

Introducción a la Programación

Trabajo Practico: Galería de imágenes de la Nasa 🚀

Alumnos:

Nicolás Falzetta

Natalia Evangelista

Ezequiel Díaz

Comisión: 10

Docentes:

José Luis Mieres

Sergio Santa Cruz

Miguel Rodríguez

Introducción

En este proyecto, tenemos la intención de implementar una aplicación web fullstack utilizando el framework Django. La aplicación permitirá a los usuarios consultar y explorar imágenes obtenidas de la API pública de la NASA.

Funcionalidades claves de la aplicación:

Visualización de Imágenes: La aplicación realizará consultas a la API de la NASA y presentará las imágenes obtenidas en una serie de cards, la cual contendrá la imagen, un título y una breve descripción.

Buscador: Se implementará un buscador que permitirá a los usuarios filtrar las imágenes mostradas según sus intereses.

Autenticación: También incluirá un sistema de autenticación básica que permitirá a los usuarios iniciar sesión.

Favoritos: Los usuarios autenticados podrán marcar imágenes como favoritas. Estas imágenes serán almacenadas en la base de datos y estarán disponibles para consulta posterior.

Paginación: La funcionalidad de paginación se implementará para manejar y presentar un gran volumen de imágenes de manera eficiente. Los usuarios podrán seleccionar el número de página y la cantidad de elementos a mostrar en ella.

Lo que está implementado

Para la realización del proyecto se nos proporcionó en términos de estructura 4 HTML:

El header.html incluye la lógica para determinar si un usuario está logueado y muestra el nombre del usuario si está autenticado.

El home.html es la sección principal donde se muestran las imágenes de la API y se incluye un buscador, así como un algoritmo que recorre y dibuja cada objeto de la API en pantalla.

El footer.html contiene el pie de página, aunque no tiene acciones de código relevantes para el desarrollo.

El index.html es el contenedor principal que incluye los 3 templates anteriormente mencionados.

En el archivo views.py, se encuentran funciones clave para el funcionamiento de la aplicación. La función index_page(request) renderiza el contenido de index.html. La función home(request) obtiene todas las imágenes de la API y los favoritos del usuario mediante la función auxiliar getAllImagesAndFavouriteList(request), que devuelve dos listas: una con las imágenes de la API y otra con las imágenes marcadas como favoritas por el usuario. Esta función luego renderiza home.html y le pasa la información obtenida.

La lógica de servicio está contenida en transport.py, que tiene todo el código necesario para consumir la API de la NASA, gestionando las solicitudes a la API y la recepción de datos. En mapper.py, se incluye la lógica para convertir y mapear los resultados de la API en objetos NASA Card, que son los objetos utilizados en el template para dibujar los resultados en pantalla.

Funciones básicas requeridas

Éstas son las encargadas de hacer que las imágenes de la galería se muestren/rendericen.

getAllImages(input=None):

```
def getAllImages(input=None):

    json_collection = transport.getAllImages(input)# obtiene un listado de imágenes desde transport.py y lo guarda

    images = []#lista vacia donde se guardaran las NasaCard

    for object in json_collection: # recorre el listado de objetos del JSON
        try: #captura el error

            if not [object['data'][0]['title'] == ""
                    and object['data'][0]['description'] == "" and
                    object['links'][0]['href'] == "" and object['data'][0]['date_created'][:10]==""]:#validacion

                nasa_card=mapper.fromRequestIntoNASACard(object)#formatea al abjeto en una Nasa Card
                images.append(nasa_card)#agrega a la lista imagenes una nasa_card
        except KeyError :
            print("error objeto incompleto")

    return images #retorna la lista de nasa_card
```

La finalidad de esta función es retornar una lista de imágenes, las cuales tendrán una estructura de Nasa Card. Para ello, en la variable json_collection se asigna la función getAllImages que se encuentra en el archivo transport.py para obtener la información de la API de la Nasa. Una vez obtenida la misma, recorre cada objeto del json con la intención de convertirlo en Nasa Card a través de la función fromRequestIntoNasaCard, para luego agregarlo en la lista images. Fue necesario validar cada campo requerido, eso quiere decir que el objeto del json debía tener imagen, título, descripción y fecha antes de sumarlo a la lista, ya que en algunas de nuestras pruebas (buscando la palabra marte) aparecía un KeyError. El mismo indicaba que el campo de descripción no existía en el objeto.

getAllImagesAndFavouriteList:

```
# auxiliar: retorna 2 listados -> uno de las imágenes de la API y otro de los favoritos del usuario.
def getAllImagesAndFavouriteList(request):

    images = services_nasa_image_gallery.getAllImages(None)

    favourite_list = services_nasa_image_gallery.getAllFavouritesByUser(request)

    return images, favourite_list
```

El objetivo de esta función es retornar dos listas:

Images: contiene una lista de Nasa Card a través de la función getAllImages donde el parámetro none indica que no se ha realizado una petición de búsqueda y por lo tanto mostrará al valor determinado “space”.

Favourite_list: contiene una lista de Nasa Card de los favoritos del usuario a través de la función getAllFavouriteByUser siempre y cuando el mismo se encuentre autenticado, de no ser así mostrará una lista vacía.

def home (request):

```
def home(request):
    images, favourite_list = getAllImagesAndFavouriteList(request) # llama a la función auxiliar getAllImagesAndFavouriteList() y obtiene 2 listados:
    #uno de las imágenes de la API y otro de favoritos por usuario.Este Último, solo si se desarrolló el opcional de favoritos; caso contrario, será un listado vacío []

    page_number = request.GET.get('page', 1) #Obtiene el número de la página actual desde los parámetros de la URL (GET request).
    # Si no se proporciona, usa 1 por defecto.
    items_per_page = request.GET.get('items_per_page', 5) #Obtiene la cantidad de elementos por página, si no se determina, usa 5 por defecto.

    try:
        items_per_page = int(items_per_page) #convierte al items_per_page en un entero
        if items_per_page < 1: #Si el número es menor que 1, lo ajusta a 5
            items_per_page = 5
    except ValueError:
        items_per_page = 5 #Si `items_per_page` no es un número válido, lo ajusta a 5

    paginator = Paginator(images, items_per_page) #Crea una instancia de 'Paginator' con la lista de imágenes y la cantidad de elementos por página.
    #Paginator es una clase de Django que facilita la división de listas largas en páginas más manejables.
    page_obj = paginator.get_page(page_number) #Obtiene el objeto de la página actual usando el número de la página.

    return render(request, 'home.html', {
        'page_obj': page_obj, 'images': images,
        'favourite_list': favourite_list,
        'items_per_page': items_per_page,
    }) # Renderiza la plantilla `home.html` y pasa el contexto a la plantilla:
```

El objetivo de la función home es redirigir las listas de imágenes y favoritos a home.html utilizando la función auxiliar getAllImagesAndFavouriteList. Además, hemos implementado la paginación de resultados la misma se explicará en “Opcionales”.

Opcionales

def search (request)

```
def search(request):

    favourite_list = getAllImagesAndFavouriteList(request)
    search_msg = request.POST.get('query', '') or request.GET.get('query', '') #Obtiene el término de búsqueda desde los parámetros POST o GET
    if search_msg=='':      #Si search_msg está vacío, llama a la función getAllImages sin parámetros para obtener todas las imágenes space.
        images= services_nasa_image_gallery.getAllImages(None)

    else:                  #Si hay un término de búsqueda, lo pasa a getAllImages y obtiene solo las imágenes de búsqueda.
        images = services_nasa_image_gallery.getAllImages(search_msg)

    page_number = request.GET.get('page', 1) #Obtiene el valor de la página actual, si no se proporciona toma por defecto 1
    items_per_page = request.GET.get('items_per_page', 5) # Obtiene la cantidad de elementos por página, sino se proporciona usa por defecto 5.
    try: #captura el error
        items_per_page = int(items_per_page)
        if items_per_page < 1:
            items_per_page = 5
    except ValueError:
        items_per_page = 5 #si el numero no es valido coloca por defecto 5
    paginator = Paginator(images, items_per_page)
    page_obj = paginator.get_page(page_number)

    return render(request, 'home.html', { #Renderiza la plantilla home.html y pasa el contexto a la plantilla:
        'page_obj': page_obj,
        'images': images,
        'favourite_list': favourite_list,
        'search_msg': search_msg, # Pasa el término de búsqueda a la plantilla
        'items_per_page': items_per_page, #La cantidad de elementos por página que se está utilizando actualmente
    })
```

La función buscar es la encargada de filtrar las imágenes según el criterio de búsqueda que coloque el usuario, con algunas consideraciones. La variable search_msg obtiene el término de búsqueda desde Post o Get, si la misma se encuentra vacía devuelve una lista de Nasa Card de “space” según la configuración del sistema ubicado en config.py. Sin embargo, si search_msg contiene un término devuelve una lista de Nasa Card según el criterio definido por el usuario. En conclusión, la función renderiza a home.html la lista images según el parámetro de búsqueda y la lista de favourite_list por defecto.

Paginación de Resultados

Con el propósito de que el usuario pueda definir la cantidad de Nasa Card a visualizar por página y además se desplace por las mismas a través de las opciones: anterior, siguiente, ultima y primera, se importó el módulo de Django core.paginator. Fue necesario aplicar la paginación en la función home como search, porque si el usuario no realiza ninguna búsqueda debe permitirle de todos modos recorrer por página. La variable page_number obtiene el valor de la página actual, sino se proporciona toma por defecto 1. Item_per_page obtiene la cantidad de elementos por página, sino se asigna algún valor mostrará por defecto 5 imágenes. Luego se convierte a ítem_per_page en un entero, con el propósito de

evitar algún tipo de error en el ingreso del numero por parte del usuario, para ello se realizó un condicional donde si el ítem es menor a 1, muestre 5 imágenes o si existe algún número no valido mostrará por defecto 5 imágenes. En la variable paginator se asigna Paginator de Django donde se abre una instancia de paginación con la lista de imágenes y de ítems por página. Por último, page_obj obtiene el objeto de la página actual usando el número de página.

Ambas funciones tanto home como search redirigen a home.html además de las listas de imágenes, page_obj y ítems_per_page.

Funciones de Login, Logout y Registration:

Login Views:

```
@login_required
def login_views(request):
    login(request)
    return redirect(request, 'login.html')
```

Esta función nos permite ingresar con un usuario y contraseña previamente registrados. Tomando el objeto request como parámetro de la función login() para compararlo con la base de datos que nos provee Django y así permitirnos ingresar. Esta función vista nos redirige a la planilla login.html junto con el objeto request.

Logout:

```
@login_required
def exit(request):
    logout(request)
    return redirect('home')
```

Esta función nos permite desloguearnos redirigiéndonos hacia la planilla home, la misma funciona utilizando el objeto request como parámetro en la función logout() siendo redirigido a la base de datos para corroborar que está logueado y así desconectarlo de forma automática.

Registration:

```
def registration(request):
    data = {
        'form': CustomUserCreationForm()
    }
    if request.method=='POST':

        user_creation_form = CustomUserCreationForm(data=request.POST)

        if user_creation_form.is_valid():
            user_creation_form.save()
```

```

        user =
authenticate(username=user_creation_form.cleaned_data['username'],
password=user_creation_form.cleaned_data['password1'],
email=user_creation_form.cleaned_data['email'])
    login(request, user)
    asunto = 'Registro Exitoso'
    mensaje = 'A creado su usuario y contraseña de forma correcta'
    email_admin = settings.EMAIL_HOST_USER
    email_usuario = request.POST.get('email', '')
    send_mail(asunto, mensaje, email_admin, [email_usuario])
    return redirect('home')

return render (request, 'registration/registration.html',data)

```

La siguiente función nos permite tomar los datos de la planilla registration.html y una vez que se presione el botón registrar los datos del formulario se les dará forma de lista y serán guardados en la variable data y pasados como parámetros de la clase

CustomUserCreationForms() importada desde Forms.py. para luego ser guardados en la variable user_cration_forms. La misma corrobora si es válida y si cumple la condición es guardada. Dentro de la variable user se guardan los datos obtenidos y se guardan en la base de datos, en la tabla de usuarios por medio de la función login. Hay que destacar que se le agrego la función sendMail() la cual envia un mail al nuevo usuario registrado notificándole que la registración fue exitosa (la cual toma los datos de registration.html). Al final se redirige a inicio de sesión y renderiza el objeto request a la planilla de registration.html en conjunto con la variable data.

Funciones de Favoritos del usuario, guardado y eliminación:

Favoritos del Usuario:

```

@login_required
def getAllFavouritesByUser(request):

    favourite_list =
services_nasa_image_gallery.getAllFavouritesByUser(request)

    return render(request, 'favourites.html', {'favourite_list':
favourite_list})

```

Esta función guarda en una lista y renderiza hacia la planilla favourites.html, los favoritos retornados en formato de lista de la siguiente función:

```

def getAllFavouritesByUser(request):
    if not request.user.is_authenticated:

```

```

        return []
else:
    user = get_user(request)

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

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

    return mapped_favourites

```

La misma retorna una lista, en el caso de estar autenticada con usuario y contraseña, de todos los favoritos del usuario guardados en la base de datos. Esto lo logramos recorriendo elemento por elemento de la lista favourite_list, convirtiendo cada favorito en una NasaCard y almacenándola en la variable NasaCard la cual se va guardando una por una en la lista mapped_favourites. Hay que aclarar que la lista favourite_list busca todos los favoritos del usuario en la función getAllFavouriteByUser(user) usando como parámetro a user. La misma trae todos los datos con estructura de tabla retornando todos los favoritos en una lista.

```

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

```

saveFavourite:

```

@login_required
def saveFavourite(request):
    if request.method == 'POST':
        nasa_card = fromTemplateIntoNASACard(request) # Crea la instancia
de NASACard desde los datos del formulario
        comment = request.POST.get('comment', '') # Obtiene el comentario
del formulario
        nasa_card.comment = comment

        services_nasa_image_gallery.saveFavourite(request)
        favourite_list =
services_nasa_image_gallery.getAllFavouritesByUser(request)

```

```

        return render(request, 'favourites.html', {'favourite_list':
favourite_list})
    else:
        return HttpResponseBadRequest('Invalid request method')

```

Obligatoriamente debe estar logueado con usuario y contraseña para guardar favoritos, la siguiente función comprueba que se cumpla la condición del método Post en la planilla de la galería, cuando se elige el favorito primero instancia la función saveFavourite de services_nasa_image_gallery para guardar el favorito elegido pasandole de parámetro al objeto request. Luego se lo guarda en una lista que trae todos los favoritos del usuario de la función getAllFavouritesByUser. Con esto si la condición del POST es verdadera guarda y renderiza el favorito elegido hacia la planilla favourite.html, guardándolo en el diccionario favourite_list. Y en el caso de un error en el guardado retorna que el objeto es invalido.

Profundizando la explicación la función saveFavourite de services_nasa_image_gallery es la siguiente:

```

def saveFavourite(request):
    fav = mapper.fromTemplateIntoNASACard(request) # transformamos un
request del template en una NASACard.
    fav.user = request.user # le setteamos el usuario correspondiente.
    existing_fav = repositories.getFavouriteByUserAndDetails(
        user=request.user,
        title=fav.title,
        description=fav.description,
        image_url=fav.image_url,
        date=fav.date)
    if existing_fav:
        existing_fav.comment = fav.comment
        return repositories.updateFavourite(existing_fav)
    else:
        return repositories.saveFavourite(fav)

```

Esta función transforma los objetos provenientes de las planillas y los guarda en la variable fav, corrobora que el usuario que esta logueado y luego crea la variable donde se va a guardar la información de los objetos de la galería como, imagen, títulos, etc. Luego compara esa variable con la información de los favoritos guardados del usuario. Si la condición es verdadera y ya está guardado en favoritos retorna por pantalla un mensaje de que ya esta guardado, en el caso de ser falsa y no existir activa la función saveFavourite de repositories.py.

```

def saveFavourite(image):
    try:

```

```

        fav = Favourite.objects.create(title=image.title,
description=image.description, 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

```

Esta función toma como parámetro el objeto image, el cual es enviado desde saveFavourite de services_nasa_image_gallery por medio del parámetro fav el cual nos trae la información de la imagen de la galería a guardar. Se crea la variable fav en la cual se identifican los datos y se guarda en la base de datos retornando fav. En el caso de error al guardar nos muestra un mensaje de error.

deleteFavourite:

```

@login_required
def deleteFavourite(request):
    favId = request.POST.get('id')
    success = services_nasa_image_gallery.deleteFavourite(request, favId)
    if success:
        # Recuperar la lista actualizada de favoritos
        favourite_list =
services_nasa_image_gallery.getAllFavouritesByUser(request)
        return render(request, 'favourites.html', {'favourite_list':
favourite_list})
    else:
        # Manejar el caso de error o redirigir según sea necesario
        return redirect('home')

```

Obligatoriamente debe estar logueado con usuario y contraseña para borrar favoritos, esta función se acciona por medio de un submit en la planilla favourites.html. Una vez accionada identifica el favorito a borrar debido a su 'id' traído desde la planilla favourite.html y guardado en la variable favId. Luego se hace uso de la función deleteFavourite de services_nasa_image_galery pasándole como parámetro la variable favId y el objeto request guardándola en la variable success. La cual si cumple la condición tomara ese favorito de la lista actualizada de favoritos previamente guardada en una lista y los eliminará, en caso de error será redirigido a home.

```

def deleteFavourite(request, favId):
    try:
        success = repositories.deleteFavourite(favId)
        return success # Retornar True si se elimina correctamente, de lo contrario False
    except Exception as e:
        print(f"Error al eliminar el favorito: {e}")

```

```
        return False
```

En el caso de la función deleteFavourite de services_nasa_image_galery tomara el objeto request y al favId pasados como parámetros y lo pasara a la función deleteFavourite pero de repositories.py la cual retornara un True si lo borra, habilitando el success de la función nombrada anteriormente o un False si hay un error al borrar.

```
def deleteFavourite(id):
    try:
        favourite = Favourite.objects.get(id=id)
        favourite.delete()
        return True
    except Favourite.DoesNotExist:
        print(f"El favorito con ID {id} no existe.")
        return False
    except Exception as e:
        print(f"Error al eliminar el favorito: {e}")
        return False
```

En el caso de la función deleteFavourite de repositories devuelve un true o false luego de eliminarlo de forma definitiva con la función de delete() para esto se necesitó la variable id pasada desde la primera función para buscarlo en la tabla de la base de datos de favoritos.

Funciones de comentarios en fotos Favoritas:

```
@login_required
def saveFavourite(request):
    if request.method == 'POST':
        nasa_card = fromTemplateIntoNASACard(request) # Crea la instancia
        de NASACard desde los datos del formulario
        comment = request.POST.get('comment', '') # Obtiene el comentario
        del formulario
        nasa_card.comment = comment

        services_nasa_image_gallery.saveFavourite(request)
        favourite_list =
services_nasa_image_gallery.getAllFavouritesByUser(request)

        return render(request, 'favourites.html', {'favourite_list':
favourite_list})
    else:
        return HttpResponseBadRequest('Invalid request method')
```

Utilizando la función saveFavourite de views.py se agregan dos líneas de códigos que son:

`comment = request.POST.get('comment', '')`: Esta línea obtiene el comentario que el usuario ha escrito en el formulario. Si el usuario no ha escrito ningún comentario, se utiliza una cadena vacía ("").

`nasa_card.comment = comment`: Esta línea añade el comentario del usuario a la tarjeta de imagen de la NASA que se creó antes, guardando así el comentario.

```
{% if card in favourite_list %}
    <button type="submit" class="btn btn-primary btn-sm
float-left" style="color:white" disabled>✓ Ya está añadida a
favoritos</button>
    <textarea name="comment" placeholder="Añadir
comentario..." required></textarea> <!-- Campo para el comentario -->
    <button type="submit" class="btn btn-primary btn-sm
float-left" style="color:white">Guardar comentario</button>
    {% else %}
        <button type="submit" class="btn btn-primary btn-sm
float-left" style="color:white">♥ Añadir a favoritos</button>
    {% endif %}
```

En el html de “Home” pondremos los botones correspondientes para que si card existe en la lista de favoritos aparezca un área de texto que guarde el comentario, que será enviado a la base de datos.

```
{% for favorito in favourite_list %}
    <tr>
        <td>-</td>
        <td>
            style="max-width: 200px; max-height: 200px;"</td>
        <td>{{ favorito.title }}</td>
        <td>{{ favorito.description }}</td>
        <td>{{ favorito.date }}</td>
        {% if favorito.comment %}
            <td> {{ favorito.comment }}</td>
        {% else %}
            <td> </td>
        {% endif %}
    <td>
```

Aquí en el HTML de “favourites” haremos que se muestre comentario que trae la nasa_card ya armada previamente con los datos, donde hay una condición de si hay un comentario lo muestre, de lo contrario que este vacío.

Funciones para el Loader:

```
path('Load-images/', views.load_images, name='cargar-imagenes'),
```

Configuramos en urls.py la URL load-images que apunta a la función load_images en views.py

```
def load_images(request):
    images, favourite_list = getAllImagesAndFavouriteList(request)
    image_data = [{ 'url': image.url} for image in images]
    return JsonResponse({ 'images': image_data})
```

En views.py, se define la función load_images(request) que maneja la solicitud cuando un usuario accede a la URL load-images (carga automática)

```
<script>
    document.addEventListener("DOMContentLoaded", function() {
        let images = document.querySelectorAll("#nasacard-container
.card img");
        let imagesLoaded = 0;

        images.forEach(image => {
            if (image.complete) {
                imagesLoaded++;
            } else {
                image.addEventListener("load", () => {
                    imagesLoaded++;
                    if (imagesLoaded === images.length) {
                        document.getElementById("Loading-
spinner").style.display = "none";
                        document.getElementById("nasacard-
container").style.display = "grid";
                    }
                });
                image.addEventListener("error", () => {
                    imagesLoaded++;
                    if (imagesLoaded === images.length) {
                        document.getElementById("Loading-
spinner").style.display = "none";
                    }
                });
            }
        });
    });
</script>
```

```

        document.getElementById("nasocard-
container").style.display = "grid";
    }
}
});

if (imagesLoaded === images.length) {
    document.getElementById("Loading-spinner").style.display =
"none";
    document.getElementById("nasocard-container").style.display
= "grid";
}
});
</script>

```

El script que pondremos en el html de “Home” verifica individualmente cada imagen dentro de #nasocard-container con `images.forEach`. Para cada imagen, comprueba si ya ha terminado de cargarse (`image.complete`). Si una imagen ya está completamente cargada, incrementa el contador `imagesLoaded`. Si alguna imagen aún no está completamente cargada, el script espera a que se complete su carga y luego realiza las acciones necesarias (como actualizar `imagesLoaded`), mientras se produce todo este proceso se mostrará un “`Loading-spinner`” que será agregado también en este HTML de esta manera:

```

<div id="Loading-spinner">
    
</div>

```

Lo que hará indicar que la página está procesando la información. Una vez que todas las imágenes están listas y la función compare el largo de “`images`” con el de “`imagesLoaded`” y este sea True, oculta el Spinner y muestra las imágenes cargadas en el contenedor `#nasocard-container`.

Implemento de Bootstrap para el Front:

The screenshot shows a web page titled "Galería de imágenes de la NASA". It features two cards. The first card, titled "EXPLORERMOONtoMARS", contains a thumbnail of a red planet and Earth, with a "Space Exploration Video" link and a "Añadir a favoritos" button. The second card, titled "Space-to-Ground_171_170407", shows a group of astronauts in blue uniforms, with a "Space Station Benefits" text overlay and a "Añadir a favoritos" button.

The screenshot shows a navigation bar with links: "Proyecto TP", "Inicio", "Galería", "Favoritos", and "Salir". A "Logout" icon is also present. Below the navigation bar, a modal window titled "Inicio de sesión" displays fields for "Usuario" and "Contraseña", and a "Ingresar" button. At the bottom of the page, there is a footer with the text "Introducción a la Programación | UNGS" and "Trabajo práctico - 1er cuatrimestre del 2024".

Se utilizó Bootstrap para el implemento de botones, modificar el NavBar para que además al achicar el ancho de la pantalla las opciones se oculten y aparezca un botón que al apretarlo despliega las opciones ya mencionadas, luego para mejorar la visión de las cards con su respectiva información.

The screenshot shows a pagination control at the bottom of a page. It includes buttons for "Anterior" and "Siguiente", a "Última" button, and a "Página 1 de 14" label. Below these buttons is a "Items por página:" dropdown menu set to the value "5".

Se modificó la paginación para que tenga los botones ya mencionados previamente.

Se implemento también para mejorar el formulario del inicio de sesión como el de registrar una nueva cuenta.

Dificultades

En la función seacrh nos hemos encontrado con la dificultad de que al buscar la palabra “Marte” nos arrojaba un “Key error”, porque algunos objetos en particular del Json no contaban con el campo de “descripción”, requerido para convertirlo en Nasa Card. En consecuencia, tomamos la decisión de modificar la función getAllImages (input=none) en donde al recorrer cada objeto del json, antes de convertirlo en una nasa card, nos aseguramos de que cuenten con todos los campos necesarios antes de guardarla en la lista a través de un condicional y un except KeyError para evitar cualquier posible error.

En el registro del usuario tuvimos la dificultad de tener que interactuar con la base de datos por lo que debíamos utilizar los formularios de django. Para esto creamos un nuevo archivo de Python llamado forms.py y dentro una clase particular del tipo constructor para poder interactuar con las tablas y autenticar los datos ingresados por el usuario. Además, la dificultad a la hora de implementar las funciones de favoritos ya que nos dábamos cuenta que se complementaban con las funciones dentro de repositories.py y de services_nasa_image_galery.py por lo que me daba error a la hora de guardar los favoritos. Luego de complementar las funciones y agregar los parámetros para que trabajen en conjunto no tuvimos problemas para hacer el guardado.

También hubo algunos inconvenientes en el momento de enviar el mail a los usuarios registrados, el cual era debido a que quería utilizar el mismo POST para dos funciones distintas, por lo que me tiraba un error de seguridad del token del html. Lo solucionamos incorporando dentro de la misma función de registro la función send_mail.

Otra dificultad fue que a la hora de crear un comentario si bien armamos todo el HTML y su funcionamiento, nos daba error que estaba vacío el comentario, ya que no habíamos

creado la tabla en la base de datos que tenga un “comment” para que se guarde ahí y luego lo traiga a favoritos y lo muestre.

Otro inconveniente fue que cuando armábamos el loader, ya teniendo nuestro script implementado y el contenedor con la imagen del loader, nunca se llegaba a activar la carga de la función “load_images” que es la que maneja la carga de las imágenes, lo cual investigando Django tiene la funcionalidad de activarlas desde urls, lo cual al agregar el “path” con esa función se solucionó el problema.