# UNIVERSIDAD NACIONAL DE GENERAL SARMIENTO

Nivel Superior

Trabajo Práctico Primer Semestre de

# Introducción a la Programación

---

"PROYECTO DE INTRODUCCIÓN A PROGRAMACIÓN – DJANGO
GALERÍA DE IMÁGENES DE LA NASA."

---


**Profesor:**

MARTINEZ, Nora - namartin@campus.ungs.edu.ar

WINOGRAD, Natalia

MONTIEL, Santiago

**Integrantes:**

DE LORENZI, Franco

VERA, Facundo

ZALAZAR FERNANDEZ, Florencia – gaby.zalazar04@gmail.com


**Comisión N°08**

*26 de junio de 2024*

# Contenido

# Introducción

El presente Trabajo Práctico Grupal de Introducción a la Programación para la Universidad Nacional General Sarmiento (UNGS), se centra en la implementación de funcionalidades importantes en una aplicación Web Full Stack ya creada, desarrollada con el Framework Django y utilizando el lenguaje de programación Python, HTML y CSS.

La siguiente aplicación desarrollada, es decir nuestra Página Web, permite a los usuarios consultar imágenes obtenidas desde una Interfaz de Programación de Aplicaciones (API) pública proporcionada por la NASA, presentando la información en tarjetas que muestran: la presente imagen, un título y una descripción.

Para lograr esto, fue necesario en un principio modificar una carpeta otorgada por un GitHub[1] de la propia UNGS donde ya algunas funciones estaban semidesarrolladas y otras había que implementarlas desde cero. Además, se incorporaron nuevas funcionalidades que mencionamos a lo largo del Proyecto.

El mismo está construido sobre una arquitectura multicapas, donde cada capa posee una única responsabilidad y se encuentra desacoplada del resto. Las capas son las siguientes: “DAO” (empleada para el alta/baja/modificación -CRUD/ABM- de objetos en una base de datos integrada, llamada SQLite). “Services” (usada para la lógica de negocio de la aplicación). Y “Transport” (utilizada para el consumo de la API en cuestión).

[1] GitHub es una forja para alojar proyectos utilizando el sistema de control de versiones Git.

# Proyecto de Introducción a Programación

## Galería de Imágenes de la NASA

A continuación, comenzaremos a explicar los cambios realizados a lo largo del código dado, como, por ejemplo, funciones utilizadas, qué fin cumplen las mismas, entre otras cosas.

### *Cambios "views.py" y "services_nasa_image_gallery.py"*

### Función getAllImages

Para reparar la funcionalidad principal de la aplicación, que consiste en obtener imágenes de una API pública, se modificó, en primer lugar, la función **`getAllImages`** ubicada en la ruta **nasa_image_gallery\layers\services\services_nasa_image_gallery.py**. Esta es la implementación de la función:

```python
def getAllImages(input=None):
    json_collection = transport.getAllImages(input)
    images = []

    for obj in json_collection:
        if 'data' in obj and 'links' in obj and 'description' in obj['data'][0]:
            nasa_card = mapper.fromRequestIntoNASACard(obj)
            images.append(nasa_card)

    return images
```

Función "getAllImages"
Obtenida en junio del año 2024

Esta función realiza los siguientes pasos:

Obtiene todas las imágenes llamando a la función **transport.getAllImages(input)**, que devuelve una lista de objetos y se guarda en la variable **json_collection**.

Filtra cada objeto en la lista, buscando aquellos que contengan los atributos **data**, **links** y **description** dentro de **data**. Transforma cada objeto filtrado en una tarjeta NASA utilizando la función **mapper.fromRequestIntoNASACard(obj)** y las guarda en la lista **images**.

Finalmente, la función devuelve la lista **images**. Para continuar fue necesario también modificar las funciones **getAllImagesAndFavouriteList** y home, ubicadas en **nasa_image_gallery\views.py**.

## Función getAllImagesAndFavouriteList

Empezando por la primera función mencionada cuya implementación es la siguiente:

```
def getAllImagesAndFavouriteList(request):
    images = services_nasa_image_gallery.getAllImages()
    favourite_list = services_nasa_image_gallery.getAllFavouritesByUser(request)

    return images, favourite_list
```

Función "getAllImagesAndFavouriteList"
Obtenida en junio del año 2024

Esta realiza los siguientes pasos:

Se llama a la función **getAllImages()** cuyo funcionamiento fue explicado anteriormente. Esta devuelve una lista de objetos, que contiene todas las imágenes obtenidas de la API y la guarda en la variable **images**. Obtiene otra lista de objetos que contiene todas las imágenes agregadas a favoritas por el usuario llamando a la función **getAllFavouritesByUser()** y la guarda en la variable **favourite_list.**

Finalmente, la función devuelve las dos listas: **images** y **favourite_list.**

## Función home

Por último, y para finalizar con la funcionalidad principal de la aplicación, que es obtener imágenes de una API pública y mostrarlas en pantalla, modificamos la función **home**, que se llama al ingresar a la URL **/home**. Inicialmente esta fue implementada de la siguiente manera. Posteriormente se agregan algunos cambios para implementar nuevas funcionalidades a la aplicación, como lo es un sistema de paginación:

```
def home(request):
    images, favourite_list = getAllImagesAndFavouriteList(request)
    return render(request, 'home.html', {'images': images, 'favourite_list': favourite_list})
```

Función "home"
Obtenida en junio del año 2024

Esta función sigue con los siguientes pasos:

Obtiene dos listas: **images** y **favourite_list**, llamando a la función definida anteriormente **getAllImagesAndFavouriteList()**. Renderiza el archivo HTML llamado "**home.html**", pasando las listas: **images** y **favourite_list** para que puedan ser usados en la vista.

De esta manera, la función principal del programa ha sido concluida, permitiendo la visualización de las imágenes provenientes de la API en la pantalla del usuario. A continuación, detallaremos algunas mejoras adicionales que se han incorporado a la aplicación.

## Inicio de sesión

Se añadió una funcionalidad de inicio de sesión, permitiendo a los usuarios autenticarse con un nombre de usuario y una contraseña previamente definidos en la base de datos. Una vez autenticados, los usuarios pueden agregar imágenes a una lista de favoritos. Para esta mejora, se implementó la siguiente función:

```python
def login_view(request):
    if request.method == "POST":
        username = request.POST["username"]
        password = request.POST["password"]
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return home(request)
        else:
            return render(request, "registration/login.html", { "error": "Usuario o contraseña incorrectos" })
    else:
        return render(request, "registration/login.html")
```

Función "login_view"
Obtenida en junio del año 2024

Esta función verifica si se está enviando un formulario mediante el método POST. Si no es así, renderiza la vista **login.html**, que muestra un formulario con dos campos de entrada: uno para el nombre de usuario y otro para la contraseña. Cuando se ingresan y envían los datos mediante el método POST, la función obtiene el nombre de usuario y la contraseña, almacenándolos en dos variables. Luego, crea la variable **user** y la define utilizando la función **authenticate** de Django. Esta función puede devolver un usuario si las credenciales son correctas, o **None** si no lo son. Si el usuario existe, la función lo autentica

llamando a la función **login** y redirecciona al usuario al **home**. Si el usuario no existe, vuelve a renderizar **login.html** pero pasando un mensaje de error.

## Alta de nuevos Usuarios

En el apartado anterior, se implementó un método de autenticación para un usuario ya definido en la base de datos. La siguiente mejora permitirá a los usuarios crear sus propias cuentas. Para ello, se definió la siguiente función:

```python
def signup_view(request):
    if request.method == 'POST':
        username = request.POST["username"]
        password = request.POST["password"]
        repeat_password = request.POST["repeat_password"]

        if password != repeat_password:
            return render(request, "registration/signup.html", { 'error': 'Las contraseñas deben coincidir' })

        user_already_exists = User.objects.filter(username=username).exists()

        if user_already_exists:
            return render(request, "registration/signup.html", { 'error': 'El nombre de usuario ya existe' })

        user = User.objects.create_user(username, None, password)
        user.save()

        return redirect('login')
    else:
        return render(request, "registration/signup.html")
```

Función "login_view"
Obtenida en junio del año 2024

Esta función verifica si se está enviando un formulario mediante el método POST. Si no es así, renderiza la vista signup.html, que muestra un formulario con tres campos de entrada: uno para el nombre de usuario, otro para la contraseña y otro para confirmar la contraseña. Cuando se ingresan y envían los datos mediante el método POST, la función obtiene el nombre de usuario, la contraseña y la contraseña repetida, almacenándolos en tres variables. Luego, verifica que ambas contraseñas sean iguales; de lo contrario, vuelve a renderizar el formulario con un mensaje de error. Después, verifica si el nombre de usuario ingresado ya existe en la base de datos. Si es así, nuevamente renderiza el formulario con un mensaje de error. Si todo es correcto, crea el nuevo usuario, lo guarda en la base de datos y redirige al usuario a la página de inicio de sesión para que pueda autenticarse con la nueva cuenta.

## Paginación

Esta función implementa un sistema de paginación para los resultados obtenidos de la API, mostrando 10 resultados por página y permitiendo al usuario seleccionar el número de página. La paginación se realiza mediante la siguiente función:

```python
def home(request):
    images, favourite_list = getAllImagesAndFavouriteList(request)

    page = int(request.GET.get('page', '1'))

    paginator = Paginator(images, Q_IMAGES_PER_PAGE)

    filtered_images = paginator.page(page)

    return render(request, 'home.html', {
        'images': filtered_images,
        'favourite_list': favourite_list,
        'page': page,
        'q_pages': range(1, paginator.num_pages + 1),
    })
```

Función "home"
Obtenida en junio del año 2024

Esta función, en primer lugar, obtiene todas las imágenes de la API y las añadidas a favoritos por el usuario. Luego, obtiene el número de la página actual y lo guarda en la variable page, en caso de no encontrar el parámetro, lo define como 1. Posteriormente, utiliza la clase Paginator de Django, que toma dos parámetros: los elementos que deben ser paginados y la cantidad de elementos que se mostrarán en cada página. Después, filtra las imágenes utilizando el método page de Paginator, pasando como parámetro el número de la página actual. Finalmente, renderiza la vista home.html, pasando como parámetros las imágenes filtradas, la lista de imágenes favoritas, la página actual y el número total de páginas.

## Favoritos

Estas tres funciones de **services_nasa_image_gallery.py** permiten a un usuario ya autenticado y al programa en sí agregar, obtener y eliminar una imagen a su lista de favoritos. La primera guardando la imagen en la base de datos, la segunda haciendo que

el programa recompile las imágenes de favoritos y la tercera eliminando las imágenes de su lista de favoritos:

```python
def saveFavourite(request):
    fav = mapper.fromTemplateIntoNASACard(request)  # Transformar request del template en NASACard
    fav.user = get_user(request)  # Asignar usuario correspondiente
    fav.comment = request.POST.get("comment", "")
    return repositories.saveFavourite(fav)  # Lo guardamos en la base.


# usados en el template 'favourites.html'
def getAllFavouritesByUser(request):
    user = get_user(request)
    favourites = repositories.getAllFavouritesByUser(user)
    return [mapper.fromRepositoryIntoNASACard(fav) for fav in favourites]


def deleteFavourite(request):
    favId = request.POST.get('id')
    return repositories.deleteFavourite(favId) # borramos un favorito por su ID.


def markImageAsUninteresting(request):
    image = mapper.fromTemplateIntoNASACard(request)
    image.user = get_user(request)
    return repositories.saveUninterestingImage(image)
```

Función "saveFavourite", “getAllFavouritesByUser”, “markImageAsUninteresting”
Obtenida en junio del año 2024

Sus funcionamientos son algo simples,

**saveFavourite** toma la solicitud POST del formulario de añadir a favoritos en home.html y convierte los datos del formulario en un objeto **NASACard** usando **mapper.fromTemplateIntoNASACard**, una vez hecho eso guarda el favorito dentro de los datos del usuario actual usando la función **repositories.saveFavourite**.

**getAllFavouritesByUser** solicita si hay un usuario autenticado desde **get_user**, si lo obtiene, solicita los datos de los mismos dentro de la base de datos llamando a **repositories.getAllFavouritesByUser** y utiliza **fromRepositoryIntoNASACard** para convertir los objetos obtenidos en objetos **NASACard**.

**deleteFavourite** recibe el ID del favorito a eliminar desde el **request**, luego utiliza la función **repositories.deleteFavourite** para eliminar el post favorito correspondiente de la base de datos.

```python
@login_required
def getAllFavouritesByUser(request):
    favourite_list = services_nasa_image_gallery.getAllFavouritesByUser(request)
    return render(request, 'favourites.html', {'favourite_list': favourite_list})


@login_required
def saveFavourite(request):
    if request.method == 'POST':
        services_nasa_image_gallery.saveFavourite(request)
    return redirect('favoritos')


@login_required
def deleteFavourite(request):
    if request.method == 'POST':
        services_nasa_image_gallery.deleteFavourite(request)
    return redirect('favoritos')
```

Función "saveFavourite", “getAllFavouritesByUser”, “markImageAsUninteresting”
Obtenida en junio del año 2024

Más allá de estas tres funciones tenemos su proyección dentro de **views.py**:

Primero que nada, es obligatorio que el usuario esté logueado para que se usen estas funciones. La función **getAllFavouritesByUser** solo llama a **services_nasa_image_gallery.getAllFavouritesByUser** para obtener la lista de favoritos y renderizar la pestaña de favoritos con la lista del usuario logueado.

Luego, tenemos **saveFavourite** que verifica si el método de solicitud es POST, si lo es, llama a **services_nasa_image_gallery.saveFavourite** para guardar el favorito y te redirige a la pestaña de favoritos para verificar su guardado de forma correcta.

Por último, tenemos a **deleteFavourite** que hace algo similar a **saveFavourite**, esta verifica si el método de solicitud es POST, y si esta lo es, llama a **services_nasa_image_gallery.deleteFavourite** para eliminar el post de favoritos.

Llegando al final tenemos los **repositories**. que, como ya vimos, estos son los encargados de varias acciones de las funciones anteriores, en este no se modificó mucho, solo se agregó a las funciones existentes la casilla de “comments” que se explicará más adelante junto con la explicación de los HTML favourites y home donde se agregaron algunas características más.

## Comentarios en Favoritos

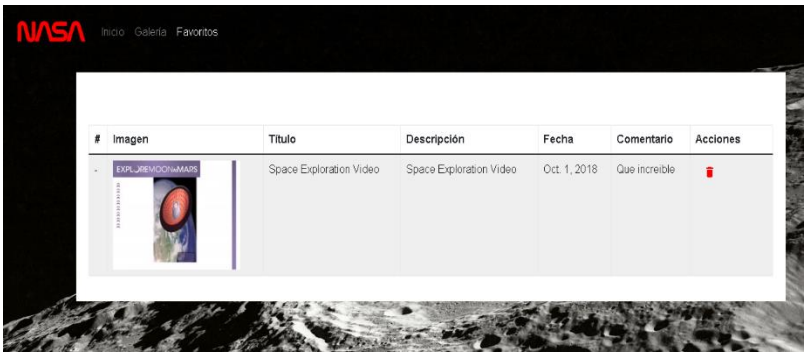
Aquí se añade el campo comment dentro de **models.py,repositories.py**, nasa_card.py y favourites.html para permitir a los usuarios agregar comentarios a las imágenes marcadas como favoritas (aqui una imagen de como se ve ya que en todas las funciones es básicamente lo mismo):

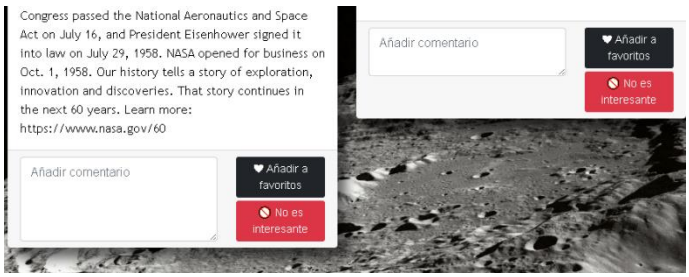
```python
class Favourite(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    image_url = models.TextField()
    date = models.DateField()
    comment = models.TextField(default='')
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE) # asociamos

    class Meta:
        unique_together = ('user', 'title', 'description', 'image_url', 'date', 'comment')
```

Función “Favourite”
Obtenida en junio del año 2024

Para cerrar el tema de Favoritos se realizaron cambios dentro de **favourites.html** y **home.html**, esto con el fin de que se proyecten el casillero de texto para agregar comentarios en el home(se agregó el **textArea** de comentarios en una línea nomás), la columna de comentarios en la pestaña favoritos(visto anteriormente), y un mínimo cambio al botón de añadir a favoritos para que se acomoden.




Muestra de cómo se ve en el sitio “Favoritos” y los botones y casilleros de texto en “home”
Obtenida en junio del año 2024

## Imágenes que no son Interesantes

Esta característica permite al usuario marcar cualquier post que desee como un post fuera de su interés, para lograr esto el conjunto de funciones que la componen realiza características diversas en relación a la proyección y guardado de posts.

```python
def markImageAsUninteresting(request):
    image = mapper.fromTemplateIntoNASACard(request)
    image.user = get_user(request)
    return repositories.saveUninterestingImage(image)


def getUninterestingImagesByUser(user):
    uninteresting_images = repositories.getUninterestingImagesByUser(user)
    return [mapper.fromRepositoryIntoNASACard(img) for img in uninteresting_images]
```

Función "markImageAsUninteresting", “getUninterestingImagesByUser”
Obtenida en junio del año 2024

Dentro de **services_nasa_image_gallery.py** encontramos las funciones **markImageAsUninteresting** y **getUninterestingImagesByUser** sus funcionamientos son similares a los de **saveFavourites** así que se repetirán explicaciones):

**markImageAsUninteresting** recibe un request de un usuario autenticado, a este lo convierte en una **NASACard** usando la función **fromTemplateIntoNASACard** dentro de **mapper**, luego obtiene el dato de si el usuario está autenticado con el **get_user** y almacena la imagen desinteresante en la base de datos del usuario llamando a **repositories.saveUninterestingImage**.

La función **getUninterestingImageByUser** obtiene todas las imágenes marcadas como no interesantes por un usuario autenticado, esto lo logra llamando a **repositories.getUninterestingImagesByUser** para obtener las imágenes desde la base de datos, luego, utilizando **fromRepositoryIntoNASACard** que está en **mapper** convierte las imágenes en objetos NASACard.

Aquí pasamos a explicar su funcionamiento dentro de **views.py**:

```python
@login_required
def markUninteresting(request):
    if request.method == 'POST':
        services_nasa_image_gallery.markImageAsUninteresting(request)
    return redirect('home')
```

Función “markUninteresting”
Obtenida en junio del año 2024

Primero que nada, el usuario debe haber iniciado sesión para poder marcar una imagen como un post desinteresaste.

La función **markUninteresting** es funcionar como una “vista protegida” para el usuario, está verifica si el método de solicitud es POST, si lo es, llama a **services_nasa_image_gallery.markUninteresting** para guardar el post como no interesante y te redirige a la pestaña de home eliminando la imagen de tus resultados.

Dentro de **repositories.py** se encuentra **saveUninterestingImage** la cual se usa para guardar la imagen como no interesante dentro de la base de datos y **getUninterestingImageByUser**, que esta obtiene las imágenes marcadas como no interesantes por el usuario desde la base de datos y retorna una lista con los posts marcados como no interesantes:

```python
def saveUninterestingImage(image):
    try:
        uninteresting_image = UninterestingImage.objects.create(title=image.title,
        description=image.description, image_url=image.image_url, date=image.date, user=image.user)
        return uninteresting_image
    except Exception as e:
        print(f"Error al guardar la imagen no interesante: {e}")
        return None

def getUninterestingImagesByUser(user):
    uninteresting_images = UninterestingImage.objects.filter(user=user).values('id', 'title', 'description', 'image_url', 'date')
    return list(uninteresting_images)
```

Función "saveUninterestingImage", “getUninterestingImagesByUser”
Obtenida en junio del año 2024

Finalizando con las imágenes de desinterés tenemos **UninterestingImage** dentro de **models.py**, este modelo define la estructura de una imagen marcada como “no interesante” en la base de datos:

Esto lo hace(como todo lo que ya vimos) replicando el sistema de favoritos, define los campos para los datos de la imágen, Usa **ForeignKey** para asociar las imágenes marcadas como no interesantes a el usuario logueado y define que cada combinación de datos de la imagen debe ser única para cada usuario.

```python
class UninterestingImage(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    image_url = models.TextField()
    date = models.DateField()
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)

    class Meta:
        unique_together = ('user', 'title', 'description', 'image_url', 'date')
```

Función "UninterestingImage"
Obtenida en junio del año 2024

Cerrando con las Imágenes no Interesantes agregamos el botón de “No es Interesante” a **home.html** así se proyecta en cada post que se vea:

```html
<form method="post" action="{% url 'marcar-no-interesante' %}">
    {% csrf_token %}
    <input type="hidden" name="title" value="{{ imagen.title }}">
    <input type="hidden" name="description" value="{{ imagen.description }}">
    <input type="hidden" name="image_url" value="{{ imagen.image_url }}">
    <input type="hidden" name="date" value="{{ imagen.date }}">
    <button type="submit" class="btn btn-danger btn-sm w-100" style="color: white"> No es interesante</button>
</form>
{% endif %}
```

Plantilla del “home.html” sobre el botón “No es interesante”
Obtenida en junio del año 2024

# Conclusión

En este trabajo, hemos desarrollado una aplicación que permite a los usuarios añadir a favoritos distintas imágenes provenientes de una API. Además, se agregaron múltiples funcionalidades para mejorar la experiencia del usuario. Para ello, fue necesario utilizar nuestro conocimiento adquirido sobre Python durante la cursada, además de documentarnos sobre tecnologías que, hasta este momento, eran desconocidas para nosotros, como Django. Otro desafío que se presentó durante el transcurso de este trabajo práctico fue trabajar sobre un código ajeno, lo que nos obligó a indagar para conocer su estructura y funcionamiento. Asimismo, destacamos que, a diferencia de los ejercicios y problemas resueltos durante la cursada, este trabajo implicó la creación de una aplicación web completa, lo que nos ayudó a aplicar mejor nuestros conocimientos en un entorno más complejo y realista, trabajando en conjunto con otras personas.