Informe: Galería de imágenes de la NASA

Alumnos: Tatarin Leonardo - Tomás Lavandeira

DNI: 34228745 - 45150322

Comisión: 07 - Grupo: 4

Docentes: Sergio Santa Cruz - Nazareno Avalos - Nahuel Sauma

Materia: Introducción a la Programación

Fecha: 28/06/2024

Informe: Galería de imágenes de la NASA

Grupo: 4
Integrantes: Tatarin Leonardo - Tomás Lavandeira

Introducción

El primer día del trabajo nos dedicamos a instalar todo y hacer el fork, en este punto hubo problemas menores como no marcar “añadir al path” cuando instalamos python, pero sin mayores complicaciones.
Otro de los problemas fue todo lo que tiene que ver con introducciones, aunque está el video, la introducción a Github fue algo complicada ya que, en un principio, no sabía usar bien los comandos y subí mis commits como si se tratara de google drive, haciendo que en vez de subir el commit, se subiera el archivo entero.

Funcionalidades obligatorias

Una vez hecho el fork fue abrumador ver la cantidad de funciones hechas y las que estaban por medio hacer así que costó mucho no marearse a la hora de empezar. Más tranquilos nos fijamos en el archivo **“views.py”** para ver qué es lo que faltaba completar. Para completar la función principal de la galería lo que se hizo fue llamar a la función **“getAlllmagesAndFavouriteList”** para definir las variables **“images”** y **“favourite_list”** y así poder retornar las imágenes, junto a la lista de favoritos (aunque está se implementó correctamente después). Pero para correr **“getAlllmagesAndFavouriteList”** faltaba terminar la función **“getAlllmages(input=None)”** para poder obtener todas las imágenes. Para eso fuimos al archivo **“services_nasa_image_gallery.py”** y completamos la función antes mencionada. Para eso hizo falta usar 2 funciones implementadas: **“getAlllmages(input)”** de transport y **“fromRequestIntoNASACard”** de mapper.
La primera función sirvió para definir la lista **“json_collection”** y llenarla con todas las imágenes. Luego se pasó por un ciclo **for** por elemento y por cada iteración se añadió **“images”** todos los elementos transformados en NASAcard de la lista **“json_collection”.**
El mayor inconveniente que hubo a la hora de realizar esto es que la primera vez que se completó la función **“getAlllmages(input=None)”** no andaba porque, en teoría, no existía el archivo **“transport”** por lo tanto, nunca se importaba la función que llamaba a todas las imágenes. Se soluciono volviendo a hacer un fork.

views.py

```python
# función principal de la galería.
def home(request):
    # llama a la función auxiliar getAllImagesAndFavouriteList() y obtiene 2 listados: uno de las imág
    # (*) este último, solo si se desarrolló el opcional de favoritos; caso contrario, será un listado
    page = request.GET.get('page', '')
    limit = request.GET.get('limit', '')

    if (page and limit):
        images, favourite_list = getAllImagesAndFavouriteList(request, page, limit)
        return render(request, 'page.html', {'images': images, 'favourite_list': favourite_list})
    else:
        return render(request, 'home.html', {'current_page': 'home'} )
```

layers/services/services_nasa_image_gallery.py

```python
def getAllImages(input=None):
    # obtiene un listado de imágenes desde transport.py y lo guarda en un json_collection.
    # ¡OJO! el parámetro 'input' indica si se debe buscar por un valor introducido en el buscado
    json_collection = transport.getAllImages(input)

    images = []
    for element in json_collection:
        images.append(mapper.fromRequestIntoNASACard(element))
    # recorre el listado de objetos JSON, lo transforma en una NASACard y lo agrega en el listad

    return images
```

Buscador

Esta utiliza la función **“search”** en el archivo **“views.py”** a la cual le llega un **“request”** (una búsqueda en página), comprueba que sea una palabra y busca todas las imágenes relacionadas con la función **“getAllImagesAndFavouriteList”,** tomando como parámetro la palabra ingresada. En caso de que no se escriba nada simplemente se vuelve a la pantalla de inicio.

views.py

```python
# función utilizada en el buscador.
def search(request):
    # si el usuario no ingresó texto alguno, debe refrescar la página; caso contrario, debe filtrar aque
    page = request.GET.get('page', '')
    limit = request.GET.get('limit', '')
    search_msg = request.POST.get('query', '')

    if search_msg != '':
        if (page and limit):
            images, favourite_list = getAllImagesAndFavouriteList(request, page, limit, search_msg)
            return render(request, 'page.html', {'images': images, 'favourite_list': favourite_list} )
        else:
            return render(request, 'home.html', {'current_page': 'home', 'search_msg': search_msg} )
    else:
        return redirect('home')
```

Iniciar sesión

Por otra parte, está la opción de iniciar sesión que usa la función **"login_page"** que se encuentra en la carpeta nasa image gallery. Está función verifica los datos para iniciar tu sesión y se fijan que sean de una cuenta registrada. Si son credenciales registradas podés acceder a las funciones antes mencionadas, caso contrario simplemente te vuelve a la pantalla de inicio.

templates/registration/login.html

```python
def login_page(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 redirect('home')
        else:
            messages.error(request, 'Usuario o contraseña incorrectos')

    return render(request, 'registration/login.html', {'current_page': 'login'})
```

Favoritos

Para lograr la función de **"favoritos"** se completó la función **"saveFavourite",** consistió en definir la variable **"fav"** con la función **"fromTemplateIntoNASACard"** para transformar la información (ya en el template) en una **NASACard**. Luego se definió **fav.user** con la función **"get_user"** para asignarlo al usuario correspondiente. Por otro lado, para obtener todas las imágenes favoritas del usuario está la función **"getAllFavouritesByUser"** que verifica que el que la pide es un usuario registrado para después llamar a todas sus imágenes favoritas, meterlas en una lista, recorrerla y "mapear" los elementos a **NASACard**. Por último, está la

función **“deleteFavourite”** que identifica el **“id”** de la imagen para poder borrarla de favoritos.

views.py

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


@login_required
def saveFavourite(request):
    services_nasa_image_gallery.saveFavourite(request)
    return redirect('home')

@login_required
def deleteFavourite(request):
    services_nasa_image_gallery.deleteFavourite(request)
    return redirect('favoritos')
```

layers/services/services_nasa_image_gallery.py

```python
def saveFavourite(request):
    fav = mapper.fromTemplateIntoNASACard(request) # transformamos un request del template
    fav.user = get_user(request) # le seteamos el usuario correspondiente.

    return repositories.saveFavourite(fav) # lo guardamos en la base.


# usados en el template 'favourites.html'
def getAllFavouritesByUser(request):
    if not request.user.is_authenticated:
        return []
    else:
        user = get_user(request)

        # buscamos desde el repositorio TODOS los favoritos del usuario (variable 'user').
        favourite_list = repositories.getAllFavouritesByUser(user)
        mapped_favourites = []

        for favourite in favourite_list:
            # transformamos cada favorito en una NASACard, y lo almacenamos en nasa_card.
            nasa_card = mapper.fromRepositoryIntoNASACard(favourite)
            mapped_favourites.append(nasa_card)

        return mapped_favourites


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

Registro de usuarios nuevos

Para la función **“Registrarse”** hubo que hacer un nuevo archivo html llamado **“register.html”** que se encuentra en la carpeta nasa image gallery → templates →

registration que tiene todos los códigos los cuales piden nombre, apellido, usuario, email y contraseña, todos son requisitos obligatorios para crearse la cuenta. También fue agregado a la interfaz de la página la opción **"Registrarse".** También se hicieron cambios en el **"header.html"** para que no de errores. De hecho, esto fue la parte más difícil, siempre que intentábamos hacer algo con una cuenta registrada daba error, entonces tuvimos que cambiar todos los errores que nos marcaba la página uno por uno, porque los errores saltaban de uno en uno. Las cuentas se guardan y se pueden usar las credenciales que usaste para registrarte.

Se agregó la función para mandar un mail cuando un usuario se registra. Para lograr esto fue necesario definir una nueva función llamada **"send"** dentro **"services_send_mail"** la cual toma como parámetro un mail recibido y usuario por la misma página. Funciona cargando los datos del mail que envía estos mensajes, el mensaje cargado entre variables en la cual se dice el asunto del mail y el mensaje y, por último, envía el mensaje al mail cargado por el parámetro. Para que esta función se pueda utilizar fue necesario **import "send_mail"** de **"django.core.mail"**. Una vez definida esta función fue agregada al archivo **"views.py"** en la cual se importa la función **"send"** y es agregada en la función **"register"** en el cual, si se cumplen todas las condiciones para que se cree un usuario, le manda un mail avisando que la cuenta fue creada de manera exitosa.

templates/registration/register.html

```
{% extends 'header.html' %} {% block content %}
<div class="register-form" style="text-align: center;">
    <form action="" method="POST" style="display: inline-block;">
        {% csrf_token %}
        <h2 class="text-center">Registro</h2>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="text" name="first_name" id="first_name" class="form-control" placeholder="Nombre" required="required">
        </div>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="text" name="last_name" id="last_name" class="form-control" placeholder="Apellido" required="required">
        </div>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="text" name="username" id="username" class="form-control" placeholder="Usuario" required="required">
        </div>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="email" name="email" id="email" class="form-control" placeholder="Email" required="required">
        </div>
        <div class="form-group" style="margin-bottom: 5%;">
            <input type="password" name="password" id="password" class="form-control" placeholder="Contraseña" required="required">
        </div>
        {% if messages %}
        <ul style="margin-bottom: 5%;">
            {% for message in messages %}
                <li>{{ message }}</li>
            {% endfor %}
        </ul>
        {% endif %}
        <div class="form-group">
            <button type="submit" class="btn btn-primary btn-block">Registrarse</button>
        </div>
    </form>

</div>
{% endblock %}
```

views.py

```python
def register(request):
    if request.method == 'POST':
        first_name = request.POST['first_name']
        last_name = request.POST['last_name']
        username = request.POST['username']
        email = request.POST['email']
        password = request.POST['password']

        if User.objects.filter(username=username).exists():
            messages.error(request, 'Usuario en uso')
        elif User.objects.filter(email=email).exists():
            messages.error(request, 'Email en uso')
        else:
            user = User.objects.create_user(first_name=first_name, last_name=last_name, username=username, email=email, password=password)
            user.save()
            login(request, user)
            services_send_mail.send(email, username)
            return redirect('home')

    return render(request, 'registration/register.html', {'current_page': 'register'})
```

layers/services/services_send_mail.py

```python
from django.core.mail import send_mail
from django.conf import settings

def send(email, username):
    subject = "Verificación de cuenta"
    body = "Hola " + username + ", usted ha creado una cuenta para la galería de la nasa"

    send_mail(subject, body, settings.EMAIL_HOST_USER, [email], fail_silently=False)
```

Agregar comentarios en imágenes guardadas en favoritos

En la página también se puede agregar comentarios a las imágenes que marcas como favoritas. Para eso hubo que cambiar la función **“saveFavourite”** para que dejara guardar los comentarios, entonces el en código se agregó **“comment=image.comment”** y en la función **“getAllFavouritesByUser”** también se cargó la variable **“comment”** para que sea mostrada. Luego, en el mapper, en **“nasa_card.py”**, **“models.py”** y **“favorite.html”** fue agregada la variable **“comment”** en todas las funciones. También se agregó **“0003_favourite_comment.py”** que se encarga de modificar la base para agregar el campo **comment**, utilizando los comandos **python manage.py makemigrations** y **python manage.py migrate.**

layers/generic/nasa_card.py

```python
class NASACard:
    def __init__(self, title, description, comment, image_url, date, user=None, id=None):
        self.title = title
        self.description = description
        self.comment = comment
        self.image_url = image_url
        self.date = date
        self.user = user
        self.id = id
```

models.py

```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 el favorito c

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

migrations/0003_favourite_comment.py

```python
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('nasa_image_gallery', '0002_alter_favourite_unique_together'),
    ]

    operations = [
        migrations.AddField(
            model_name='favourite',
            name='comment',
            field=models.TextField(default=''),
        ),
    ]
```

layers/dao/repositories.py

```python
from nasa_image_gallery.models import Favourite

def saveFavourite(image):
    try:
        fav = Favourite.objects.create(title=image.title, description=image.description, comment=image.comment,
                                       image_url=image.image_url, date=image.date, user=image.user)
        return fav
    except Exception as e:
        print(f"Error al guardar el favorito: {e}")
        return None

def getAllFavouritesByUser(user):
    favouriteList = Favourite.objects.filter(user=user).values('id', 'title', 'description', 'comment', 'image_url', 'date')
    return list(favouriteList)
```

static/scripts.js

```javascript
$('.home').on('click', '.saveFavourite', function() {
    let comment = prompt('Ingrese un comentario para la imagen: ');

    if (comment !== null) {
        let form = $(this).closest('form');
        form.find('.comment').val(comment);

        let data = form.serialize();
        $.post(form.attr('action'), data, function(){
            form.find('.saveFavourite').hide();
            form.find('.isFavourite').show();
        });
    }
});
```

Infinite scroll

Para que la pagina tuviera infinite scroll (bajar sin necesidad de pasar a otra página como lo hacen facebook o instagram) hubo que rehacer la función **“getAllImages”** que nos devolvía todas las imágenes que tuvieran la palabra que pusiéramos en el buscador, se cambió por **“getPaginatedImages”** el cual llama a json_collection al cual se le asignan todas las imágenes, como anteriormente se mencionó, a través de **“transport.getAllImages”** y se cargan las variables **page**, para saber en qué página arrancar y **limit**, hasta cuál cargar. Seguido está la variable **start** y **end**, las cuales van a ser usadas como parámetros para saber cuándo tiene que empezar el recorrido de la lista json_collection y cuando va a terminar. Para recorrerla se usó un ciclo **for** recorriéndose por el índice, se agregó una condición para que se recorra mientras el índice sea mayor o igual al parámetro **“start”** y sea menor que **“end”** se va a agregar a la lista **“images”** las imágenes solicitadas siendo transformadas antes en una **NASACard**.
Se agregó **“page.html”,** para mostrar cada imagen con las opciones antes dichas (por ejemplo, añadir a favoritos, esta imagen ya está en favoritos).
Se agregó la opción de seleccionar cuantas imágenes querés que cargue la página por línea. Para ello se modificó el archivo **“home.html”** para que pudiera reconocer la opción y que te deje elegir entre 3,6,9 o 12 para las imágenes cargadas. Después se modificó **“views.py”** para que tomara el **“page”** y el **“limit”,** dicho de otra manera, se modificó para que el programa identifique cual es la página elegida, la configuración que tiene y el límite

establecido de carga. Para eso en la función **“getAllImagesAndFavouriteList”** se agregaron los parámetros **“page”**, **“limit”** y se modificó la función **“getPaginatedImages”** la cual sigue la misma lógica que la función antes explicada **“getAllImages”** pero tomando los parámetros de **“page”** y **“limit”** para tener la configuración ya cargada.
Se sacó la función **“getAllImagesAndFavouriteList”** de **“home”** para poner **“page”** y **“limit”,** en este caso, para que el usuario pueda elegir la configuración que más desee.
Seguido hay un condicional que toma como parámetros lo **“page”** y **“limit”** sirve para cargar las imágenes y los favoritos con la función **“getAllImagesAndFavouriteList”,** si se cumplen todas las condiciones devuelve las imágenes encontradas y las imágenes favoritas, si no, simplemente muestra la pantalla de inicio para q se empiecen a cargar las imágenes.
Se modificó también la función **“search”** (utilizada para que el buscador) que toma como nuevos datos **“page”** y **“limit”.** Sigue la misma lógica antes explicada, nada más que luego de la comprobación de que el usuario ingresó una palabra sigue un condicional para tener en cuenta **“page”** y **“limit”** y retornar todo con la configuración establecida.
Mediante js se chequea si el scroll de la pagina llego al final y cuando este ocurre se hace una llamada AJAX para traer las images que corresponden tomando en cuenta **“page”** y **“limit”**. Se agregaron variables de chequeo como **“isLoading”** o **“endPagination”** para evitar hacer llamadas innecesarias.
También se modificó la llamada para agregar imagen a favoritos para realizarla mediante AJAX así no es necesario recargar la página cada vez que se agrega uno.

layers/services/services_nasa_image_gallery.py

```python
def getPaginatedImages(page, limit, input=None):
    json_collection = transport.getAllImages(input)

    page = int(page)
    limit = int(limit)
    start = (page-1) * limit
    end = page * limit
    images = []
    for i in range(len(json_collection)):
        if i >= start and i < end:
            images.append(mapper.fromRequestIntoNASACard(json_collection[i]))

    return images
```

views.py

```python
def home(request):
    # llama a la función auxiliar getAllImagesAndFavouriteList() y obtiene 2 listados: uno de las imágen
    # (*) este último, solo si se desarrolló el opcional de favoritos; caso contrario, será un listado v
    page = request.GET.get('page', '')
    limit = request.GET.get('limit', '')

    if (page and limit):
        images, favourite_list = getAllImagesAndFavouriteList(request, page, limit)
        return render(request, 'page.html', {'images': images, 'favourite_list': favourite_list})
    else:
        return render(request, 'home.html', {'current_page': 'home'} )

# función utilizada en el buscador.
def search(request):
    # si el usuario no ingresó texto alguno, debe refrescar la página; caso contrario, debe filtrar aque
    page = request.GET.get('page', '')
    limit = request.GET.get('limit', '')
    search_msg = request.POST.get('query', '')

    if search_msg != '':
        if (page and limit):
            images, favourite_list = getAllImagesAndFavouriteList(request, page, limit, search_msg)
            return render(request, 'page.html', {'images': images, 'favourite_list': favourite_list} )
        else:
            return render(request, 'home.html', {'current_page': 'home', 'search_msg': search_msg} )
    else:
        return redirect('home')
```

static/scripts.py

```
var page = 1;
var limit = 3;
var endPagination = false;
var isLoading = false;

$(function() {
    limit = $('.imagesByPage').val();
    $('.imagesByPage').change(function() {
        limit = $(this).val();
    })

    if ($('.home').length) {
        $(window).scroll(function() {
            if (($(window).innerHeight() + $(window).scrollTop()) >= $('body').height()-10) {
                if (!isLoading && !endPagination) {
                    page += 1;
                    showPage(page, limit);
                }
            }
        });

        showPage(page, limit);
    }
});

function showPage(page, limit) {
    let query = $('#search').val() ? $('#search').val() : '';
    isLoading = true;
    $('.loadingSpinner').show();

    let data = {};
    if (query) {
        data = {query, csrfmiddlewaretoken: $('[name="csrfmiddlewaretoken"]').val()};
    }

    $.ajax(window.location.pathname + '?page=' + page + '&limit=' + limit,
        {
            method: query ? 'POST' : 'GET',
            data,
            success: function(r) {
                isLoading = false;
                $('.loadingSpinner').hide();

                if (r.length) {
                    $(r).insertBefore('.loadingSpinner');
                } else {
                    endPagination = true;
                }
            }
        }
    );
}
```

templates/page.html

```
{% for imagen in images %}
<div class="col">
    <div class="card">
        <div class="container-card-img" style="background-image: url('{{ imagen.image_url}}')">
        </div>
        <div class="card-body">
            <h5 class="card-title">{{ imagen.title}}</h5>
            <p class="card-text">{{ imagen.description}}</p>
        </div>
        {% if request.user.is_authenticated %}
        <div class="card-footer text-center">
            <form method="post" action="{% url 'agregar-favorito' %}">
                {% csrf_token %}
                <input type="hidden" name="title" value="{{ imagen.title }}">
                <input type="hidden" name="description" value="{{ imagen.description }}">
                <input type="hidden" name="comment" class="comment">
                <input type="hidden" name="image_url" value="{{ imagen.image_url }}">
                <input type="hidden" name="date" value="{{ imagen.date }}">
                <button type="submit" class="btn btn-primary btn-sm float-left isFavourite" style="color:white; display:
                {% if imagen in favourite_list %}inline-block{% else %}none{% endif %}" disabled>✔ Ya está añadida a favoritos</button>
                {% if imagen not in favourite_list %}
                    <button type="button" class="btn btn-primary btn-sm float-left saveFavourite" style="color:white">❤ Añadir a favoritos
                {% endif %}
            </form>
        </div>
        {% endif %}
    </div>
</div>
{% endfor %}
```

templates/home.html

```
{% extends 'header.html' %} {% block content %}
<main>
    <h1 class="text-center">Galería de Imágenes de la NASA</h1>
    <div class="d-flex justify-content-center" style="margin-bottom: 1%">
        <!-- Buscador del sitio -->
        <form class="d-flex" action="{% url 'buscar' %}" method="POST">
            {% csrf_token %}
            <input class="form-control me-2" id="search" type="search" value="{{ search_msg }}" name="query"
            <button class="btn btn-outline-success" type="submit">Buscar</button>
        </form>
    </div>
    <div class="d-flex justify-content-end" style="margin-bottom: 1%">
        <div class="form-group row col-md-6">
            <label class="col-form-label col-md-11" style="text-align: right;">Imágenes por página: </label>
            <select class="form-control imagesByPage">
                <option value="3">3</option>
                <option value="6">6</option>
                <option value="9">9</option>
                <option value="12">12</option>
            </select>
        </div>
    </div>

    <div class="row row-cols-1 row-cols-md-3 g-4 home">
        <div class="loadingSpinner"></div>
    </div>
</main>
{% endblock %}
```

Loading spinner

Se agregó una animación de loading en css mientras cargan las imágenes de la galería, para esto se modificó **“home.html”** donde se utilizó el código ** ** para que la pantalla de inició lo reconociera y se agregó en archivo **“scripts.js”** en la función **“showPage”** para mostrarlo cuando esté cargando, en

la función se pregunta si la página está cargando, si nos devuelve el valor **“True”** se muestra la imagen de carga.

static/scripts.js

```
function showPage(page, limit) {
    let query = $('#search').val() ? $('#search').val() : '';
    isLoading = true;
    $('.loadingSpinner').show();

    let data = {};
    if (query) {
        data = {query, csrfmiddlewaretoken: $('[name="csrfmiddlewaretoken"]').val()};
    }

    $.ajax(window.location.pathname + '?page=' + page + '&limit=' + limit,
        {
            method: query ? 'POST' : 'GET',
            data,
            success: function(r) {
                isLoading = false;
                $('.loadingSpinner').hide();

                if (r.length) {
                    $(r).insertBefore('.loadingSpinner');
                } else {
                    endPagination = true;
                }
            }
        }
    );
}
```

templates/home.html

```
{% extends 'header.html' %} {% block content %}
<main>
    <h1 class="text-center">Galería de Imágenes de la NASA</h1>
    <div class="d-flex justify-content-center" style="margin-bottom: 1%">
        <!-- Buscador del sitio -->
        <form class="d-flex" action="{% url 'buscar' %}" method="POST">
            {% csrf_token %}
            <input class="form-control me-2" id="search" type="search" value="{{ search_msg }}" name="query"
            <button class="btn btn-outline-success" type="submit">Buscar</button>
        </form>
    </div>
    <div class="d-flex justify-content-end" style="margin-bottom: 1%">
        <div class="form-group row col-md-6">
            <label class="col-form-label col-md-11" style="text-align: right;">Imágenes por página: </label>
            <select class="form-control imagesByPage">
                <option value="3">3</option>
                <option value="6">6</option>
                <option value="9">9</option>
                <option value="12">12</option>
            </select>
        </div>
    </div>

    <div class="row row-cols-1 row-cols-md-3 g-4 home">
        <div class="loadingSpinner"></div>
    </div>
</main>
{% endblock %}
```

static/styles.css

```
.loadingSpinner {
    border: 16px solid #f3f3f3; /* Light grey */
    border-top: 16px solid #555;
    border-radius: 50%;
    width: 120px;
    height: 120px;
    animation: spin 1s linear infinite;
    left: -60px;
    margin-left: 50%;
    position: relative;
  }

  @keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
  }
```

Extra: Favicon

Se agregó un icono para los favoritos llamado **“favicon”**. Para agregarlo se tuvo que cambiar el archivo **“header.html”** para que reconozca el nuevo archivo y lo cargue en la página.

templates/header.html

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi primera aplicación con Django</title>
    {% load bootstrap5 %} {% bootstrap_css %} {% bootstrap_javascript %} {% load static %} {% load custom_tags %}
    <link rel="icon" href="{% static 'favicon.ico' %}" type="image/x-icon"/>
    <link rel="stylesheet" type="text/css" href="{% static 'styles.css' %}">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
</head>
```

Extra: Destacar sección actual

Se implementó un **simple_tag** llamado **check_current_page**, para para detectar cual es la página actual y así poder agregar una clase en el ítem del menú y destacarlo en negrita. Para ello cada función de **views.py** define una variable **current_page** si es necesario para luego chequearla en **header.html**

templates/header.html

```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
            <a class="nav-link {% check_current_page 'index' %}" href="{% url 'index-page' %}">Inicio</a>
        </li>
        <li class="nav-item">
            <a class="nav-link {% check_current_page 'home' %}" href="{% url 'home' %}">Galería</a>
        </li>
        {% if request.user.is_authenticated %}
        <li class="nav-item">
            <a class="nav-link {% check_current_page 'favourites' %}" href="{% url 'favoritos' %}">Favoritos</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{% url 'exit' %}">Salir</a>
        </li>
        <li class="nav-item">
            <span class="username">Hola, {{ request.user.username }}</span>
        </li>
        {% endif %}
        {% if not request.user.is_authenticated %}
        <li class="nav-item">
            <a class="nav-link {% check_current_page 'login' %}" href="{% url 'login' %}">Iniciar sesión</a>
        </li>
        <li class="nav-item">
            <a class="nav-link {% check_current_page 'register' %}" href="{% url 'register' %}">Registrarse</a>
        </li>
        {% endif %}
    </ul>
</div>
```

templatetags/custom_tags.py

```python
from django import template

register = template.Library()

@register.simple_tag(takes_context=True)
def check_current_page(context, value):
    if context['current_page'] == value:
        return 'active'
```

Extra: Mejora visual para acomodar el alto de imágenes

Se implementó una mejora visual para que todas las imágenes tengan el mismo alto y se adapten al contenedor. Para ello se colocó la imagen como fondo de un div con un alto fijo, para que la imagen se adapte a ese tamaño sin deformarse.

templates/page.html

```
{% for imagen in images %}
<div class="col">
    <div class="card">
        <div class="container-card-img" style="background-image: url('{{ imagen.image_url}}')">
        </div>
        <div class="card-body">
            <h5 class="card-title">{{ imagen.title}}</h5>
            <p class="card-text">{{ imagen.description}}</p>
        </div>
```

static/styles.css

```css
.container-card-img {
    width: 100%;
    height: 320px;
    background-repeat: no-repeat;
    background-position: center;
    background-size: contain;
}
```

Desafíos iniciales

La instalación y configuración del trabajo presentó algunos problemas. Al principio instalamos python sin agregarlo al path, lo que nos dio los errores para instalar varias de las herramientas que usamos. Por suerte nos dimos cuenta al inicio porque nos impedía instalar cosas necesarias para trabajar entonces no nos llevó mucho tiempo.
Una vez instalado todo, comprobamos la página y las distintas funciones. Luego, la introducción al trabajo con todas sus funciones, identificar los archivos, entender la estructura del proyecto, para qué sirve cada función, donde están, qué hay que modificar,

que podemos modificar y que no podemos nos llevó un tiempo. Una vez que nos centramos y entendimos las herramientas que teníamos a disposición el desarrollo fue próspero.

Complicaciones

La primera complicación se nos presentó a la hora de subir las cosas a Github. Primero, por un mal entendimiento de como subir archivos la tratamos como si fuera Google Drive y subimos el archivo en vez del commit. No pasó mucho antes de aprender a hacer bien los commit de la manera en la que se debe. Otro problema que se nos presentó en alguna ocasión fue que no identificaba la rama **"main"** del repositorio, lo arreglamos, pero quedó un commit que dice **"Merge brach main of https://github.com/ltatarin/ip-public-repo"**.

Otra complicación fue el envío de mails, la función fue desarrollada sin mayores problemas fuera del programa, los problemas comenzaron cuando la implementamos en el programa principal. En un principio, no reconocía que el archivo en el cual se hizo la función, no dejaba importarlo y siempre saltaban errores distintos. Al final decidimos crear un nuevo archivo en **services** llamado **services_send_mail.py** para toda la lógica de envió de emails.

Encontrar una animación para el **"loading"** también fue un problema sobre todo porque no encontrábamos una imagen con animación y que tenga fondo transparente para que no quede fea en el sitio ya que el fondo tiene un color. Finalmente decidimos ir por una implementación en css.

Detectar el fin del scroll en la página fue otra complicación, no encontrábamos una manera de encontrar cuando se llegaba al límite sin tener en cuenta el final de la página, ya que este iba a cambiar constantemente. Lo solucionamos con un script en js que chequea cada vez que hacemos scroll si ya llegamos al final de la página haciendo una comparación entre el alto total de la página y la posición de la barra de scroll.

Aprendizajes

Antes de este trabajo no habíamos trabajado con Django en profundidad, abordarlo e investigarlo nos hizo aprender que es una buena opción para programar páginas de internet ya que es muy personalizable, es cómodo de usar y tiene unas funciones fáciles de usar y entender.

La introducción de Github, como ya comentamos, fue difícil, pero una vez aprendimos a usar la plataforma es super comoda para el trabajo remoto grupal. Sus comandos son sencillos, la vinculación con Visual Studio Code es sencilla y muy práctica.

Aprendimos a usar CSS como idioma para el diseño de la página el cual también es bastante sencillo cuando y personalizable. Solamente hay que fijarse dónde empiezan y terminan las funciones.

El uso de Javascript, jQuery y Ajax para las funciones para implementar el scroll infinito y agregar favoritos fue efectivo para poder agregarle interacción a la página web.

La función que se encarga de enviar mails fue totalmente nueva para nosotros, importar las bibliotecas necesarias para desarrollar esta función fue todo un descubrimiento.

Aprender a manejar las **Tags** en los códigos HTML nos permitió desarrollar y entender nuevas formas de manejar los datos y funciones en estos códigos.

Manejar base de datos fue toda una novedad. Guardar usuarios, modificar las tablas para poder agregar campos y poder loguear un usuario para obtener sus datos fueron algunas de las cosas que aprendimos mientras usábamos SQL.

Conclusión

Este trabajo fue todo un desafío para nosotros, al principio nos costó arrancar, pero una vez que entendimos a qué nos enfrentábamos fue un trabajo divertido y muy educativo, aprendimos a usar herramientas nuevas y nos familiarizamos aún más con las que sabíamos usar. Fue una buena oportunidad para aprender a trabajar en equipo, dividiendo tareas y haciendo pruebas sobre lo que el otro desarrollaba para que todo funcione correctamente.