Django (Telusko)

Project - Travello

*(we got the 'Travello' related templates/files from Telusko - Django Tutorial (01:04:02) & saved them in travello-site-files)

① → created a project

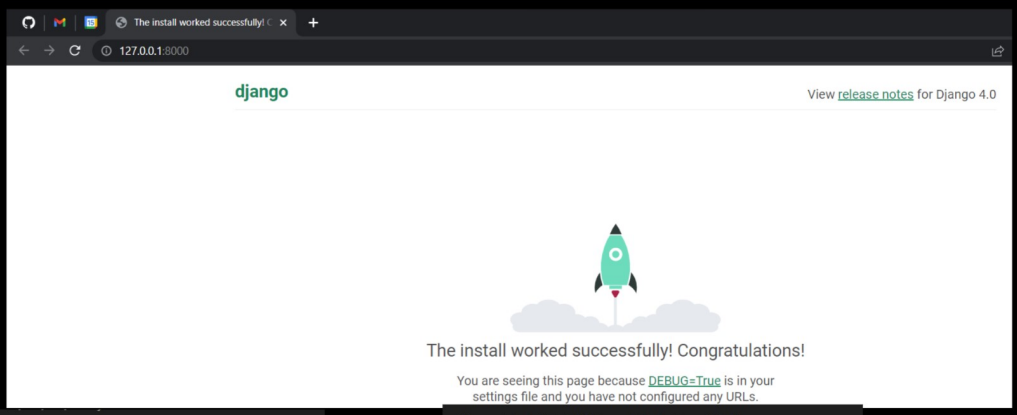
```
(django_telusko) D:\Edu\Django_telusko>django-admin startproject travello

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

```
(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    <DIR>          travello
               1 File(s)            686 bytes
               3 Dir(s)  944,756,375,552 bytes free

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

**Note: Refer to Django_inNeuron_Telusko.pdf befor starting this file.

→ run server

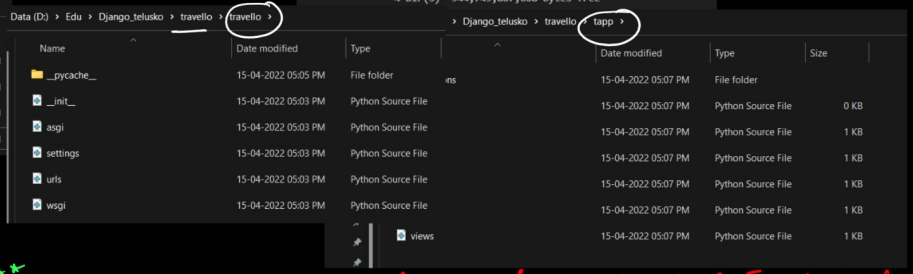
```
(django_telusko) D:\Edu\Django_telusko\travello>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

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 - 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.
```



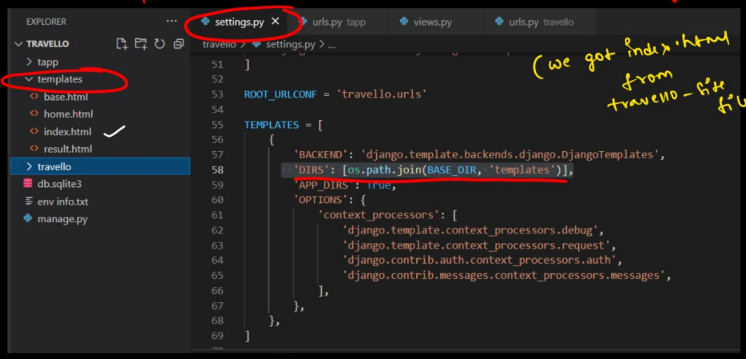

```
(django_telusko) D:\Edu\Django_telusko\travello>python manage.py startapp tapp

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

② → created app

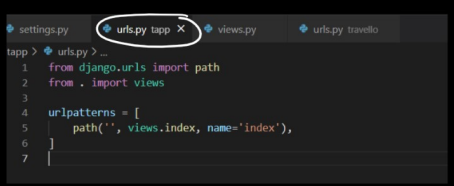
```
(django_telusko) D:\Edu\Django_telusko\travello>dir
 Volume in drive D is Data
 Volume Serial Number is B49E-AE1F

 Directory of D:\Edu\Django_telusko\travello

15-04-2022  05:07 PM    <DIR>          .
15-04-2022  05:03 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)  944,749,867,008 bytes free
```

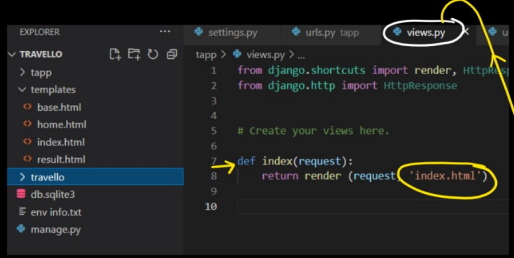


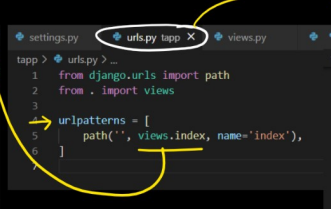
③ *** → created 'templates' folder & added its path in the 'DIRS' of Templates of settings.py

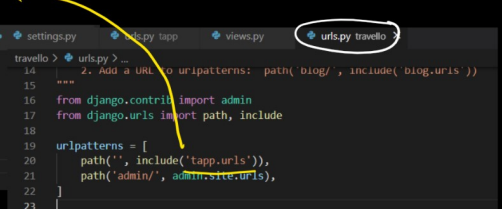


④ → Create urls.py (in the app)

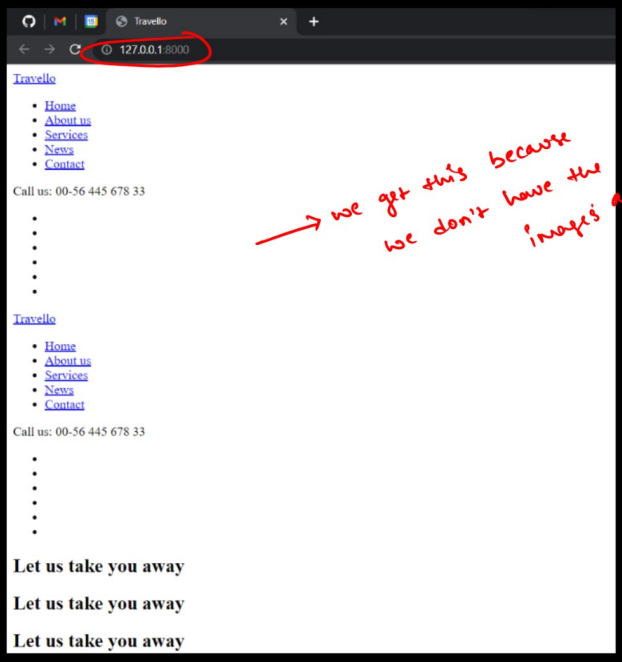








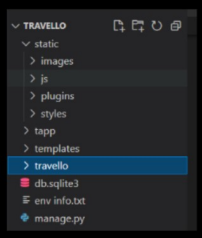
Passing static files:

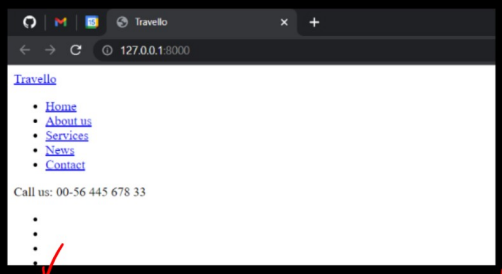


→ we get this because we don't have the images and other related files of travello_site

(5)

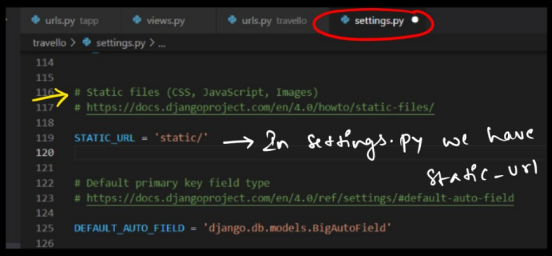
→ Create a folder 'static' for all the static files (Copy the static folder from travello_site_files)





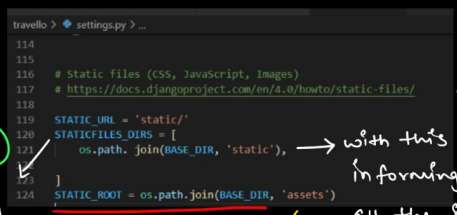
Still we get the same page without any layout/images.

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



→ In settings.py we have static_url field.

(6)



Creating static files directory

→ with this we are informing Django that all the static files will be present in the 'Base_DIR/static' directory.

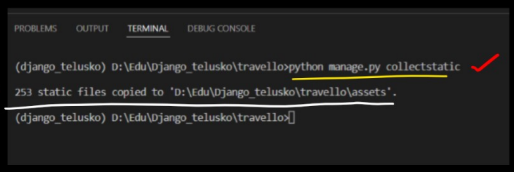
we can give any name

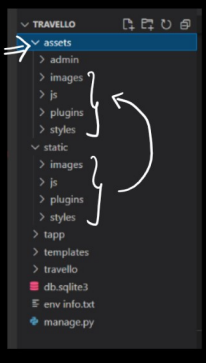
What happens here is, to improve the process, Django will collect all the files in our 'static' folder & keeps in an other Django folder ('assets')

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

(7)

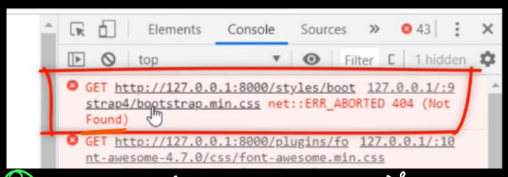
⇒ python manage.py collectstatic





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.

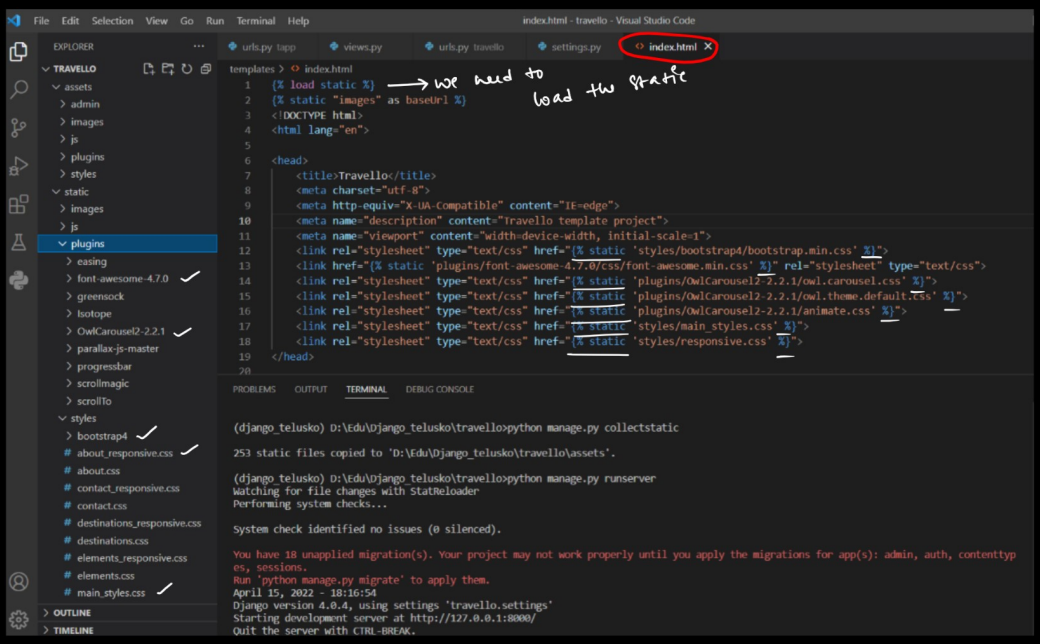


⇒ This is because in the index.html we have all the links to static pages but these won't be understandable by Django

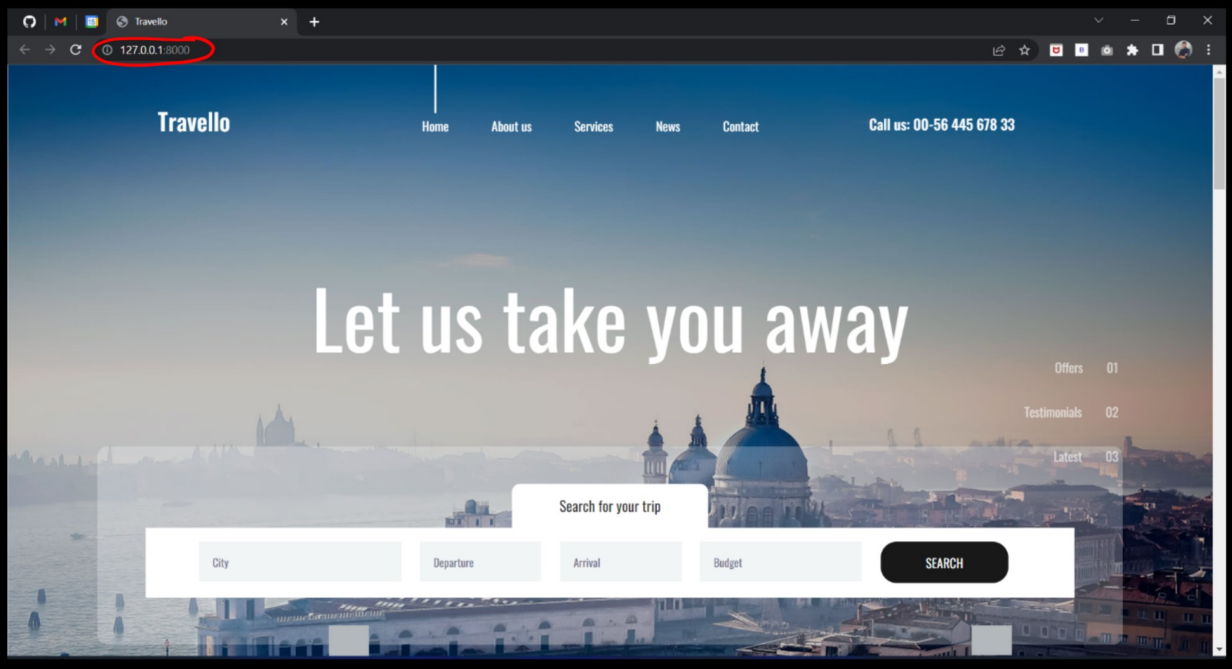
(8) → To make it understand to Django, we need to enclose the urls in {% static ' ' %}. Which says that all the files of these urls are coming from 'static' folder.

```
et="utf-8">
equiv="X-UA-Compatible" content="IE=edge">
"description" content="Travello template project">
"viewport" content="width=device-width, initial-scale=1">
stylesheet" type="text/css" href="styles/bootstrap4/bootstrap.min.css">
"plugins/font-awesome-4.7.0/css/font-awesome.min.css" rel="stylesheet" type="text/css">
stylesheet" type="text/css" href="plugins/OwlCarousel2-2.2.1/owl.carousel.css">
stylesheet" type="text/css" href="plugins/OwlCarousel2-2.2.1/owl.theme.default.css">
stylesheet" type="text/css" href="plugins/OwlCarousel2-2.2.1/animate.css">
stylesheet" type="text/css" href="styles/main_styles.css">
stylesheet" type="text/css" href="styles/responsive.css">
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Travello</title>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="description" content="Travello template project">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" type="text/css" href="{% static 'styles/bootstrap4/bootstrap.min.css' %}">
    <link href="{% static 'plugins/font-awesome-4.7.0/css/font-awesome.min.css' %}" rel="stylesheet" type="text/css">
    <link rel="stylesheet" type="text/css" href="{% static 'plugins/OwlCarousel2-2.2.1/owl.carousel.css' %}">
    <link rel="stylesheet" type="text/css" href="{% static 'plugins/OwlCarousel2-2.2.1/owl.theme.default.css' %}">
    <link rel="stylesheet" type="text/css" href="{% static 'plugins/OwlCarousel2-2.2.1/animate.css' %}">
    <link rel="stylesheet" type="text/css" href="{% static 'styles/main_styles.css' %}">
    <link rel="stylesheet" type="text/css" href="{% static 'styles/responsive.css' %}">
</head>
```

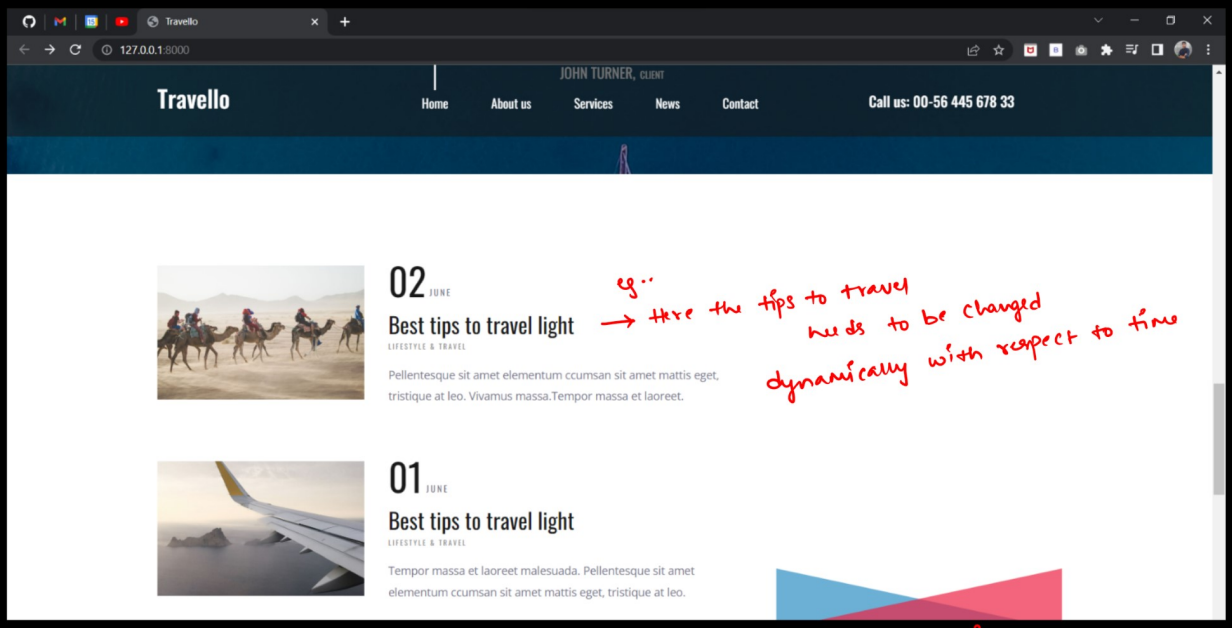


(9)



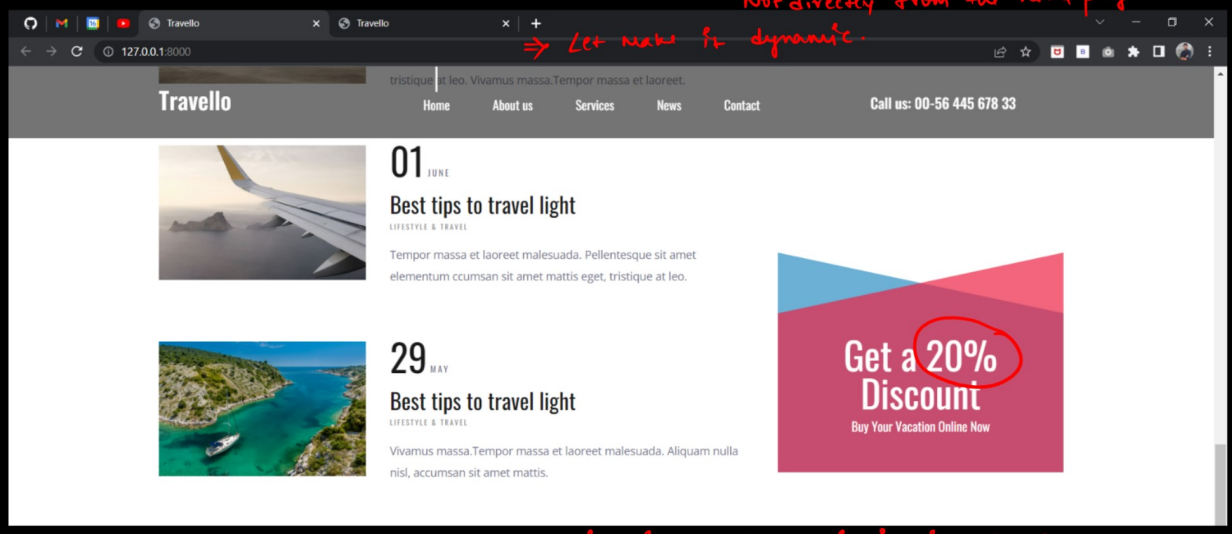
For those who couldn't figure out how to put {% static ' ' %} in the background images!
When you type {% static 'Image-Name' %}, you are essentially just adding the base directory of the 'Static' folder we created before the image location. You can type /static/ before the image location and it will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

passing Dynamic Data in HTML:

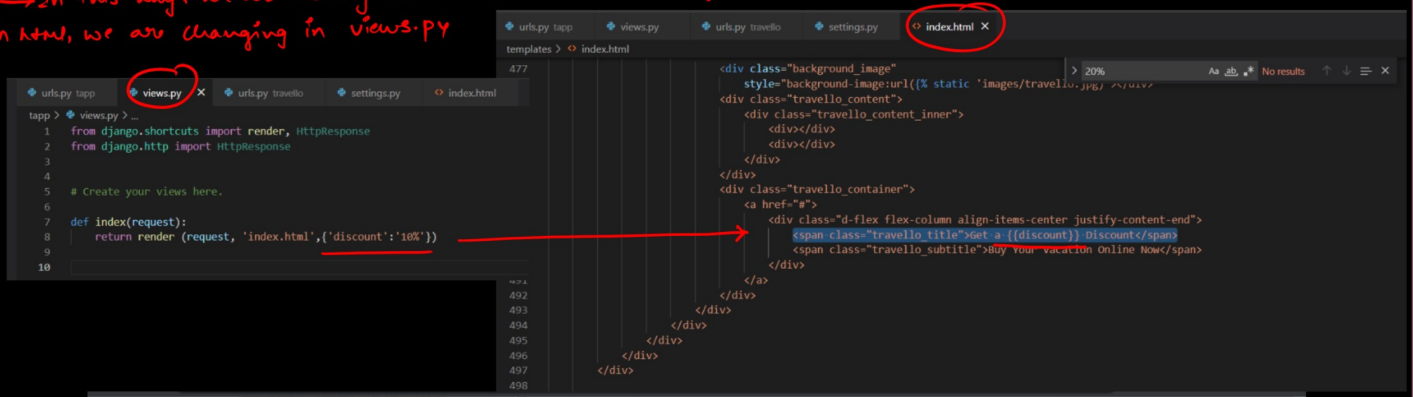


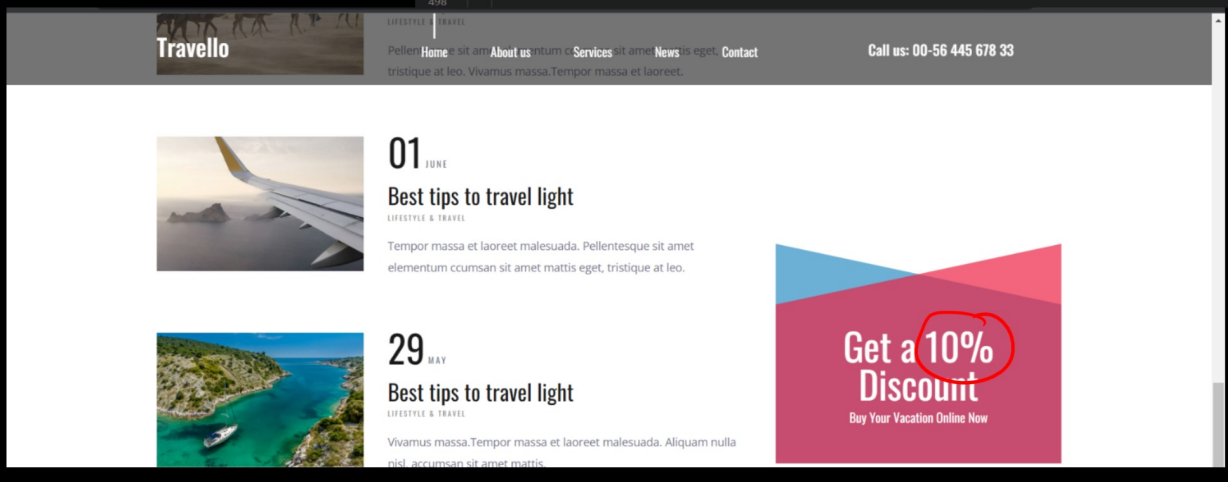
eg..
→ Here the tips to travel needs to be changed dynamically with respect to time

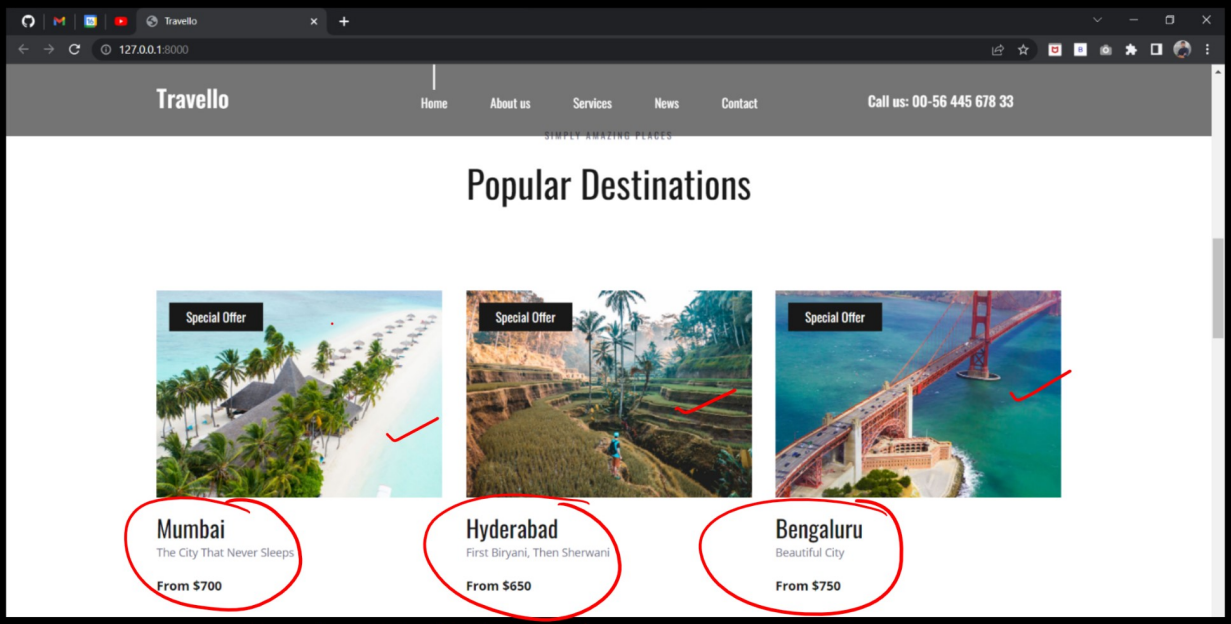
→ Initially the discount in the page is 20%, but I want to change it.
Not directly from the html page.
⇒ Let make it dynamic.



⇒ 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







⇒ 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.

```python
# models.py
from django.db import models

# Create your models here.
#  Models are used to connect with databases and data.

class Destination:
    id : int
    name : str
    img : str
    desc : str
    price : int
```

```python
# views.py
from django.shortcuts import render, HttpResponse
from django.http import HttpResponse
# from . import models
from .models import Destination


# Create your views here.

# def index(request):
#     return render (request, 'index.html',{'discount':'10%'})

def index(request):

    dest1 = Destination()
    dest1.name = 'Mumbai'
    dest1.desc = 'The City That Never Sleeps'
    dest1.img = 'destination_1.jpg'
    dest1.price = 700

    dest2 = Destination()
    dest2.name = 'Hyderabad'
    dest2.desc = 'First Biryani, Then Sherwani'
    dest2.img = 'destination_2.jpg'
    dest2.price = 650

    dest3 = Destination()
    dest3.name = 'Bengaluru'
    dest3.desc = 'Beautiful City'
    dest3.img = 'destination_3.jpg'
    dest3.price = 750

    dests = [dest1, dest2, dest3]

    return render(request, "index.html", {'dests': dests})
```

Objects → dest1, dest2, dest3 & 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

```
{% load static %}
{% static "images" as baseUrl %}
<!DOCTYPE html>
<html lang="en">
```

→ we are defining a variable 'baseUrl' for 'images' present in 'static' folder

```
<!-- Destinations -->

<div class="destinations" id="destinations">
    <div class="container">
        <div class="row">
            <div class="col text-center">
                <div class="section_subtitle">simply amazing places</div>
                <div class="section_title">
                    <h2>Popular Destinations</h2>
                </div>
            </div>
        </div>
        <div class="row destinations_row">
            <div class="col">
                <div class="destinations_container item_grid">

                    {% for dest in dests %}
                    <!-- Destination -->
                    <div class="destination item">
                        <div class="destination_image">
                            <img src="{{baseUrl}}/{{dest.img}}" alt="">
                            <div class="spec_offer text-center"><a href="#">Special Offer</a></div>
                        </div>
                        <div class="destination_content">
                            <div class="destination_title"><a href="destinations.html">{{dest.name}}</a>
                            </div>
                            <div class="destination_subtitle">
                                <p>{{dest.desc}}</p>
                            </div>
                            <div class="destination_price">From ${{dest.price}}</div>
                        </div>
                    </div>

                    {% endfor %}

                </div>
            </div>
        </div>
```

Creating a for loop & passing the values as dynamic content

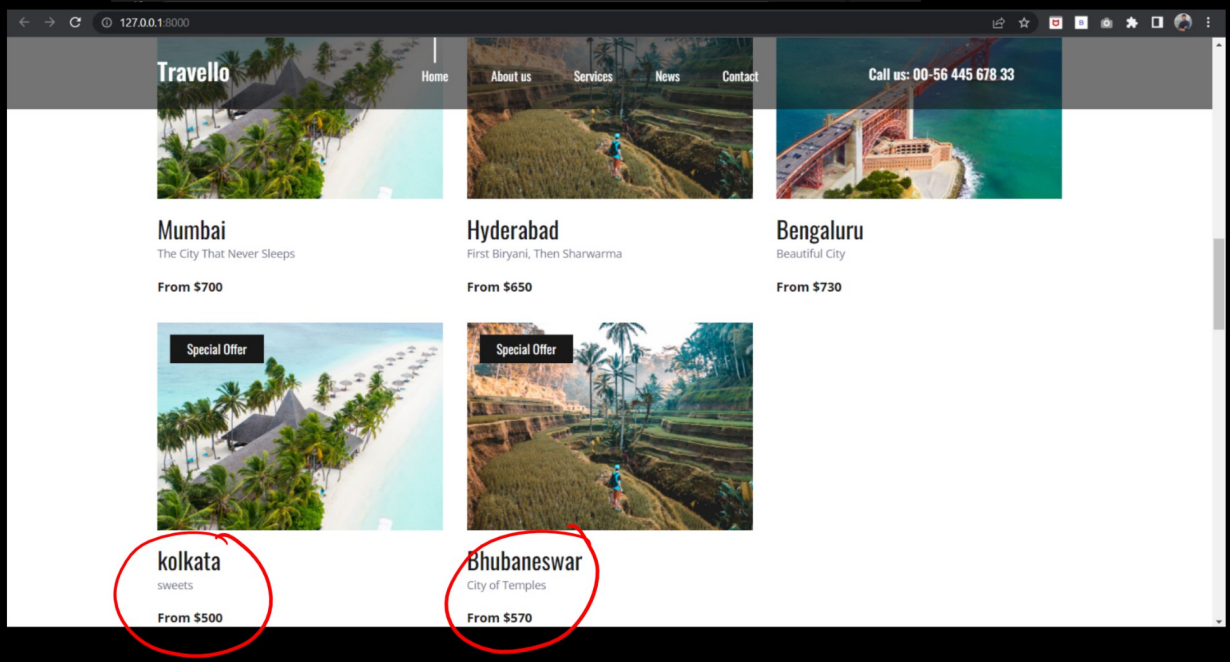
we get images from the 'images' folder of 'static' folder defined as 'baseUrl'

→ 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.

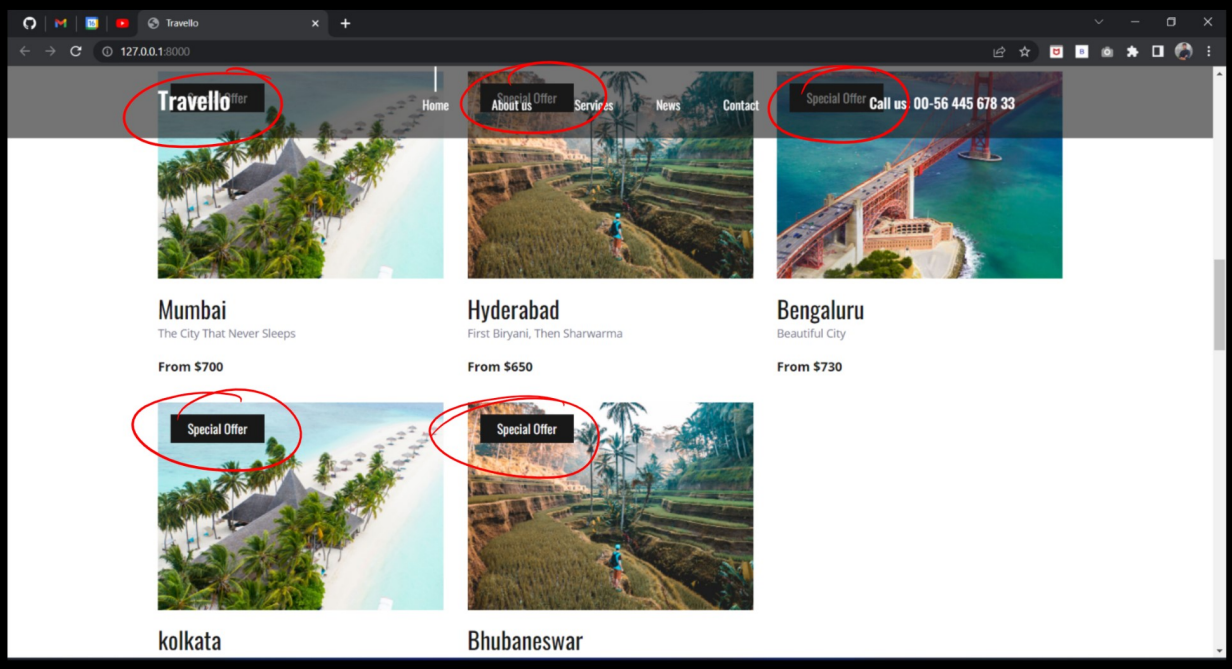
```python
def index(request):

    dest1 = Destination()
    dest1.name = 'Mumbai'
    dest1.desc = 'The City That Never Sleeps'
    dest1.img = 'destination_1.jpg' #we need to mention the name of image which is in the images folder od stati
    dest1.price = 700

    dest2 = Destination()
    dest2.name = 'Hyderabad'
    dest2.desc = 'First Biryani, Then Sharwarma'
    dest2.img = 'destination_2.jpg'
    dest2.price = 650

    dest3 = Destination()
    dest3.name = 'Bengaluru'
    dest3.desc = 'Beautiful City'
    dest3.img = 'destination_3.jpg'
    dest3.price = 730

    dest4 = Destination()
    dest4.name = 'kolkata'
    dest4.desc = 'sweets'
    dest4.img = 'destination_1.jpg'
    dest4.price = 500

    dest5 = Destination()
    dest5.name = 'Bhubaneswar'
    dest5.desc = 'City of Temples'
    dest5.img = 'destination_2.jpg'
    dest5.price = 570


    dests = [dest1, dest2, dest3,dest4,dest5]

    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 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.

```python
# models.py
from django.db import models

# Create your models here.
#  Models are used to connect with databases and data.

class Destination:
    id : int
    name : str
    img : str
    desc : str
    price : int
    offer : bool
```

```python
# views.py
def index(request):
    dest1 = Destination()
    dest1.name = 'Mumbai'
    dest1.desc = 'The City That Never Sleeps'
    dest1.img = 'destination_1.jpg' #we need to mention the name of image which is in the images folder od stati
    dest1.price = 700
    dest1.offer = False

    dest2 = Destination()
    dest2.name = 'Hyderabad'
    dest2.desc = 'First Biryani, Then Sharwarma'
    dest2.img = 'destination_2.jpg'
    dest2.price = 450
    dest2.offer = True

    dest3 = Destination()
    dest3.name = 'Bengaluru'
    dest3.desc = 'Beautiful City'
    dest3.img = 'destination_3.jpg'
    dest3.price = 730
    dest3.offer = False

    dest4 = Destination()
    dest4.name = 'kolkata'
    dest4.desc = 'sweets'
    dest4.img = 'destination_1.jpg'
    dest4.price = 500
    dest4.offer = False

    dest5 = Destination()
    dest5.name = 'Bhubaneswar'
    dest5.desc = 'City of Temples'
    dest5.img = 'destination_2.jpg'
    dest5.price = 570
    dest5.offer = False
    dests = [dest1, dest2, dest3,dest4,dest5]
    return render(request, 'index.html', {'dests': dests,'discount':'10%'})
```

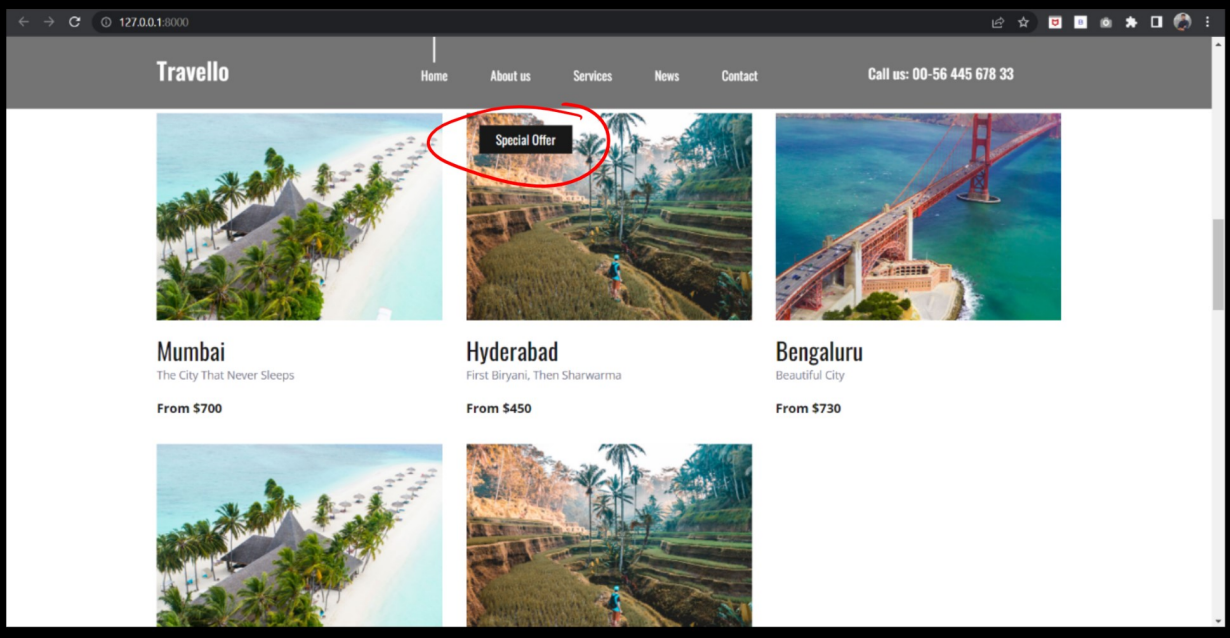
⇒ 'offer' should be applicable only for Hyderabad.

```html
<!-- Destinations -->

<div class="destinations" id="destinations">
    <div class="container">
        <div class="row">
            <div class="col text-center">
                <div class="section_subtitle">simply amazing places</div>
                <div class="section_title">
                    <h2>Popular Destinations</h2>
                </div>
            </div>
        </div>
        <div class="row destinations_row">
            <div class="col">
                <div class="destinations_container item_grid">

                    {% for dest in dests %}
                    <!-- Destination -->
                    <div class="destination item">
                        <div class="destination_image">
                            <img src="{{baseUrl}}/{{dest.img}}" alt="">
                            {% if dest.offer %}
                            <div class="spec_offer text-center"><a href="#">Special Offer</a></div>
                            {% endif %}
                        </div>
                        <div class="destination_content">
                            <div class="destination_title"><a href="destinations.html">{{dest.name}}</a></div>
                            <div class="destination_subtitle">
                                <p>{{dest.desc}}</p>
                            </div>
                            <div class="destination_price">From ${{dest.price}}</div>
                        </div>
                    </div>

                    {% endfor %}
```

'If block'

⇒ We kept the offer statement in the 'If block'

⇒ Only when dest.offer = True, the 'special offer' will be displayed.

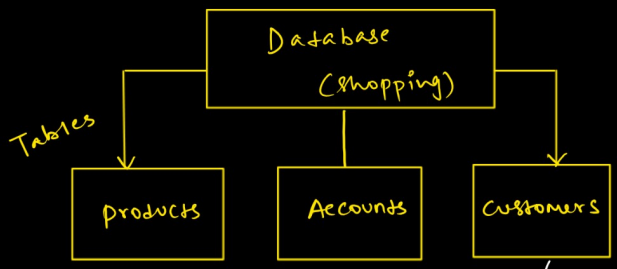


Now we want to fetch the data from the database.

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

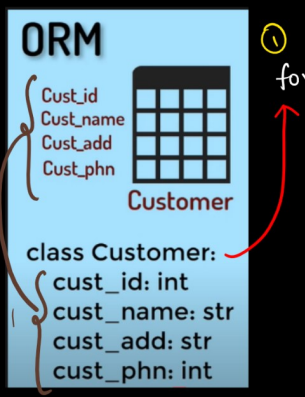
Object Relational Mapper (ORM) Theory

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

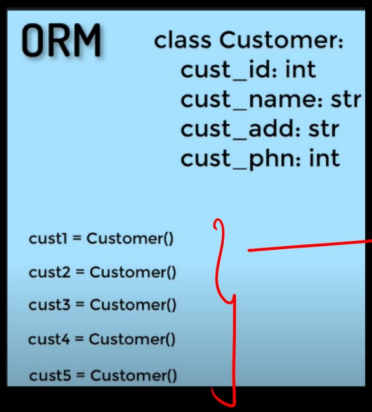


Lets say that this Customer table has 4 columns/fields: Cust_id, Cust_name, Cust_ph, Cust_add.

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

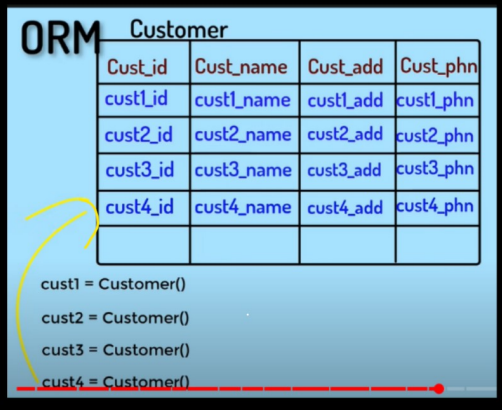


① for the customer table, we create a 'Customer' class. & the the fields/columns will be the class variables/properties.



② * Now we can have multiple customers, so we create one object for each Customer.

⇒ Ever object will have different values (name, id, add, ph)



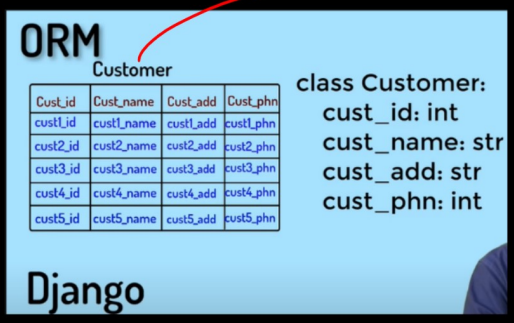
** So, when we map the data/to from the database.

→ 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.

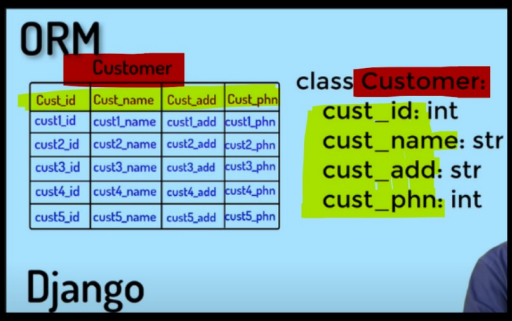
↓

Is it possible? — YES

The framework that we use (Django) has the special power to look at the class and says, hey ! (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

↙ mapping of the objects to the Relational databases.

Class → database / Table

Object → data in table

mapping

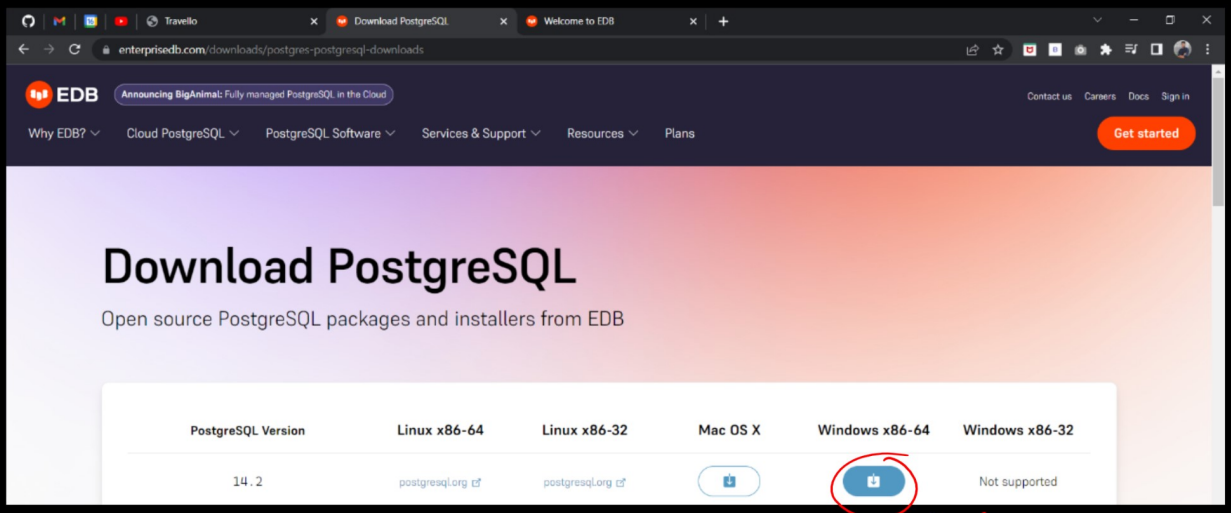
———— ✗ ————

Postgres and PgAdmin Setup:

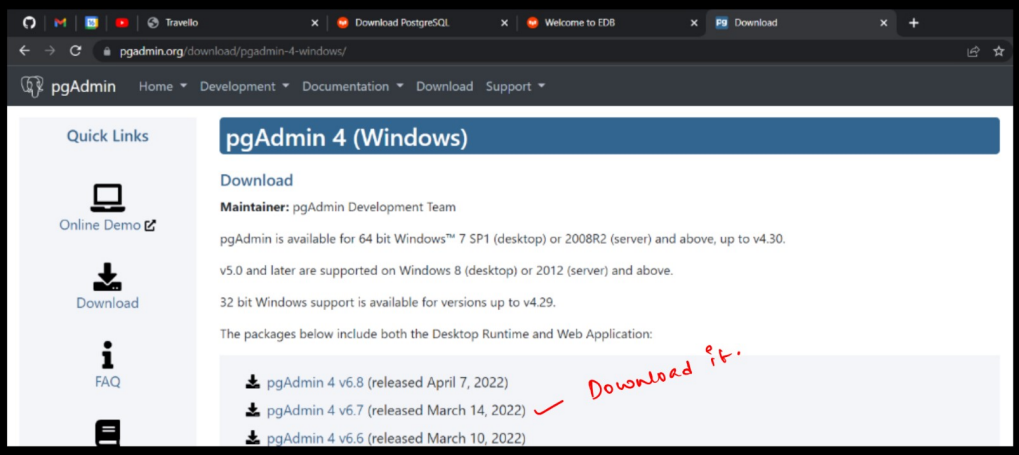
We have many RDMS - MySQL, PostgreSQL etc.. to work with Python.

→ We will use PostgreSQL here.

→ We need to download the PostgreSQL software



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



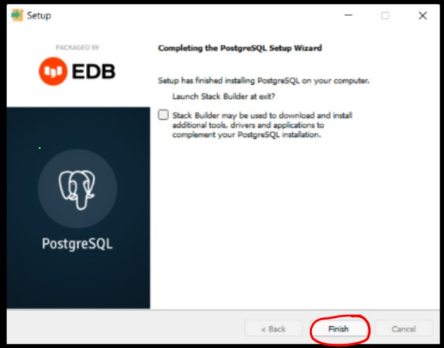
⇒ PostgreSQL Installation:

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

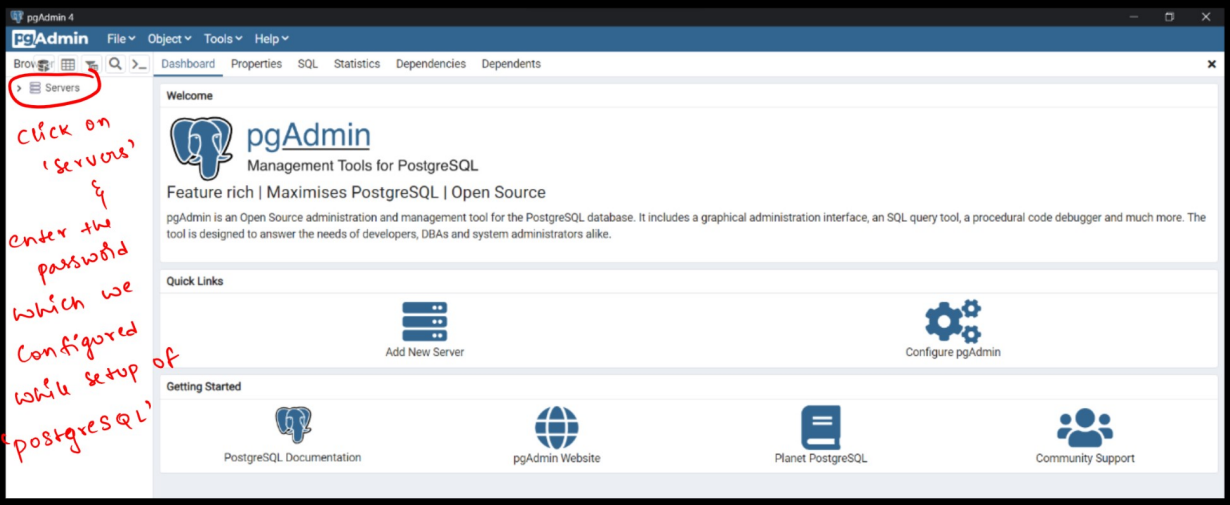
→ port number - 5432 (fixed by default)

→ Now we need to install PgAdmin.

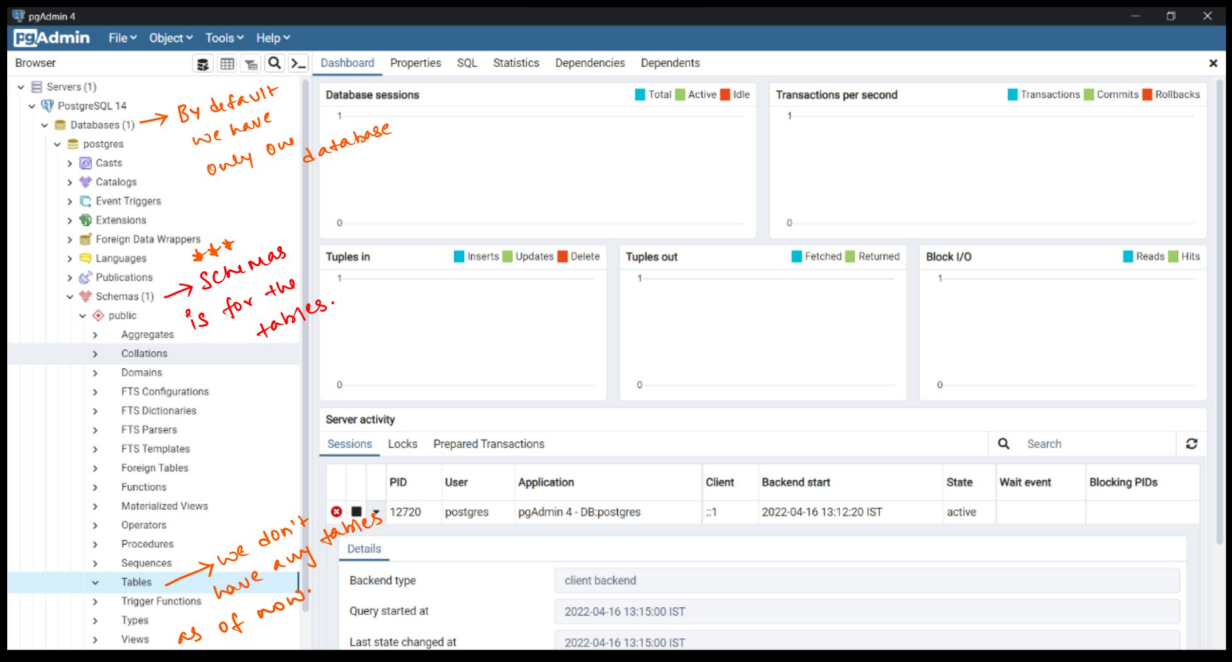
Keep all the default values &
Click next > Install > Finish.



→ Open the Pg Admin in Administrator Mode.



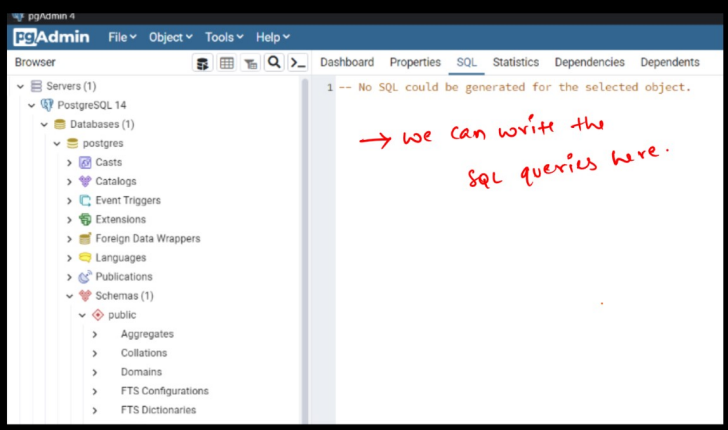
Click on 'Servers' & enter the password which we configured while setup of 'postgreSQL'



→ By default we have only one database

→ Schemas is for the tables.

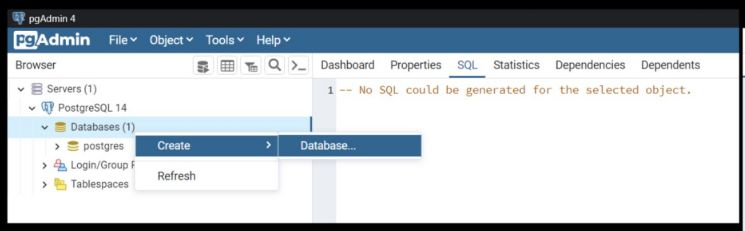
→ we don't have any tables as of now.

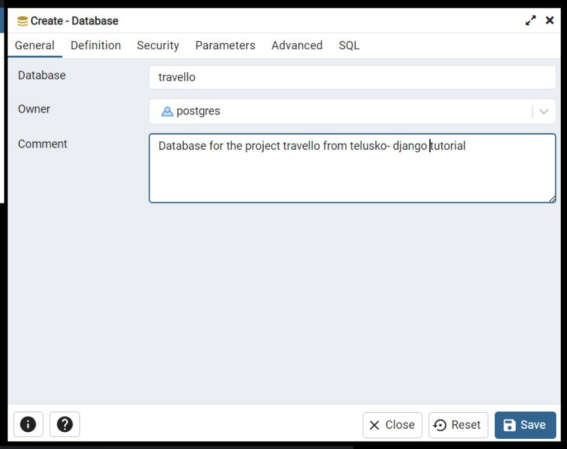


→ we can write the SQL queries here.

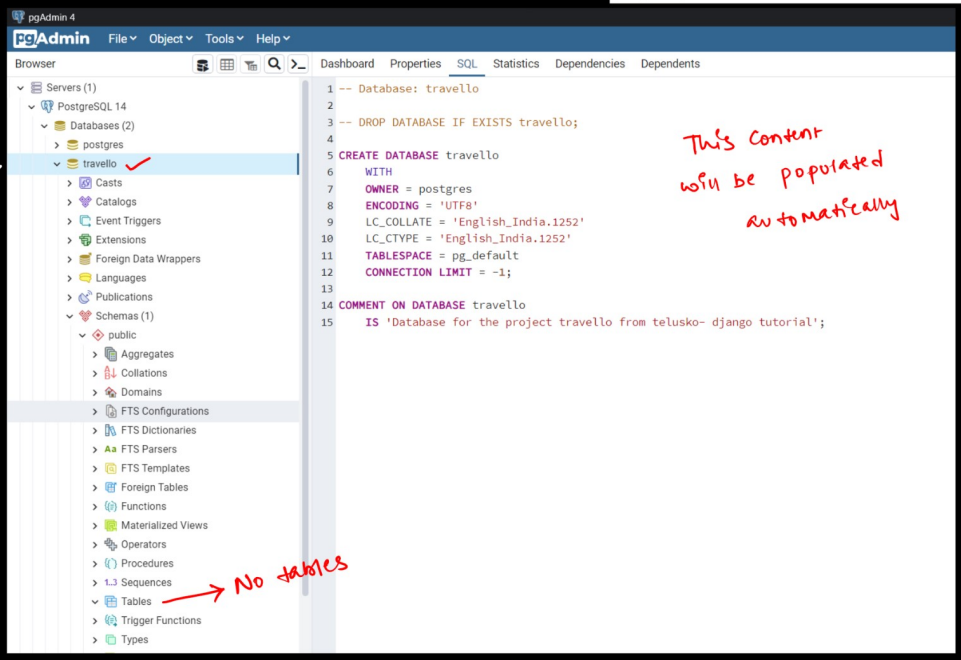
Models & Migrations

① → Create a database





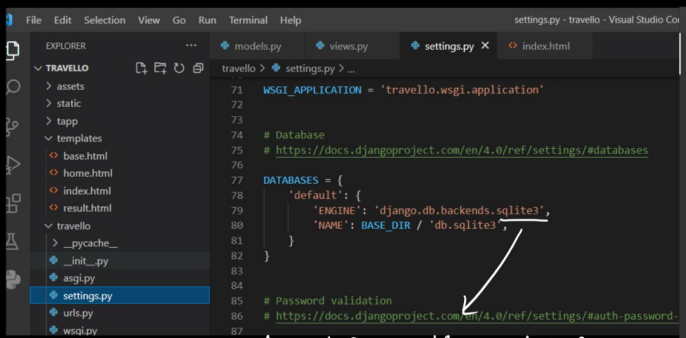
Database created →



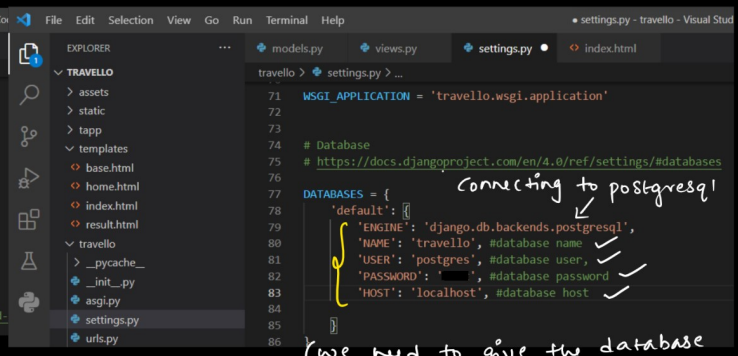
This content will be populated automatically

No tables

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



by default django takes sqlite3 database. we need to change it to postgresql.

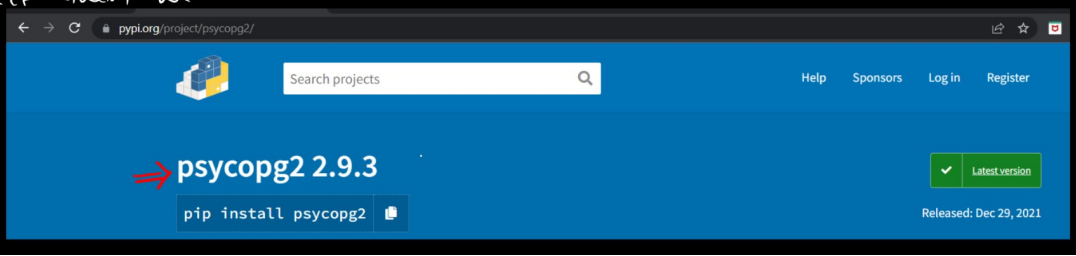


Connecting to postgresql

(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



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.

```python
from django.db import models

# Create your models here.
#  Models are used to connect with databases and data.

class Destination:
    id : int
    name : str
    img : str
    desc : str
    price : int
    offer : bool
```

→ Converting class to model:

(Class name = Destination)

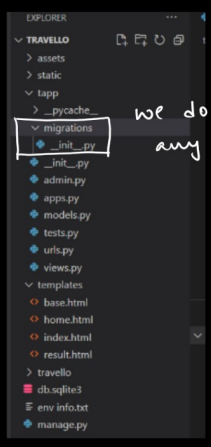
```python
from django.db import models

# Create your models here.
#  Models are used to connect with databases and data.

class Destination(models.Model): #we are inheritng the feature of models.
    # https://docs.djangoproject.com/en/4.0/ref/models/fields/#autofield
    name = models.CharField(max_length=100)
    img = models.ImageField(upload_to='pics') #basically we upload the images to a specific folder amd we need to give that folder name here.
    desc = models.TextField()
    price = models.IntegerField()
    offer = models.BooleanField(default=False)
```

defining the datatypes of the fields/columns

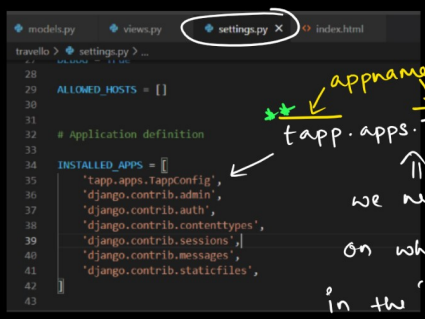
(we get the datatypes info in django model fields documentation)

(5) ***

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



we don't have any migration files except __init__.py



```python
ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'tapp.apps.TappConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

(6)

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

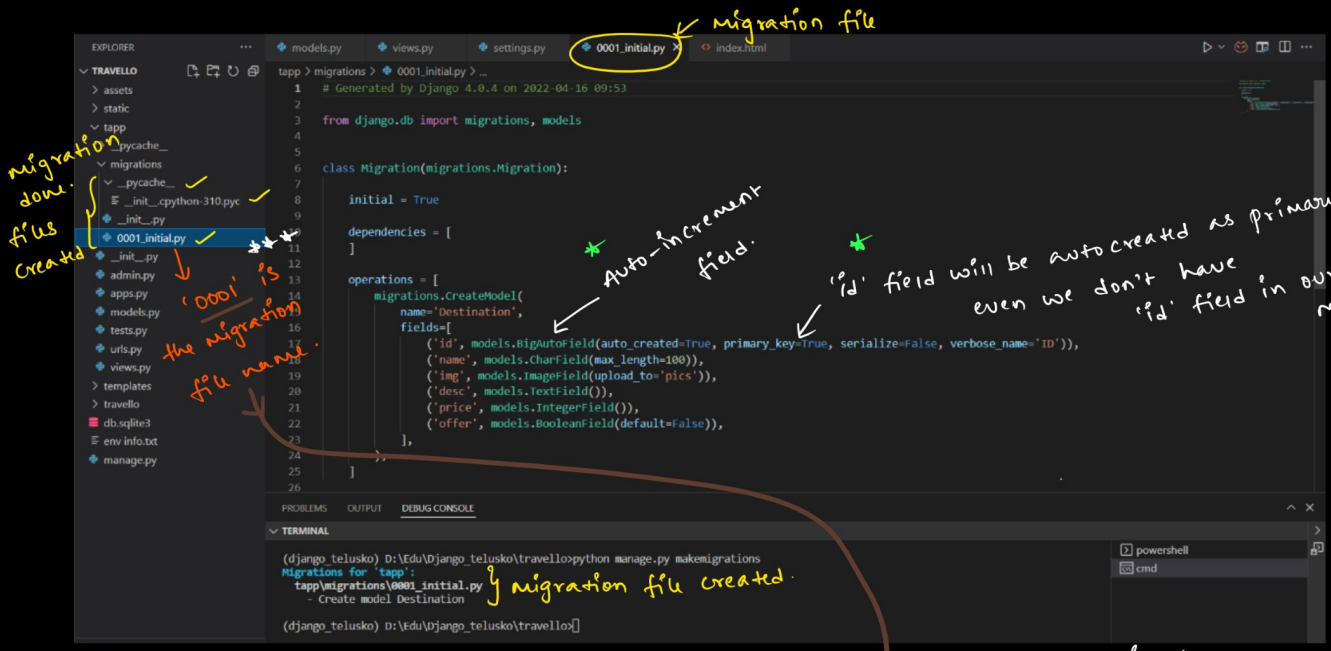
```
(django_telusko) D:\Edu\Django_telusko\travello>python manage.py makemigrations
SystemCheckError: System check identified some issues:

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".

(django_telusko) D:\Edu\Django_telusko\travello>python -m pip install Pillow
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

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

(Error)

To work with images we need the library 'pillow'

(7)

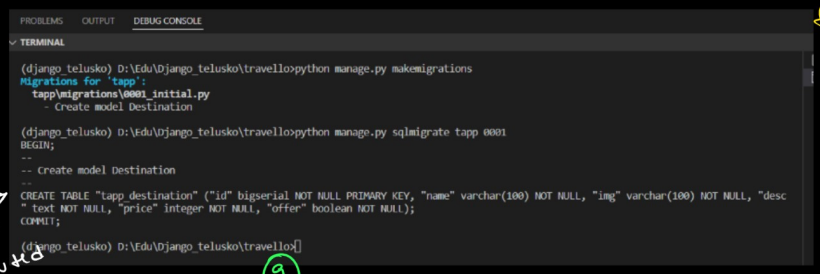
installing pillow



(8) → We had made/created the migration file now we need to migrate it to create the table.

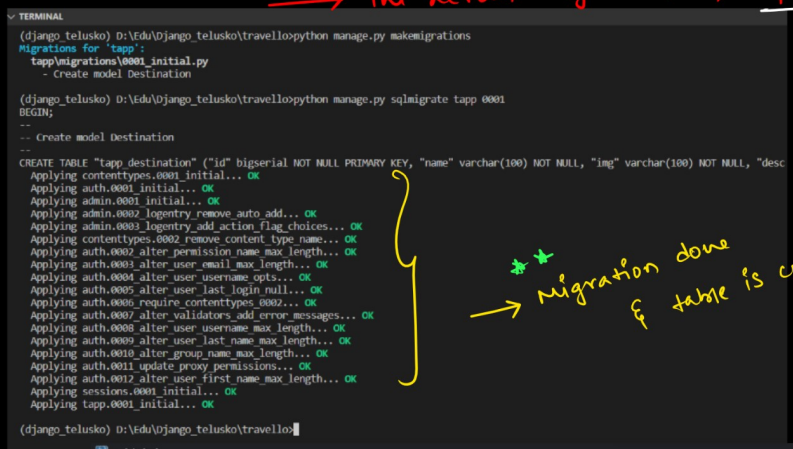
⇒ python manage.py sqlmigrate tapp 0001

app name | migration file name

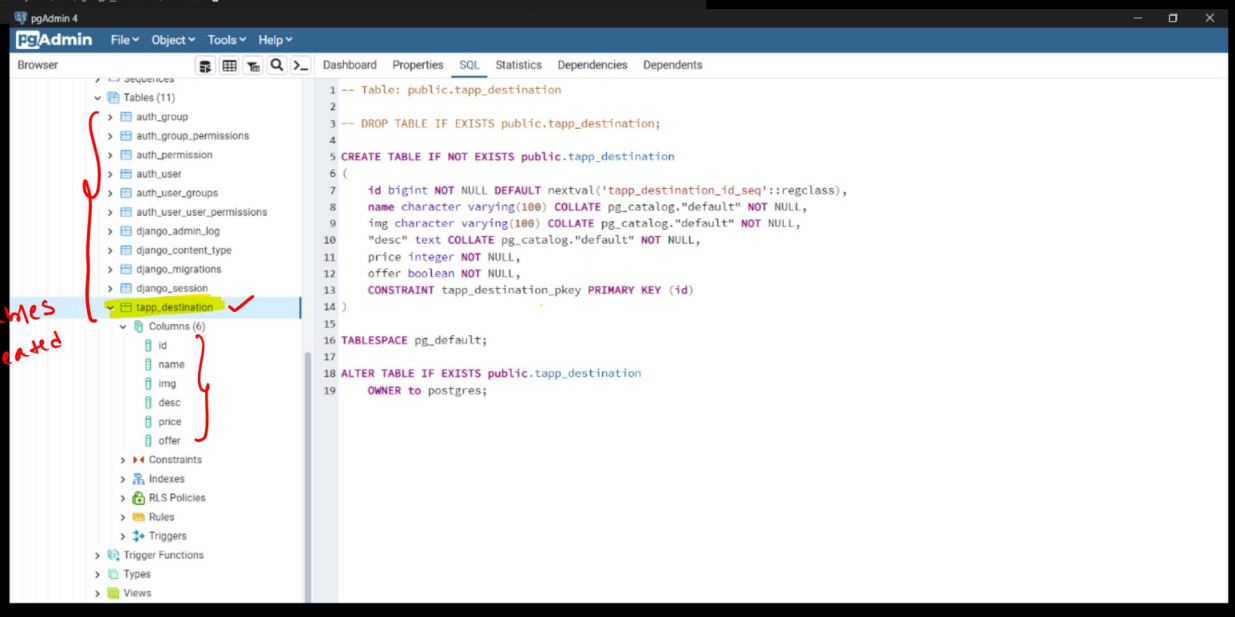


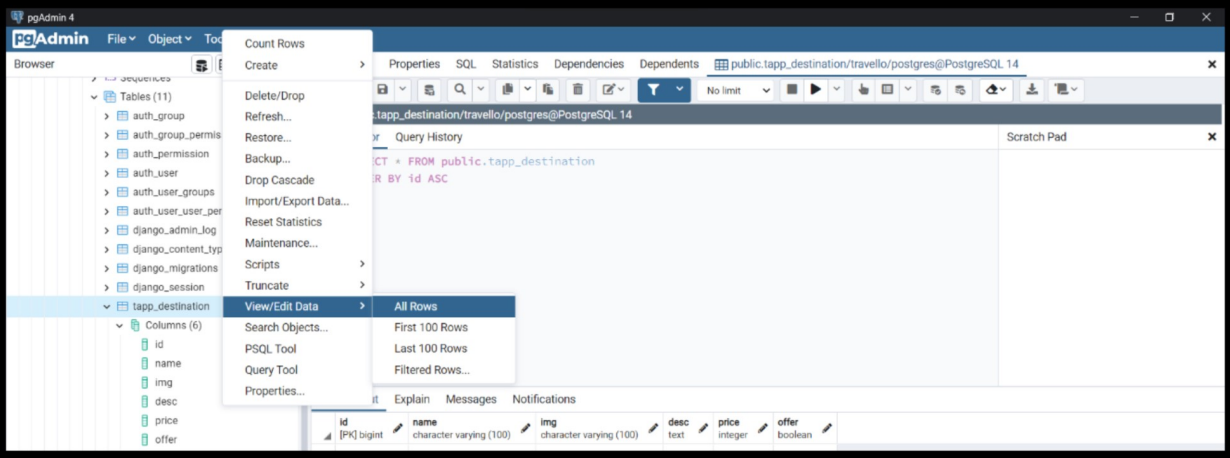
This query will be executed

(9) → The actual migration ⇒ python manage.py migrate



Migration done & table is created.





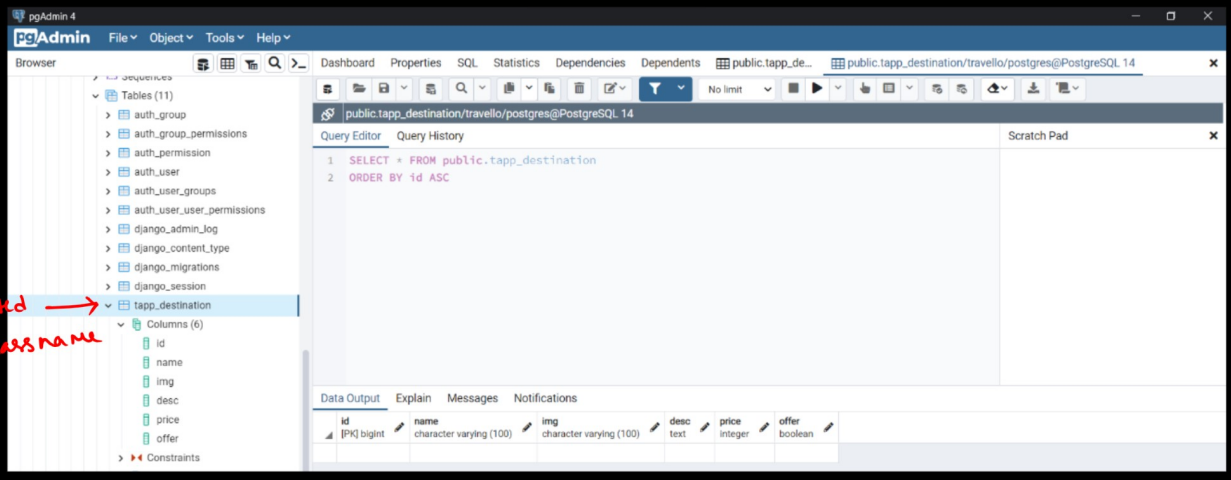


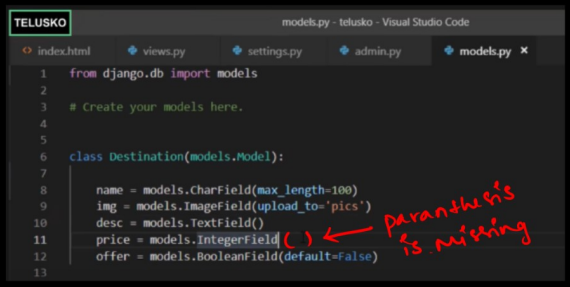
table created → appname_classname (class in models.py)

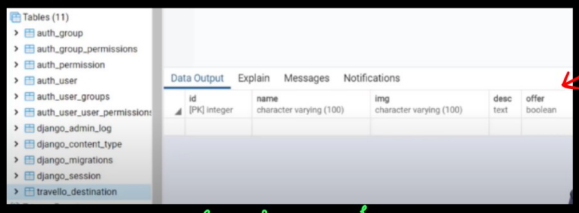
Re-Migration

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

We need to correct them & remigrate again.

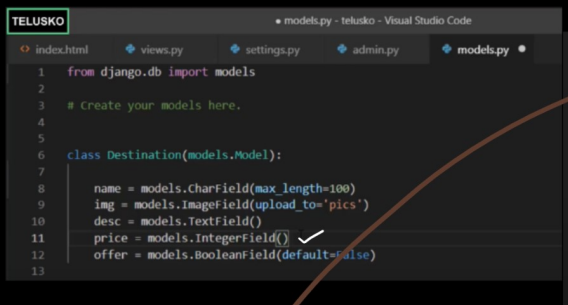
for eg.. In the tutorial, there was a mistake in the price field. So, the 'price' column didn't created in the table.



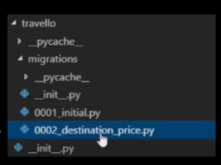


→ Migrating again:

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 val
 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. ti
Type 'exit' to exit this prompt
>>> 0   → default value
Migrations for 'travello':
  travello\migrations\0002_destination_price.py
    - Add field price to destination

(test) C:\Users\Navin\projects\telusko>
```



New migration file →

Migrating to database.

```
(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.
```

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.

1. By creating a page into which the user can push/post the data.

(Or)

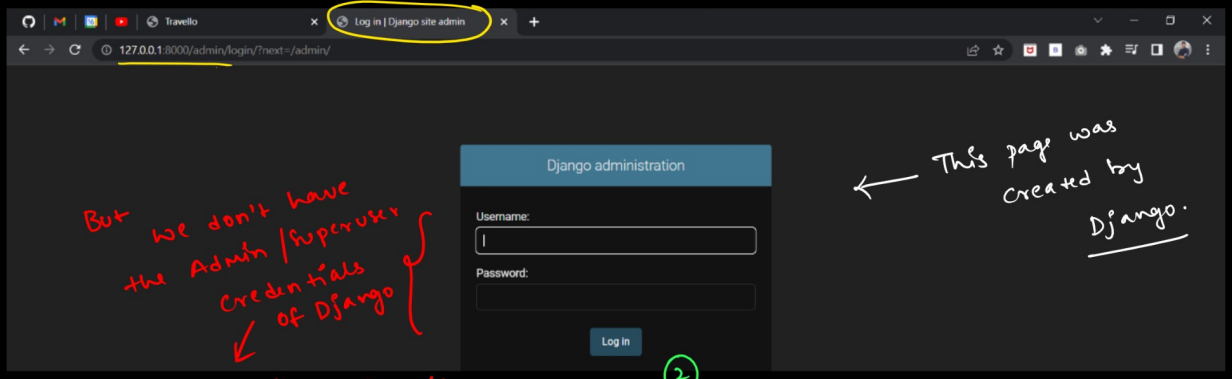
2. We can have the Admin page.
   ↓
   Is accessible by Admins/Superusers.

→ A normal user can push the data by in our project but we have destinations info which is added by the admins.

⇒ (1) Lets go the Admin page:

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

Admin page



But we don't have the Admin/superuser credentials of Django

↓

we need to create

← This page was created by Django.

(2) → Creating Super user:

python manage.py CreateSuperuser

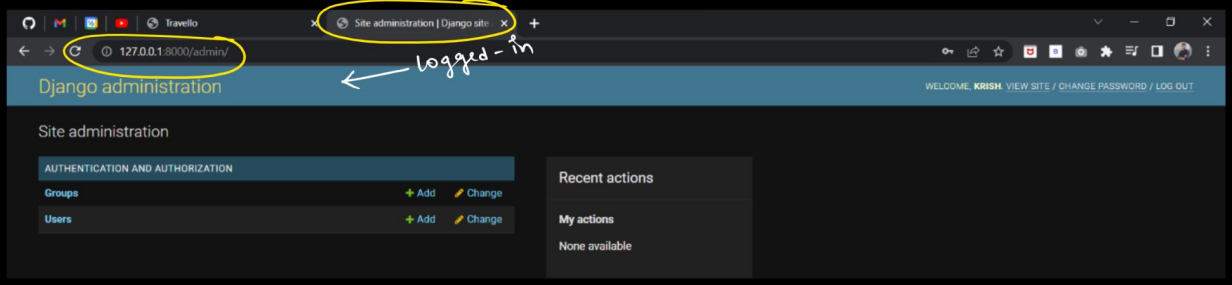
→ Note: python manage.py help gives the subcommands.

```
TERMINAL

(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
    makemessages
    makemigrations
    migrate
    sendtestemail
    shell
    showmigrations
    sqlflush
    sqlmigrate
    sqlsequencereset
    squashmigrations
    startapp
    startproject
    test
    testserver

[sessions]
    clearsessions

[staticfiles]
```

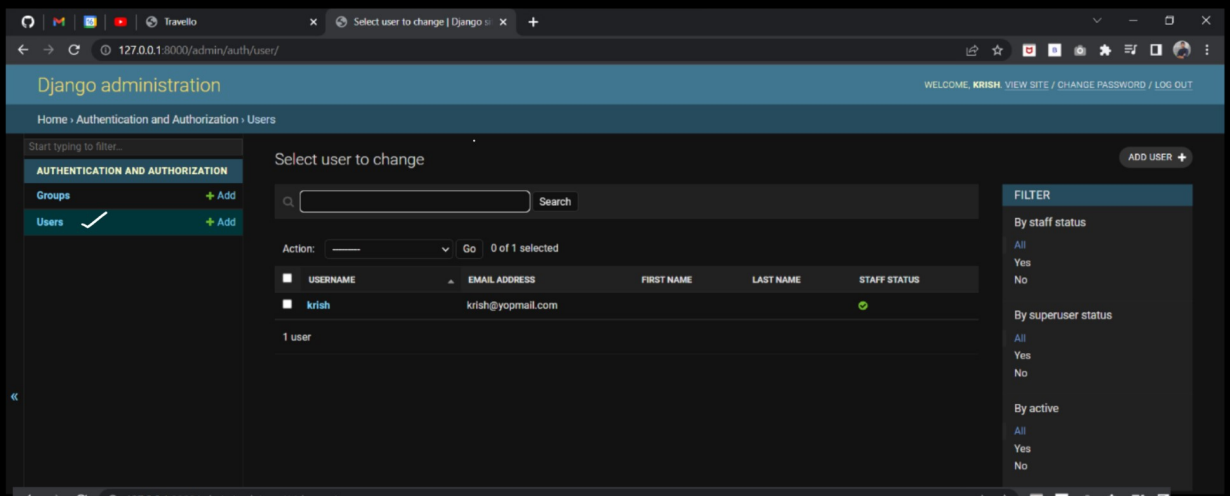
⇒

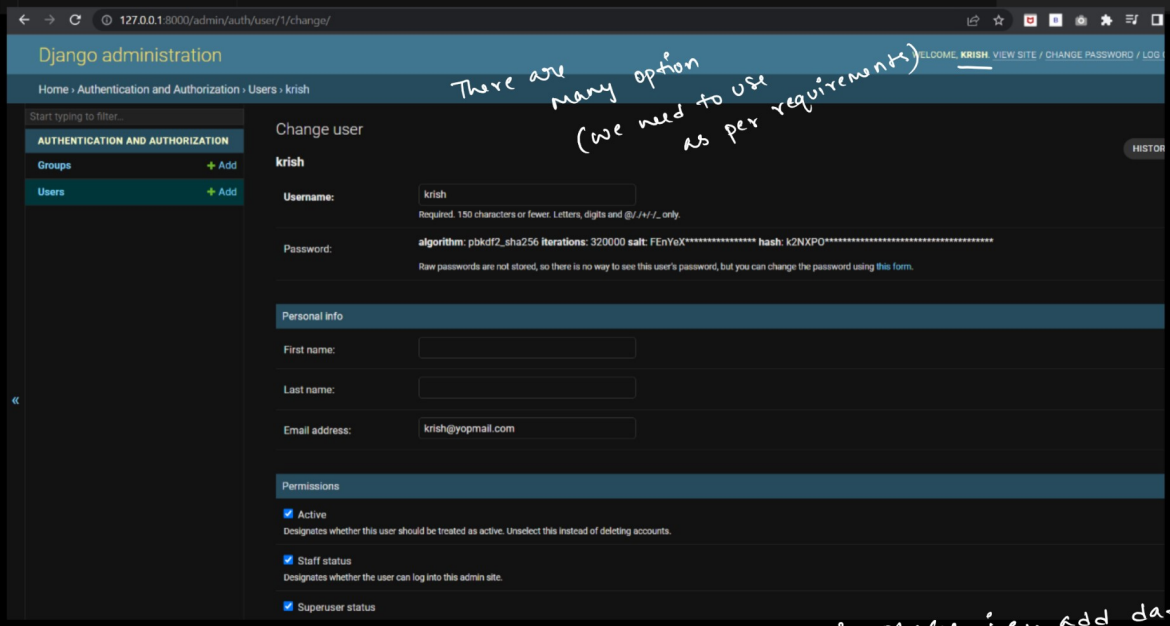
```
(django_telusko) D:\Edu\Django_telusko\travello>python manage.py createsuperuser
Username (leave blank to use 'krishnaaleti'): krish
Email address: krish@yopmail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass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

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

(3) login using the created superuser credentials.



← logged-in





→ 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.

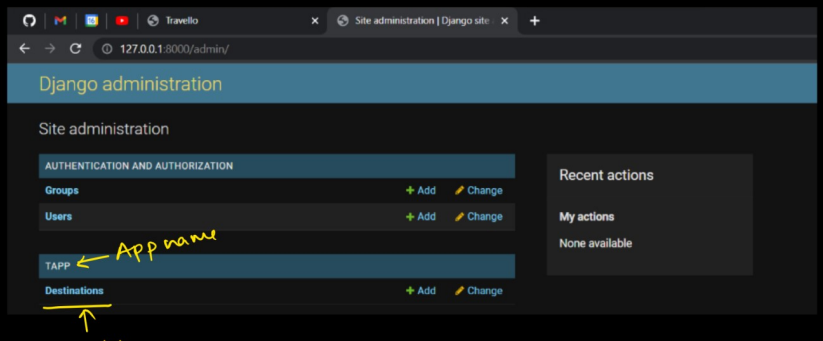
it means we have to create a page.

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

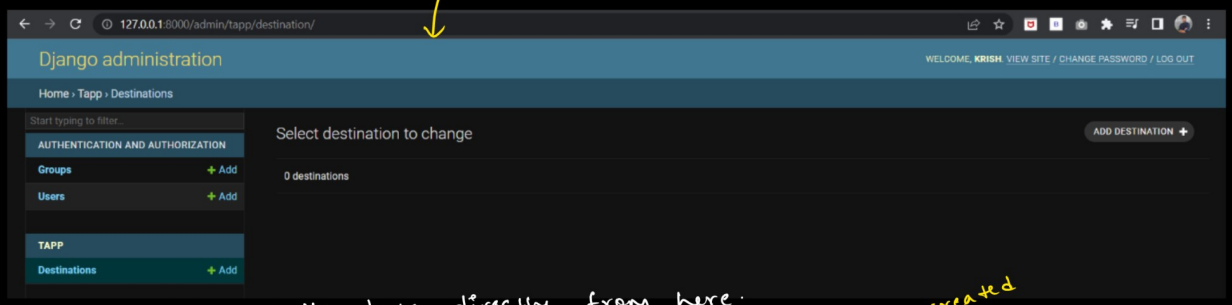
views.py | models.py | admin.py | settings.py | 0001_initial.py | 0001_initial.cpython-310.pyc

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
6  #  here we need to register the model and automatically it we create a page for us.
7  admin.site.register(Destination)  #need to give th model name here
```

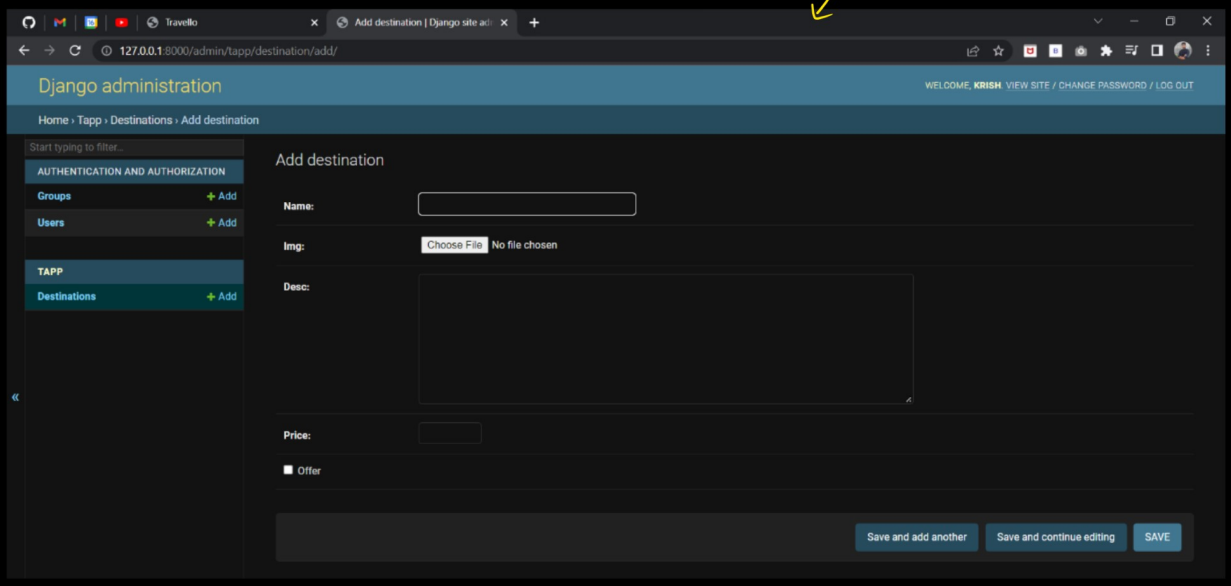


page created.



⇒ Now we can add the data directly from here.

page created



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.

folder name to create

→ 'media' folder will be created in base directory.

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

Media url

of project

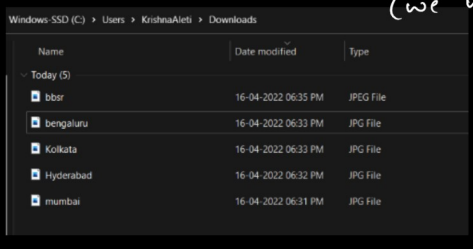
media-url

media path

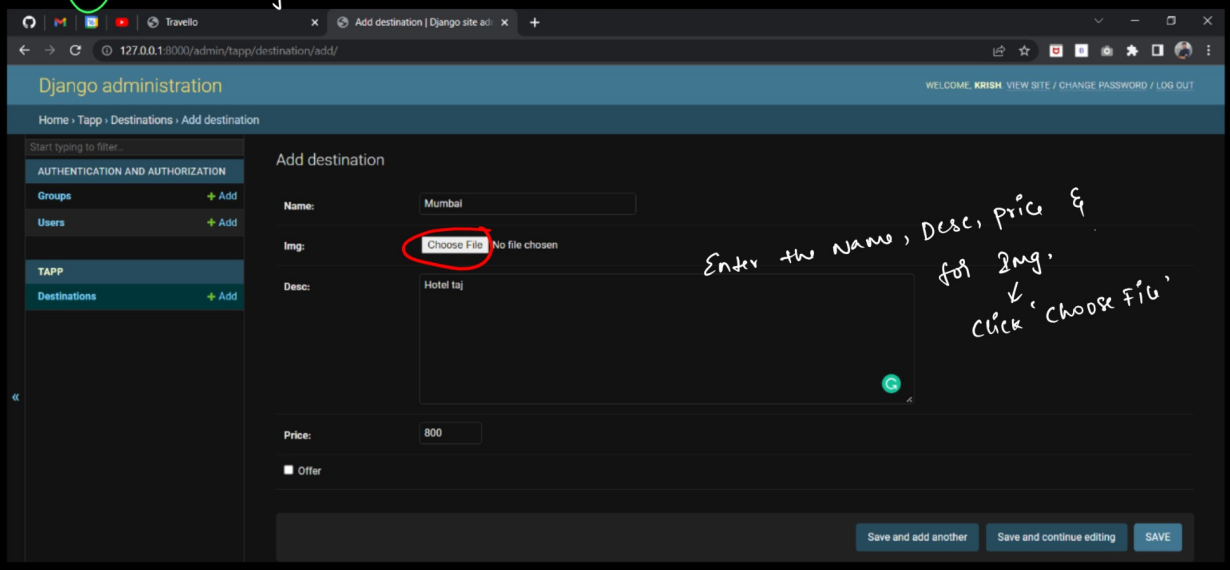
need to add path of the 'media' folder & its url.

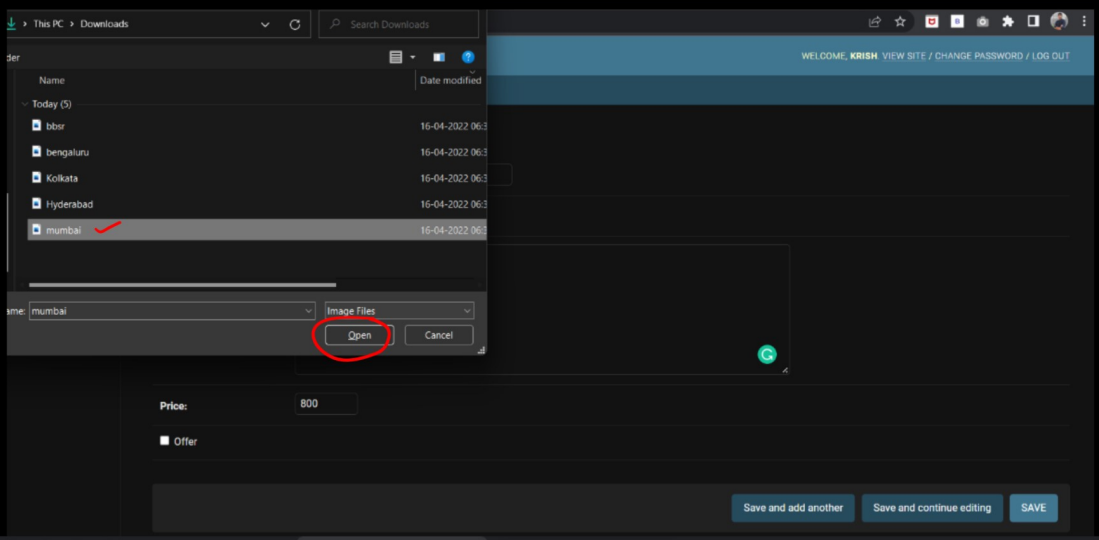
→ Download images from the internet.

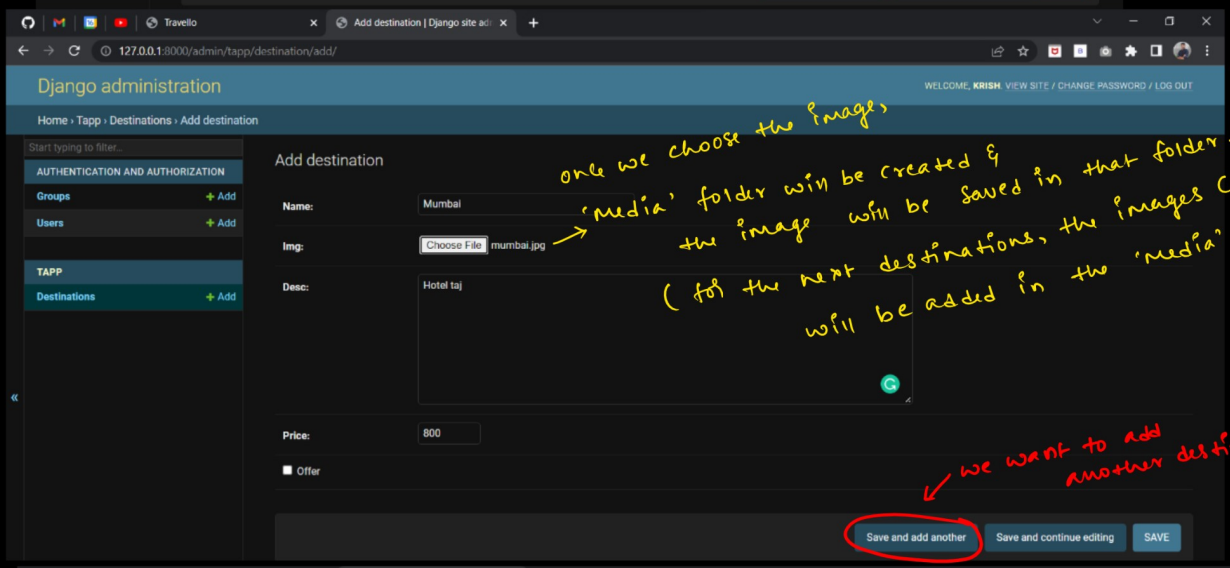
(we will give these while creating the destinations)

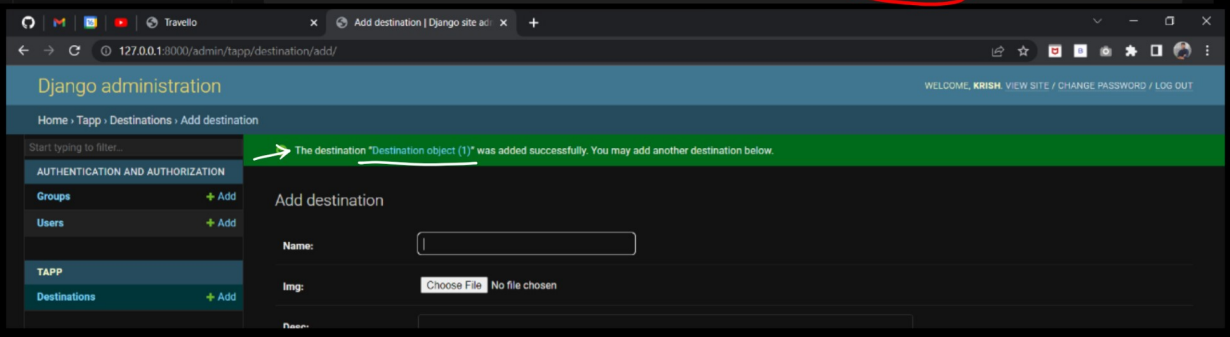


(2) → Adding the Destinations:

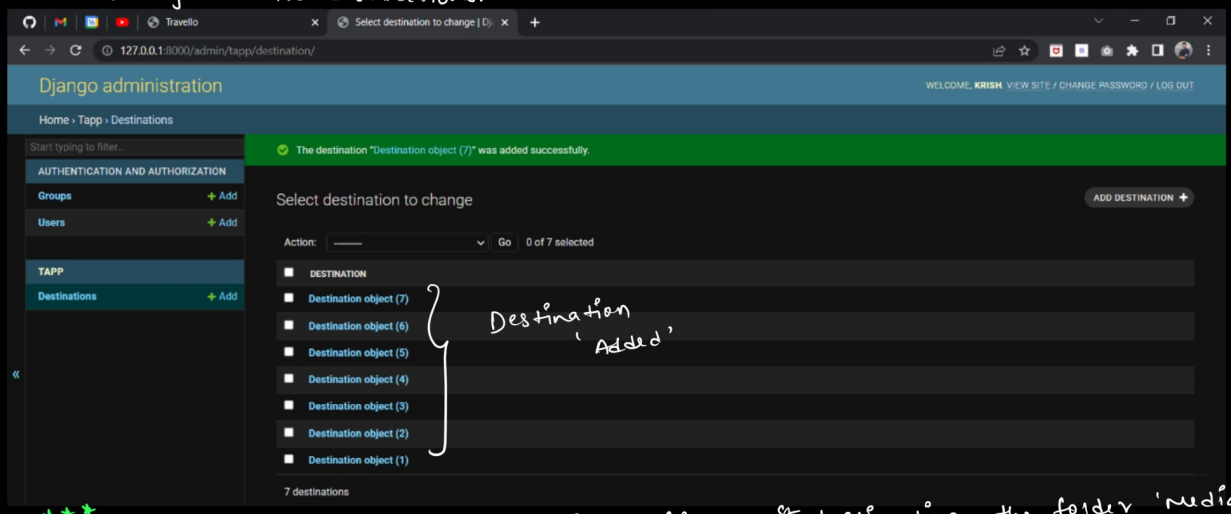




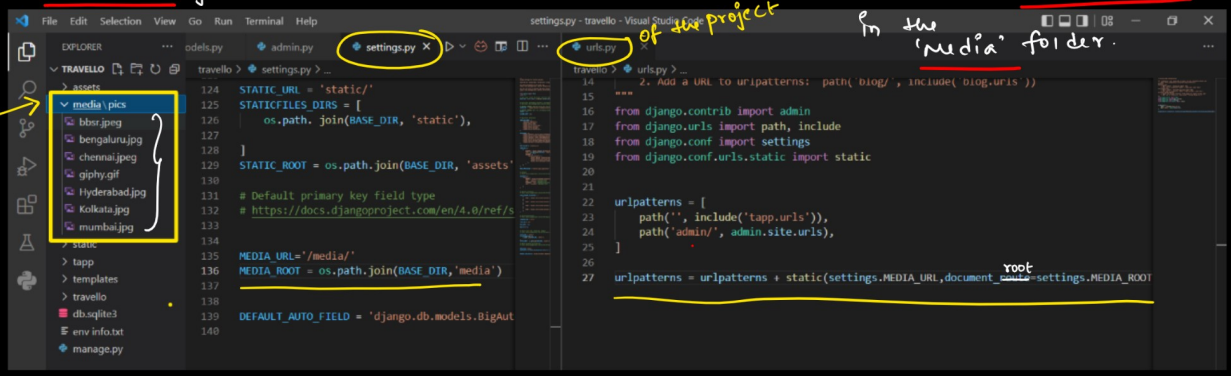




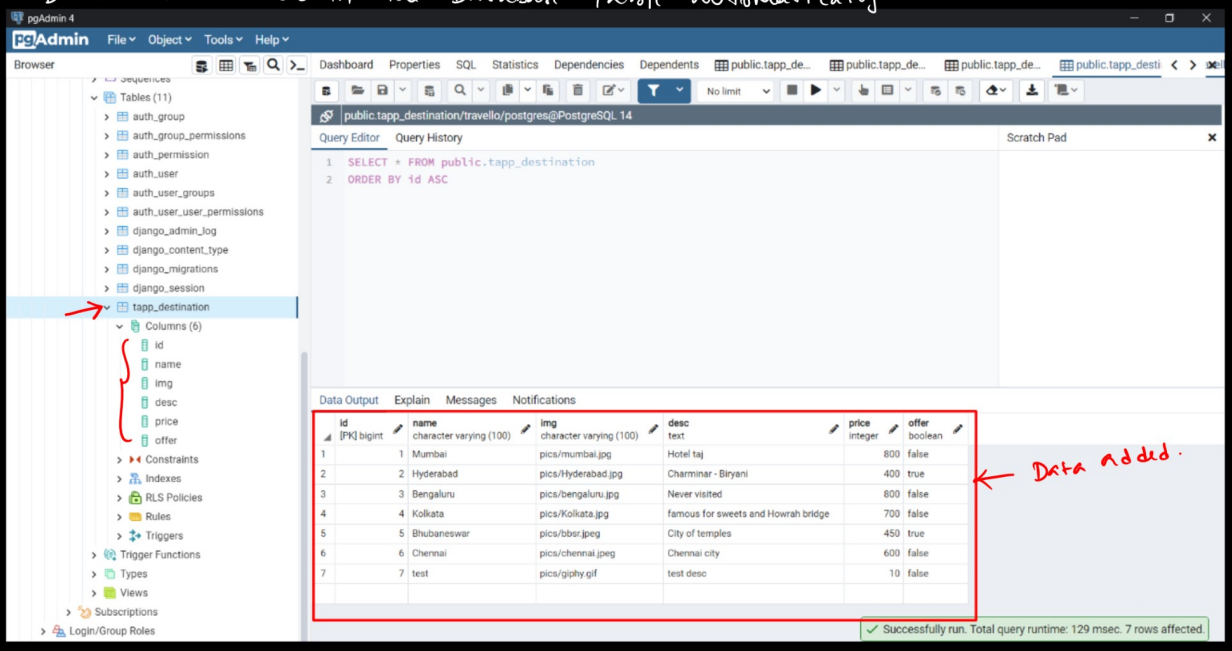
→ Similarly add the destinations:



→ As soon as we choose a files 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 in the 'media' folder.

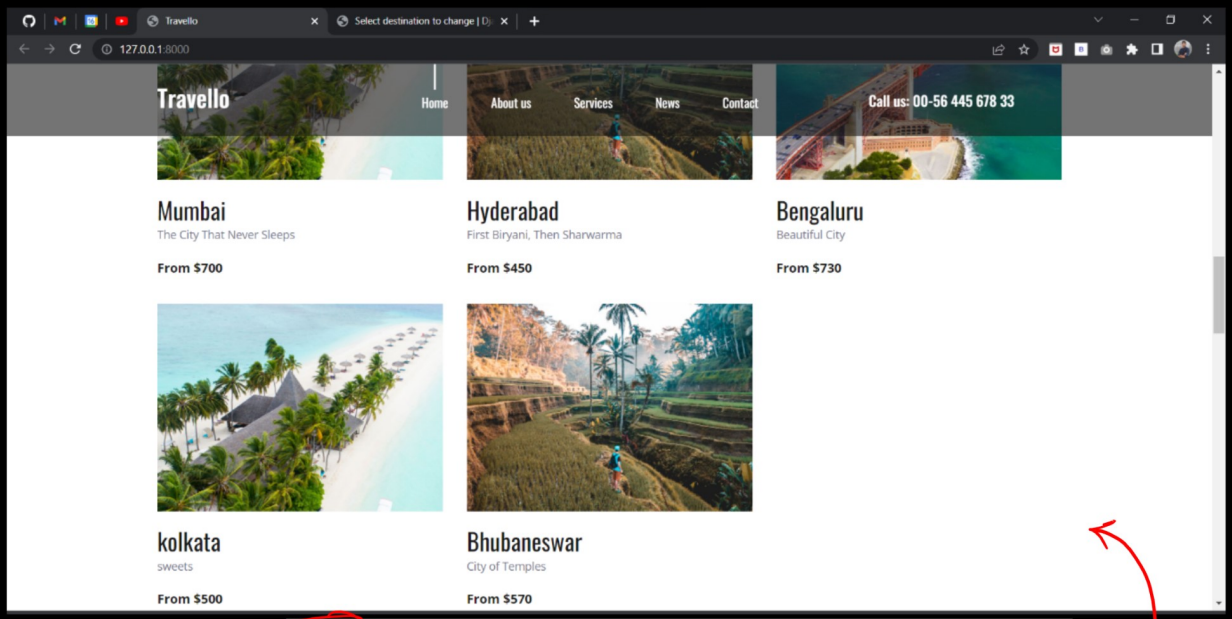


(3) Data will be added in the Database - Table automatically.



id [PK] bigint	name character varying (100)	img character varying (100)	desc text	price integer	offer boolean
1	Mumbai	pics/mumbai.jpg	Hotel taj	800	false
2	Hyderabad	pics/Hyderabad.jpg	Charminar - Biryani	400	true
3	Bengaluru	pics/bengaluru.jpg	Never visited	800	false
4	Kolkata	pics/Kolkata.jpg	famous for sweets and Howrah bridge	700	false
5	Bhubaneswar	pics/bbsr.jpeg	City of temples	450	true
6	Chennai	pics/chennai.jpeg	Chennai city	600	false
7	test	pics/giphy.gif	test desc	10	false

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




```python
dest1 = Destination()
dest1.name = 'Mumbai'
dest1.desc = 'The City That Never Sleeps'
dest1.img = 'destination_1.jpg' #we need to mention the name of image which is in the images folder od static folder
dest1.price = 700
dest1.offer = False

dest2 = Destination()
dest2.name = 'Hyderabad'
dest2.desc = 'First Biryani, Then Sharwarma'
dest2.img = 'destination_2.jpg'
dest2.price = 450
dest2.offer = True

dest3 = Destination()
dest3.name = 'Bengaluru'
dest3.desc = 'Beautiful City'
dest3.img = 'destination_3.jpg'
dest3.price = 730
dest3.offer = False

dest4 = Destination()
dest4.name = 'kolkata'
dest4.desc = 'sweets'
dest4.img = 'destination_1.jpg'
dest4.price = 500
dest4.offer = False

dest5 = Destination()
dest5.name = 'Bhubaneswar'
dest5.desc = 'City of Temples'
dest5.img = 'destination_2.jpg'
dest5.price = 570
dest5.offer = False
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.

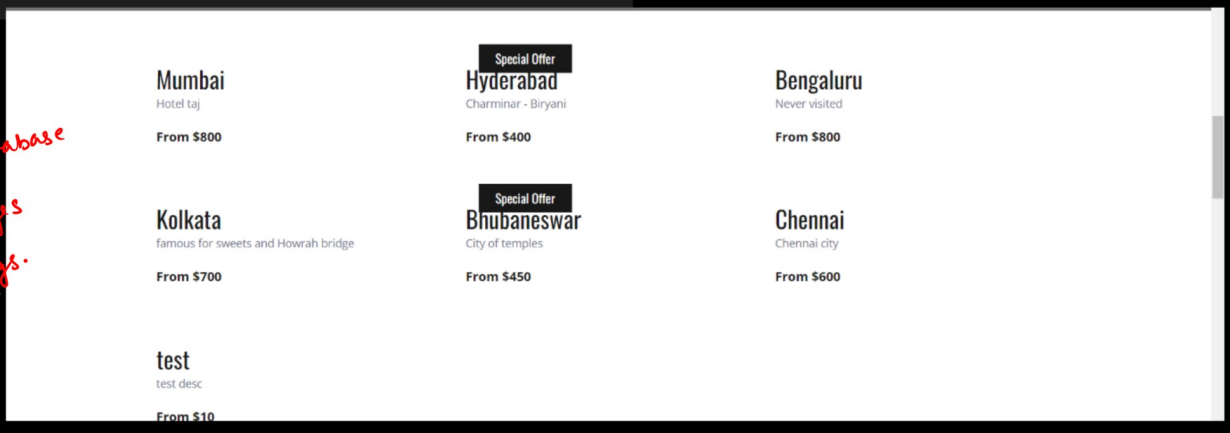
dynamic data

```python
from django.shortcuts import render, HttpResponse
from django.http import HttpResponse
# from . import models
from .models import Destination


# Create your views here.

# def index(request):
#     return render (request, 'index.html',{'discount':'10%'})

def index(request):

    dests = Destination.objects.all()
    #we are getting all the data from the database and storing it in a 'dests' variable.

    return render(request, 'index.html', {'dests': dests,'discount':'10%'})

```

← fetching all the objects from the database

we got the destinations from the database

BUT the images are missings.



***

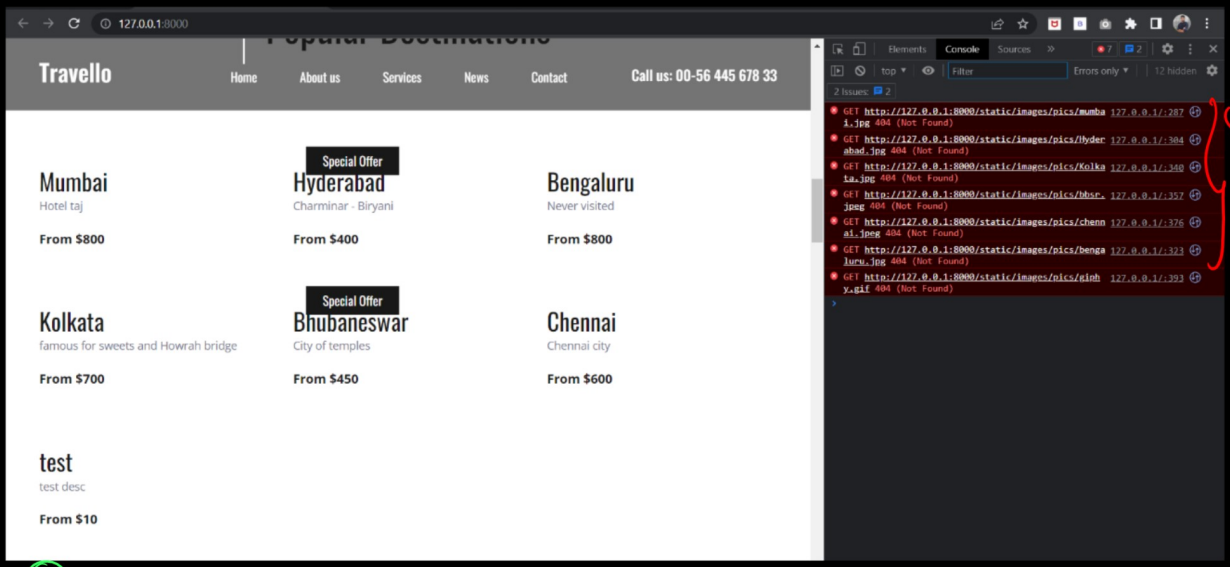
Steps to display the dynamic data from the database:

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

↓

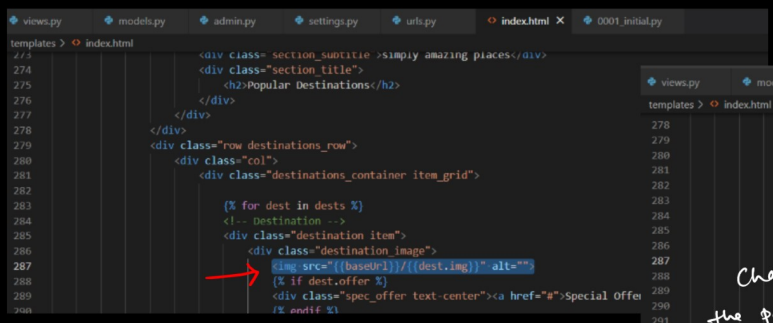
The data will be inserted into the database_table automatically → Data fetched from the database will be displayed in the browser.

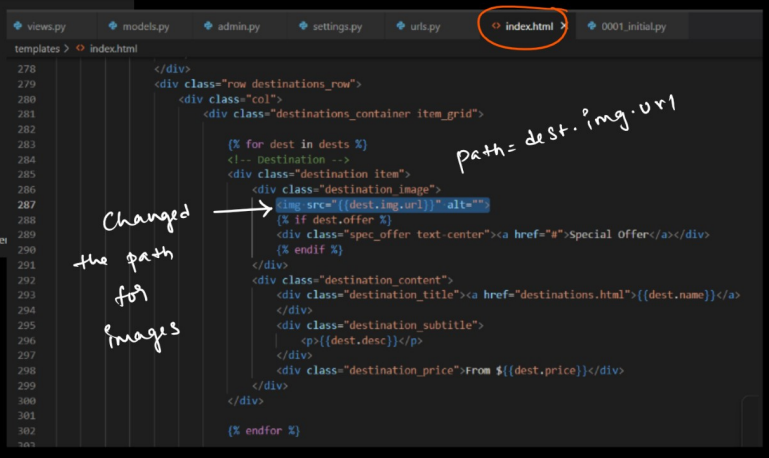
→ Here the destinations (details) are getting displayed except the images.



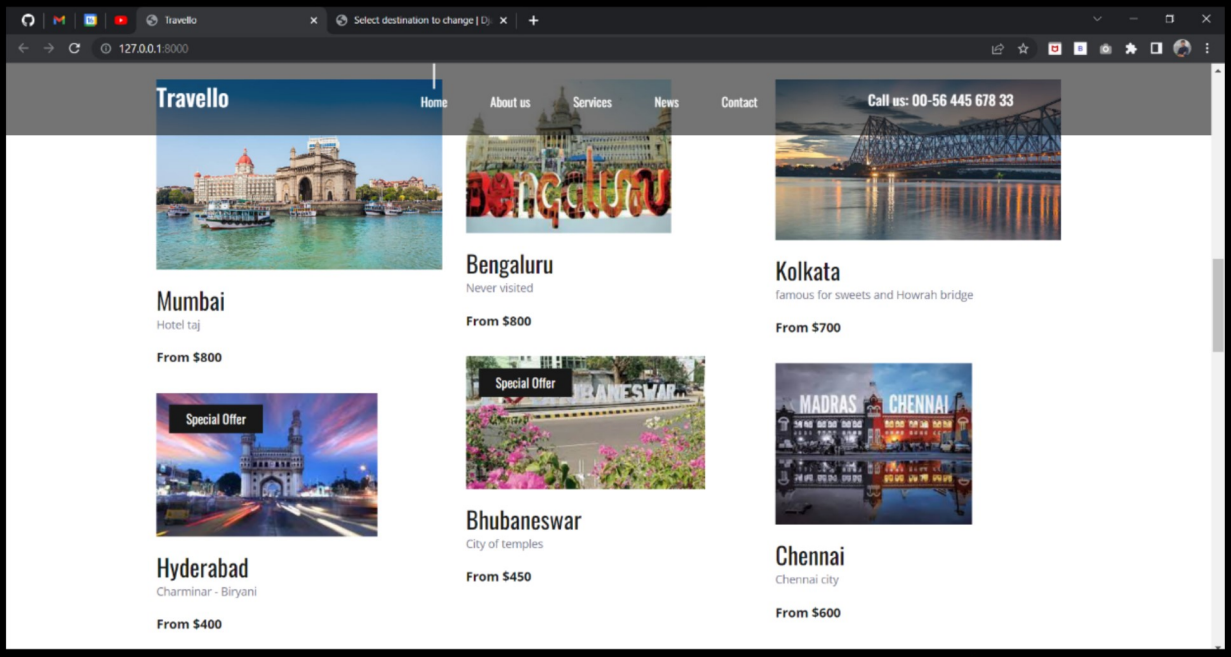
(5)

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





⇒ Destinations are fetched from database & displayed in browser.



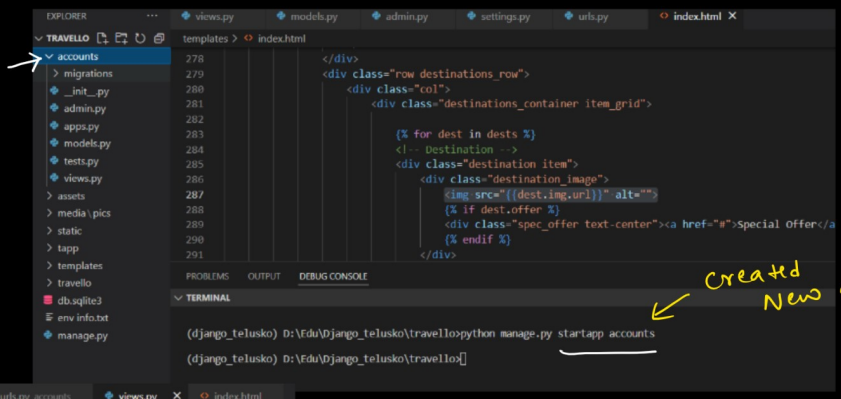
(I had removed the 'test' place)

User Registration in Django:

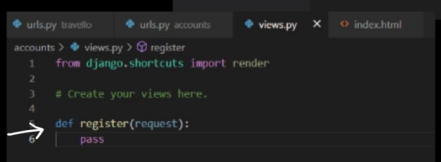
We will creates 2 pages — Registration, login.

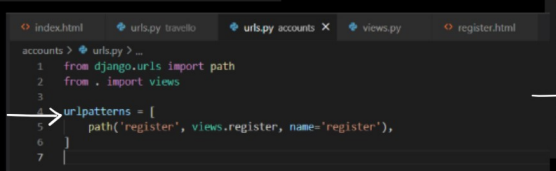
→ Now we can create them in the same app but lets try something new.

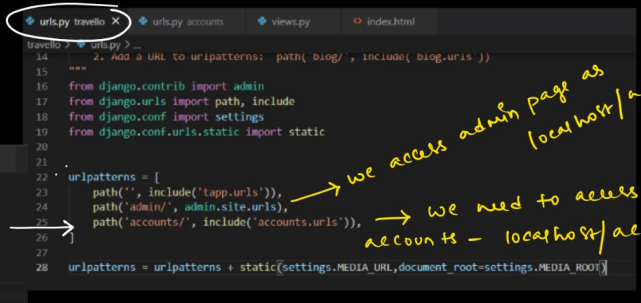
lets create another app & keep the Registration, login features in it.



Created New app - 'accounts'

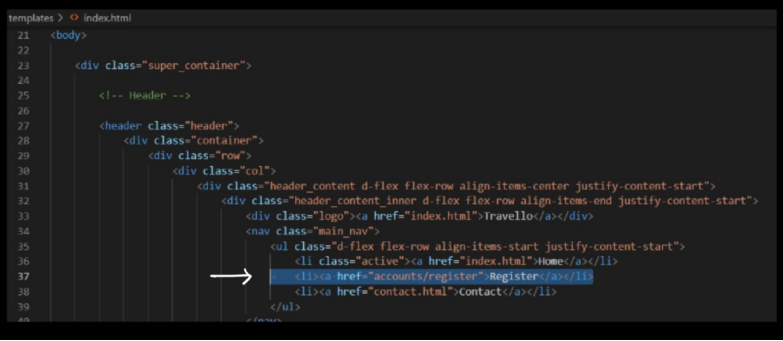


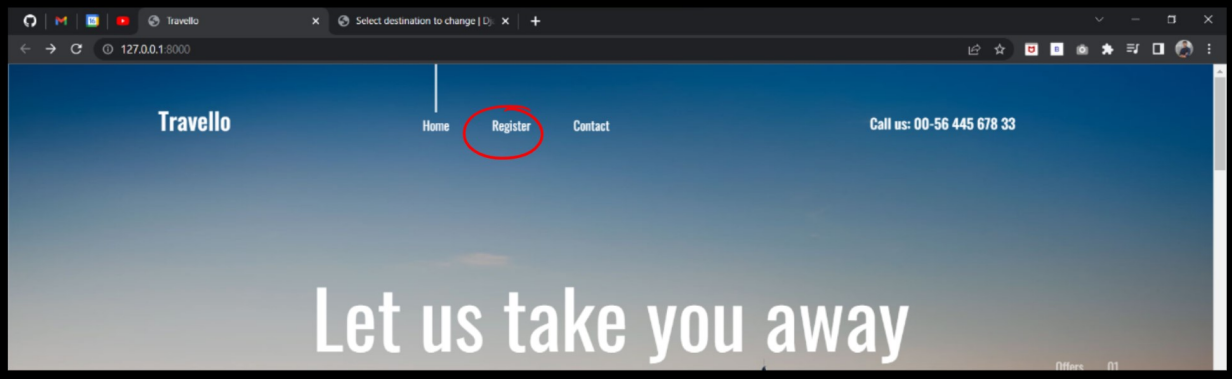




→ we access admin page as localhost/admin

→ we need to access accounts — localhost/accounts.



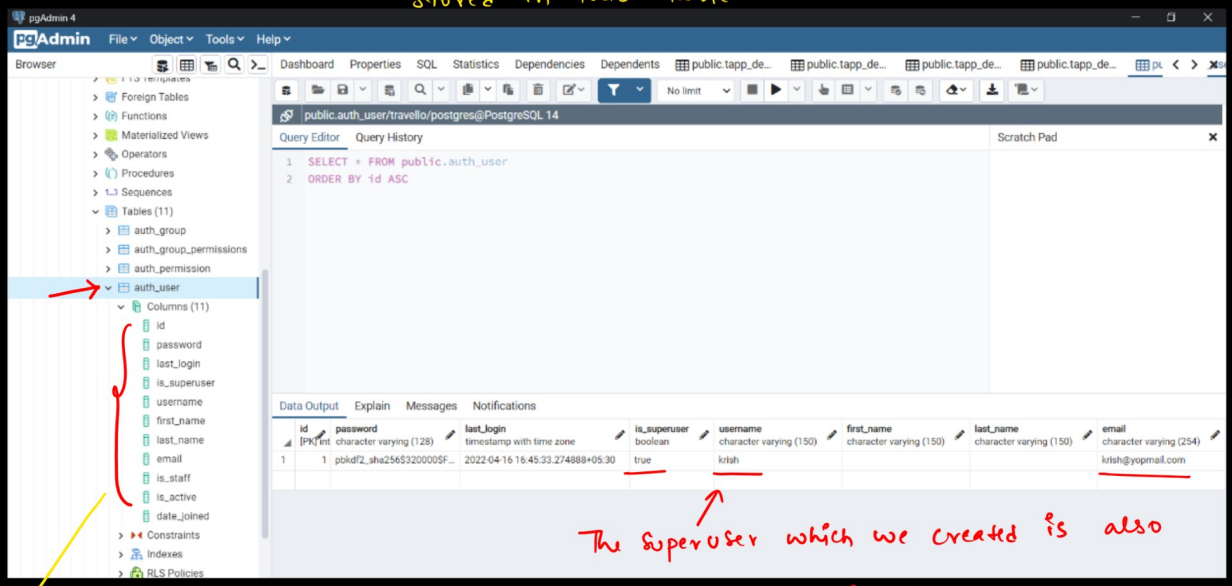



```python
# accounts > views.py > register
from django.shortcuts import render

# Create your views here.

def register(request):
    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.

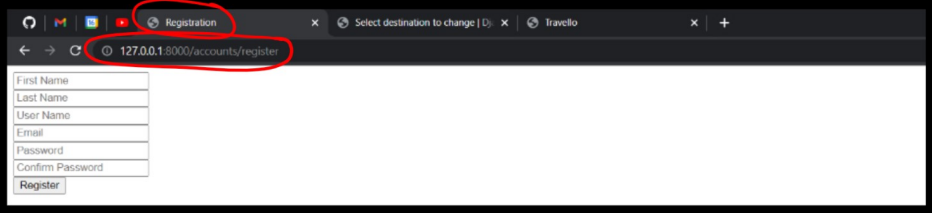


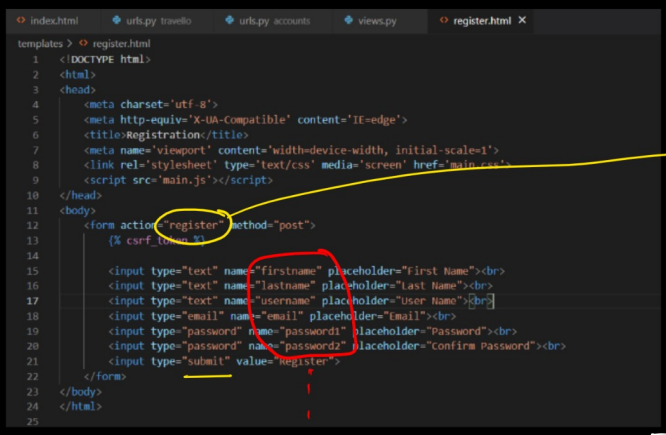
The superuser which we created is also present in the table.

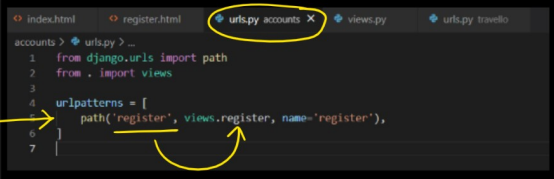
→ we need to create a Registration form/page as per the fields in auth_user table.

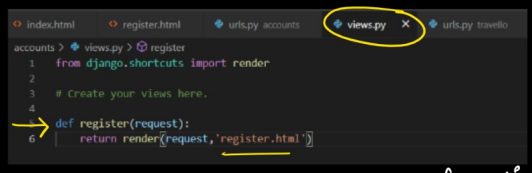
Registration form →

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>Registration</title>
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
    <script src='main.js'></script>
</head>
<body>
    <form action="register" method="post">
        {% csrf_token %}

        <input type="text" name="firstname" placeholder="First Name"><br>
        <input type="text" name="lastname" placeholder="Last Name"><br>
        <input type="text" name="username" placeholder="User Name"><br>
        <input type="email" name="email" placeholder="Email"><br>
        <input type="password" name="password1" placeholder="Password"><br>
        <input type="password" name="password2" placeholder="Confirm Password"><br>
        <input type="submit" value="Register">
    </form>
</body>
</html>
```



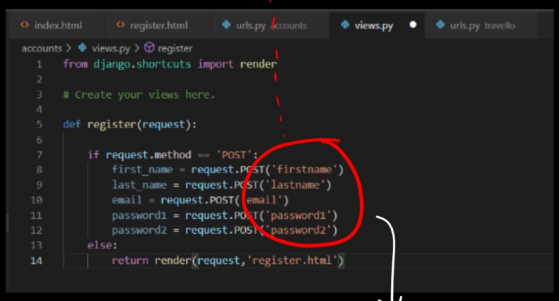






⇒ Here we are using the same function register() for fetching the form & returning register.html

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



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.

```python
from django.shortcuts import render, redirect
from django.contrib.auth.models import User, auth   # importing the user model. auth is for the login

# Create your views here.

def register(request):

    if request.method == 'POST':
        firstname = request.POST['firstname']
        lastname = request.POST['lastname']
        username = request.POST['username']
        email = request.POST['email']
        password1 = request.POST['password1']
        password2 = request.POST['password2']

        user=User.objects.create_user(username=username, password=password1, email=email, first_name=firstname, last_name=lastname)
        user.save();
        print('User created')
        return redirect('/')
    else:
        return render(request,'register.html')
```

this is of the model

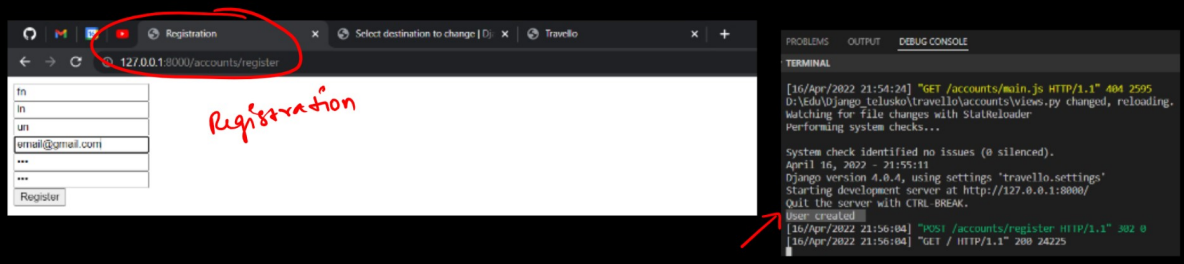
password that we enter

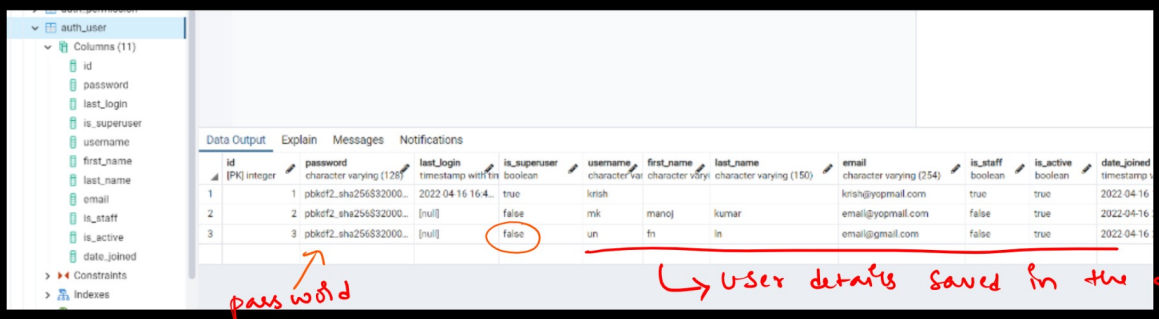
Creating user

Creating an object

we need to save the data

after the registration is done, the page should redirect to homepage.





password stored in secured format.

User details saved in the database

→ 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.

```python
from django.shortcuts import render, redirect
from django.contrib.auth.models import User, auth   # importing the user model. auth is for the login

# Create your views here.

def register(request):

    if request.method == 'POST':
        firstname = request.POST['firstname']
        lastname = request.POST['lastname']
        username = request.POST['username']
        email = request.POST['email']
        password1 = request.POST['password1']
        password2 = request.POST['password2']

        if password1==password2:
            if User.objects.filter(username=username).exists(): #checking whether the username already exists in the database
                print('error:Username already exists')
            elif User.objects.filter(email=email).exists():
                print('error:email already taken')
            else:
                user=User.objects.create_user(username=username, password=password1, email=email, first_name=firstname, last_name=lastname)
                user.save();
                print('User created')
                return redirect('/')
        else:
            print("The Confirm password doesn't match with the entered password")
    else:
        return render(request,'register.html')
```

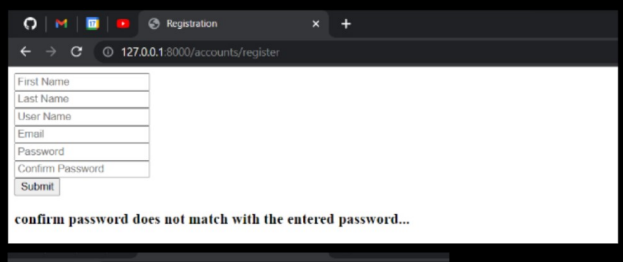
→ here we are printing the error messages in the console.
⇒ we need to display them on the page.

Passing Messages in Django:

```python
from django.shortcuts import render, redirect
from django.contrib.auth.models import User, auth   # importing the user model. auth is for the login
from django.contrib import messages

# Create your views here.

def register(request):

    if request.method == 'POST':
        firstname = request.POST['firstname']
        lastname = request.POST['lastname']
        username = request.POST['username']
        email = request.POST['email']
        password1 = request.POST['password1']
        password2 = request.POST['password2']

        if password1==password2: # checking if the confirm password matches with the entered password
            if User.objects.filter(username=username).exists(): # checking whether the username already exists in the database
                messages.info(request, 'Username already exists')  # message.info is the method in django to display the message
                return redirect('register') #if we get the error we need get redirected to the registration page
            elif User.objects.filter(email=email).exists(): # checking whether the email is taken and  already exists in the database
                messages.info(request, 'email already taken')
                return redirect('register') #if we get the error we need get redirected to the registration page
            else:
                user=User.objects.create_user(username=username, password=password1, email=email, first_name=firstname, last_name=lastname)
                user.save();
                print('User created')
                return redirect('/')
        else:
            messages.info(request, 'confirm password does not match with the entered password...')
            return redirect('register') #if we get the error we need get redirected to the registration page
        return redirect('/')
    else:
        return render(request,'register.html')
```

In django we have inbuilt method 'messages' to display the messages on the pages.

```html
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>Registration</title>
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    {% comment %} <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
    <script src='main.js'></script> {% endcomment %}
</head>
<body>
    <form action="register" method="post">
        {% csrf_token %}

        <input type="text" name="firstname" placeholder="First Name"><br>
        <input type="text" name="lastname" placeholder="Last Name"><br>
        <input type="text" name="username" placeholder="User Name"><br>
        <input type="email" name="email" placeholder="Email"><br>
        <input type="password" name="password1" placeholder="Password"><br>
        <input type="password" name="password2" placeholder="Confirm Password"><br>
        <input type="submit" >
    </form>
    <div>
        {% for message in messages %}
        <h3> {{message}}  </h3>
        {% endfor %}
    </div>

</body>
</html>
```



Database →

Data Output | Explain | Messages | Notifications

	id [PK]	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active	date_joined
		character varyin	timestamp with	boolean	character var	character varyin	character var	character varying (254)	boolean	boolean	timestamp with time zone
1	1	pbkdf2_sha2...	2022-04-16 1...	true	krish			krish@yopmail.com	true	true	2022-04-16 16:40:02.566712+05:30
2	9	pbkdf2_sha2...	[null]	false	learncode	learner	coder	lc@yopmail.com	false	true	2022-04-17 07:04:25.347306+05:30
3	10	pbkdf2_sha2...	[null]	false	un	fn	ln	email@yopmail.com	false	true	2022-04-17 07:08:29.771166+05:30
4	11	pbkdf2_sha2...	[null]	false	dj	django	user	dj@ypmail.com	false	true	2022-04-17 07:12:08.774945+05:30

User login

templates > login.html

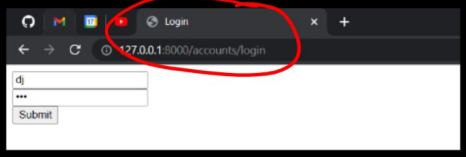
```html
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>Login</title>
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    {% comment %} <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
    <script src='main.js'></script> {% endcomment %}
</head>
<body>
    <form action="login" method="post">
        {% csrf_token %}

        <input type="text" name="username" placeholder="User Name"><br>
        <input type="password" name="password" placeholder="Password"><br>
        <input type="Submit" >
    </form>
    <div>
        {% for message in messages %}
        <h3> {{message}}  </h3>
        {% endfor %}
    </div>

</body>
</html>
```

accounts > views.py > login

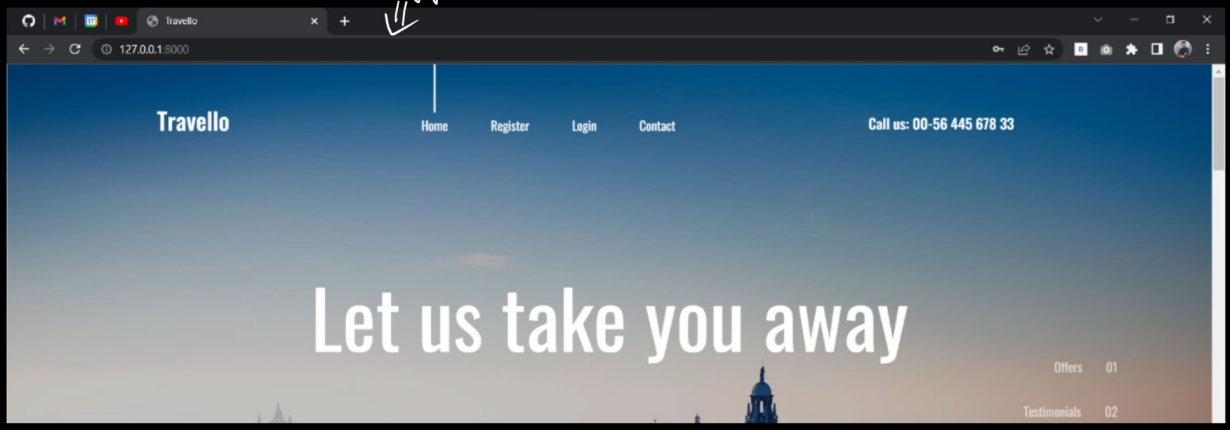
```python
from django.shortcuts import render, redirect
from django.contrib.auth.models import User, auth   # importing the user model. auth is for the login
from django.contrib import messages

# Create your views here.

def login (request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password'] # we need to verify the login creds now

        user = auth.authenticate(username=username, password=password) # auth is for the login
        #  if the username and password exists, it returns the user object which means it is not none

        if user is not None:
            auth.login(request, user) # we are providing the login access to the current user
            return redirect("/") # we nee to redirect to the home page if the login is successful.
        else:
            messages.error(request, 'Invalid credentials')
            return redirect('login') # if the user is not authenticated, we need to redirect to the login page

    else:
        return render(request,'login.html')
```

accounts > urls.py > ...

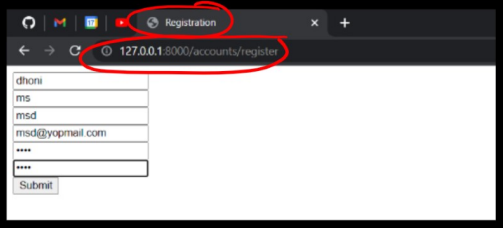
```python
from django.urls import path
from . import views

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

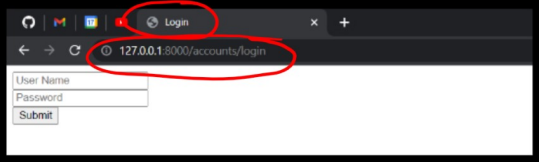


User logged-in



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

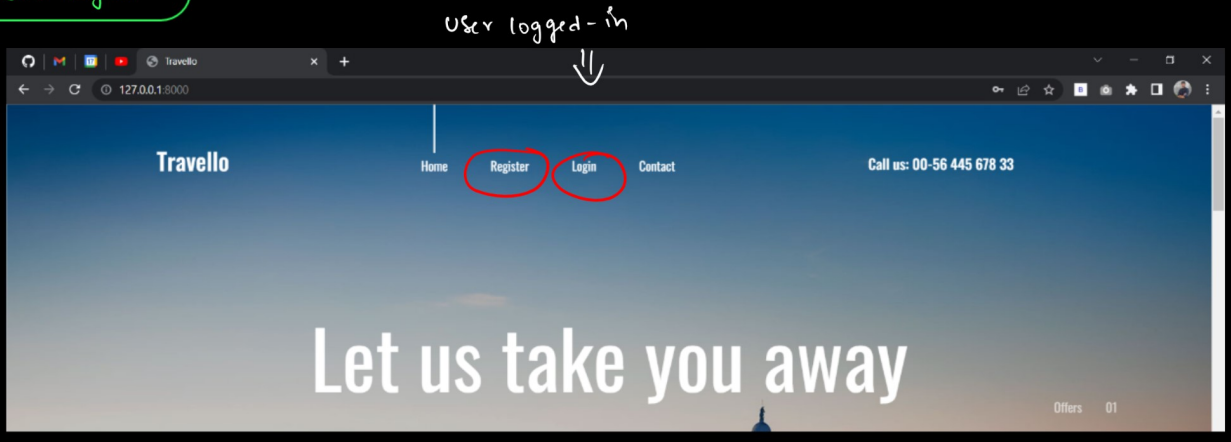




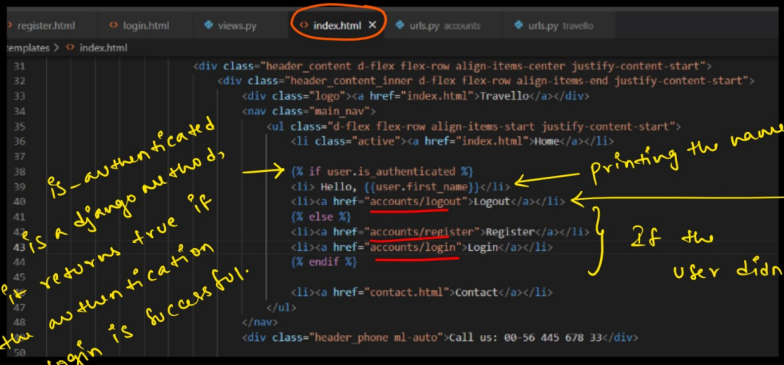
accounts > views.py > register

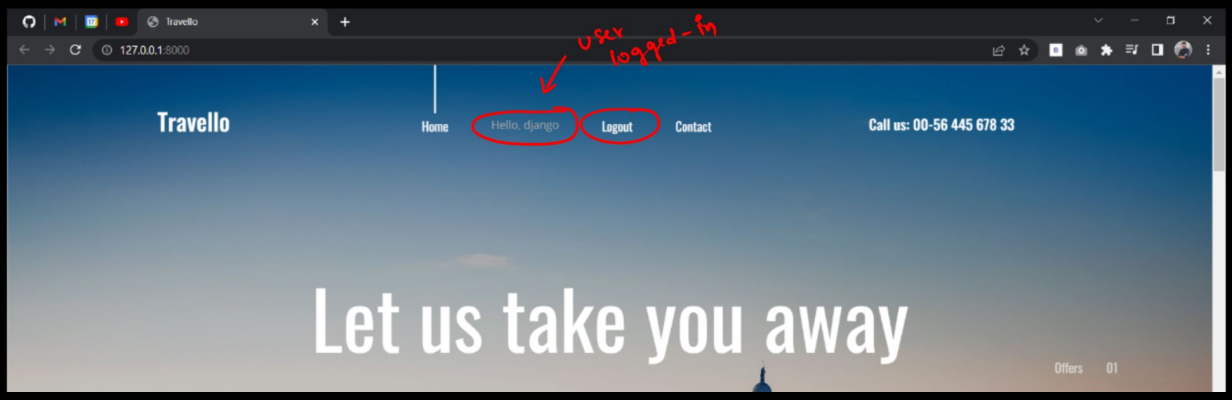
```python
def register(request):

    if request.method == 'POST':
        firstname = request.POST['firstname']
        lastname = request.POST['lastname']
        username = request.POST['username']
        email = request.POST['email']
        password1 = request.POST['password1']
        password2 = request.POST['password2']

        if password1==password2: # checking if the confirm password matches with the entered password
            if User.objects.filter(username=username).exists(): # checking whether the username already exists in the database
                messages.info(request, 'Username already exists')  # message.info is the method in django to display the message
                return redirect('register') #if we get the error we need get redirected to the registration page
            elif User.objects.filter(email=email).exists(): # checking whether the email is taken and  already exists in the database
                messages.info(request, 'email already taken')
                return redirect('register') #if we get the error we need get redirected to the registration page
            else:
                user=User.objects.create_user(username=username, password=password1, email=email, first_name=firstname, last_name=lastname)
                user.save();
                return redirect('login') # if the user is created successfully, we need to redirect to the login page
        else:
            messages.info(request, 'confirm password does not match with the entered password...')
            return redirect('register') #if we get the error we need get redirected to the registration page
        return redirect('/')
    else:
        return render(request,'register.html')
```

User logout

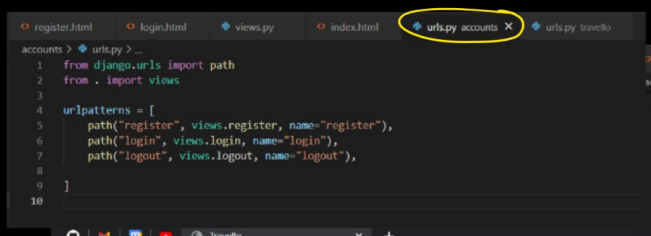


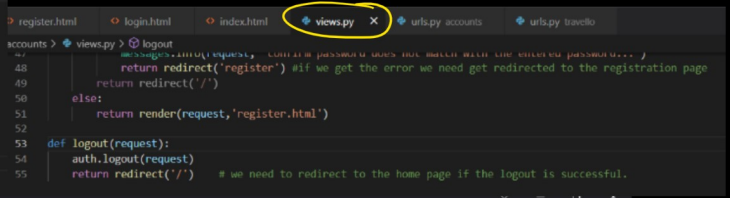
→ After login, we still see the Register, login button and we don't have the logout button. ⇒ we need to fix these.

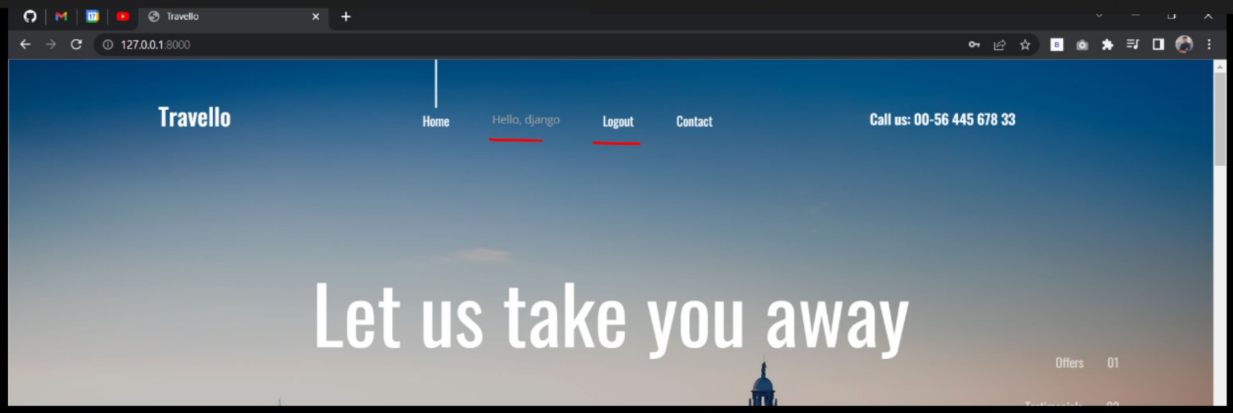




→ for logout:







The logged-in user details in database:

Data Output | Explain | Messages | Notifications

	id [PK]	password character varying (128)	last_login timestamp with time zone	is_superuser boolean	username	first_name	last_name	email character varying (254)	is_staff	is_active	date_joined timestamp with time zone
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+05:3
2	9	pbkdf2_sha256$320...	[null]	false	learncode	learner	coder	lc@yopmail.com	false	true	2022-04-17 07:04:25.347306+05:3
3	10	pbkdf2_sha256$320...	[null]	false	un	fn	ln	email@yopmail.com	false	true	2022-04-17 07:08:29.771166+05:3
4	11	pbkdf2_sha256$320...	2022-04-17 10:51:06.292101+05:30	false	dj	django	user	dj@ypmail.com	false	true	2022-04-17 07:12:08.774945+05:3
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