean NOT NULL);
COMMIT;

```

* * migration done & table is created.

Tables created

Browser

Tables (11)

- auth_group
- auth_group_permissions
- auth_permission
- auth_user
- auth_user_groups
- auth_user_permissions
- django_admin_log
- django_content_type
- django_migrations
- django_session
- tapp_destination**

Columns (6)

- id
- name
- img
- desc
- price
- offer

Constraints

Indexes

RLS Policies

Rules

Triggers

Trigger Functions

Types

Views

```

1 -- Table: public.tapp_destination
2
3 -- DROP TABLE IF EXISTS public.tapp_destination;
4
5 CREATE TABLE IF NOT EXISTS public.tapp_destination
6 (
7     id bigint NOT NULL DEFAULT nextval('tapp_destination_id_seq'::regclass),
8     name character varying(100) COLLATE pg_catalog."default" NOT NULL,
9     img character varying(100) COLLATE pg_catalog."default" NOT NULL,
10    "desc" text COLLATE pg_catalog."default" NOT NULL,
11    price integer NOT NULL,
12    offer boolean NOT NULL,
13    CONSTRAINT tapp_destination_pkey PRIMARY KEY (id)
14 )
15
16 TABLESPACE pg_default;
17
18 ALTER TABLE IF EXISTS public.tapp_destination
19     OWNER to postgres;

```

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Tables (11)', there is a new entry for 'tapp_destination'. A context menu is open over this entry, with 'View/Edit Data' selected. Under 'View/Edit Data', 'All Rows' is highlighted. The main pane shows a query editor with the following SQL:

```
1 SELECT * FROM public.tapp_destination
2 ORDER BY id ASC
```

The screenshot shows the pgAdmin 4 interface. The 'tapp_destination' table is now listed under 'Tables (11)'. A handwritten note on the left side of the screen says: 'table created → appname - class name (class in models.py)'. The table structure is displayed in the main pane.

Re-Migration

→ If we miss any fields (or) make any mistakes in the `models.py` file.

→ If we miss any fields (or) make any mistakes in the `models.py` file.

We need to correct them & remigrate again.

for eg.: In the tutorial, there was a mistake in the `price` field. So, the `'price'` column didn't

created in the table.

```
1 from django.db import models
2
3 # Create your models here.
4
5
6 class Destination(models.Model):
7
8     name = models.CharField(max_length=100)
9     img = models.ImageField(upload_to='pics')
10    desc = models.TextField()
11    price = models.IntegerField()  # parenthesis is missing
12    offer = models.BooleanField(default=False)
```

A handwritten note next to the code says: 'parenthesis is missing'.

The screenshot shows the pgAdmin 4 interface. The 'tapp_destination' table is listed under 'Tables (11)'. The table structure is shown in the main pane, but the `'price'` column is missing from the list.

→ Migrating again:

Creating the migration file ↑

(test) C:\Users\Navin\projects\telusko>python manage.py makemigrations ✓
You are trying to add a non-nullable field 'price' to destination without a default

Please select a fix:

- 1) Provide a one-off default now (will be set on all existing rows with a null value)
- 2) Quit, and let me add a default in `models.py`

Select an option: 1

Please enter the default value now, as valid Python

The `datetime` and `django.utils.timezone` modules are available, so you can do e.g. this

Type 'exit' to exit this prompt

>>> 0 → default value

Migrations for 'travello':

travello\migrations\0002_destination_price.py

- Add field price to destination

```
1 from django.db import models
2
3 # Create your models here.
4
5
6 class Destination(models.Model):
7
8     name = models.CharField(max_length=100)
9     img = models.ImageField(upload_to='pics')
10    desc = models.TextField()
11    price = models.IntegerField() ✓
12    offer = models.BooleanField(default=False)
```

The terminal window shows the command `python manage.py migrate` being run. It displays the operations to perform and the fact that no migrations are applied.

```
(test) C:\Users\Navin\projects\telusko>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, travello
Running migrations:
  No migrations to apply.
```

A handwritten note says: 'migrating to database'.

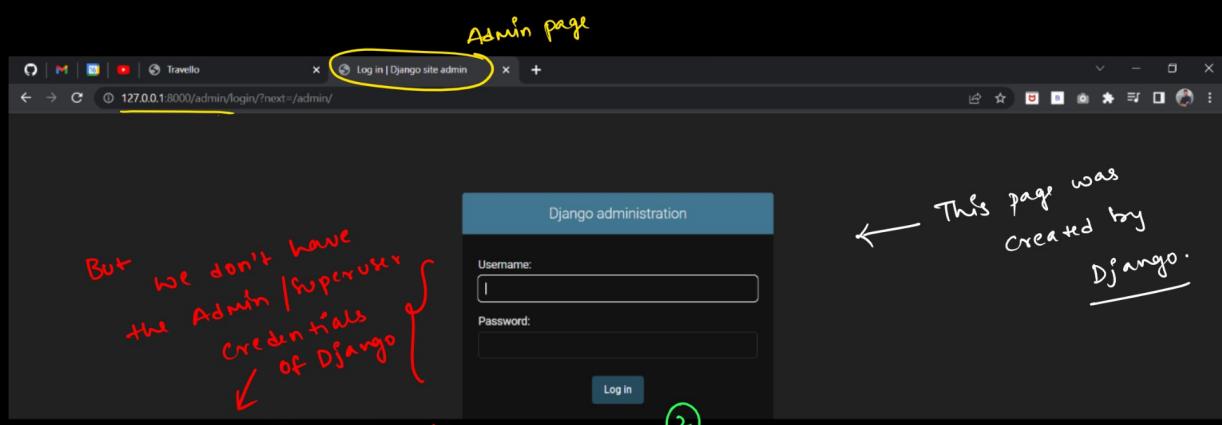
New migration file →

Admin panel:

- Now we have the database/table but not data in it.
- ⇒ we need to push data into the database table. This can be done in 2 ways.
- ↳ By creating a page into which the user can push/post the data.
 - (or) ↳ we can have the Admin page.
 - ↙ Is accessible by Admins/supervisors.
but
↑ we have
- A normal user can push the data by in our project which is added by the admins.

① Let's go the Admin page:

→ we can access the admin page - http://127.0.0.1:8000/admin



② Creating super user:

python manage.py createsuperuser

```
(django_telusko) D:\Edu\ Django_telusko\travello>python manage.py createsuperuser
Username (leave blank to use 'krishnaleti'): krish ✓
Email address: krish@opmail.com ✓
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

→ Note: python manage.py help
gives the subcommands.

```
(django_telusko) D:\Edu\ Django_telusko\travello>python manage.py help
Type 'manage.py help <subcommand>' for help on a specific subcommand.

Available subcommands:

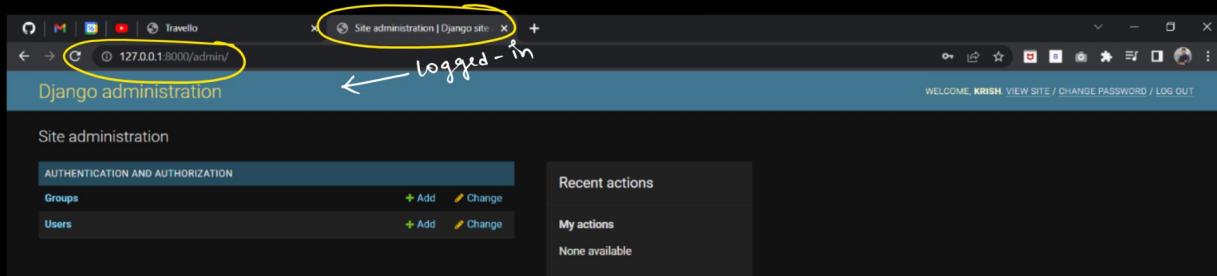
[auth]
    changepassword

[django]
    check
    compilemessages
    createcachetable
    dbshell
    diffsettings
    dumpdata
    flush
    inspectdb
    loaddata
    loadmessages
    makemessages
    makemigrations
    migrate
    sendtestemail
    shell
    showmigrations
    sqlflush
    sqlmigrate
    sqlsequencereset
    squashmigrations
    startapp
    startproject
    test
    testserver

[sessions]
    clearsessions

[staticfiles]
```

③ login using the created superuser credentials.



→ In the travello website/page, we need to add more places i.e., add data.

we want to work with the database.

⇒ ④ I want to create a way to add destinations/places in the database.

it means we have to create a page.

⇒ we just need to register our model/class in the models.py
the page will be created automatically.

```
tapp > admin.py
1 from django.contrib import admin
2 from .models import Destination # importing the class or model
3 # Register your models here.
4
5
# here we need to register the model and automatically it will create a page for us.
6 admin.site.register(Destination) # need to give the model name here
```

page created.

Now we can add the data directly from here.

page created

Add destination

Name:

Img: Choose File No file chosen

Desc:

Price:

Offer

Add and Fetch data from the database

→ In the above page the user can give the required data but for the images, the user need to choose the image.

⇒ ① We need to inform Django that whatever image the user select save all of those images in the 'Media-Root' path i.e., in the 'media' folder.

Then we need to add the 'media-url', media-root in the url-patterns of urls.py of project.

Creating media-ROOT (path to create the 'media' folder & save the images in it)

media url

Creating media-ROOT (path to create the 'media' folder & save the images in it)

folder name to create

'Media' folder will be created in base directory

media-url media path

needed to add path of the 'media' folder & its url.

→ Download images from the internet.

(we will give these while creating the destinations)

Name	Date modified	Type
Today (5)		
bbsr	16-04-2022 06:35 PM	JPEG File
bengaluru	16-04-2022 06:33 PM	JPG File
Kolkata	16-04-2022 06:33 PM	JPG File
Hyderabad	16-04-2022 06:32 PM	JPG File
mumbai	16-04-2022 06:31 PM	JPG File

②

→ Adding the Destinations:

Django administration

Add destination

Name: Mumbai

Img: Choose File No file chosen

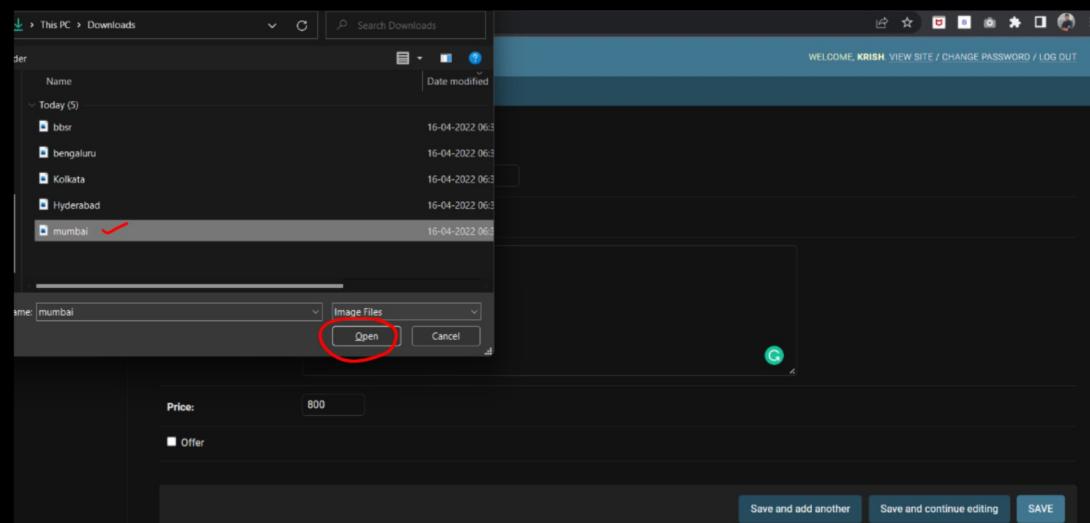
Desc: Hotel taj

Price: 800

Offer:

[Save and add another](#) [Save and continue editing](#) [SAVE](#)

Enter the Name, Desc, price &
for img.
↓
click 'choose file'



Django administration

Add destination

Name: Mumbai

Img: Choose File mumbai.jpg

Desc: Hotel taj

Price: 800

Offer:

[Save and add another](#) [Save and continue editing](#) [SAVE](#)

once we choose the image,
'media' folder will be created &
the image will be saved in that folder.
(for the next destinations, the images chosen
will be added in the 'media' folder).

we want to add
another destination

Django administration

Add destination

Name: [empty input]

Img: Choose File No file chosen

[Save and add another](#) [Save and continue editing](#) [SAVE](#)

The destination "Destination object (1)" was added successfully. You may add another destination below.

→ Similarly add the destinations:

The screenshot shows the Django Admin interface for the 'Destinations' model. A green success message at the top right says 'The destination "Destination object (7)" was added successfully.' Below it, a list of destination objects is shown, numbered 1 through 7. A handwritten note 'Destination Added' is written above the list. A curly brace on the left side groups the destination objects.

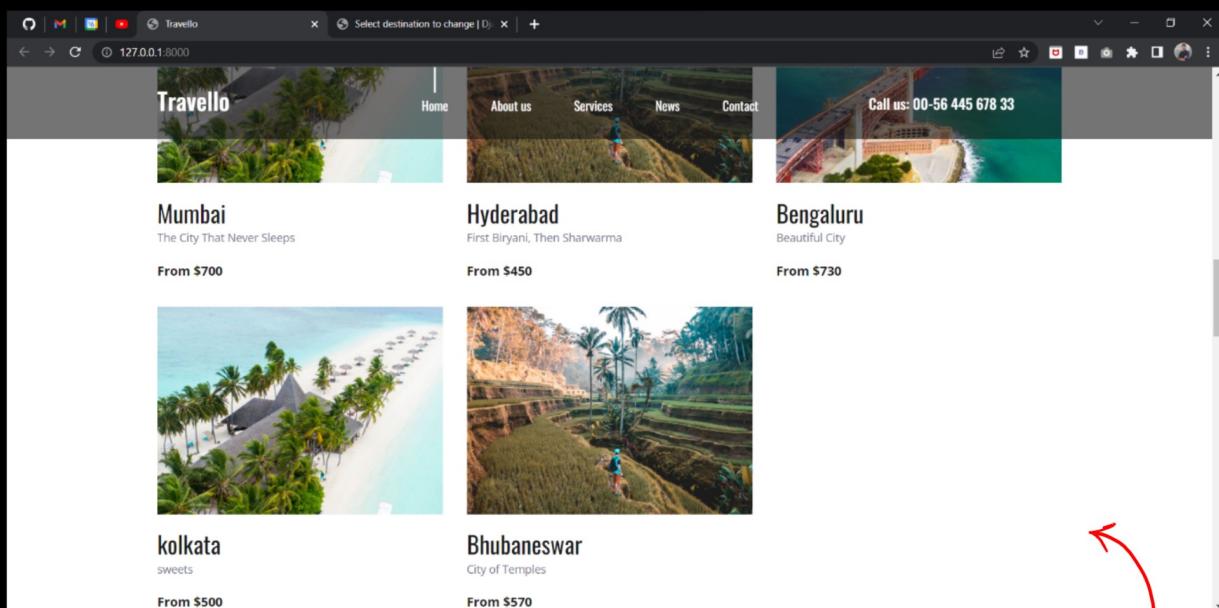
→ As soon as we choose a file for the image for the 1st destination, the folder 'media' will be created in 'base directory' as mentioned in 'MEDIA-ROOT' & the images we select for the destinations will be saved

This screenshot shows a Visual Studio Code window with two files open: 'settings.py' and 'urls.py'. The 'settings.py' file has several lines highlighted with yellow boxes. One line, 'MEDIA_ROOT = os.path.join(BASE_DIR, 'media')', is specifically highlighted. A handwritten note 'media folder created & all the images are saved.' points to this line. The 'urls.py' file also has some code highlighted. A handwritten note 'in the media folder.' points to the 'MEDIA_ROOT' line in 'settings.py'.

③ Data will be added in the Database - Table automatically.

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane lists various database objects like sequences, tables, and triggers. An arrow points to the 'tapp_destination' table under the 'Tables (11)' section. The 'Columns (6)' section is expanded, showing columns: id, name, img, desc, price, and offer. In the center, the 'Query Editor' pane contains a simple SQL query: 'SELECT * FROM public.tapp_destination ORDER BY id ASC'. The results are displayed in a table below, with a red box highlighting the entire data row. A red arrow points to the bottom right of the table with the handwritten note 'Data added.' A status bar at the bottom right says 'Successfully run. Total query runtime: 129 msec. 7 rows affected.'

→ we are still getting the details which we had passed in 'views.py' previously.



```
views.py
models.py
admin.py
settings.py
urls.py
0001_initial.py
index.html

13 dest1 = Destination()
14 dest1.name = 'Mumbai'
15 dest1.desc = 'The City That Never Sleeps'
16 dest1.img = 'destination_1.jpg' # we need to mention the name of image which is in the images folder od static folder
17 dest1.price = 700
18 dest1.offer = False
19
20
21 dest2 = Destination()
22 dest2.name = 'Hyderabad'
23 dest2.desc = 'First Biryani, Then Sharwarma'
24 dest2.img = 'destination_2.jpg'
25 dest2.price = 450
26 dest2.offer = True
27
28 dest3 = Destination()
29 dest3.name = 'Bengaluru'
30 dest3.desc = 'Beautiful City'
31 dest3.img = 'destination_3.jpg'
32 dest3.price = 730
33 dest3.offer = False
34
35 dest4 = Destination()
36 dest4.name = 'Kolkata'
37 dest4.desc = 'sweets'
38 dest4.img = 'destination_1.jpg'
39 dest4.price = 500
40 dest4.offer = False
41
42 dest5 = Destination()
43 dest5.name = 'Bhubaneswar'
44 dest5.desc = 'City of Temples'
45 dest5.img = 'destination_2.jpg'
46 dest5.price = 570
47 dest5.offer = False
48
49 dests = [dest1, dest2, dest3, dest4, dest5]
return render(request, 'index.html', {'dests': dests, 'discount': '10%'})
```

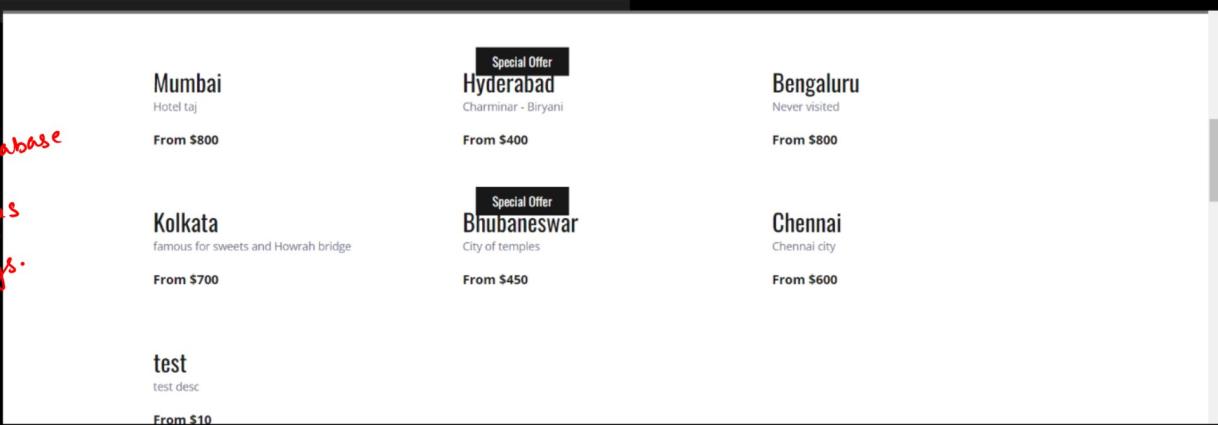
⇒ we don't want to write the data like this & make it display in the browser. We want to fetch the data from the database dynamically.

④ → We need to display the data that we had added in the forms & that got inserted into the database-table.
⇒ We need to fetch the data from database and display it in browser.

```
views.py
models.py
admin.py
settings.py
urls.py
0001_initial.py
index.html

1 from django.shortcuts import render, HttpResponseRedirect
2 from django.http import HttpResponseRedirect
3 # from . import models
4 from .models import Destination
5
6
7 # Create your views here.
8
9 # def index(request):
10 #     return render (request, 'index.html',{'discount':'10%'})
11
12 def index(request):
13
14     dests = Destination.objects.all() ← fetching all the objects from the database
15     #we are getting all the data from the database and storing it in a 'dests' variable.
16
17     return render(request, 'index.html', {'dests': dests,'discount':'10%'})
```

We got the destinations from the database
BUT the images are missing.



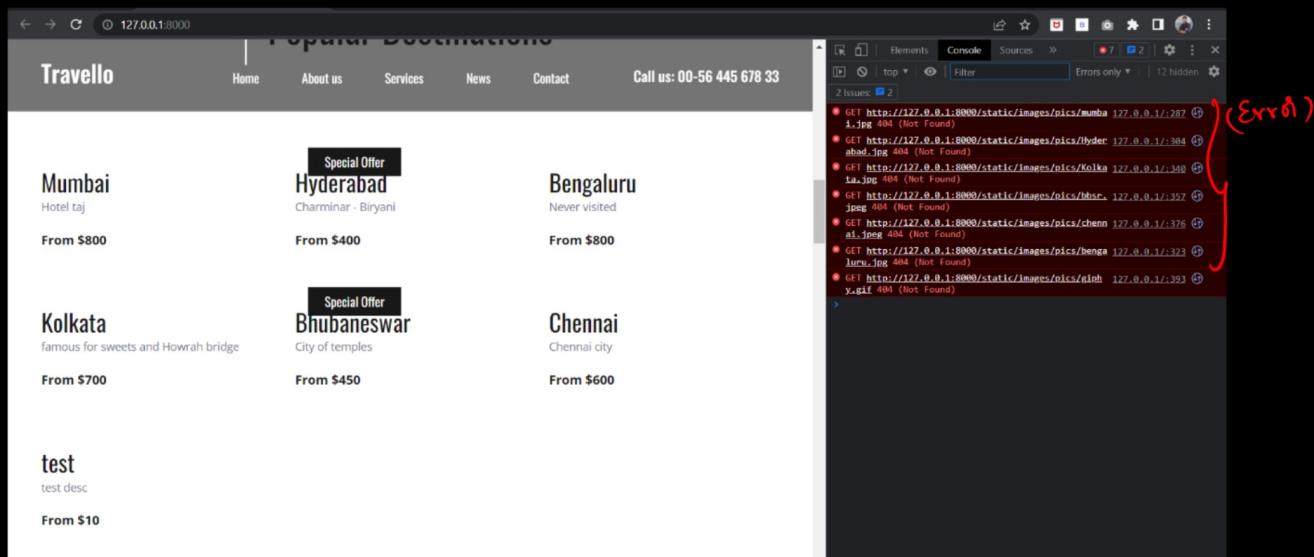
**** Steps to display the dynamic data. from the database.

Admin creates the page → Admin adds the data in the page → media files will be stored in the defined path

↓

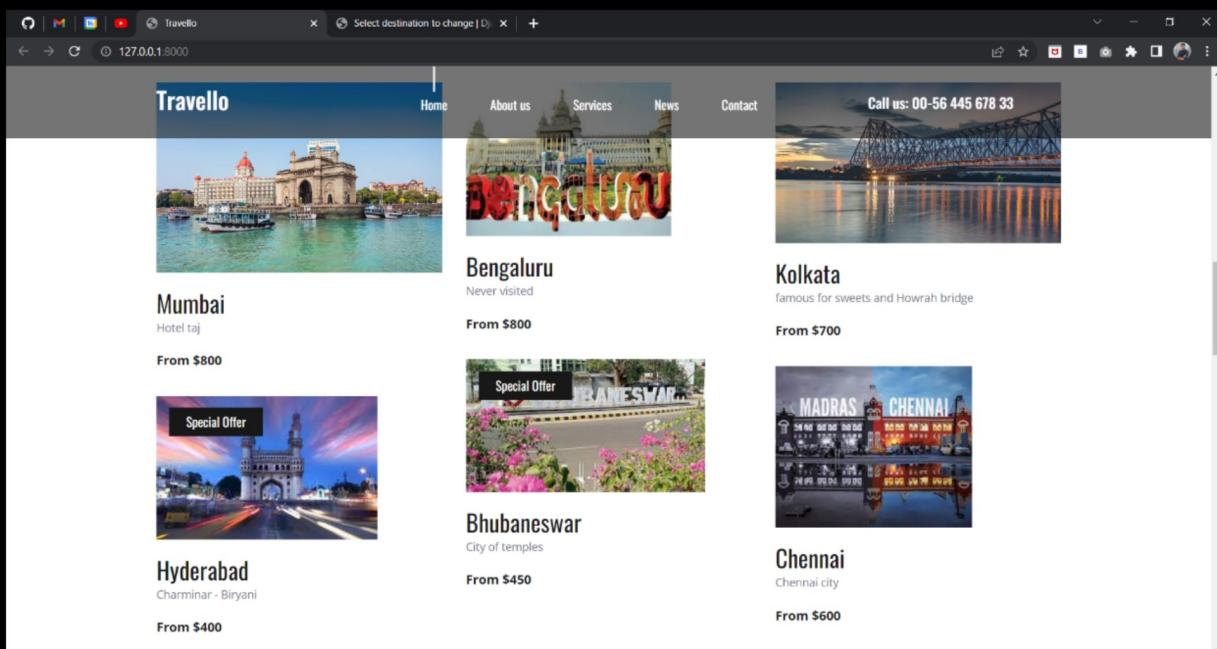
Data fetched from the database will be displayed in the browser. ← The data will be inserted into the database - table automatically

→ here the destinations^{details} are getting displayed except the images.



→ The images are not displayed because in the index.html we had given the 'static_images' path for images (we used it previously \Rightarrow we need to change the path) + this

⇒ Destinations are fetched from database & displayed in browser.



(I had removed the 'zero' place)

User Registration in Django :

We will creates 2 pages - Registration, login.

→ Now we can create them in the same app but lets try something new.

lets create another app & keep the Registration, login features in it.

The screenshot shows a code editor with several files open:

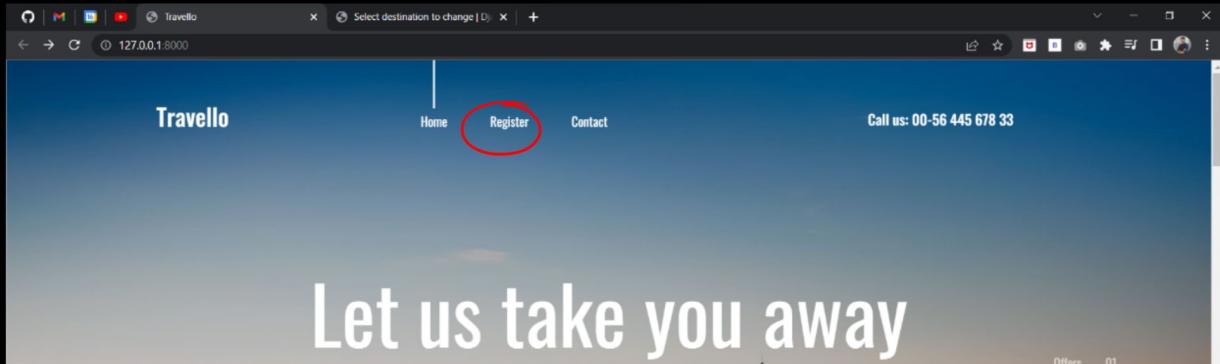
- accounts** folder: migrations, __init__.py, admin.py, apps.py, models.py, tests.py, views.py.
- index.html**: A template file with code for displaying destinations.
- views.py**: Contains a register function.
- models.py**: Contains destination models.
- admin.py**: Registers the destination model in the admin panel.
- urls.py**: Includes the accounts app's URLs.
- settings.py**: Configuration settings.
- index.html**: Another template file.

In the terminal, the command `python manage.py startapp accounts` is run, and the output shows the app has been created.

The screenshot shows two template files:

- index.html**: Contains a header section with a logo and navigation links.
- register.html**: Contains a registration form with fields for name, email, password, and a checkbox for terms and conditions.

we access admin page as localhost/admin
accounts - localhost/accounts.



```
index.html  urls.py travello  urls.py accounts  views.py  register.html
accounts > views.py > register
1 from django.shortcuts import render
2
3 # Create your views here.
4
5 def register(request):
6     return render(request, 'register.html')
```

→ we need to create register.html

⇒ we will register the users, but where will we store the details of the registered users. → we need to create a table for it. --- NO

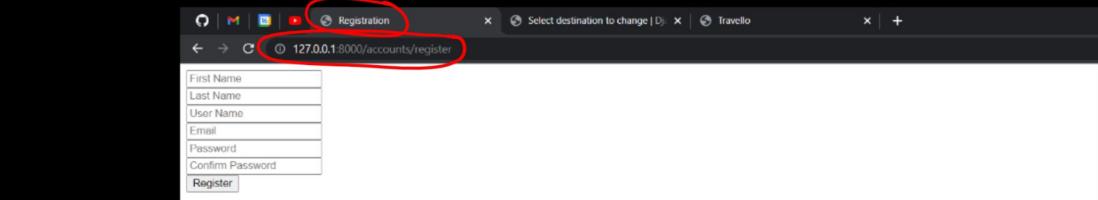
Because when we had migrated the models to database last time, few tables are created. → one of them is auth_user. The user details will be stored in this table.

ID	Password	Last Login	Is Superuser	Username	First Name	Last Name	Email
1	pbkdf2_sha256\$320000\$F...	2022-04-16 16:45:33.274888+05:30	true	krish			krish@yopmail.com

we need to create a registration form/page as per the fields in auth_user table.

Registration form

```
index.html  urls.py travello  urls.py accounts  views.py  register.html
templates > register.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset='utf-8'>
5     <meta http-equiv='X-UA-Compatible' content='IE=edge'>
6     <title>Registration</title>
7     <meta name='viewport' content='width=device-width, initial-scale=1'>
8     <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
9     <script src='main.js'></script>
10 </head>
11 <body>
12     <form action="register" method="post">
13         <% csrf_token %>
14
15         <input type="text" name="firstname" placeholder="First Name"><br>
16         <input type="text" name="lastname" placeholder="Last Name"><br>
17         <input type="text" name="username" placeholder="User Name"><br>
18
19         <input type="email" name="email" placeholder="Email"><br>
20         <input type="password" name="password1" placeholder="Password"><br>
21         <input type="password" name="password2" placeholder="Confirm Password"><br>
22         <input type="submit" value="Register">
23     </form>
24 </body>
25 </html>
```



```
index.html   urls.py travello   urls.py accounts   views.py   register.html
templates > register.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset='utf-8'>
5     <meta http-equiv='X-UA-Compatible' content='IE=edge'>
6     <title>Registration</title>
7     <meta name='viewport' content='width=device-width, initial-scale=1'>
8     <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
9     <script src='main.js'></script>
10    </head>
11    <body>
12        <form action="register" method="post">
13            {% csrf_token %}<br>
14            <input type="text" name="firstname" placeholder="First Name"><br>
15            <input type="text" name="lastname" placeholder="Last Name"><br>
16            <input type="text" name="username" placeholder="User Name"><br>
17            <input type="email" name="email" placeholder="Email"><br>
18            <input type="password" name="password1" placeholder="Password"><br>
19            <input type="password" name="password2" placeholder="Confirm Password"><br>
20            <input type="submit" value="Register">
21        </form>
22    </body>
23 </html>
```

```
index.html   register.html   urls.py accounts   views.py   urls.py travello
accounts > urls.py ...
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('register', views.register, name='register'),
6 ]
7
```

```
index.html   register.html   urls.py accounts   views.py   urls.py travello
accounts > views.py > register
1 from django.shortcuts import render
2
3 # Create your views here.
4
5 def register(request):
6     return render(request, 'register.html')
```

⇒ Here we are using the same function register() for fetching the form & returning register.html

→ When we are calling register.html we are sending GET request
 &
 When we are submitting data we are sending POST request.

```
index.html   register.html   urls.py accounts   views.py   urls.py travello
accounts > views.py > register
1 from django.shortcuts import render
2
3 # Create your views here.
4
5 def register(request):
6
7     if request.method == 'POST':
8         first_name = request.POST['firstname']
9         last_name = request.POST['lastname']
10        email = request.POST['email']
11        password1 = request.POST['password1']
12        password2 = request.POST['password2']
13
14    else:
15        return render(request, 'register.html')
```

we got the details of the registered user.

Now we need to save these in the database.

→ We have the 'ORM' of django, once we get the data we can push it to the data but we need to have the model.

If we have model object, we can set the data in it & save into the database directly.

⇒ But do we have model object for the users ???

↙
 we need not worry about the model object for the user,
 it is there in the django framework — we just have to use it.

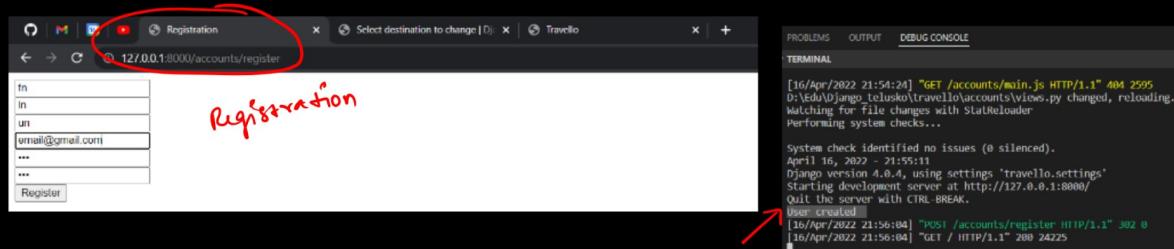
⇒ we need to import the 'User'

of 'accounts' app

```

1 from django.shortcuts import render, redirect
2 from django.contrib.auth.models import User, auth
3
4 # Create your views here.
5
6 def register(request):
7
8     if request.method == 'POST':
9         firstname = request.POST['firstname']
10        lastname = request.POST['lastname']
11        username = request.POST['username']
12        email = request.POST['email']
13        password1 = request.POST['password1']
14        password2 = request.POST['password2']
15
16        user=User.objects.create_user(username=username, password=password1, email=email, first_name=firstname, last_name=lastname)
17        user.save()
18        print('User created')
19        return redirect('/')
20    else:
21        return render(request, 'register.html')

```



auth_user

	id [PK] integer	password character varying(128)	last_login timestamp with time zone	is_superuser boolean	username character varying	first_name character varying	last_name character varying(150)	email character varying(254)	is_staff boolean	is_active boolean	date_joined timestamp
1	1	pbkdf2_sha256\$3200...	2022-04-16 16:4...	true	krish			krishi@yopmail.com	true	true	2022-04-16
2	2	pbkdf2_sha256\$3200...	[null]	false	mk	manoj	kumar	email1@yopmail.com	false	true	2022-04-16
3	3	pbkdf2_sha256\$3200...	[null]	false	un	fn	ln	email@gmail.com	false	true	2022-04-16

→ we need to verify few things while registration:

- 1, password matches with the confirm password
- 2, if the entered Username is already taken (present in database)
- 3, if the entered email already exists in database,
because no 2 users can have the same
username, email Id.

```

index.html   urls.py accounts   register.html   views.py   urls.py travello
accounts > views.py > register
1 from django.shortcuts import render, redirect
2 from django.contrib.auth.models import User, auth  # importing the user model. auth is for the login
3
4 # Create your views here.
5
6 def register(request):
7
8     if request.method == 'POST':
9         firstname = request.POST['firstname']
10        lastname = request.POST['lastname']
11        username = request.POST['username']
12        email = request.POST['email']
13        password1 = request.POST['password1']
14        password2 = request.POST['password2']
15
16        if password1==password2:
17            if User.objects.filter(username=username).exists(): #checking whether the username already exists in the database
18                print('error:username already exists')
19            elif User.objects.filter(email=email).exists():
20                print('error:email already taken')
21            else:
22                user=User.objects.create_user(username=username, password=password1, email=email, first_name=firstname, last_name=lastname)
23                user.save();
24                print('User created')
25                return redirect('/')
26            else:
27                print("The confirm password doesn't match with the entered password")
28        else:
29            return render(request,'register.html')

```

→ Here we are printing the error messages in the console.
 ⇒ we need to display them on the page.

Passing messages in Django:

```

register.html   views.py > register
accounts > views.py > register
1 from django.shortcuts import render, redirect
2 from django.contrib.auth.models import User, auth  # importing the user model. auth is for the login
3 from django.contrib import messages
4
5 # Create your views here.
6
7 def register(request):
8
9     if request.method == 'POST':
10        firstname = request.POST['firstname']
11        lastname = request.POST['lastname']
12        username = request.POST['username']
13        email = request.POST['email']
14        password1 = request.POST['password1']
15        password2 = request.POST['password2']
16
17        if password1==password2: # checking if the confirm password matches with the entered password
18            if User.objects.filter(username=username).exists(): # checking whether the username already exists in the database
19                messages.info(request, 'Username already exists') # message.info is the method in django to display the message
20                return redirect('register') # if we get the error we need get redirected to the registration page
21            elif User.objects.filter(email=email).exists(): # checking whether the email is taken and already exists in the database
22                messages.info(request, 'Email already taken')
23                return redirect('register') # if we get the error we need get redirected to the registration page
24            else:
25                user=User.objects.create_user(username=username, password=password1, email=email, first_name=firstname, last_name=lastname)
26                user.save();
27                print('User created')
28                return redirect('/')
29            else:
30                messages.info(request, 'confirm password does not match with the entered password...')
31                return redirect('register') # if we get the error we need get redirected to the registration page
32        return redirect('/')
33
34    else:
35        return render(request,'register.html')

```

In django we have built-in method 'messages' to display the messages on the pages.

```

templates > register.html
1 (% load static %)
2 <!DOCTYPE html>
3 <html>
4 <head>
5     <meta charset='utf-8'>
6     <meta http-equiv='X-UA-Compatible' content='IE=edge'>
7     <title>Registration</title>
8     <meta name='viewport' content='width=device-width, initial-scale=1'>
9     (% comment %) <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
10    <script src='main.js'></script> (% endcomment %)
11
12 </head>
13 <body>
14     <form action="register" method="post">
15         (% csrf_token %)
16
17         <input type="text" name="firstname" placeholder="First Name"><br>
18         <input type="text" name="lastname" placeholder="Last Name"><br>
19         <input type="text" name="username" placeholder="User Name"><br>
20         <input type="email" name="email" placeholder="Email"><br>
21         <input type="password" name="password1" placeholder="Password"><br>
22         <input type="password" name="password2" placeholder="Confirm Password"><br>
23         <input type="submit" value="Submit" />
24     </form>
25     <div>
26         (% for message in messages %)
27             {{message}}
28         (% endfor %)
29     </div>
30 </body>
31 </html>

```

Data Output	Explain	Messages	Notifications
<code>id</code>	<code>password</code>	<code>last_login</code>	<code>is_superuser</code>
<code>1</code>	<code>pokd2_sha2...</code>	<code>2022-04-16 1...</code>	<code>True</code>
<code>2</code>	<code>pokd2_sha2...</code>	<code>[null]</code>	<code>False</code>
<code>3</code>	<code>pokd2_sha2...</code>	<code>[null]</code>	<code>False</code>
<code>4</code>	<code>pokd2_sha2...</code>	<code>[null]</code>	<code>False</code>

Database →

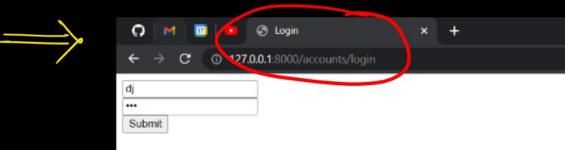
ID	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active	date_joined
1	pokd2_sha2...	2022-04-16 1...	True	krish			krish@yopmail.com	True	True	2022-04-16 16:40:02,566712+05:30
2	pokd2_sha2...	[null]	False	learner	coder		lc@yopmail.com	False	True	2022-04-17 07:04:25,347306+05:30
3	pokd2_sha2...	[null]	False	un	fn	In	email@yopmail.com	False	True	2022-04-17 07:08:29,771166+05:30
4	pokd2_sha2...	[null]	False	dj	django		user@yopmail.com	False	True	2022-04-17 07:12:08,774945+05:30

User login

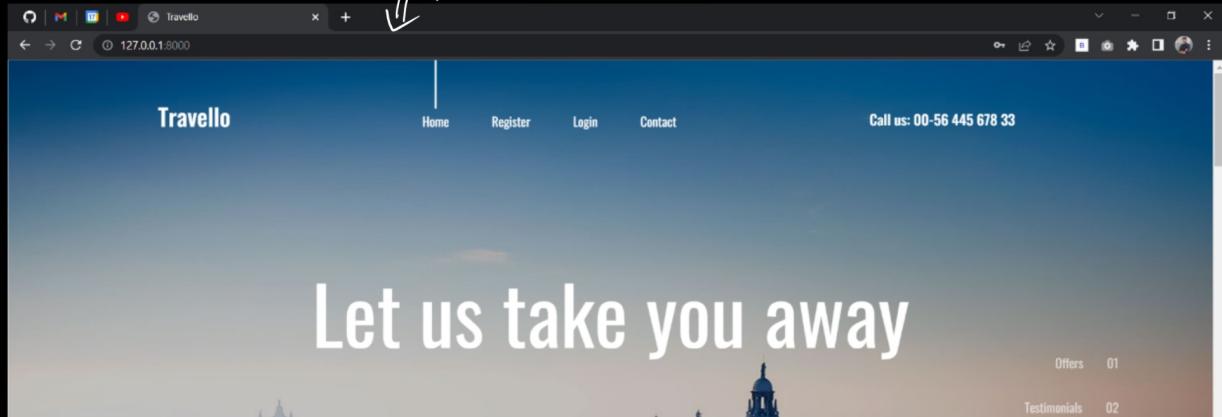
```
register.html <-- login.html x views.py index.html urls.py accounts urls.py travelo
templates > login.html
1  {% load static %}
2  <!DOCTYPE html>
3  <html>
4  <head>
5      <meta charset='utf-8'>
6      <meta http-equiv='X-UA-Compatible' content='IE=edge'>
7      <title>Login</title>
8      <meta name='viewport' content='width=device-width, initial-scale=1'>
9      {% comment %}<link rel='stylesheet' type='text/css' media='screen' href='main.css'>
10     <script src='main.js'></script> {% endcomment %}
11 </head>
12 <body>
13     <form action="login" method="post">
14         {% csrf_token %}
15
16         <input type="text" name="username" placeholder="User Name"><br>
17         <input type="password" name="password" placeholder="Password"><br>
18         <input type="submit" >
19     </form>
20
21     <div>
22         {% for message in messages %}
23             {{message}} </h3>
24         {% endfor %}
25     </div>
26
27 </body>
28 </html>
29
```

```
accounts > views.py < login
1 from django.shortcuts import render, redirect
2 from django.contrib.auth.models import User, auth    # importing the user model. auth is for the login
3 from django.contrib import messages
4
5 # Create your views here.
6
7 def login(request):
8     if request.method == 'POST':
9         username = request.POST['username']
10        password = request.POST['password'] # we need to verify the login creds now
11
12        user = auth.authenticate(username=username, password=password) # auth is for the login
13        # if the username and password exists, it returns the user object which means it is not none
14
15        if user is not None:
16            auth.login(request, user) # we are providing the login access to the current user
17            return redirect('/') # we nee to redirect to the home page if the login is successful.
18        else:
19            messages.error(request, 'Invalid credentials')
20            return redirect('login') # if the user is not authenticated, we need to redirect to the login page
21
22    else:
23        return render(request, 'login.html')
```

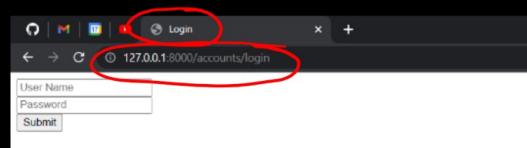
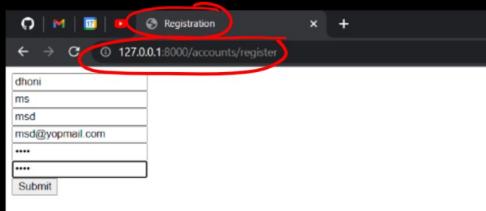
```
o register.html o login.html o views.py o index.html o urls.py accounts > uufs.py accounts > urls.py travello  
accounts > urls.py ...  
1 from django.urls import path  
2 from . import views  
3  
4 urlpatterns = [  
5     path("register", views.register, name="register"),  
6     path("login", views.login, name="login"),  
7 ]  
8
```

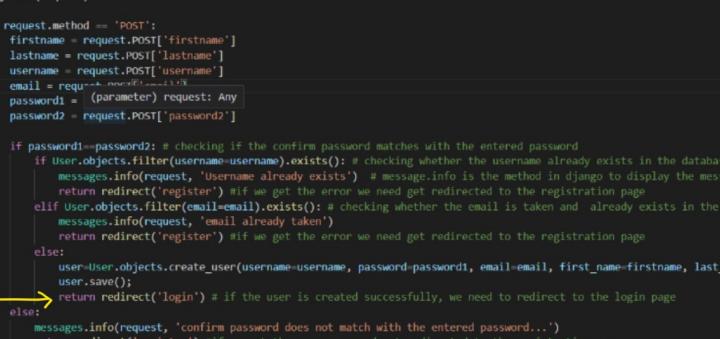


User logged-in



→ Registering a new user & after registering, page is redirecting to login page

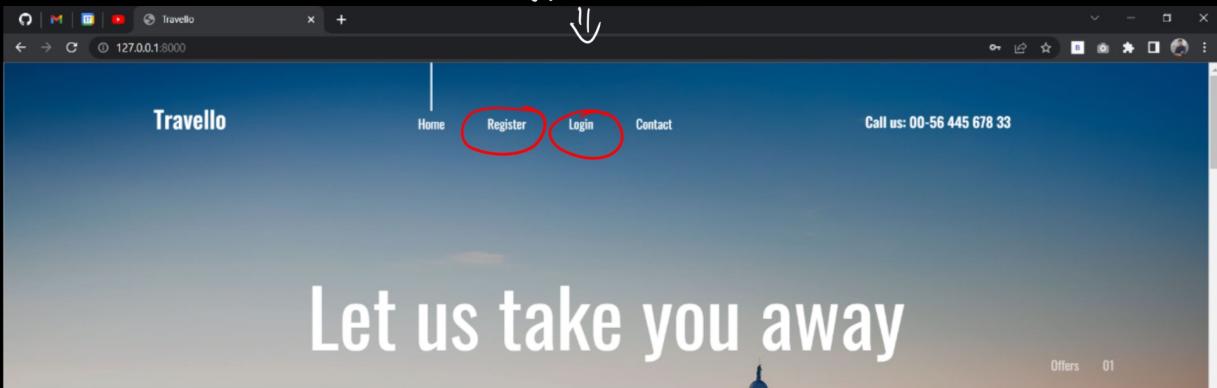




```
register.html login.html views.py index.html urls.py accounts urls.py travello
accounts > views.py > register
24
25     def register(request):
26
27         if request.method == "POST":
28             firstname = request.POST['firstname']
29             lastname = request.POST['lastname']
30             username = request.POST['username']
31             email = request.POST['email']
32             password1 = (request.POST['password1'])
33             password2 = request.POST['password2']
34
35             if password1==password2: # checking if the confirm password matches with the entered password
36                 if User.objects.filter(username=username).exists(): # checking whether the username already exists in the database
37                     messages.info(request, 'User name already exists') # message.info is the method in django to display the message
38                     return redirect('register') #if we get the error we need get redirected to the registration page
39                 elif User.objects.filter(email=email).exists(): # checking whether the email is taken and already exists in the database
40                     messages.info(request, 'Email already taken')
41                     return redirect('register') #if we get the error we need get redirected to the registration page
42                 else:
43                     user=User.objects.create_user(username=username, password=password1, email=email, first_name=firstname, last_name=lastname)
44                     user.save();
45                     return redirect('login') # if the user is created successfully, we need to redirect to the login page
46             else:
47                 messages.info(request, 'confirm password does not match with the entered password...')
48                 return redirect('register') #if we get the error we need get redirected to the registration page
49             return redirect('/')
50         else:
51             return render(request,'register.html')
```

User logout

User logged-in



→ After login, we still see the Register, login button and we don't have the logout button. ⇒ we need to fix these.

is-authenticated
is a django method
it returns true if
the authentication
(or) login is successful.

is-authenticated
it returns true if
the authentication
(or) login is successful.

if user.is_authenticated:
Hello, {{user.first_name}}
accounts/logout
accounts/register
accounts/login
endif

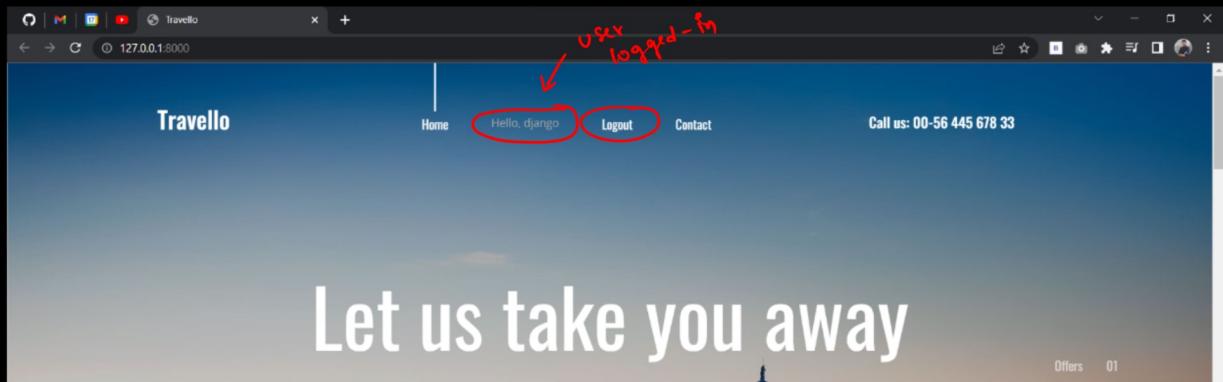
else:
accounts/logout
accounts/register
accounts/login

endif

if the user didn't login, we should see Register & login button.

register.html login.html views.py index.html urls.py accounts urls.py travello

```
<div class="header_content d-flex flex-row align-items-center justify-content-start">
    <div class="header_content_inner d-flex flex-row align-items-end justify-content-start">
        <div class="logo"><a href="index.html">Travello</a></div>
        <nav class="main_nav">
            <ul class="d-flex flex-row align-items-start justify-content-start">
                <li class="active"><a href="index.html">Home</a></li>
                <% if user.is_authenticated %>
                    <li>Hello, {{user.first_name}}</li>
                    <li><a href="accounts/logout">Logout</a></li>
                <% else %>
                    <li><a href="accounts/register">Register</a></li>
                    <li><a href="accounts/login">Login</a></li>
                <% endif %>
                <li><a href="contact.html">Contact</a></li>
            </ul>
        </nav>
    <div class="header_phone ml-auto">Call us: 00-56 445 678 33</div>
</div>
```



→ for logout:

accounts > urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path("register", views.register, name="register"),
    path("login", views.login, name="login"),
    path("logout", views.logout, name="logout"),
]
```

accounts > views.py

```
def logout(request):
    auth.logout(request)
    return redirect('/')
```

register.html login.html index.html views.py urls.py accounts urls.py travello

register.html login.html index.html views.py urls.py accounts urls.py travello

accounts > views.py > @logout

```
def logout(request):
    if request.POST.get('password') != request.session['password']:
        return redirect('register') # if we get the error we need get redirected to the registration page
    else:
        auth.logout(request)
        return redirect('/') # we need to redirect to the home page if the logout is successful.
```

register.html login.html index.html views.py urls.py accounts urls.py travello

The screenshot shows the Travello homepage with a blue header bar. On the left is the 'Travello' logo. In the center, there are navigation links: 'Home', 'Hello, django' (which is underlined in red), 'Logout' (which is also underlined in red), and 'Contact'. On the right, it says 'Call us: 00-56 445 678 33' and 'Offers 01'. Below the header, a large white banner with the text 'Let us take you away' is centered against a background of a city skyline at sunset.

The logged-in user details in database:

→

	Data Output	Explain	Messages	Notifications							
	id	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active	date_joined
1	1	pbkdf2_sha256\$320...	2022-04-17 10:50:38.406484+05:30	true	krish			krish@yopmail.com	true	true	2022-04-16 16:40:02.566712+05:30
2	9	pbkdf2_sha256\$320...	[null]	false	learnercode	learner	coder	log@yopmail.com	false	true	2022-04-17 07:04:25.347306+05:30
3	10	pbkdf2_sha256\$320...	[null]	false	un	fn	ln	email@yopmail.com	false	true	2022-04-17 07:08:29.771166+05:30
4	11	pbkdf2_sha256\$320...	2022-04-17 10:51:06.292101+05:30	false	dj	django	user	dj@yopmail.com	false	true	2022-04-17 07:12:08.774945+05:30
5	12	pbkdf2_sha256\$320...	[null]	false	msd	dhoni	ms	msd@yopmail.com	false	true	2022-04-17 10:12:07.16829+05:30

After logout

