

# Universidad Nacional San Agustín de Arequipa

Facultad de Ingeniería de Producción y Servicios

Escuela Profesional de Ingeniería de Sistemas



**Curso:**

Programación Web 2

**Profesor:**

Carlo Jose Luis Corrales Delgado

**Tema:**

Laboratorio 7 - DJANGO - Plantillas Turísticas

**Integrante:**

Cuno Cahuari Armando Steven

**Repositorio:**

<https://github.com/SteveArms/pwebDJANGOPlantillas.git>

**2024**

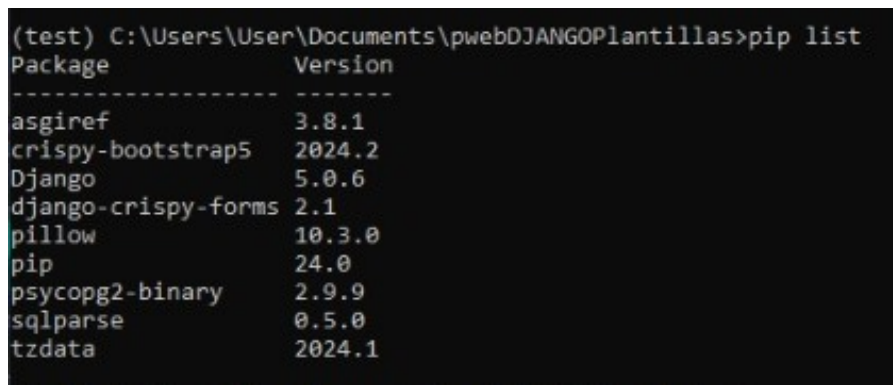
## Informe Nro 7 - Django Plantillas Turisticas

La tarea actual implica utilizar una plantilla web de Lugares Turísticos para replicar su código utilizando las diversas funciones proporcionadas por Django, como modelos, vistas, plantillas, conexión a la base de datos, entre otros. Además, se nos solicita implementar métodos para agregar, modificar, listar y eliminar estos datos que provienen del modelo, que actúa como la base de datos.

### Actividad del Video - Plantilla Turistica

#### Entorno virtual

El entorno virtual que utilizaremos para este proyecto contendrá las herramientas necesarias para su funcionamiento, como Django, PostgreSQL, Pillow y otras bibliotecas requeridas. Esto garantizará un entorno aislado y adecuado para el desarrollo de nuestra aplicación.



```
(test) C:\Users\User\Documents\pwebDJANGOPlantillas>pip list
Package            Version
-----
asgiref            3.8.1
crispy-bootstrap5  2024.2
Django             5.0.6
django-crispy-forms 2.1
pillow             10.3.0
pip               24.0
psycpg2-binary     2.9.9
sqlparse           0.5.0
tzdata            2024.1
```

Figure 1: Entorno virtual

#### App - Travello

La aplicación "travello" contendrá los datos correspondientes en forma de modelo, con el cual trabajaremos. Además, definiremos las vistas que generarán las respuestas en HTML, teniendo en cuenta las funciones y solicitudes del usuario. Utilizaremos una estructura de HTML basada en una plantilla que rescatará el index para su funcionamiento. Además, en la actividad del video, implementaremos un sistema de inicio de sesión, registro y cierre de sesión (Login, Register, y Logout).

## Models Travello

El modelo "Destination" representa la estructura de datos para los Lugares Turísticos y se almacenará en la base de datos predeterminada de Django, que es SQLite3. Hereda los métodos y clases de models y especifica cinco datos: nombre, imagen, descripción, precio y estado de oferta. Las imágenes se guardan en una carpeta especial llamada "pics", separada de los elementos estáticos.

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

Figure 2: Modelo Destination

## Forms Travello

Crearemos un objeto llamado "DestinationForm" para gestionar los datos ingresados por el usuario. Este formulario contendrá campos correspondientes a los datos que deseamos recolectar. Utilizaremos el atributo "fields" para especificar qué campos incluir en el formulario.

```
1  from django import forms
2  from .models import Destination
3  from django import forms
4
5  class DestinationForm(forms.ModelForm):
6      class Meta:
7          model = Destination
8          fields = '__all__'
```

Figure 3: Forms Destination

## Vistas Travello

Las vistas son responsables de manejar las solicitudes del usuario y, utilizando las funciones proporcionadas por Django, devolverán una plantilla HTML con los datos solicitados. Estas vistas pueden interactuar tanto con formularios como con los modelos previamente definidos, proporcionando así una respuesta adecuada al usuario.

```
1 from django.shortcuts import render, redirect
2 from django.conf import settings
3 import os
4 from .models import Destination
5 from .forms import DestinationForm
6
7 def index(request):
8     dests = Destination.objects.all()
9     return render(request, "index.html", {'dests': dests})
10
11 def listarDestinos(request):
12     dests = Destination.objects.all().order_by('id')
13     return render(request, "listaDest.html", {'destinos': dests})
14
15 def editDestinos(request, id_destino):
16     destino = Destination.objects.filter(id = id_destino).first()
17     form = DestinationForm(instance= destino)
18     return render(request, "destinoEdit.html", {"form": form, 'destino': destino})
19
20 def actualizarDestinos(request, id_destino):
21     destino = Destination.objects.get(pk = id_destino)
22     form = DestinationForm(request.POST, instance= destino)
23     if form.is_valid():
24         form.save()
25     destinos = Destination.objects.all().order_by('id')
26     return render(request, "listaDest.html", {"destinos": destinos})
27
28 def eliminarDest(request, id_destino):
29     destino = Destination.objects.get(pk = id_destino)
30     destino.delete()
31     destinos = Destination.objects.all().order_by('id')
32     return render(request, "listaDest.html", {"destinos": destinos, "mensaje": 'OK'})
```

Figure 4: Vistas de la App

## Admin Travello

Para facilitar la gestión de la base de datos proporcionada por el modelo, es crucial vincularla con el sitio de administración de Django. A través del panel de administración, al acceder a la URL con 'admin/', podemos realizar acciones como agregar, eliminar o modificar objetos directamente en la base de datos. Este flujo simplifica significativamente la administración de datos del sitio web.

```
1 from django.contrib import admin
2 from .models import Destination
3 # Register your models here.
4
5 admin.site.register(Destination)
```

Figure 5: Admin de la App

## Urls

Las URLs se encargan de listar las rutas a las que podemos acceder a través de la barra de direcciones del navegador. Estas rutas especifican los diferentes recursos y páginas disponibles en nuestra aplicación web.

```
17 from django.contrib import admin
18 from django.urls import path, include
19 from django.conf import settings
20 from django.conf.urls.static import static
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('', include('travello.urls')),
24     path('accounts/', include('accounts.urls'))
25 ]
26
27 urlpatterns = urlpatterns + static(settings.MEDIA_URL, document_root = settings.MEDIA_ROOT)
```

Figure 6: URLs del proyecto

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.index, name="index"),
6     path('addDest/', views.addDestination, name="addDest"),
7     path('listaDest/', views.listarDestinos, name="listaDest"),
8     path('editDest/<int:id_destino>', views.editDestinos, name="editDest"),
9     path('actDest/<int:id_destino>', views.actualizarDestinos, name="actDest"),
10    path('elimDest/<int:id_destino>', views.eliminarDest, name="delDest"),
11 ]
```

Figure 7: URLs de la app

## Templates

La estructura HTML está encargada de la presentación visual que verá el usuario. Esta estructura, además, incluirá elementos dinámicos y estáticos, como la implementación del lenguaje Django para renderizar contenido de manera dinámica en las plantillas HTML.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Login</title>
5 </head>
6 <body>
7     <form action="{% url 'login' %}" method="POST">
8         {% csrf_token %}
9         <input type="text" name="username" placeholder="Username"><br>
10        <input type="password" name="password" placeholder="Password"><br>
11        <input type="submit">
12    </form>
13    <div>
14        {% for message in messages%}
15        <h3> {{message}} </h3>
16        {% endfor %}
17    </div>
18 </body>
19 </html>
```

Figure 8: Ejemplo de Template

## Settings del proyecto

En nuestro proyecto, utilizaremos PostgreSQL en lugar de SQLite3 como base de datos. Además, al manejar elementos estáticos, estableceremos una ruta absoluta donde se ubicarán dichos elementos. En el formulario, hay un campo para subir imágenes, las cuales se guardarán en una ruta específica al momento de ser almacenadas.

```
127 STATIC_URL = '/static/'
128 STATICFILES_DIRS = [
129     os.path.join(BASE_DIR, 'static')
130 ]
131 STATIC_ROOT = os.path.join(BASE_DIR, 'assets')
```

Figure 9: Settings de elementos estaticos

```
82 DATABASES = {
83     'default': {
84         'ENGINE': 'django.db.backends.postgresql',
85         'NAME': 'telusko',
86         'USER': 'postgres',
87         'PASSWORD': '1234',
88         'HOST': 'localhost',
89     }
90 }
```

Figure 10: Settings de la base de datos

## Plantilla de Lugares Turisticos

### Plantilla Turistica

Para comenzar esta aplicacion, se nos pidió utilizar una plantilla turística a la que deberemos agregar destinos, así como la capacidad de modificarlos o eliminarlos de la base de datos. Esta es la plantilla que utilizaremos.

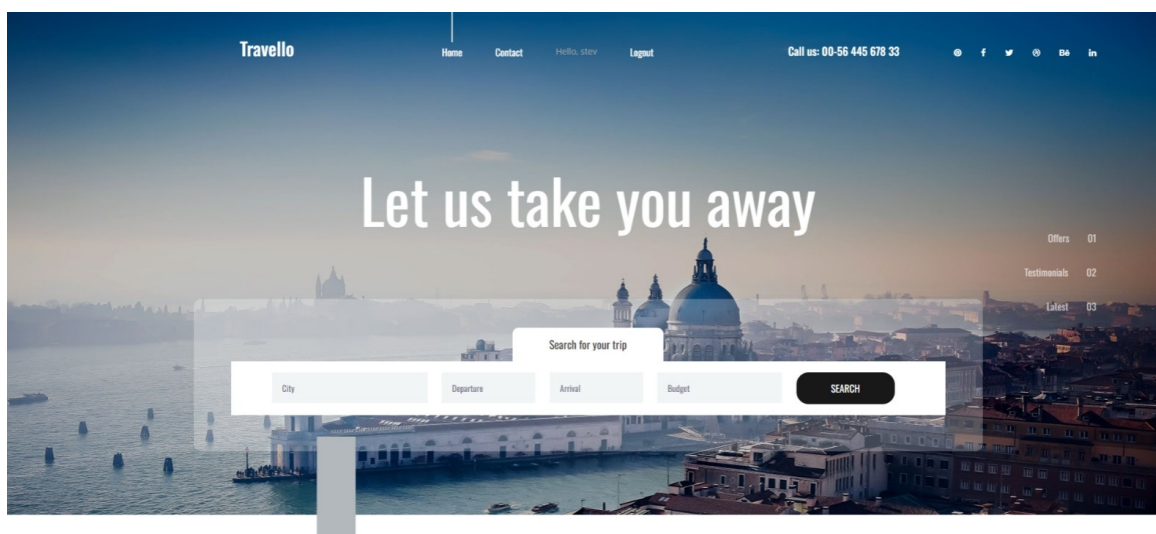


Figure 11: Plantilla

## Configuración de la base de Datos

Para comenzar con la aplicación turística, primero será necesario cambiar la base de datos predeterminada de Django, que es SQLite3, a PostgreSQL para una mejor gestión de datos. Además, utilizaremos la interfaz de administración de Django para gestionar la aplicación de manera eficiente.

```
82 DATABASES = {  
83     'default': {  
84         'ENGINE': 'django.db.backends.postgresql',  
85         'NAME': 'telusko',  
86         'USER': 'postgres',  
87         'PASSWORD': '1234',  
88         'HOST': 'localhost',  
89     }  
90 }
```

Figure 12: Base de Datos

Para conectarnos a la base de datos PostgreSQL, será necesario instalar la biblioteca 'psycopg2-binary' utilizando 'pip'.



## Vistas del Aplicativo

En las vistas, definiremos varias funciones para agregar, modificar, eliminar y listar los destinos correspondientes en la base de datos, mediante funciones que reciben las solicitudes.

Para agregar un destino sin utilizar el módulo 'forms', recopilaremos los datos en variables. Dado que vamos a ingresar imágenes, estas se guardarán en una ruta específica. Una vez que tengamos todos los datos, crearemos una instancia del modelo 'Destination' para guardarla en la base de datos. La función evaluará si el método de la solicitud es POST para determinar si debe guardar los datos. Si no lo es, creará una instancia vacía. Utilizaremos 'render' para elegir la plantilla en la que se trabajará, así como los datos seleccionados para los elementos dinámicos.

```
34 def addDestination(request):
35     if request.method == 'POST':
36         nombre = request.POST['nombre']
37         imagen = request.FILES['imagen'] # Aquí debes usar request.FILES para acceder a la imagen enviada
38         descripcion = request.POST['descripcion']
39         precio = int(request.POST['precio'])
40         oferta = bool(request.POST['oferta'])
41
42         # Generamos la ruta completa donde se guardará la imagen en la carpeta 'pics' dentro del directorio de medio
43         img_path = os.path.join(settings.MEDIA_ROOT, 'pics', imagen.name)
44
45         # Guardamos la imagen en la ruta especificada
46         with open(img_path, 'wb') as img_file:
47             for chunk in imagen.chunks():
48                 img_file.write(chunk)
49
50         # Creamos el objeto Destination con la ruta de la imagen
51         destino = Destination(name=nombre, img=os.path.join('pics', imagen.name), desc=descripcion, price=precio, of
52         destino.save()
53         print('Imagen creada')
54         return redirect('index')
55     else:
56         destino = Destination()
57     return render(request, 'addDest.html', {'destino': destino})
```

Figure 13: Vistas - agregar

Luego tenemos las funciones modificar, eliminar y enlistar que trabajaran unidas ya que mostraremos el enlistado de los destinos con botones a sus costados ya sea para modificar o eliminar los datos correspondientes, para ello en la función eliminar tendrá como parámetro la solicitud y la id del objeto que deseamos eliminar en este caso. Luego, tenemos las funciones para modificar, eliminar y enlistar destinos, que trabajarán de manera integrada. Mostraremos una lista de los destinos con botones al lado de cada uno, permitiendo modificar o eliminar los datos correspondientes.

La función eliminar recibirá como parámetros la solicitud y el ID del objeto que se desea eliminar.

Recopilaremos el ID correspondiente del módulo 'Destination' y eliminaremos el elemento de la base de datos. Luego, los datos se ordenarán y redirigiremos a la página de listado. caso recopilaremos desde el módulo Destination la id correspondiente al elementos que deseamos eliminar y luego los datos correspondientes a la base de datos serán ordenados asimismo nos redirijirá hacia la página enlistar

La función de modificar estará compuesta por dos subfunciones: editar y actualizar los datos correspondientes. La función editar recopilará, mediante la ID asignada, los valores correspondientes a esa ID. Luego, estos serán instanciados en un formulario para que podamos modificarlos y nos redirigirá a una página llamada 'destinoEdit'. La segunda función, actualizar, se encargará, una vez en 'destinoEdit', de validar los datos y guardar estos datos nuevos y modificados en el mismo objeto.



```

11 def listarDestinos(request):
12     dests = Destination.objects.all().order_by('id')
13     return render(request, "listaDest.html", {'destinos': dests})
14
15 def editDestinos(request, id_destino):
16     destino = Destination.objects.filter(id = id_destino).first()
17     form = DestinationForm(instance= destino)
18     return render(request, "destinoEdit.html", {"form": form, 'destino': destino})
19
20 def actualizarDestinos(request, id_destino):
21     destino = Destination.objects.get(pk = id_destino)
22     form = DestinationForm(request.POST, instance= destino)
23     if form.is_valid():
24         form.save()
25     destinos = Destination.objects.all().order_by('id')
26     return render(request, "listaDest.html", {"destinos": destinos})
27
28 def eliminarDest(request, id_destino):
29     destino = Destination.objects.get(pk = id_destino)
30     destino.delete()
31     destinos = Destination.objects.all().order_by('id')
32     return render(request, "listaDest.html", {"destinos":destinos, "mensaje": 'OK'})
33

```

Figure 14: Vistas - modificar, eliminar y enlistar

## Plantillas de Agregar

En el formulario que utilizaremos en esa página, consideraremos que el método será POST. La acción que realizará el formulario al interactuar con el botón corresponderá a una URL que implementamos dentro de la aplicación y será gestionada por una función. Este formulario incluirá un token para proteger los datos contra ataques CSRF.

```

1 {% load static %}
2 <DOCTYPE html>
3 <html>
4 <head>
5     <title>Agregar Destino</title>
6     <link href="{% static 'styles/agregar_styles.css' %}" rel="stylesheet">
7 </head>
8 <body>
9     <div class="content">
10         <h1>Agregar Destino</h1>
11         <div class="formulario">
12             <form action="{% url 'addDest' %}" method="POST" enctype="multipart/form-data">
13                 {% csrf_token %}
14                 <input type="text" name="nombre" placeholder="Nombre" class="form-control"><br>
15                 <input type="file" name="imagen" placeholder="Imagen" class="form-control"><br>
16                 <input type="text" name="descripcion" placeholder="Descripcion" class="form-control"><br>
17                 <input type="text" name="precio" placeholder="Precio" class="form-control"><br>
18                 <label for="oferta">Oferta:</label>
19                 <select name="oferta" id="oferta" class="form-control" required>
20                     <option value="True">True</option>
21                     <option value="False">False</option>
22                 </select><br>
23                 <input type="submit" class="boton btn">
24             </form>
25         </div>
26     </div>
27 </body>
28 </html>

```

Figure 15: Estructura HTML de agregar

## Templates de Enlistar - Modificar y Eliminar

En este archivo, al ser dinámico y recibir datos de la vista, utilizaremos una tabla para enlistar los destinos. Las filas se agregarán mediante una iteración sobre los datos. Cada fila contendrá dos botones: uno para modificar y otro para eliminar los datos correspondientes a esa fila.

```
1  {% load static %}
2  <!DOCTYPE html>
3  <html>
4  <head>
5      <title>Lista Destinos</title>
6      <link type="text/css" href="{% static 'styles/lista_styles.css' %}" rel="stylesheet"></link>
7  </head>
8  <body>
9      <div class="content">
10         {% if mensaje %}
11         <div class="mensaje">
12             <p> Destino eliminado exitosamente</p>
13         </div>
14         {% endif %}
15         <table class="tabla">
16             <thead>
17                 <tr>
18                     <th scope="col">ID</th>
19                     <th scope="col">Nombre</th>
20                     <th scope="col">Descripcion</th>
21                     <th scope="col">Precio</th>
22                     <th scope="col">Oferta</th>
23                     <th scope="col">Editar</th>
24                     <th scope="col">Eliminar</th>
25                 </tr>
26             </thead>
27             <tbody>
```

Figure 16: Estructura HTML de enlistar 1

```
27         {% for destino in destinos %}
28         <tr>
29             <th scope="row"> {{destino.id}} </th>
30             <td {{ destino.name }} </td>
31             <td {{ destino.desc }} </td>
32             <td {{ destino.price }} </td>
33             <td {{ destino.offer }} </td>
34             <td class="boton amarillo"><a href="{% url 'editDest' destino.id %}"> Editar </a></td>
35             <td class="boton rojo"><a href="{% url 'delDest' destino.id %}"> Eliminar </a></td>
36         </tr>
37         {% endfor %}
38     </tbody>
39 </table>
40 </div>
41 </body>
42 </html>
```

Figure 17: Estructura HTML de enlistar 2

## Templates para modificar

La estructura que utilizamos para presentar la opción de modificar será mediante un formulario que contendrá los datos ya instanciados de una fila específica, permitiendo modificar los datos correspondientes. Para esto, utilizamos Crispy Forms, que facilita la creación de formularios con configuraciones predefinidas y una presentación coherente.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Editar Destinos</title>
5      <link href="https://stackpath.bootstrapcdn.com/bootstrap/5.1.3/css/bootstrap.min.css" rel="stylesheet">
6  </head>
7  <body>
8      {% load crispy_forms_tags %}
9      <div class="container mt-5">
10         <h1>Editar Destino</h1>
11         <form method="post" action="{% url 'actDest' destino.id %}">
12             {% csrf_token %}
13             {% crispy form %}
14             <button type="submit" class="btn btn-primary mt-3">Guardar</button>
15         </form>
16         <a href="{% url 'listaDest' %}" class="btn btn-secondary mt-3">Ver todos los Destinos</a>
17     </div>
18 </body>
19 </html>

```

Figure 18: Estructura HTML de modificar

```

34  INSTALLED_APPS = [
35      'django.contrib.admin',
36      'django.contrib.auth',
37      'django.contrib.contenttypes',
38      'django.contrib.sessions',
39      'django.contrib.messages',
40      'django.contrib.staticfiles',
41      'travello.apps.TravelloConfig',
42      'crispy_forms',
43      'crispy_bootstrap5',
44  ]
45  CRISPY_ALLOWED_TEMPLATE_PACKS = 'bootstrap5'
46  CRISPY_TEMPLATE_PACK = 'bootstrap5'

```

Figure 19: Crispy

